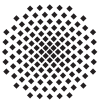

Universität Stuttgart

Modellierung regelkonformer Geschäftsprozesse

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Daniel Schleicher

aus Waiblingen

Hauptberichter: Prof. Dr. Frank Leymann

Mitberichter: Prof. Dr.-Ing. habil. Bernhard Mitschang

Tag der mündlichen Prüfung: 30.07.2014

Institut für Architektur von Anwendungssystemen der Universität Stuttgart

2014

INHALTSVERZEICHNIS

Zusammenfassung

Regelkonformes Verhalten ist für viele Firmen und Konzerne ein wichtiger Punkt auf der Agenda hin zu einer nachhaltigen Wachstumsstrategie. Durch den in den letzten Jahren gestiegenen regulatorischen Druck und die zu erwartenden Strafen bei Verstößen gegen Regeln und Gesetze, sind Firmen gezwungen, sich intensiver mit der Überwachung ihrer Geschäftsprozesse zu befassen. Die wiederkehrenden Skandale um nicht regelkonformes Verhalten von Mitarbeitern und die daraus resultierenden Konsequenzen unterstreichen die Richtigkeit dieser Richtungswendung.

Viele Firmen arbeiten mit IT-unterstützten Geschäftsprozessen, in deren automatische Ausführung Menschen eingebunden sind. Diese Geschäftsprozesse müssen bezüglich der Einhaltung neuer oder sich ändernder Regeln und Gesetze auf dem neuesten Stand gehalten werden. Den Aufwand für die Aktualisierung der Geschäftsprozesse möglichst gering zu halten, ist eine Herausforderung, der die Unternehmen gegenüber stehen.

Der Begriff *Compliance* drückt im Englischen das Einhalten von Regeln und Gesetzen aus. Im günstigsten Fall sollte Compliance schon bei der Erstellung eines neuen Prozesses in Betracht gezogen werden, da in dieser Phase Entwicklungsfehler mit dem geringsten Aufwand behoben werden können. Durch die Zunahme der Regeln und Gesetze, die von Geschäftsprozessen eingehalten werden müssen, ist es wichtig, die Anforderungen mit Bezug auf Compliance von den wirtschaftlichen Zielen der Prozessentwicklung zu trennen. Menschliche Prozessentwickler sollen sich voll und ganz auf die Entwicklung der Geschäftslogik eines Prozesses konzentrieren können. Die Überprüfung von Gesetzen und Regularien soll automatisiert durch Werkzeuge geschehen, die weitgehend im Hintergrund arbeiten.

Graphische Entwicklungswerkzeuge müssen Mittel bereitstellen, um Regularien und Gesetze zu verwalten und diese mit Geschäftsprozessen zu verbinden. Es müssen dabei zwei Szenarien behandelt werden können. Erstens müssen Regeln und Gesetze vor Beginn der Entwicklung eines neuen Prozesses festgelegt und automatisch überprüfbar gemacht werden. Zweitens müssen bestehende Geschäftsprozesse mit neuen Regeln und Gesetzen verknüpft werden können. Desweiteren müssen von den Überprüfungswerkzeugen verschiedene Arten von Regeln und Gesetzen verarbeitet werden können. Beispiele hierfür sind Regeln, die auf den Datenfluss in Prozessen angewendet werden oder Regeln, die auf den Kontrollfluss in Prozessen Anwendung finden.

Diese Arbeit erweitert eine bestehende Entwicklungsumgebung für Geschäftsprozesse und implementiert die oben aufgeführten Anforderungen. Bei der Entwicklung der Konzepte und deren Umsetzung im Prototyp wurde darauf geachtet, dass diese Konzepte die Entwicklungsarbeit am Geschäftsprozess so wenig wie möglich behindern.

Die bestehende Entwicklungsumgebung für Geschäftsprozesse wurde intern umstrukturiert, so dass alle Änderungen am aktuell angezeigten Prozessmodell auf Verstöße gegen Complianceregeln überprüft werden können.

Die beiden grundlegenden Forschungsbeiträge dieser Arbeit sind das Compliancetemplate und der Compliancescope. Das Compliancetemplate ist eine Prozessvorlage, die an bestimmten Stellen unvollständig ist. Nur diese Stellen können von einem Prozessentwickler mit Prozessaktivitäten gefüllt werden, um einen vollständig spezifizierten Prozess zu erhalten. Durch diese Vorgabe wird verhindert, dass Complianceregeln umgangen werden können.

Der Compliancescope ist ein Mittel, um Teile von bestehenden Prozessmodellen mit Complianceregeln zu verknüpfen. Diese Teile von Prozessmodellen werden automatisch überprüft, wenn eine Änderung an ihnen vorgenommen wird. Die automatische Überprüfung von Teilen von Prozessmodellen übernimmt das zur Entwicklung des Prozesses verwendete graphische Entwicklungswerkzeug.

Aufbauend auf diesen beiden Konzepten beschreibt die vorliegende Dissertation drei weitere Beiträge. Das Konzept der *Compliancedomain* baut auf dem Konzept des Compliancescopes auf und erweitert diesen, um mit datenbasierten Complianceregeln arbeiten zu können. *Vervollständigungsebenen* sind ein Konzept, verschiedenen Partnern die Arbeit an einem, mit Complianceregeln versehenen Prozessmodell, zu ermöglichen. Die *Architektur* des Prototyps dieser Dissertation zeigt, wie diese neuen Konzepte umgesetzt und somit anwendbar gemacht werden können.

Abstract

Compliance is an important issue for many enterprises on their way to a sustainable growth. Due to the increased regulatory pressure stemming from more and more rules and regulations being set in place and associated penalties, enterprises are more intensively forced to cope with compliance issues concerning their business processes. The returning scandals concerning non-compliant behaviour of employees and the subsequent penalties stress the correctness of this turn.

Many enterprises are running IT-driven business processes. Humans are integrated into the automatic execution of these business processes. These business processes must be kept up to date in order to meet changing regulations. Holding the effort on a reasonable level for keeping business processes up to date is a challenge enterprises are facing today.

Compliance should be considered from the beginning of the development phase of a new business process. It takes less effort for removing compliance issues in this phase. Due to the increased num-

ber of regulations it is important to separate compliance requirements from business requirements during the development of a new business process. Human business process developers should be able to fully concentrate on the development of the business logic of a new business process. The adherence to compliance rules should automatically be guaranteed by graphical development tools. These tools should check compliance rules in the background.

Graphical business process development tools have to provide means to manage regulations and laws. These tools must also support the linking of compliance rules to business processes. When working with these tools two scenarios should be possible: First of all compliance rules must be automatically verifiable from the beginning of the development of a business process. Second, it must be possible to link compliance rule to existing business processes. Apart from that the tools used to automatically check compliance rules in the background must be capable of dealing with different kinds of compliance rules. Examples for different kinds of compliance rules are compliance rules restricting the control flow of a business process in contrast to compliance rules restricting the data flow of a business process.

This thesis extends an existing integrated development environment (IDE) for business processes. It realises the requirements stated above. One goal for the development of the prototypical implementation of the new concepts is that the additions to the existing platform do not hamper human developers during the creation phase of a business process.

The existing IDE has been restructured from the ground up. With these changes it is possible to introduce compliance checking mechanisms which are capable of coping with different kinds of compliance

rules.

The two fundamental contributions of this thesis are the Compliance Template and the Compliance Scope. The compliance template is a process template which is kept incomplete in a number of places. Only these places can be filled with business activities by human business process developers in order to get a fully specified business process. This development restriction prevents human business process developers from circumventing compliance rules, which are already present in the original compliance template.

The compliance scope is a means to attach compliance rules to certain areas in a business process. A modification of the business process within such an area makes it only necessary to automatically check the area where the modification was made.

Based on the fundamental concepts this thesis describes three further new concepts. The concept of a Compliance Domain is based on the compliance scope. Compliance scopes are extended to be able to work with data-based compliance rules. The concept of a refinement layer allows for integrating different stake-holders during the development of a new business process. The architecture of the prototype shows the practicability of the approach of this thesis.

Danksagungen

Es ist geschafft! Nach Jahren der Arbeit, vielen Veröffentlichungen, Vorträgen und Gesprächen liegt meine Dissertation vor Ihnen. Damit ist es an der Zeit mich bei denen zu bedanken, die mich in dieser spannenden Phase meines Lebens begleitet haben.

Ich möchte mich besonders bei Herrn Professor Frank Leymann für das Vertrauen bedanken, das er in mich setzte, als ich mit meiner Dissertation ganz am Anfang stand. Ohne dieses Vertrauen, die Anleitung und die Motivation, die ich aus den Gesprächen mit ihm gewonnen habe, hätte die vorliegende Arbeit nicht entstehen können.

Die Forschungsrichtung der vorliegenden Arbeit, sowie die Herangehensweise an die Forschungsfragen wurden durch viele Gespräche mit Mitgliedern des Instituts für Architektur von Anwendungssystemen der Universität Stuttgart positiv beeinflusst. Folgende Personen möchte ich in diesem Zusammenhang besonders erwähnen. Tobias Anstett und Ralph Retter für ihre Unterstützung bei der Suche nach der zündenden Idee und der Hilfe bei der Festlegung meiner Forschungsrichtung.

David Schumm als Ratgeber mit vielen hilfreichen Tipps im Bereich Compliance. Tammo van Lessen, Jörg Nitzsche und Ralph Retter für die Starthilfe beim Verfassen eines wissenschaftlichen Artikels und dessen Veröffentlichung.

Besonders danken möchte ich auch Maike Buhr, Christoph Fehling, Daniel Gerlach, Christoph Schleicher und David Schumm für ihren großen Einsatz als Lektoren des vorliegenden Dokuments.

Weiterhin bedanke ich mich bei Tobias Binz, Uwe Breitenbücher, Hanna Eberle, Christoph Fehling, Katharina Görlach, Dimka Karastoyanova, Oliver Kopp, Daniel Martin, Alexander Nowak, Sebastian Wagner, Branimir Wetzstein, Matthias Wieland, Daniel Wutke und Sema Zor für ihr Lob, ihre konstruktive Kritik, ihre Anregungen und Ideen, die die vorliegende Arbeit positiv beeinflusst haben.

KAPITEL 1

EINLEITUNG

Noch vor einigen Jahren wenig beachtet, ist Compliance in den letzten Jahren zu einem Schlagwort in den Führungsetagen der Unternehmen avanciert [SALS10, HM10, KBE+10a, KBE+10b]. Unter dem Begriff Compliance versteht die vorliegende Arbeit Regelkonformität von Abläufen in Unternehmen. Im Folgenden wird der Begriff Compliance mit dem Begriff Regelkonformität synonym verwendet.

Ausgehend von einer allgemeinen Betrachtung von Compliance in Unternehmen wird das der vorliegenden Arbeit zugrunde liegende konkrete Problem in diesem Kapitel näher umrissen. Einerseits wird gezeigt, wie wichtig Compliance für Unternehmen und deren Geschäftsprozesse ist. Andererseits wird geklärt, welchen Beitrag die vorliegende Arbeit im Bereich der Entwicklung regelkonformer Geschäftsprozesse leistet.

1.1. Bedeutung von Compliance für Unternehmen heute

Eines der wohl bekanntesten Regelwerke sind die Zehn Gebote der Bibel [LB01]. Hier wurden schon vor Jahrtausenden Richtlinien definiert, die für die damalige Gesellschaft bindend waren. Richtlinien sind ein wichtiger Bestandteil des menschlichen Zusammenlebens. Dies setzt sich auch bei der Arbeit in Unternehmen und den Beziehungen zwischen Unternehmen fort.

Compliance wird in den nächsten Jahren für Unternehmen eine immer größere Rolle spielen. Dies zeigt eine Studie von A.T. Kearney [MPRS13]. In dieser Studie wurden 40 Complianceexperten führender Unternehmen zum Thema Compliance befragt. Demnach sind die immer weiter steigende Zahl von Unternehmen, die in bestimmten Rechtssystemen tätig sind, die steigende Zahl der Vorschriften und die persönliche Haftbarkeit des Topmanagements für Complianceverstöße, Hauptgründe für die zunehmende Bedeutung von Compliance in Unternehmen. Die Unternehmen erwarten vermehrt Complianceverstöße in den Bereichen Produktsicherheit, Datensicherheit und Korruption. Aus diesen Gründen wollen die meisten der befragten Unternehmen ihre Compliancesysteme ausbauen.

Weiter zeigt die Studie auf, dass viele in Unternehmen ausgeführte Prozesse im Hinblick auf Compliance neu aufgebaut werden müssen. Die Integration von Compliance-Regeln in Prozesse sehen dabei 70% der Manager als wesentlich an.

Die Möglichkeit der Verhängung hoher Strafen ist ein weiterer Grund für Unternehmen sich mit Compliancefragen zu beschäftigen. Der Autokonzern Daimler musste im April 2010 185 Millionen Dollar Strafe zahlen aufgrund von Korruptionsvorwürfen. Siemens musste 800 Mil-

lionen Dollar Strafe aufgrund von Bestechungsvorwürfen zahlen. Beide Konzerne standen daraufhin unter der Aufsicht der Securities and Exchange Commission (SEC), der US-Börsenaufsichtsbehörde. Unter dieser Aufsicht mussten beide Unternehmen Auflagen der SEC erfüllen, um von weiteren Geldstrafen unbehelligt zu bleiben [Haw11, Mat12]. Weiterhin zeigt die Einführung eines Vorstandsressorts für Compliance, dass dieses Thema für Daimler langfristige Relevanz hat [PJ12]. Im Jahr 2011 hatte ein Händler der Schweizer Großbank UBS 1,5 Milliarden Euro Verlust gemacht. Interne Kontrollsysteme hätten bei diesem Betrag Alarm schlagen müssen.[1] Weiterhin zahlte die UBS in einem Vergleich mit den USA 780 Millionen Dollar Strafe, um einer Anklage in einem Steuerhinterziehungsfall zu entgehen.[2]

Beispiele für Regelwerke, die von Firmen beachtet werden müssen, sind der Sarbanes-Oxley Act (SOX) [Uni02] von 2002, der Gramm-Leach-Bliley Act [Uni99] oder Basel II [Bas06] von 2006. Ähnliche Regelwerke findet man in weiteren Ländern, wie zum Beispiel in China, mit dem Gesetz der Chinesischen Volksrepublik zur Volksbank Chinas (Englisch: The law of the people's republic of China on the people's bank of China) [Chi03]. Weitere Regularien beziehen sich auf firmeninterne Vereinbarungen. Beispielsweise kann eine Vereinbarung die Förderung des *Green Business Process Management (BPM)* [NLS+11] sein.

Im Folgenden werden Geschäftsprozesse der Kürze wegen als *Prozesse* bezeichnet. Der Begriff *Prozessmodell* wird verwendet, um einen

[1] http://www.sueddeutsche.de/geld/untersuchung-zu-ubs-skandal-wie-ein-haendler-milliarden-verzocken-konnte-1.1172370

[2] http://www.spiegel.de/wirtschaft/einigung-in-steuerhinterzieher-fall-ubs-verraet-geheimdaten-hunderter-kunden-an-us-justiz-a-608573.html

mit einer graphischen oder textuellen Notation erstellten Prozess zu beschreiben.

Diese Arbeit beschäftigt sich mit der Compliance von IT-unterstützten Prozessen. Bei IT-unterstützten Prozessen sind für einen Teil der Ausführung eines Prozesses IT-Systeme verantwortlich. Für die Erstellung von IT-unterstützten Prozessen werden spezielle Prozessbeschreibungssprachen, wie zum Beispiel die Business Process Execution Language (BPEL) [OAS07] eingesetzt. Schon bei der Erstellung von BPEL-Prozessen sollte die Einhaltung von Complianceregeln gewährleistet werden. Dies ist schon aus dem Grund notwendig, da die Beseitigung eines Fehlverhaltens eines Prozesses in der Entwicklungsphase relativ wenig Zeit und Mühe kostet im Vergleich zur Behebung eines Fehlers in einem produktiv eingesetzten Prozess [Bro95].

Die fortschreitende Automatisierung ist eines der Mittel, derer sich Unternehmen heutzutage bedienen, um wettbewerbsfähig zu bleiben und Kunden und Aktionäre zufrieden zu stellen. Dies ist einer der Gründe für ein immer größer werdendes Interesse von Unternehmen am Prozessmanagement [LR00]. Parallel dazu sehen sich Unternehmen einer immer größer werdenden Zahl von regulatorischen Anforderungen gegenüber gestellt. Diese Anforderungen sind in den meisten Fällen in Gesetzestexten niedergeschrieben (siehe oben).

1.2. Zentrales Anliegen

Das Hauptanliegen der vorliegenden Arbeit besteht darin, den menschlichen Prozessentwickler dabei zu unterstützen, Prozesse zu entwickeln, die bestimmten Complianceregeln genügen. Ein Mitarbeiter soll zum Beispiel beim Einfügen einer Aktivität in einen Prozess dar-

auf aufmerksam gemacht werden, wenn diese eine Complianceregel verletzt. Die bisher in der Literatur aufgeführten Konzepte konzentrieren sich darauf, die Grundlagen für die automatische Überprüfung von Prozessen bereitzustellen. Die vorliegende Arbeit geht davon aus, dass Complianceregeln mittels formaler Sprachen definiert und mit speziellen Programmen überprüft werden können.

Darüber hinaus werden graphische Mittel vorgestellt, die den menschlichen Prozessmodellierer bei der Erstellung regelkonformer Prozesse leiten. Zum Beispiel werden Prozessmodellierer durch eine Prozessvorlage, oder die Definition von Bereichen in Prozessen, die mit bestimmten Complianceregeln verknüpft sind, bei der Entwicklung geleitet. Auf diesem neuen Ansatz bauen weitere Beiträge der vorliegenden Dissertation auf.

Das Ziel, menschliche Prozessmodellierer bei der Entwicklung regelkonformer Prozesse zu unterstützen, wird durch die Erstellung eines Prototyps, der die in dieser Arbeit vorgestellten Konzepte implementiert, erreicht. Die folgenden Eigenschaften des Prototyps ermöglichen dies:

- Der Prototyp zeigt dem Prozessmodellierer durch automatische Überprüfung, ob Verstöße gegen Complianceregeln im Prozess vorliegen. Er zeigt weiterhin den Ort, wo im Prozess diese Verstöße vorliegen.

- Mit dem Prototyp kann eine Prozessvorlage geladen werden, die die Grundlage für die Entwicklung eines regelkonformen Prozesses darstellt.

- Der Prototyp lässt es zu, bestimmte Bereiche in einem Prozess

mit Complianceregeln zu versehen. Danach können nur noch die Änderungen in diesen Bereichen vorgenommen werden, die diese Complianceregeln nicht verletzen.

- Mit dem Prototyp ist es möglich, Complianceregeln zu definieren, die aus atomaren Ausdrücken bestehen, die mit verschiedenen formalen Sprachen beschrieben sein können.

Die Technik des Modelchecking wird verwendet, um die automatische Überprüfung der Complianceregeln möglich zu machen. Modelchecking ist ein Werkzeug, das für die Umsetzung der neuen Konzepte dieser Arbeit im Prototyp verwendet wird.

Neben anderen Beiträgen sind die folgenden Beiträge Schwerpunkte dieser Arbeit:

- Das Compliancetemplate. Siehe Kapitel 4.2
- Der Compliancescope. Siehe Kapitel 4.3
- Die Compliancedomain. Siehe Kapitel 5.2

1.3. Problemstellung und Motivation

Prozesse sind die Grundlage der Zusammenarbeit innerhalb von Unternehmen und zwischen Partnern. Für die Erstellung von Produkten sind sie unverzichtbar. Complianceregeln, die auf Prozessen Anwendung finden, decken somit einen wichtigen Teil aller möglichen Complianceregeln ab, die in Unternehmen vorstellbar sind. Die vorgestellten neuen Konzepte im Bereich der Entwicklung regelkonformer Prozesse sind somit ein wichtiger Schritt hin zur vollständigen Kontrolle von

Unternehmensprozessen. In dieser Arbeit wird ein Prototyp vorgestellt, der zeigt, wie Unternehmen in Zukunft Prozesse erstellen und diese gleichzeitig regelkonform halten können.

Die Verwaltung, Anwendung und Durchsetzung von Complianceregeln ist für Organisationen ein komplexes Problem. Die Komplexität dieses Problems wird durch die Zahl und den Umfang der relevanten Gesetzestexte bestimmt. Wie oben erläutert, nimmt die Zahl der Gesetzestexte, die Organisationen befolgen müssen, seit Jahren stetig zu. Auch der Grad der Automatisierung und der IT-unterstützten Prozesse nimmt in Organisationen zu. Organisationen müssen bei Prüfungen nachweisen, wie ihre Prozesse abgelaufen sind. Weiter müssen sie zeigen, dass sie ausreichende Maßnahmen ergriffen haben, um Verstöße gegen Complianceregeln zu vermeiden.

Diese Arbeit ist durch mehrere Faktoren motiviert. Der erste Faktor ist der wirtschaftliche Aspekt der Einhaltung von Complianceregeln. Es können hohe Strafen oder hohe Verluste auf die Unternehmen zukommen, wenn sie Complianceregeln verletzen.

Auch die Entlastung des menschlichen Prozessmodellierers ist eine Motivation dieser Arbeit. Der Prozessmodellierer soll sich bei der Erstellung von regelkonformen Prozessen auf das eigentliche Ziel der Entwicklung des Prozesses konzentrieren. Die Einhaltung bestimmter Regeln soll während der Entwicklung eines Prozesses für den Menschen zweitrangig sein.

Bestehende Ansätze zur Überprüfung der Compliance eines Prozesses bauen darauf auf, das gesamte Prozessmodell zu überprüfen [ADW08], das bis zu diesem Zeitpunkt erstellt wurde. Diese Überprüfung des gesamten Prozessmodells kann jedoch überflüssig sein, wenn nur eine unwesentliche Änderung an diesem Prozessmodell vorgenom-

men wurde. Es sollte demnach möglich sein, nur die geänderten Teile eines Prozesses automatisch zu überprüfen und somit Zeit und Kosten zu sparen. Dieses Ziel verfolgen die in dieser Arbeit vorgestellten Ansätze für die Überprüfung von Prozessen.

Im Gegensatz zur Ausführungsphase gibt es für die Phase der Erstellung eines Prozesses nur unzureichende Werkzeuge und Konzepte, um Verstöße gegen Complianceregeln automatisch zu erkennen. Der wissenschaftliche Beitrag dieser Arbeit besteht aus zwei Komponenten.

1. Der Präsentation neuer Konzepte zur Verwaltung und Überprüfung von Complianceregeln zur Entwicklungszeit eines Prozesses.

2. Der Architektur und Implementierung eines Prototyps, der diese Konzepte umsetzt.

Anforderungen an diesen Prototyp sind:

1. Die Möglichkeit der Verknüpfung bestimmter Bereiche in einem Prozess mit Complianceregeln.

2. Der Prototyp unterstützt die Beteiligung mehrerer Partner an der Erstellung eines Prozesses, ohne sich gegenseitig zu behindern. Insbesondere soll es möglich sein, dass verschiedene Partner bei der Erstellung des Prozesses neue Complianceregeln in das Prozessmodell integrieren. Bei Unvereinbarkeit neuer Complianceregeln mit bereits im Prozess vorhandenen, soll der menschliche Prozessmodellierer informiert werden.

3. Complianceregeln sollen in verschiedenen Sprachen beschreibbar sein. Diese Sprachen sollen es zumindest unterstützen, Compliance-

regeln zu schreiben, die den Kontrollfluss und den Datenfluss in einem Prozess einschränken.

4. Die Möglichkeit zur automatischen Überprüfung von Complianceregeln, die mit Prozessen verknüpft sind.

5. Der Prototyp soll nach einer Überprüfung von Complianceregeln bei einer Verletzung einer Complianceregel anzeigen, durch welche Konstrukte im Prozessmodell diese Verletzung zustande gekommen ist.

Der Prototyp erweitert eine bestehende Umgebung zur Erstellung von Prozessen mit Funktionalität, um Complianceregeln zu verwalten, mit Prozessen zu verknüpfen und automatisch zu überprüfen.

Um die Erstellung regelkonformer Prozesse optimal zu unterstützen, benötigt man Werkzeuge, die in verschiedenen Entwicklungssituationen und im Umgang mit verschiedenartigen Complianceregeln Lösungen bieten. Die in dieser Arbeit behandelten Entwicklungssituationen sind die Neuentwicklung von Prozessen und die Änderung bestehender Prozesse.

Die Neuentwicklung von Prozessen stand bislang nicht im wissenschaftlichen Fokus. Ein Konzept in diesem Bereich muss die Festlegung der Complianceregeln für einen neu zu entwickelnden Prozess vor der eigentlichen Entwicklung des Prozesses ermöglichen. Weiter muss bereits zu Beginn der Prozessentwicklung sichergestellt werden, dass Überprüfungsmechanismen für die Einhaltung von Complianceregeln greifen. Diese Überprüfungsmechanismen leiten den Prozessmodellierer während der Modellierung hin zu einem regelkonformen Prozess.

Ein anderer Fall, für den in dieser Arbeit ein neues Konzept prä-

sentiert wird, ist die Änderung existierender Prozesse, die nicht mit Complianceregeln versehen sind. Überprüfungsmechanismen, die in graphische Entwicklungswerkzeuge eingebaut sind, greifen hier nicht. Für solche existierenden Prozesse zeigt diese Arbeit eine Lösung, mit der Prozessmodelle mit Complianceregeln verknüpft und automatisch überprüft werden.

Eine weiteres Ziel dieser Dissertation ist die Bereitstellung von Konzepten für die abteilungsübergreifende Entwicklung regelkonformer Prozesse. Teammitglieder sollen zum Beispiel bei der Entwicklung von Prozessen über Landesgrenzen hinweg miteinander arbeiten können. Es wird daher ein Konzept und eine Implementierung benötigt, die es möglich machen, dass Complianceregeln an verschiedenen Stellen im Prozessmodell und von verschiedenen Teams im Entwicklungsprozess eingeführt werden können.

Diese Anforderungen sollen mit unterschiedlichen Arten von Complianceregeln zusammen arbeiten. Es ist daher notwendig, dass Complianceregeln, die in verschiedenen Sprachen geschrieben worden sind, verbunden werden können. Weiterhin sollen die oben geforderten Konzepte und Algorithmen mit zukünftigen Sprachen zur Definition von Complianceregeln zusammen arbeiten können.

1.4. Forschungsbeiträge der Arbeit

Der wissenschaftliche Beitrag dieser Arbeit gliedert sich in einen theoretischen und einen praktischen Teil. Im theoretischen Teil werden Konzepte dargestellt, die Lösungen für die Entwicklung regelkonformer Prozesse repräsentieren. Außerdem zeigen diese Konzepte, wie menschliche Prozessmodellierer bei der Entwicklung regelkonformer

Prozesse unterstützt werden können.

Im praktischen Teil wird gezeigt, wie die theoretischen Konzepte in einem Prototyp zusammen arbeiten.

Im Folgenden werden die Forschungsbeiträge dieser Arbeit präsentiert.

1.4.1. Erweiterung eines Variabilitätskonzepts und eines Prozessmetamodells für die Unterstützung der Entwicklung regelkonformer Prozesse

Das in [ML08, Mie08] vorgestellte und in Abschnitt 4 beschriebene Variabilitätskonzept macht es möglich, Variabilitätspunkte in beliebigen Dokumenten zu definieren. Diese Variabilitätspunkte können dann anhand verschiedener Kriterien zeitlich variabel gefüllt werden. Ein Beispiel für einen Einsatzbereich dieses Variabilitätskonzepts sind BPEL-Prozesse, die zur Entwicklungszeit mit Variabilitätspunkten versehen werden. Zur Deploymentzeit werden diese Variabilitätspunkte gefüllt. So wird der BPEL Prozess lauffähig gemacht.

In dieser Arbeit wird dieses Konzept erweitert und für die Bedürfnisse der Anwendung von Complianceregeln auf einen Prozess angepasst. Der Name dieses neuen Konzepts ist *Compliancedeskriptor*. Hierzu wird eine Prozessvorlage mit Variabilitätspunkten versehen, die zur Entwicklungszeit von einem menschlichen Prozessmodellierer mit Aktivitäten gefüllt wird. Der Compliancedeskriptor wird dazu verwendet, die Menge der Aktivitäten einzuschränken, mit der die Variabilitätspunkte befüllt werden können. Diese Einschränkung wird mittels formaler Complianceregeln definiert, die im Compliancedeskriptor enthalten sind. Ein graphisches Entwicklungswerkzeug ist somit in der

Lage bei einer Modifikation eines Prozessmodells Complianceregeln auszuwerten und bei einer Verletzung den Prozessmodellierer zu benachrichtigen. Dieser Beitrag erfüllt durch die Umsetzung im Prototyp die Anforderung 1.

Im Folgenden werden die in [ML08] eingeführten Variabilitätspunkte *Complianceregionen* genannt. Diese Bezeichnung ist sinnvoll, da in dieser Arbeit nur Variabilitätspunkte verwendet werden, die sich in Prozessmodellen befinden und mit Complianceregeln versehen sind. Weiterhin wird mit diesem Namen auf dem in [EUL09] von Eberle et al. vorgestellten Konzept einer Region aufgebaut, welches nicht spezifizierte Bereiche in einem Prozessmodell definiert.

Die Kombination eines Compliancedeskriptors, eines Variabilitätsdeskriptors und einer mit Complianceregionen versehenen Prozessvorlage wird *Compliancetemplate* genannt.

Die Konzepte dieses Beitrags der Dissertation wurden zum ersten Mal in [SALM09] vorgestellt.

1.4.2. Algorithmus zur Überprüfung des Kontrollflusses von Teilbereichen von Prozessen

In dieser Arbeit wird ein neues Konzept zur Annotation von Teilprozessen mit Complianceregeln, der Compliancescope [SWLS10], präsentiert. Darauf aufbauend zeigt Abschnitt 4.5 einen Algorithmus, der es mit der Technik des Modelcheckings ermöglicht, die annotierten Teilprozesse auf Verstöße gegen Complianceregeln zu untersuchen.

Die oben vorgestellte Erweiterung eines Variabilitätskonzepts zur Definition von Complianceanforderungen auf Prozessmodellen kann im Wesentlichen auf neu erstellte Prozessmodelle angewendet wer-

den. Jedoch muss auch ein Mittel bereitgestellt werden, um existierende Prozessmodelle mit Complianceregeln zu versehen und somit Modifikationen automatisch überprüfbar zu machen. Ein solches Konzept, das zur Annotation von Prozessmodellen mit Complianceregeln dient, ist das Konzept des Compliancescopes. Mit Compliancescopes können Bereiche eines Prozessmodells definiert werden, für die bestimmte Complianceregeln gelten. Die oben erwähnten Compliancedeskriptoren werden für die Verknüpfung von Compliancescopes mit Complianceregeln verwendet. Werden die in einem Compliancescope enthaltenen Aktivitäten von einem Prozessmodellierer geändert, müssen nur die mit dem entsprechenden Compliancescope verknüpften Complianceregeln überprüft werden und nicht das gesamte Prozessmodell. Compliancescopes erfüllen durch die Umsetzung im Prototyp die Anforderung 1.

Zur Verifikation von Modifikationen an Compliancetemplates und Compliancescopes werden Modelchecker verwendet. Hierbei wird der zu verifizierende Teil eines Prozessmodells in ein Modell in der Eingabesprache eines Modelcheckers transformiert. Die Antwort des Modelcheckers dient dazu, den menschlichen Prozessmodellierer darüber zu unterrichten, ob die aktuelle Modifikation Complianceregeln verletzt hat. Die Umsetzung der automatischen Überprüfung von Complianceregeln erfüllt die Anforderung 4.

1.4.3. Algorithmus zur Überprüfung des Datenflusses in Prozessmodellen

Die in den vorhergehenden Abschnitten vorgestellten Konzepte zur Annotation und Verifikation von Complianceregeln auf Prozessmo-

dellen, beschäftigen sich ausschließlich mit dem Kontrollfluss eines Prozessmodells. Das heißt, es können hiermit Eigenschaften überprüft werden, die zum Beispiel die Abfolge bestimmter Aktivitäten betreffen.

Die automatische Untersuchung des Datenflusses ist ein weiterer wissenschaftlicher Beitrag dieser Arbeit, der in Abschnitt 5.3 beschrieben wird. Hierfür wurde das Konzept eines Compliancescopes erweitert und verallgemeinert. Diese Erweiterung wird *Compliancedomain* [SFG+11] genannt.

Compliancedomains dienen dazu, den Datenfluss in einem Prozessmodell einzuschränken. Wie mit Compliancescopes können auch mit Compliancedomains Bereiche in einem Prozessmodell definiert werden, in denen bestimmte Daten verarbeitet werden können, andere aber nicht. Im Gegensatz zu Compliancescopes ist der Anwendungsbereich von Compliancedomains viel weiter gefasst. Die mit Compliancedomains in Bereiche eingeteilten Prozessmodelle können anhand dieser Bereiche aufgespalten und auf verschiedenen Plattformen zum Laufen gebracht werden. Diese Plattformen können zum Beispiel das private Rechenzentrum einer Organisation oder eine Public-Cloud [MG09] sein. In diesem Beispiel können dann vertrauliche Daten des Prozesses im privaten Rechenzentrum der Organisation verarbeitet werden, während andere Operationen in der Public-Cloud ausgeführt werden. Das heißt, die Aktivitäten einer Compliancedomain, die die Verarbeitung von nicht sensiblen Daten erlaubt, können in einer Public-Cloud Infrastruktur ausgeführt werden, während andere Aktivitäten in einem privaten Rechenzentrum ausgeführt werden müssen. Compliancedomains erfüllen durch die Umsetzung im Prototyp die Anforderung 3 der Möglichkeit zur Definition von Complianceregeln, die den Datenfluss in einem Prozess einschränken.

1.4.4. Ein Mechanismus zur Unterstützung der Zusammenarbeit bei der Erstellung regelkonformer Prozesse

Bei der Erstellung von Prozessmodellen in Organisationen sind meist mehrere Personen in verschiedenen Abteilungen beteiligt. Diesen Personen kommen unterschiedliche Aufgaben bei der Erstellung eines Prozesses zu. Neben der Rolle eines Prozessmodellierers gibt es zum Beispiel auch Complianceexperten oder Sicherheitsexperten, die ihren Beitrag zur Erstellung eines Prozessmodells leisten.

Ein Konzept zur Koordination dieser, an der Erstellung eines Prozessmodells beteiligten Personen mit verschiedenen Rollen, stellt einen wissenschaftlichen Beitrag dieser Arbeit dar, der in Kapitel 6 beschrieben wird. Dieses Konzept heißt *Vervollständigungsebene* [SALS10]. Das Konzept einer Vervollständigungsebene dient dazu, Compliancetemplates schichtweise zu vervollständigen. Als erster Schritt wird ein Compliancetemplate von Personen mit einer bestimmten Rolle durch das Einfügen neuer Aktivitäten in freie Stellen des Prozesses, sogenannte Complianceregionen, verfeinert. Diese Aktivitäten können auch wieder Complianceregionen sein. Werden weitere Complianceregionen in vorhandene Complianceregionen eingefügt, so öffnet dies eine weitere Vervollständigungsebene. Das so veränderte Compliancetemplate wird an eine andere Personengruppe oder Abteilung einer Organisation weitergegeben und vervollständigt. Mehrere Personen können so in die schichtweise Entwicklung eines Prozesses eingebunden werden, indem sie Compliancetemplates zu einem syntaktisch korrekten Prozess vervollständigen. In jeder Modellierungsschicht können von unterschiedlichen Personen mit unterschiedlichen Anforderungen und Qualifikationen Änderungen vorgenommen werden. Das Konzept der

Vervollständigungsebenen erfüllt Anforderung 2 durch die Umsetzung im Prototyp dieser Dissertation.

1.4.5. Architektur eines Prototyps zur Evaluierung der vorgestellten Konzepte und Algorithmen

Die in den vorhergehenden Abschnitten vorgestellten wissenschaftlichen Beiträge sind prototypisch umgesetzt. Die Architektur dieses Prototyps stellt einen wesentlichen wissenschaftlichen Beitrag dieser Arbeit dar. Die Architektur und die Entscheidungswege, die zu ihr führten, sind in einem eigenen Kapitel beschrieben (siehe Kapitel 7). Dieses Kapitel zeigt die Komplexität der Lösung sowie die erstellten Softwarekomponenten, um die neu entwickelten Konzepte dieser Dissertation zu implementieren. Ferner zeigt es die Lösungen für Probleme, die bei der Entwicklung und Integration der Softwarekomponenten auftraten. Der Prototyp verbindet die neuen Konzepte dieser Dissertation und zeigt wie diese zusammen spielen, um eine vollständige Lösung für die regelkonforme Prozessentwicklung zu bilden. Der Prototyp basiert auf dem web-basierten BPMN Editor Oryx [DOW08]. Außerdem setzt der Prototyp die Anforderung 5 um, indem er dem Benutzer nach einer Überprüfung bei Verletzung einer Complianceregel anzeigt, in welchem Bereich des Prozesses die Complianceregel verletzt wurde. Weiterhin wird der Ausführungspfad, der zur Verletzung der Complianceregel führt, angezeigt.

1.5. Definition des Arbeitsbereichs

Diese Arbeit präsentiert Konzepte für die automatische Überprüfung von Prozessmodellen. Diese Konzepte greifen in der ersten, der Entwicklungsphase, des BPM-Lebenszyklus. Prozessmodelle werden mittels graphischer Werkzeuge entwickelt. Die neuen Konzepte können in graphische Entwicklungswerkzeuge eingebaut werden. Mit diesen Konzepten ist es möglich, jede Veränderung an einem Prozessmodell zu überprüfen. Ein Prozessmodellierer kann darauf aufmerksam gemacht werden, wenn seine Änderung an einem Prozessmodell eine Complianceregel verletzt.

Die Übersetzung von Gesetzestexten in formale Ausdrücke ist eine Aufgabe, die nicht in dieser Dissertation bearbeitet wird. Diese Arbeit bietet eine Plattform, die bei angemessener Verwendung, die Entwicklung regelkonformer Prozesse unterstützt. Die Plattform wird angemessen verwendet, wenn sie so eingesetzt wird, dass menschliche Prozessmodellierer so wenig wie möglich daran gehindert werden, ihr Ziel der Erstellung neuer Prozesse zu verfolgen.

Die wissenschaftlichen Beiträge dieser Arbeit sind im Bereich der graphischen Modellierung von Prozessen und der Architektur des Prototyps angesiedelt. Um die in dieser Arbeit neu vorgestellten Konzepte im Prototyp zu realisieren, wurden Arbeiten aus dem Bereich des Modelchecking und der Petrinetze verwendet. Die Verwendung dieser Arbeiten ermöglicht es, Prozessmodelle automatisch im Prototyp zu überprüfen.

Complianceregeln können ein sehr weites Feld von Eigenschaften abdecken, die für einen Prozess gelten müssen. Denkbare Eigenschaften sind zum Beispiel der Energieverbrauch eines Prozesses zur Laufzeit,

der zur Entwicklungszeit berechnet werden könnte oder eine Obergrenze für die Antwortzeit, die ein Prozess zur Laufzeit nicht überschreiten darf. Die im Folgenden vorgestellten Lösungen beziehen sich jedoch auf den Kontrollfluss und den Datenfluss eines Prozesses. Das Ziel, das mit dieser Einschränkung erreicht wird, ist, eine umfassende Lösung zu präsentieren, die für diese Arten von Prozesseigenschaften angewendet werden kann.

Die Eckpunkte dieser Arbeit werden in der folgenden Liste vorgestellt:

- Die verwendeten Complianceregeln wurden nicht aus Gesetzestexten abgeleitet. Jedoch beruhen die Beispiele dieser Arbeit auf Beispielen aus der Literatur und Erfahrungen, die in den EU-Projekten MASTER[1] und COMPAS[2] gemacht wurden.

- Der durchweg in dieser Arbeit verwendete Beispielprozess stammt aus der Literatur [TLF+10]. Er ist ein Modell eines real existierenden Prozesses in einem Krankenhaus.

Zusammenfassend zeigt diese Arbeit erste Schritte in Richtung eines Werkzeugs für die regelkonforme Entwicklung von Prozessen. Die Beiträge liegen auf der Ebene der graphischen Prozessentwicklung. Sie reichern Prozesse mit Informationen an, die von Werkzeugen verwendet werden, um sie zur Entwicklungszeit auf Verstöße gegen Complianceregeln zu untersuchen.

In der Arbeit sind alle graphisch dargestellten Prozesse mit der Prozessbeschreibungssprache Business Process Model and Notation 1.0

[1] http://www.master-fp7.eu
[2] http://compas-ict.eu

(BPMN 1.0) [Bus04] erstellt. Da die Ausführungssemantik von BPMN 1.0 in der Spezifikation nicht formal beschrieben ist, wird die Ausführungssemantik von BPMN 2.0 [Obj11] verwendet. Weiterhin wird der Erweiterungsmechanismus von BPMN 2.0 verwendet. BPMN ist ein weit verbreiteter Standard zur Beschreibung von Prozessmodellen, der in dieser Arbeit im Gegensatz zu nicht standardisierten Prozessnotationen verwendet wird, um die Möglichkeit von Fehlinterpretationen von Prozessen gering zu halten.

Die in dieser Arbeit vorgestellten Techniken und Lösungsansätze sind jedoch erweiterbar und bilden daher die Grundlage für die Ausweitung des Einsatzgebiets.

1.6. Aufbau der Arbeit

Diese Dissertation ist wie folgt gegliedert. Kapitel 2 präsentiert die Technologien und Konzepte auf denen diese Arbeit aufbaut. Es legt die Grundlagen, die zum Verständnis der Arbeit notwendig sind. Kapitel 3 erörtert verwandte Arbeiten für den Bereich der regelkonformen Prozessentwicklung. Die automatische Überprüfung des Kontrollflusses von Prozessen zur Entwicklungszeit ist ein Thema, das in Kapitel 4 bearbeitet wird. Hier werden zwei Konzepte vorgestellt, die menschlichen Prozessmodellierern dabei helfen sollen, einen Prozess mit regelkonformem Kontrollfluss zu erstellen. In Kapitel 5 werden Konzepte präsentiert, die den Datenfluss eines Prozesses zur Entwicklungszeit automatisch überprüfen. Die Kombination von kontrollfluss- mit datenflussbasierten Complianceregeln ist Thema des Kapitels 5.5. Ein Mechanismus zur Erstellung regelkonformer Prozesse unter Einbindung mehrerer Partner stellt Kapitel 6 vor. Die Architektur und

Implementierung des Prototyps wird in Kapitel 7 erläutert. Kapitel 8 fasst die in dieser Arbeit gezeigten Erkenntnisse zusammen und gibt einen Ausblick auf mögliche darauf aufbauende Arbeiten.

KAPITEL 2

GRUNDLAGEN DER ENTWICKLUNG REGELKONFORMER PROZESSE

2.1. Bedeutung des Begriffs Compliance im Kontext dieser Arbeit

Der Begriff *Compliance* wird in dieser Arbeit mit dem deutschen Begriff *Regelkonformität* gleichgesetzt. Das Thema *Compliance von Prozessen* beschäftigt sich mit der Einhaltung von Richtlinien und Regeln, die auf Prozesse Anwendung finden.

Compliance ist ein Thema, das Firmen heutzutage immer mehr bewegt und auch in Zusammenhang mit Geschäftsprozessen in Zukunft bewegen wird [MPRS13, SGN07, KSMP07, LSG08]. Gerade im

Bereich des Prozessmanagement ist Compliance somit unverzichtbar. Ein Beispiel hierfür ist die weltweite Finanzkrise, die damit begann, dass 2007 die US Immobilienblase platzte. Viele Banken hatten an Privatpersonen Kredite vergeben, die später von den Kreditnehmern nicht zurückbezahlt werden konnten.

Regeln können aus verschiedenen Quellen stammen. Folgende drei Quellen sind relevant: Gesetzestexte, Regeln, die durch Werte der Gesellschaft geprägt sind und firmeninterne Regeln. Beispiele für wichtige Gesetze im Bereich Business Process Management sind Basel II [Bas06] und der Sarbanes-Oxley Act (SOX) [Uni02].

Diese Arbeit leitet keine auf Prozesse anwendbare Complianceregeln aus Gesetzestexten ab. Die in den folgenden Kapiteln gezeigten Regeln sind fiktiv. Sie leiten sich aus den Erfahrungen des Autors mit wissenschaftlichen Projekten aus dem Bereich Compliance von Prozessen ab.

2.2. Business Process Management

Beginnend mit der Entwicklungsphase im Geschäfsprozesslebenszyklus beschäftigt sich das Business Process Management mit allen Tätigkeiten, die bei der Arbeit mit Geschäftsprozessen anfallen.

2.3. Business Process Model and Notation 1.0 (BPMN 1.0)

BPMN wurde in der Version 1.0 als eine graphische Notation zur Beschreibung von Geschäftsprozessen entworfen. Sie besteht aus einer Vielzahl graphischer Symbole, von denen die wichtigsten in den folgenden Abschnitten vorgestellt werden. Im Januar 2011 wurde die Version 2.0 von BPMN fertiggestellt. Zu den wichtigsten Neuerungen von BPMN 2.0 zählen die vollständige Spezifikation der Ausführungssemantik und eine Beschreibung der Verarbeitung von Daten in BPMN-Prozessen. Die Beschreibung der Ausführungssemantik und der Datenflusssemantik von BPMN 2.0-Prozessen ist für diese Arbeit grund-

legend, da Complianceregeln den Kontrollfluss und den Datenfluss in einem Prozess betreffen. Ohne die Einführung dieser Semantiken hätten automatische Prüfverfahren keine Grundlage zur Überprüfung dieser Complianceregeln. In dieser Arbeit wird aus Gründen der Komplexitätsreduktion BPMN 1.0 zusammen mit der Ausführungssemantik und dem Erweiterungsmechanismus von BPMN 2.0 verwendet, da in BPMN 2.0 unter anderem auch einige neue Symbole eingeführt wurden, die für die Ergebnisse der vorliegenden Arbeit nicht ausschlaggebend sind.

Im Folgenden werden die wichtigsten graphischen Elemente von BPMN 1.0 erläutert. Die Auswahl der dargestellten BPMN Elemente erfolgte im Hinblick auf die Verwendung im Rahmen der vorliegenden Arbeit.

2.3.1. Tasks

Tasks werden in BPMN 1.0 als Rechtecke mit abgerundeten Ecken dargestellt. Die verschiedenen Tasks unterscheiden sich durch die Linienbreite des Rechtecks und durch das im Rechteck enthaltene Piktogramm. Tasks können als einzelne auszuführende Schritte in einem Prozess angesehen werden. Abbildung 2.1 zeigt zwei Tasks.

2.3.2. Kontrollfluss

Der Begriff Kontrollfluss bezeichnet im Zusammenhang mit Geschäftsprozessen die Einschränkung der möglichen Ausführungsabfolgen von Aktivitäten. Realisiert wird dies zum Beispiel in BPMN 1.0 mit Hilfe von Kontrollflusskonnektoren (siehe Abbildung 2.1). Dies sind Pfeile, die die Aktivitäten eines Prozessmodells verbinden und so eine

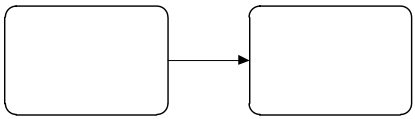

Abbildung 2.1.: Einschränkung der Möglichkeiten der Abfolge von Aktivitäten mit Hilfe eines Kontrollflusskonnektors

Ausführungsreihenfolge vorgeben.

2.3.3. Datenfluss

Für die Ausführung einer Aktivität werden Eingabedaten benötigt. Weiterhin wird von einer Aktivität in der Regel ein Resultat in Form von Ausgabedaten erzeugt. Diese Ausgabedaten werden mittels Datenkonnektoren zum Beispiel zu weiteren Aktivitäten oder Datenobjekten weitergeleitet. Die so entstehende Bewegung von Daten in einem Prozess wird als Datenfluss des Prozesses bezeichnet.

2.3.4. Ereignisse

Ereignisse können in Prozessmodellen verwendet werden, um bestimmte Situationen, die zur Ausführungszeit auftreten können, zu modellieren. Es gibt drei Arten von Ereignissen in BPMN 1.0: Startereignisse, Intermediate-Ereignisse und Endereignisse. Abbildung 2.2 zeigt beispielhaft ein Startereignis. Alle Ereignisse in BPMN sind durch eine Kreisform gekennzeichnet. Ereignisse unterscheiden sich durch die Linienbreite des Kreises und durch das im Kreis enthaltene Piktogramm. Das Startereignis enthält kein Piktogramm. Es zeigt an, wo ein Prozess gestartet werden kann.

Abbildung 2.2.: Beispiel für ein Ereignis in BPMN: Startereignis

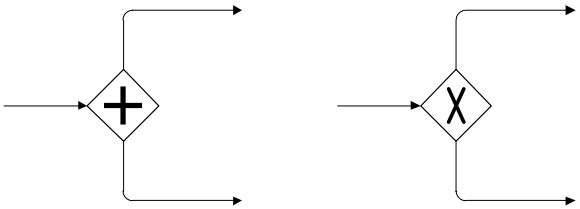

Abbildung 2.3.: Und-Gateway links; Exklusiv-oder-Gateway rechts

2.3.5. Gateways

Gateways werden in BPMN 1.0 dazu verwendet, Verzweigungen in Prozessmodellen zu modellieren. Abbildung 2.3 zeigt zwei der wichtigsten Gateways: das Und-Gateway (links) und das Exklusiv-oder-Gateway (rechts).

Mit dem Und-Gateway kann der Kontrollfluss in einem Prozessmodell in zwei oder mehrere Zweige aufgeteilt werden, die parallel ausgeführt werden. Mit dem Exklusiv-oder-Gateway kann der Kontrollfluss im Prozessmodell anhand von Bedingungen aufgeteilt werden. Die ausgehenden Kontrollflusskonnektoren eines Exklusiv-oder-Gateways sind jeweils mit Bedingungen verknüpft. Ist eine solche Bedingung wahr, so wird der Prozess in dem Zweig weiter ausgeführt, mit dem die Bedingung verknüpft ist.

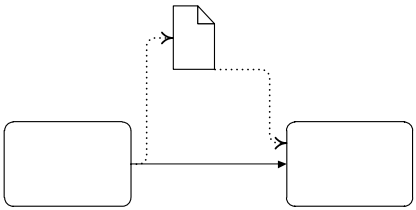

Abbildung 2.4.: Datenobjekt

2.3.6. Datenobjekte

Datenobjekte dienen der graphischen Abbildung des Datenflusses in einem Prozessmodell. Abbildung 2.4 zeigt zwei miteinander durch einen Kontrollflusskonnektor verbundene Tasks. Weiterhin zeigt die Abbildung den Datenfluss, der durch die mit dem Datenobjekt und den Tasks verbundenen Datenassoziationen (gestrichelte Pfeile) dargestellt ist.

2.4. Oryx

Oryx [DOW08] ist ein webbasierter, graphischer BPMN-Editor. Er wird in dieser Arbeit als Grundlage für die prototypische Implementierung der neuen Konzepte verwendet. Oryx wurde am Hasso-Plattner-Institut entwickelt und ist in zwei Hauptkomponenten aufgeteilt. Das Frontend ist eine hauptsächlich in JavaScript entwickelte Komponente, die die graphische Entwicklungsumgebung für den Benutzer bereitstellt. Sie wird in einem Browser ausgeführt. Das Backend ist hauptsächlich in Java geschrieben und bietet unter anderem Funktionalität zum Speichern und Laden von Prozessmodellen.

2.5. Lineare Temporale Logik

Das Konzept der Linearen Temporalen Logik [Pnu77, Pnu86] (LTL) wurde Anfang der 1980er Jahre von Armir Pnueli in die Informatik eingeführt. Anwendungsgebiete für LTL sind Zustandssysteme wie zum Beispiel Betriebssysteme oder Protokolle. Zustandssysteme generieren bei der Ausführung Abfolgen von Zuständen, die sie erreicht haben. Mit LTL ist es möglich diese Abfolgen von Zuständen zu beschreiben.

Heute wird die temporale Logik in vielen Anwendungsgebieten eingesetzt. Ein Beispiel ist das Feld der Prozessverwaltung.

Mit Linearer Temporaler Logik wird eine Sprache bezeichnet, in der ausschließlich mit diskreten Zeitschritten gearbeitet wird. Das bedeutet, dass zwischen zwei aufeinander folgenden Zuständen eines reakiven Systems immer ein Zeitintervall der Länge eins liegt.

Mit V als eine Menge aussagenlogischer Konstanten ist die Sprache $\mathcal{L}_{LTL}(V)$ der Linearen Temporalen Aussagenlogik wie folgt definiert: Das Alphabet von $\mathcal{L}_{LTL}(V)$ besteht aus...

- allen aussagenlogischen Konstanten von V
- und den Zeichen **false** | □ | ◇ |○ | U | W | → | (|).

Weiter definieren wir:

- Jede aussagenlogische Konstante V_k aus V ist eine Formel.
- **false** ist Formel.
- Wenn A und B Formeln sind, dann sind □A, ◇A, ○A, A U B, A W B und A ⇒ B Formeln.

Es können die aus der Aussagenlogik bekannten Operatoren ¬, ∧, ∨, ⇔ und **true** verwendet werden.

Die oben neu eingeführten Operatoren haben die folgenden Bedeutungen:

- □A: A ist in allen folgenden Zeitpunkten wahr.
- ◇A: A ist nach einer bestimmten Zeitspanne wahr.
- ○A: A ist zum nächsten Zeitpunkt wahr.
- A U B: A ist so lange wahr, bis B wahr ist. Danach darf A nicht mehr wahr sein.
- A W B: A ist so lange wahr, bis B wahr ist. Danach darf A nicht mehr wahr sein. Oder A ist immer wahr.

Weiterhin gilt: ◇ A = ¬□¬ A.

2.6. Modelchecking

Unter Modelchecking wird die automatische Überprüfung eines Modells anhand formal definierter Eigenschaften verstanden, die dieses Modell erfüllen muss [JGP99]. Das Modell bildet dabei die wichtigsten Eigenschaften eines real existierenden Systems ab. Ein solches System kann zum Beispiel ein Geschäftsprozess oder ein anderes verteiltes System sein.

2.7. SPIN

SPIN [Hol03] ist ein in den Bell Labs entwickelter Modelchecker, der seit 1991 frei verfügbar ist. Die Eingabesprache von SPIN ist PROMELA.

Zur Spezifikation der Eigenschaften, die ein zu überprüfendes Modell erfüllen muss, dient die Lineare Temporale Logik.

2.8. PROMELA

PROMELA (Process oder Protocol Meta Language) [Hol03] ist die Eingabesprache des SPIN Modelcheckers. Ursprünglich wurde sie zur Erstellung von nebenläufigen, verteilten Systemen konzipiert. Aufgrund ihrer Mächtigkeit kann sie jedoch auch in anderen Bereichen, zum Beispiel zur Erstellung von Petrinetzmodellen, verwendet werden. Die Kommunikation zwischen zwei Komponenten wird in PROMELA mittels Nachrichtenkanälen modelliert. Diese können sowohl synchron als auch asynchron arbeiten. PROMELA hat eine an die Programmiersprache C angelehnte Syntax.

2.9. JSON

JSON[1] ist ein Datenformat, mit dem es möglich ist, komplexe Datenobjekte über textbasierte Datentransferprotokolle, wie zum Beispiel HTTP, zu übertragen. JSON wird hier beschrieben, da der in der vorliegenden Arbeit vorgestellte Algorithmus 4.1 damit arbeitet.

Listing 2.1: Arrayrepräsentation der Plätze eines Petrinetzes

```
{
  "Name": "Max Musterknabe",
  "Alter": 32,
  "Adresse": {
    "Street": "Langestraße",
```

[1] http://json.org/json-de.html

```
    "City": "Musterstadt"
  },
  "Kinder": [
    {
      "Name": "Maike"
    },
    {
      "Name": "Karl"
    }
  ]
}
```

Codelisting 2.1 zeigt ein JSON-Datenobjekt, das eine Person beschreibt. Hierzu werden die gängigsten Modellierungskonstrukte von JSON verwendet. Objekte sind in JSON in geschweifte Klammern eingerahmt. Innerhalb dieser Objekte werden Schlüsselwertpaare dazu verwendet, um einem Objekt Daten zuzuordnen. Weiterhin können Arrays definiert werden. Diese sind mit eckigen Klammern eingerahmt.

2.10. Prozessfragment

Eberle et al. definieren Prozessfragmente [EUL09] als Teilstücke von Prozessen. Dabei wird beschrieben, dass Prozessfragmente in sich unvollständig spezifiziert sein können. So können sie beispielsweise Lücken aufweisen. Für Lücken ist nicht definiert, wie der Prozess sich dort zur Ausführungszeit verhält. Um einen vollständigen Prozess zu erhalten, müssen Lücken mit Aktivitäten gefüllt werden. Weiterhin können Prozessfragmente Kontrollflusskonnektoren enthalten, die entweder an deren Anfang oder Ende nicht mit einer Aktivität verbunden sind.

KAPITEL 3

VERWANDTE ARBEITEN

Dieses Kapitel setzt die in dieser Arbeit vorgestellten Konzepte in den aktuellen wissenschaftlichen Kontext und gibt einen Forschungsüberblick. Die Abschnitte dieses Kapitels sind so aufgebaut, dass zuerst die verwandten Arbeiten zu einem speziellen Themengebiet vorgestellt werden. Anschließend werden die verwandten Arbeiten in den Zusammenhang mit dieser Dissertation gebracht. Die behandelten Themengebiete sind dabei nach dem Abstraktionsgrad sortiert. Zu Beginn wird ein Überblick über Arbeiten im Bereich der Unterstützung menschlicher Prozessmodellierer bei der Entwicklung regelkonformer Prozesse (Abschnitt 3.1) gegeben. Weiterhin werden die Bereiche des regelkonformen Geschäftsprozessmanagements (Abschnitt 3.2) und der regelkonformen Prozessmodellierung (Abschnitt 3.3) bearbeitet. Den Schluss bilden Arbeiten zu Linearer Temporaler Logik und Modelchecking (Abschnitt 3.4).

3.1. Unterstützung menschlicher Prozessmodellierer

Das Hauptziel dieser Arbeit ist es, einen menschlichen Prozessmodellierer dabei zu unterstützen, regelkonforme Prozesse zu entwickeln. Die Unterstützung soll dabei so weit gehen, dass der menschliche Prozessmodellierer nur mit Complianceproblemen konfrontiert wird, wenn eine Complianceregel verletzt wird. Im Folgenden werden Arbeiten aufgezeigt, die Lösungen für die Unterstützung menschlicher Prozessmodellierer anbieten.

Awad et al. zeigen in [ADW08] einen Ansatz zur Überprüfung von Prozessmodellen mit dem Ziel Verstöße gegen Complianceregeln aufzudecken. Es werden in diesem Ansatz Complianceregeln verwendet, die den Kontrollfluss in einem Prozessmodell einschränken. Zur Überprüfung von Complianceregeln wird in diesem Ansatz BPMN-Q, eine Abfragesprache für BPMN-Modelle in Repositories verwendet, um die Prozessmodelle in einem Repository herauszufiltern, die für eine Überprüfung auf Verstöße gegen Complianceregeln in Frage kommen. Die in Frage kommenden Prozessmodelle werden mit Hilfe eines Modelcheckers auf Verstöße gegen Complianceregeln überprüft. Diejenigen Prozessmodelle, die gegen die zur Überprüfung verwendete Complianceregel verstoßen, werden dem Benutzer angezeigt. Ein Prozessmodell, das gerade bearbeitet wird, muss folglich vor der Überprüfung auf Complianceverstöße in das Repository eingecheckt werden. Das Konzept wird in [AWW09] um datenbasierte Complianceregeln erweitert.

Weiterhin zeigen Awad et al. in [AW09] eine auf dem vorhergehenden Ansatz aufbauende Lösung zur Überprüfung von Prozessmodellen in einem Repository auf Verletzungen von Complianceregeln. In die-

sem Ansatz werden alle Prozessmodelle in einem Repository auf Verletzungen von Complianceregeln untersucht. Die Erweiterung des vorhergehenden Ansatzes besteht in der automatischen Generierung von sogenannten Anti-Patterns aus BPMN-Q-Anfragen. Anti-Patterns sind BPMN-Q-Anfragen, die auf einem Prozessmodell ausgeführt werden. Ist eine solche Anfrage erfolgreich, so ist das betreffende Prozessmodell nicht regelkonform. Anti-Patterns zeigen dem Prozessmodellierer die Stelle in einem Prozessmodell an, die eine bestimmte Complianceregel verletzt. Ein weiterer Vorteil von Anti-Patterns besteht in der Vermeidung der Verwendung von Modelcheckern, um Verletzungen von Complianceregeln in einem Prozessmodell anzuzeigen.

In seiner Dissertation zeigt Awad [Awa10], aufbauend auf den oben besprochenen Veröffentlichungen, wie ein ganzheitlicher Ansatz aussehen kann, der einen menschlichen Prozessmodellierer bei der Entwicklung regelkonformer Prozesse unterstützt. BPMN-Q Anfragen dienen hier zur graphischen Illustration von Complianceregeln. Dies soll es Prozessmodellierern mit nicht-technischem Hintergrund erleichtern, mit Complianceregeln umzugehen. Diese graphische Repräsentation von Complianceregeln wird dann in eine formale Sprache transformiert, die als Eingabesprache für einen Modelchecker dient. Da dies automatisch geschieht, muss sich der menschliche Prozessmodellierer nicht mit den technischen Eigenheiten der hinter der Lösung liegenden Konzepte auseinandersetzen, möchte er einen regelkonformen Prozess erstellen.

In [TEHP11] werden Muster für die am häufigsten in der Literatur erwähnten Complianceregeln vorgestellt. Diese dienen der einfacheren Arbeit mit Complianceregeln für Menschen ohne technischen Hintergrund. Referenziert werden die Muster anhand ihres Namens. Somit

bleibt dem menschlichen Prozessmodellierer die formale Repräsentation bei der Arbeit mit einer Complianceregel verborgen. Jedes in [TEHP11] vorgestellte Muster beschreibt eine Complianceregel, die den Kontrollfluss in einem Prozessmodell einschränkt.

Datenbasierte Complianceregeln sind das Thema in [KLRM+10]. Sie werden in dieser Arbeit so aufbereitet, dass sie mit einem Modelchecker überprüft werden können. Bei der Überprüfung ganzer Prozessmodelle wird der menschliche Prozessmodellierer mit dem graphischen Werkzeug *Aristaflow Process Template Editor* unterstützt. Dieses Werkzeug stößt auch die Untersuchung des angezeigten Prozessmodells auf Verletzungen von datenbasierten Complianceregeln an.

In [BDSV05] wird eine Abbildung von LTL-Formeln auf BPMN-Konstrukte gezeigt. Dies ist eine Grundlage für die Unterstützung menschlicher Prozessmodellierer beim Umgang mit Complianceregeln, da Complianceregeln in einer formalen Sprache vorliegen müssen, damit sie als Eingabe für die automatische Überprüfung von Prozessmodellen verwendet werden können.

Die automatische Überprüfung von Complianceregeln auf Prozessmodellen ist Thema in [LMX07]. Es wird hier darauf Wert gelegt, dass Prozessmodelle und Complianceregeln getrennt voneinander erstellt und bearbeitet werden können. In diesem Ansatz wird ein in der Business Process Execution Language (BPEL) geschriebenes Prozessmodell in ein π-Kalkül-Modell übertragen. Complianceregeln werden in einer graphischen Modellierungssprache erstellt und danach in LTL transformiert. Modelchecking wird verwendet, um Prozessmodelle automatisch auf Verletzungen von Complianceregeln zu untersuchen.

In der in [Elg12] vorgestellten Dissertation wird ein Rahmenwerk

für die Verwaltung und Erstellung regelkonformer Geschäftsprozesse beschrieben. Hier wird ein Ansatz gezeigt, der es menschlichen Prozessmodellierern erleichtern soll, mit formalen Ausdrücken umzugehen, die gebraucht werden, um Complianceregeln zu spezifizieren. Dies geschieht mit Hilfe von Vorlagen. Vorlagen können mit Hilfe ihres Namens referenziert werden. Sie werden als atomare Complianceregeln gesehen. In diesen Vorlagen verbergen sich die komplexen formalen Ausdrücke, die Complianceregeln beschreiben. Die Arbeit beschreibt weiter, wie menschliche Prozessmodellierer diese Vorlagen verwenden, um Prozessmodelle auf Regelverletzungen zu überprüfen. Zur Überprüfung von Prozessmodellen werden Techniken aus dem Modelchecking herangezogen. Konkret wird LTL verwendet, um Complianceregeln zu spezifizieren. Der SPIN Modelchecker dient als Untersuchungswerkzeug.

In den in diesen Arbeiten vorgestellten Lösungen werden ganze Prozessmodelle als Grundlage für die Überprüfung von Complianceverstößen herangezogen. Viele Lösungen verwenden Techniken aus dem Modelchecking, um die automatische Überprüfung von Prozessmodellen durchzuführen. Der Vorteil von Modelcheckern ist die Qualität der Untersuchungsergebnisse. Verletzt ein Prozessmodell eine Complianceregel, so wird dieser Fehler gefunden. Weiterhin wird ein Beispielpfad im betreffenden Prozessmodell gezeigt, der zu diesem Fehler führte. Der Nachteil von Modelcheckern ist deren Laufzeit. Bei der Überprüfung ganzer Prozessmodelle kann es zu einer sehr langen Laufzeit eines Modelcheckers kommen, da sie ein exponentielles Laufzeitverhalten haben [Var01].

Weiterhin werden Lösungen präsentiert, die zeigen, wie formale Sprachen verwendet werden, um Complianceregeln auszudrücken.

Einige Ansätze verwenden graphische Notationen, um Complianceregeln zu erstellen. Aus diesen graphischen Modellen werden dann die formalen Modelle erstellt, die als Eingabesprache für Modelchecker verwendet werden.

Die vorliegende Dissertation hingegen legt Wert darauf, eine Lösung zu präsentieren, die bei der Entwicklung von Prozessen den Menschen in den Vordergrund stellt. Bei der Erstellung der Konzepte stand die Benutzerfreundlichkeit im Vordergrund. Der Hauptaspekt, der in diesem Zusammenhang betrachtet wird, ist die Antwortzeit des zu erstellenden Prototyps. Diese sollte so gering wie möglich sein [Nie93]. Die Einbeziehung der Benutzerfreundlichkeit ist notwendig, da die Erstellung regelkonformer Prozesse immer von Menschen durchgeführt wird.

Jedes in dieser Arbeit vorgestellte Konzept wurde mit Blick auf die Benutzerfreundlichkeit entwickelt. Gegenüber bereits existierenden Ansätzen ist es zum Beispiel möglich, Complianceregeln mit frei definierbaren Bereichen in einem Prozessmodell zu verknüpfen. Mit der Größe des mit einer Complianceregel verknüpften Prozessbereichs kann zum Beispiel die Wartezeit auf ein Überprüfungsergebnis angepasst werden.

3.2. Regelkonformes Geschäftsprozessmanagement

Wie in Kapitel 1 dargestellt, ist die Einhaltung von Regularien im Feld des Geschäftsprozessmanagement in den letzten Jahren zu einem zentralen Thema des Unternehmensmanagements avanciert. Diese Entwicklung führte auch in der Wissenschaft zu verstärkten Anstrengungen in dieser Richtung neue Konzepte und Lösungen zu präsentie-

ren.

Zwei von der Europäischen Union (EU) geförderte Projekte beschäftigten sich mit neuen Konzepten in unterschiedlichen Teilen des Geschäftsprozesslebenszyklus. Im ersten EU-Projekt mit dem Namen Compliance-driven Models, Languages, and Architectures for Services (COMPAS)[1] wurde unter anderem der Geschäftsprozesslebenszyklus um einige Phasen erweitert, die für den Bereich Compliance wichtig sind. Es entstand der Compliancelebenszyklus. Im Compliancelebenszyklus wurden zu den bestehenden Phasen Modellierung, Ausführung, Überwachung und Prüfung des Geschäftsprozesslebenszyklus die Phasen Compliancebeurteilung, Anpassung der IT-Systeme und statische Überprüfung hinzugefügt. COMPAS befasste sich hauptsächlich mit den Unterphasen der Entwicklungsphase des Compliancelebenszyklus. Für die Phase *Compliancebeurteilung* wurde erforscht, wie sich natürlichsprachliche Gesetzestexte in Ausdrücke übersetzen lassen, die mit einer formalen Sprache geschrieben sind. Weiterhin wurde für die Phase der Modellierung das Konzept eines Compliancefragments entwickelt [SLM+10]. Compliancefragmente bauen auf der Definition von Prozessfragmenten [EUL09] auf. Sie sind Prozessteile, die nicht ausführbar sind, weil sie typischerweise nicht voll spezifiziert sind. Ein Compliancefragment erfüllt durch seine Implementierung einen oder mehrere Complianceanforderungen. Compliancefragmente sind dazu gedacht, in bestehende Prozessmodelle eingefügt zu werden. Das Einfügen eines Compliancefragments passt das Verhalten des betreffenden Prozesses in bestimmten Situationen an Vorgaben an.

Die Hauptherausforderungen der Verwaltung von Compliancefrag-

[1] http://compas-ict.eu

menten werden im Artikel [SLS10] herausgestellt. Weiterhin werden Techniken für das Herausfiltern und Verbergen von Compliancefragmenten gezeigt, die auf Transformationen aufbauen, die für die Erstellung von Prozesssichten entwickelt wurden. Dies geschieht mit Hilfe von Transformationsvorschriften, die ein Prozessmodell in eine Prozesssicht übertragen.

In [SAL+10] wird ein Verwaltungsmodell für Complianceregeln im Bereich des Geschäftsprozessmanagement vorgestellt. Dieses Verwaltungsmodell zeigt, dass Compliance nicht nur für die in einem Unternehmen eingesetzten Prozesse wichtig ist, sondern auch für alle für die Ausführung eines Prozesses benötigten Ressourcen. In diesem Verwaltungsmodell werden Compliancekontrollen in zwei Typen aufgeteilt: Kontrollen die beschreiben wie Complianceanforderungen untersucht werden müssen und Kontrollen die beschreiben, wie ein Prozess abzulaufen hat. Weiterhin wird gezeigt, wie BPEL erweitert werden kann, um Compliancefragmente mit BPEL zu verwenden.

Auch der Artikel [SLM+10] befasst sich mit Prozessfragmenten und deren Einsatz im Bereich der regelkonformen Prozessmodellierung. Es werden hier zwei Mechanismen für den Einsatz von Prozessfragmenten in Prozessen gezeigt. Der erste Mechanismus, das Ankleben von Prozessfragmenten an Prozesse, fügt Prozessfragmente direkt in einen Prozess ein, so dass sie Teil des Prozesses werden. Der zweite Mechanismus, das Hineinweben von Prozessfragmenten in Prozesse, verwendet eine Softwarekomponente. Diese überwacht die Ausführung eines Prozessmodells und führt an bestimmten Stellen die in den Prozess hineingewebten Prozessfragmente aus.

Der Beitrag von COMPAS in der Phase *statische Überprüfung* befasst sich mit der automatischen Überprüfung von Complianceregeln auf

Prozessmodellebene. Hier werden Techniken aus dem Modelchecking verwendet, um ganze Prozessmodelle automatisch zu überprüfen [STK+10].

Das zweite EU-Projekt mit dem Namen Managing Assurance, Security and Trust for sERvices (MASTER)[1] erforschte den gesamten Geschäftsprozesslebenszyklus mit Fokus auf die Ausführungszeit von IT-unterstützten Geschäftsprozessen. In der Phase der Modellierung wurden hier Konzepte entwickelt, um Informationen, die für spätere Phasen im Prozesslebenszyklus wichtig sind, in Prozessmodelle einzufügen. In der Ausführungsphase wurde ein Konzept mit dem Namen *Enforcement* [AKL+09, GCS+10] verwendet. Mit diesem Konzept ist es möglich, zur Laufzeit Gegenmaßnahmen zu ergreifen, sollte ein Prozess Complianceregeln verletzen. Weiterhin wurden in der Phase Überwachung Konzepte und Prototypen entwickelt, um anhand von Laufzeitereignissen, zu ermitteln, ob von einem laufenden Prozess nicht erlaubte Aktionen durchgeführt werden.

In der Phase Prüfung wurden Konzepte und Prototypen entwickelt, um anhand der Ausführungsdaten vieler abgelaufener Prozesse zu entscheiden, ob die ihnen zugrunde liegenden Prozessmodelle geändert werden müssen, um den ihnen auferlegten Richtlinien zu genügen.

In [WPD+11] zeigen Weidlich et al., dass das Konzept der *Behavioural Profiles* (Verhaltensprofile) auch auf Compliance in Prozessmodellen angewendet werden kann. Behavioural Profiles werden dazu verwendet, um Aussagen über miteinander in Beziehung stehenden Paaren von Aktivitäten in Prozessmodellen zu machen. In dem vorgestellten Ansatz werden sie als Metrik verwendet, um die Regelkon-

[1] http://www.master-fp7.eu

formität von Prozessmodellen für drei Arten von Complianceregeln zu überprüfen. Es können damit die Ausführungsreihenfolge von Aktivitäten, die zwingende Ausführung einer Aktivität und die kausale Abhängigkeit von Aktivitäten zueinander überprüft werden.

Die in diesem Abschnitt vorgestellten Arbeiten haben das Ziel, viele Teilbereiche des Prozesslebenszyklus um Konzepte zur Einhaltung und Überprüfung von Complianceregeln zu erweitern.

Die vorliegende Arbeit befasst sich mit der ersten Phase des Prozesslebenszyklus, der Modellierungsphase. Sie präsentiert hier Konzepte, die sich eingehend mit dem Thema der Unterstützung menschlicher Prozessmodellierer bei der Erstellung regelkonformer Prozesse befassen.

3.3. Regelkonforme Prozessmodellierung

Ein Überblick über die Forschungsherausforderungen im Bereich des regelkonformen Geschäftsprozessmanagements wird in der Veröffentlichung [SGN07] gezeigt. Das zweite große Themengebiet dieser Veröffentlichung ist ein Ansatz zur Modellierung von Complianceregeln.

Im Hinblick auf die Modellierung von Complianceregeln sehen die Autoren Bedarf für Vereinfachung. Sie sind der Meinung, dass derzeit der Mensch nicht ausreichend dabei unterstützt wird, Mengen von Complianceregeln zu verwalten. Dies sei zum Beispiel der Fall bei der Verwaltung von Complianceregeln, die zur Laufzeit von Geschäftsprozessen eingesetzt werden, um diese automatisch zu untersuchen.

Der Artikel zeigt wie Complianceregeln mit Formal-Contract-Logic (FCL) [GM06] modelliert werden können. FCL bringt Eigenschaften mit, die bei der Modellierung von Complianceregeln hilfreich sind.

Eine dieser Eigenschaften ist die Möglichkeit normative oder normgebende Ausdrücke zu erstellen. Weiterhin zeigt er einen Ansatz wie diese Complianceregeln graphisch mit Prozessmodellen verknüpft werden können.

Der in [BBD+11] beschriebene Ansatz zur regelkonformen Modellierung von Prozessen im Finanzsektor zeigt, wie die Semantic Business Process Modeling Language (SBPML) mit Complianceregeln verknüpft werden kann. Diese Complianceregeln können automatisch überprüft werden. Bei der automatischen Überprüfung wird das Auftreten von durch Complianceregeln beschriebener Muster im Prozessmodell überprüft.

Ein Policy-basiertes Rahmenwerk für die Verwaltung von Complianceregeln und den mit ihnen verknüpften Prozessen wird in [KSMP07] präsentiert. Weiterhin werden in diesem Artikel acht Anforderungen an ein solches Rahmenwerk gestellt. Im Folgenden werden diese acht Anforderungen skizziert:

- **Änderungsmanagement**: Da Complianceregeln eine Umsetzung von Regularien in formale Definitionen darstellen, kann davon ausgegangen werden, dass sie regelmäßig geändert werden müssen. Ebenso soll es die Möglichkeit geben Prozessmodelle anzupassen.

- **Nachvollziehbarkeit und Zuweisbarkeit von Effekten bei der Ausführung eines Prozessen beim Einsatz von Complianceregeln**: Es muss klar sein, welche Effekte die Verknüpfung von Complianceregeln mit einem Prozessmodell zur Ausführungszeit haben.

- **Komplexität der Compliancerегeln**: Das Rahmenwerk soll die Verwaltung von Compliancerегeln vereinfachen. Dazu müssen Compliancerегeln in einer generischen Form vorliegen, um für verschiedene Verwendungszwecke einsetzbar zu sein.
- **Effektivität der eingesetzten Compliancerегeln**: Es muss mittels der eingesetzten Complianceüberprüfungsalgorithmen bestimmt werden können, ob die mit einem Prozessmodell verknüpften Compliancerегeln die erwünschte Wirkung erzielen.
- **Kosten**: Der Einsatz des Rahmenwerks muss die Gesamtkosten der regelkonformen Prozessverwaltung reduzieren.
- **Durchsetzung von Compliancerегeln**: Das Rahmenwerk muss sicherstellen, dass mit Prozessmodellen verknüpfte Compliancerегeln nicht umgangen werden können.
- **Skalierbarkeit**: Compliancerегeln und Prozesse können beliebig komplex werden. Das Rahmenwerk muss mit dieser Komplexität umgehen können.
- **Einflussanalyse von Änderungen**: Compliancerегeln können voneinander abhängig sein. Die Wirkung der Änderung einer Compliancerегel muss vor dem Einsatz überprüft werden können.

Im Artikel [LSG08] werden die Nachteile von Überwachungssystemen aufgezeigt, die anhand von Ereignissen die Regelkonformität dieser Prozesse überwachen.

Systeme können erst *nach* der Verletzung einer Complianceregel reagieren. Weiterhin sind die für die Überprüfung verwendeten Complianceregeln oft hart in die Systeme integriert. Dies führt zu einer schlechten Wartbarkeit und Änderbarkeit. Deshalb ist es notwendig Complianceaspekte schon zur Entwicklungszeit eines Prozesses zu betrachten.

Governatori et al. zeigen in [Gov08] eine Methode für die Prozessverwaltung, die in drei Schritte gegliedert ist. Diese Schritte sind die Anreicherung von Prozessen mit Complianceregeln, die automatische Überprüfung von Complianceregeln und die Rückmeldung von Regelverstößen zur Anpassung von Prozessmodellen.

In diesem Artikel wird die Formal-Contract-Logic (FCL) verwendet, um Prozessmodelle mit Complianceanforderungen anzureichern. Weiterhin zeigt der Artikel Algorithmen, um in FCL geschriebene Complianceregeln zusammen mit einem Prozessmodell automatisch zu überprüfen.

Überprüfungen von Complianceregeln sollen an drei Stellen im Geschäftsprozesslebenszyklus vorgenommen werden [STK+10]. Die erste Stelle ist die Entwicklungsphase. Es müssen hier statische Überprüfungen des entstehenden Prozesses vorgenommen werden. Die zweite Stelle ist die Ausführungszeit eines Prozesses. Die ausgeführten Instanzen von Prozessen müssen hier anhand von bei der Ausführung entstandenen Ereignissen auf Verletzungen von Complianceregeln überprüft werden. Die dritte Stelle ist die Überprüfung nach der Ausführung von Instanzen. Dies bedingt, dass alle während der Ausführung erzeugten Ereignisse in einer Datenbank gespeichert werden.

Im Artikel [LGRMD08] werden Ansätze gezeigt, die Complianceregeln eines Prozesses während des gesamten BPM Lebenszyklus überprüfen. Er zeigt an Prozessverwaltungssysteme gestellte Anforderun-

gen, um die Überprüfung von Complianceregeln zu unterstützen und bewertet bestehende Lösungen auf Grundlage dieser Anforderungen. Die folgende Liste zeigt diese Anforderungen an ein Prozessverwaltungssystem, das den gesamten BPM-Lebenszyklus im Hinblick auf Compliance abdeckt:

- **Eine formale Sprache zur Spezifikation von Complianceregeln.**
- **Verwaltungswerkzeuge für Complianceregeln**: Complianceregeln können sich ändern und werden von der Entwicklungsphase bis zur Audit-Phase im BPM-Lebenszyklus eingesetzt. Dies soll durch entsprechende Werkzeuge unterstützt werden.
- **Unterstützung implementierungsunabhängiger sowie domänenspezifischer Complianceregeln**: Complianceregeln können in unterschiedlichen Anwendungsgebieten eingesetzt werden. Es wäre hinderlich für die Wiederverwendbarkeit, wenn sie in einer für ein bestimmtes Anwendungsgebiet eingesetzten Sprache spezifiziert wären. Aus diesem Grund sollen Complianceregeln auch in domänenübergreifenden Sprachen spezifiziert werden können. Eine Complianceregel, die in einer domänenübergreifenden Sprache spezifiziert ist, kann in eine domänenspezifische Sprache transformiert werden.
- **Überprüfung von Complianceregeln zur Entwicklungszeit und zur Laufzeit.**
- **Überprüfung von Änderungen an laufenden Prozessmodellen**: Änderungen an laufenden Prozessmodellen können zu nicht regelkonformen Prozessinstanzen führen.

- **Überprüfung der Einhaltung von Complianceregeln bei der Änderung eines Prozessmodells**: Änderungen an laufenden Prozessinstanzen und parallele Änderungen an deren Prozessmodellen können zu Inkonsistenzen bei der Einhaltung von Complianceregeln führen. Dies muss durch die automatische Überprüfung jeder Änderung eines Prozessmodells verhindert werden.

- **Unterstützung von prozessübergreifenden Complianceregeln**: Da Prozesse oft aus mehreren Teilprozessen zusammengesetzt sind, ist es notwendig Complianceregeln über Prozessgrenzen hinweg zu definieren und überprüfen zu können.

- **Klar verständliche Rückmeldungen der Überprüfungswerkzeuge**: Es ist für die Benutzbarkeit der Werkzeuge zur automatischen Überprüfung von Prozessen unabdingbar, dass Meldungen der Systeme klar verständlich sind und wenn möglich direkt auf Fehlerquellen hinweisen.

- **Außer-Kraft-Setzen von Complianceregeln**: Manche Complianceregeln sind eher als Vorschläge für einen reibungslosen Ablauf eines Prozesses zu verstehen. Wird das Außer-Kraft-Setzen von Complianceregeln vom System nicht unterstützt, so kann dies zu einer Ablehnung des Systems durch die Benutzer führen.

- **Mechanismen zur Zurückverfolgung**: Änderungen an Prozessmodellen oder das Außer-Kraft-Setzen von Complianceregeln müssen vom System dokumentiert werden. Somit kann nach Prozessende nachvollzogen werden, welche Entscheidungen zur

Laufzeit getroffen wurden. Dies ist zum Beispiel im klinischen Bereich von zentraler Bedeutung.

Aufbauend auf [LGRMD08] zeigt [LRD08] ein Rahmenwerk für die Integration von Wissensbereichen in Prozessverwaltungssysteme. Weiter wird gezeigt, wie semantische Einschränkungen an Prozessmodellen definiert werden können. Der Hauptteil der Veröffentlichung befasst sich mit der Definition eines Maßstabs für semantische Korrektheit von Prozessmodellen und der effizienten Überprüfung von semantischen Einschränkungen. Abschließend wird eine Architektur eines Repositorys für die Verwaltung semantischer Einschränkungen gezeigt.

Ein weiteres Rahmenwerk zur Überprüfung von Complianceregeln wird in [LMX07] vorgestellt. Zur Modellierung von Complianceregeln wird die graphische Sprache BPSL (Business Property Specification Language) verwendet. Es wird argumentiert, dass es mit dieser Sprache im Gegensatz zur Arbeit mit Linearer Temporaler Logik (LTL) für Menschen einfacher sei, Complianceregeln zu erstellen. Zur Definition von Prozessmodellen wird BPEL verwendet. Auch hier wird argumentiert, dass es für Menschen einfacher sei mit BPEL Geschäftsprozesse zu erstellen, als mit π-Kalkül.

Zur Überprüfung von Complianceregeln werden Techniken aus dem Bereich des Modelchecking verwendet. Dazu werden die in BPSL geschriebenen Complianceregeln in LTL-Ausdrücke und die BPEL Prozessmodelle in π-Kalkül-Prozesse übersetzt. Werden Verletzungen von Complianceregeln in einem Prozessmodell gefunden, so können Beispiele für Ausführungspfade im originalen BPEL-Prozessmodell angegeben werden. Ein solches Beispiel ist das Ergebnis, das ein Modelchecker

ausgibt, sobald er eine Regelverletzung gefunden hat.

Im Artikel [DCD+09] unterstreichen Daniel et al. die Notwendigkeit, das Thema Compliance schon ab der Designphase des BPM-Lebenszyklus zu beachten. Zu diesem Zweck wird ein Konzept präsentiert, um Prozessmodelle mit Metadaten anzureichern, welche von einer Ausführungsumgebung ausgelesen werden, um Ereignisse zu bestimmten Zeitpunkten während der Ausführung zu erzeugen. Diese Ereignisse können in einem sogenannten Reporting-Dashboard graphisch dargestellt werden. Menschen können mit diesen Informationen Regelverletzungen, die zur Laufzeit eines Prozesses aufgetreten sind, ausfindig machen. Weiterhin wird vorgeschlagen mit Prozesssichten zu arbeiten, um dem menschlichen Prozessdesigner zu jedem Zeitpunkt die Informationen anzubieten, die er für eine Aufgabe benötigt. Mit Prozesssichten können zum Beispiel Teile von Prozessmodellen ausgeblendet werden, um die Informationsdichte für menschliche Prozessmodellierer zu reduzieren.

Viele Wissenschaftler, die sich mit der regelkonformen Implementierung von Prozessen beschäftigen, sind der Meinung, dass menschliche Prozessmodellierer mehr Unterstützung durch Modellierungswerkzeuge benötigen. Die meisten in diesem Abschnitt gezeigten Konzepte zielen darauf ab, den Umgang mit Complianceregeln und deren Verknüpfung mit Prozessmodellen zu erleichtern. Hierfür werden Konzepte beschrieben, die die Complianceregeln von Hochsprachen in niedrigere, für Modelchecker als Eingabesprachen geeignete Sprachen, übersetzen. Weiterhin wird gezeigt, wie Modelchecker eingesetzt werden können, um Prozessmodelle auf Verletzungen von Complianceregeln zu überprüfen.

Die vorliegende Arbeit präsentiert für einige, der in [KSMP07] auf-

gestellten Anforderungen, Lösungen. Eine Lösung, die das Änderungsmanagement erleichtert, wird durch die Möglichkeit der graphischen Modellierung von LTL-Formeln in Abschnitt 7.7 gezeigt. Die Effektivität der eingesetzten Compliancerегeln kann mit Hilfe der in dieser Arbeit vorgestellten Konzepte zur Überprüfung von Teilen von Prozessmodellen evaluiert werden. Dies ist in Abschnitt 4 beschrieben. Die Skalierbarkeit des in dieser Arbeit vorgestellten Ansatzes ist durch die Implementierung des im Abschnitt 7.3 vorgestellten Compliancewizards sichergestellt. Es können hiermit zum Beispiel beliebig viele Compliancerегeln mit einem Prozess verknüpft werden. Weiterhin sind die in dieser Arbeit verwendeten Modelchecker weit verbreitet und in der Industrie sowie in Großprojekten häufig eingesetzt [JGP99].

Die Verwendung von Compliancefragmenten zur Umsetzung von Complianceanforderungen ist ein Schritt in die Richtung dieser Arbeit. Fragmente werden auch in den hier vorgestellten Konzepten verwendet.

3.4. Automatische Überprüfung von Prozessmodellen anhand von Compliancerегeln zur Entwicklungszeit

Um Prozessmodelle anhand von Compliancerегeln zur Entwicklungszeit zu überprüfen, werden in der Literatur Methoden beschrieben, die auf der Verwendung von Modelcheckern aufbauen [ETHP10, Awa10, WMM09, STK+10].

In [WMM09] und [Wol10] zeigen Wolter et al. wie Zugangskontrolleigenschaften angewendet auf BPMN-Prozesse mit dem SPIN Modelchecker überprüft werden können. Vorhandene Arbeiten [DDO08, RMF07] dienen hier als Basis. In diesen Arbeiten wird gezeigt, wie

BPMN Prozessmodelle in Petri-Netze überführt werden können. Weiterhin wird gezeigt, wie Petrinetze mit der Sprache PROMELA spezifiziert werden können. PROMELA ist die Eingabesprache des SPIN Modelcheckers.

Ein weiterer Ansatz [Awa10] baut auf der Verwendung des NuSMV Modelcheckers auf. Um in einigen Fällen den teuren Aufruf dieses Modelcheckers zu vermeiden, wird das zu untersuchende Prozessmodell als erstes einer syntaktischen Prüfung unterzogen. Da die in diesem Artikel behandelten Complianceregeln den Kontrollfluss eines Prozesses einschränken, wird bei dieser Überprüfung getestet, ob ein Pfad im Prozessmodell existiert, der zur Erfüllung einer Complianceregel durch das Prozessmodell führt. BPMN-Q wird für diese Prüfung verwendet. Ist diese Prüfung positiv verlaufen, das heißt, eine Complianceregel könnte theoretisch von einem Prozessmodell erfüllt werden, so wird im Anschluss NuSMV verwendet, um eine semantische Prüfung durchzuführen. Im anderen Fall muss NuSMV nicht aufgerufen werden, da kein Ausführungspfad im Prozessmodell besteht, der zur Erfüllung der Complianceregel führen kann. Das Low Level Petri net Analyser (LoLa) wird in dieser Arbeit für die Überprüfung des Nichtvorhandenseins von Deadlocks verwendet.

In [STK+10] wird ein Ansatz zur automatischen Überprüfung vorgestellt, der die Transformation eines Prozessmodells nach Reo [Arb04], einer graphischen auf Nachrichtenkanälen aufbauenden Prozesssprache, beschreibt. Prozessmodele, die in Reo vorliegen, können nach einer Bearbeitung durch einen Experten mit dem Modelchecker PRISM [KNP02] überprüft werden.

3.5. Zusammenfassung und Einordnung

Von den in diesem Abschnitt gezeigten Ansätzen ist der in [Wol10] umgesetzte Ansatz der am stärksten wissenschaftlich bearbeitete. Der Ansatz beruht auf der Transformation eines BPMN-Prozessmodells in ein Petrinetz. Dieses Petrinetz wird dann in die Eingabesprache des in dieser Arbeit verwendeten Modelcheckers SPIN übersetzt. Eine Eigenschaft eines Modelcheckers ist die Möglichkeit zur Bereitstellung eines Gegenbeispiels im Falle der Aufdeckung einer Regelverletzung. Ein Gegenbeispiel zeigt den Ausführungspfad in einem Prozessmodell an, das zu dieser Regelverletzung führte. In der vorliegenden Arbeit wird gezeigt, wie die Ausgabe des SPIN Modelcheckers automatisch auf den ursprünglichen BPMN-Prozess abgebildet werden kann.

KAPITEL 4

ENTWICKLUNG VON PROZESSEN MIT REGELKONFORMEM KONTROLLFLUSS

Das Kapitel befasst sich mit den Konzepten, die dazu dienen, einen menschlichen Prozessmodellierer dabei zu unterstützen, regelkonforme Prozesse zu erstellen. Das heißt, dass alle Modifikationen, die an einem Prozess durchgeführt werden, gegen die mit dem Prozess verbundenen Complianceregeln geprüft werden müssen.

Beginnend mit der Erläuterung eines laufenden Prozessbeispiels (siehe Abschnitt 4.1) beschreibt dieses Kapitel zwei der in Kapitel 1.4 vorgestellten wissenschaftlichen Beiträge, Compliancetemplates (Abschnitt 4.2) und Compliancescopes (Abschnitt 4.3), im Detail. In Abschnitt 4.5 wird ein Algorithmus zur Überprüfung von Prozessmo-

dellen auf Verstöße gegen Complianceregeln gezeigt. Die lückenlose Präsentation der einzelnen Schritte des Algorithmus vom graphischen Prozessmodell bis hin zum Modelchecking soll die Umsetzbarkeit der zuvor gezeigten theoretischen Ansätze unterstreichen.

Viele Complianceregeln, die auf Prozesse angewendet werden, beeinflussen den Kontrollfluss eines Prozesses. Der Kontrollfluss eines Prozesses stellt die möglichen Ausführungspfade eines Prozessmodells dar [LR00]. Um bestimmte kontrollflussbasierte Complianceregeln einzuhalten, dürfen manche Pfade dieses Kontrollflusses zur Laufzeit nicht ausgeführt werden. Das heißt, es muss schon zur Entwicklungszeit eines Prozesses untersucht werden, ob eine Modifikation an einem Prozessmodell einen unerlaubten Ausführungspfad in das Prozessmodell einfügt. Um Kosten zu sparen ist es ratsam, die Regelkonformität eines Prozessmodells so früh wie möglich zu gewährleisten [Boe87]. Des Weiteren können Prozessmodelle schnell unübersichtlich werden. Menschliche Prozessmodellierer sind mit der ihnen auferlegten Aufgabe der Entwicklung eines Prozesses ausgelastet. Aus diesem Grund müssen Prozessmodellierer bei der Erstellung von regelkonformen Prozessen unterstützt werden, damit sie sich auf die eigentliche Aufgabe konzentrieren können: der Entwicklung eines Prozesses, der die gestellten Anforderungen erfüllt.

4.1. Beispielszenario: Blutspendeprozess des Roten Kreuz Hong Kong

Abbildung 4.1 zeigt ein Szenario, welches auf einem existierenden Prozess des Roten Kreuz Hong Kong basiert. Beschrieben wurde dieser Prozess in [TLF+10]. Da der Prozess mit einer nicht standardisierten

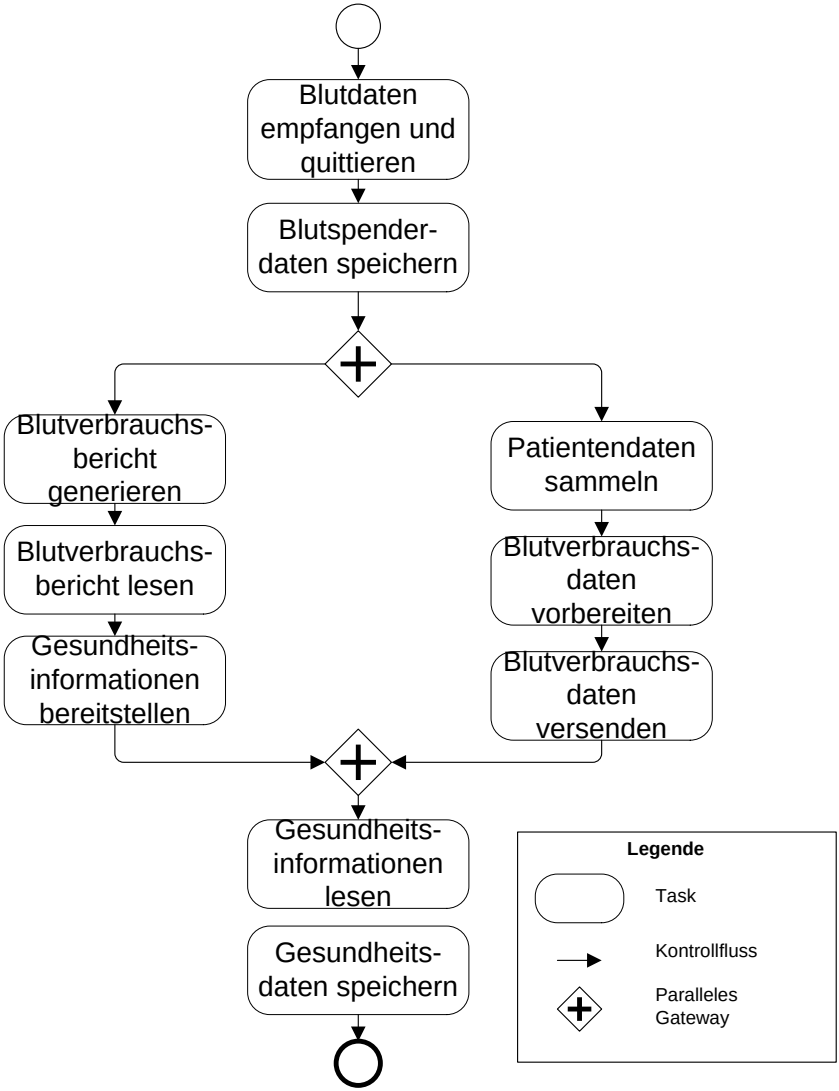

Abbildung 4.1.: Beispielprozess. (Vgl. [SFG+11])

Notation beschrieben wurde, wurde er in Abbildung 4.1 in Business Process Model and Notation (BPMN) übertragen. Dieser Prozess dient als Beispiel zur Erläuterung der Beiträge dieser Dissertation. Um die Komplexität dieses Beispielprozesses in einem Rahmen zu halten, der es ermöglicht, die in diesem Kapitel vorgestellten Konzepte zu erläutern, wurde darauf verzichtet, den Datenfluss im Prozessmodell explizit zu modellieren. Weiterhin wurde darauf verzichtet spezielle Task-Typen, wie zum Beispiel Service-Tasks, zu verwenden. Wissen über die Implementierung der Tasks in diesem Prozessmodell ist für die Erläuterung der neuen Konzepte dieser Arbeit nicht vonnöten. Anhang B zeigt zwei Prozesse, die mit den neuen Konzepten dieser Arbeit versehen wurden. Diese Prozesse beruhen auf Erfahrungen des Autors der vorliegenden Arbeit im Automobilkonzern Daimler in der Funktion als IT Architekt und zeigen somit die Verwendbarkeit der neuen Konzepte dieser Dissertation zur Lösung wirklichkeitsgetreuer Probleme.

Der Prozess in Abbildung 4.1 beschreibt die Schritte einer Blutentnahme über die Lagerung bis zur statistischen Erfassung der Blutproben. So beginnt dieser Prozess mit dem Empfang der bei einer Blutentnahme erhobenen Daten. Diese Daten werden statistisch erfasst und es wird ein Blutverbrauchsbericht erstellt. Anhand dieses Berichts werden der Öffentlichkeit eine über alle Blutproben konsolidierte Gesundheitsinformation mitgeteilt. Parallel dazu werden in den Krankenhäusern des Roten Kreuz Hong Kong Patientendaten gesammelt. Die Gesundheitsinformationen und Blutverbrauchsdaten werden dann zusammen gespeichert. Dies dient dazu, eine spätere Auswertung der gesamten Daten über bestimmte Zeiträume möglich zu machen.

Im Folgenden werden Beispiele für Complianceregeln aufgeführt, die für diesen Prozess denkbar wären.

- Die Aktivität *Blutdaten empfangen und quittieren* und *Blutspenderdaten speichern* müssen immer in dieser Reihenfolge ausgeführt werden. Im Beispielprozess gilt die Annahme, dass ein Gesetz existiert, das die Quittierung elektronisch empfangener Blutdaten als ersten Schritt bei der Datenverarbeitung vorschreibt.

- In diesem Prozess ist ein sogenanntes *Separation of Duties Szenario (Deutsch: Vier-Augen-Prinzip)* vorstellbar. Unter dem Begriff Separation of Duties versteht man die Vorgabe, dass bestimmte Aufgaben von verschiedenen Personen durchgeführt werden müssen. Die Aktivitäten *Patienten-Daten sammeln* und *Blutverbrauchs-Daten vorbereiten* müssen von verschiedenen, mit entsprechenden Fähigkeiten versehenen Personen ausgeführt werden. Diese Aktivitäten müssen weiterhin in der im Beispiel aufgeführten Reihenfolge ausgeführt werden.

- Kontrollflussbasierte Complianceregel: Die Aktivitäten *Blutverbrauchsbericht generieren*, *Blutverbrauchsbericht lesen* und *Gesundheitsinformationen bereitstellen* müssen parallel zu den Aktivitäten *Patientendaten sammeln*, *Blutverbrauchsdaten vorbereiten* und *Blutverbrauchsdaten versenden* ausgeführt werden. Dies lässt schließen, dass der in Abbildung 4.1 dargestellte Kontrollfluss so bestehen muss, damit der gezeigte Prozess syntaktisch korrekt ist und damit er die oben skizzierten Complianceregeln einhält.

Dwyer et al. stellen in [DAC99] wiederkehrende Muster bei der Erstellung von Eigenschaften von Systemen vor. Außerdem zeigt Dwyer

die Häufigkeit mit der bestimmte Muster vorkommen. Das am meisten vorkommende Muster ist das so genannte *Response*-Muster. Dieses Muster beschreibt einen Ausführungspfad in einem System in dem nach Eintreten eines Ereignisses ein bestimmtes weiteres Ereignis in einem unbestimmten Zeitabstand eintreten muss. In Beispielprozess könnte dieses weitere Ereignis die Quittierung der empfangenen Blutdaten sein.

Turetken et al. verwenden diese Muster in [TEHP11]. Hier werden wiederkehrende Muster für Complianceregeln aufgezeigt, die auf Prozesse Anwendung finden. Auch die oben aufgeführten Complianceregeln für das laufende Beispiel dieser Arbeit wurden von Dwyer et al. abstrakt beschrieben. Tabelle 4.1 zeigt einige Beispiele für wiederkehrende Muster von Dwyer et al. [DAC98] sowie Beispiele aus weiterführender Literatur [TEHP11].

Diese Complianceregeln sind mit Hilfe von LTL [Pnu77] geschrieben. In der formalen Definition dieser Regeln sind die Variablen A und B mit den Namen von Aktivitäten in einem Prozess gleichzusetzen. Zur Beschreibung der Beispiele wird die Abbildung von Variablennamen von Complianceregeln auf Namen von Aktivitäten in Prozessen verwendet. Der Ansatz kann jedoch mit beliebigen Abbildungen von Variablen in Complianceregeln auf Konstrukte in Prozessen verwendet werden. Zum Beispiel könnten Variablen in Complianceregeln auf Endpunkte von Services abgebildet werden, die von Aktivitäten in einem Prozess aufgerufen werden. Somit ist die Complianceregel unabhängig vom tatsächlichen Namen einer Aktivität. Damit können zum Beispiel die Services eingeschränkt werden, die in einem bestimmten Prozess aufgerufen werden können.

In den folgenden Kapiteln werden zwei Ansätze für die Entwicklung

Tabelle 4.1.: Liste von kontrollflussbasierten Complianceregeln (einige basierend auf [DAC98]). Die Funktionsweise der in diesen Ausdrücken verwendeten Operatoren wird in Abschnitt 2.5 beschrieben.

Regelbeschreibung	**Definition in LTL**
Aktivität A muss vor dem Prozessende ausgeführt werden.	$\Diamond$ A
Aktivität A soll nie ausgeführt werden.	$\neg\Diamond A$
Aktivität A muss sich immer in der Ausführung befinden. Das heißt, der Prozess muss aus mindestens zwei parallelen Zweigen bestehen. In einem der beiden Zweige muss immer A gelten.	$\Box$ A
Ausführungsreihenfolge: Nachdem A ausgeführt wurde, wird vor dem Prozessende B ausgeführt.	$\Box(A \Rightarrow \Diamond B)$
Ausführungsreihenfolge: Nachdem A und danach B ausgeführt wurde wird vor dem Prozessende C ausgeführt.	$\Diamond C \Rightarrow (\neg C\ U(A \wedge \neg C \wedge \bigcirc(\neg C\ U\ B)))$
Gemeinsames Auftreten von Aktivitäten.	$\Diamond A \wedge \Diamond B$
Ausführung von A hat Vorrang vor der Ausführung von B.	$\neg B\ W\ A$
Entweder A oder B sollen ausgeführt werden.	$\Diamond(A\ \vee\ B)$
Entweder wird A oder B ausgeführt aber nicht beide zusammen oder keine von beiden.	$\Diamond(A \vee B) \vee (\Diamond\neg A \wedge \Diamond\neg B)$
Zuerst wird A und danach vor dem Prozessende B ausgeführt. Dies führt zur Ausführung von C.	$\Box(A\ \wedge\ \bigcirc\Diamond B\ \Rightarrow\ \bigcirc(\Diamond(B\ \wedge\ \Diamond C)))$

von regelkonformen Prozessen gezeigt. Der in Abschnitt 4.2 vorgestellte Ansatz kann bei der Neuentwicklung von Prozessen eingesetzt werden. Der in Abschnitt 4.3 vorgestellte Ansatz wird dazu verwendet, existierende Prozesse mit Complianceregeln zu versehen. Diese Complianceregeln können von graphischen Entwicklungswerkzeugen bei Modifikationen an Prozessmodellen dazu verwendet werden, Regelkonformität sicher zu stellen.

4.2. Vorlagenbasierte Entwicklung regelkonformer Prozesse

Das in diesem Abschnitt vorgestellte Konzept eines *Compliancetemplates* wird als Grundlage für die Entwicklung eines neuen Prozesses verwendet. Dieses Kapitel stellt zunächst die drei Bestandteile von Compliancetemplates, das abstrakte Prozessmodell (Abschnitt 4.2.1), den Variabilitätsdeskriptor (Abschnitt 4.2.2) und den Compliancedeskriptor (Abschnitt 4.2.3), vor. Danach wird erläutert, wie mit dem Compliancetemplate ein neuer Prozess erstellt werden kann (Abschnitt 4.2.4).

Ein Beispielszenario für die Verwendung von Compliancetemplates könnte die Notwendigkeit der Neuentwicklung des in Kapitel 4.1 vorgestellten Beispielprozesses sein. Nachdem der Prozess im Beispielszenario mehrfach ausgeführt wurde, könnten zum Beispiel Schwachstellen auffällig geworden sein. Der benötigte neue Prozess wird auf Grundlage eines Compliancetemplates erstellt. Compliancetemplates werden von Experten in den Gebieten der Prozessentwicklung und Compliance erstellt. Nach der Erstellung werden sie zur Vervollständigung an die eigentlichen Prozessentwickler weitergegeben.

Compliancetemplates [SALM09] sind *unvollständig* spezifizierte Pro-

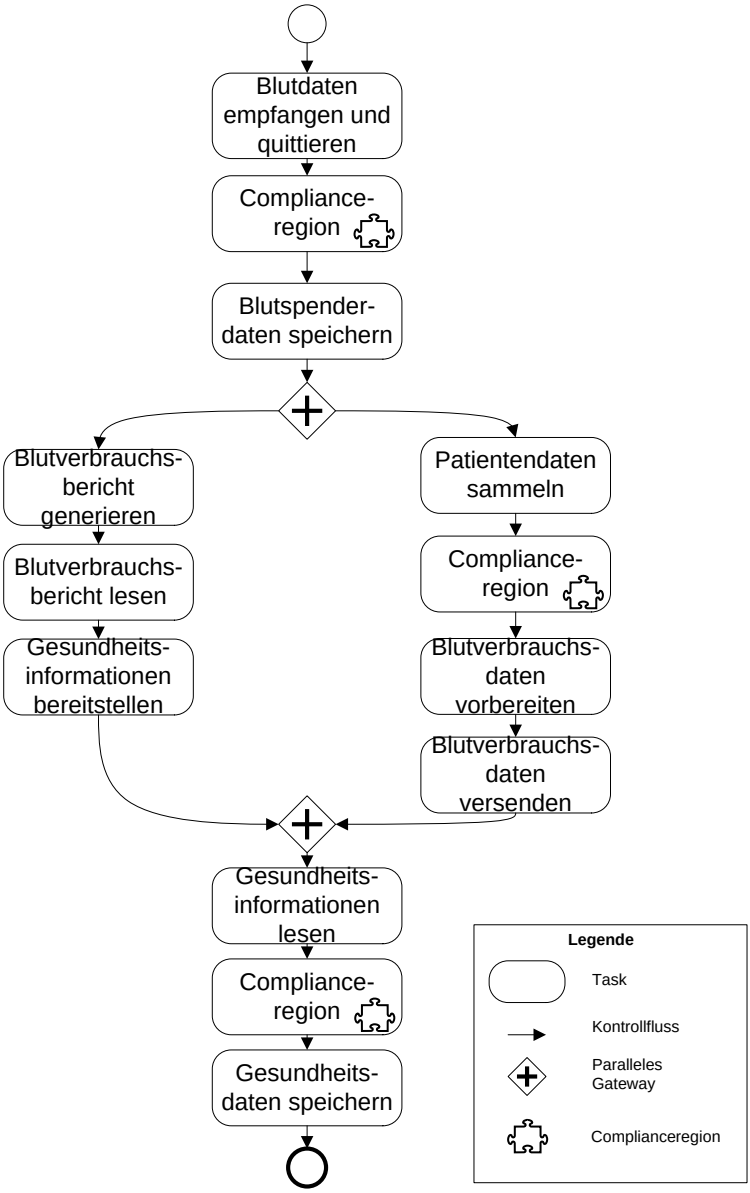

Abbildung 4.2.: Abstraktes Prozessmodell eines Compliancetemplates

zessmodelle. Sie dienen als Vorlage für die Neuentwicklung von Prozessen. Das abstrakte Prozessmodell eines Compliancetemplates implementiert bestimmte Complianceregeln, die für den neuen Prozess gelten müssen. Das heißt, es legt die spätere Struktur eines Prozesses in Grundzügen fest. Der Variabilitätsdeskriptor stellt die Mengen an Aktivitäten bereit, die allgemein zur Füllung eines abstrakten Prozessmodells zur Verfügung stehen. Der Compliancedeskriptor enthält Complianceregeln. Diese werden auf die Complianceregionen eines abstrakten Prozessmodells angewendet und schränken somit die Menge der Aktivitäten ein, die in diese Complianceregionen eingefügt werden können. Mit den Complianceregeln kann somit auf die Ausführungsreihenfolge der Aktivitäten in einem Prozess Einfluss genommen werden.

In einem Unternehmen kann es mehrere Arten von Compliancetemplates geben, die die Grundlage für verschiedene Prozesse bilden. Damit die implizit in einem Compliancetemplate enthaltenen Complianceregeln nicht durch Modifikationen am abstrakten Prozessmodell verändert werden können, kann ein Compliancetemplate nur an speziellen dafür vorgesehenen Punkten geändert und somit vervollständigt werden. In den folgenden Abschnitten werden die drei Bestandteile eines Compliancetemplates vorgestellt.

4.2.1. Das abstrakte Prozessmodell eines Compliancetemplates

Dieser Abschnitt stellt die Erweiterung der BPMN 1.0 Spezifikation um eine Complianceregion dar. Mit dieser Erweiterung wird das abstrakte Prozessmodell eines Compliancetemplates formal definiert.

Die vorliegende Dissertation verwendet eine in Abbildung 4.10 ge-

zeigte Grundmenge von BPMN 1.0 Elementen mit der Annahme der Ausführungssemantik von BPMN 2.0. Bevor das abstrakte Prozessmodell eines Compliancetemplates definiert werden kann, muss definiert werden, aus welchen Teilen ein BPMN Prozess aufgebaut ist. Siehe hierzu Definition 1.

Das in Abbildung 4.2 gezeigte Prozessmodell ist eine Erweiterung des in Abbildung 4.1 gezeigten Blutspendeprozesses. Der ursprüngliche Blutspendeprozess wurde mit *Complianceregionen* versehen. Ein abstraktes Prozessmodell ist nicht vollständig spezifiziert. Die darin enthaltenen Complianceregionen müssen mit einzelnen Aktivitäten oder Prozessfragmenten [EUL09] gefüllt werden, um einen vollständigen Prozess zu bekommen.

BPMN wurde in der vorliegenden Arbeit erweitert und Complianceregionen hinzugefügt. Für die Erweiterung wurde der Erweiterungsmechanismus von BPMN 2.0 verwendet. Eine Kurzbeschreibung von BPMN findet sich in Abschnitt 2.3.

Abbildung 4.3 zeigt eine vereinfachte Darstellung des in der BPMN 2.0 Spezifikation vorgestellten Erweiterungsmechanismusses. Alle BPMN 2.0 Elemente wie Tasks oder Gateways erben von der in Abbildung 4.3 dargestellten Klasse *BaseElement*. Alle Klassen, die von der Klasse BaseElement erben, können erweitert werden.

Eine BPMN-Erweiterung wird durch die Implementierung einer Klasse, die vom Typ *ExtensionDefinition* erbt, erstellt. ExtensionDefinition-Klassen können unabhängig von einem BPMN-Modell erstellt werden. Um ein BPMN-Modell zu erweitern werden sie mit der Klasse *Extension* komponiert, welche wiederum mit der Klasse *Definitions* komponiert ist. Die Klasse Definitions erbt von der Klasse *BaseElement*. Dies macht die Klasse Definitions und die mit ihr komponierten Klassen zu einem

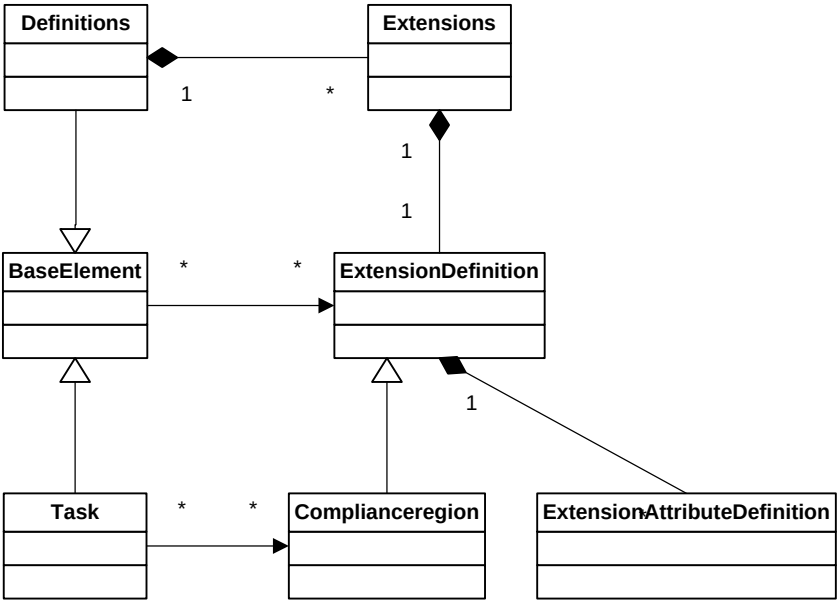

Abbildung 4.3.: BPMN 2.0-Erweiterungsmechanismus skizziert in UML (Vgl. [Obj11])

vollwertigen BPMN-Element, das als Erweiterung eines BPMN-Modells dienen kann.

BPMN-Elemente können wie folgt erweitert werden. Erstellung einer Klasse, die vom Typ ExtensionDefinition erbt. Assoziation dieser Klasse mit einer Klasse, die von der Klasse BaseElement erbt. Der Zweck der Klasse *ExtensionAttributeDefinition* ist die Erweiterung vorhandener BPMN 2.0-Elemente mit neuen Attributen.

Der oben vorgestellte Erweiterungsmechanismus von BPMN 2.0 wurde verwendet, um Complianceregionen in BPMN 1.0 einzuführen.

Dies ist in Abbildung 4.3 durch die Erstellung der Klasse *Complianceregion* und deren Assoziation mit der Klasse *Task* geschehen. Eine Complianceregion erweitert somit einen Task. Complianceregionen haben dieselben Eigenschaften wie BPMN 1.0-Tasks. Das heißt zum Beispiel, dass mehrere Kontrollflusskonnektoren auf sie zeigen können. Und es können auch mehrere Kontrollflusskonnektoren von ihnen weg führen.

Mit diesen Erweiterungen ist es möglich, ein abstraktes Prozessmodell eines Compliancetemplates in einem graphischen Entwicklungswerkzeug zu einem syntaktisch korrekten BPMN-Prozess zu vervollständigen. Bei dieser Vervollständigung muss vom graphischen Entwicklungswerkzeug erzwungen werden, dass nur die im abstrakten Prozessmodell enthaltenen Complianceregionen verändert werden können. Es dürfen keine Modifikationen an anderen Teilen des abstrakten Prozessmodells vorgenommen werden, da sonst die darin implizit enthaltenen Complianceregeln verletzt werden könnten. Es ist jedoch von Vorteil bei der Befüllung einer Complianceregion aus einer Menge möglicher Prozessfragmente auswählen zu können. So können Prozessfragmente wiederverwendet werden, die schon einmal in anderen Prozessen eingesetzt wurden.

Definition 1 (BPMN Prozess [ODHA06])**.** *Ein BPMN Prozess ist ein Tupel* $P = (\mathcal{K}, \mathcal{E}, \mathcal{G}, \mathcal{F})$ *mit:*

- $\mathcal{K}$ *als der Menge der Knoten, die in die disjunkten Mengen der Aktivitäten* $\mathcal{A}$ *(zum Beispiel Tasks), Ereignisse* $\mathcal{E}$ *und Gateways* $\mathcal{G}$ *aufgeteilt werden können.*

- $\mathcal{E}$ *als der Menge der Ereignisse, die in die disjunkten Mengen der Start-Ereignisse* $\mathcal{E}^S$*, Intermediate-Ereignisse* $\mathcal{E}^I$ *und End-Ereignisse*

$\mathcal{E}^E$ *aufgeteilt werden können. Intermediate-Ereignisse könnten in die disjunkten Mengen der Intermediate-Message-Ereignisse* $\mathcal{E}^I_M$ *und Intermediate-Timer-Ereignisse* $\mathcal{E}^I_T$ *aufgeteilt werden.*

- $\mathcal{G}$ *als der Menge der Gateways, die in die disjunkten Mengen der parallelen Gateways* $\mathcal{G}^P$*, der daten-basierten exklusiven Gateways* $\mathcal{G}^D$*, der ereignis-basierten exklusiven Gateways* $\mathcal{G}^E$*, der inklusiven Gateways* $\mathcal{G}^I$ *und der komplexen Gateways* $\mathcal{G}^C$ *aufgeteilt werden kann.*

- *Der Kontrollflussrelation* $F \subseteq K \times K$*, die die Menge der Kontrollflusskonnektoren beschreibt. Die Relation F beschreibt einen gerichteten Graphen.*

Im Folgenden wird definiert, wann ein BPMN Prozess frei von Syntaxfehlern ist. Mit $x \in \mathcal{K}$ berechnet die Funktion $in(x)$ den Eingangsgrad eines Knotens x, das heißt die Anzahl der auf den Knoten zeigenden Kontrollflusskonnektoren. Die Funktion $out(x)$ berechnet entsprechend den Ausgangsgrad eines Knotens x. Um einen gültigen BPMN Prozess zu erstellen, müssen bestimmte Syntax-Regeln eingehalten werden.

Definition 2 (Gültiger BPMN Prozess [ODHA06])**.** *Ein BPMN Prozess ist gültig, wenn die folgenden Syntax-Regeln eingehalten werden:*

- $\forall e \in \mathcal{E}^S \mid in(e) = 0 \wedge out(e) \geq 1$*. Alle Startereignisse haben den Eingangsgrad 0 und einen Ausgangsgrad größer oder gleich 1.*

- $\forall e \in \mathcal{E}^E \mid out(e) = 0 \wedge in(e) \geq 1$*. Alle Endereignisse haben den Ausgangsgrad 0 und einen Eingangsgrad größer oder gleich 1.*

- $\forall e \in \mathcal{E}^I \mid in(e) = out(e) = 1$. *Alle Intermediate-Ereignisse haben einen Eingangsgrad von 1 und einen Ausgangsgrad von 1.*
- $\forall g \in \mathcal{G}^F \wedge \forall g \in \mathcal{G}^D \mid in(e) = 1 \wedge out(e) \geq 1$. *Parallel-Fork-Gateways und Event-Based-XOR-Decision-Gateways haben einen Eingangsgrad von 1 und einen Ausgangsgrad großer oder gleich 1.*
- $\forall g \in \mathcal{G}^J \wedge \forall g \in \mathcal{G}^M \mid in(e) \geq 1 \wedge out(e) = 1$. *Parallel-Join-Gateways und XOR-Merge–Gateways haben einen Eingangsgrad größer 1 und einen Ausgangsgrad von 1.*

Definition 3 ($BPMN^+$ (Mit Complianceregionen erweitertes BPMN))**.** *Sei $\mathcal{C}$ die Menge der Complianceregionen und $\mathcal{K}$ die Menge der Knoten in einem Prozessmodell. Sei weiterhin $\mathcal{K}^+ = \mathcal{K} \cup \mathcal{C}$. Dann gilt:*
$BPMN^+ = \{\mathcal{K}^+, \mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{F}\}$.

Ein abstraktes Prozessmodell eines Compliancetemplates besteht aus einem BPMN Prozessmodell, welches mit mindestens einer Complianceregion versehen wurde. Dies wird in Definition 4 beschrieben.

Definition 4 (Abstraktes Prozessmodell $\mathcal{A}$ eines Compliancetemplates)**.** *Sei ein Prozessmodell $P \in BPMN^+$. Das bedeutet, P ist ein Tupel $(\mathcal{K}_P^+, \mathcal{A}_P, \mathcal{E}_P, \mathcal{G}_P, \mathcal{F}_P)$. Dann gilt: $P \in \mathcal{A} \Leftrightarrow \exists k \in \mathcal{K}_P^+ : type(k) = C$*

4.2.2. Der Variabilitätsdeskriptor eines Compliancetemplates

Das Konzept eines Variabilitätsdeskriptors wurde von Mietzner et al. [MLP08, ML08, Mie08] entwickelt. Es wurde allgemein definiert, um Software-Artefakte jeglicher Art mit Variabilitäten zu versehen. In dieser Arbeit werden Variabilitätsdeskriptoren dazu verwendet, um

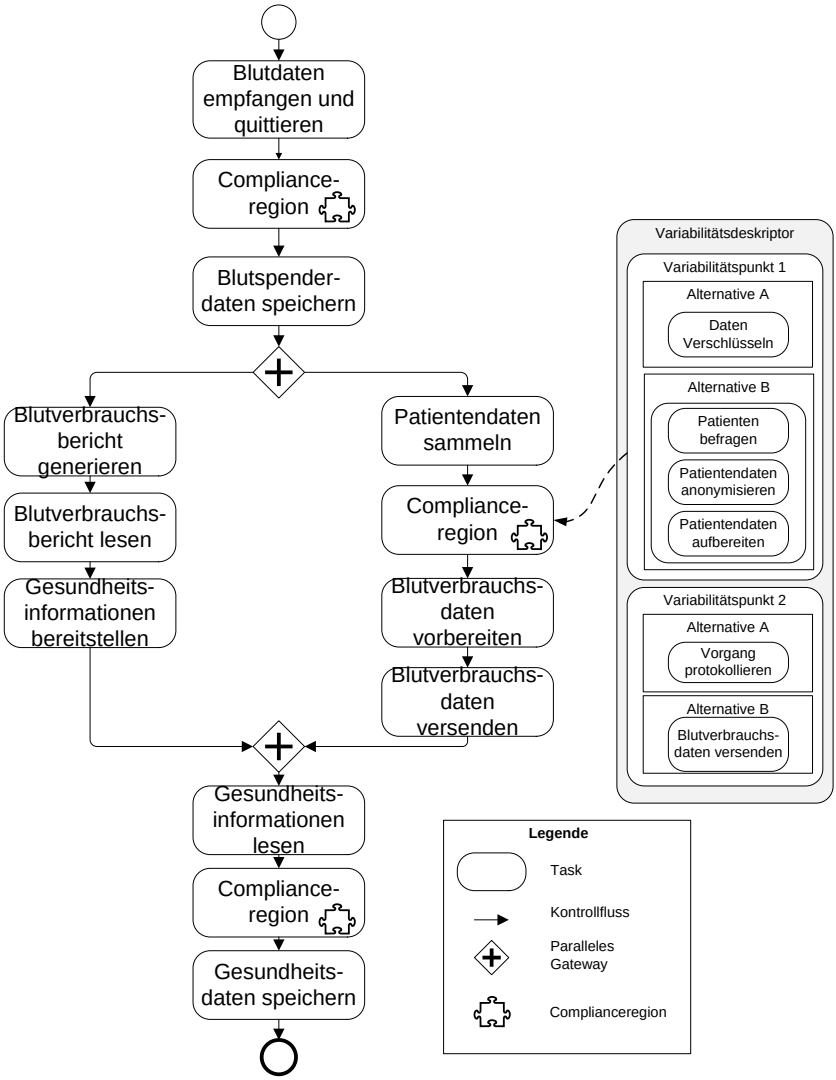

Abbildung 4.4.: Abstraktes Prozessmodell eines Compliancetemplates in Verbund mit Variabilitätsdeskriptor

anzuzeigen, welche Mengen von Aktivitäten zur Füllung von Complianceregionen verwendet werden können. Abbildung 4.4 zeigt den abstrakten Prozess eines Compliancetemplates zusammen mit einem Variabilitätsdeskriptor. Variabilitätsdeskriptoren sind aus Variabilitätspunkten aufgebaut, die Alternativen enthalten. Alternativen können als Mengen von Aktivitäten angesehen werden, die zusammen in eine Complianceregion eingefügt werden können. Der in Abbildung 4.4 gezeigte Variabilitätspunkt 1 enthält zum Beispiel die Alternativen A und B. Ein weiterer Bestandteil von Variabilitätspunkten sind Lokatoren. Ein Lokator ist in Abbildung 4.4 als gestrichelter Pfeil dargestellt. Er beginnt bei einem Variabilitätspunkt und endet bei einer Complianceregion. Variabilitätspunkte können voneinander abhängig sein. Dies bedeutet für den Fall der Abhängigkeit zweier Variabilitätspunkte A und B: wenn A in einem Prozessmodell verwendet wurde, muss der von diesem Variabilitätspunkt abhängige Variabilitätspunkt B auch verwendet werden. Weiterhin können *aktivierende Bedingungen* mit jeder Abhängigkeit definiert werden. Diese Bedingungen werden ausgewertet, wenn eine bestimmte Abhängigkeit ausgewertet werden soll. Mit aktivierenden Bedingungen ist es im obigen Beispiel möglich, nach der Verwendung einer Alternative in A eine Alternative in B nicht mehr verfügbar zu machen.

Zeigen ein oder mehrere Lokatoren auf eine Complianceregion, so dürfen nur die Mengen an Aktivitäten in die betreffende Complianceregion eingefügt werden, die in dem Variabilitätspunkt enthalten sind, in dem auch der betreffende Lokator definiert ist. Zeigt kein Lokator auf eine Complianceregion, so dürfen alle Variabilitätspunkte in dem mit dem Compliancetemplate verbundenen Variabilitätsdeskriptor zur Befüllung verwendet werden.

Die formale Definition eines Variabilitätsdeskriptors zeigt [Mie10].

4.2.3. Der Compliancedeskriptor eines Compliancetemplates

Abbildung 4.5 zeigt das gesamte Compliancetemplate mit allen drei Komponenten. Vergleicht man Abbildung 4.4 mit Abbildung 4.5, so ist der Compliancedeskriptor auf der linken Seite hinzugekommen. Das Konzept des Compliancedeskriptors ist an das Konzept des Variabilitätsdeskriptors angelehnt. Das Gegenstück zu Variabilitätspunkten bei den Variabilitätsdeskriptoren sind Compliancepunkte bei Compliancedeskriptoren. Auch hier können, genau wie bei den Variabilitätsdeskriptoren, Abhängigkeiten zwischen den einzelnen Compliancepunkten eines Compliancedeskriptors definiert werden. Für die Definition dieser Abhängigkeiten wurde dasselbe Modell wie bei den Variabilitätsdeskriptoren gewählt. Dieses Modell ist in [Mie08] beschrieben.

Ein Metamodell, das die Zusammenhänge der Komponenten des Compliancedeskriptors zeigt, ist in Abbildung 4.6 in UML dargestellt. Ein Compliancedeskriptor enthält demnach eine beliebige Zahl von Abhängigkeiten und eine beliebige Zahl von Compliancepunkten. Das Konzept einer Abhängigkeit ist an die Definition eines Variabilitätspunkts in [Mie10] angelehnt. Abhängigkeiten enthalten genau eine Abhängigkeitsquelle und ein Abhängigkeitsziel. Compliancepunkte enthalten Complianceregeln.

Complianceregeln besitzen eine formale Definition. Diese kann von graphischen Entwicklungswerkzeugen dazu verwendet werden, Modifikationen am abstrakten Prozess automatisch zu untersuchen. Verletzt eine Modifikation eine solche formale Definition einer Complianceregel, kann das graphische Entwicklungswerkzeug den menschlichen

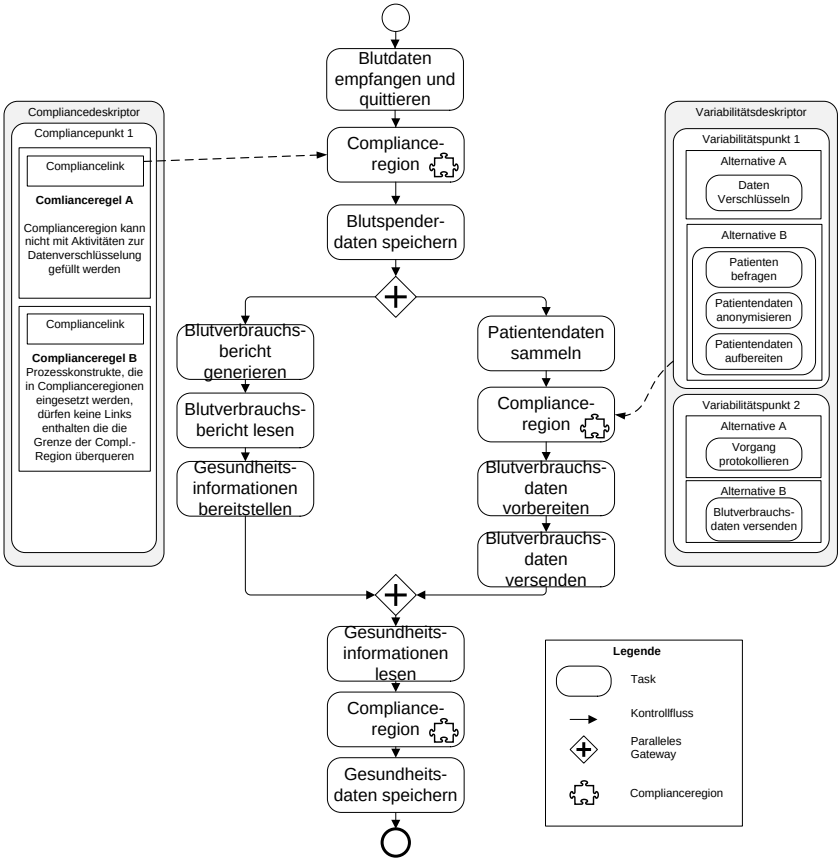

Abbildung 4.5.: Abstraktes Prozessmodell eines Compliancetemplates in Verbund mit Variabilitätsdeskriptor und Compliancedeskriptor

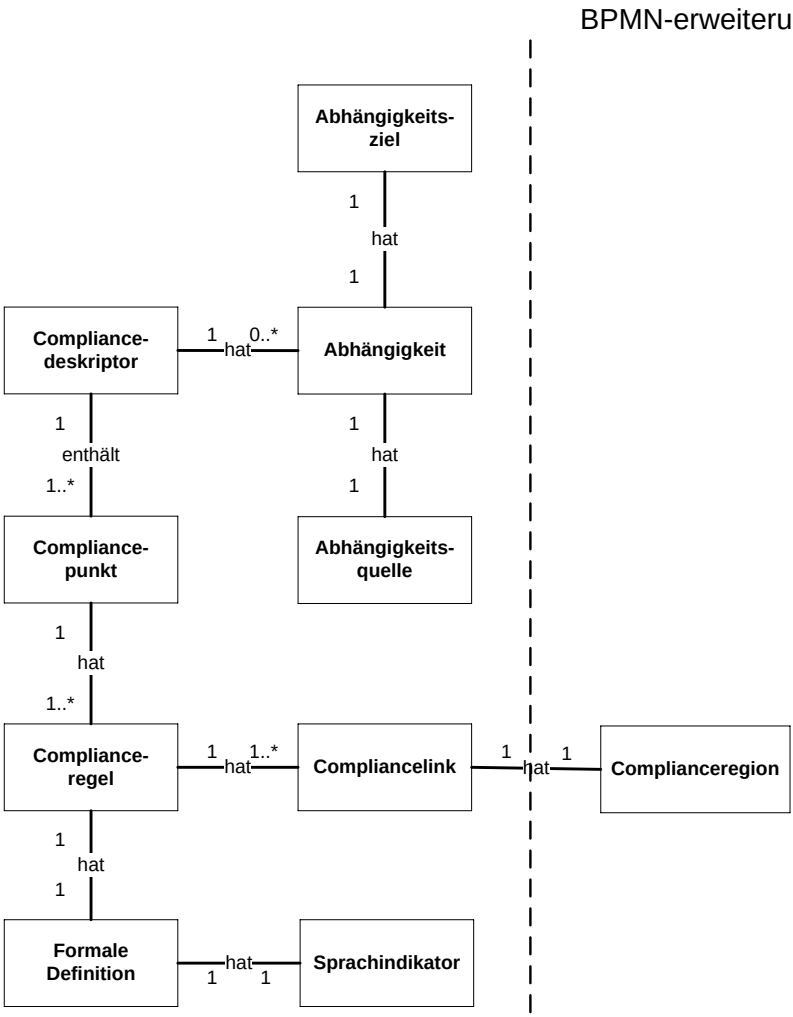

Abbildung 4.6.: Metamodell eines Compliancedeskriptors

Prozessmodellierer darauf aufmerksam machen. Eine formale Definition enthält ihrerseits einen Sprachindikator. Dieser dient dazu, den Werkzeugen, die mit der formalen Definition einer Complianceregel arbeiten, anzuzeigen, in welcher Sprache diese geschrieben ist. Mögliche Werte, die ein Sprachindikator annehmen kann, sind *LTL* oder *XPath*.

Eine Complianceregel enthält weiterhin einen oder mehrere Compliancelinks, welche auf Complianceregionen zeigen. Mit diesen Compliancelinks wird festgelegt, auf welche Complianceregionen die betreffende Complianceregel angewendet werden soll.

4.2.4. Vervollständigen von Compliancetemplates

In diesem Abschnitt werden die Ergebnisse der Arbeiten [WKK+11, KWS11] verwendet, die zum besseren Verständnis zusammengefasst werden.

Wie oben erwähnt müssen Compliancetemplates vervollständigt werden, um aus ihnen einen syntaktisch korrekten Prozess zu machen. Diese Vervollständigung erfolgt in zwei Schritten. Zuerst werden die Complianceregionen bestimmt, die zu einem bestimmten Zeitpunkt im Vervollständigungsprozess befüllt werden können. Die Entscheidung, welche Complianceregionen dies sind, wird anhand von Abhängigkeiten zwischen den Complianceregionen automatisch getroffen. Diese Abhängigkeiten sind durch die Abhängigkeiten zwischen den Alternativen in Variabilitätsdeskriptoren definiert. Betrachtet wird der Fall der Abhängigkeit der Alternative B von der Alternative A in einem Variabilitätsdeskriptor. Alternative A ist mittels eines Compliancelinks mit einer Complianceregion X verbunden. Alternative B ist auf dieselbe Weise

mit einer Complianceregion Y verbunden. In diesem Fall muss zunächst Complianceregion X befüllt werden. Danach kann Complianceregion Y befüllt werden. Für die Auflistung der für eine Complianceregion in Frage kommenden Aktivitäten werden Abhängigkeiten zwischen Variabilitätspunkten mit einbezogen. Eine vollständige Beschreibung des in der vorliegenden Arbeit verwendeten Mechanismus zur Auflösung von Abhängigkeiten und des Berechnens von Complianceregionen, die befüllt werden dürfen, ist in [WKK+11, KWS11] nachzulesen. Diese Veröffentlichungen bauen auf dem Konzept eines Compliancetemplates auf.

Mit dem Ergebnis kann in einem zweiten Schritt vom Prozessmodellierer eine der verbleibenden Alternativen für die Befüllung einer Complianceregion ausgewählt und eingesetzt werden. Bei der Vervollständigung dürfen nur Complianceregionen mit Aktivitäten befüllt werden. Neben der Möglichkeit, Complianceregionen mit einer oder mehrerer Alternativen aus dem Variabilitätsdeskriptor zu befüllen, ist es einem Prozessmodellierer auch erlaubt, die eingefügten Alternativen zu verändern. Es dürfen keine anderen Modifikationen am Compliancetemplate vorgenommen werden. Insbesondere dürfen keine Kontrollflusskonnektoren geändert werden, die sich außerhalb der Complianceregionen befinden. Mit dem Einsetzen von Aktivitäten in Complianceregionen können jedoch Complianceregeln verletzt werden, die im Compliancedeskriptor definiert sind, welcher in jedem Compliancetemplate vorhanden ist. Für die Untersuchung, ob eine Aktivität, die in eine bestimmte Complianceregion eingefügt wurde, eine oder mehrere mit dieser Complianceregion verknüpfte Complianceregeln verletzt, werden Konzepte und Algorithmen verwendet, die in Kapitel 4.3 beschrieben werden.

4.3. Compliancescope

Compliancetemplates werden verwendet, um regelkonforme Prozesse zu erstellen, die von einem bestimmten Punkt aus neu entwickelt werden müssen. Dieser Punkt ist das Compliancetemplate. Compliancetemplates werden nach dem Anwendungsbereich ausgewählt, in welchem der zu entwickelnde Prozess eingesetzt werden soll.

Im Gegensatz dazu haben Firmen eine Fülle existierender Prozesse, die, wie andere Software auch, gewartet und an neue Gegebenheiten angepasst werden müssen. Auch diese Prozesse unterliegen denselben Complianceanforderungen wie neu zu erstellende Prozesse. Änderungen an einem Prozessmodell, die bei einer Wartung durchgeführt werden, dürfen ein regelkonformes Prozessmodell nicht in ein nicht regelkonformes Prozessmodell überführen. Im Folgenden wird ein Konzept vorgestellt, das dazu dient, Bereiche in einem Prozessmodell zu markieren, für die bestimmte Complianceregeln gelten. Dieses Konzept heißt *Compliancescope* [SWLS10].

Compliancescopes und Complianceregionen haben die Gemeinsamkeit, dass die sich in ihnen befindlichen Aktivitäten bestimmte Complianceregeln einhalten müssen, die mit diesen beiden Konstrukten verknüpft sind. Weiterhin markieren beide Konzepte Bereiche in einem Prozessmodell, um bestimmte mit ihnen verknüpfte Complianceregeln auf diese Bereiche anzuwenden. Eine weitere Gemeinsamkeit ist, dass Compliancedeskriptoren bei beiden Konzepten für die Definition von Complianceregeln verwendet werden. Jedoch ist das Konzept eines Compliancescopes weiter gefasst als das einer Complianceregion. Ein Compliancescope kann wie schon oben erwähnt nachträglich auf einen bereits existierenden Prozess angewendet werden. Compliance-

scopes können sich überlappen und auch das gesamte Prozessmodell umspannen, um global gültige Complianceregeln auf Prozessmodelle anzuwenden.

Compliancescopes können auf zwei Arten in Prozesse eingefügt werden:

1. Complianceregionen werden zu Compliancescopes transformiert, wenn bei der Vervollständigung eines Prozessmodells Aktivitäten in sie eingefügt werden.

2. Compliancescopes werden von Experten in existierende Prozesse eingefügt. Diese Prozesse werden damit automatisch auf Verstöße gegen Complianceregeln zum Beispiel bei Änderungen am Prozess überprüfbar gemacht.

Genau wie bei Compliancetemplates können die mit Compliancescopes verknüpften Complianceregeln von graphischen Entwicklungswerkzeugen dazu verwendet werden, um den Prozessmodellierer auf die Verletzung einer Complianceregel aufmerksam zu machen.

4.3.1. Definition Compliancescope aufbauend auf der Definition eines Hypergraphen

Ein Compliancescope stellt eine Hyperkante in einem Hypergraphen dar. Hypergraphen sind Graphen, die Kanten enthalten, die nicht wie gewöhnliche Kanten nur zwei Knoten des Graphen miteinander verbinden, sondern beliebig viele. Solche Kanten nennt man Hyperkanten. Hyperkanten werden in einem Graphen als Kreise gezeichnet, in denen eine oder mehrere Knoten des Graphen enthalten sind. Es folgt

die Definition eines Hypergraphen, auf welcher die Definition eines Compliancescopes aufgebaut ist.

Definition 5 (Hypergraph)**.** *Ein Hypergraph G besteht aus der Menge von Knoten N und der Menge von Hyperkanten H zwischen den Knoten. Jede Hyperkante ist eine Menge von Knoten:* $H \subseteq \{2^N \setminus \emptyset\}$*. Hyperkanten sind ungerichtet [Ber89].*

Auf dieser Grundlage kann die Definition eines Compliancescopes erfolgen:

Definition 6. *Ein Compliancescope ist eine Hyperkante* $h \in H$ *in einem Hypergraphen G, der mit einem Compliancedeskriptor verknüpft ist. Ein Compliancescope übernimmt die Eigenschaften einer Hyperkante.*

Überträgt man das Konzept der Hypergraphen auf ein BPMN 1.0 Prozessmodell und bildet die in der Definition von BPMN (Definition 1) beschriebenen Knoten K auf die Knoten N in einem Hypergraphen ab, dann stellen die Hyperkanten H die Compliancescopes in diesem Prozessmodell dar.

4.3.2. Erweiterung von BPMN 1.0 mit Compliancescopes

Der in Kapitel 4.2.1 beschriebene Erweiterungsmechanismus der Business Process Model and Notation 2.0 (BPMN 2.0) wird auch verwendet, um BPMN mit Compliancescopes zu versehen.

Abbildung 4.7 zeigt ein mit einem Compliancescope versehenes Prozessmodell. Aus Gründen der Übersichtlichkeit wurde das Prozessmodell mit nur einem Compliancescope versehen. In dieser Abbildung sieht man auch einen Compliancedeskriptor, der diejenigen

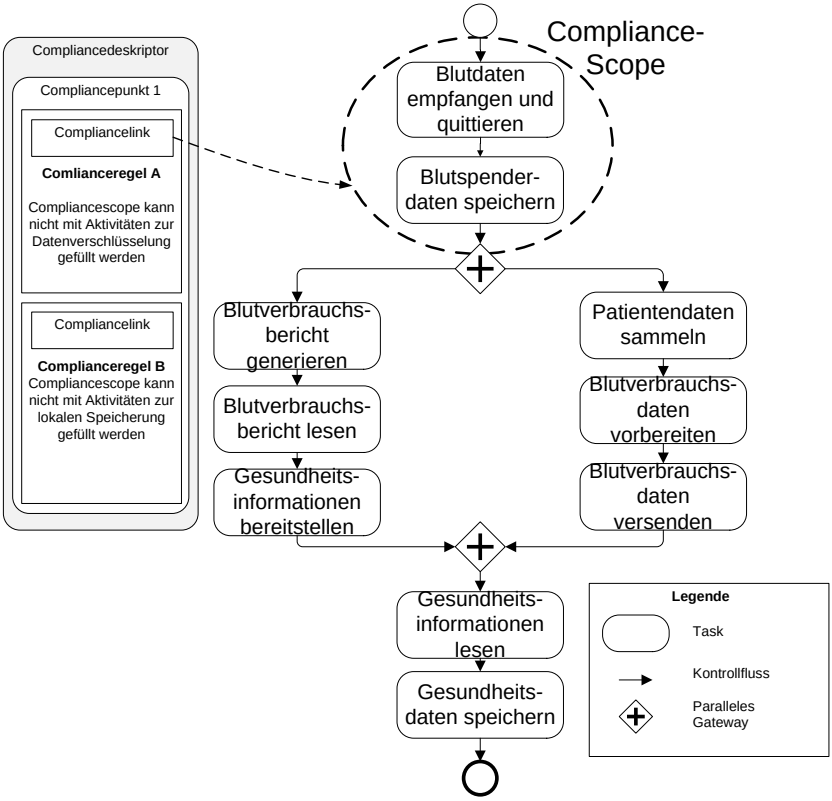

Abbildung 4.7.: Annotation eines Prozessmodells mit einem Compliancedeskriptor

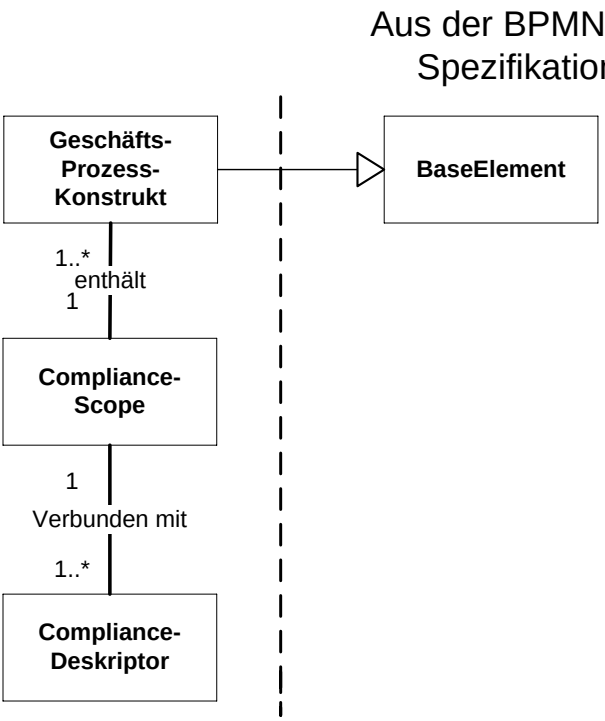

Abbildung 4.8.: Metamodell eines Compliancescopes

Complianceregeln beinhaltet, die auf das gezeigte Prozessmodell angewendet werden sollen. Im Speziellen sieht man, dass die Complianceregel A mit dem Compliancescope verbunden ist.

Abbildung 4.8 zeigt das Metamodell eines Compliancescopes in UML. Ein Compliancescope enthält mindestens ein Prozesskonstrukt. Prozesskonstrukte sind dadurch definiert, dass sie von der Klasse *BaseElement* erben, welche in der BPMN 2.0 Spezifikation definiert

ist. Ein in der BPMN 2.0 Spezifikation beschriebenes Prozesskonstrukt kann zum Beispiel eine Aktivität sein. Weiterhin ist ein Compliancedeskriptor mit einem Compliancescope verbunden. Das Metamodell eines Compliancedeskriptors ist in Abbildung 4.6 zu sehen. Der mit dem Compliancescope verbundene Compliancedeskriptor enthält die Complianceregeln, die auf die im Compliancedeskriptor enthaltenen Aktivitäten angewendet werden sollen.

4.4. Gegenüberstellung der Anwendungsgebiete von Compliancetemplates und Compliancescopes

Sowohl das Konzept des Compliancetemplates als auch das Konzept des Compliancescopes ist dazu geeignet, Prozesse mit regelkonformem Kontrollfluss zu erstellen. Die Konzepte werden jedoch in verschiedenen Anwendungsgebieten eingesetzt. Mit den Compliancetemplates wurde ein Ansatz vorgestellt, der bei der *Neuerstellung* von regelkonformen Prozessen einsetzbar ist. Hierbei wird eine Vorlage für einen Prozess verwendet, die neben freien Stellen vordefinierte Aktivitäten enthält. Die freien Stellen dieser Vorlage können bei der Entwicklung des neuen Prozesses mit Aktivitäten gefüllt werden, während die vordefinierten Aktivitäten bestimmte Aspekte von Complianceregeln implementieren, die nicht geändert werden dürfen. Ein Compliancetemplate definiert somit implizit die Complianceregeln, die ein neuer Prozess einhalten muss.

Compliancescopes werden bei der *Modifikation von bestehenden* Prozessen eingesetzt. Bestehende Prozesse werden hierbei mittels sogenannter Compliancescopes in Bereiche aufgeteilt, in denen bestimmte Complianceregeln gelten. Die Complianceregeln, die für einen be-

stimmten Compliancescope gelten müssen, werden aus einer Menge an bestehenden Complianceregeln ausgewählt und mit dem entsprechenden Compliancescope verknüpft. Es können aber auch Complianceregeln neu definiert und mit einem Compliancescope verbunden werden. Auch dieser Ansatz befasst sich ausschließlich mit kontrollflussbasierten Complianceregeln. Nimmt ein menschlicher Prozessmodellierer Änderungen an einem solchen mit einem Compliancescope versehenen Bereich vor, so kann das verwendete graphische Entwicklungswerkzeug Rückmeldung geben, ob die Modifikation gültig war. Gültige Modifikationen verletzen keine mit dem entsprechenden Compliancescope verbundenen Complianceregel.

In beiden Ansätzen werden Techniken aus dem Bereich des Modelchecking verwendet, um Änderungen automatisch überprüfen zu können. Hierfür wird ein Teil des aktuellen Prozessmodells in ein Petrinetz übersetzt und dann dem Modelchecker in seiner Eingabesprache übergeben.

4.5. Verifizierungsalgorithmus für den Kontrollfluss eines Prozesses

Der Kontrollfluss von Compliancescopes und Complianceregionen wird auf die gleiche Art verifiziert. Wie bei den anderen in diesem Kapitel vorgestellten Konzepten, liegt der Fokus auch bei dem in diesem Abschnitt vorgestellten Verifizierungsalgorithmus, auf dem Kontrollfluss eines Prozessmodells. Der in dieser Arbeit vorgestellte Ansatz verwendet die Technologie des *Modelcheckings* [CGP01], um den Kontrollfluss eines Prozessmodells auf Unvereinbarkeiten mit bestimmten Complianceregeln zu überprüfen.

Modelchecking ist eine Technologie, die ursprünglich zur Untersuchung von komplexen Systemen, wie Prozessoren verwendet wurde. Mit dieser Technologie lassen sich Eigenschaften, die ein solches System erfüllen muss, prüfen. Hierbei wird dieses System in ein Modell des Systems übersetzt. Dieses Modell wird mit einer Sprache beschrieben, die von einem Modelchecker interpretiert und analysiert werden kann. Ein solches Modell kann zum Beispiel die Schaltkreise eines Prozessors nachbilden. Dieses Modell muss auf der einen Seite stark genug abstrahiert sein, damit die Zahl der zu erwartenden Zustände in Grenzen gehalten wird. Auf der anderen Seite muss es komplex genug sein, um nach der Überprüfung von Eigenschaften, Rückschlüsse auf das originale System zuzulassen.

Es gibt viele Softwarewerkzeuge, wie zum Beispiel NuSMV [CCG+02], Prism [KNP02] oder SPIN [Hol03], welche die Konzepte des Modelcheckings umsetzen. In dieser Arbeit wird der SPIN Modelchecker verwendet, da er schon seit 1980 entwickelt wird und in vielen Anwendungen eingesetzt wird [Spi]. Das lange Bestehen des SPIN Modelcheckers führte zu einem ausgereiften Produkt. SPIN wurde seit der Anfangszeit stetig weiterentwickelt und findet auch heute noch in vielen kommerziellen wie auch nicht kommerziellen Projekten Verwendung [JGP99]. In [WMM09] wird weiterhin gezeigt, wie von SPIN erzeugte Gegenbeispiele auf ein ursprüngliches BPMN-Modell abgebildet werden können. Dies ist ein wichtiges Merkmal, das dem menschlichen Prozessmodellierer dabei hilft, die Verletzung einer Complianceregel zu verstehen. Ein weiteres Merkmal des SPIN Modelcheckers ist die Verwendung von Linearer Temporaler Logik (LTL) als Sprache zur Spezifikation von Eigenschaften des zu überprüfenden Modells. Viele Modelchecker wie zum Beispiel LoLA [Sch00] verwenden hierfür

Computation Tree Logic (CTL). Vardi argumentiert in [Var01], dass LTL Spezifikationen für Menschen leichter zu schreiben sind, da bei LTL die Zustände eines Systems linear betrachtet werden, während mit CTL Zustandsbäume beschrieben werden. Weiter argumentiert Vardi, dass der oft in der Literatur angenommene Geschwindigkeitsvorteil von CTL Modelcheckern in der Praxis verloren geht.

Die folgende Beschreibung des Verifizierungsalgorithmus basiert auf der Entscheidung für den SPIN Modelchecker. Aus dieser Entscheidung resultieren einige der Schritte des Verifizierungsalgorithmus. Beispielsweise die Transformation des BPMN-Prozessmodells in die Eingabesprache für SPIN, PROMELA.

Modelle, die mit dem SPIN Modelchecker überprüft werden sollen, müssen in der Eingabesprache PROMELA [Ger97] (siehe Abschnitt 2.8) geschrieben werden. PROMELA ist eine C-ähnliche Sprache, die entwickelt wurde, um die drei wichtigsten Bestandteile eines SPIN-Modells zu definieren: Prozesse, Kanäle und Variablen.

Die in der Literatur vorgestellten Konzepte zur Untersuchung von Prozessmodellen auf Regelkonformität untersuchen das gesamte Prozessmodell [ETHP10, Awa10, WMM09, STK+10]. Das in der vorliegenden Arbeit vorgestellte Konzept verwendet Compliancescopes für die Definition von Regionen in einem Prozessmodell, in denen bestimmte Complianceregeln gelten. Beispiele für solche Complianceregeln finden sich in Tabelle 4.1 und in den Abbildungen 4.2 und 4.7. Wird durch das Ändern von Aktivitäten innerhalb eines Compliancescopes eine mit diesem Compliancescope verknüpfte Complianceregel verletzt, muss nur der Prozessteil in diesem Compliancescope überprüft werden.

Da diese Arbeit sich mit der Erstellung von regelkonformen Pro-

zessmodellen in BPMN 1.0 auseinandersetzt, muss das BPMN 1.0-Prozessmodell in eine Repräsentation in PROMELA übersetzt werden, um mit dem SPIN Modelchecker überprüft werden zu können.

Es existieren mehrere Ansätze für die Überführung von BPMN 1.0 Prozessmodellen nach PROMELA, die im Folgenden erläutert werden.

Im ersten Ansatz [VF07] werden alle Konstrukte eines BPMN 1.0 Prozessmodells in eigenständige PROMELA-Prozesse übersetzt. Der Nachrichtenaustausch zwischen den Konstrukten des BPMN 1.0-Prozessmodells wird mittels Kanälen zwischen den PROMELA-Prozessen umgesetzt [Gro11]. Die Verwendung dieses Ansatzes führt zu unübersichtlichen PROMELA Programmen, die eine Zuordnung von PROMELA Programmabschnitten zu BPMN-Prozessteilen erschweren.

Der zweite Ansatz verwendet für die Übersetzung von BPMN Prozessmodellen in PROMELA ein Petrinetz als Zwischenmodell. Da Petrinetze sehr ausführlich in der Literatur behandelt werden und in vielen Bereichen Anwendung finden, wird dieser Ansatz in der vorliegenden Dissertation für die Transformation von BPMN 1.0 Prozessmodellen in PROMELA Programme verwendet. Für die Definition der BPMN 2.0 Ausführungssemantik wurde ein sogenanntes Tokenmodell herangezogen. Ein Token ist ein Konzept, um die Ausführung von Sequenzen in einem Prozessmodell zu verdeutlichen. Ein Token wird, ähnlich wie bei Petrinetzen, zwischen den BPMN 2.0 Konstrukten weitergereicht.

Abbildung 4.9 zeigt die Schritte die bei der Überprüfung von Prozessmodellen durch Modelchecker durchgeführt werden müssen. In den folgenden Abschnitten werden diese Schritte erläutert.

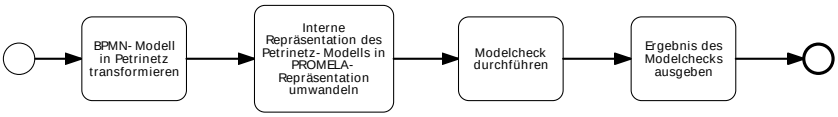

Abbildung 4.9.: Überprüfungsschritte für BPMN Prozesse geschrieben in BPMN

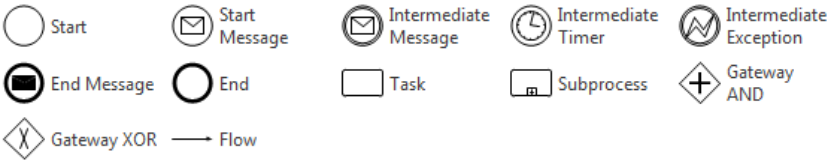

Abbildung 4.10.: In dieser Dissertation verwendete Grundmenge von BPMN 1.0 Elementen

4.5.1. Transformation von BPMN in Petrinetze

Für die Überprüfung von Compliancescopes werden die in dem betreffenden Compliancescope enthaltenen Aktivitäten in ein Petrinetzmodell transformiert. In [DDO08] werden Transformationen der wichtigsten BPMN 1.0 Konstrukte auf entsprechende Petrinetzfragmente gezeigt. Die in der vorliegenden Dissertation verwendete Grundmenge von BPMN 1.0 Elementen ist in Abbildung 4.10 dargestellt. Diese Menge deckt sich mit der in [DDO08] verwendeten Grundmenge an BPMN-Elementen. In der vorliegenden Dissertation wird nur mit dieser Menge an BPMN 1.0 Elementen gearbeitet, da die in [DDO08] vorgestellten Ergebnisse unverändert weiterverwendet werden sollen. Die Bedingung für eine unveränderte Weiterverwendung ist es, die

Grundannahmen von [DDO08] nicht zu verändern.

Abbildung 4.11 zeigt einige der grundlegenden BPMN-Sprachelemente und deren Entsprechung als Petrinetzkonstrukt. Die mit *Inklusive Aufteilung* und *Inklusive Zusammenführung* bezeichneten Transformationen wurden in [DDO08] nicht dargestellt und sind im Rahmen der vorliegenden Arbeit erstellte Ergänzungen. Um ein BPMN-Prozessmodell in ein Petrinetz zu überführen wird das BPMN-Prozessmodell durchlaufen und jedes BPMN Konstrukt nacheinander auf ein Petrinetzkonstrukt abgebildet. Dabei werden die gestrichelten Plätze in den Petrinetzkonstrukten ersetzt.

Die Beschreibung der Verwendung von Modelcheckern und die Beschreibung der Transformation von BPMN-Prozessmodellen verdeutlicht die Anwendbarkeit der wissenschaftlichen Beiträge der vorliegenden Arbeit.

BPMN-Elemente, die mit dem Datenfluss innerhalb eines BPMN-Prozesses zu tun haben, werden bei der Abbildung in ein Petrinetz nicht berücksichtigt, da sich dieses Kapitel ausschließlich mit dem Kontrollfluss eines Geschäftsprozesses befasst. In Kapitel 5 wird dargelegt, wie der Datenfluss eines Geschäftsprozesses auf Regelkonformität untersucht werden kann.

4.5.2. Repräsentation von Petrinetzen in PROMELA

Um mit dem SPIN Modelchecker überprüft werden zu können, muss das aus einem BPMN 1.0 Prozess erzeugte Petrinetz in ein PROMELA Programm übersetzt werden. Der Zwischenschritt bei der Transformation über ein Petrinetz hat sich in der von Stefan Grohe [Gro11] durchgeführten Untersuchung als am vorteilhaftesten herausgestellt.

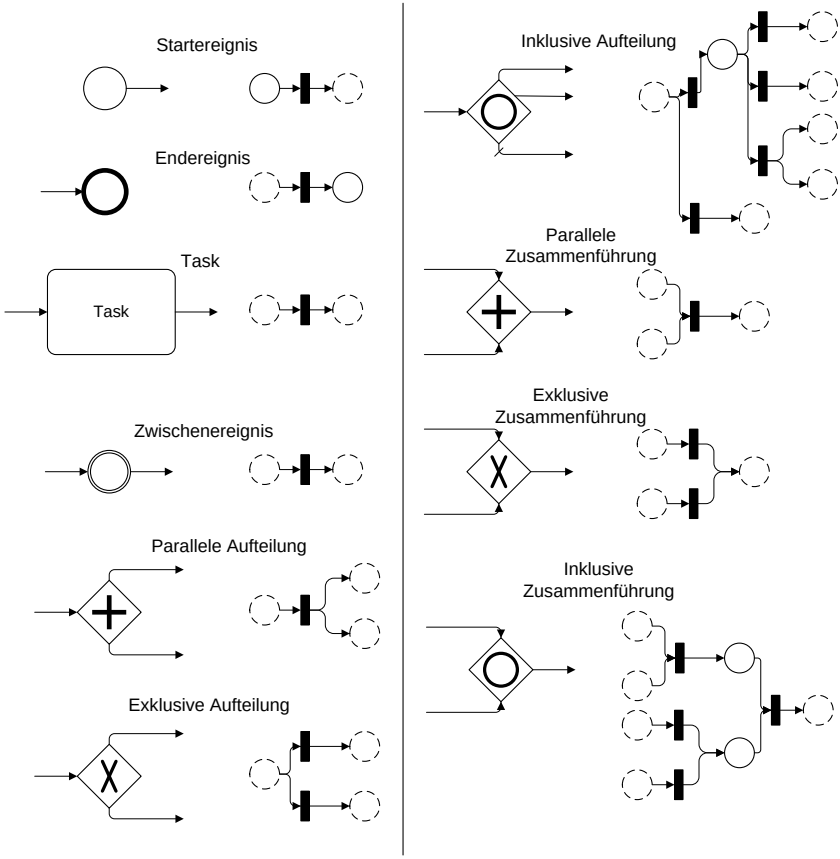

Abbildung 4.11.: Abbildung von BPMN 1.0 Konstrukten auf Petrinetze (angelehnt an [DDO08])

Es ist mit diesem Ansatz einfacher, Rückschlüsse auf das ursprüngliche BPMN-Prozessmodell zu ziehen. Für die Repräsentation der Plätze eines Petrinetzes wird der in [RMF07] vorgestellte Ansatz verwendet. In diesem Ansatz repräsentiert ein Array von Integerwerten die Plätze eines Petrinetzes. Abgesehen davon werden Makros verwendet. Makros werden zum Beispiel verwendet, um Konstanten zu definieren. Diese Konstanten können dann im Quelltext eines Programms verwendet werden. Der Compiler ersetzt bei der Übersetzung des Programms die Konstanten mit den ihnen zugewiesenen Werten. In dieser Arbeit wird mit Makros festgelegt, wann eine Transition schalten kann und was beim Schalten einer Transition passiert. Das folgende Makro zeigt einen Array, welcher die Plätze des entsprechenden Petrinetzes repräsentiert. Dieser Codezeile nach zu urteilen hat das Petrinetz 12 Plätze: die Plätze 0 bis 11.

```
byte places[11];
```

Die Makros, die dazu verwendet werden, das Verhalten des Petri-Netzes zu modellieren, sind in zwei Bereiche aufgeteilt. Im ersten Bereich wird definiert, in welchen Zuständen des Petri-Netzes welche Transitionen schalten können. Das folgende Codebeispiel zeigt ein Makro, welches definiert, dass die Transition 0 schalten kann, wenn auf Platz 0 eine Marke liegt und auf Platz 2 keine Marke liegt.

```
#define transition0Ready (places[0] && !places[2])
```

Im zweiten Verwendungsbereich von Makros wird definiert, welche Marken von einer Transition konsumiert werden und welche Marken erzeugt werden, wenn diese Transition aktiviert wird. Das nächste Codebeispiel zeigt das Makro mit Namen *transition0Fire* welches definiert, dass beim Aktivieren der Transition 0 eine Marke von Platz 0

konsumiert wird und auf Platz 2 eine Marke erzeugt wird.

```
#define transition0Fire  places[0] = 0;
places[2] = 1;
```

Codebeispiel 4.1 zeigt eine *do*-Schleife, welche die Ausführung eines Petrinetzes in PROMELA steuert. In jedem Schleifendurchgang wird ein Zweig, der mit einem doppelten Doppelpunkt beginnt, ausgewählt. Nach den jeweiligen Doppelpunkten steht der sogenannte *Guard*. Ein Guard ist eine Bedingung, die eintreffen muss, damit die nachfolgenden Instruktionen ausgeführt werden können. Im Fall eines PROMELA Programms, das die Ausführung eines Petrinetzes simuliert, wird ein solcher Guard durch die ready-Makros definiert. Sie beschreiben, welche Plätze des Petrinetzes mit Marken versehen sein müssen, damit eine bestimmte Transition schalten kann.

Die nach dem Guard stehenden Anweisungen werden in der *d_step* Umgebung ausgeführt. Die d_step Umgebung sorgt dafür, dass die in ihr enthaltenen Anweisungen wie eine einzelne Anweisung gesehen werden und deterministisch ausgeführt werden. Dies bedeutet, dass die Ausführung dieser Anweisungen nicht durch andere nebenläufige Prozesse unterbrochen werden kann.

In den in Codebeispiel 4.1 dargestellten d_step Umgebungen wird mittels der *printf* Funktion auf der Konsole ausgegeben, welche Transition gerade geschalten hat, und es wird das Makro aufgerufen, das die Anweisungen zum Schalten der jeweiligen Transition enthält. In Codebeispiel 4.1 wird auch ein Beispiel gezeigt, wie die do-Schleife abgebrochen wird. Befindet sich eine Marke auf dem Platz 2, so wird die Ausführung des Programms mittels der nachfolgenden Goto-Anweisung zum Label *accept* geleitet.

Listing 4.1: Hauptteil des PROMELA-Programms, das für die Ausführung der Makros zuständig ist

```
do
  :: transition0Ready -> d_step{printf("PROCESSED_transition
       0");
     transition0Fire}
  :: transition1Ready -> d_step{printf("PROCESSED_transition
       1");
     transition1Fire}
  :: p[2] -> goto accept
od;
accept: printf("Accepted");
```

Das Codebeispiel A.1 im Anhang zeigt das gesamte laufende Beispiel aus Abbildung 4.1 als PROMELA Programm.

Im Folgenden wird das PROMELA Konstrukt einer *Never-Claim* beschrieben. Das Verständnis von Never-Claims trägt zum Verständnis des Algorithmus zur Überprüfung von Compliancescopes bei. Eine sogenannte Never-Claim wird in Codebeispiel 4.2 gezeigt. Eine Never-Claim ist die negierte Version der LTL-Formel, die mit dem zu überprüfenden Modell dem Modelchecker übergeben wird. Der in dieser Arbeit verwendete Modelchecker SPIN erzeugt aus einer ihm übergebenen LTL-Formel durch Negation eine Never-Claim. Diese negierten LTL-Formeln sind insofern beim Vorgang des Modelchecking nützlich, als dass damit die Erstellung eines Gegenbeispiels für den Modelchecker möglich wird. Tritt die Bedingung einer Never-Claim ein, die nicht eintreten darf, so kann der Modelchecker den Ausführungspfad, der zum Eintreten dieser Bedingung geführt hat, als Gegenbeispiel zurückgeben. Das Gegenbeispiel zeigt, dass das zu überprüfende Modell nicht den in der mitgegebenen LTL-Formel beschriebenen Eigenschaften

genügt.

Listing 4.2: Beispiel für eine Never-Claim

```
never  {    / !(<>(Task3)) /
accept_init:
T0_init:
  if
  :: (! ((Task3))) -> goto T0_init
  fi;
}
```

Die Transformation eines Prozessmodells in ein Petrinetz wird in [RMF07, DDO08] verwendet und in [Gro11, Wol10, WMM09] umgesetzt, um BPMN-Prozessmodele in PROMELA Programme zu überführen.

Algorithmus 4.1 Überprüfung von Compliancescopes

```
1: function überprüfeCompliance(Parameter: Compliancescope, ComplianceRegel)
2:     Petrinetz = erstellePetriNetz(Compliancescope);
3:     PromelaProgramm = erstellePromelaProgramm(Petrinetz);
4:     NeverClaim = erstelleNeverClaim(ComplianceRegel);
5:     if überprüfeMitModelChecker(PromelaProgramm, NeverClaim) then
6:         zeigeNachricht: "Compliancescope ist regelkonform.";
7:     else
8:         zeigeGegenbeispiel();
9:     end if
10: end function
```

Algorithmus 4.1 zeigt die Schritte, die für die Überprüfung eines Compliancescopes durchgeführt werden müssen. Die dort gezeigte Funktion *überprüfeCompliance* zeigt die Hauptbestandteile des Verifizierungsalgorithmus. Eingabeparameter dieser Funktion sind zwei Ob-

jekte. Die Eingabeobjekte enthalten den zu überprüfenden Compliancescope und eine zur Überprüfung herangezogene Complianceregel in Linearer Temporaler Logik (LTL). In Zeile zwei des Programms wird der übergebene Compliancescope in eine interne Repräsentation eines Petrinetzes transformiert. LTL-Formeln können in dem in dieser Arbeit vorgestellten Prototyp graphisch erstellt werden. Die nächsten beiden Zeilen des Algorithmus transformieren die LTL-Formel in eine *Never Claim* in PROMELA und das Petrinetz in eine Repräsentation in PROMELA. Nach den Transformationsschritten wird die Never Claim zusammen mit dem zu überprüfenden Modell an den Modelchecker übergeben. Genügt das Modell den in der LTL-Formel definierten Anforderungen wird dies dem Benutzer mit der Meldung „Compliancescope ist regelkonform.“ angezeigt. Im anderen Fall wird ein Gegenbeispiel generiert, das zeigt, welcher Ausführungspfad im Modell die Complianceregel verletzt.

4.6. Zusammenfassung

Die in diesem Kapitel beschriebenen Konzepte zeigen den menschlichen Prozessmodellierern bei der Erstellung regelkonformer Prozesse Complianceverletzungen an. Der Fokus in diesem Kapitel liegt auf Complianceregeln, die den Kontrollfluss eines Prozesses betreffen. Es wurden mit dem Compliancetemplate (siehe Abschnitt 4.2) und dem Compliancescope (siehe Abschnitt 4.3) zwei Mechanismen vorgestellt, mit denen diese Ziele erreicht werden.

Compliancetemplates unterstützen menschliche Prozessentwickler bei der Erstellung neuer, regelkonformer Prozesse. Compliancescopes werden auf bestehende Prozesse angewendet, um Änderungen an

diesen Prozessen automatisch überprüfbar zu machen. Beide Konzepte sind formal definiert und es wurde ein Verifizierungsalgorithmus beschrieben, der in beiden Konzepten verwendet wird.

Gegenüber der Überprüfung des gesamten Prozessmodells bei jeder Änderung hat die Überprüfung mit Hilfe von Compliancescopes den Vorteil, dass der Benutzer bestimmen kann, wie viele Aktivitäten in einem Compliancescope enthalten sind und wie viele Complianceregeln mit diesem Compliancescope verknüpft sind. Diese beiden Faktoren haben starken Einfluss auf das Laufzeitverhalten der Complianceprüfung. Mit der Verwendung von Compliancescopes kann ein Benutzer das Gebiet in einem Prozess, dass bei einer Modifikation automatisch überprüft wird einschränken, und somit die Laufzeit der Complianceprüfung auf einem angemessenen Niveau halten. Compliancescopes unterscheiden sich in mehreren Bereichen von Subprozessen, wie sie in der BPMN Spezifikation definiert sind.

- Compliancescopes können eine beliebige Form annehmen. Mit dieser Flexibilität ist es möglich beliebige Mengen von Aktivitäten eines Prozesses in einen Compliancescope aufzunehmen. BPMN Subprozesse werden als Rechtecke mit abgerundeten Kanten gezeichnet.

- Im Gegensatz zu den Grenzen von BPMN Subprozessen können die Grenzen von Compliancescopes von Kontrollflusskonnektoren überquert werden. Dies ist ein weiterer Punkt die Flexibilität des Einsatzes von Compliancescopes zu erhöhen.

- Ein Compliancescope kann auch nur Objekte, die keine BPMN-Tasks sind, beinhalten.

- Compliancescopes können sich mit anderen Compliancescopes überlappen. Das heißt, sie können Teilmengen von BPMN-Elementen von anderen Compliancescopes enthalten.

- Zum Konzept der Compliancescopes gehört der Überprüfungsalgorithmus. Im Gegensatz zu bestehenden Arbeiten wird in diesem Algorithmus neben der Auswertung der Complianceregel auch die Erfüllbarkeit der Complianceregel getestet.

Die Flexibilität, die durch die oben aufgeführten Eigenschaften von Compliancescopes bedingt ist, hilft menschlichen Prozessentwicklern dabei, Compliancescopes so zu gestalten, dass verletzte Complianceregeln schnell lokalisiert werden können. Somit muss nicht das gesamte Prozessmodell bei der Behebung einer verletzten Complianceregel betrachtet werden, sondern nur der betroffene Compliancescope.

Weiterhin sind Compliancescopes ein Werkzeug, um den Umgang mit Complianceregeln, die auf einen Prozess angewendet werden, zu erleichtern. Würden alle diese Complianceregeln mit dem Gesamtprozess verknüpft werden, so würde die Menge der Complianceregeln ab einer bestimmten Zahl unübersichtlich und unverständlich werden. Mit Compliancescopes kann man Complianceregeln dort mit dem Prozess verknüpfen, wo sie Anwendung finden.

Im folgenden Kapitel wird ein Konzept vorgestellt, welches auch den Datenfluss in einem Prozessmodell einschränkt. Der Datenfluss eines Prozesses ist neben dem Kontrollfluss die zweite Dimension, die es bei der Entwicklung regelkonformer Prozesse zu beachten gibt.

Prozesse können heute aufgeteilt und an verschiedenen Orten ausgeführt werden [Kha08]. Bei der Ausführung eines verteilten Prozesses

kann es zu Einschränkungen des Datenflusses bedingt durch Complianceanforderungen kommen. Dies ist beispielsweise der Fall, wenn bestimmte Daten bestimmte Landesgrenzen nicht überschreiten dürfen.

KAPITEL 5

ENTWICKLUNG VON PROZESSEN MIT REGELKONFORMEM DATENFLUSS

Dieses Kapitel befasst sich mit datenbasierten Complianceregeln. Es definiert hierfür das Konzept einer Compliancedomain in Abschnitt 5.2. Ein Verifizierungsalgorithmus für Compliancedomains wird in Abschnitt 5.3 vorgestellt. Das Kapitel schließt mit Abschnitt 5.5, in welchem ein Ansatz zur Kombination von datenbasierten mit kontrollflussbasierten Complianceregeln zu einer zusammengesetzten Complianceregel gezeigt wird.

Neben dem Kontrollfluss spielt auch der Datenfluss in Prozessen eine große Rolle, wenn es um die Einhaltung von Gesetzen oder Regeln geht. Der Datenfluss eines Prozesses beschreibt, wie die Daten

in einem Prozess fließen und transformiert werden. Daten werden zum Beispiel von Aktivitäten verändert und zur weiteren Verarbeitung für andere Aktivitäten freigegeben. Im Folgenden werden zwei unterschiedliche Arten von Regeln und Gesetzen berücksichtigt. Zum einen die Regeln und Gesetze, die auf die Abfolge von Aktivitäten, also den Kontrollfluss eines Prozesses, zielen. Zum anderen die Arten von Regeln und Gesetzen, die auf die in einem Prozess verwendeten Daten Anwendung finden.

5.1. Beispielprozess

Ein wichtiger Bereich sind Gesetze, die den Umgang mit persönlichen Daten betreffen. Gerade im medizinischen Bereich ist der verantwortungsvolle Umgang mit solchen Daten unabdingbar. Das in dieser Arbeit verwendete Beispielszenario ist in diesem Bereich platziert und stellt die Abfolge von Schritten dar, die durchlaufen werden, wenn eine Blutspende durchgeführt wird. Die einzelnen Schritte sind in Abbildung 5.1 dargestellt. Um die bei einer Blutspende entstehenden Patientendaten zu schützen, muss ein solcher Blutspendeprozess einige Regeln einhalten. Beispielsweise sollen Patientendaten nur anonymisiert von der Blutspendestation an Krankenhäuser weitergegeben werden.

Datengetriebene Complianceregeln werden in dieser Arbeit losgelöst von kontrollflussgetriebenen Complianceregeln eingeführt. Diese Trennung ist sinnvoll, um die Besonderheiten dieser Arten von Complianceregeln besser vermitteln zu können.

Ein Anwendungsgebiet, auf welchem datengetriebene Complianceregeln eine große Rolle spielen, ist das Cloud Computing [Ley09].

Konzepte des Cloud Computing werden sich in nächster Zeit vermehrt in IT-Infrastrukturen von Unternehmen durchsetzen. Cloud Computing verspricht Unternehmen viel Einsparungspotential für die Unterhaltung ihrer IT-unterstützten Operationen. Dies wird einerseits durch die Auslagerung von IT-gesteuerten Prozessen und Programmen zu Public-Clouds [MG09] von externen Cloud-Providern erreicht, auf der anderen Seite können Unternehmen ihre eigene IT-Infrastruktur effizienter nutzen, wenn sie eine sogenannte Private Cloud [MG09] einführen. Zwischen diesen beiden Modellen befindet sich das Konzept einer Hybrid-Cloud [MG09], welches die beiden zuerst genannten Cloudarten vereint. In einer Hybrid-Cloud können zum Beispiel Kundendaten in der eigenen Private-Cloud verarbeitet werden, während Operationen wie zum Beispiel die Steuerung der Maschinen eines Unternehmens in der Public-Cloud durchgeführt werden können. Ein Teil eines Prozesses könnte zum Beispiel in der Private-Cloud eines Unternehmens ausgeführt werden, während ein anderer Teil desselben Prozesses bei einem Partner des Unternehmens ausgeführt wird, der eine Public-Cloud betreibt. Für diese beiden Teile gelten entweder vom Gesetzgeber oder vom Unternehmen selbst vorgeschriebene Regeln bezüglich der Daten, die in ihm verarbeitet werden dürfen. Diese Regeln wirken sich darauf aus, welche Daten zwischen den Teilen des Prozesses hin- und hergeschoben werden dürfen.

In Bezug auf den Datenschutz ist es notwendig schon bei der Entwicklung von Prozessen Mechanismen bereitzustellen, die den menschlichen Prozessmodellierer dabei unterstützen, regelkonforme Prozesse zu erstellen. Ein Konzept, welches zur Entwicklungszeit eines Prozesses eingesetzt werden kann und diese Vorgaben erfüllt, wird in diesem Kapitel vorgestellt.

Dieses Konzept trägt den Namen *Compliancedomain* [SFG+11]. Eine solche Compliancedomain stellt in einem Prozess einen Bereich dar, der auf einer bestimmten Infrastruktur ausgeführt wird. Gleichzeitig gelten für diesen Bereich bestimmte, datenbasierte Complianceregeln, die an eine Compliancedomain annotiert sind.

5.2. Compliancedomains

Compliancedomains erben alle Eigenschaften von Compliancescopes und erweitern sie mit Eigenschaften, die bei der Untersuchung von datenbasierten Complianceregeln herangezogen werden können.

Compliancedomains haben zwei grundsätzliche Aufgaben. Sie werden erstens benötigt, um ein Prozessmodell in Bereiche aufzuteilen. Diese Bereiche können auf verschiedenen Ausführungsumgebungen laufen [Kha08]. Ausführungsumgebungen können zum Beispiel das private Rechenzentrum einer Firma oder eine Public-Cloud sein. Die zweite Aufgabe ist die Annotation von Bereichen eines Prozessmodells mit datenbasierten Complianceregeln. Compliancedomains werden somit auf schon bestehende Prozessmodelle angewendet.

Abbildung 5.1 zeigt den Beispielprozess aus Abbildung 4.1. Dieser Prozess ist mit Compliancedomains versehen. Diese Compliancedomains stellen die verschiedenen physischen Orte dar, an denen dieser Beispielprozess ausgeführt wird. Compliancedomain 0 repräsentiert die Orte, an denen Prozessteile, die mit Blutspenden zu tun haben, ausgeführt werden. Compliancedomain 1 repräsentiert die Verwaltung der Krankenhäuser und Blutspendeeinrichtungen. Hier werden Statistiken über die verfügbaren und bereits verbrauchten Blutkonserven geführt. Compliancedomain 2 repräsentiert die öffentlichen

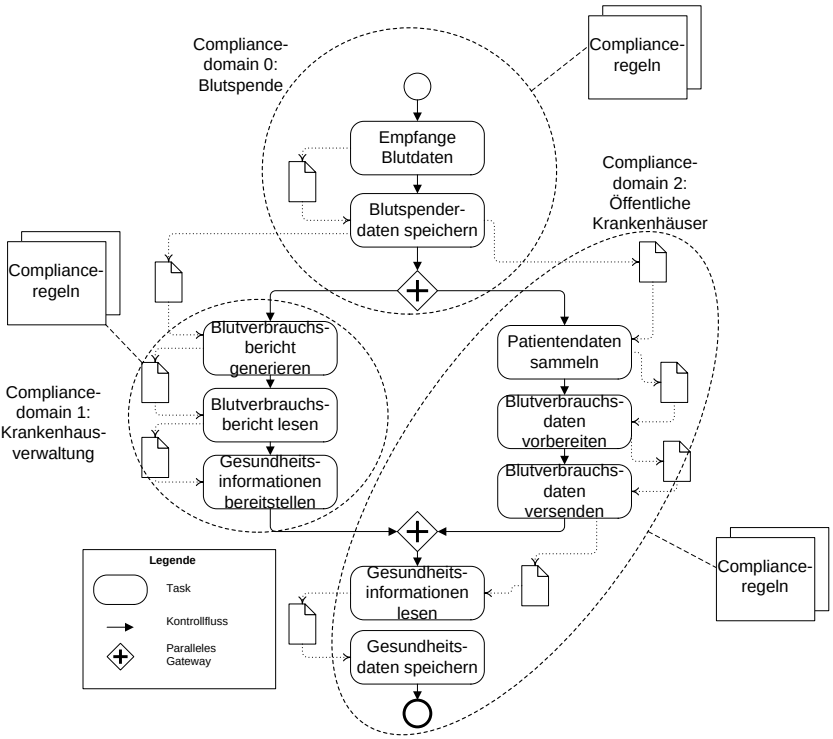

Abbildung 5.1.: Beispielprozess versehen mit Datenobjekten und Compliancedomains

Krankenhäuser. Hier werden die Blutspenden weiterverarbeitet und den Patienten verabreicht.

An diesen Compliancedomains ist jeweils ein Satz von datenbasierten Complianceregeln angeheftet. Beispiele für solche datenbasierten Complianceregeln werden in der folgenden Liste angeführt:

- Ein Beispiel für eine datenbasierte Complianceregel für Com-

pliancedomain 0 ist, dass Daten von Blutspenden in diesem Prozess nicht mit Patientenakten abgeglichen werden dürfen. Dies bedeutet, dass keine Patientendaten aus den öffentlichen Krankenhäusern in dieser Compliancedomain verarbeitet werden dürfen.

- In Compliancedomain 2 dürfen keine Namen von Blutspendern verarbeitet werden. Dies hat zur Folge, dass solche Nachrichten von Compliancedomain 0 zu Compliancedomain 2 keine Namen enthalten dürfen. Beispielsweise müssen Blutkonserven in Compliancedomain 2 mittels Nummern identifiziert werden.

- Anhand von Daten der öffentlichen Krankenhäuser und der Daten aus den Blutspendestationen generiert die Verwaltung der Krankenhäuser und Blutspendestationen einen Blutverbrauchsbericht. Eine für Compliancedomain 1 geltende Complianceregel ist, dass keine Daten über die Anzahl der Blutspender verarbeitet werden dürfen. Es dürfen also von Compliancedomain 0 nur zusammengefasste Daten über die Menge des gespendeten Blutes an Compliancedomain 1 geschickt werden.

Vergleicht man Compliancedomains mit den in Kapitel 4.3 vorgestellten Compliancescopes, so werden einige Gemeinsamkeiten deutlich. Zum einen ist beiden Konzepten gemein, dass sie in einem Prozessmodell bestimmte Bereiche markieren, für die bestimmte Eigenschaften gelten. Zum anderen werden, wie bei Compliancescopes auch, Complianceregeln mit den in einem Prozessmodell eingefügten Compliancedomains verknüpft. Der Unterschied zwischen beiden Konzepten liegt im jeweiligen Anwendungsbereich. Während Compliance-

scopes dafür verwendet werden, um den Kontrollfluss eines Prozessmodells zu regulieren, hat der Einsatz von Compliancedomains in einem Prozessmodell deutlich weitere Auswirkungen auf den späteren produktiven Einsatz eines Prozesses. Die Umrahmung eines Teils eines Prozessmodells weist den darin enthaltenen Prozesskonstrukten eine bestimmte Ausführungsumgebung zu, die für alle in einer Compliancedomain enthaltenen Prozesskonstrukte gleich ist. Dies ist bei der Anwendung von Compliancescopes nicht der Fall. Die Bindung von Prozessfragmenten, die von einem Compliancescope umrahmt werden, ist somit nicht so stark, wie die durch eine Compliancedomain erwirkte Bindung.

Im Beispiel in Abbildung 5.1 könnte die IT-Infrastruktur, die diesen Prozess unterstützt, wie folgt aussehen:

Die Services, die von den Tasks aufgerufen werden, die von Compliancedomain 0 eingerahmt sind, könnten auf einem Laptop in einem Blutspendemobil des Roten Kreuzes ausgeführt werden. Der Teil des Prozesses, der von Compliancedomain 1 eingerahmt wird, könnte in einer Public-Cloud ausgeführt werden, da hier unkritische Daten verarbeitet werden. Das heißt alle Daten, die in Compliancedomain 1 verarbeitet werden sind anonymisiert und zusammengefasst. Es wird angenommen, dass aus diesen Daten keine Rückschlüsse auf real existierende Personen gezogen werden können. Der Teil des Prozesses, der von Compliancedomain 2 eingerahmt ist, könnte in einer sogenannten Community-Cloud ausgeführt werden. Diese stellen Cloud-Lösungen für Kunden mit bestimmten Anforderungen bereit. Diese Anforderungen könnten zum Beispiel auf die bereitgestellte Software abzielen. Im oben gezeigten Beispiel könnte eine solche Community-Cloud von einem Verbund von öffentlichen Krankenhäusern bezahlt werden, so

dass die von den Krankenhäusern erhobenen Daten immer noch unter der Datenhoheit dieser Krankenhäuser bleiben. Die Krankenhäuser können so den gesetzlichen Anforderungen bezüglich der Verarbeitung von Patientendaten gerecht werden.

5.2.1. Definition von Compliancedomains

Formal sind Compliancedomains an die Definition von Compliancescopes angelehnt. Compliancedomains sind auch Hyperkanten H in einem Hypergraphen G. Auch für Compliancedomains gilt entsprechend Definition 5.

Übertragen auf Business Process Model and Notation 1.0 (BPMN 1.0) entspricht ein Hypergraph H einem BPMN Prozess. Die Knoten in H entsprechen den Aktivitäten in diesem BPMN Prozess-Fragment und die Kanten in H entsprechen den Compliancedomains.

Abbildung 5.2 zeigt ein UML Modell der Komponenten einer Compliancedomain. Compliancedomains erben alle Eigenschaften von Compliancescopes. Demnach beinhaltet eine Compliancedomain mindestens ein Prozesskonstrukt und sie ist mit einem Compliancedeskriptor verbunden. Prozesskonstrukte sind allgemein alle Elemente, die in einer Prozesssprache enthalten sind. Weiter enthält eine Compliancedomain ein Input-Datenformat und ein Output-Datenformat. Diese beiden Eigenschaften beschreiben jeweils, welche Daten in eine Compliancedomain hinein und welche herauskopiert werden dürfen. Als Beschreibungssprache dieser Eigenschaften wird in der vorliegenden Arbeit XML verwendet. Das vorliegende Konzept zur Beschreibung von Datenformaten kann jedoch auf beliebige Beschreibungssprachen angewendet werden.

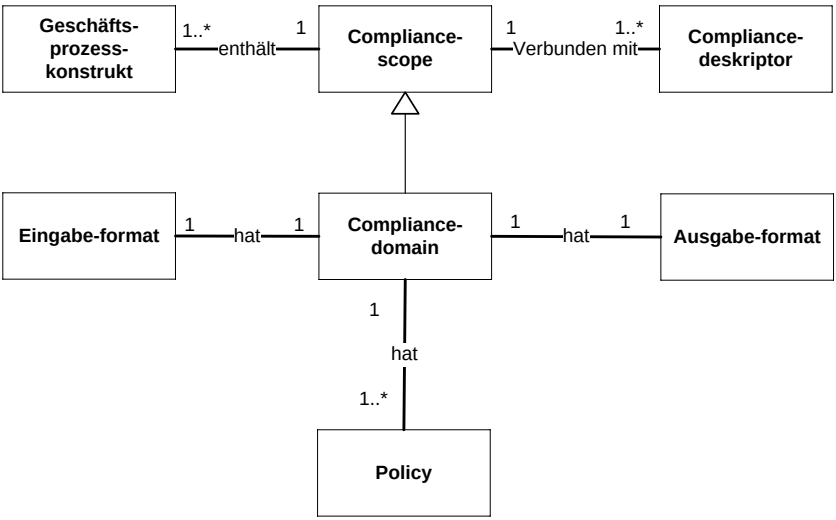

Abbildung 5.2.: Meta-Modell einer Compliancedomain

Aus technischer Sicht sind die Input- und Output-Formate einer Compliancedomain Formatvorlagen für die Daten, die in einer Compliancedomain verarbeitet werden dürfen. Ein Beispiel für eine solche Formatvorlage ist, dass alle Attribute eines Datensatzes, der eine Person darstellt, in einer Compliancedomain verarbeitet werden dürfen. Somit sind alle anderen Datensätze, die nicht dieser Spezifikation genügen, von der Verarbeitung in dieser Compliancedomain ausgeschlossen. Des Weiteren ist eine Compliancedomain mit einer oder mehreren Policies verknüpft. Mit diesen Policies können Eigenschaften von Ausführungsumgebungen beschrieben werden, auf denen die jeweiligen Compliancedomains ausgeführt werden dürfen. Policies können zum Beispiel mit dem Policy-Framework WS-Policy [Web07]

erstellt werden. Das vorliegende Konzept ist jedoch auf keine spezielle Sprache beschränkt. Das Konzept einer Policy wird vorgestellt, um ein vollständiges Bild einer Compliancedomain und ihres Anwendungsbereichs zu präsentieren.

Im Folgenden wird anhand eines Beispiels gezeigt, wie Compliancedomains bestimmten Ausführungsumgebungen, wie beispielsweise Clouds, zugeordnet werden können. Dieses Beispiel arbeitet mit den drei Cloud-Arten: Public-Cloud, Hybrid-Cloud und Private-Cloud. Die Zuordnung von Compliancedomains auf Ausführungsumgebungen ist jedoch nicht auf diese drei Cloud-Arten beschränkt. Welche Cloud-Arten bei der Zuordnung in Betracht gezogen werden, muss von Fall zu Fall entschieden werden. Als weitere Ausführungsumgebungen wären zum Beispiel auch private Rechenzentren, Outsourced Private-Clouds oder Outsourced Community-Clouds denkbar.

Wie in Tabelle 5.1 dargestellt, sind die möglichen Ausführungsumgebungen, die für eine Compliancedomain in Frage kommen, gegliedert. Die Ausführungsumgebung wird danach festgelegt, mit welcher Wichtigkeit die in ihr verarbeiteten Daten vom menschlichen Prozessmodellierer bewertet wurden. Die Wichtigkeit dieser Daten richtet sich danach, wie bedeutsam diese Daten für die Firma oder Organisation sind, der sie gehören. Die Wichtigkeit wird in Stufen von eins bis drei festgelegt. Wobei eins die höchste Wichtigkeit darstellt und drei die niedrigste.

Compliancedomains, die Daten mit der höchsten Wichtigkeit verarbeiten, dürfen laut Tabelle 5.1 nur in einem organisationseigenen Rechenzentrum verarbeitet werden. Daten, die mit Wichtigkeit zwei bewertet wurden, dürfen in einem privaten Rechenzentrum, in einer Community-Cloud oder einer Hybrid-Cloud verarbeitet werden. Für

Tabelle 5.1.: Zusammenhang der Wichtigkeit von Daten für eine Organisation und deren mögliche Verarbeitung in Cloud-Umgebungen

		Wichtigkeit		
		1	**2**	**3**
Cloud-Umgebungen	Public-Cloud			X
	Hybrid-Cloud		X	X
	Private-Cloud	X	X	X

Community-Clouds gilt, dass ein Anbieter für mehrere Organisationen oder Firmen Dienste anbietet, die den Anforderungen genügen, die diese Organisationen oder Firmen in einem Bereich gemein haben. Daten mit der niedrigsten Wichtigkeit dürfen in Public-Clouds verarbeitet werden. Weiterhin dürfen solche Daten auch in allen anderen Cloud-Arten und privaten Rechenzentren verarbeitet werden.

5.3. Verifizierungsalgorithmus

Im Gegensatz zu Compliancescopes wird bei Compliancedomains kein Modelchecker verwendet, um Complianceregeln zu überprüfen. Datenbasierte Complianceregeln, die mit Compliancedomains verknüpft werden, verlangen ein anderes Verfahren, um die Einhaltung dieser Regeln zu überprüfen. Das Verfahren und die Grundbausteine solcher datenbasierter Complianceregeln werden im Folgenden erläutert.

Datenbasierte Complianceregeln schränken den Datenfluss in einem Prozessmodell ein. In den nächsten Abschnitten werden Eigenschaften von Kontrollflusskonnektoren und Compliancedomains definiert, die in datenbasierten Complianceregeln Anwendung finden. Diese

Eigenschaften wurden in [SLS+11] vorgestellt. Sie müssen von einer Sprache unterstützt werden, die für die Definition von datenbasierten Complianceregeln eingesetzt werden soll.

Definition 7 (Abstrakter Datentyp)**.**

Nach [OW12] besteht ein abstrakter Datentyp aus „einer oder mehreren, mit üblichen mathematischen Methoden festgelegten Mengen von Objekten und darauf definierten Operationen."

Ein ADT ist demnach ein Tupel $ADT = (W_1..W_n, O_1..O_n)$ *von n Werten W und den mit ihnen assoziierten Operationen O.*

5.3.1. Eigenschaften von Datenflusskonnektoren

Datenflusskonnektoren kopieren Daten zwischen Datenobjekten und Aktivitäten. Ein Datenflusskonnektor ist ein Tupel $D = (q, z, d)$ mit der Datenquelle q, dem Datenziel z und dem Datentyp d der zu kopierenden Daten. Es gilt $q \in A$ mit A als der Menge aller Aktivitäten in einem Prozess und $z \in A$.

5.3.2. Eigenschaften von Compliancedomains

Definition 8 (Ein- und Ausgabedatentyp einer Compliancedomain)**.**

Der Eingabedatentyp einer Compliancedomain CD ist ein Tupel $Eingabe_{CD} = (W_1...W_m, O_1...O_k)$ *von Werten* W_m *und Operationen* O_k*.*

Analog wird mit $Ausgabe_{CD}$ *der Datentyp bezeichnet, den die Daten haben müssen, die aus einer Compliancedomain CD heraus kopiert werden.*

Daten werden zum Beispiel unter Verwendung von Datenflusskonnektoren zwischen Compliancedomains ausgetauscht. Dies kann bei-

spielsweise durch Zuweisungen von Daten zwischen einer Aktivität A und einem Datenobjekt B geschehen. Wobei sich die Aktivität A außerhalb und das Datenobjekt B innerhalb der betreffenden Compliancedomain befindet.

Die Zuweisung $Eingabe_{CD} = D$ schränkt beispielsweise den Eingabe-Datentyp $Eingabe_{CD}$ auf den Datentyp D ein.

Komplexere datenbasierte Complianceregeln können durch Konkatenation mit den logischen Operatoren $\wedge$ und $\vee$ erstellt werden. Weiterhin können Formeln mit dem $\neg$ Operator verneint werden und es können Klammern eingesetzt werden. Die zugrundeliegende Sprache ist in Abschnitt 5.5.2 beschreiben.

Die Einhaltung einer solchen Complianceregel wird durch das verwendete graphische Entwicklungswerkzeug durchgesetzt. Wird eine Änderung an den in einer Compliancedomain befindlichen Prozessteilen durchgeführt, so überprüft das graphische Entwicklungswerkzeug diese Änderung auf Übereinstimmung mit allen an die betreffende Compliancedomain angehefteten Complianceregeln.

Ist die Eingabeeigenschaft einer Compliancedomain mittels einer Complianceregel eingeschränkt, so kann vom graphischen Entwicklungswerkzeug automatisch überprüft werden, ob eine Datenassoziation, die Daten von außerhalb einer Compliancedomain in diese Compliancedomain kopiert, zulässig ist. Algorithmus 5.1 zeigt, wie dies technisch umgesetzt werden kann.

Das hier angewendete Verfahren basiert auf dem Vergleich von Datenformaten. Hierbei wird beim Kopieren von Daten untersucht, ob das Datenformat der Quelle mit dem in der Complianceregel definierten Datenformat vereinbar ist. Bei dieser Untersuchung wird mit den Datenformaten gearbeitet, da zur Entwicklungszeit keine Instanzdaten

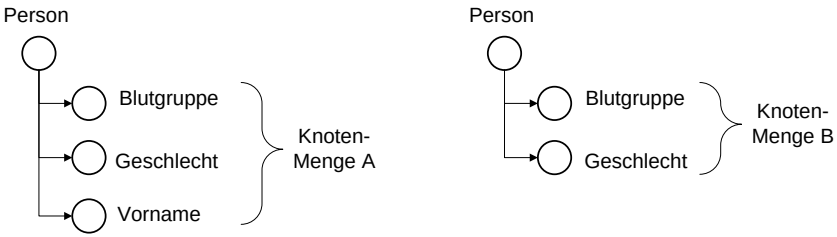

Abbildung 5.3.: Vergleich von Daten-schemas zur Überprüfung einer Complianceregel.

vorhanden sind, die untersucht werden könnten. Die Untersuchung der Datenformate ist für den in Abbildung 5.1 gezeigten Anwendungsfall ausreichend, da man hiermit zum Beispiel verhindern kann, dass bestimmte Daten wie der Name einer Person nicht in eine bestimmte Compliancedomain kopiert werden können. Es wird konkret verglichen, ob das Format der Daten, die innerhalb eines Prozesses kopiert werden, in dieser Compliancedomain verarbeitet werden kann. Dies wird im Folgenden anhand von Abbildung 5.3 näher erläutert. Algorithmus 5.1 beschreibt den Ablauf der Überprüfung der Datenformate.

Abbildung 5.3 zeigt beispielhaft zwei Datentypen, die jeweils Eigenschaften einer Person darstellen. Diese Eigenschaften spielen im Beispielszenario aus Abbildung 5.1 beim Kopieren von Personendaten eine Rolle. In diesem Beispielszenario dürfen in Compliancedomain 0 die in Abbildung 5.3 links dargestellten Daten einer Person verarbeitet werden. Während in Compliancedomain 2 nur die in Abbildung 5.3 rechts dargestellten Daten einer Person verarbeitet werden dürfen.

Wir nehmen an, dass Daten einer Person von Compliancedomain 0 in Compliancedomain 2 kopiert werden sollen. Um Daten zu kopieren

wird in Business Process Model and Notation 1.0 (BPMN 1.0) eine Datenassoziation zwischen diesen beiden Compliancedomains eingefügt. Eine solche Datenassoziation hat drei Eigenschaften, die beim Einfügen in ein Prozessmodell gesetzt werden müssen: Die Datenquelle, das Datenziel und das Format der zu kopierenden Daten. Im oben gezeigten Beispiel ist dieses Datenformat der Datenassoziation durch den in Abbildung 5.3 links dargestellten Baum von Eigenschaften einer Person repräsentiert.

Für die Validierung von Eingabe- und Ausgabeeigenschaften von Compliancedomains sind Validatoren zuständig. Ein solcher Validator wird beim Einfügen einer Datenassoziation in ein Prozessmodell mit der Überprüfung von Eingabe- und Ausgabeeigenschaften der betreffenden Compliancedomains betraut.

Dieser Validator vergleicht nur die beiden in Abbildung 5.3 dargestellten Eigenschaften einer Person und deckt auf, dass diese nicht übereinstimmen. Der menschliche Prozessdesigner, der die Datenassoziation in das betreffende Prozessmodell eingefügt hat, kann darauf aufmerksam gemacht werden. Eine Maßnahme, um die Complianceregel von Compliancedomain 2 einzuhalten, besteht darin, das Datenformat der in Compliancedomain 2 zu kopierenden Daten an das gültige Datenformat von Compliancedomain 2 anzupassen. Am Beispiel von Abbildung 5.3 müsste also die Eigenschaft *Vorname* aus dem linken Datenformat entfernt werden.

Algorithmus 5.1 zeigt in Pseudo-Code, wie der oben beschriebene Vergleich von Datenformaten implementiert wurde. Der Algorithmus wurde für die Verarbeitung von BPMN 1.0 Prozessmodellen entwickelt. Die in diesem Algorithmus gezeigte Funktion *checkComplianceDomain* dient dazu, von einem graphischen Entwicklungswerkzeug aufgerufen

Algorithmus 5.1 Überprüfung von Compliancedomains

```
1: function checkComplianceDomain(BpmnModel complDomain,
   DataSchema complRule)
2:     DA relDataAssocs = getRelevantDataAssocs(complDomain);
3:     for DA currDataAssoc in relDataAssocs do
4:         return checkDataAssoc(currDataAssoc, complRule);
5:     end for
6: end function
7: function checkDataAssoc(DA currDataAssoc, DataSchema com-
   plRule)
8:     DataSchema dataToBeCopied = currDataAs-
   soc.getDataSchema();
9:     return DataSchema result = checkComplRu-
   le(dataToBeCopied, complRule);
10: end function
```

zu werden. Der Algorithmus kann in mehreren Situationen aufgerufen werden. Beispielsweise nach einer bestimmten Anzahl von Änderungen oder nach einer bestimmten Zeit, in der das Prozessmodell bearbeitet wird. Die Eingabeparameter sind von links nach rechts:

- *bpmnProcModel*: Das betreffende Prozessmodell, welches durch das Einfügen einer Datenassoziation geändert wurde.
- *complDomain*: Der Teil des Prozessmodells, der sich in der betreffenden Compliancedomain befindet, in der Änderungen vorgenommen worden sind.
- *complRule*: Beschreibt den Datentyp, der in der betreffenden Compliancedomain verarbeitet werden darf.

In Zeile 2 werden die relevanten Datenassoziationen gesammelt. Da-

tenassoziationen sind dann relevant für den Algorithmus, wenn sie die Grenze der betreffenden Compliancedomain überschreiten. In den Zeilen 3 bis 5 werden alle relevanten Datenassoziationen auf Unvereinbarkeit mit der übergebenen Compliancerегel untersucht, wobei die Funktion *checkDataAssoc* aufgerufen wird. In dieser Funktion findet in den Zeilen 7 bis 10 die eigentliche Überprüfung der betreffenden Compliancerегel statt. Hierbei wird aus der übergebenen Datenassoziation das Datenformat der zu kopierenden Daten ausgelesen und der nicht aufgeführten Funktion *checkComplRule* übergeben. Diese Funktion vergleicht die Datenformate der Compliancerегel und der relevanten Datenassoziation miteinander. Das Ergebnis dieses Vergleichs ist entweder ein leeres Objekt, sofern die betreffende Compliancerегel nicht verletzt wurde oder aber ein Datenschema, das die Elemente enthält, die nicht übereingestimmt haben.

5.4. Datenflussanalyse im Feld der Compilerentwicklung

Datenflussanalyse ist im Compilerbau ein Mittel, um den Datenfluss möglicher Programmausführungspfade zu untersuchen [ASU86]. Die Ergebnisse werden dazu verwendet, Programme zur Compilezeit zu optimieren. Die Optimierungen zielen auf eine Beschleunigung der Laufzeit oder des Speicherverbrauchs eines Programms ab. Im Gegensatz zu der in dieser Arbeit vorgestellten Datenflussanalyse, wird bei der Datenflussanalyse im Compilerbau die Ausführung eines Programms simuliert. Ein Ergebnis einer solchen Simulation ist die Aufdeckung von toten Pfaden in einem Programm und die Vorhersage von Wertebereichen, die die Variablen eines untersuchten Programms bei der Ausführung annehmen.

Diese Informationen sind für die Erreichung des Ziels der Unterstützung von menschlichen Prozessmodellierern bei der Entwicklung von regelkonformen Geschäftsprozessen nebensächlich. Man könnte die Wertebereiche, die Variablen während der Ausführung einnehmen, zur Überprüfung von Complianceregeln heranziehen. So könnte zum Beispiel eine Complianceregel überprüft werden, die festlegt, dass eine bestimmte Variable in einem Prozess nur einen bestimmten Wert annehmen darf.

Es gibt jedoch einige Argumente, die dagegen sprechen. Im Compilerbau werden erstens Datenflussanalysen nur für bestimmte Eingaben durchgeführt. Für die Überprüfung von datenbasierten Complianceregeln wäre es unerlässlich, Datenflussanalysen für den gesamten Eingaberaum eines Prozesses durchzuführen. Zweitens müsste die gesamte Datenflussanalyse erneut berechnet werden, würde das betreffende Prozessmodell an einer beliebigen Stelle geändert. Eine solch zeitaufwändige Berechnung ist bei dem in diesem Kapitel vorgestellten Mechanismus zur Überprüfung von datenbasierten Complianceregeln nicht notwendig. Bei einer Änderung müssen hier lediglich die Complianceregeln überprüft werden, die mit der Compliancedomain verknüpft sind, in der die Änderung vorgenommen wurde.

Im folgenden Abschnitt wird die Zusammenführung kontrollflussbasierter und datenbasierter Complianceregeln gezeigt. Complianceregeln können einerseits einen Teil, der den Kontrollfluss in einem Prozess einschränkt und andererseits einen Teil, der den Datenfluss einschränkt beinhalten [ETHP10]. Ein Beispiel für eine solche Complianceregel ist die Einhaltung des Vier-Augen-Prinzips. Eine Complianceregel, die dieses Prinzip beschreibt, muss sicherstellen, dass ein Dokument (Daten-Teil) von mindestens zwei verschiedenen Personen

untersucht wird. Die Untersuchung des Dokuments von den beiden Personen kann dabei parallel oder sequenziell ablaufen (Kontrollfluss-Teil).

5.5. Kombination von datenfluss- mit kontrollflussbasierten Complianceregeln

Die Forschung im Bereich der Compliance von Prozessen begann mit der Untersuchung des Kontrollflusses in Prozessen. Turetken et. al zeigen in [TEHP11] einige Regelmuster, wie sie häufiger bei der praktischen Auseinandersetzung mit Complianceregeln im Prozessbereich auftreten. Die in dem Artikel gezeigten Complianceregeln zielen ausschließlich auf den Kontrollfluss eines Prozesses ab. Ein Grund für die anfängliche Ausblendung des Datenflusses bei der Untersuchung von Complianceregeln für Prozesse mag die einfachere Handhabbarkeit des Kontrollflusses im Gegensatz zum Datenfluss in einem Prozess gewesen sein. Für die Erstellung von Complianceregeln, die den Datenfluss eines Prozesses einschränken, gibt es Sprachen wie die Lineare Temporale Logik (LTL), die weit verbreitet Anwendung finden.

Dieser Abschnitt zeigt die Notwendigkeit der Verbindung von kontrollflussbasierten mit datenbasierten Complianceregeln und präsentiert eine generische Compliancesprache (Abschnitt 5.5.1), die es ermöglicht diese beiden Complianceregeltypen in einem Ausdruck zusammenzufassen. Jeder Teilausdruck eines zusammengefassten Ausdrucks kann von einem dafür vorgesehen Plugin verifiziert werden. Weiterhin wird ein Algorithmus präsentiert (Abschnitt 5.5.4), der es ermöglicht, kombinierte Complianceregeln, die mit der generischen Compliancesprache erstellt wurden, automatisch auszuwerten.

Das Verständnis des Datenflusses ist in den oben genannten Veröffentlichungen jedoch ein anderes als im Folgenden. In diesen Veröffentlichungen wird der Datenfluss aus der den Kontrollfluss beeinflussenden Richtung gesehen. Es kann anhand der in einem Prozessmodell vorhandenen Daten berechnet werden, welche Teile eines Prozesses ausgeführt werden, da die vorhandenen Daten oft Kontrollflussentscheidungen in einem Prozessmodell beeinflussen. Dies geschieht zum Beispiel bei einer If-Verzweigung. Hier wird eine Bedingung ausgewertet. Dies kann unter Zuhilfenahme beliebiger Daten im Prozess geschehen. Mit diesen Arbeiten kann also gezeigt werden, welche Bereiche eines Prozesses bei welchen Eingabedaten ausgeführt werden. Dieses Ergebnis macht eine Aussage über den Kontrollfluss eines Geschäftsprozesses und bringt wenig Klarheit über den Datenfluss. Die Untersuchung, welche Teile eines Programms bei welcher Eingabe ausgeführt werden, ist im Bereich des Compilerbaus gut erforscht und aufgearbeitet [ASU86].

In der vorliegenden Arbeit wird deshalb nicht auf diese Art der Abhängigkeit zwischen dem Kontrollfluss und dem Datenfluss in einem Prozess eingegangen. Vielmehr werden Complianceregeln untersucht, die ihre Relevanz aus dem Bereich der verteilten Ausführung von Prozessen haben. Einen Ansatz Prozesse aufzuteilen und in einer verteilten Umgebung auszuführen findet man in [KL06]. Die in dieser Arbeit vorgestellten datenbasierten Complianceregeln beruhen auf der Tatsache, dass es bei einer Aufteilung eines Prozesses wichtig ist, welche Daten zwischen den Teilen des Prozesses fließen. Mit Complianceregeln kann der Datenfluss zwischen bestimmten Bereichen in einem aufgeteilten Prozess eingeschränkt werden.

Ein Anwendungsfall für die Aufteilung eines Prozesses ist der Ein-

stieg eines Unternehmens oder einer Organisation in die Verwendung des Cloud-Computing. Cloud-Computing wird für Unternehmen nicht nur wegen der großen Einsparpotentiale, sondern auch aufgrund besserer Möglichkeiten der Skalierbarkeit von Anwendungen immer interessanter. Die Verwendung von Public-Clouds verspricht hierbei das größte Einsparpotential, da Anbieter von Public-Clouds dieselben Ressourcen für verschiedene Kunden wiederverwenden und sie somit optimal auslasten können. Dies führt zu einem Kostenvorteil, den die Betreiber von Public-Clouds an ihre Kunden weitergeben können. Hat sich ein Unternehmen für die Verwendung einer Public-Cloud entschieden, gilt es die Frage zu beantworten, welche Teile der Prozesse in der Public-Cloud ausgeführt werden sollen. Im Hinblick auf rechtliche Vorgaben sind einige Dinge zwingend zu beachten. So muss ein Unternehmen beispielsweise darauf achten, dass Daten, die an einen Public-Cloud Provider übertragen werden, in einem passenden rechtlichen Rahmen auch beim Provider weiterverarbeitet werden.

Falls eine Firma die Kostenvorteile einer Public-Cloud nutzen und dennoch bestimmte Daten schützen möchte, ist eine Hybrid-Cloud das richtige Mittel: Hierbei wird der Teil eines Prozesses, der zum Beispiel sensible Daten verarbeitet in der Private-Cloud eines Unternehmens ausgeführt, während der Teil eines Prozesses, der unkritische Daten verarbeitet, in einer Public-Cloud ausgeführt werden kann. Für die Einteilung eines Prozesses in kritische und unkritische Bereiche ist das Konzept der Compliancedomain anwendbar.

Wie die oben aufgeführten Beispiele zeigen, beschäftigt sich die Literatur entweder mit kontrollflussbasierten oder mit datenbasierten Complianceregeln. Es ist jedoch gerade im Bereich des Geschäftsprozessmanagements wichtig, kontrollflussbasierte mit datenbasierten

Complianceregeln zu verbinden. Dies wird am folgenden Beispiel einer kombinierten Complianceregel deutlich.

Beispiel 1 (Kombinierte Complianceregel)**.** *„Die gesammelten Blutproben sollen an zwei unabhängigen Stellen A und B überprüft werden. Danach sollen die Ergebnisse von A und B von einer weiteren unabhängigen Stelle C verglichen werden. Weichen die Ergebnisse von A und B voneinander ab, so kann die betreffende Blutprobe nicht weiterverarbeitet werden.“*

Mit dieser Complianceregel wird erstens festgelegt, dass zwei Aktivitäten A und B vor Aktivität C ausgeführt werden. Dies ist folglich eine kontrollflussbasierte Complianceregel. Zweitens wird deutlich, dass Aktivität C mit den Ergebnissen der Aktivitäten A und B arbeiten muss. Arbeitet Aktivität C mit anderen Daten, wäre die Complianceregel nicht erfüllt.

Nun wird diese Complianceregel auf das in dieser Arbeit durchgängig verwendete Prozessbeispiel angewendet, welches mit BPMN Pools versehen wurde, um Verantwortlichkeiten für die im Prozess enthaltenen Aktivitäten zuzuweisen. Abbildung 5.4 zeigt den Prozess, auf den die oben vorgestellte kombinierte Complianceregel angewendet werden soll. Um die in Beispiel 1 gezeigte Complianceregel auf dieses Prozessmodell anzuwenden, müssen die Platzhalter A, B und C mit Aktivitäten in diesem Prozess verknüpft werden. Die Verknüpfungen sehen wie folgt aus:

- Platzhalter A ist verknüpft mit der Aktivität *Blutprobe Überprüfen* im Pool der Blutspendestation.

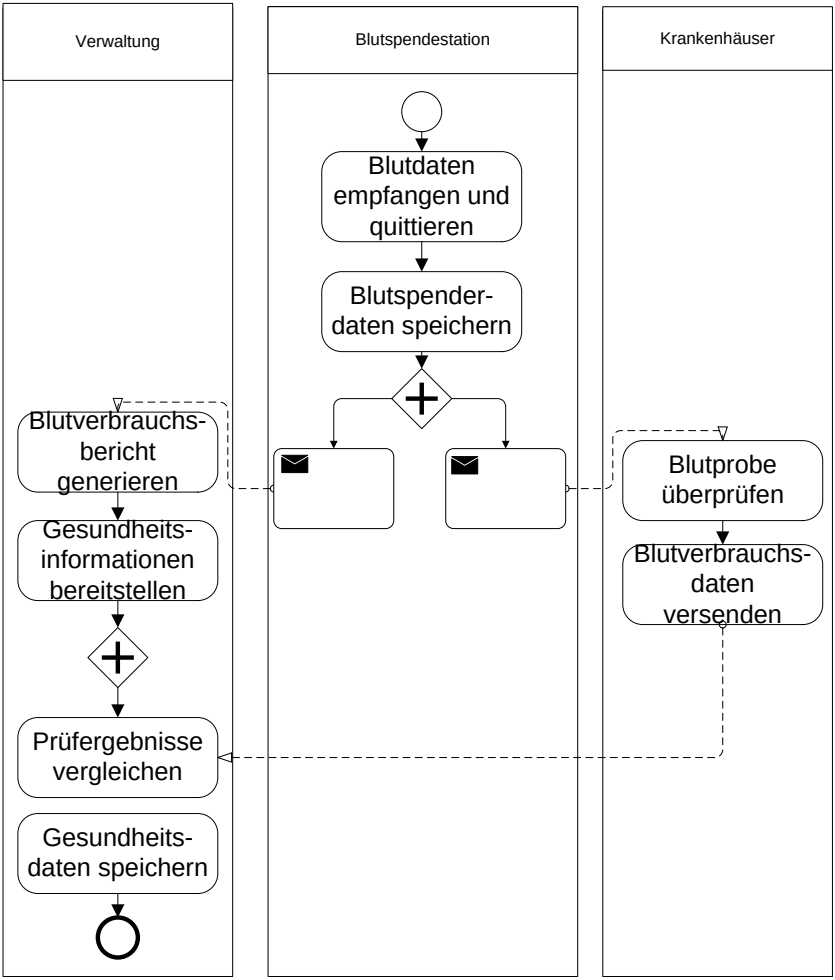

Abbildung 5.4.: Beispielprozess, der den gesamten Kontrollfluss und einen Teil des Datenflusses zeigt.

- Platzhalter B ist verknüpft mit der Aktivität *Blutprobe Überprüfen* im Pool der Krankenhäuser.
- Platzhalter C ist verknüpft mit der Aktivität *Prüfergebnisse vergleichen* im Pool der Verwaltung.

5.5.1. Generische Compliancesprache für die Kombination von datenbasierten mit kontrollflussbasierten Complianceregeln

Die in Kapitel 4 und in diesem Kapitel gezeigten zwei Arten von Complianceregeln wurden in der Literatur getrennt voneinander behandelt. Diese beiden Arten sind kontrollflussbasierte und datenflussbasierte Complianceregeln. Der Bedarf für eine übergeordnete Compliancesprache, mit der Complianceregeln ausgedrückt werden können, die aus den zwei Complianceregelarten aufgebaut sind, wird in [HWG09] aufgezeigt. Hier wird argumentiert, dass formale Sprachen sich für bestimmte Zwecke eignen und für andere weniger. Diesen Bedarf greift die vorliegende Arbeit auf und zeigt, wie Teile von Complianceregeln, die in beliebigen formalen Sprachen geschrieben sind, zu einer Complianceregel kombiniert werden können. Dieser Abschnitt verfolgt bei der Definition der übergeordneten Compliancesprache einen pragmatischen Ansatz, indem er zeigt, wie die Aussagenlogik erweitert werden kann, um dieser Anforderung gerecht zu werden.

Die Beschreibung einer Sprache, die diese Anforderung erfüllt, ist das Ziel dieses Abschnitts. Abbildung 5.5 zeigt die Kombination von Beispielsprachen mittels einer übergeordneten generischen Compliancesprache. Sowohl Lineare Temporale Logik (siehe Abschnitt 4.1) als auch XPath [BBC+07] können für die Definition von Complianceregeln verwendet werden. Mit LTL können kontrollflussbasierte Compliance-

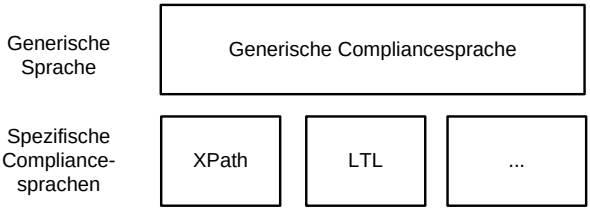

Abbildung 5.5.: Generische Compliancesprache (vergleiche:[SWLS10])

regeln entworfen werden. XPath wird im Prototyp der vorliegenden Dissertation als Beschreibungssprache für datenbasierte Complianceregeln eingesetzt.

Begonnen wird mit der Beschreibung der einzelnen Schritte, die notwendig sind, um eine solche übergeordnete Compliancesprache zu erstellen. Eine Eigenschaft dieser Sprache soll sein, dass beliebige Beschreibungssprachen für logische Zusammenhänge mit ihr verwendet werden können. Weiterhin soll die Sprache auf einer weit verbreiteten Grundsprache aufbauen, um so das Erlernen dieser Sprache zu erleichtern. Ausdrücke, die in dieser Sprache geschrieben sind, sollen somit für Personen mit Vorwissen auf dem Gebiet logischer Sprachen intuitiv verständlich sein. Diese Eigenschaften dienen dem Ziel der vorliegenden Dissertation: der Erleichterung für den Menschen regelkonforme Prozesse zu erstellen.

5.5.2. Formale Definition einer generischen Compliancesprache

Dieser Abschnitt stellt die formale Definition der generischen Compliancesprache bereit. Abbildung 5.6 zeigt eine Erweiterung der Backus-

⟨Satz⟩ → ⟨AtomarerSatz⟩ | ⟨Satz⟩ ⟨Verbinder⟩ ⟨Satz⟩ | ¬⟨Satz⟩

⟨AtomarerSatz⟩ → ⟨Term⟩ | ⟨Term⟩ = ⟨Term⟩

⟨Term⟩ → ⟨**Sprachbezeichner**⟩"–"⟨**BeliebigsprachigerTerm**⟩

⟨Verbinder⟩ → ⇒ | ∧ | ∨ | ⇔

⟨**Sprachbezeichner**⟩ → **LTL** | **XPath** | **...**

⟨**BeliebigsprachigerTerm**⟩ → **Sprachausdruck**

Abbildung 5.6.: Übergeordnete Sprache zur Definition von Complianceregeln (Erweiterung der BNF der Aussagenlogik)

Naur-Form (BNF) der Aussagenlogik. Die hier gezeigte BNF der Aussagenlogik wurde mit dem Ziel erweitert, die in Abschnitt 5.5.1 aufgeführten Eigenschaften einer generischen Compliancesprache in diese Sprache einzuführen. Die formale Semantik der Aussagenlogik wird dabei beibehalten. Mit dieser Erweiterung ist es möglich, aussagenlogische Ausdrücke zu erweitern, dass Ausdrücke, die in verschiedenen Sprachen geschrieben sind, in einen Ausdruck eingebettet werden. Die Aussagenlogik wurde verwendet, weil sie eine der bekanntesten Sprachen für die Erstellung logischer Ausdrücke ist. Sie bietet sich daher als Basis für eine Erweiterung an. Durch die Verwendung der

Aussagenlogik können Modellierer von Complianceregeln ihre Kenntnisse über die Aussagenlogik anwenden und mit dem Wissen über die Erstellung von Complianceregeln verknüpfen.

Die BNF der Aussagenlogik wurde an zwei Stellen erweitert. Die erste Stelle, an der eine Erweiterung vorgenommen wurde, ist die Produktionsregel für das ⟨Term⟩-Nichtterminal. Terme gehen in zwei Nichtterminale über, von denen das Erste ein Sprachbezeichner sein muss und das zweite einen Ausdruck in einer „fremden“ Sprache beschreibt. Anhand der Auswertung dieses Sprachbezeichners kann dann zum Beispiel eine grafische Entwicklungsumgebung ein bestimmtes Programm zur Auswertung des nachfolgenden Ausdrucks heranziehen.

Die zweite Änderung an der BNF der Aussagenlogik wurde in der vorletzten Zeile der in Abbildung 5.6 gezeigten BNF eingeführt. Hier wird der weiter oben eingeführte Sprachbezeichner spezifiziert. Als Beispiele für Sprachbezeichner sind LTL und XPath aufgeführt.

Die dritte Erweiterung ist in der letzten Zeile zu sehen. Hier wurde ebenfalls eine neue Produktionsregel eingeführt, die dazu dient, das Nichtterminal ⟨BeliebigsprachigerTerm⟩ in ein Terminal zu überführen, das die Spezifikation eines Ausdrucks in einer beliebigen Sprache ermöglicht.

5.5.3. Beispiele

Die in Beispiel 1 gezeigte Complianceregel besteht aus zwei Teilen, dem Kontrollflussteil und dem Datenflussteil. Für jeden dieser Teile muss eine andere Sprache zur Beschreibung verwendet werden. Es ist nicht möglich, beide Arten von Complianceregeln mit einer existierenden Sprache auszudrücken, da die Anwendungsgebiete zu

unterschiedlich sind. Dies wird deutlich, wirft man einen Blick in die in Kapitel 4 und in diesem Kapitel vorgestellten Compliancerегeln. Im Folgenden wird gezeigt, wie eine solche kombinierte Compliancerегel von automatischen Werkzeugen ausgewertet wird.

Beispiel 2 (Verknüpfung von Teilen einer Compliancerегel)**.** *Eine Compliancerегel R wird durch Verknüpfung logischer Ausdrücke A, B und C mit Konjunktionen erstellt. Die Ausdrücke A, B und C können in verschiedenen logischen Sprachen geschrieben sein.*

$R = LTL - A \ \wedge \ XPath - B \ \wedge \ LTL - C$

Beispiel 2 zeigt, wie die Ausdrücke A, B und C mit einer Konjunktion verbunden werden können. Vor jedem dieser Ausdrücke steht ein beispielhafter Sprachbezeichner. Prinzipiell können beliebig viele Ausdrücke auf diese Weise miteinander verbunden werden. Diese Ausdrücke können in einer beliebigen logischen Sprache geschrieben sein. Konjunktionen werden zum Beispiel in der Aussagenlogik verwendet. Die Verifizierung eines verbundenen Ausdrucks wird dann von einem Programm angestoßen, das den umgebenden prädikatenlogischen Ausdruck analysiert und für die darin enthaltenen Ausdrücke in anderen Sprachen weitere Programme (Plugins) zur Verifizierung aufruft.

In Kapitel 4 wurde Lineare Temporale Logik als Beschreibungssprache für kontrollflussbasierte Compliancerегeln verwendet, während in dem vorliegenden Kapitel gezeigt wurde, wie datenflussbasierte Compliancerегeln spezifiziert werden können. Es wird nun beispielhaft gezeigt, wie diese Sprachen für die Definition einer kombinierten Compliancerегel verwendet werden können. Dazu wird die Beispielcompliancerегel aus Beispiel 1 herangezogen. Wie oben beschrieben, besagt der kontrollflussbasierte Teil dieser Compliancerегel, dass die

Aktivitäten A und B vor der Aktivität C ausgeführt werden sollen. Ausgedrückt in Linearer Temporaler Logik (LTL) ergibt dies diesen Ausdruck:

$$\Box((\Diamond A \wedge \Diamond B) \Rightarrow \Diamond C) \tag{5.1}$$

Die Complianceregel stellt sicher, dass im gesamten Prozess (global) gilt, dass letztendlich (finally) C ausgeführt wird, wenn A und B ausgeführt wurden. Es ist hierbei unerheblich, in welcher Reihenfolge A und B ausgeführt werden. Die Aktivitäten A und B müssen nicht direkt hintereinander ausgeführt werden. Sie können zum Beispiel auch parallel ausgeführt werden.

Der datenbasierte Teil der in Beispiel 1 gezeigten Complianceregel ist in diesem Beispiel nicht explizit beschrieben. Dies ist bei einer real existierenden Complianceregel meist der Fall. Implizit muss jedoch angenommen werden, dass die Ergebnisse der Aktivitäten A und B als Eingabe für Aktivität C dienen. Der datenbasierte Teil dieser Complianceregel würde folglich mit den in Abschnitt 5.3 vorgestellten datenbasierten Eigenschaften von Datenobjekten und Aktivitäten wie in Gleichung 5.2 aussehen:

$$Eingabe_C = Ausgabe_A \wedge Ausgabe_B \tag{5.2}$$

Kombiniert man nun diese beiden Regeltypen mittels einer Konjunktion, so erhält man nachstehende Formel. Der Sprachbezeichner für

den datenbasierten Teil ist *data*.

$$\overbrace{data - (Eingabe_C = Ausgabe_A \wedge Ausgabe_B)}^{DatenbasierterTeil} \wedge \overbrace{LTL - (\square((\Diamond A \wedge \Diamond B) \Rightarrow \Diamond C))}^{KontrollflussbasierterTeil} \tag{5.3}$$

Die beiden Formelteile werden mittels einer Konjunktion verbunden, da sowohl der datenbasierte als auch der kontrollflussbasierte Teil der Complianceregel wahr sein müssen, damit der Wahrheitswert der gesamten Complianceregel *wahr* ist. Für die beiden in Gleichung 5.3 markierten Teile sind dann bei der Überprüfung der gesamten Complianceregel jeweils verschiedene Programme zuständig.

5.5.4. Automatische Überprüfung von Ausdrücken in einer generischen Compliancesprache, die aus verschiedensprachigen Ausdrücken aufgebaut sind

Das Ziel der Zusammenführung verschiedener Arten von Complianceregeln in einem logischen Ausdruck besteht in der Möglichkeit solche Ausdrücke automatisch überprüfbar zu machen. Die Grundlage des weiter unten präsentierten Algorithmus bildet das Konzept des Regelbaumes [Gro11]. Regelbäume sind Binärbäume, deren innere Knoten aussagenlogische Operatoren, wie beispielsweise die Konjunktion ($\wedge$), sind. Die Blätter von Regelbäumen repräsentieren Complianceregeln, die in beliebigen Sprachen geschrieben sein können. Wenn für eine solche in einem Blatt des Regelbaums verwendete Sprache ein Werkzeug zur Verarbeitung eines Ausdrucks dieser Sprache existiert, so kann diese Teilcomplianceregel automatisch überprüft werden.

Das Konzept wird in dieser Arbeit dazu verwendet, kombinierte Complianceregeln graphisch darzustellen, um die einzelnen Schritte des Überprüfungsalgorithmus zu erklären. Regelbäume sind ein Hilfsmittel, um Ausdrucke, die in der generischen Compliancesprache geschrieben sind, besser verständlich darzustellen.

Definition 9 (Regelbaum)**.** *Ein Regelbaum* $R = (\mathcal{K}_\mathcal{I}, \mathcal{K}_\mathcal{B}, \mathcal{K}_\mathcal{H})$ *mit* $\mathcal{K}_\mathcal{I}$ *als der Menge der inneren Knoten,* $\mathcal{K}_\mathcal{B}$ *als der Menge der Blattknoten (innere Knoten mit Grad 2) und* $\mathcal{K}_\mathcal{H}$ *als der Menge der Halbblatt-Knoten ist ein Binärbaum. Der Regelbaum hat folgende Eigenschaften:*

1. $\forall k \in \mathcal{K}_\mathcal{I} \ : \ k \in \mathcal{K}_\wedge \cup \mathcal{K}_\vee \cup \mathcal{K}_=$. $\mathcal{K}_\wedge$ *ist die Menge Knoten, die die Konjunktion repräsentieren,* $\mathcal{K}_\vee$ *ist die Menge der Knoten, die die Disjunktion repräsentieren und* $\mathcal{K}_=$ *ist die Menge der Knoten, die Gleichheit repräsentieren.*

2. $\forall k \in \mathcal{K}_\mathcal{H} \ : \ k \in \mathcal{K}_\neg$. $K_\neg$ *ist die Menge der Knoten, die die Negation repräsentieren.*

3. $\forall k \in \mathcal{K}_\mathcal{B} \ : \ k \in \mathcal{K}_\mathcal{L}$. $K_\mathcal{L}$ *beschreibt die Menge der Knoten, die einen beliebigsprachigen logischen Ausdruck repräsentieren.*

Ein Regelbaum ist ein Binärbaum, dessen innere Knoten aussagenlogische Operatoren repräsentieren. Die Operatoren $\wedge$, $\vee$ und $=$ haben als Knoten in diesem Baum einen Eingang und zwei Ausgänge. Abbildung 5.7 zeigt dies für den $\wedge$-Operator. Der $\neg$-Operator hat einen Eingang und einen Ausgang. Die Blätter dieses Baumes repräsentieren Ausdrücke in beliebigen logischen Sprachen.

Abbildung 5.7 zeigt die in Gleichung 5.3 aufgestellte Complianceregel als Regelbaum.

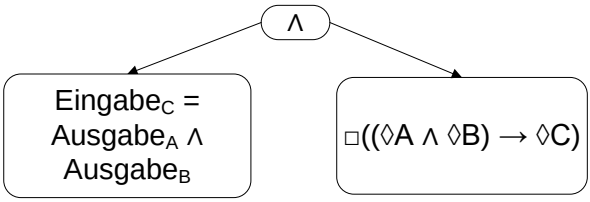

Abbildung 5.7.: Darstellung des Regelbaumes für das laufende Beispiel

Wie die Überprüfung von mit der generischen Compliancesprache beschriebenen Complianceregeln erfolgt, zeigt Algorithmus 5.2. Dieser Algorithmus basiert auf dem Algorithmus der *in-order* Traversierung [OW12] von Binärbäumen. Er ist hier aufgeführt, um zu zeigen wie dieser theoretische Ansatz für die Untersuchung von Complianceproblemen eingesetzt werden kann.

Wie in Zeile eins dargelegt, beschreibt der Algorithmus die Implementierung der Funktion *verify*. Diese Funktion liefert entweder *true* oder *false* zurück. Als Parameter wird ihr ein Regelbaum-Objekt übergeben. Dieses Regelbaum-Objekt ist vom Typ BinaryTree.

In dieser Methode *verify* wird der Regelbaum unter Anwendung einer Tiefensuche durchlaufen. Die Implementierung der Methode *verify* ist in zwei Teile geteilt. Diese Teile sind durch die in Zeile 3 gezeigte If-Anweisung definiert. Wie oben definiert, werden die inneren Knoten eines Regelbaumes auf die logischen Operatoren abgebildet, die die in einem Ausdruck kombinierten Complianceregeln verbinden. Die Blätter des Regelbaumes repräsentieren die einzelnen Complianceregeln, die zu einer großen Complianceregel zusammengefügt worden sind. Die in Zeile 3 gezeigte If-Anweisung überprüft, ob es sich bei

Algorithmus 5.2 Überprüfung eines Regelbaumes (vergleiche [SLS+11])

```
 1: function VERIFY(BinaryTree ruleTree)
 2:     boolean result = false;
 3:     if not ruleTree.isOperator() then
 4:         return ruleTree.getComplianceRule().check();
 5:     else
 6:         if ruleTree.getParent().isNot() then
 7:             return not (verify(ruleTree.getChild());)
 8:         else
 9:             result = verify(ruleTree.getLeft());
10:             if ruleTree.getParent().isAnd() then
11:                 return result and verify(ruleTree.getRight());
12:             end if
13:             if ruleTree.getParent().isOr() then
14:                 return result or verify(ruleTree.getRight());
15:             end if
16:         end if
17:     end if
18:     return false;
19: end function
```

dem aktuell zu untersuchenden Knoten des übergebenen Regelbaumes um einen inneren Konten, also einen Operator, oder um ein Blatt, also eine Compliancerегel handelt. Handelt es sich um eine Compliancerегel, so wird in Zeile 4 die Compliancerегel aus dem aktuellen Knoten überprüft. Für diese Überprüfung wird der oben erwähnte *Sprachbezeichner* ausgewertet, um die Sprache zu bestimmen, mit der diese Compliancerегel geschrieben wurde. Anschließend kann das Programm aufgerufen werden, das Compliancerегeln in dieser Sprache verarbeiten kann.

Jeder Knoten im zu untersuchenden Regelbaum stellt die Methode *getComplianceRule()* bereit. Mit ihr ist es möglich, das Objekt, das eine Complianceregel repräsentiert, die mit dem aktuellen Knoten des Regelbaums verknüpft ist, abzufragen. Alle Complianceregel-Objekte implementieren die Methode *check()*. In dieser Methode wird das Plugin aufgerufen, das mit der Sprache umgehen kann, in der die betreffende Complianceregel geschrieben ist.

Handelt es sich um einen inneren Knoten, der einen Operator repräsentiert, so wird die Ausführung des Algorithmus in Zeile 6 fortgeführt. Zunächst wird hier untersucht, ob es sich um einen *not*-Operator handelt. Ist dies der Fall, so wird die Methode *verify* rekursiv mit dem Kindknoten des aktuellen Knotens aufgerufen. Das Ergebnis wird negiert zurückgegeben. In Zeile 9 wird durch einen rekursiven Aufruf der Methode *verify* begonnen, den linken Teilbaum zu durchlaufen. Die Rekursion stoppt, wenn die Suche an einem Blatt des Regelbaumes angekommen ist. In diesem Fall wird die Anweisung in Zeile vier ausgeführt.

In den Zeilen 10 bis 15 wird für jeden bei der Tiefensuche gefundenen Knoten untersucht, ob dieser einen Operator der Form *and* oder *or* repräsentiert. Basierend auf dem Ergebnis dieser Untersuchung wird in den Zeilen 11 und 14 das Ergebnis der Funktion berechnet. In dieser Phase des Algorithmus ist der Wahrheitswert für den linken Teilbaum bekannt. Dieser wurde in Zeile 9 berechnet. Ist der aktuelle Knoten zum Beispiel ein *and*-Operator, so wird der Wahrheitswert des linken Teilbaumes mit dem Wahrheitswert des rechten Teilbaumes konjugiert und als Ergebnis des Algorithmus zurückgegeben.

Tritt keiner der in der Methode *verify* behandelten Fälle ein, so wird der Wahrheitswert *false* zurückgeliefert. Der Algorithmus muss

hier aber noch nicht beendet werden, da ein anderer Teilbaum den Wahrheitswert *true* haben könnte. Der Algorithmus kann zum Beispiel fortgeführt werden, wenn zwei Teilbäume des Regelbaumes mit einer Disjunktion verbunden sind. Hier muss nur einer der beiden Teilbäume den Wahrheitswert *true* aufweisen. Das heißt, die Auswertung der Complianceregeln in nur einem Teilbaum muss den Wahrheitswert *true* ergeben, um der Auswertung des gesamten Regelbaumes den Wahrheitswert *true* zu geben.

Sind alle Teilbäume des Regelbaums durchlaufen, kann das Ergebnis der Complianceuntersuchung ausgegeben werden. Dieses Ergebnis lautet entweder *true*, sofern die durch den Regelbaum repräsentierte Complianceregel durch das zugrunde liegende Prozessmodell erfüllt ist, oder es lautet *false*, wenn dies nicht der Fall ist.

5.6. Zusammenfassung

In diesem Kapitel wurde aufgezeigt, dass bei der Definition von Complianceregeln auch der Datenfluss in einem Prozessmodell eine Rolle spielt. Anhand des in dieser Arbeit durchgehend verwendeten Beispiels wurde diese Behauptung untermauert. Um datenflussbasierte Complianceregeln für menschliche Prozessmodellierer leichter handhabbar zu machen, wurde das Konzept einer Compliancedomain vorgestellt. Mit diesem Konzept ist es möglich, Bereiche in einem Prozessmodell zu markieren. Für jeden dieser Bereiche können datenbasierte Complianceregeln definiert und mit ihnen verknüpft werden. Die mit einer Compliancedomain verknüpften Complianceregeln gelten für alle in ihr enthaltenen Prozesskonstrukte. Sie dienen dazu, den Datenfluss in einem Prozessmodell einzuschränken. Weiterhin können

Compliancedomains mit Ausführungsumgebungen, wie zum Beispiel einer Public-Cloud, verknüpft werden. Somit kann ein Prozess an den Grenzen der Compliancedomains mit den in [KL06] vorgestellten Mechanismen zerteilt und auf den durch die Compliancedomains zugewiesenen Ausführungsumgebungen ausgeführt werden. Die mit den Compliancedomains verknüpften Complianceregeln stehen in direktem Zusammenhang mit den mit ihnen verknüpften Ausführungsumgebungen. Manche Ausführungsumgebungen, wie zum Beispiel eine Public-Cloud, erfordern eine restriktivere Handhabung des Datenflusses als beispielsweise ein privates Rechenzentrum. So könnte der Versand von personenbezogenen Daten in eine Public-Cloud von einer Firma verboten worden sein. Dies könnte in einer entsprechenden Complianceregel umgesetzt sein. Neben einer formalen Definition von Compliancedomains wurde ein Algorithmus erläutert, der zeigt, wie datenbasierte Complianceregeln überprüft werden können.

Dieser Abschnitt definiert eine generische Compliancesprache, die es erstmals ermöglicht, unterschiedliche Sprachen für die Spezifizierung einer Complianceregel zu verwenden. Diese generische Compliancesprache basiert auf der weit verbreiteten Aussagenlogik und erweitert diese, um andere Sprachen in aussagenlogische Ausdrücke einzubetten. Die einzelnen Teile eines solchen Ausdrucks werden auf Grundlage der mit dem Ausdruck verknüpften Compliancedomain ausgewertet. Für jeden dieser Teile des Ausdrucks wird ein Wahrheitswert ermittelt. Die Kombination der einzelnen Wahrheitswerte ergibt den Wahrheitswert des Gesamtausdrucks.

Eine leicht verständliche Darstellung für Ausdrücke dieser generischen Compliancesprache sind Regelbäume. Sie werden dazu verwendet, den als letzten Punkt in diesem Abschnitt vorgestellten Untersu-

chungsalgorithmus für die generische Compliancesprache zu erklären.

KAPITEL 6

GEMEINSAME ERSTELLUNG REGELKONFORMER PROZESSE

Die in dieser Arbeit vorgestellten Konzepte dienen der Unterstützung von menschlichen Prozessmodellierern bei der Erstellung regelkonformer Prozesse. Lösungen, um dieses Ziel zu erreichen, wurden in den vorhergehenden Kapiteln gezeigt. Doch nur durch eine genau festgelegte Abfolge von Schritten zur Erstellung von Prozessen, die nicht umgangen werden kann, lässt sich die Erstellung von nicht regelkonformen Prozessen vermeiden. Die in den vorhergehenden Kapiteln gezeigten Lösungen lassen sich in diese Abfolge von Schritten einbinden.

Dieses Kapitel befasst sich mit der Methodik, die bei der Erstellung regelkonformer Prozesse in Organisationen umgesetzt werden muss. Es zeigt in Abschnitt 6.1 den Prozess, der die Arbeit mit mehreren Part-

nern an einem Prozess regelt. Danach wird in Abschnitt 6.2 gezeigt, welche Werkzeuge notwendig sind, um diesen Prozess bestmöglich zu unterstützen und um eine Umgehung unmöglich zu machen. Der Hauptbeitrag dieses Kapitels, das Konzept der Vervollständigungsebenen, wird in Abschnitt 6.3 ausführlich erläutert.

Die im Folgenden vorgestellten Abläufe müssen in einer Organisation installiert und durchgesetzt werden.

6.1. Erstellung regelkonformer Prozesse unter Beteiligung mehrerer Partner

Abbildung 6.1 zeigt, wie die in den vorangegangenen Kapiteln vorgestellten Konzepte, Compliancetemplate und Compliancescope, in einen Prozess zur Erstellung regelkonformer Prozesse eingebunden sind. Der Prozess ist mit Business Process Model and Notation (BPMN) beschrieben. Die Abbildung zeigt eine Zusammenarbeit zwischen drei Organisationen. *Compliancetemplate Lieferant*, einem Unternehmen, das sich auf Beratung im Bereich Compliance von Prozessen spezialisiert hat (*Complianceberatung*) und ein *Kunde* mit Bedarf an Lösungen zur Erstellung von regelkonformen Prozessen.

Der in Abbildung 6.1 gezeigte Prozess beginnt mit der Erkenntnis einer Organisation, dass ein neuer Prozess für eine bestimmte Aufgabe erstellt werden muss. Es wird davon ausgegangen, dass bei dem Hersteller von Compliancelösungen für Prozesse der Bedarf für ein neues Compliancetemplate erkannt wird. Dieser Bedarf kann zum Beispiel durch einen Kunden ausgelöst werden. Das laufende Beispiel dieser Arbeit aus Abbildung 4.1 aufgreifend, könnte die Krankenhausverwaltung des Roten Kreuzes von Hong Kong dieser Kunde sein.

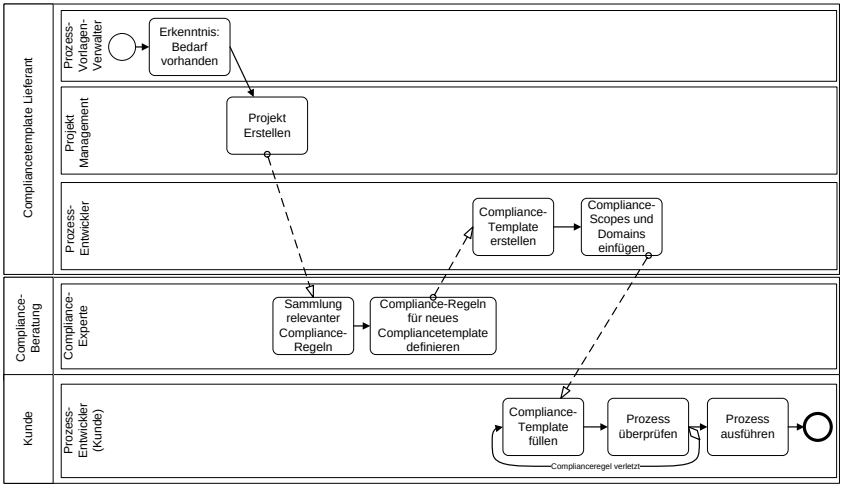

Abbildung 6.1.: Ablauf der Erstellung regelkonformer Prozesse. Notation: angelehnt an BPMN

Der Bedarf könnte hier darin bestanden haben, die Schritte, die bei einer Blutabnahme durchzuführen sind, zu standardisieren. Weiterhin könnte die verbesserte Dokumentation von Blutentnahmen eine Anforderung gewesen sein. Um die Komplexität von Abbildung 6.1 in Grenzen zu halten, wird das Auslösen des Bedarfs nicht gezeigt. Die Anwendung der Methodik verlangt die Definition verschiedener Rollen. Diese Rollen können von Teilbereichen oder Einzelpersonen in einer Organisation ausgefüllt werden. Die folgende Liste erklärt die in Abbildung 6.1 verwendeten Rollen.

- **Verantwortlicher für Compliancetemplates:** Ein Inhaber dieser Rolle ist für die Erstellung und Wartung einzelner Compliancetemplates oder Mengen von Compliancetemplates verantwort-

lich, wobei die eigentliche Entwicklung an weitere Personen vergeben oder ausgelagert werden kann. Der Verantwortliche für ein Compliancetemplate ist Ansprechpartner für alle Belange, die das Compliancetemplate betreffen.

- **Prozess-Modellierer:** Ein Prozessmodellierer ist mit der Erstellung und Änderung von Compliancetemplates betraut.
- **Complianceexperte:** Ein Complianceexperte verfügt über Fachwissen im Bereich der Gesetze, die bei der Erstellung und dem Betrieb von Prozessen Anwendung finden. Ihm kommt die Rolle zu, mit den Rechtsprechungen in verschiedenen Ländern bezüglich der Erstellung und der Ausführung von Prozessen vertraut zu sein.
- **Prozessmodellierer (Kunde):** Der Prozess-Modellierer auf Seiten des Kunden ist für die Vervollständigung von Compliancetemplates zuständig. Inhaber dieser Rolle stellen aus einem Compliancetemplate, einen vollständigen Prozess her. Dies geschieht mit den in Kapitel 4.2 vorgestellten Mitteln.

Nachdem der Bedarf für ein neues Compliancetemplate festgestellt wurde, wird ein Projekt erstellt, an dem Personen mit den beschriebenen Rollen mitarbeiten. Dieses Projekt wird vom Projektmanagement bis zur Fertigstellung des neuen Compliancetemplates betreut. Das Compliancetemplate ist nach dem Schritt *Compliance- Scopes und Domains einfügen* fertiggestellt. Im Schritt der Projekterstellung werden Personen mit den benötigten Kenntnissen in das Projekt eingebunden. Weiterhin wird ein Einsatzbereich abgesteckt, in dem das zu erstellende Compliancetemplate angesiedelt sein soll. Dieser Bereich kann

zum Beispiel mit wirtschaftlichen oder geographischen Eigenschaften des Compliancetemplates beschrieben sein. In dem laufenden Beispiel dieser Arbeit ist dies der medizinische Bereich. In diesem Projektschritt werden auch die funktionalen Anforderungen festgelegt, die das neue Compliancetemplate implementieren muss. Hiermit wird festgelegt, welchen Zweck das neue Compliancetemplate erfüllen muss.

Nachdem das Projekt erstellt wurde, kann die Sammlung der für das neue Compliancetemplate relevanten Complianceregeln beginnen. Dies geschieht unter Beachtung des vom Projektmanagement für das neue Compliancetemplate gesteckten wirtschaftlichen Rahmens. In diesem Schritt wird analysiert, welche Anforderungen bezüglich der Rechtssicherheit des zu erstellenden Compliancetemplates beachtet werden müssen. Es muss beispielsweise herangezogen werden, in welchem wirtschaftlichen Bereich ein Prozess ausgeführt werden soll, der mit dem neuen Compliancetemplate erstellt wird. Weiterhin müssen die für das zu erstellende Compliancetemplate relevanten Gesetzestexte ausgewählt werden. Nach der Auswahl müssen diese Gesetzestexte analysiert und in Complianceregeln übersetzt werden, die auf IT-unterstützte Prozesse anwendbar sind. Dieser Schritt ist notwendig, da Gesetzestexte nicht direkt mit Hinblick auf Prozesse geschrieben werden. Im laufenden Beispiel dieser Arbeit ist der wirtschaftliche Bereich, für den die geltenden Complianceregeln ausgewählt werden sollen, der medizinische Bereich. Hier ist der Schutz der Privatsphäre der Patienten von zentraler Bedeutung. Complianceregeln, die den Umgang mit Personendaten betreffen, sind in diesem Bereich restriktiver als in anderen Bereichen.

Die so entstandene Sammlung von Complianceregeln, die für den wirtschaftlichen Rahmen gelten, in dem das neue Compliancetempla-

te eingesetzt werden soll, wird im nächsten Schritt verwendet, um diejenigen Complianceregeln auszuwählen, die auf das neue Compliancetemplate angewendet werden sollen. Die Entscheidungen, welche Complianceregeln dies sind, wird anhand mehrerer Kriterien getroffen. Erstens werden nur Complianceregeln in Betracht gezogen, die im vorhergegangenen Schritt des Erstellungsprozesses als relevant bezeichnet wurden. Zweitens werden die funktionalen Anforderungen in Betracht gezogen, die das zu erstellende Compliancetemplate implementieren muss.

Nach der Auswahl der Complianceregeln folgt deren Übersetzung in eine maschinenlesbare Sprache. Dieser Schritt ist notwendig, da viele Complianceregeln Interpretationen von Gesetzestexten sind. Die Übersetzung von Complianceregeln in maschinenlesbare Ausdrücke wird von Complianceexperten durchgeführt. Eine solche maschinenlesbare Sprache ist zum Beispiel LTL oder die in Kapitel 5.5.1 vorgestellte Sprache zur Definition von datenbasierten Complianceregeln. Im Schritt *Compliancetemplate erstellen* wird das Compliancetemplate von einem Prozessmodellierer unter Berücksichtigung der einzuhaltenden Complianceregeln erstellt. Das Compliancetemplate wird so erstellt, dass es die einzuhaltenden Complianceregeln implementiert. Im laufenden Beispiel stellt eine solche Complianceregel die Überprüfung der Blutproben von mindestens zwei unabhängigen Seiten sicher. Des Weiteren werden die Regionen definiert, die beim Vervollständigen des Compliancetemplates mit neuen Aktivitäten gefüllt werden können.

Sollte es notwendig sein, können im nächsten Schritt Compliancescopes und Compliancedomains in das neue Compliancetemplate eingefügt werden. Compliancescopes können beim Vervollständigen von Compliancetemplates wichtig sein, wenn durch den Anwender Än-

derungen an bereits eingefügten Prozessfragmenten vorgenommen werden. Mittels der Compliancescopes kann der Umfang der zugelassenen Änderungsmöglichkeiten eingeschränkt werden. Zum Beispiel ist es vorstellbar, dass im Compliancetemplate für die Blutentnahme des Roten Kreuzes Hong Kong, die Blutentnahmeaktivität durch eine neuere Version ersetzt wird.

Nach der Fertigstellung kann das Compliancetemplate an einen Kunden übergeben werden. Dieser verwendet Werkzeuge, die die unautorisierte Arbeit mit Compliancetemplates unterbinden. Diese Werkzeuge implementieren die in dieser Arbeit vorgestellten Konzepte.

6.2. Werkzeuge für die Erstellung regelkonformer Prozesse

Alle Softwarekomponenten, die an der Erstellung eines regelkonformen Prozesses beteiligt sind, müssen bestimmten Anforderungen genügen. In der folgenden Liste sind diese Komponenten aufgelistet und die dazugehörigen Anforderungen beschrieben. Abbildung 6.2 zeigt, wie diese Komponenten verbunden sind.

- **Sicheres Templaterepository**: Prüft beim Lesen und Schreiben die Integrität von Compliancetemplates und Prozessen. Das Lesen aus dem Repository und Schreiben in das Repository ist nur für Personen und Werkzeuge erlaubt, die sich am Repository erfolgreich über eine dafür bereitgestellte Schnittstelle authentifiziert haben. Dies kann mit Authentifizierungsmechanismen, die mit asymmetrischen Schlüsseln arbeiten, erreicht werden. Das Templaterepository dient somit dem Ziel den Zugang zu Compliancetemplates und Prozessen, die mit Compliancescopes und Compliancedomains versehen sind, zu beschränken.

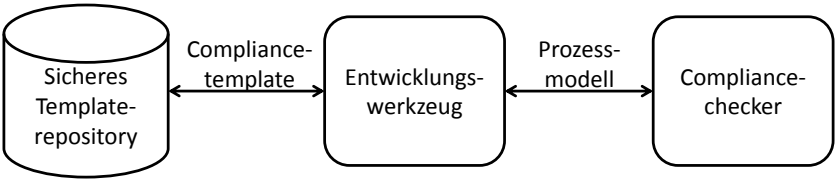

Abbildung 6.2.: Konzeptionelle Übersicht über die Komponenten, die für die Entwicklung regelkonformer Prozesse miteinander arbeiten müssen.

- **Entwicklungswerkzeug für Prozesse**: Ein Entwicklungswerkzeug für die Entwicklung regelkonformer Prozesse muss die in dieser Arbeit vorgestellten Konzepte implementieren. Das Entwicklungswerkzeug kann auch ein einfacher Texteditor sein.
- **Compliancechecker**: Diese Komponente ist für die automatische Untersuchung von Prozessmodellen zuständig. Mit ihr können Prozessmodelle nach einer Modifikation auf Einhaltung von Complianceregeln überprüft werden.

Mit diesen Werkzeugen kann das Compliancetemplate zu einem syntaktisch korrekten Prozess vervollständigt werden. Es wird hierbei an den dafür vorgesehenen Stellen, den Complianceregionen, mit weiteren Aktivitäten befüllt. Ein Compliancetemplate kann nach jeder Modifikation automatisch auf Complianceregelverletzungen überprüft werden. Es kann auch nach Abschluss einer Reihe von Modifikationen automatisch überprüft werden. Bei der Erstellung von Prozessen sind oft mehrere Personen mit unterschiedlichen Fähigkeiten beteiligt, die verschiedene Ziele bei der Vervollständigung eines Compliancetempla-

tes verfolgen. In heutigen, global agierenden Unternehmen, können Personen weit voneinander entfernt sein, so dass eine direkte Zusammenarbeit erschwert ist. Das folgende Kapitel beschreibt ein Konzept zur gemeinschaftlichen Vervollständigung von Compliancetemplates.

6.3. Vervollständigungsebenen: Ein Konzept zur gemeinschaftlichen Entwicklung regelkonformer Prozesse

Analog zur Erstellung herkömmlicher Software können am Entstehungsprozess von Prozessmodellen mehrere Modellierer beteiligt sein. Ausschlaggebend ist hierbei zum einen, dass für den Entwicklungsprozess Modellierer mit unterschiedlichen Fähigkeiten benötigt werden, und zum anderen die durch Parallelentwicklung von Teilprozessen erreichte Zeitersparnis. Ein Konzept, das dabei hilft, die Zusammenarbeit verschiedener Personen bei der Erstellung eines Prozessmodells zu steuern, ist das Konzept der *Vervollständigungsebenen* [SALS10]. Die beiden Hauptmerkmale von Vervollständigungsebenen sind:

- Kontrollflusskonnektoren können die Grenzen von Vervollständigungsebenen überqueren. Dies resultiert daraus, dass Vervollständigungsebenen von ihren zugrundeliegenden Compliancescopes begrenzt werden
- Sie können beliebige Mengen von Elementen eines Prozesses enthalten. Zum Beispiel ist es erlaubt Vervollständigungsebenen zu bilden, die keine Aktivitäten enthalten. Dies resultiert daraus, dass Vervollständigungsebenen von ihren zugrundeliegenden Compliancescopes begrenzt werden

- Vervollständigungsebenen können sich mit anderen Compliancescopes überlappen. Das heißt, sie können Teilmengen von BPMN-Elementen von anderen Vervollständigungsebenen enthalten.

- Sie repräsentieren Phasen des Prozesses, mit dem auf Grundlage von Compliancetemplates, Prozessmodelle vervollständigt werden. Diese Phasen werden zum Beispiel in Abbildung 6.3 als Ebenen dargestellt.

- Sie stellen ein Sichtenkonzept nach [SLS10] dar, das zeigt, welche Informationen in einer bestimmten Phase des Prozesses der Vervollständigung eines Compliancetemplates herangezogen werden können.

Das in diesem Kapitel vorgestellte Konzept ist in der in Abbildung 6.1 gezeigten Methodik an der Stelle *Compliancetemplate füllen* einsetzbar.

Vervollständigungsebenen können bei der Arbeit mit externen Experten unterstützend eingesetzt werden, wenn ihnen zum Beispiel lediglich der Zugang zu den Vervollständigungsebenen und somit Prozessinformationen gewährt wird, die für die Beratung nötig sind. Dadurch müssen Firmen nicht komplette Prozesse offen legen, um Expertenrat einzuholen.

Weiterhin dienen Vervollständigungsebenen der Reduktion der Komplexität bei der Arbeit an einem Prozessmodell. Auf bestimmten Vervollständigungsebenen ist zum Beispiel nicht das gesamte Prozessmodell zu sehen, sondern nur der Teil, der gerade von einer bestimmten Person vervollständigt wird. Dieser Ansatz des Ausblendens von Informationen, die für ein bestimmtes Problem unnötig sind, nennt man

den *Teile-und-herrsche-Ansatz*. Dieser Ansatz ist grundlegend für die Bearbeitung von Problemen, die zu groß sind, um in einem Schritt gelöst zu werden. Dabei wird das Gesamtproblem in Teilprobleme zerlegt, die für sich gelöst werden müssen, um eine Lösung des Gesamtproblems zu bekommen. Vervollständigungsebenen *zwingen* hierbei die an der Erstellung eines Prozessmodells beteiligten Personen den Teile-und-herrsche-Ansatz für die Erstellung des Prozessmodells zu verwenden. Dies folgt zum Beispiel daraus, dass zu einem bestimmten Zeitpunkt während der Vervollständigung nur bestimmte Teile des zu bearbeitenden Prozessmodells sichtbar sind. Im Gegensatz zu BPMN Subprozessen können Compliancescopes beliebige Teilmengen von Elementen eines Prozesses enthalten (siehe Kapitel 4.3.1). Vervollständigungsebenen visualisieren diese Teilmengen von Elementen eines Prozesses.

Abbildung 6.3 zeigt, wie aus einem Compliancetemplate über mehrere Vervollständigungsebenen hinweg ein vollständiger Prozess entsteht. Vollständig ist ein Prozess dann, wenn im Prozess keine nicht befüllte Complianceregion mehr existiert.

Auf Vervollständigungsebene 1 wird ein Compliancetemplate verwendet, welches schon eine für den späteren Prozess wichtige Menge an Complianceregeln implementiert. Um beim Beispielszenario aus Kapitel 4 zu bleiben, könnte dieses Compliancetemplate das in Abbildung 4.2 gezeigte sein. Es könnte somit auf Vervollständigungsebene 1 die Erstellung eines neuen Blutentnahmeprozesses angestoßen werden. Durch das Einfügen neuer Aktivitäten in eine Complianceregion auf einer Vervollständigungsebene wird eine neue Vervollständigungsebene aufgespannt. In Abbildung 6.3 wird durch das Einfügen von Aktivitäten in Complianceregion A die Vervollständigungsebene zwei

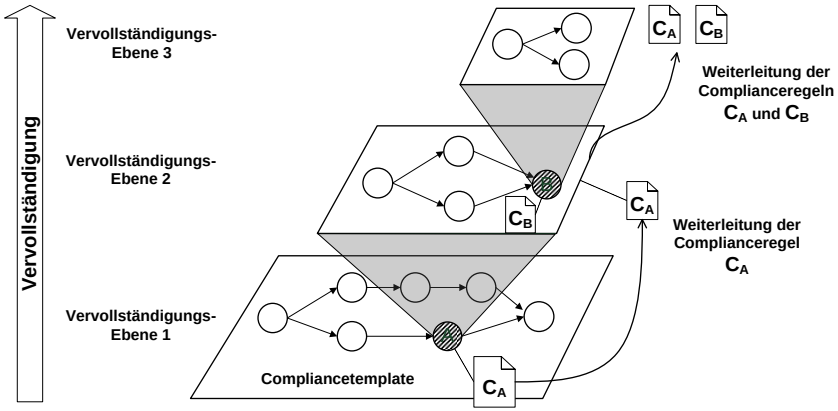

Abbildung 6.3.: Beispiel: Vervollständigungsebenen; Weiterleitung von Complianceregeln (vergleiche [SALS10])

aufgespannt.

Die in Complianceregion A eingefügte Menge von Aktivitäten enthält wiederum eine Complianceregion. Diese muss mit Aktivitäten gefüllt werden, um einen syntaktisch korrekten Prozess zu bekommen. Dies geschieht auf Vervollständigungsebene 3. Die auf Vervollständigungsebene 3 eingefügte Menge von Aktivitäten enthält keine weitere Complianceregion. Somit sind alle Complianceregionen in diesem Beispiel mit Aktivitäten gefüllt. Der Prozess ist syntaktisch korrekt. Abbildung 6.3 zeigt weiterhin, wie Complianceregeln zwischen den Vervollständigungsebenen weitergereicht werden. Darauf wird im folgenden Kapitel eingegangen.

Eine Vervollständigungsebene ist durch ein Tupel

$v = (c, v_V, \mathcal{V}_\mathcal{K})$ mit

- $c \in \mathcal{C}$ als einem Compliancetemplate aus der Menge aller Compliancetemplates C,
- $v_V \in \mathcal{V}$ als der Vatervervollständigungsebene aus der Menge der Vervollständigungsebenen $\mathcal{V}$ und
- V_K als der Menge der Kind-Vervollständigungsebenen von $\mathcal{V}$ beschrieben.

Um die Vatervervollständigungsebene v_v einer Vervollständigungsebene v zu bestimmen, wird die Funktion $vater$ definiert. Jede Vervollständigungsebene hat keine oder höchstens eine Vatervervollständigungsebene. Deshalb definieren wir

$$vater : V \rightarrow V \cup \varnothing.$$

Die Vatervervollständigungsebene v_V wird berechnet, indem man einen Compliancescope s_i aus der Menge aller Compliancescopes S_i einer anderen Vervollständigungsebenen v_i findet, der dieselben Aktivitäten enthält, die in der Vervollständigungsebene v enthalten sind.

Weiter wird die Funktion akt mit A als der Menge aller Aktivitäten und P als der Menge aller Prozesselemente, die Aktivitäten enthalten können, definiert:

$$akt : P \rightarrow A.$$

Mit $akt(v)$, als der Menge aller Aktivitäten einer Vervollständigungsebene v und $akt(s)$, als der Menge aller Aktivitäten eines Complian-

cescopes s kann die Vatervervollständigungsebene v_v einer Vervollständigungsebene v wie folgt berechnet werden:

$$vater(v) = v_V \in V \text{ mit } s_i \in S_i \wedge akt(v) = akt(s)$$

Zur Bestimmung der Kindervervollständigungsebenen wird die Funktion $kinder$ definiert. Jede Vervollständigungsebene hat entweder kein Kind oder eine beliebig große Menge von Kindern:

$$kinder : V \rightarrow 2^V.$$

Die Menge der Kindervervollständigungsebenen V_K wird berechnet, indem man für jeden Compliancescope s_i aus der Menge aller Compliancescopes S_i der Vervollständigungsebene v prüft, ob die Menge der Aktivitäten $akt(v_i)$ gleich der Menge der Aktivitäten $akt(s_i)$ ist. Diese Vervollständigungsebene wird zur Menge der Kindervervollständigungsebenen V_K von v hinzugefügt.

$$kinder(v) = V_K \in 2^V \text{ mit } \forall v_K \in V_K \wedge s_i \in S_i : akt(s_i) = akt(v_k)$$

6.3.1. Verschachtelte Complianceregeln und Flexibilität

Complianceregionen sind mit Complianceregeln verknüpft. Fügt man eine Complianceregion in ein Prozessmodell ein, so werden automatisch auch die mit ihr verknüpften Complianceregeln in das Prozessmodell übernommen. Auf jeder der in Abbildung 6.3 gezeigten Vervollständigungsebenen können durch das Einfügen weiterer Complianceregionen neue Complianceregeln eingeführt werden, die im

ursprünglichen Compliancetemplate nicht vorhanden waren. Damit ist es möglich, dass zum Beispiel externe Berater bei der Vervollständigung eines Compliancetemplates spezielle, für ihr Fachgebiet wichtige Complianceregeln einfügen.

Durch die Möglichkeit des Einfügens weiterer Complianceregionen in ein Compliancetemplate ergeben sich Probleme, die im Folgenden erläutert werden. Das erste Problem tritt beim Einfügen einer Menge von Aktivitäten in ein Compliancetemplate auf, welches mindestens eine Complianceregion enthält. Dies wird in Abbildung 6.4 auf Vervollständigungsebene 1 gezeigt. Hier wird in die Complianceregion (schraffiert) die Menge von vier Aktivitäten eingefügt, die auf Vervollständigungsebene zwei gezeigt wird. In dieser Menge ist wiederum eine Complianceregion (schraffiert) enthalten. Mit beiden Complianceregionen sind Complianceregeln verknüpft, die in Abbildung 6.4 vereinfacht dargestellt sind. Die Complianceregeln auf Vervollständigungsebene 1 verhindern, dass eine Aktivität vom Typ A in die mit ihr verknüpfte Complianceregion eingefügt wird. Die Complianceregel auf Vervollständigungsebene zwei dagegen besagt, dass eine Aktivität vom Typ A in diese mit ihr verbundene Complianceregion eingefügt werden muss. Beim Einfügen von Aktivitäten in die in Vervollständigungsebene zwei platzierte Complianceregion müssen beide Complianceregeln beachtet werden. Die auf Vervollständigungsebene drei eingefügten Aktivitäten werden in die Complianceregion auf Vervollständigungsebene zwei eingefügt und damit implizit auch in die Complianceregion auf Vervollständigungsebene 1. Die beiden mit diesen Complianceregionen verknüpften Complianceregeln schließen sich jedoch gegenseitig aus. Das bedeutet, dass das Erfüllen der einen Complianceregel das Nichterfüllen der anderen Complianceregel zur

Folge hat. Fügt man zum Beispiel auf Vervollständigungsebene drei eine Aktivität vom Typ A ein, so ist die Complianceregel von Vervollständigungsebene zwei erfüllt. Jedoch ist die Complianceregel von Vervollständigungsebene 1 verletzt.

Das zugrundeliegende Konzept der Verarbeitung von Complianceregeln bei der Arbeit mit Vervollständigungsebenen ist die Weiterleitung von Complianceregeln. Complianceregeln, die auf tieferen Vervollständigungsebenen eingefügt wurden, werden, wie in Abbildung 6.3 gezeigt, an höhere Vervollständigungsebenen weitergeleitet. Bei der Weiterleitung von Complianceregeln zwischen Vervollständigungsebenen werden die für eine Vervollständigungsebene geltenden Complianceregeln miteinander verschmolzen. Da diese Arbeit mit Linearer Temporaler Logik (LTL) und einer neuen Sprache zur Definition datenbasierter Complianceregeln arbeitet, ist im Folgenden die Funktion $verschmelze$ zur Verschmelzung von Complianceregeln, die mit diesen Sprachen geschrieben sind, definiert. Mit M, als der Menge zu verschmelzender Ausdrücke, und V, als der Menge aller logischen Ausdrücke, gilt:

$$verschmelze : M \rightarrow V$$

Die Funktion $verschmelze$ ist wie folgt definiert:

$$verschmelze(a, b) = a \wedge b \text{ mit } a, b \in M$$

In dieser Arbeit wird angenommen, dass die Funktion *verschmelze* so implementiert ist, dass sie zwei logische Ausdrücke A und B mittels einer Konjunktion verbindet. Die Konjunktion wird verwendet, da ein

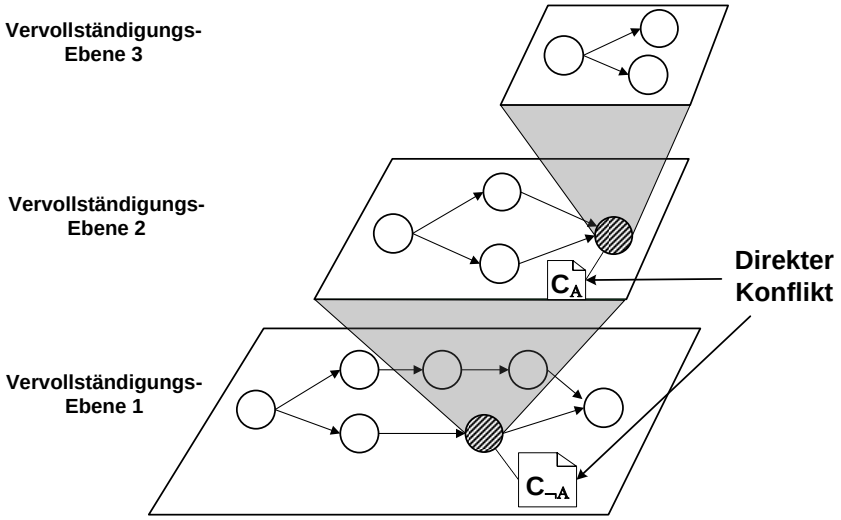

Abbildung 6.4.: Entstehung eines Konflikts beim Einfügen von Complianceregionen (vergleiche [SALS10])

durch Verschmelzung entstandener Ausdruck dann den Wahrheitswert 1 annehmen soll, wenn alle Teilausdrücke wahr sind.

6.3.2. Erfüllbarkeit verschmolzener Regelsätze

Kontrollflussbasierte Compliancerегeln werden in der vorliegenden Arbeit mittels Linearer Temporaler Logik (LTL) beschrieben. Eine Eigenschaft von Ausdrücken in LTL ist die Möglichkeit, dass ein solcher Ausdruck nicht erfüllbar ist. Das bedeutet, dass es für einen unerfüllbaren Ausdruck in LTL keine Belegung gibt, für die der Ausdruck den Wahrheitswert *wahr* annimmt. Eine Belegung definiert die Zuweisung von Werten zu den Variablen einer logischen Formel.

Complianceregeln, die durch Verschmelzung, wie im Kapitel 6.3.1 gezeigt, entstanden sind, können unerfüllbar sein. Grund dafür ist, dass bei der Auswertung der verschmolzenen Complianceregel die Teilausdrücke separat mittels Funktionen ausgewertet werden. Diese Teilausdrücke können somit als Variablen der verschmolzenen aussagenlogischen Formel angesehen werden. Da Aussagenlogische Ausdrücke unerfüllbar sein können, gilt für eine durch Verschmelzung entstandene Complianceregel der Erfüllbarkeitsbegriff der Aussagenlogik.

6.3.3. Behandlung erfüllter Complianceregeln

Im Hinblick auf die automatische Überprüfung von Prozessmodellen anhand von mit ihnen verknüpften Complianceregeln sollten diese Complianceregeln so einfach wie möglich sein. Dafür spricht die exponentielle Laufzeit [Var01] von Modelchecking Algorithmen.

Beim Einfügen von Mengen von Aktivitäten in ein Prozessmodell kann der Fall auftreten, dass eine Complianceregel erfüllt wird. Wird durch das Einfügen von Aktivitäten in Complianceregionen ein Teil einer Complianceregel erfüllt, so muss dieser Teil bei der weiteren Befüllung des Prozessmodells nicht mehr bei automatischen Überprüfungen beachtet werden. Diese Complianceregeln werden nicht an die nächst höhere Vervollständigungsebene weitergeleitet. Dies wird im Folgenden erläutert.

Abbildung 6.5 zeigt mehrere Beispiele für die Weiterleitung nicht erfüllter Complianceregeln. Die auf Vervollständigungsebene 1 eingeführte Complianceregel C_X drückt aus, dass eine Aktivität vom Typ X eingefügt werden muss. Die Complianceregel C_X wird durch das

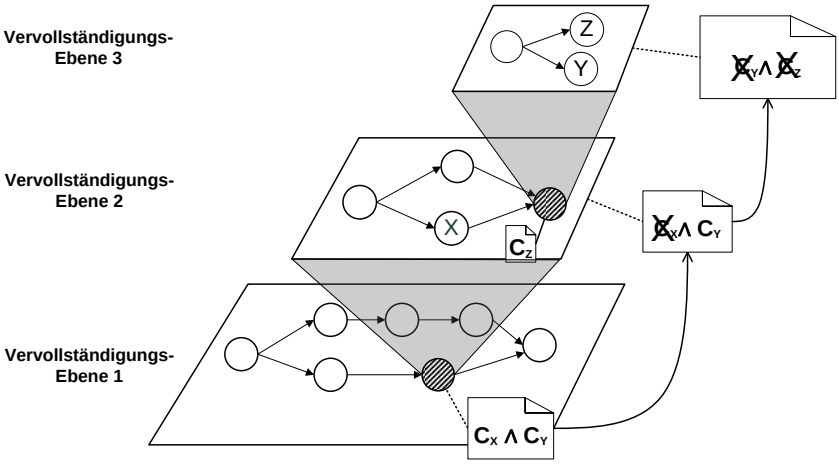

Abbildung 6.5.: Weiterleitung von nicht erfüllten Complianceregeln (vergleiche [SALS10])

Einfügen der Aktivität X auf Vervollständigungsebene 2 erfüllt und somit nicht weitergeleitet. Welche Complianceregeln weitergeleitet werden dürfen, kann automatisch mit einem Modelchecker berechnet werden. Dafür wird jede Teilregel einer kombinierten Complianceregel einzeln auf Erfüllung durch die zugrundeliegende Complianceregion überprüft. Ist die Teilregel erfüllt, muss sie nicht weitergeleitet werden. Einzig die Complianceregel C_Y wird weitergeleitet. Parallel dazu wird auf Vervollständigungsebene zwei die Complianceregel C_Z eingeführt. C_Z wird mit der Complianceregel C_Y zur Vervollständigungsebene drei weitergeleitet. Hier werden die beiden Complianceregeln C_Y und C_Z durch das Einfügen der Aktivitäten Y und Z erfüllt. Weiterhin ist hier der Vorgang der Vervollständigung dieses Compliancetemplates

beendet, da alle Complianceregionen mit mindestens einer Aktivität befüllt worden sind.

6.3.4. Auftreten von Konflikten zwischen Complianceregeln

Durch das Weiterleiten und Verschmelzen von Complianceregeln entstehen zwei Arten von Konflikten, die beim Vervollständigen von Compliancetemplates auftreten können, *direkte* und *indirekte Konflikte*. Ein *Konflikt* tritt auf, wenn zwei sich gegenseitig ausschließende logische Ausdrücke bei der Weiterleitung von Complianceregeln zwischen Vervollständigungsebenen miteinander verknüpft werden. Hierbei ist zu beachten, dass für zwei sich gegenseitig ausschließende Ausdrücke A und B gilt:

$$(A \Leftrightarrow \neg B) \tag{6.1}$$

Mit V, als der Menge aller Vervollständigungsebenen gilt. Ein direkter Konflikt wird mit zwei Vervollständigungsebenen $v_a \in V$ und $v_b \in V$ wie folgt bestimmt. v_b ist ein Element der Menge der Kindvervollständigungsebenen von v_a:

$$v_b \in kinder(v_a)$$

Weiter wird die Funktion cr mit S als der Menge aller Compliancescopes und R als der Menge aller Complianceregeln definiert:

$$cr : S \rightarrow R.$$

Mit S_a als der Menge aller Compliancescopes der Vervollständigungsebene v_a und $akt(s_a)$, als der Funktion, die die Menge der im Compliancescope $s_a \in S_a$ enthaltenen Aktivitäten zurück gibt, gilt. Die Funktion $cr(s_a)$ gibt die mit dem Compliancescope $s_a \in S_a$ verknüpfte Complianceregel zurück. Ein Konflikt wird *direkt* genannt, wenn die Negation von $cr(s_b)$ mit einer auf v_a liegenden Complianceregion s_a verknüpft ist.

Sei der Compliancescope $s_b \in akt(s_a)$. Ein Konflikt wird *indirekt* genannt, wenn die Negation von $cr(s_a)$ mit einer auf v_b liegenden Complianceregion s_b verknüpft ist.

Mit der Funktion $konfl$ wird berechnet, ob ein Konflikt direkt oder indirekt ist, oder ob kein Konflikt vorliegt.

$$konfl(s_a, s_b) = \begin{cases} direkt, & \text{mit } s_a \in S_a \wedge s_b \in akt(S_a) \wedge cr(s_a) = \neg cr(S_b) \\ indirekt, & \text{mit } s_a \in S_a \wedge s_b \in akt(S_a) \wedge \neg cr(s_a) = cr(S_b) \\ kein, & andernfalls \end{cases}$$

Ein Konflikt kann beseitigt werden, wenn mindestens eine der beiden am Konflikt beteiligten Formeln geändert wird. Unter Verwendung des in Abschnitt 6.3.3 beschriebenen Konzepts zur Löschung von erfüllten Teilausdrücken von Complianceregeln, kann eine Complianceregel auch durch das Einfügen einer neuen Aktivität geändert werden. Da eine erfüllte Teil-Complianceregel nicht mehr zur automatischen Überprüfung herangezogen wird, hat sich die effektiv zu überprüfende Complianceregel geändert.

Ein indirekter Konflikt kann durch das Einfügen von Aktivitäten in ein Prozessmodell beseitigt werden, wenn das Einfügen einer Men-

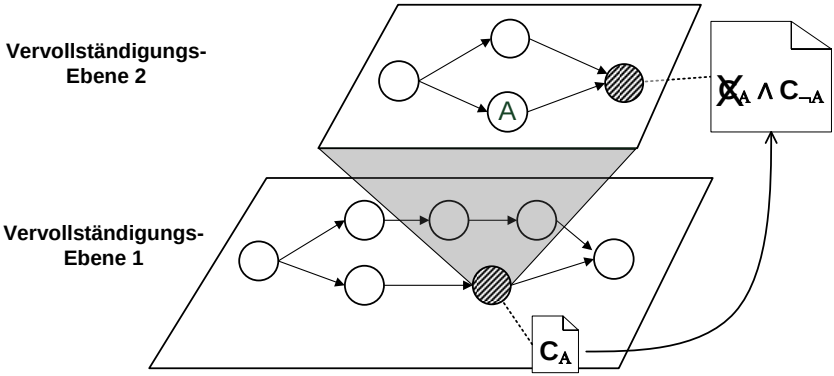

Abbildung 6.6.: Beseitigung eines indirekten Konflikts. Durch das Einfügen der Aktivität A auf Vervollständigungsebene 2 wird die Complianceregel C_A gelöscht, so dass sie nicht mehr bei der automatischen Überprüfung herangezogen wird. Folglich wird bei der automatischen Überprüfung der unerfüllbare Ausdruck $C_A \wedge \neg C_A$ in den erfüllbaren Ausdruck $\neg C_A$ überführt.

ge von Aktivitäten die positive der beiden in Konflikt stehenden Complianceregeln erfüllt. Somit kann der erfüllte Teil der verschmolzenen Complianceregel aus dem Ausdruck entfernt werden. Der Ausdruck ist damit erfüllbar.

Direkte Konflikte können nicht durch das Einfügen von Aktivitäten beseitigt werden.

Abbildung 6.6 zeigt ein Beispiel für das Beseitigen eines indirekten Konflikts. Die auf Vervollständigungsebene 1 eingeführte Complianceregel C_A wird durch das Einfügen der Aktivität A auf Vervollständigungsebene 2 erfüllt. Sie kann deshalb gelöscht werden und wird

damit nicht mehr bei der automatischen Überprüfung beachtet. Die Complianceregeln, die zur Überprüfung des Prozessmodells verwendet werden, sind somit erfüllbar. Die auf der Vervollständigungsebene 2 gezeigte verschmolzene Complianceregel stellt dies dar.

Ein direkter Konflikt kann nicht durch Einfügen von Aktivitäten in einen Prozess beseitigt werden. Dies rührt daher, dass bei einem direkten Konflikt der umgekehrte Fall des in Abbildung 6.6 gezeigten Szenarios eintritt. Das heißt, die auf Vervollständigungsebene 1 gezeigte Complianceregel müsste statt C_A ($C_{\neg A}$) heißen. Dies bedeutet, dass in die schraffierte Aktivität auf Vervollständigungsebene 1 keine Aktivität vom Typ A eingefügt werden darf. Diese Regel gilt erst dann als erfüllt, wenn der Prozess mit anderen Aktivitäten befüllt und somit syntaktisch korrekt ist. Eine negative Complianceregel muss also zu jeder Zeit während der Befüllung eines Prozessmodells erfüllt sein. Tritt ein Konflikt mit einer negativen, auf einer niedrigen Vervollständigungsebene verknüpften Complianceregel auf, so muss dieser Konflikt durch Umschreiben der Complianceregeln aufgelöst werden. Zum Beispiel könnte die negative Complianceregel erst auf einer höheren Vervollständigungsebene Anwendung finden.

Eine Einschränkung des Ansatzes der Prozesserstellung mit Vervollständigungsebenen ist, dass Änderungen an einem Prozessmodell nur von einer niedrigeren Vervollständigungsebene an eine höhere Vervollständigungsebene weitergegeben werden können, da das Weiterreichen von Änderungen in die andere Richtung mit einigen Problemen behaftet ist. Diese Probleme sollen hier genannt aber nicht bearbeitet werden, da sie nicht im Fokus dieser Arbeit liegen.

6.4. Überprüfung von Complianceregeln von verschachtelten Compliancescopes

Der Algorithmus zur Untersuchung von Complianceverstößen in Prozessen mit ineinander verschachtelten Compliancescopes ist in Algorithmus 6.1 dargestellt. Im Folgenden wird der Algorithmus erläutert.

Der Algorithmus untersucht rekursiv alle in einem Startcompliancescope enthaltenen Compliancescopes auf Verletzungen von Complianceregeln. Er beginnt mit dem aktuell äußersten Compliancescope und endet mit dem Überprüfungsergebnis des innersten Compliancescopes. Der Algorithmus endet zudem, wenn mindestens eine Complianceregel in einem beliebigen Compliancescope verletzt oder wenn eine Complianceregel gefunden wurde, die unerfüllbar ist.

In Zeile 3 wird die mit dem als Parameter übergebenen Compliancescope verknüpfte Complianceregel mit den Complianceregeln verknüpft, die von äußeren Compliancescopes weitergegeben wurden. Danach wird überprüft, ob diese kombinierte Complianceregel erfüllbar ist. Ist dies nicht der Fall, kann hier die Überprüfung abgebrochen werden, da eine unerfüllbare Complianceregel per Definition nie von einem Prozess erfüllt werden kann.

Ist die kombinierte Complianceregel erfüllbar (Zeile 4), so wird das Modelchecking des aktuellen Compliancescopes angestoßen (Zeile 7). Bei diesem Schritt werden zusätzlich erfüllte Teilregeln der kombinierten Complianceregel gelöscht. Da diese Teilregeln erfüllt sind, müssen sie nicht an innere Compliancescopes weitergegeben werden. Diese Verkürzung der aktuellen Complianceregel führt zu kürzeren Laufzeiten bei den Überprüfungen der inneren Compliancescopes. Im besten Fall kann die aktuelle Complianceregel vollständig erfüllt sein, so dass

Algorithmus 6.1 Beschreibung des Algorithmus der Überprüfung von Complianceregeln verschachtelter Compliancescopes in Pseudocode. Vergleiche: [Bur12]

```
1: function checkCompliance(ComplScope scope, ComplRule outerRules)
2:     ComplRule complRuleCurrentScope = scope.getComplRule();
3:     ComplRule combinedComplRule = concat(complRuleCurrentScope, outerRules);
4:     if satCheck(combinedComplRule) then
5:         //Fulfilled compliance rules are deleted from
6:         //combinedComplRule in method modelcheck.
7:         if modelcheck(scope, combinedComplRule) then
8:             List innerComplScopes = generateInnerComplScopes(scope);
9:             if size(innerComplScopes) < 1 then
10:                print(true);
11:            else
12:                for ComplScope innerScope in innerComplScopes do
13:                    checkCompliance(innerScope, combinedComplRule);
14:                end for
15:            end if
16:        else
17:            print(false);
18:        end if
19:    else
20:        print("Combined compliance rule not satisfiable.");
21:    end if
22: end function
```

die Compliancеprüfung innerer Compliancescopes nicht angestoßen werden muss, falls der innere Compliancescope mit keiner eigenen Complianceregel verknüpft ist.

Zeigt der Modelchecker einen Verstoß gegen die kombinierte Complianceregel des aktuellen Compliancescopes an, so wird ein Beispiel generiert, das den Ausführungspfad im Prozess bis zur Verletzung der Complianceregel anzeigt. Wird keine Verletzung einer Complianceregel des aktuellen Compliancescopes gefunden, so wird die Liste der inneren Compliancescopes generiert (Zeile 8).

Sind keine inneren Compliancescopes vorhanden, kann die Überprüfung abgebrochen und das Überprüfungsergebnis ausgegeben werden (Zeile 9). Das Ergebnis ist in diesem Fall positiv. Sind innere Compliancescopes vorhanden, wird diese Liste durchlaufen (Zeile 12). Jeder in dieser Liste enthaltene Compliancescope ist ein Eingabeparameter für den rekursiven Aufruf dieses Algorithmus. Abbildung 6.7 zeigt eine in BPMN erstellte graphische Repräsentation des oben beschriebenen Algorithmus.

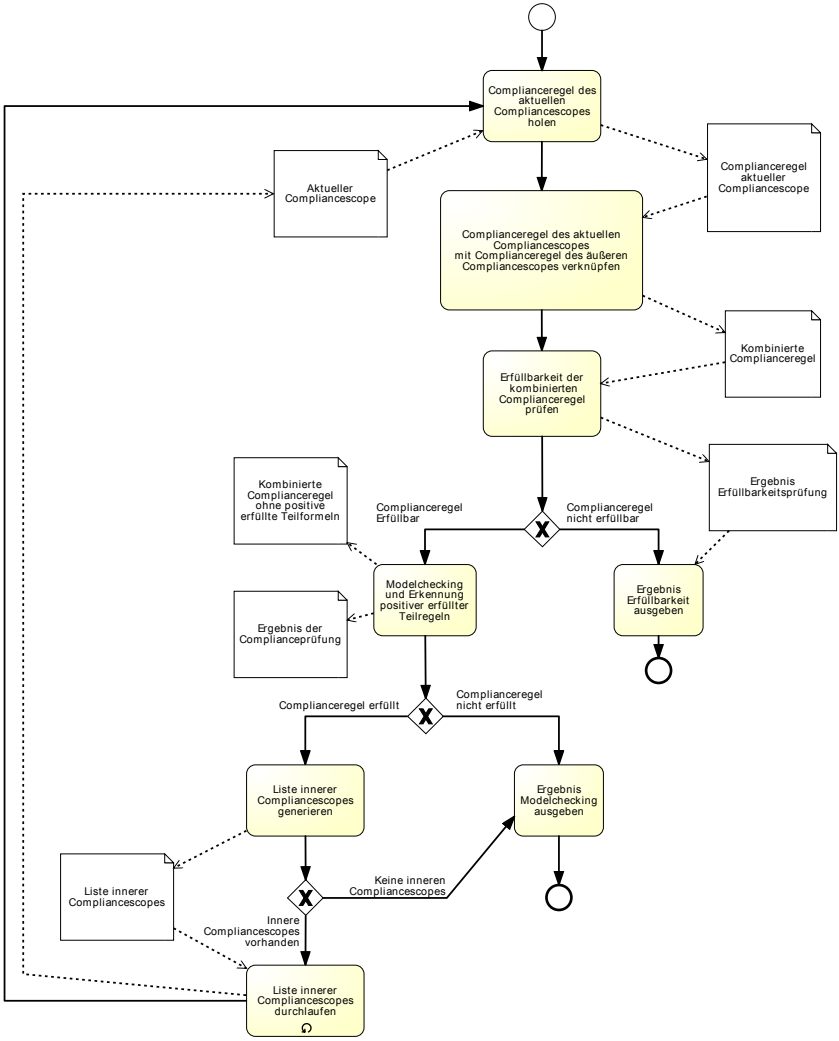

Abbildung 6.7.: Beschreibung des Algorithmus der Überprüfung von Complianceregeln verschachtelter Compliancescopes in BPMN. Quelle: [Bur12]

6.5. Zusammenfassung

Die in dieser Arbeit vorgestellten Konzepte, die der Unterstützung menschlicher Prozessmodellierer dienen, müssen sinnvoll miteinander kombiniert werden, um ihre volle Wirksamkeit entfalten zu können. Bei der Erstellung von Prozessen müssen Werkzeuge eingesetzt werden, die bestimmte Complianceregeln auf den Prozessen überprüfen. Daraus folgt, dass es nicht möglich sein darf, die überwachte Erstellung von Prozessen zu umgehen.

Dies wird durch eine in diesem Kapitel vorgestellte Methodik erreicht. Die Umsetzung dieser Methodik verlangt die Verwendung der in dieser Arbeit vorgestellten Konzepte.

Der in diesem Kapitel vorgestellte BPMN-Prozess dient Organisationen als Anhaltspunkt, wie die verschiedenen Konzepte verbunden werden sollten. Die Methodik besteht aber nicht nur aus diesem BPMN-Prozess, sondern auch aus begleitenden Anforderungen. Die beiden wichtigsten Anforderungen sind die im BPMN-Prozess verwendeten Rollen, die in einer Organisation eingeführt werden müssen, damit die Methodik umgesetzt werden kann. Weiterhin müssen auf der Seite der IT-Infrastruktur Anforderungen erfüllt werden. Es muss zum Beispiel ein Repository bereitgestellt werden, auf welches nur mit bestimmten Werkzeugen zugegriffen werden kann. Diese Werkzeuge sind im Sinne dieser Arbeit Prozessmodellierungswerkzeuge. Durch die Einschränkung des Zugriffs auf das Prozessrepository ist von Beginn der Entwicklung eines Prozesses festgelegt, dass Änderungen an Prozessen nur mit den dafür vorgesehenen Prozessmodellierungswerkzeugen vorgenommen werden können.

Die Erstellung von Prozessen in Unternehmen verlangt oft die Ein-

beziehung mehrerer Personen, die verschiedene Expertisen aufweisen. Die Zusammenarbeit verschiedener Personen an einem Prozessmodell wird innerhalb der in diesem Kapitel vorgestellten Methodik durch das Konzept der Vervollständigungsebenen geregelt. Die erste dieser Vervollständigungsebenen ist das Compliancetemplate auf dem der spätere Prozess basiert. Hier werden die ersten Änderungen vollzogen indem neue Aktivitäten eingefügt werden. Eine weitere Vervollständigungsebene wird geschaffen, wenn eine Complianceregion mit diesen neuen Aktivitäten in das Prozessmodell eingefügt wird. Auf der nächsten Vervollständigungsebene kann der Prozess weiter mit Aktivitäten gefüllt werden, bis keine leere Complianceregion mehr übrig bleibt.

In diesem Kapitel wurde weiterhin ein Problem bei der Arbeit mit Vervollständigungsebenen aufgezeigt. Beim Weiterleiten von Complianceregeln von einer niedrigeren Vervollständigungsebene zu einer höheren kann es vorkommen, dass die daraus resultierenden Regelsätze unerfüllbar werden. Dies mündet in Konflikten zwischen den einzelnen Vervollständigungsebenen, die entweder durch das Einfügen von Aktivitäten auf höheren Vervollständigungsebenen oder durch menschliches Eingreifen behoben werden können. Konflikte können sowohl Complianceregeln, die den Datenfluss eines Prozesses einschränken, als auch Complianceregeln, die den Kontrollfluss eines Prozesses einschränken, betreffen. Auch Complianceregeln, die den Datenfluss einschränken, können sich gegenseitig ausschließen.

KAPITEL 7

PROTOTYP

Eine Evaluation der hier vorgestellten Konzepte erfolgt durch die Implementierung in einem Prototyp. Der Prototyp implementiert und integriert alle in dieser Arbeit vorgestellten Konzepte und Lösungsansätze und ist unter http://www.danielschleicher.com erreichbar.

7.1. Funktionalität des Prototyps

Dieser Abschnitt bietet eine Sicht auf die Funktionen des Prototyps dieser Arbeit. Der Prototyp baut auf dem webbasierten BPMN-Editor Oryx [DOW08] auf.

Die folgende Liste an Funktionen wurde Oryx in dieser Arbeit hinzugefügt:

1. **Spezieller Task für Complianceregion**: Die Menge von Sym-

bolen der BPMN 1.0 Spezifikation wurde durch einen neuen Task erweitert. Dieser Task stellt eine Complianceregion (siehe Abschnitt 4.2) dar. Er ist durch ein Puzzle-Teil gekennzeichnet.

2. **Modus zur Komplettierung von Compliancetemplates**: Bei der Befüllung von Compliancetemplates ist durch einen speziellen Modus sichergestellt, dass nur Complianceregionen mit neuen Aktivitäten befüllt werden können.

3. **Neue Form für die Erstellung von Compliancescopes und Compliancedomains**: Die Menge von Symbolen der BPMN 1.0 Spezifikation wurde durch eine neue Form erweitert. Mit ihr kann man Compliancescopes (siehe Abschnitt 4) oder Compliancedomains (siehe Abschnitt 5) modellieren. Diese Form kann alle BPMN 1.0 Elemente enthalten. Außerdem können Complianceregeln mit ihr verknüpft werden.

4. **Überprüfung von Compliancescopes und Compliancedomains**: Compliancescopes und Compliancedomains teilen ein Prozessmodell in verschiedene Bereiche auf. Mit dem Prototyp ist es möglich, einzelne Compliancescopes automatisch auf Verletzungen von Complianceregeln untersuchen zu lassen. Hierbei können beliebige in einem Prozessmodell vorhandene Compliancescopes oder Compliancedomains zur Überprüfung ausgewählt werden.

5. **Verknüpfung Complianceregeln mit Complianceregionen, Compliancedomains oder Compliancescopes**: Die in dieser Arbeit gezeigten neuen Modellierungskonstrukte Complianceregion, Compliancescope und Compliancedomain, mit denen BPMN

1.0 erweitert wurde, können mit Complianceregeln verknüpft werden.

6. **Beispiel für Verletzung einer Complianceregel**: Die in dieser Arbeit verwendeten Modelchecker erzeugen beim Auffinden der Verletzung einer Complianceregel ein Beispiel, das zeigt, wie diese Verletzung zustande kam. Dieses Beispiel wird in der Sprache des Modelcheckers ausgegeben. Im Prototyp wird diese Ausgabe transformiert und auf den aktuell überprüften BPMN-Prozess abgebildet. Damit ist es für menschliche Prozessmodellierer leichter, Maßnahmen zur Behebung von Regelverletzungen zu ergreifen.

7. **Verschachtelung von Compliancescopes**: Compliancescopes können ineinander verschachtelt werden.

8. **Behandlung von Complianceregeln ineinander verschachtelter Compliancescopes**: Die Complianceregeln von ineinander verschachtelten Compliancescopes werden nach dem in Abschnitt 6.3 vorgestellten Mechanismus miteinander verknüpft und auf Erfüllbarkeit geprüft.

9. **Graphische Repräsentation von Vervollständigungsebenen**: Vervollständigungsebenen werden graphisch so repräsentiert, dass alle Bereiche in einem Prozessmodell, die in einer Vervollständigungsebene geändert werden dürfen, dunkel dargestellt werden.

10. **Verarbeitung von kontrollfluss- und datenflussbasierten Complianceregeln**: Der Prototyp ist dafür ausgelegt, kontrollfluss-

basierte (Sprache LTL) und datenflussbasierte (Sprache XPath) Compliancerегeln auf einem Prozessmodell zu überprüfen. Auch können diese beiden Arten von Compliancerегeln miteinander zu einer komplexen Compliancerегeln (siehe Abschnitt 5.5) kombiniert werden.

Bei den Arbeiten am Prototyp sind weitere Funktionalitäten entwickelt worden, die nicht direkt auf einen Beitrag dieser Dissertation zurückgeführt werden können. Sie entstanden, um Menschen weitere Möglichkeiten zur Verfügung zu stellen komfortabel mit Oryx zu arbeiten. Sie werden in der folgenden Liste kurz erläutert.

11. **Graphische Modellierung von kontrollflussbasierten Compliancerегeln**: Oryx wurde mit einem Plugin und einer Menge von Symbolen erweitert, die es ermöglichen, LTL-Formeln graphisch zu entwickeln.

    Der zweite Grund für die Einführung der graphischen Modellierung von LTL-Formeln ist die Erleichterung des Umgangs mit LTL-Formeln, damit von der abstrakten textuellen Repräsentation Abstand genommen werden kann.

12. **Bestimmung der Zeitabstände der Überprüfung von Compliancerегeln**: Es ist mit dem Prototyp möglich einzustellen, wie oft eine automatische Überprüfung des Prozessmodells durchgeführt werden soll. Es kann ein Zeitintervall eingestellt werden, nach dessen Ablauf der Prozess automatisch auf Verletzungen von Compliancerегeln überprüft wird. Außerdem kann festgelegt werden, dass das Prozessmodell nach Durchführung einer

bestimmten Zahl von Änderungen automatisch überprüft werden soll.

13. **Verwendung eines Regelbaums (siehe 5.5.4)**: Die Repräsentation einer kombinierten Complianceregel in einem Baum macht es für Menschen einfacher, mit der Verschachtelung der einzelnen Teilausdrücke umzugehen.

7.2. Architektur des Prototyps

Es handelt sich bei dem für diese Arbeit implementierten Prototyp um ein graphisches Entwicklungswerkzeug für Prozesse. Er wurde auf Grundlage des web-basierten BPMN Editors Oryx entwickelt.

Die Benutzeroberfläche von Oryx wird in Abbildung 7.1 gezeigt. Sie ist in vier Teilbereiche aufgeteilt:

- **Prozessmodell (Mitte):** In der Mitte ist das Prozessmodell zu sehen, das gerade bearbeitet wird. Es enthält ein das abstrakte Prozessmodell eines Compliancetemplates, welcher der Erstellung des in dieser Arbeit verwendeten Beispielprozesses dient. Die grün markierten Aktivitäten können im nächsten Schritt mit Aktivitäten oder Prozessfragmenten gefüllt werden.

- **Symbolleiste (oben):** Die Symbolleiste im oberen Teil des Oryx-Fensters enthält Schaltflächen für die wichtigsten Funktionen von Oryx, wie zum Beispiel Speichern oder Kopieren und Einfügen. Mittels Plugins können hier neue Schaltflächen eingebunden werden, wie dies für den Prototyp der vorliegenden Arbeit geschehen ist.

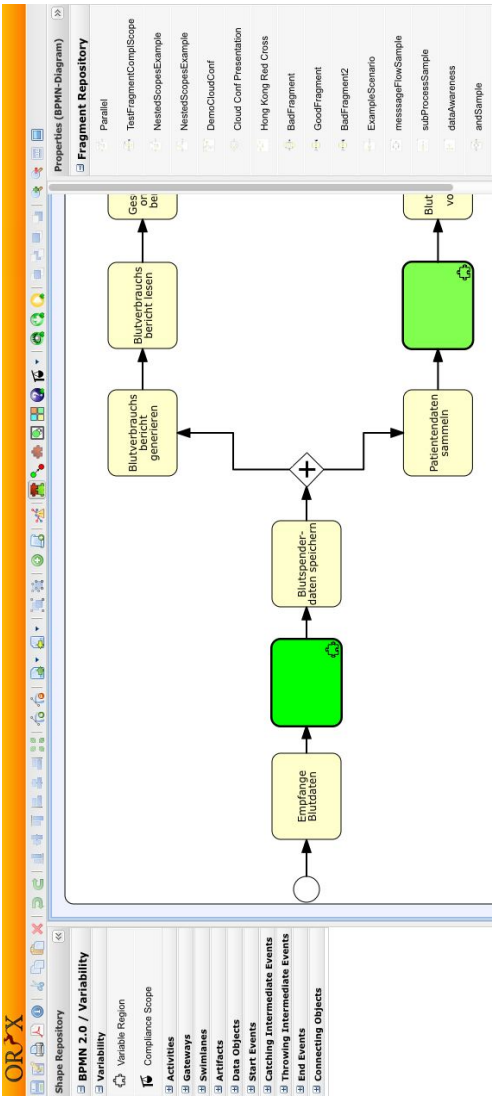

Abbildung 7.1.: Oberfläche von Oryx. Rechts befindet sich das Sidebar-Plugin. Es zeigt, abhängig von der aktuellen Modellierungssituation, entweder Eigenschaften des gerade markierten Teils des Prozessmodells oder die für das Füllen von Complianceregionen verfügbaren Prozessfragmente.

- **Shape-Repository (links):**Auf der linken Seite sieht man das sogenannte Shape-Repository. Hier sind die Objekte hinterlegt, mit denen gearbeitet werden kann, um einen Prozess zu modellieren. In Abbildung 7.1 sind Objekte zur Erstellung von BPMN 1.0-Prozessmodellen geladen. Sie zeigt weiterhin zwei für diesen Prototyp erstellte neue Objekte: Die Variable Region und den Compliancescope.

- **Fragment-Repository (rechts):** Auf der rechten Seite wird das sogenannte Fragment-Repository gezeigt. Es enthält Prozessfragmente, die in das in der Mitte gezeigte Prozessmodell eingefügt werden können.

Abbildung 7.2 gibt eine Übersicht über die wichtigsten Komponenten von Oryx und den Komponenten des Prototyps. Die Pfeile stellen Aufrufbeziehungen dar, wobei ein Pfeil von der aufrufenden zur aufgerufenen Komponente zeigt. Oryx besteht aus zwei Komponenten, dem Backend und dem Frontend. Beide Komponenten sind als sogenannte Web-Applications implementiert und laufen auf einem Servlet-Container, wie zum Beispiel Tomcat[1].

Das Frontend von Oryx ist hauptsächlich in JavaScript implementiert. Der in JavaScript implementierte Teil von Oryx verwendet die JavaScript-Frameworks Prototype[2] und EXTJs[3]. Prototype stellt Konzepte aus der Objektorientierung zu Verfügung, während EXTJs ein Framework zur Erstellung von dynamischen Weboberflächen ist. Die in Abbildung 7.2 gezeigte Komponente Oryx-Core implementiert die

[1] http://tomcat.apache.org
[2] http://www.prototypejs.org
[3] http://www.sencha.com/products/extjs

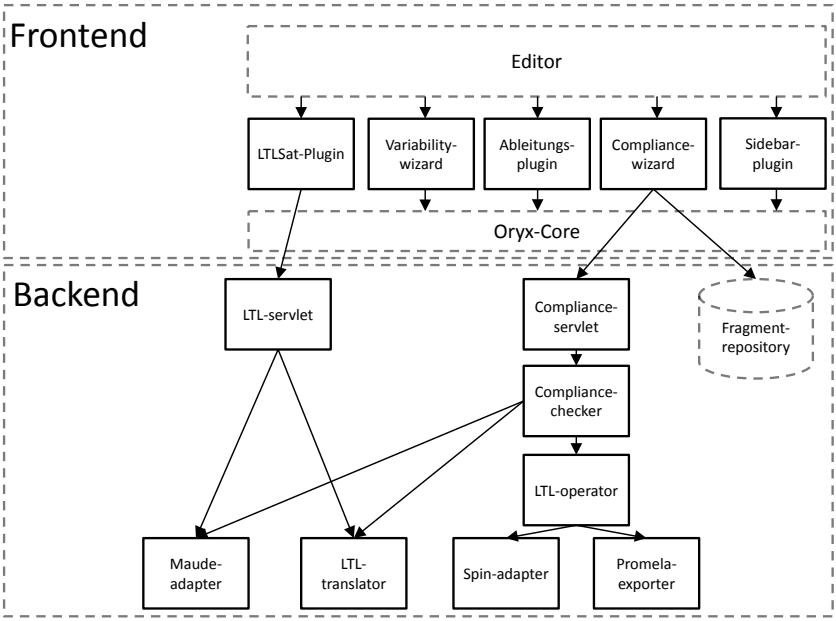

Abbildung 7.2.: Überblick über die Architektur des Prototyps. Pfeile zeigen von der aufrufenden zur aufgerufenen Komponente. Die in der vorliegenden Dissertation erstellten Komponenten sind mit durchgezogenen Linien gezeichnet. Alle schon vorhandenen Komponenten sind mit unterbrochenen Linien gezeichnet.

grundlegenden graphischen und funktionalen Aspekte von Oryx. Zum Beispiel ist hier ein Plugin-Mechanismus implementiert.

Alle weiteren Komponenten von Oryx sind als Plugins implementiert. Zum Beispiel werden mit Oryx Plugins zur Speicherung von Prozessmodellen, zum Export von Prozessmodellen oder zum Rückgängigmachen von Arbeitsschritten bereitgestellt. Die in Abbildung 7.2 gezeigten

Plugins auf Frontend-Seite wurde im Rahmen in dieser Arbeit erstellt. Tabelle 7.2 zeigt sie zusammen mit der von ihnen implementierten Funktionalität.

Das Backend von Oryx ist in Java geschrieben und verwendet die Servlet-Technologie, um mit dem Frontend zu kommunizieren. Auch dieser Teil von Oryx verfügt über einen Plugin-Mechanismus, welcher in der vorliegenden Arbeit genutzt wurde, um Schnittstellen für das Aufrufen von Modelcheckern zu schaffen. In dieser Arbeit werden die beiden Modelchecker SPIN (Spin-Adapter) und Maude (Maude-Adapter) über Plugins im Backend aufgerufen.

Im Folgenden werden alle neu zu Oryx hinzugekommenen Softwarekomponenten, wie zum Beispiel Plugins näher erläutert. Danach wird das Zusammenspiel dieser Softwarekomponenten bei der Ausführung eines Anwendungsfalls, der aus dem Beispielszenario stammt, gezeigt.

Tabelle 7.1.: Funktionen und implementierende Komponenten des Prototyps

Nummer der Funktionalität aus Liste in Kap. 7.1	**Komponente des Prototyps**							
	Compliance-wizard	Variabilitäts-Wizard	Sidebar-Plugin	Ableitungs-Plugin	LTL-Plugin	Compliance-servlet	LTL-Servlet	Compliance-checker
1								
2				X				
3								
4						X		X
5	X	X	X					
6								X
7								X
8	X							X
9								
10	X	X			X		X	X
11								
12	X							
13	X							

7.3. Compliancewizard

Der Compliancewizard ist ein Frontend-Plugin, welches einen neuen Knopf in die Liste im oberen Bereich des Oryx Editorfensters platziert. Abbildung 7.3 zeigt die Liste der Auswahlmöglichkeiten, die erscheint, wenn dieser neue Knopf gedrückt wird. Die genaue Funktionalität jedes Elements in der Liste kann in [Gro11] nachgelesen werden. In dieser Arbeit soll auf die wichtigsten dieser Auswahlelemente eingegangen werden. Der Compliancewizard setzt die Funktionalitäten 8, 10, 5, 12 und 13 der Liste in Abschnitt 7.1 um.

Wird das oberste Auswahlelement gedrückt, so gelangt man in die Ansicht des Compliancewizards, welche im Folgenden genauer erläutert wird. Mit dem zweiten und dritten Auswahlelement können automatische Complianceüberprüfungen sowohl vom gesamten Pro-

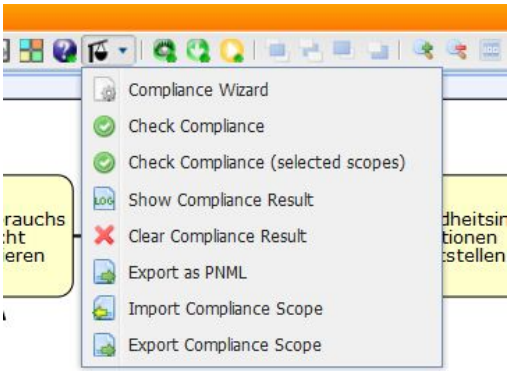

Abbildung 7.3.: Neuer Knopf mit Funktionalitäten zur Überprüfung von an den Prozess annotierten Complianceregeln.

zessmodell als auch von einzelnen ausgewählten Compliancescopes angestoßen werden.

Abbildung 7.4 zeigt das Popup-Fenster des Compliancewizards. Dieses Fenster wird nur angezeigt, wenn ein Compliancescope oder eine Complianceregion zuvor im Editor mit der Maus markiert wurde. Ist dies der Fall, so werden die mit diesem Compliancescope oder dieser Complianceregion verknüpften Complianceregeln angezeigt.

Die Anzeige der Regeln erfolgt als Regelbaum. Mit diesem Konzept ist es möglich, graphisch die Verschachtelung von LTL-Formeln darzustellen. Die Knöpfe im oberen Bereich des Fensters des Compliancewizards können dazu verwendet werden, das Aussehen dieses Regelbaumes zu verändern. Im Besonderen können mit dem mit *LTL* bezeichneten Knopf neue in LTL geschriebene Complianceregeln in den Regelbaum eingefügt werden. Mit dem mit *DATATRANSFER* bezeichneten Knopf können Complianceregeln, die in der Sprache zur Definition von datenbasierten Complianceregeln geschrieben sind, in den Regelbaum eingefügt werden.

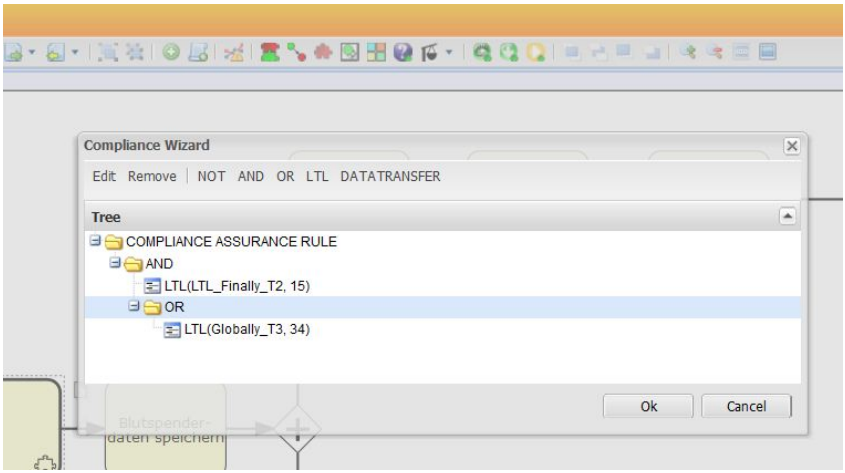

Abbildung 7.4.: Compliancewizard: Dient der Annotation von Complianceregeln an Complianceregionen oder Compliancescopes

7.4. Variabilitäts-Wizard

Der Variabilitäts-Wizard ist ein weiteres Frontend-Plugin, welches bei der Erstellung von Compliancetemplates hilft. Es stellt die folgenden Funktionalitäten bereit, welche in der Diplomarbeit von Falko Kötter [Köt10] im Detail beschrieben sind:

- Annotation von Alternativen an Complianceregionen: Alternativen sind Prozessfragmente, die in ein Compliancetemplate oder einen Compliancescope eingefügt werden können [MMLP09] (Funktionalität 5 in der Liste in Abschnitt 7.1).
- Definition von Abhängigkeiten zwischen Complianceregionen in

einem Compliancetemplate:

Diese Abhängigkeiten geben eine Reihenfolge vor, in der die Complianceregionen mit Aktivitäten befüllt werden müssen (Funktionalität 10 in der Liste in Abschnitt 7.1).

7.5. Sidebar-Plugin

Das Sidebar-Plugin ist auf der rechten Seite im Oryx-Editor sichtbar und enthält Prozessfragmente. Diese Prozessfragmente können zur Befüllung von Compliancetemplates verwendet werden. Das Sidebar-Plugin setzt Funktionalität 5 der Liste in Abschnitt 7.1 um. Es zeigt Prozessfragmente an, wenn zwei Bedingungen erfüllt sind: Erstens muss sich der Oryx-Editor im Ableitungs-Modus befinden. Im Ableitungs-Modus können nur Complianceregionen verändert werden. Es kann in diesem Modus keine Änderung am vorliegenden Compliancetemplate durchgeführt werden. Zweitens muss im Oryx-Editor eine Complianceregion markiert sein, welche vom Oryx-Editor zur Ableitung freigegeben ist. Die Auswahl, welche Prozessfragmente angezeigt werden, trifft ein Complianceexperte bei der Erstellung eines Compliancetemplates. Dies geschieht mit dem in Abschnitt 7.4 vorgestellten Variabilitätswizard.

7.6. Ableitungs-Plugin

Diese Oryx-Erweiterung ist für die Befüllung eines Compliancetemplates zuständig. Das Ableitungs-Plugin setzt Funktionalität 2 der Liste in Abschnitt 7.1 um. Es wurde von Weidmann [WKK+11] und Kötter [Köt10] entwickelt und wird als Bestandteil des Prototyps der Voll-

ständigkeit wegen erwähnt. In einem Compliancetemplate kann die Reihenfolge, in der Complianceregionen mit Aktivitäten befüllt werden können, durch Abhängigkeiten zwischen den Complianceregionen vorgegeben sein. Diese Abhängigkeiten wurden bei der Erstellung des Compliancetemplates eingefügt. Ableitungen können mit dem Variabilitäts-Wizard definiert werden. Das Ableitungs-Plugin besteht aus einem Knopf in der oberen Leiste des Oryx-Editors und einem Mechanismus, der die Ableitung, also die Erstellung eines vollständigen Prozesses, leitet. Wird der zum Ableitungs-Plugin gehörende Knopf betätigt, gelangt der Oryx-Editor in den Ableitungs-Modus. In diesem Modus werden diejenigen Complianceregionen grün dargestellt, die zur Befüllung mit Aktivitäten freigegeben sind. Alle anderen Complianceregionen werden rot dargestellt. Befüllt man die zu einem Zeitpunkt freigegebenen Complianceregionen, so können im Zuge dessen weitere Complianceregionen zur Befüllung freigegeben werden, da bestimmte Konditionen erfüllt wurden. Solche Konditionen können zum Beispiel Abhängigkeiten zwischen Complianceregionen, wie in Kapitel 4.2.4 beschrieben, sein.

7.7. LTL-Plugin

Das LTL-Plugin wurde von Stefan Grohe im Zuge seiner Masterarbeit [Gro11] entwickelt. Dies geschah unter Anleitung des Autors der vorliegenden Dissertation. Es erweitert die Funktionalität des Frontends. Das Plugin wird nur geladen, wenn mit dem Editor graphisch eine LTL-Formel bearbeitet wird. Es stellt zum Beispiel eine Funktion bereit, mit der es möglich ist, die graphisch gezeigte LTL-Formel in einer textuellen Repräsentation anzuzeigen (siehe Funktionalität 10 in der

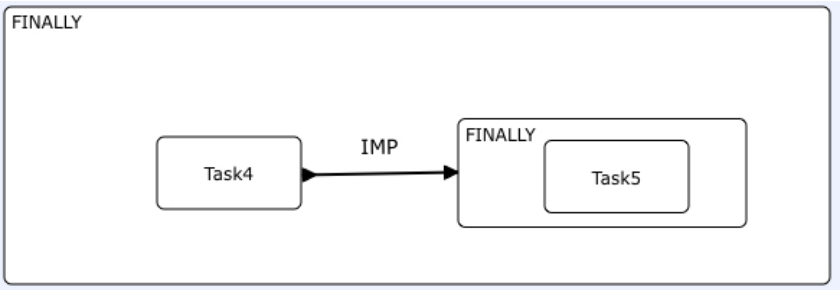

Abbildung 7.5.: Graphische Modellierung von LTL-Formeln

Liste in Abschnitt 7.1). Abbildung 7.5 zeigt die graphisch entwickelte Version der LTL-Formel $\Box(Task4 \Rightarrow \Box(Task5))$.

Abbildung 7.5 zeigt außerdem, (wie auch in Abschnitt 5.5.2 beschrieben) wie Complianceregeln erstellt werden müssen, damit sie im Prototyp eingesetzt werden können. Die Eigenschaften einer LTL-Formel werden durch ihren Namen mit dem BPMN-Task verknüpft, dessen Zustand sie repräsentieren. In dieser Complianceregel spielen Task4 und Task5 eine Rolle. Weiterhin können alle Elemente verwendet werden, die in LTL definiert sind. Abbildung 7.5 zeigt zum Beispiel den Finally-Operator.

Diese graphische Repräsentation einer Complianceregel wird beim Speichern in einen textuellen Ausdruck umgewandelt, der von den an Oryx angeschlossenen Modelcheckern und SAT-Checkern verarbeitet werden kann. Dies wurde, unter Anleitung des Autors dieser Dissertation, in den zwei Diplomarbeiten [Gro11] und [Bur12] umgesetzt. Weiterhin wurde die graphische Notation von LTL aus dieser Quelle entnommen: [BDSV05].

7.8. Complianceservlet

Das Complianceservlet hat zwei Funktionen. Es kann aufgerufen werden, um eine Überprüfung der Complianceregeln eines Prozessmodells anzustoßen. Das Complianceservlet enthält außerdem Funktionalität um ein Prozessmodell zu exportieren. Das Complianceservlet setzt Funktionalität 4 der Liste in Abschnitt 7.1 um.

Wird im Complianceservlet die Funktion zur Überprüfung eines Prozessmodells aufgerufen, so wird das als Parameter übergebene Prozessmodell zunächst in ein Petrinetz-Modell überführt. Dieses Petrinetz-Modell bildet den Kontrollfluss des originalen Prozessmodells ab. Im nächsten Schritt wird das Petrinetz-Modell in die Eingabesprache PROMELA [RMF07] des SPIN Modelcheckers überführt. Zusammen mit den mit dem Prozessmodell verknüpften Complianceregeln ist es die Aufgabe von SPIN das Prozessmodell auf Verletzungen der Complianceregeln zu überprüfen. Konnte SPIN eine Verletzung von Complianceregeln erkennen, so wird eine Beschreibung, welcher Ausführungspfad zu dieser Verletzung führte, von SPIN an das Complianceservlet zurückgegeben. Das Complianceservlet gibt diese Information wiederum an den Oryx-Editor zurück, der sie dem Benutzer anzeigt.

7.9. LTL-Servlet

Das LTL-Servlet implementiert Funktionalität, um LTL-Formeln, die mit dem graphischen Formeleditor erstellt wurden, in eine textuelle Repräsentation umzuwandeln (siehe Funktionalität 10 in der Liste in Abschnitt 7.1). Das LTL-Servlet kann LTL-Formeln in zwei verschiedene Repräsentationen umwandeln. Die erste Repräsentation ist für die Ver-

wendung mit dem SPIN Modelchecker, die andere für die Verwendung mit dem Maude Modelchecker geeignet.

7.10. Compliancechecker

Das Compliancechecker-Plugin ist dafür zuständig, verschiedenartige Complianceregeln automatisch zu überprüften. Der Compliancechecker setzt Funktionalität 4, 6, 7, 8 und 10 der Liste in Abschnitt 7.1 um. Weiterhin enthält es Funktionalität für die Transformation von Oryx-Prozessmodellen in die Eingabesprachen dieser beiden Modelchecker.

Der SPIN Modelchecker wird vom Compliancechecker-Plugin aufgerufen, wenn eine Überprüfung eines Prozessmodells auf Verletzungen von Complianceregeln angestoßen wurde. Der Maude Modelchecker dient zur Überprüfung der Erfüllbarkeit von LTL-Formeln und wird entweder parallel zum Aufruf von SPIN oder für eine Überprüfung der Erfüllbarkeit einer LTL-Formel aufgerufen.

Abbildung 7.6 zeigt das Überprüfungsergebnis einer Complianceprüfung zweier ineinander geschachtelter Compliancescopes, das dem Benutzer präsentiert wird. Der Überprüfungsmechanismus, der diesem Ergebnis zugrunde liegt wird, in Kapitel 6.3 beschrieben. Die mit diesen Compliancescopes verknüpften Complianceregeln sind:

- **Compliancescope 1**: $\Diamond Task2 \wedge \Diamond Task3$
- **Compliancescope 2**: $\neg\Diamond Task3$

Diese Ergebnisdarstellung wurde unter Anleitung des Autors dieser Dissertation von Alexej Burkow in seiner Masterarbeit [Bur12] umgesetzt. Alle Compliancescopes, deren Complianceregeln verletzt sind,

werden rot dargestellt. Alle Compliancescopes, deren Complianceregeln nicht verletzt sind, werden grün dargestellt.

Die Abbildung zeigt unten ein detailliertes Ergebnis der Complianceprüfung. Für jeden untersuchten Compliancescope enthält diese Ergebnisdarstellung einen Reiter. Auf diesem Reiter wird gezeigt, in welchen Schritten die mit dem Compliancescope verknüpften Complianceregeln überprüft wurden. Die markierte Stelle in dieser Abbildung zeigt die Complianceregel, die für Compliancescope 2 gilt. Diese besagt, dass eine Aktivität mit dem Namen Task3 nicht auftreten darf. Da dies aber in Compliancescope 2 der Fall ist, wird dieser rötlich als nicht erfüllt gekennzeichnet.

Abbildung 7.7 zeigt bis auf eine Änderung denselben Prozess wie Abbildung 7.6. Diese Änderung ist die Umbenennung von Task3 in Task4. Diese Umbenennung führt dazu, dass der zweite Teil der mit dem Compliancescope 1 verknüpften Complianceregel nicht erfüllt wird, da weder in diesem Compliancescope noch in einem in ihm enthaltenen Compliancescope eine Aktivität mit dem Namen Task3 enthalten ist. Weiter zeigt die Abbildung das Ergebnis der Überprüfung von Compliancescope 1. Markiert ist die mit diesem Compliancescope verknüpfte Complianceregel. Der erste Teil dieser Complianceregel ($\Diamond Task2$) wird durch das Vorhandensein einer Aktivität mit dem Namen Taks2 in Compliancescope 1 erfüllt. Der zweite Teil dieser Complianceregel wird in Compliancescope 1 nicht erfüllt. Deshalb wird dieser Teil an Compliancescope 2 weitergegeben. Dies ist in Abbildung 7.7 in der vorletzten Zeile des Prüfergebnisses von Compliancescope 1 ersichtlich.

Abbildung 7.8 zeigt das Überprüfungsergebnis von Compliancescope 2. Es wurde von dem Programm festgestellt, dass die mit

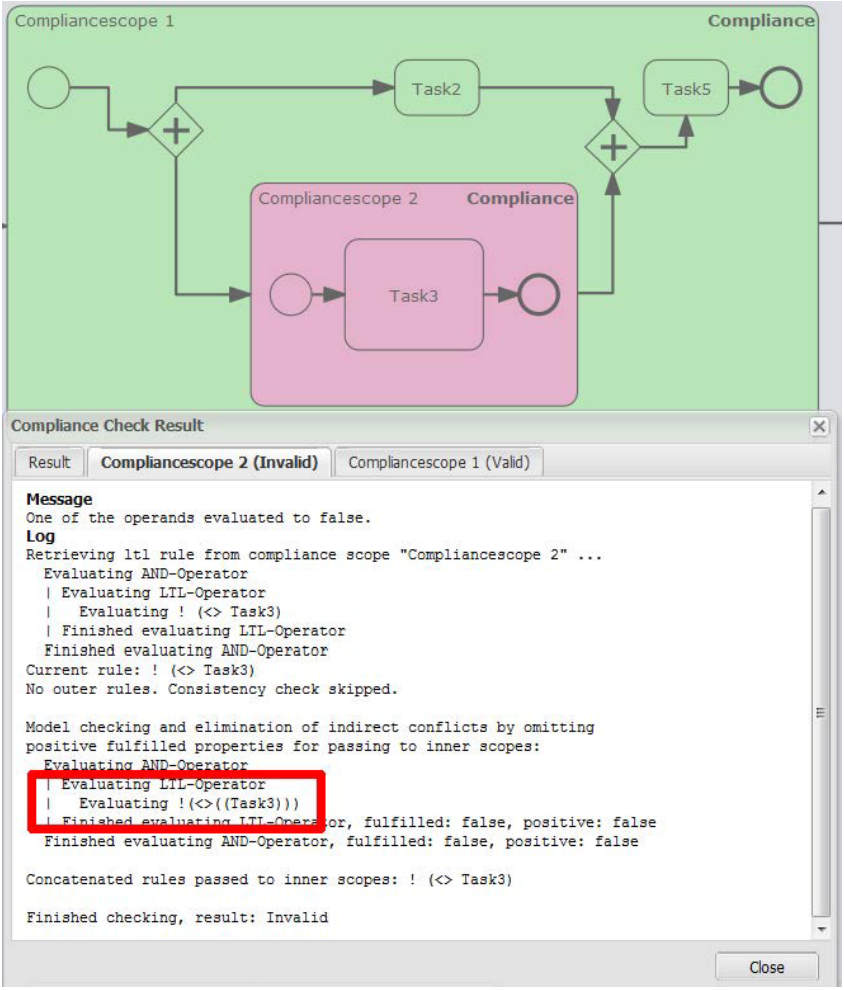

Abbildung 7.6.: Anzeige des Überprüfungsergebnisses bei geschachtelten Compliancescopes

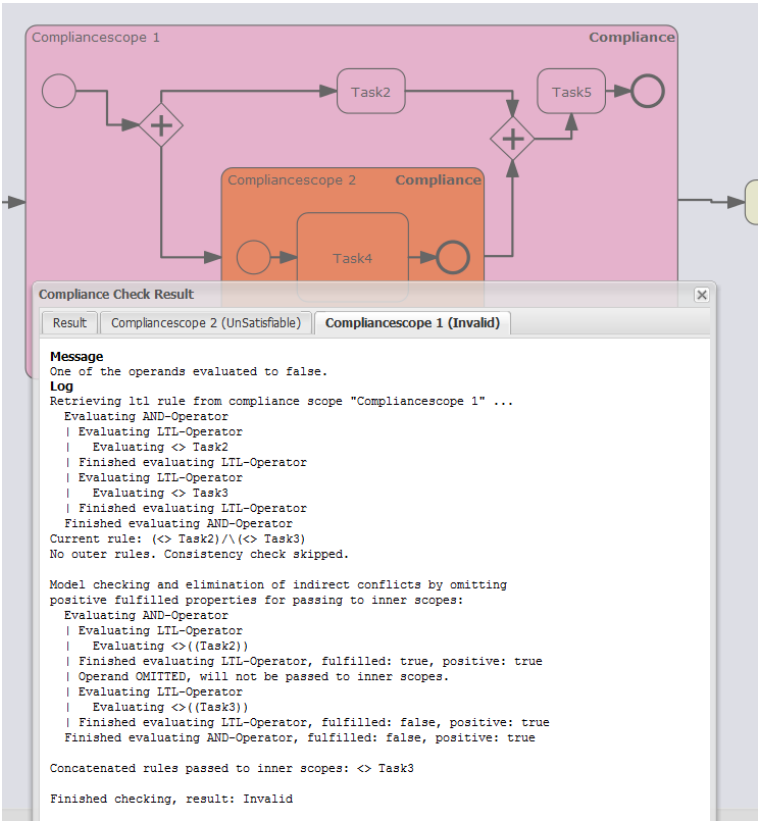

Abbildung 7.7.: Anzeige des Überprüfungsergebnisses bei geschachtelten Compliancescopes: Erfüllung des ersten Teils der mit Compliancescope 1 verknüpften Complianceregel

Compliancescope 2 verknüpfte Complianceregel zusammen mit der weitergegebenen Complianceregel von Compliancescope 1 nicht erfüllbar ist. In der Abbildung ist diese verbundene unerfüllbare Complianceregel markiert. Diese Situation trat ein, da die mit Compliancescope 1 verknüpfte Complianceregel nicht durch Compliancescope 1 erfüllt werden konnte. Diese Complianceregel hätte mit dem Vorhandensein einer Aktivität mit dem Namen Task3 erfüllt werden können. Die Complianceregel wurde dadurch an Compliancescope 2 weitergereicht. Dies führte zu der in Abbildung 7.8 markierten konkatenierten unerfüllbaren Complianceregel.

Datengetriebene Complianceregeln werden in der Komponente *Compliancechecker* ebenfalls überprüft. Der Compliancechecker setzt Funktionalität 4 der Liste in Abschnitt 7.1 um. Es wird dafür der in Kapitel 5.3 erläuterte Algorithmus verwendet. Dieser Algorithmus wurde erstmals in [SFG+11] präsentiert und in [Gro11] umgesetzt.

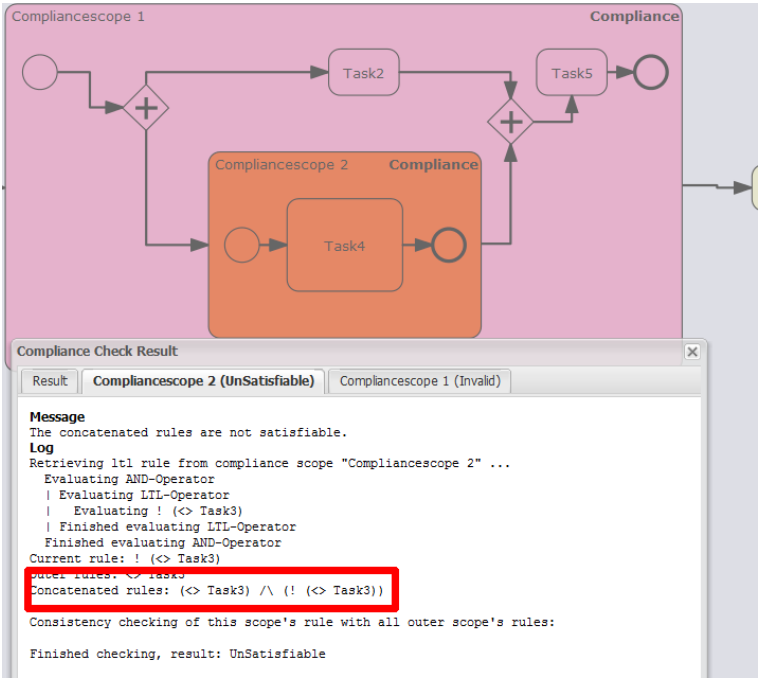

Abbildung 7.8.: Anzeige des Überprüfungsergebnisses geschachtelter Compliancescopes: Unerfüllbarkeit weitergereichter Complianceregeln

7.11. Performanzmessungen

Das Ziel dieser Arbeit besteht in der Konzeption und Implementierung von benutzerfreundlichen Konzepten zur automatischen Überprüfung von Prozessmodellen. Ein Aspekt, der Benutzerfreundlichkeit ausweist, besteht in kurzen Antwortzeiten von Programmen. Ein Hauptfaktor für die Laufzeit einer Überprüfung eines Prozessmodells liegt in der

Parallele Zweige	8	14	16	18	19	20
Laufzeit in ms	1.362	2.343	6.259	26.337	54.188	124.849

Tabelle 7.2.: Messergebnisse der Laufzeiten (in Millisekunden) von Complianceuntersuchungen eines Prozessmodells mit parallelen Zweigen [Gro11].

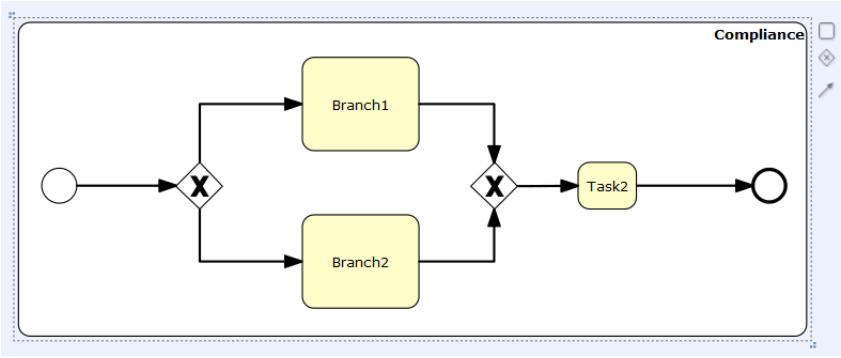

Abbildung 7.9.: Prozessmodell mit dem die Geschwindigkeitsuntersuchungen durchgeführt wurden.

Laufzeit des aufgerufenen Modelcheckers begründet. Diese liegt im PSPACE-vollständigen Bereich [SC85].

Tabelle 7.2 zeigt Geschwindigkeitsmessungen der Überprüfung eines Test-Prozessmodells (siehe Abbildung 7.9). Dieses Test-Prozessmodell besteht aus einem Startereignis, auf der linken Seite, und einem Endereignis, auf der rechten. Dazwischen teilt ein paralleles Gateway den Kontrollfluss des Prozessmodells in zwei parallel laufende Kontrollflussstränge auf. Ein zweites, paralleles Gateway führt diese beiden Kontrollflussstränge wieder zusammen. Bevor das Endereignis erreicht

wird, wird Task 2 ausgeführt.

Für den Geschwindigkeitstest wurde die LTL-Formel $\Diamond Task2$ mit dem Prozessmodell verknüpft. Diese LTL-Formel besagt, dass bei jeder Ausführung dieses Prozesses der Task 2 ausgeführt werden muss. Dieses Prozessmodell wurde in Oryx geladen und auf Verletzungen von Complianceregeln mit Oryx untersucht. Dabei wurden für die Durchführung der Geschwindigkeitstests die Anzahl der parallelen Pfade wie in Tabelle 7.2 gezeigt erhöht. In der Diplomarbeit von Stefan Grohe [Gro11] sind die Testbedingungen im Detail erläutert. Tabelle 7.2 zeigt weiterhin das exponentielle Wachstum der Ausführungszeit eines solchen Compliancechecks mit Oryx. Dieses Wachstum ist auf die Laufzeit von Modelcheckern zurückzuführen, welche PSPACE-Vollständig ist. Um lange Wartezeiten auf das Ergebnis eines Compliancechecks zu vermeiden, besteht die Möglichkeit, die Untersuchung auf Teile des Prozessmodells zu beschränken. Dies ist mit dem in dieser Arbeit vorgestellten Konzept der Compliancescopes möglich, da es der Prototyp zulässt, die Inhalte bestimmter Compliancescopes zu überprüfen und nicht das gesamte Prozessmodell. Compliancescopes sind somit notwendig, um die Benutzbarkeit des Prototyps zu erhöhen, indem lange Wartezeiten auf Prüfergebnisse vermieden werden.

Ein weiteres Mittel, um die Antwortzeiten und somit die Benutzbarkeit des Prototyps zu erhöhen, ist Caching. Zum Beispiel werden im Prototyp LTL-Formeln, die in das Eingabeformat für den Modelchecker SPIN transformiert wurden, für eine spätere Verwendung aufbewahrt. Damit entfällt bei einer erneuten Überprüfung eines Compliancescopes der Schritt der Transformation der dazugehörigen Complianceregel in das Eingabeformat von SPIN, wenn dieser Compliancescope zuvor schon überprüft wurde.

7.12. Zusammenfassung

Dieses Kapitel zeigt die Architektur des Prototyps, der die in dieser Arbeit vorgestellten Konzepte und Lösungen implementiert. Es zeigt, welche Erweiterungen hierfür am webbasierten BPMN Editor Oryx vorgenommen wurden. Besonders zu erwähnen sind hier das Ableitungs-Plugin, welches das Konzept der Compliancetemplates umsetzt, das Sidebar-Plugin, welches dazu dient, Prozessfragmente in Prozesse einzufügen, der Compliancewizard, der die Funktionalität implementiert, um Compliancescopes und Complianceregionen mit Complianceregeln zu verknüpfen und der Compliancechecker, der die automatische Überprüfung von Prozessmodellen mittels Modelcheckern durchführt. Die Erweiterungen sind in einem Architekturdiagramm aufgeführt und miteinander in Verbindung gesetzt. Das Kapitel enthält eine detaillierte Beschreibung jeder Erweiterung.

Ein wichtiges Kriterium für die Benutzbarkeit des Prototyps ist die Laufzeit bis das Ergebnis der Überprüfung einer Complianceregel angezeigt werden kann. Im letzten Teil dieses Kapitels sind Geschwindigkeitsmessungen für unterschiedlich große Prozessmodelle beschrieben, da die Laufzeit der Compliancеprüfung von der Anzahl der Knoten im Prozess abhängt.

KAPITEL 8

ZUSAMMENFASSUNG UND AUSBLICK

Diese Dissertation befasst sich mit der Unterstützung von Prozessmodellierern bei der Entwicklung regelkonformer Geschäftsprozesse. Gezeigt werden eine Erweiterung eines Variabilitätskonzepts für die Unterstützung der Entwicklung regelkonformer Prozesse (siehe Beitrag 1.4.1), ein Algorithmus zur Überprüfung des Kontrollflusses von Prozessmodellen (siehe Beitrag 1.4.2), ein Algorithmus zur Überprüfung des Datenflusses in Prozessmodellen (siehe Beitrag 1.4.3), ein Mechanismus zur Zusammenarbeit bei der Erstellung regelkonformer Prozesse (siehe Beitrag 1.4.4) und die Architektur eines Prototyps zur Verifikation der vorgestellten Konzepte und Algorithmen (siehe Beitrag 1.4.5). Diese Konzepte können zur Entwicklungszeit angewendet werden, um Geschäftsprozesse automatisch auf Verletzungen von

Complianceregeln zu überprüfen. Die zwei Anwendungsfälle, die diese Konzepte, wie im Folgenden beschrieben, abdecken sind:

- die Entwicklung eines von Grund auf neuen Geschäftsprozesses
- die Wartung eines bestehenden Geschäftsprozesses.

8.1. Anwendungsgebiet der Dissertation

Im ersten Anwendungsfall beginnt der menschliche Prozessmodellierer auf Grundlage eines Compliancetemplates mit der Entwicklung eines neuen Prozesses. Solche Compliancetemplates können für verschiedene Anwendungsfälle in einer Firma vorhanden sein. Zum Beispiel könnte ein Compliancetemplate in einer Bank für die Entwicklung von Kreditantragsprozessen bereits bestehen.

Compliancetemplates implementieren bestimmte Complianceregeln, die für alle Prozesse gelten müssen, für die sie die Grundlage bilden. Bei einem Kreditantragsprozess könnte dies die Einhaltung des Vier-Augen-Prinzips sein. Das Vier-Augen-Prinzip besagt, dass zwei unterschiedliche Personen einen Kreditantrag prüfen müssen.

Bei der Erstellung eines neuen Geschäftsprozesses wird der menschliche Prozessmodellierer von einem graphischen Entwicklungswerkzeug unterstützt. Dieses Entwicklungswerkzeug stellt sicher, dass die von einem Compliancetemplate implementierten Complianceregeln nicht durch Modifikationen umgangen werden können.

Compliancetemplates enthalten unter anderem Complianceregionen. Sie sind die Orte, an denen Modifikationen durchgeführt werden dürfen. Das Compliancetemplate kann durch Befüllen der Complianceregionen mit Aktivitäten vollständig gemacht werden.

Im zweiten Anwendungsfall wird ein bestehender Prozess vor der Wartung durch einen menschlichen Prozessmodellierer mit Compliancescopes versehen. Durchgeführt wird dies von einem Complianceexperten. Compliancescopes stellen Bereiche in einem Prozessmodell dar, die mit Complianceregeln verknüpft sind. Prozesse, die mit Compliancescopes versehen sind, können automatisch auf Verletzungen von Complianceregeln überprüft werden. In dieser Dissertation wird dies, wie auch im vorhergehenden Anwendungsfall, mit Techniken des Modelchecking bewerkstelligt. Die in diesem Zusammenhang neu erstellten und weiterentwickelten Konzepte, *Compliancetemplate*, *Compliancescope* und *Vervollständigungsebenen* können in einem graphischen Entwicklungswerkzeug realisiert werden, wie dies im Rahmen dieser Dissertation geschehen ist.

Im Zusammenhang mit den oben vorgestellten Anwendungsfällen wurde untersucht, welche Arten von Complianceregeln für die Prozessentwicklung von Bedeutung sind. Dies sind kontrollflussbasierte und datenflussbasierte Complianceregeln. Bei der Arbeit mit Compliancetemplates werden kontrollflussbasierte Complianceregeln verwendet. Bei der Arbeit mit Compliancedomains werden datenflussbasierte Complianceregeln verwendet.

Abschließend wurde ein Mechanismus vorgestellt, die verdeutlicht, wie die eingangs vorgestellten Konzepte in einer Organisation umgesetzt werden können. Darüber hinaus wurden Rollen definiert, die in Unternehmen eingeführt werden müssen, damit die dargestellten Konzepte umgesetzt werden können.

Weiterhin wurde ein Konzept entwickelt, das die Zusammenarbeit bei der Erstellung eines Prozessmodells zwischen verschiedenen Abteilungen einer Firma ermöglicht. Dieses Konzept arbeitet mit Complian-

cetemplates. Es beschreibt die Vervollständigung eines Compliancetemplates zu einem syntaktisch korrekten Prozess auf verschiedenen Ebenen. Ein Compliancetemplate stellt die erste Ebene bei dieser Art der Vervollständigung dar. Weitere Ebenen können durch Einfügen von Complianceregionen in das Compliancetemplate erstellt werden. Sind alle Complianceregionen bei der Vervollständigung mit Aktivitäten befüllt, ist der Prozess vollständig.

Die Arbeit an dem Thema der regelkonformen Prozessmodellierung hat weiterreichende Fragen aufgeworfen, die im Folgenden dargestellt werden.

Eine Frage, die mit Abschluss dieser Arbeit offen bleibt, ist, in wieweit durch Complianceregeln die Möglichkeiten der Prozessmodellierung eingeschränkt werden solle. Im ersten Extrem arbeitete man gänzlich ohne gesondert definierte Complianceregeln. Hier würde man alle Complianceregeln im Prozessmodell durch strukturierende Prozesskonstrukte umsetzen. Dies können zum Beispiel Pfeile zwischen Aktivitäten sein. Im anderen Extrem würde ein Prozessmodell nur durch Complianceregeln definiert werden. Dies streift das Feld der deklarativen Prozessmodellierung. In diesem Feld des BPM werden Prozesse anhand von Anforderungen modelliert. Durch diese Anforderungen wird implizit eine Abfolge der im Prozessmodell enthaltenen Aktivitäten definiert. Diese Anforderungen können sehr eng gefasst sein, so dass der Prozess nur durch sie definiert ist. Complianceregeln stehen in einem engen Zusammenhang mit diesen Anforderungen der deklarativen Prozessmodellierung. Mit Complianceregeln ist es, wie auch mit den Anforderungen möglich, die Ausführungsreihenfolge der in einem Prozessmodell enthaltenen Aktivitäten zu bestimmen. Aus diesen Überlegungen lassen sich weitere Forschungsfragen ableiten.

Hierbei geht es erstens darum in wieweit sich Complianceregeln und Anforderungen für die deklarative Modellierung von Prozessen ähneln. Ist diese Frage geklärt muss überlegt werden, in wieweit Anforderungen und Complianceregeln vermischt werden können. Es liegt nahe, dass der Mittelweg zwischen diesen beiden Extremen die beste Vorgehensweise bei der Erstellung regelkonformer Prozesse ist. Dies muss jedoch wissenschaftlich untersucht werden.

8.2. Ausblick

Der in dieser Dissertation gezeigte Ansatz zur automatischen Überprüfung von Complianceregeln ist nicht nur für die Überprüfung von Complianceregeln geeignet. Er kann auf weitere Gebiete wie zum Beispiel das Green Business Process Management oder Datensicherheit ausgeweitet werden. Hier ist es denkbar, dass Aktivitäten mit einer Umweltverträglichkeitszahl verknüpft werden. Damit kann automatisch bestimmt werden, wie umweltverträglich der Gesamtprozess ist. Die in der vorliegenden Arbeit vorgestellten Konzepte Compliancetemplate und Compliancescope sind allgemein ausgelegt und können mit dieser Anforderung umgehen. Es müssen jedoch geeignete Prüfwerkzeuge entwickelt werden, die die in der vorliegenden Dissertation gezeigten Konzepte umsetzten.

Das Konzept der Compliancedomains wurde in dieser Dissertation aus Sicht der Entwicklungszeit eines Prozesses entworfen. Es kann auch in Richtung der Ausführung eines Geschäftsprozesses ausgeweitet werden. Dies wird im folgenden Abschnitt gezeigt.

Das Konzept der Compliancedomains befasst sich mit der Einschränkung des Datenflusses in einem Prozess. Es ist aus der Überlegung

heraus entstanden, dass es für Organisationen verboten sein kann, Daten an einen bestimmten Ort zu transferieren. Somit grenzen Compliancedomains den Bereich ein, in dem bestimmte Daten verarbeitet werden können. Diese Eingrenzung muss zur Laufzeit eines Prozesses auch gelten und überprüft werden. In [SFG+11] wird ein Konzept vorgestellt, das Laufzeitprüfungen des Datenflusses eines Prozesses beschreibt. Services werden mit einer Annotation verknüpft, die beschreibt, an welchem physikalischen Ort sich der Service befindet. Weiterhin zeigt das Konzept, wie Softwarekomponenten, die sich an der Grenze einer physikalischen Umgebung befinden, Nachrichten untersuchen und anhand von Complianceregeln entscheiden, ob eine Nachricht die Grenze überschreiten darf. Diese Complianceregeln werden zur Entwicklungszeit mit Compliancedomains verknüpft und zur Laufzeit an die Softwarekomponenten weitergegeben.

Der ABIS Ansatz [WKK+11] kann mit den in dieser Dissertation vorgestellten Konzepten erweitert werden. Im ABIS Ansatz wird beschrieben, wie Geschäftsprozesse auf verschiedenen Komplexitätsebenen modelliert werden können. Außerdem zeigt der Ansatz, wie Änderungen zwischen diesen Komplexitätsebenen weitergereicht werden können. Es ist denkbar, dass mit diesen weitergereichten Änderungen auch Complianceregeln zwischen den Komplexitätsebenen weitergereicht werden können. Im von der Deutschen Forschungsgesellschaft (DFG) geförderten Projekt *Konzepte und Methoden zur Unterstützung von Fachanwendern bei der Umsetzung von Adaptivität und Compliance-Richtlinien in Geschäftsprozessen* [LS13] wird auf den Forschungsergebnissen der vorliegenden Arbeit aufgebaut. Unter anderem werden in diesem Forschungsprojekt die Möglichkeiten einer übergeordneten Compliancesprache ergründet.

Diese Dissertation befasst sich ausschließlich mit Complianceregeln, die sich auf die Syntax in einem Prozess beziehen. Zum Beispiel kann eine Complianceregel die Ausführungsreihenfolge zweier Aktivitäten in einem Prozess vorschreiben. Es kann jedoch keine Complianceregel erstellt werden, die verlangt, dass in einem Prozess das Vier-Augen-Prinzip umgesetzt werden muss. Eine solche Complianceregel muss zunächst in einen logischen Ausdruck überführt werden, der dann automatisch verarbeitet werden kann. In Arbeiten, die auf dieser Dissertation aufbauen, könnten neue Überprüfungskonzepte erstellt werden, die mit Ontologien arbeiten, um Verletzungen semantischer Complianceregeln aufzudecken.

Ein weiteres Themengebiet für zukünftige Arbeiten stellt die Prozessübergreifende Überprüfung von Complianceregeln dar. In dieser Dissertation gilt die Annahme, dass Complianceregeln nur im Rahmen des ihnen zugewiesenen Prozesses gelten. Es liegt jedoch nahe, dass Anwendungen aus mehreren Prozessen bestehen, die sich gegenseitig aufrufen und miteinander interagieren. Dies muss in den zukünftigen Konzepten zur Überprüfung von Complianceregeln beachtet werden.

LITERATURVERZEICHNIS

[ADW08] AWAD, Ahmed ; DECKER, Gero ; WESKE, Mathias: Efficient compliance checking using BPMN-Q and temporal logic. In: *BPM '08 Proceedings of the 6th International Conference on Business Process Management*, 2008, S. 326–341

[AKL+09] ANSTETT, Tobias ; KARASTOYANOVA, Dimka ; LEYMANN, Frank ; MIETZNER, Ralph ; MONAKOVA, Ganna ; SCHLEICHER, Daniel ; STRAUCH, Steve: MC-Cube: Mastering customizable compliance in the cloud. In: SPRINGER (Hrsg.): *Proceedings of the 7th International Joint Conference on Service Oriented Computing*, Springer Verlag, November 2009, 592-606

[Arb04] ARBAB, Farhad: Reo: a channel-based coordination model for component composition. In: *Mathematical. Structures in Comp. Sci.* 14 (2004), Juni, Nr. 3, 329–366. http://dx.doi.org/10.1017/S0960129504004153. – DOI 10.1017/S0960129504004153. – ISSN 0960–1295

[ASU86] AHO, A.V. ; SETHI, R. ; ULLMAN, J.D.: *Compilers: principles, techniques, and tools*. Addison-Wesley, 1986 (Addison-Wesley series in computer science). http://books.google.de/books?id=-CpTewAACAAJ. – ISBN 9780201100884

[AW09] AWAD, Ahmed ; WESKE, Mathias: Visualization of compliance violation using anti-patterns / Business Process Technology Group at Hasso Platter Institute at the University of Potsdam. Version: 2009. http://bpt.hpi.uni-potsdam.de/pub/Public/BptPublications/VoV.pdf. 2009 (BPT Technical Report 02-2009 02-2009). – Forschungsbericht

[Awa10] AWAD, Ahmed Mahmoud Hany A.: *A compliance management rramework for business process models*, Business Process Technology Group, Hasso Plattner Institute, University Potsdam, Diss., 2010

[AWW09] AWAD, Ahmed ; WEIDLICH, Matthias ; WESKE, Mathias: Specification, verification and explanation of violation for data aware compliance rules. In: *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*. Berlin, Heidelberg : Springer-Verlag, 2009 (ICSOC-ServiceWave '09). – ISBN 978–3–642–10382–7, 500–515

[Bas06] BASEL COMMITTEE ON BANKING SUPERVISION: Basel II capital accord: international convergence of capital measurements and capital standards: A revised framework

(comprehensive version) / Basel Committee on Banking Supervision. Version: Juni 2006. http://www.bis.org/publ/bcbs128.pdf. 2006. – Forschungsbericht. – ISBN 92–9197–720–9

[BBC+07] Boag, Scott ; Berglund, Anders ; Chamberlin, Don ; Siméon, Jérôme ; Kay, Michael ; Robie, Jonathan ; Fernández, Mary F.: XML Path Language (XPath) 2.0 / W3C. 2007. – W3C Recommendation. – http://www.w3.org/TR/2007/REC-xpath20-20070123/

[BBD+11] Becker, Jörg ; Bergener, Philipp ; Delfmann, Patrick ; Eggert, Mathias ; Weiss, Burkhard: Supporting business process compliance in financial institutions - A Model-Driven Approach. In: *Wirtschaftinformatik Proceedings 2011*, 2011

[BDSV05] Brambilla, Marco ; Deutsch, Alin ; Sui, Liying ; Vianu, Victor: The role of visual tools in a web application design and verification framework: A visual notation for LTL formulae. In: *ICWE*, 2005, S. 557–568

[Ber89] Berge, Claude (Hrsg.): *Hypergraphs combinatorics of finite sets*. Bd. 45. Elsevier, 1989. http://dx.doi.org/DOI:10.1016/S0924-6509(08)70093-X. http://dx.doi.org/DOI:10.1016/S0924-6509(08)70093-X. ISSN 0924–6509

[Boe87] Boehm, Barry W.: Improving software productivity. In:

Computer 20 (1987), September, Nr. 9, 43–57. http://dx.doi.org/10.1109/MC.1987.1663694. – DOI 10.1109/MC.1987.1663694. – ISSN 0018–9162

[Bro95] BROOKS, Frederick P. Jr.: *The mythical man-month (anniversary ed.)*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–83595–9

[Bur12] BURKOW, Alexej: *LTL- Erfüllbarkeitsprüfung für inkrementelle Entwicklung von Geschäftsprozessen*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Masterarbeit, August 2012. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=MSTR-3386&engl=0. – 103 S.

[Bus04] BUSINESS PROCESS MASNAGEMENT INITATIVE: *Business process modeling notation (BPMN) version 1.0*, Mai 2004. – http://www.bpmn.org/

[CCG+02] CIMATTI, A. ; CLARKE, E. ; GIUNCHIGLIA, E. ; GIUNCHIGLIA, F. ; PISTORE, M. ; ROVERI, M. ; SEBASTIANI, R. ; TACCHELLA, A.: NuSMV version 2: an openSource tool for symbolic model checking. In: *Proc. International Conference on Computer-Aided Verification (CAV 2002)* Bd. 2404. Copenhagen, Denmark : Springer, July 2002 (LNCS)

[CGP01] CLARKE, Edmund M. ; GRUMBERG, Orna ; PELED, Doron: *Model checking*. Cambridge, Mass. : MIT Press, 2001. – ISBN 0262032708

[Chi03] China: *Law of the people's republic of China on the people's bank of China*. http://www.china.org.cn/business/laws_regulations/2007-06/22/content_1214826.htm, 2003

[DAC98] Dwyer, Matthew B. ; Avrunin, George S. ; Corbett, James C.: Property specification patterns for finite-state verification. In: *FMSP*, 1998, S. 7–15

[DAC99] Dwyer, Matthew B. ; Avrunin, George S. ; Corbett, James C.: Patterns in property specifications for finite-state verification. In: *Proceedings of the 21st international conference on Software engineering - ICSE '99* (1999), 411–420. http://dx.doi.org/10.1145/302405.302672. – DOI 10.1145/302405.302672. ISBN 1581130740

[DCD+09] Daniel, Florian ; Casati, Fabio ; D'Andrea, Vincenzo ; Strauch, Steve ; Schumm, David ; Leymann, Frank ; Mulo, Emmanuel ; Zdun, Uwe ; Dustdar, Schahram ; Sebahi, Samir ; Marchi, Fabien de ; Hacid, Mohand-Said: Business compliance governance in service-oriented architectures. In: Awan, Irfan (Hrsg.) ; Younas, Muhammad (Hrsg.) ; Hara, Takahiro (Hrsg.) ; Durresi, Arjan (Hrsg.): *Proceedings of the IEEE Twenty-Third International Conference on Advanced Information Networking and Applications (AINA'09), Bradford, United Kingdom, May 26-29, 2009*, IEEE Press, Mai 2009. – ISBN 978–1–4244–4000–9, 113–120

[DDO08] Dijkman, Remco M. ; Dumas, Marlon ; Ouyang, Chun: *Formal semantics and analysis of BPMN process models using petri nets*. 2008

[DOW08] Decker, Gero ; Overdick, Hagen ; Weske, Mathias: Oryx – an open modeling platform for the BPM community. In: *BPM '08 Proceedings of the 6th International Conference on Business Process Management* Bd. 5240, Springer, 2008 (LNCS). – ISBN 978–3–540–85757–0

[Elg12] Elgammal, A.F.S.A.: *Towards a comprehensive framework for business process compliance*. http://ideas.repec.org/p/ner/tilbur/urnnbnnlui12-5470311.html, 2012

[ETHP10] Elgammal, Amal ; Türetken, Oktay ; Heuvel, Willem-Jan van d. ; Papazoglou, Mike P.: On the formal specification of regulatory compliance: a comparative analysis. In: *ICSOC Workshops*, 2010, S. 27–38

[EUL09] Eberle, Hanna ; Unger, Tobias ; Leymann, Frank: Process fragments. In: Meersman, Robert (Hrsg.) ; Dillon, Tharam (Hrsg.) ; Herrero, Pilar (Hrsg.): *On the Move to Meaningful Internet Systems: OTM 2009* Bd. 5870, Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–05147–0, S. 398–405

[GCS+10] Gheorghe, Gabriela ; Crispo, Bruno ; Schleicher, Daniel ; Anstett, Tobias ; Leymann, Frank ; Mietzner, Ralph ; Monakova, Ganna: Combining enforcement strategies

in service oriented architectures. In: *ICSOC 2010 proceedings*, Springer, Dezember 2010, 288–302

[Ger97] GERTH, Rob: *Concise promela reference*. http://spinroot.com/spin/Man/Quick.html, 1997

[GM06] GOVERNATORI, Guido ; MILOSEVIC, Zoran: A formal analysis of a business contract language. In: *Int. J. Cooperative Inf. Syst.* 15 (2006), Nr. 4, 659-685. http://dblp.uni-trier.de/db/journals/ijcis/ijcis15.html#GovernatoriM06

[Gov08] GOVERNATORI, Guido: The journey to business process compliance. In: *Public Law* (2008), S. 1–32

[Gro11] GROHE, Stefan: *Visualisierung und Implementierung von Compliance Scopes*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Diplomarbeit, Mai 2011. – 98 S.

[Haw11] HAWRANEK, Dietmar: *Trapped in the US web: daimler upset with over-eager american oversight*. http://www.spiegel.de/international/business/trapped-in-the-us-web-daimler-upset-with-over-eager-american-oversight-a-803350-2.html, December 2011

[HM10] HARDER, Bernd H. ; MARUHN, Ralf: *Auftragsdatenverarbeitung*. http://www.cloud-practice.de/know-how/auftragsdatenverarbeitung, 2010

[Hol03] Holzmann, Gerard: *Spin model checker, the: primer and reference manual*. First. Addison-Wesley Professional, 2003. – ISBN 0–321–22862–6

[HWG09] Hoffmann, Jörg ; Weber, Ingo ; Governatori, Guido: On compliance checking for clausal constraints in annotated process models. In: *Information Systems Frontiers* (2009), Mai. http://dx.doi.org/10.1007/s10796-009-9179-7. – DOI 10.1007/s10796–009–9179–7. – ISSN 1387–3326

[JGP99] Jr., Edmund M. C. ; Grumberg, Orna ; Peled, Doron A.: *Model checking*. The MIT Press, 1999 http://www.amazon.com/Model-Checking-Edmund-Clarke-Jr/dp/0262032708%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0262032708. – ISBN 0262032708

[KBE+10a] Kessler, Dr. A. ; Böhm, Dr. M. ; Ermold, Eiko ; Wehran, Heino ; Vehlow, Markus: *Cloud compliance*. http://www.cloud-practice.de/know-how/cloud-compliance, 2010

[KBE+10b] Kessler, Dr. A. ; Böhm, Dr. M. ; Ermold, Eiko ; Wehran, Heino ; Vehlow, Markus: *Cloud Compliance - Motive, Herausforderungen und Hürden*. http://www.cloud-practice.de/know-how/cloud-compliance-motive-herausforderungen-und-huerden, 2010

[Kha08] KHALAF, Rania: *Supporting business process fragmentation while maintaining operational semantics : a BPEL perspective*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Dissertation, März 2008. – 193 S.

[KL06] KHALAF, Rania ; LEYMANN, Frank: Role-based decomposition of business processes using BPEL. In: *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2006 (ICWS '06). – ISBN 0–7695–2669–1, 770–780

[KLRM+10] KNUPLESCH, David ; LY, Linh T. ; RINDERLE-MA, Stefanie ; PFEIFER, Holger ; DADAM, Peter: On enabling data-aware compliance checking of business process models. In: *Proceedings of the 29th international conference on Conceptual modeling*, 2010

[KNP02] KWIATKOWSKA, M. ; NORMAN, G. ; PARKER, D.: PRISM: probabilistic symbolic model checker. In: FIELD, T. (Hrsg.) ; HARRISON, P. (Hrsg.) ; BRADLEY, J. (Hrsg.) ; HARDER, U. (Hrsg.): *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)* Bd. 2324, Springer, 2002 (LNCS), S. 200–204

[Köt10] KÖTTER, Falko: *Prozessvarianten in unternehmensübergreifenden Servicenetzwerken*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Diplomarbeit, November

2010. http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-3046&engl=0. – 119 S.

[KSMP07] KHARBILI, Marwane E. ; STEIN, Sebastian ; MARKOVIC, Ivan ; PULVERMÜLLER, Elke: *Towards a Framework for Semantic Business Process Compliance Management*. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.9939. Version: 2007

[KWS11] KOETTER, Falko ; WEIDMANN, Monika ; SCHLEICHER, Daniel: Guaranteeing soundness of adaptive business processes using ABIS. In: ABRAMOWICZ, Witold (Hrsg.) ; AALST, Wil (Hrsg.) ; MYLOPOULOS, John (Hrsg.) ; ROSEMANN, Michael (Hrsg.) ; SHAW, Michael J. (Hrsg.) ; SZYPERSKI, Clemens (Hrsg.) ; AALST, Wil (Hrsg.) ; MYLOPOULOS, John (Hrsg.) ; ROSEMANN, Michael (Hrsg.) ; SHAW, Michael J. (Hrsg.) ; SZYPERSKI, Clemens (Hrsg.): *Business Information Systems* Bd. 87. Springer Berlin Heidelberg, 2011, S. 74–85

[LB01] LUTHER, M. ; BIBELGESELLSCHAFT, Deutsche: *Die Bibel:*. Amer Bible Society, 2001 http://books.google.de/books?id=uGw4QwAACAAJ. – ISBN 9783438011022

[Ley09] LEYMANN, Frank: Cloud computing: the next revolution in IT. In: *Proc. 52th Photogrammetric Week*, Wichmann Verlag, September 2009. – ISBN 978–3–87907–483–9, S. 3–12

[LGRMD08] Ly, Linh T. ; Göser, Kevin ; Rinderle-Ma, Stefanie ; Dadam, Peter: Compliance of semantic constraints - a requirements analysis for process management systems. In: *Proc. 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08)*, 2008

[LMX07] Liu, Y. ; Müller, S. ; Xu, K.: A static compliance-checking framework for business process models. In: *IBM Syst. J.* 46 (2007), Nr. 2, S. 335–361. – ISSN 0018–8670

[LR00] Leymann, Frank ; Roller, Dieter: *Production workflow: concepts and techniques*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2000. – ISBN 0–13–021753–0

[LRD08] Ly, Linh T. ; Rinderle, Stefanie ; Dadam, Peter: Integration and verification of semantic constraints in adaptive process management systems. In: *Data & Knowledge Engineering* 64 (2008), Januar, Nr. 1, 3–23. http://dx.doi.org/10.1016/j.datak.2007.06.007. – DOI 10.1016/j.datak.2007.06.007. – ISSN 0169023X

[LS13] Leymann, Frank ; Spath, Dieter: *Konzepte und Methoden zur Unterstützung von Fachanwendern bei der Umsetzung von Adaptivität und Compliance-Richtlinien in Geschäftsprozessen*. http://gepris.dfg.de/gepris/OCTOPUS/?module=gepris&task=showDetail&context=projekt&id=219206707, January 2013

[LSG08] Lu, Ruopeng ; Sadiq, Shazia ; Governatori, Guido: Compliance aware business process design. In: *Proceedings of the 2007 international conference on Business process management*. Berlin, Heidelberg : Springer-Verlag, 2008 (BPM'07). – ISBN 3–540–78237–0, 978–3–540–78237–7, 120–131

[Mat12] Matthews, Christopher M.: *Daimler not out of the woods in bribery case*. http://blogs.wsj.com/corruption-currents/2012/04/05/daimler-not-out-of-the-woods-in-bribery-case/, April 2012

[MG09] Mell, Peter ; Grance, Tim: *The NIST definition of cloud computing*. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2009

[Mie08] Mietzner, Ralph: Using variability descriptors to describe customizable SaaS application templates / University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany. University of Stuttgart, Institute of Architecture of Application Systems, January 2008 (2008/01). – Technical Report Computer Science. – 27 S.

[Mie10] Mietzner, Ralph: *A method and implementation to define and provision variable composite applications, and its usage in cloud computing*. Holzgartenstr. 16, 70174 Stuttgart,

Universität Stuttgart, Diss., 2010. http://elib.uni-stuttgart.de/opus/volltexte/2010/5614

[ML08] Mietzner, Ralph ; Leymann, Frank: Generation of BPEL customization processes for SaaS applications from Variability Descriptors. In: *IEEE SCC*, 2008, S. 359–366

[MLP08] Mietzner, Ralph ; Leymann, Frank ; Papazoglou, Mike P.: Defining composite configurable SaaS application packages using SCA, variability descriptors and SaaS multi-tenancy patterns. In: *Proceedings of the 3rd Intl. Conf. on Internet and Web Applications and Services ICIW 2008*. Athens, Greece : IEEE, Januar 2008

[MMLP09] Mietzner, Ralph ; Metzger, Andreas ; Leymann, Frank ; Pohl, Klaus: Variability modeling to support customization and deployment of multi-tenant-aware Software as a Service applications. In: *PESOS '09: Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems*. Washington, DC, USA : IEEE Computer Society, 2009, S. 18–25

[MPRS13] Malkwitz, Alexander ; Pleines, Rüdiger ; Rehling, Timm ; Schikora, Jan: *Compliance in Industrieunternehmen - Eine sehr persönliche Angelegenheit*. http://www.germany.atkearney.com/documents/856314/1305459/Compliance+in+manufacturing.pdf/94d99f86-c52e-466a-b4dc-f5e830a2e8b2, June 2013

[Nie93] NIELSEN, Jakob: *Usability engineering*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1993. – ISBN 0125184050

[NLS+11] NOWAK, Alexander ; LEYMANN, Frank ; SCHLEICHER, Daniel ; SCHUMM, David ; WAGNER, Sebastian: Green business process patterns. In: *Proceedings of the 18th Conference on Pattern Languages of Programs, PLoP 2011*, ACM, Oktober 2011

[OAS07] OASIS: *Web services business process execution language version 2.0 – OASIS gstandard*, 2007

[Obj11] OBJECT MANAGEMENT GROUP: *Business Process Model and Notation (BPMN) - Version 2.0*. http://www.omg.org/spec/BPMN/2.0/, 2011

[ODHA06] OUYANG, Chun ; DUMAS, Marlon ; HOFSTEDE, Arthur H. M. ; AALST, Wil M. P. d.: From BPMN process models to BPEL web services. In: *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2006 (ICWS '06). – ISBN 0–7695–2669–1, S. 285–292

[OW12] OTTMANN, Thomas ; WIDMAYER, Peter: *Algorithmen und Datenstrukturen, 5. Auflage*. Spektrum Akademischer Verlag, 2012. – I–XXII, 1–774 S. – ISBN 978–3–8274–2803–5

[PJ12] PREUSS, Susanne ; JAHN, Joachim: *Daimler rechnet mit schnellerem Ende der Überwachung*.

http://www.faz.net/aktuell/wirtschaft/amerikanische-boersenaufsicht-daimler-rechnet-mit\-schnellerem-ende-der-ueberwachung-11741661.html, 2012

[Pnu77] PNUELI, Amir: The temporal logic of programs. In: *FOCS*, 1977, S. 46–57

[Pnu86] PNUELI, A: Current trends in concurrency. Overviews and tutorials. Version: 1986. http://dl.acm.org/citation.cfm?id=19518.19527. New York, NY, USA : Springer-Verlag New York, Inc., 1986. – ISBN 0–387–16488–X, Kapitel Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, 510–584

[RMF07] RIBEIRO, Óscar R. ; M. FERNANDES, Jo ao: Translating synchronous petri nets into PROMELA for verifying behavioural properties. In: *SIES*, 2007, S. 266–273

[SAL+10] SCHUMM, David ; ANSTETT, Tobias ; LEYMANN, Frank ; SCHLEICHER, Daniel ; STRAUCH, Steve: Essential aspects of compliance management with focus on business process automation. In: ABRAMOWICZ, Witold (Hrsg.) ; ALT, Rainer (Hrsg.) ; FÄHNRICH, Klaus-Peter (Hrsg.) ; FRANCZYK, Bogdan (Hrsg.) ; MACIASZEK, Leszek A. (Hrsg.): *INFORMATIK 2010: Business Process and Service Science Proceedings of ISSS and BPSC* Bd. 177, Gesellschaft für Informatik e.V. (GI), September 2010 (Lecture Notes in Informatics), S. 127–138

[SALM09] SCHLEICHER, Daniel ; ANSTETT, Tobias ; LEYMANN, Frank ; MIETZNER, Ralph: Maintaining compliance in customizable process models. In: MEERSMAN, Robert (Hrsg.) ; DILLON, Tharam (Hrsg.) ; HERRERO, Pilar (Hrsg.): *Proceedings of the 17th International Conference on Cooperative Information Systems (CoopIS 2009)* Bd. 5870. Heidelberg : Springer Verlag, November 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–05147–0, 60–75

[SALS10] SCHLEICHER, Daniel ; ANSTETT, Tobias ; LEYMANN, Frank ; SCHUMM, David: Compliant business process design using refinement layers. In: R. MEERSMAN, T. D. a. (Hrsg.): *OTM 2010 Conferences*, Springer Verlag, Oktober 2010

[SC85] SISTLA, A. P. ; CLARKE, E. M.: The complexity of propositional linear temporal logics. In: *J. ACM* 32 (1985), Juli, Nr. 3, S. 733–749. – ISSN 0004–5411

[Sch00] SCHMIDT, Karsten: LoLA: a low level analyser. In: NIELSEN, Mogens (Hrsg.) ; SIMPSON, Dan (Hrsg.): *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000, Aarhus, Denmark, June 2000. Proceedings* Bd. 1825, Springer-Verlag, Juni 2000 (Lecture Notes in Computer Science), S. 465–474

[SFG+11] SCHLEICHER, Daniel ; FEHLING, Christoph ; GROHE, Stefan ; LEYMANN, Frank ; NOWAK, Alexander ; SCHNEIDER, Patrick ; SCHUMM, David: Compliance domains: a means to model data-restrictions in cloud environments. In: *En-*

terprise Distributed Object Computing Conference (EDOC), IEEE Xplore, 2011

[SGN07] SADIQ, Shazia ; GOVERNATORI, Guido ; NAMIRI, Kioumars: Modeling control objectives for business process compliance. In: *Proceedings of the 5th international conference on Business process management*. Berlin, Heidelberg : Springer-Verlag, 2007 (BPM'07). – ISBN 3–540–75182–3, 978–3–540–75182–3, 149–164

[SLM+10] SCHUMM, David ; LEYMANN, Frank ; MA, Zhilei ; SCHEIBLER, Thorsten ; STRAUCH, Steve: Integrating compliance into business processes: process fragments as reusable compliance controls. In: SCHUMANN/KOLBE/BREITNER/FRERICHS (Hrsg.): *Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI)*, Universitätsverlag Göttingen, Februar 2010, S. 2125–2137

[SLS10] SCHUMM, David ; LEYMANN, Frank ; STREULE, Alexander: Process views to support compliance management in business processes. In: BUCCAFURRI, Francesco (Hrsg.) ; SEMERARO, Giovanni (Hrsg.): *Proceedings of the 11th International Conference on Electronic Commerce and Web Technologies (EC-Web 2010)* Bd. 61. Bilbao, Spain : Springer-Verlag, September 2010 (Lecture Notes in Business Information Processing), 131–142

[SLS+11] SCHLEICHER, Daniel ; LEYMANN, Frank ; SCHNEIDER, Patrick ; SCHUMM, David ; WOLF, Tamara: An approach to

combine data-related and control-flow-related compliance rules. In: *Proceedings of SOCA*, IEEE Computer Society, Dezember 2011

[Spi] SPINROOT.COM: *Inspiring applications of spin*. http://spinroot.com/spin/success.html,

[STK+10] SCHUMM, David ; TURETKEN, Oktay ; KOKASH, Natallia ; ELGAMMAL, Amal ; LEYMANN, Frank ; HEUVEL, Willem-Jan van d.: Business process compliance through reusable units of compliant processes. In: *Proceedings of the 1st Workshop on Engineering SOA and the Web (ESW'10)*, Springer, Juli 2010

[SWLS10] SCHLEICHER, Daniel ; WEIDMANN, Monika ; LEYMANN, Frank ; SCHUMM, David: Compliance scopes: Extending the BPMN 2.0 meta model to specify compliance requirements. In: *Proceedings of SOCA 2010*, IEEE Computer Society, Dezember 2010

[TEHP11] TURETKEN, Oktay ; ELGAMMAL, Amal ; HEUVEL, Willem-Jan van d. ; PAPAZOGLOU, Mike: Enforcing compliance on business processes through the use of patterns. In: *European Conference on Information Systems (ECIS 2011)*, Elsevier, 2011

[TLF+10] TROJER, Thomas ; LEE, Cheuk kwong ; FUNG, Benjamin C. M. ; NARUPIYAKUL, Lalita ; HUNG, Patrick C. K.: Privacy-aware health information sharing. In: *Privacy-Aware*

Knowledge Discovery: Novel Applications and New Techniques, Chapman and Hall/CRC Press, 2010

[Uni99] UNITED STATES: *Gramm-Leach-Bliley act*. U.S. G.P.O., 1999 http://books.google.com/books?id=4mhiQwAACAAJ

[Uni02] UNITED STATES CODE: *Sarbanes-Oxley Act of 2002, PL 107-204, 116 Stat 745*. Codified in Sections 11, 15, 18, 28, and 29 USC, July 2002

[Var01] VARDI, Moshe Y.: Branching vs. linear time: final showdown. In: *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. London, UK, UK : Springer-Verlag, 2001 (TACAS 2001). – ISBN 3–540–41865–2, 1–22

[VF07] VAZ, CÃitia ; FERREIRA, Carla: Towards automated verification of web services. In: *Proceedings of the IADIS International Conference on WWW/Internet*, 2007. – ISBN 978–972–8924–44–7, S. 84–92

[Web07] WEB SERVICES POLICY WORKING GROUP ; W3C (Hrsg.): *Web services policy 1.5 - framework*. : W3C, Sep 2007. http://www.w3.org/TR/ws-policy/

[WKK+11] WEIDMANN, Monika ; KÖTTER, Falko ; KINTZ, Maximilien ; SCHLEICHER, Daniel ; MIETZNER, Ralph ; LEYMANN, Frank: Adaptive business process modeling in the internet of services (ABIS). In: INTERNET, Proceedings of the Sixth International Conference o. (Hrsg.) ; APPLICATIONS, Web

(Hrsg.) ; 2011, Services (. (Hrsg.): *Adaptive Business Process Modeling in the Internet of Services (ABIS)*, Xpert Publishing Services, März 2011, S. 29–34

[WMM09] Wolter, Christian ; Miseldine, Philip ; Meinel, Christoph: Verification of business process entailment constraints using SPIN. In: Massacci, Fabio (Hrsg.) ; Redwine, Samuel (Hrsg.) ; Zannone, Nicola (Hrsg.): *Engineering Secure Software and Systems* Bd. 5429. Springer Berlin / Heidelberg, 2009. – ISBN 978–3–642–00198–7

[Wol10] Wolter, Christian: *A methodology for model-driven process security*, Hasso-Plattner Institute for IT Systems Engineering, Diss., 2010

[WPD+11] Weidlich, Matthias ; Polyvyanyy, Artem ; Desai, Nirmit ; Mendling, Jan ; Weske, Mathias: Process compliance analysis based on behavioural profiles. In: *Inf. Syst.* 36 (2011), Nr. 7, S. 1009–1025

Angegebene URLs wurden zuletzt am 09.11.2013 aufgerufen.

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ANHANG A

CODEBEISPIELE

Dieses Codebeispiel zeigt das laufende Beispiel aus Abbildung 4.1. In den Zeilen 4 bis 15 ist zu sehen, wie die Tasks des Beispielszenarios auf die Plätze des Petrinetzes abgebildet sind. Bis Zeile 45 werden die Transitionen und die Plätze, die belegt sein müssen, damit sie schalten können, definiert. In den Zeilen 48 bis 50 wird das Petri-Netz mit der Startbelegung initialisiert. Die Ausführung des Petri-Netzes wird in den Zeilen 51 bis 67 simuliert.

Listing A.1: Repräsentation des Beispielszenarios als Petrinetz, geschrieben in PROMELA

```
1 byte p[20];
2
3 #define Task3 false
4 #define Empfange_Blutdaten p[3]
5 #define Blutspender-daten_speichern p[4]
6 #define Patientendaten_sammeln p[8]
```

```
7  #define Blutverbrauchsdaten_vorbereiten p[10]
8  #define Blutverbrauchsdaten_versenden p[11]
9  #define Gesundheitsinformationen_bereitstellen p[13]
10 #define Blutverbrauchsdaten_versenden p[11]
11 #define Gesundheitsinformationen_lesen p[15]
12 #define Gesundheitsdaten_speichern p[16]
13 #define Blutverbrauchsbericht_generieren p[18]
14 #define Blutverbrauchsbericht_lesen p[13]
15 #define Gesundheitsinformationen_bereitstellen p[19]
16 #define rd_22_transition0 p[1] && !p[2]
17 #define fire_22_transition0 p[1] = 0; p[2] = 1;
18 #define rd_23_transition1 p[2] && !p[3]
19 #define fire_23_transition1 p[2] = 0; p[3] = 1;
20 #define rd_24_transition2 p[3] && !p[4]
21 #define fire_24_transition2 p[3] = 0; p[4] = 1;
22 #define rd_25_transition3 p[4] && !p[6] && !p[7]
23 #define fire_25_transition3 p[4] = 0; p[6] = 1; p[7] = 1;
24 #define rd_26_transition4 p[7] && !p[8]
25 #define fire_26_transition4 p[7] = 0; p[8] = 1;
26 #define rd_27_transition5 p[8] && !p[9]
27 #define fire_27_transition5 p[8] = 0; p[9] = 1;
28 #define rd_28_transition6 p[9] && !p[10]
29 #define fire_28_transition6 p[9] = 0; p[10] = 1;
30 #define rd_29_transition7 p[10] && !p[11]
31 #define fire_29_transition7 p[10] = 0; p[11] = 1;
32 #define rd_30_transition8 p[19] && p[11] && !p[14]
33 #define fire_30_transition8 p[19] = 0; p[11] = 0; p[14] = 1;
34 #define rd_31_transition9 p[14] && !p[15]
35 #define fire_31_transition9 p[14] = 0; p[15] = 1;
36 #define rd_32_transition10 p[15] && !p[16]
37 #define fire_32_transition10 p[15] = 0; p[16] = 1;
38 #define rd_33_transition11 p[16] && !p[17]
39 #define fire_33_transition11 p[16] = 0; p[17] = 1;
40 #define rd_34_transition12 p[6] && !p[18]
```

```
41 #define fire_34_transition12 p[6] = 0; p[18] = 1;
42 #define rd_35_transition13 p[18] && !p[13]
43 #define fire_35_transition13 p[18] = 0; p[13] = 1;
44 #define rd_36_transition14 p[13] && !p[19]
45 #define fire_36_transition14 p[13] = 0; p[19] = 1;
46 active proctype test()
47 {
48   d_step { p[0] = 0; p[1] = 1; p[2] = 0; p[3] = 0; p[4] = 0;
        p[5] = 0; p[6] = 0; p[7] = 0; p[8] = 0; p[9] = 0; p
       [10] = 0; p[11] = 0; p[12] = 0; p[13] = 0; p[14] = 0; p
       [15] = 0; p[16] = 0; p[17] = 0; p[18] = 0; p[19] = 0; }
49   do
50     :: rd_22_transition0 -> d_step{printf("
          PROCESSED_22_transition0"); fire_22_transition0}
51     :: rd_23_transition1 -> d_step{printf("
          PROCESSED_23_transition1"); fire_23_transition1}
52     :: rd_24_transition2 -> d_step{printf("
          PROCESSED_24_transition2"); fire_24_transition2}
53     :: rd_25_transition3 -> d_step{printf("
          PROCESSED_25_transition3"); fire_25_transition3}
54     :: rd_26_transition4 -> d_step{printf("
          PROCESSED_26_transition4"); fire_26_transition4}
55     :: rd_27_transition5 -> d_step{printf("
          PROCESSED_27_transition5"); fire_27_transition5}
56     :: rd_28_transition6 -> d_step{printf("
          PROCESSED_28_transition6"); fire_28_transition6}
57     :: rd_29_transition7 -> d_step{printf("
          PROCESSED_29_transition7"); fire_29_transition7}
58     :: rd_30_transition8 -> d_step{printf("
          PROCESSED_30_transition8"); fire_30_transition8}
59     :: rd_31_transition9 -> d_step{printf("
          PROCESSED_31_transition9"); fire_31_transition9}
60     :: rd_32_transition10 -> d_step{printf("
          PROCESSED_32_transition10"); fire_32_transition10}
```

```
61      :: rd_33_transition11 -> d_step{printf("
            PROCESSED_33_transition11"); fire_33_transition11}
62      :: rd_34_transition12 -> d_step{printf("
            PROCESSED_34_transition12"); fire_34_transition12}
63      :: rd_35_transition13 -> d_step{printf("
            PROCESSED_35_transition13"); fire_35_transition13}
64      :: rd_36_transition14 -> d_step{printf("
            PROCESSED_36_transition14"); fire_36_transition14}
65      :: p[17] -> goto accept
66    od;
67    accept: printf("Accepted");
68  }
```

Es folgt ein Minimalbeispiel eines Compliancetemplates in XML in Listing A.2. Dieses Beispiel basiert auf dem XML Schema für ein Compliancetemplate in Listing A.3.

Listing A.2: Minimalbeispiel für ein Compliancetemplate

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <complTemplate:complianceTemplate
3  xmlns:complTemplate="http://www.danielschleicher.com/
       complianceTemplate"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
6  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
7  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
8  xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
9  xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/
       MODEL BPMN20.xsd
10 http://www.danielschleicher.com/complianceTemplate
       complianceTemplate.xsd">
11   <complTemplate:complianceDescriptor>
12     <complTemplate:compliancePunkt>
13       <complTemplate:complianceRegel>
```

```
14          <complTemplate:complianceLink>
15            <complTemplate:complianceRegionXpath>/
                  complTemplate:complianceTemplate/
                  bpmn2:definitions/bpmn2:process/
                  complTemplate:complianceRegion</
                  complTemplate:complianceRegionXpath>
16          </complTemplate:complianceLink>
17          <complTemplate:formalComplianceRule>
18            <complTemplate:languageIndicator>ltl</
                  complTemplate:languageIndicator>
19            <complTemplate:complianceRule>[]Task1</
                  complTemplate:complianceRule>
20          </complTemplate:formalComplianceRule>
21        </complTemplate:complianceRegel>
22      </complTemplate:compliancePunkt>
23    </complTemplate:complianceDescriptor>
24    <complTemplate:variabilityModel>
25      <complTemplate:variabilityPoint>
26        <complTemplate:name/>
27        <complTemplate:alternative>
28          <complTemplate:name>free</complTemplate:name>
29          <complTemplate:default>true</complTemplate:default>
30          <complTemplate:empty/>
31        </complTemplate:alternative>
32        <complTemplate:xPathLocator>/
              complTemplate:complianceTemplate/bpmn2:definitions/
              bpmn2:process/complTemplate:complianceRegion</
              complTemplate:xPathLocator>
33      </complTemplate:variabilityPoint>
34    </complTemplate:variabilityModel>
35    <bpmn2:definitions targetNamespace="http://sample.bpmn2.
          org/bpmn2/sample/process">
36      <bpmn2:process id="process_1" name="Default Process">
37        <bpmn2:startEvent id="StartEvent_1">
```

```
38          <bpmn2:outgoing>SequenceFlow_1</bpmn2:outgoing>
39        </bpmn2:startEvent>
40        <bpmn2:sequenceFlow id="SequenceFlow_1" sourceRef="
              StartEvent_1" targetRef="ComplianceRegion_1"/>
41        <bpmn2:endEvent id="EndEvent_1">
42          <bpmn2:incoming>SequenceFlow_2</bpmn2:incoming>
43        </bpmn2:endEvent>
44        <complTemplate:complianceRegion id="ComplianceRegion_1
              " name="Compliance Region">
45          <bpmn2:incoming>SequenceFlow_1</bpmn2:incoming>
46          <bpmn2:outgoing>SequenceFlow_2</bpmn2:outgoing>
47        </complTemplate:complianceRegion>
48        <bpmn2:sequenceFlow id="SequenceFlow_2" name=""
              sourceRef="ComplianceRegion_1" targetRef="
              EndEvent_1"/>
49      </bpmn2:process>
50      <bpmndi:BPMNDiagram id="BPMNDiagram_1" name="Default
            Process Diagram">
51        <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="
              process_1">
52          <bpmndi:BPMNShape id="BPMNShape_1" bpmnElement="
                StartEvent_1">
53            <dc:Bounds height="36.0" width="36.0" x="100.0" y=
                  "100.0"/>
54          </bpmndi:BPMNShape>
55          <bpmndi:BPMNShape id="BPMNShape_2" bpmnElement="
                EndEvent_1">
56            <dc:Bounds height="36.0" width="36.0" x="500.0" y=
                  "100.0"/>
57          </bpmndi:BPMNShape>
58          <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_1"
                bpmnElement="SequenceFlow_1" sourceElement="
                BPMNShape_1" targetElement="
                BPMNShape_ComplianceRegion_1">
```

```
59            <di:waypoint xsi:type="dc:Point" x="136.0" y="
                  118.0"/>
60            <di:waypoint xsi:type="dc:Point" x="264.0" y="
                  119.0"/>
61          </bpmndi:BPMNEdge>
62          <bpmndi:BPMNShape id="BPMNShape_ComplianceRegion_1"
                bpmnElement="ComplianceRegion_1">
63            <dc:Bounds height="50.0" width="110.0" x="264.0" y
                  ="94.0"/>
64          </bpmndi:BPMNShape>
65          <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_2"
                bpmnElement="SequenceFlow_2" sourceElement="
                BPMNShape_ComplianceRegion_1" targetElement="
                BPMNShape_2">
66            <di:waypoint xsi:type="dc:Point" x="374.0" y="
                  119.0"/>
67            <di:waypoint xsi:type="dc:Point" x="500.0" y="
                  118.0"/>
68          </bpmndi:BPMNEdge>
69        </bpmndi:BPMNPlane>
70      </bpmndi:BPMNDiagram>
71    </bpmn2:definitions>
72 </complTemplate:complianceTemplate>
```

Listing A.3: XML Schema für ein Compliancetemplate

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- edited with XMLSpy v2013 sp1 (x64) (http://www.altova.
      com) by Patricia Bart-Plange (Daimler AG) -->
3 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:ns1="http://www.danielschleicher.com/
      complianceTemplate" xmlns:bpmn="http://www.omg.org/spec/
      BPMN/20100524/MODEL" targetNamespace="http://www.
      danielschleicher.com/complianceTemplate"
      elementFormDefault="qualified" attributeFormDefault="
```

```
    unqualified">
4   <xs:import namespace="http://www.omg.org/spec/BPMN
        /20100524/MODEL" schemaLocation="Semantic.xsd"/>
5   <xs:import namespace="http://www.omg.org/spec/BPMN
        /20100524/MODEL" schemaLocation="BPMN20.xsd"/>
6   <xs:element name="complianceTemplate" type="
        ns1:tcomplianceTemplate"/>
7   <xs:complexType name="tcomplianceTemplate">
8     <xs:all>
9       <xs:element ref="ns1:complianceDescriptor" minOccurs="
            0" maxOccurs="1"/>
10      <xs:element ref="ns1:variabilityModel" minOccurs="0"
            maxOccurs="1"/>
11      <xs:element ref="bpmn:definitions" minOccurs="1"
            maxOccurs="1"/>
12    </xs:all>
13  </xs:complexType>
14  <xs:element name="complianceRegion" type="
        ns1:tcomplianceRegion" substitutionGroup="
        bpmn:flowElement"/>
15  <xs:complexType name="tcomplianceRegion">
16    <xs:complexContent>
17      <xs:extension base="bpmn:tActivity">
18        <xs:sequence>
19          <xs:element ref="bpmn:laneSet" minOccurs="0"
                maxOccurs="unbounded"/>
20          <xs:element ref="bpmn:flowElement" minOccurs="0"
                maxOccurs="unbounded"/>
21          <xs:element ref="bpmn:artifact" minOccurs="0"
                maxOccurs="unbounded"/>
22        </xs:sequence>
23        <xs:attribute name="triggeredByEvent" type="
              xs:boolean" default="false"/>
24      </xs:extension>
```

```
25      </xs:complexContent>
26    </xs:complexType>
27    <xs:element name="complianceScope" type="
          ns1:tcomplianceScope" substitutionGroup="
          bpmn:flowElement"/>
28    <xs:complexType name="tcomplianceScope">
29      <xs:complexContent>
30        <xs:extension base="ns1:tcomplianceRegion"/>
31      </xs:complexContent>
32    </xs:complexType>
33    <xs:element name="languageIndicator" type="
          ns1:tlanguageIndicator"/>
34    <xs:simpleType name="tlanguageIndicator">
35      <xs:restriction base="xs:string"/>
36    </xs:simpleType>
37    <xs:element name="formalComplianceRule" type="
          ns1:tformalComplianceRule"/>
38    <xs:complexType name="tformalComplianceRule">
39      <xs:sequence>
40        <xs:element ref="ns1:languageIndicator" minOccurs="1"
              maxOccurs="1"/>
41        <xs:element name="complianceRule" type="xs:string"
              minOccurs="1" maxOccurs="1"/>
42      </xs:sequence>
43    </xs:complexType>
44    <xs:element name="complianceLink" type="
          ns1:tcomplianceLink"/>
45    <xs:complexType name="tcomplianceLink">
46      <xs:sequence>
47        <xs:element name="complianceRegionXpath" type="
              xs:string" minOccurs="1" maxOccurs="unbounded"/>
48      </xs:sequence>
49    </xs:complexType>
```

```
50  <xs:element name="complianceRegel" type="
        ns1:tcomplianceRegel"/>
51  <xs:complexType name="tcomplianceRegel">
52    <xs:sequence>
53      <xs:element ref="ns1:complianceLink"/>
54      <xs:element ref="ns1:formalComplianceRule"/>
55    </xs:sequence>
56  </xs:complexType>
57  <xs:element name="compliancePunkt" type="
        ns1:tcomliancePunkt"/>
58  <xs:complexType name="tcomliancePunkt">
59    <xs:sequence>
60      <xs:element ref="ns1:complianceRegel"/>
61    </xs:sequence>
62  </xs:complexType>
63  <xs:element name="dependencySourceXPath" type="
        ns1:tdependencySourceXPath"/>
64  <xs:simpleType name="tdependencySourceXPath">
65    <xs:restriction base="xs:string"/>
66  </xs:simpleType>
67  <xs:element name="dependencyTargetXPath" type="
        ns1:tdependencyTargetXPath"/>
68  <xs:simpleType name="tdependencyTargetXPath">
69    <xs:restriction base="xs:string"/>
70  </xs:simpleType>
71  <xs:element name="dependency" type="ns1:tdependency"/>
72  <xs:complexType name="tdependency">
73    <xs:sequence>
74      <xs:element ref="ns1:dependencySourceXPath" minOccurs=
            "1" maxOccurs="1"/>
75      <xs:element ref="ns1:dependencyTargetXPath" minOccurs=
            "1" maxOccurs="1"/>
76    </xs:sequence>
77  </xs:complexType>
```

```
78  <xs:element name="complianceDescriptor" type="
        ns1:tcomplianceDescriptor"/>
79  <xs:complexType name="tcomplianceDescriptor">
80    <xs:sequence>
81      <xs:element ref="ns1:compliancePunkt"/>
82      <xs:element ref="ns1:dependency" minOccurs="0"
            maxOccurs="unbounded"/>
83    </xs:sequence>
84  </xs:complexType>
85  <xs:element name="xPathLocator" type="xs:string"/>
86  <xs:element name="alternative" type="ns1:talternative"/>
87  <xs:complexType name="talternative">
88    <xs:sequence>
89      <xs:element name="name" type="xs:string"/>
90      <xs:element name="default" type="xs:boolean"/>
91      <xs:any namespace="##targetNamespace" processContents=
            "lax" maxOccurs="unbounded"/>
92    </xs:sequence>
93  </xs:complexType>
94  <xs:element name="variabilityPoint" type="
        ns1:tvariabilityPoint"/>
95  <xs:complexType name="tvariabilityPoint">
96    <xs:sequence>
97      <xs:element name="name" type="xs:string"/>
98      <xs:element name="dependentOn" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
99      <xs:element ref="ns1:alternative" minOccurs="1"
            maxOccurs="unbounded"/>
100     <xs:element ref="ns1:xPathLocator" minOccurs="1"
            maxOccurs="unbounded"/>
101   </xs:sequence>
102 </xs:complexType>
103 <xs:element name="variabilityModel" type="
        ns1:tvariabilityModel"/>
```

```
104   <xs:complexType name="tvariabilityModel">
105     <xs:sequence>
106       <xs:element ref="ns1:variabilityPoint" minOccurs="1"
              maxOccurs="unbounded"/>
107     </xs:sequence>
108   </xs:complexType>
109 </xs:schema>
```

ANHANG B

KOMPLEXE PROZESSBEISPIELE

Die in den Abbildungen B.1 und B.5 gezeigten Prozesse zeigen die Anwendbarkeit der in dieser Dissertation vorgestellten Konzepte an Beispielen, deren Komplexität an die Komplexität realer Prozesse angelehnt ist.

Der Fokus des in Abbildung B.1 gezeigten Prozesses liegt in der Anwendbarkeit von Compliancescopes mit realitätsnahen Prozessen. Die in diesem Beispiel gezeigten Compliancescopes sind mehrfach ineinander verschachtelt. Mit jedem Compliancescope sind Complianceregeln verknüpft. Daraus ergibt sich die Situation, dass nicht erfüllte Complianceregeln äußerer Compliancescopes an die inneren Compliancescopes weitergegeben werden müssen.

Im Folgenden werden die mit den Compliancescopes *Gesamtprozess*, *Rohbau* und *Lackieren* verknüpften Complianceregeln gezeigt. Damit wird die Plausibilität der in diesem Prozess verwendeten Compliancere-

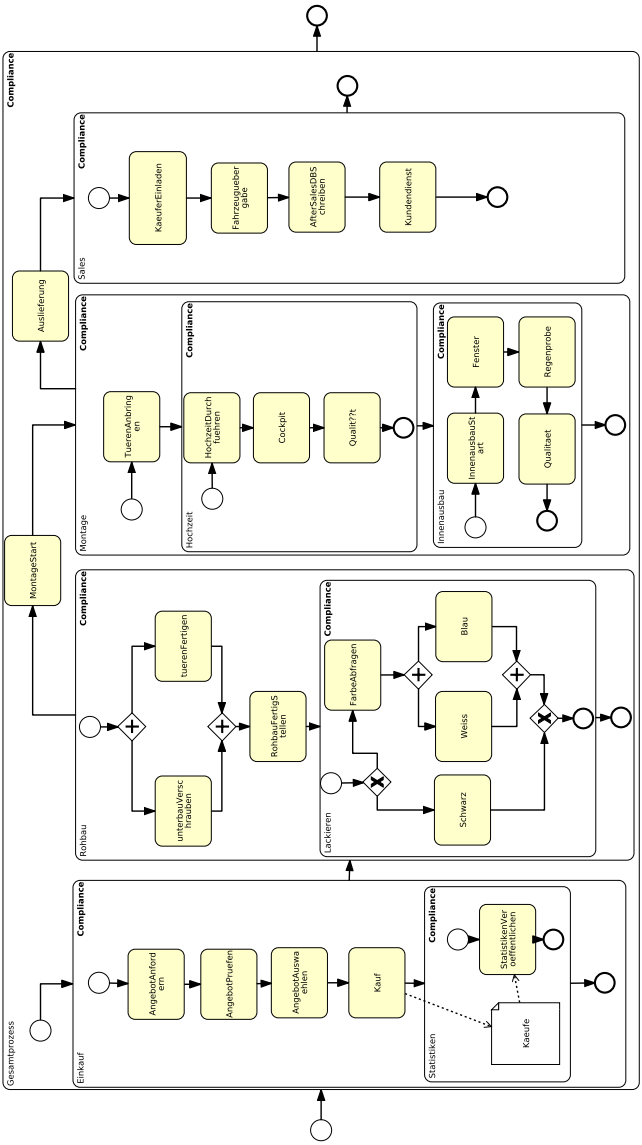

Abbildung B.1.: Fiktiver Prozess zum Bau eines Autos. Der Prozess wird von links nach rechts gelesen und enthält mehrfach verschachtelte Compliancescopes.

geln und somit die Anwendbarkeit des Konzepts des Compliancescopes mit realitätsnahen Prozessen gezeigt. Alle anderen Compliancescopes in diesem Beispiel sind auch mit Complianceregeln versehen. Aus Platzgründen werden diese nicht aufgeführt.

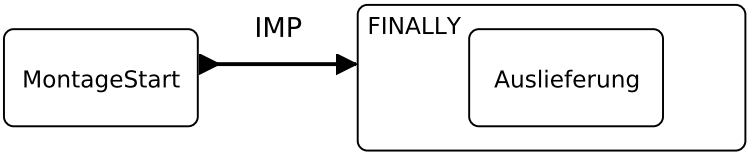

Abbildung B.2.: Graphische Repräsentation der mit dem Compliancescope mit dem Namen *Gesamtprozess* aus Abbildung B.1 verknüpften Complianceregel.

- **Complianceregel Gesamtprozess:** Die in Abbildung B.2 gezeigte Complianceregel bedeutet, dass nach dem Task *MontageStart* immer der Task *Auslieferung* ausgeführt werden muss. Es ist sinnvoll solche allgemeinen Complianceregeln mit dem äußersten Compliancescope eines Prozesses zu verknüpfen. Damit wird das Grundgerüst des Prozesses festgelegt.

- **Complianceregel Rohbau:** Die in Abbildung B.3 gezeigte Complianceregel bedeutet, dass der Task *unterbauVerschrauben* immer ausgeführt werden muss, wenn der Task *tuerenFertigen* ausgeführt wird und umgekehrt. Diese Complianceregel stellt die strukturelle Integrität des Teilprozesses im Compliancescope Rohbau sicher.

- **Complianceregel Lackieren:** Die in Abbildung B.4 gezeigte Complianceregel bedeutet, dass der Task *Schwarz* alleine ausgeführt wird oder die Tasks *Weiss* und *Blau* zusammen ausgeführt werden müssen. Mit dieser Complianceregel kann sichergestellt

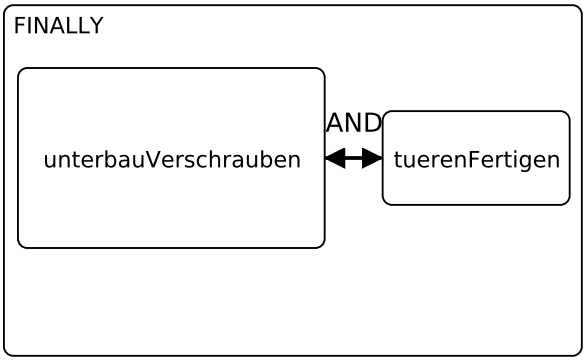

Abbildung B.3.: Graphische Repräsentation der mit dem Compliancescope mit dem Namen *Rohbau* aus Abbildung B.1 verknüpften Complianceregel.

werden, dass zum Beispiel Einschränkungen von Maschinen bei der Prozessmodellierung beachtet werden. Es könnte hier der Fall sein, dass die Lackiermaschine entweder nur Schwarz in einem Durchlauf lackieren kann oder Blau und Weiß zusammen in einem Durchlauf.

Abbildung B.5 zeigt einen anderen fiktiven Prozess. Er zeigt die Schritte die nötig sein könnten mit der App mit dem Namen *Moovel* von Daimler ein öffentliches Verkehrsmittel zu buchen. Moovel integriert mehrere Anbieter von öffentlichen Verkehrsmitteln in einer App. In den Suchergebnissen können Reisemöglichkeiten unter Verwendung mehrerer Anbieter angezeigt werden.

Im Folgenden wird beispielhaft eine datenbasierte Complianceregel

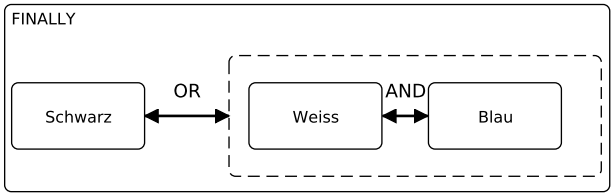

Abbildung B.4.: Graphische Repräsentation der mit dem Compliancescope mit dem Namen *Lackieren* aus Abbildung B.1 verknüpften Complianceregel.

gezeigt, die in diesem Prozess mit der Compliancedomain mit dem Namen *PublicCloud* verknüpft sein könnte.

Es muss in diesem Prozess zum Beispiel verhindert werden, dass sensible Daten, die in der Compliancedomain mit dem Namen *PrivateCloud* verarbeitet werden nach außen gelangen. Daher muss unter anderem die Datenassoziation überprüft werden, die vom Task mit dem Namen *Forward Booked Tickets* zum Datenobjekt *Ticket List* zeigt. Es kann hier zum Beispiel festgelegt werden, dass nur die Ticketnummer und der Name des Käufers übertragen werden, jedoch nicht die Zahlungsmethode und Bankverbindung. Dies kann mit der in Abschnitt 5.2 vorgestellten auf XPath basierenden Compliancesprache festgelegt und automatisch überprüft werden.

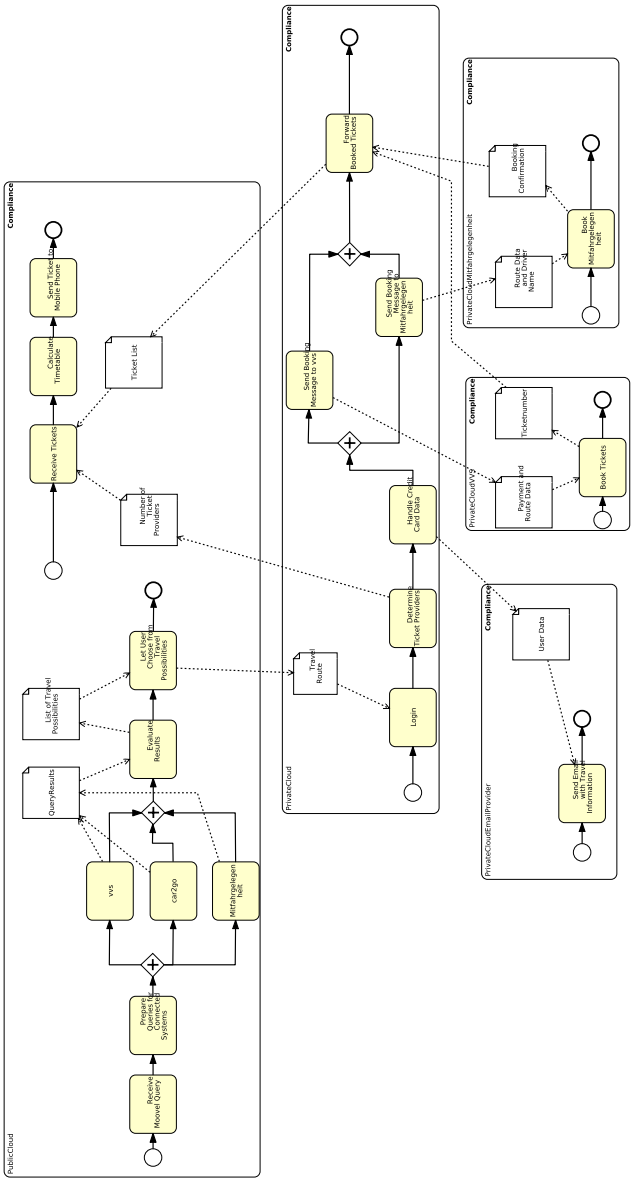

Abbildung B.5.: Fiktiver Prozess zur Buchung öffentlicher Verkehrsmittel