Strukturbasierte Verifikation von BPMN-Modellen

Dissertation

der Fakultät für Informations- und Kognitionswissenschaften

der Eberhard-Karls-Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Jens Müller

aus Villingen-Schwenningen

Tübingen

2010

Tag der mündlichen Qualifikation:	22.12.2010
Dekan:	Prof. Dr.-Ing. Oliver Kohlbacher
1. Berichterstatter:	Prof. Dr. rer. nat. Wolfgang Rosenstiel
2. Berichterstatter:	Prof. Dr.-Ing. Wilhelm G. Spruth

Für meine liebe Mutter Jutta Müller,

meinen verstorbenen Vater
László Ferenc Müller
(* 15.07.1943; † 12.10.1999)

und meinen verstorbenen Freund
Dr. med. dent. Christian Bernard Köstermenke
(* 15.04.1922; † 20.04.2009)

Danksagung

Die vorliegende Arbeit wurde im Rahmen eines Arbeitsverhältnisses bei der SAP AG in der Abteilung SAP Research am Standort Karlsruhe erstellt.

Zunächst möchte ich mich herzlich bei Herrn Prof. Dr. rer. nat. Wolfgang Rosenstiel und Herrn Prof. Dr.-Ing. Wilhelm G. Spruth für die hervorragende Betreuung meiner Arbeit bedanken. Insbesondere möchte ich die fruchtbaren Diskussionen, die ich mit ihnen geführt habe, und ihre Unterstützung auch in schwierigen Situationen hervorheben.

Meinen Vorgesetzten Herrn Dr. rer. nat. Orestis Terzidis, Herrn Dr.-Ing. Elmar Dorner, und Herrn Dr.-Ing. Zoltán Nochta möchte ich für die Möglichkeit, die vorliegende Arbeit bei der SAP AG durchführen zu können, und für ihre Unterstützung meines Promotionsvorhabens meinen Dank aussprechen. Darüber hinaus möchte ich meinen Kollegen Herrn Mario Graf und Herrn Thorsten Schneider für ihre große Hilfe bei der Verfeinerung und Implementierung der von mir entwickelten Konzepte danken. Für konstruktive inhaltliche Diskussionen und die Begutachtung meiner Arbeit möchte ich mich bei meinen Kollegen Herrn Michael Altenhofen, Herrn Dr. sc. ETH Zürich Achim Brucker und meinem Projektleiter Herrn Dr. sc. ETH Zürich Harald Vogt bedanken.

Für die Unterstützung bei der Dokumentation des Ablaufs der Flugzeugwartung als Grundlage des in dieser Arbeit angeführten Szenarios möchte ich Herrn Hermann Schmidt-Schieferstein (ehem. Airbus) sowie Herrn Stefan Schröder und Herrn Dirk Hasselbring vom Deutschen Zentrum für Luft- und Raumfahrt meinen Dank aussprechen.

Schließlich möchte ich mich herzlich bei meiner Mutter Frau Jutta Müller und Frau Asma Alazeib für ihre Geduld und moralische Unterstützung bedanken.

Jens Müller

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Dieses Dokument wurde mit LaTeX (pdfTex 1.40.11) gesetzt. Alle Vektorgraphiken wurden mit CorelDRAW X5, Microsoft Office Excel 2010 (Säulendiagramme) und Microsoft Visio Professional 2010 (Klassen- und Ontologiediagramme) erstellt.

Zusammenfassung

Das Ziel der Modellierung von Geschäftsprozessen eines Unternehmens ist normalerweise deren Dokumentation, Optimierung oder Ausführung durch informationstechnische Systeme. Die Geschäftsprozesse eines Unternehmens müssen üblicherweise gewisse Anforderungen der betriebswirtschaftlichen Ebene erfüllen, die beispielsweise auf interne Vorgaben oder gesetzliche Auflagen zurückzuführen sind. Bei der Modellierung von Geschäftsprozessen muss darauf geachtet werden, dass derartige Anforderungen von den resultierenden Modellen korrekt abgebildet werden. Gegenstand dieser Arbeit sind betriebswirtschaftliche Anforderungen, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung machen. Die Verletzung derartiger Anforderungen durch die entsprechenden Geschäftsprozessmodelle aufgrund einer fehlerhaften Modellierung könnte bei der Ausführung (durch Menschen oder informationstechnische Systeme) zu unerwarteten Ergebnissen oder im ungünstigsten Fall kritischen Situationen führen.

Derzeit erhältliche Werkzeuge zur Geschäftsprozessmodellierung bieten keine Möglichkeit, die in dieser Arbeit untersuchte Kategorie von Anforderungen explizit zu repräsentieren und automatisch auszuwerten. Diese Tatsache führt zwangsläufig dazu, dass sich das Risiko der Verletzung von Anforderungen mit deren zunehmender Anzahl erhöht. Zudem erschwert eine große Anzahl von Anforderungen die Beurteilung, ob ein vorhandenes Geschäftsprozessmodell diese Anforderungen erfüllt oder nicht. Eine weitere Problematik ergibt sich aus Geschäftsprozessmodellen, die aufgrund von Änderungen auf der betriebswirtschaftlichen Ebene im Lauf der Zeit unterschiedliche Anforderungen erfüllen müssen, da jede Änderung eine Überprüfung der modifizierten und hinzugefügten Anforderungen notwendig macht.

Zur Lösung der beschriebenen Probleme werden in der vorliegenden Arbeit entsprechende Konzepte vorgestellt. Diese Konzepte ermöglichen es, Anforderungen der untersuchten Kategorie explizit zu repräsentieren und automatisch auszuwerten. Zur Repräsentation derartiger Anforderungen wird eine Modellierungsmethode vorgestellt, die auch weniger technisch versierten Benutzern zugänglich ist. Zur automatischen Auswertung derartiger Anforderungen wird eine dreistufige Vorgehensweise präsentiert. Die entwickelten Konzepte wurden als Erweiterung der für das Geschäftsprozessmanagement zuständigen Komponente der NetWeaver-Plattform der Firma SAP implementiert und auf Grundlage dieser Erweiterung anhand eines Szenarios aus der Luftfahrtindustrie validiert.

1. Einleitung

Das Ziel der Modellierung von Geschäftsprozessen eines Unternehmens ist normalerweise deren Dokumentation, Optimierung oder Ausführung durch informationstechnische Systeme. Die Modellierung wird meist mithilfe graphischer Modellierungswerkzeuge auf Grundlage einer entsprechenden Modellierungssprache durchgeführt.

Die Geschäftsprozesse eines Unternehmens müssen üblicherweise gewisse Anforderungen der betriebswirtschaftlichen Ebene erfüllen, die beispielsweise auf interne Vorgaben oder gesetzliche Auflagen zurückzuführen sind. Bei der Modellierung von Geschäftsprozessen muss darauf geachtet werden, dass derartige Anforderungen von den resultierenden Modellen korrekt abgebildet werden.

In dieser Arbeit werden innovative Konzepte vorgestellt, die es ermöglichen, eine bestimmte Kategorie von Anforderungen der betriebswirtschaftlichen Ebene in maschinenlesbarer Form zu repräsentieren und Geschäftsprozessmodelle in Bezug auf Anforderungen, die auf diese Weise abgebildet wurden, automatisch zu verifizieren.

1.1. Problemstellung

Die Modellierung von Geschäftsprozessen unter Berücksichtigung zu erfüllender Anforderungen erfolgt durch menschliches Handeln. Gegenstand dieser Arbeit sind betriebswirtschaftliche Anforderungen, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung machen. Auf die Modellebene übertragen, könnte eine solche Anforderung beispielsweise fordern, dass ein bestimmtes Modellelement niemals auf gewisse Sequenzen von Modellelementen innerhalb eines Geschäftsprozessmodells folgen darf. Eine weitere Anforderung könnte verlangen, dass eine bestimmte Anordnung von Elementen innerhalb eines Geschäftsprozessmodells vorhanden sein muss. Die Verletzung derartiger Anforderungen durch Geschäftsprozessmodelle aufgrund einer fehlerhaften Modellierung könnte bei der Ausführung (durch Menschen oder informationstechnische Systeme) zu unerwarteten Ergebnissen oder im ungünstigsten Fall kritischen Situationen führen.

Auch wenn letztendlich menschliches Versagen die Verletzung von Anforderungen bei der Modellierung von Geschäftsprozessmodellen verursacht, ist der eigentliche Kern des Problems, dass derzeit erhältliche Werkzeuge zur Geschäftsprozessmodellierung keine Möglichkeit bieten, die in dieser Arbeit untersuchte Kategorie von Anforderungen explizit zu repräsentieren und automatisch auszuwerten. Diese Tatsache führt zwangsläufig dazu,

dass sich das Risiko der Verletzung von Anforderungen mit deren zunehmender Anzahl erhöht. Zudem erschwert eine große Anzahl von Anforderungen die Beurteilung, ob ein vorhandenes Geschäftsprozessmodell diese Anforderungen erfüllt oder nicht. Eine weitere Problematik ergibt sich aus Geschäftsprozessmodellen, die aufgrund von Änderungen auf der betriebswirtschaftlichen Ebene im Lauf der Zeit unterschiedliche Anforderungen erfüllen müssen, da jede Änderung eine Überprüfung der modifizierten und hinzugefügten Anforderungen notwendig macht.

1.2. Lösungsansatz und Szenario

Zur Lösung der beschriebenen Probleme werden in der vorliegenden Arbeit entsprechende Konzepte vorgestellt. Diese Konzepte ermöglichen es, betriebswirtschaftliche Anforderungen, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung machen, explizit zu repräsentieren und automatisch auszuwerten.

Die entwickelten Konzepte wurden als Erweiterung der für das Geschäftsprozessmanagement zuständigen Komponente [1] der *NetWeaver-Plattform* [2] der Firma SAP [3] implementiert. Ein von NetWeaver zur Verfügung gestelltes Werkzeug – der *Process Composer* [4] – ermöglicht die graphische Modellierung von Geschäftsprozessen auf Grundlage der *Business Process Modeling Notation* (BPMN). Hinsichtlich der Funktionalität ist der Process Composer mit den Werkzeugen zur Geschäftsprozessmodellierung der *ARIS Platform* [5] vergleichbar. Im Rahmen der ARIS Platform wird ein Geschäftsprozess normalerweise in Form einer *Ereignisgesteuerten Prozesskette* (EPK) [KNS92] [6] modelliert. Da BPMN im Vergleich zu EPK über differenziertere Modellierungskonstrukte verfügt, ist BPMN einerseits komplexer, andererseits aber auch ausdrucksmächtiger. BPMN-Diagramme werden vom Process Composer in Form von Modellen gespeichert, deren Repräsentation auf einem anerkannten Industriestandard basiert. Die BPMN-Spezifikation beschreibt, wie ein BPMN-Diagramm in einen ausführbaren BPEL-Prozess (*Business Process Execution Language*) [OAS07] übersetzt werden kann. Zur Ausführung von BPMN-Diagramm werden die zugrunde liegenden Modelle von NetWeaver jedoch in ein proprietäres Format übersetzt. Ausführbare Geschäftsprozessmodelle stellen wiederum ein wichtiges Element einer *serviceorientierten Architektur* (SOA) dar.

Die vorliegende Arbeit beschäftigt sich zunächst mit Konzepten, die eine präzisere Analyse der inhaltlichen Bedeutung von Elementen innerhalb von BPMN-Modellen ermöglichen (siehe Kapitel 4). Im weiteren Verlauf der Arbeit wird gezeigt, wie im Rahmen der automatischen Verifikation von BPMN-Modellen von diesen Konzepten profitiert werden kann. Die grundlegende Idee der entwickelten Konzepte besteht darin, Elemente innerhalb von BPMN-Modellen mit maschinenlesbarer Semantik anzureichern. Zu diesem Zweck wurde der Process Composer entsprechend erweitert.

Zur Repräsentation von Anforderungen der in dieser Arbeit untersuchten Kategorie wurden zwei graphische Modellierungssprachen konzipiert (siehe Kapitel 5): die *Process*

Pattern Modeling Language (PPML) und die *Process Constraint Modeling Language* (PCML). Darüber hinaus wurden zwei entsprechende Modellierungswerkzeuge entwickelt: der *Pattern Composer* und der *Constraint Composer*.

Auf Grundlage von PPML (siehe Abschnitt 5.2) können innerhalb des Pattern Composers zunächst gesuchte Anordnungen von Elementen innerhalb von BPMN-Modellen (z. B. eine bestimmte Folge von Arbeitsschritten), die eine bestimmte inhaltliche Bedeutung aufweisen (z. B. eine bestimmte Beschriftung oder maschinenlesbare Semantik), mithilfe einer an BPMN angelehnten Notation beschrieben werden. Derartige Beschreibungen gesuchter Anordnungen werden in dieser Arbeit als *strukturelle Muster* bezeichnet. Der Schwerpunkt von PPML liegt auf der Bereitstellung von generischen und flexiblen Modellierungskonstrukten, die es erlauben, ähnliche Anordnungen mittels eines einzigen strukturellen Musters zu beschreiben. Anordnungen von Elementen innerhalb von BPMN-Modellen, die mit einem strukturellen Muster übereinstimmen, werden als *Instanzen* des strukturellen Musters bezeichnet.

Auf Grundlage von PCML (siehe Abschnitt 5.3) können innerhalb des Constraint Composers logisch verknüpfte Aussagen, die sich auf die Existenz von Instanzen struktureller Muster oder die temporale Beziehung zwischen Instanzen verschiedener Muster innerhalb von BPMN-Modellen beziehen, modelliert werden. Auch in diesem Fall erfolgt die Modellierung mithilfe einer graphischen Notation. Derartige Aussagen werden in dieser Arbeit als *musterbasierte Bedingungen* bezeichnet. Darüber hinaus bezeichnen *Bedingungsausdrücke* logisch verknüpfte musterbasierte Bedingungen. Durch Verknüpfung eines BPMN-Modells mit einem Bedingungsausdruck wird festgelegt, dass das BPMN-Modell den Bedingungsausdruck erfüllen muss. Um eine derartige Verknüpfung zu bewerkstelligen, wurde der Process Composer um eine entsprechende Funktion ergänzt.

Zur Verifikation eines BPMN-Modells in Bezug auf einen Bedingungsausdruck, der mit dem Modell verknüpft ist, wurde eine dreistufige Vorgehensweise konzipiert. Im Rahmen dieser Vorgehensweise wird das BPMN-Modell zunächst nach Instanzen struktureller Muster durchsucht, die von musterbasierten Bedingungen innerhalb des Bedingungsausdrucks referenziert werden (siehe Kapitel 6). Zu diesem Zweck wurden zwei unterschiedliche Verfahren entwickelt. Zur Lösung des Suchproblems greifen beide Verfahren auf bereits vorhandene Werkzeuge zurück (regelbasierte Systeme und ontologiebasierte Inferenzmaschinen, genauer gesagt die Geschäftslogik-Integrationsplattform *Drools* bzw. die ontologiebasierte Inferenzmaschine *KAON2*), die eine deklarative Formulierung des Suchproblems ermöglichen und deren Mechanismen zur Suche verwendet werden können. Die entwickelten Verfahren machen sich diese Tatsache zunutze, indem sie strukturelle Muster mithilfe entsprechender Algorithmen in die vom jeweiligen Werkzeug benötigte Sprache zur Repräsentation des Suchproblems übersetzen, die Suche selbst jedoch den dafür optimierten Werkzeugen überlassen.

Im nächsten Schritt (siehe Kapitel 7) werden die musterbasierten Bedingungen innerhalb des Bedingungsausdrucks auf Grundlage des Resultats der Suche nach Instanzen struktureller Muster ausgewertet. Die Vorgehensweise zur Auswertung entscheidet sich auf Grundlage der jeweiligen Bedingung. Während existenzielle Bedingungen mithilfe

relativ einfacher Algorithmen ausgewertet werden (siehe Abschnitt 7.1), ist zur Auswertung temporaler Bedingungen normalerweise eine *Modellprüfung* (Model Checking) erforderlich (siehe Abschnitt 7.2), wobei im Rahmen dieser Arbeit der Modellprüfer *Spin* verwendet wurde. Zur Durchführung einer Modellprüfung wird das zu verifizierende BPMN-Modell in eine Systembeschreibung in Form eines Programms auf Grundlage der von Spin verwendeten Programmiersprache *Process Meta Language* (PROMELA) transformiert. Darüber hinaus wird je nach Bedingung eine entsprechende LTL-Formel (*Lineare temporale Logik*) [Pnu77] ausgewählt, deren Einhaltung im darauffolgenden Schritt anhand der generierten Systembeschreibung verifiziert wird.

Im Anschluss an die Auswertung der musterbasierten Bedingungen innerhalb des Bedingungsausdrucks wird schließlich der Bedingungsausdruck selbst ausgewertet (siehe Abschnitt 7.3). Nach Abschluss dieses Vorgangs steht fest, ob das zu verifizierende BPMN-Modell den Bedingungsausdruck verletzt. Sofern eine oder mehrere Verletzungen vorliegen, werden detaillierte Fehlerinformationen bereitgestellt, die eine genaue Analyse des zugrunde liegenden Problems ermöglichen. Auch zu diesem Zweck wurde eine entsprechende Erweiterung des Process Composers implementiert.

Die im Rahmen dieser Arbeit entwickelten Konzepte wurden auf Grundlage des entwickelten Prototyps anhand eines Szenarios aus der Luftfahrtindustrie (siehe Kapitel 3) validiert. Mittelpunkt dieses Szenarios ist ein Flugzeugwartungsprozess eines mittelgroßen luftfahrttechnischen Betriebs. Im Szenario wird dieser bisher nur unzureichend dokumentierte Prozess modelliert und adaptiert. Da sowohl bei der Modellierung als auch bei der Adaption bestimmte Anforderungen der betriebswirtschaftlichen Ebene eingehalten werden müssen, wird anhand des Prototyps gezeigt, wie diese Vorgänge durch Modellierung entsprechender Bedingungsausdrücke und deren automatischer Auswertung unterstützt werden können (siehe Kapitel 8).

1.3. Aufbau der Arbeit

Die vorliegende Arbeit besteht aus neun Kapiteln. Nach diesem Einführungskapitel werden in **Kapitel 2** zunächst Grundlagen vermittelt, die zum Verständnis der in dieser Arbeit vorgestellten Konzepte benötigt werden. Zunächst wird eine Einführung in die Thematik der modellgetriebenen Softwareentwicklung, des Geschäftsprozessmanagements und der Geschäftsprozessmodellierung gegeben. Anschließend werden grundlegende Methoden wissensbasierter Systeme zusammengefasst, wobei die logikbasierte Wissensrepräsentation und regelbasierte Systeme im Vordergrund stehen.

In **Kapitel 3** wird ein Szenario aus dem Bereich der Luftfahrtindustrie beschrieben. Ausgehend von diesem Szenario werden Probleme, die aufgrund von Verletzungen betriebswirtschaftlicher Anforderungen bei der Modellierung von Geschäftsprozessen und der Adaption bestehender Geschäftsprozessmodelle auftreten können, beispielhaft veranschaulicht. Am Ende dieser Arbeit wird das beschriebene Szenario wiederum zur Validierung der zur Lösung dieser Probleme entwickelten Konzepte herangezogen.

Um Benutzern eine Möglichkeit zu eröffnen, die Elemente eines Geschäftsprozessmodells mit eindeutiger Semantik zu versehen, die eine genauere rechnergestützte Analyse des Modells ermöglicht, werden in **Kapitel 4** Techniken diskutiert, die als Grundlage der im Anschluss vorgeschlagenen Lösung dienen. Die Quintessenz dieser Lösung ist die Kombination von Techniken aus dem Bereich modellgetriebener Softwareentwicklung mit solchen aus dem Bereich des semantischen Webs. Am Ende des Kapitels wird die Integration der vorgestellten Konzepte in das im Rahmen dieser Arbeit verwendete Werkzeug zur Geschäftsprozessmodellierung – den Process Composer – demonstriert.

In **Kapitel 5** werden mit der Process Pattern Modeling Language und der Process Constraint Modeling Language zwei graphische Modellierungssprachen zur Spezifikation struktureller Muster und darauf basierender Bedingungen an Geschäftsprozessmodelle eingeführt. Dabei wird jeweils sowohl das zugrunde liegende Metamodell, das auf anerkannten Industriestandards basiert, als auch die darauf basierende graphische Notation erläutert. Schließlich werden zwei graphische Modellierungswerkzeuge präsentiert, welche die Erstellung und Verwaltung struktureller Muster und musterbasierter Bedingungen auf Grundlage einer kommerziellen Entwicklungsumgebung der Firma SAP erlauben.

Zur Suche nach Instanzen struktureller Muster innerhalb von Geschäftsprozessmodellen werden in **Kapitel 6** zwei entsprechende Verfahren vorgestellt. Während das erste Verfahren auf Grundlage eines regelbasierten Systems arbeitet, bedient sich das zweite Verfahren einer ontologiebasierten Inferenzmaschine. Zu diesem Zweck wird jeweils die Transformation struktureller Muster in die Eingabedatenrepräsentation der verwendeten Werkzeuge beschrieben. Am Ende des Kapitels wird auf die Integration dieser Verfahren in die verwendete Entwicklungsumgebung eingegangen.

In **Kapitel 7** wird die Vorgehensweise zur Auswertung von Bedingungsausdrücken und darin enthaltener musterbasierter Bedingungen dargelegt. Das Hauptaugenmerk dieses Kapitels richtet sich auf die Auswertung von Bedingungen, die aufgrund ihrer Komplexität eine Modellprüfung erfordern, zu deren Durchführung ebenfalls ein bereits vorhandenes Werkzeug verwendet wird. Um dies zu realisieren, wird wiederum die Transformation von Geschäftsprozessmodellen und musterbasierten Bedingungen in die von diesem Werkzeug benötigte Repräsentation beschrieben. Zum Abschluss wird sowohl die Integration dieses Verfahrens in die verwendete Entwicklungsumgebung als auch der gesamte Ablauf der Auswertung eines Bedingungsausdrucks geschildert.

Die in den vorangehenden Kapiteln vorgestellten Konzepte werden in **Kapitel 8** anhand des in Kapitel 3 beschriebenen Szenarios auf Grundlage des entwickelten Prototyps validiert. Darüber hinaus wird über die Verwertung der erzielten Ergebnisse berichtet.

In **Kapitel 9** werden die entwickelten Konzepte und der wissenschaftliche Beitrag dieser Arbeit zusammengefasst. Zu guter Letzt wird ein Ausblick auf mögliche Weiterentwicklungen dieser Konzepte gegeben.

Am Ende dieser Arbeit befinden sich der **Anhang**, eine Übersicht der verwendeten **Akronyme**, das **Literaturverzeichnis** und das **Internetseitenverzeichnis**.

2. Grundlagen

Zum besseren Verständnis der nachfolgenden Kapitel werden in diesem Kapitel Grundlagen vermittelt, auf denen die in dieser Arbeit vorgestellten Konzepte basieren. Bei diesen Grundlagen handelt es sich um modellgetriebene Softwareentwicklung, Geschäftsprozess-Management, Geschäftsprozess- und Workflow-Modellierung sowie Methoden wissensbasierter Systeme, wobei die logikbasierte Wissensrepräsentation auf Grundlage semantischer Techniken und regelbasierte Systeme im Vordergrund stehen. Zum besseren Verständnis befindet sich der Stand der Wissenschaft und Technik jeweils am Ende der nachfolgenden Kapitel in den Abschnitten 4.5, 5.5, 6.5 und 7.5.

2.1. Modellgetriebene Softwareentwicklung

Das Prinzip *modellgetriebener Softwareentwicklung* (Model-Driven Engineering) besteht darin, Modelle als Grundlage der Softwareentwicklung zu verwenden, um die Spezifikation eines Systems von dessen Implementierung zu trennen. Ein standardisierter Ansatz zur modellgetriebenen Softwareentwicklung ist die von der *Object Management Group* (OMG) [7] entwickelte *Model Driven Architecture* (MDA) [OMG03] [8]. Die Vorgehensweise beim MDA-Ansatz ist in drei Phasen unterteilt. Zunächst wird eine plattformunabhängige Spezifikation des geplanten Systems in Form eines *plattformunabhängigen Modells* (Platform Independent Model) erstellt. Dieser Vorgang wird üblicherweise mithilfe der graphischen Modellierungssprache *Unified Modeling Language* (UML) [OMG07b] [9] bewerkstelligt. Dieses Modell weist einen hohen Abstraktionsgrad auf. Im nächsten Schritt wird das plattformunabhängige Modell in ein *plattformspezifisches Modell* (Platform Specific Model) transformiert. Plattformspezifische Modelle sind bereits auf eine konkrete Implementierungstechnik ausgerichtet und verwenden Modellierungskonstrukte, die mit Konzepten der jeweiligen Implementierungstechnik korrespondieren. Schließlich wird das plattformspezifische Modell in ausführbaren Code transformiert. Im MDA-Ansatz werden diese Transformationsschritte mithilfe entsprechender Werkzeuge automatisiert, wobei die Schwierigkeit darin besteht, plattformunabhängige in plattformspezifische Modelle zu übersetzen.

2.1.1. Metamodellierung

Zur Automatisierung der Verwaltung und zum Austausch von Metadaten (z. B. Modellen) wurde von der OMG ein entsprechender Standard namens *Meta Object Facility*

(MOF) [OMG06a] [10] entwickelt, der einen zentralen Bestandteil des MDA-Ansatzes darstellt. Die MOF-Spezifikation beschreibt eine mehrschichtige Metadatenarchitektur. Auf oberster Ebene befindet sich das selbstbeschreibende MOF-Meta-Metamodell. Mit diesem Meta-Metamodell können wiederum Metamodelle spezifiziert werden. Zu diesem Zweck stellt das Meta-Metamodell mehrere Modellierungskonstrukte zur Verfügung, beispielsweise Klassen oder Assoziationen. Darüber hinaus ist es möglich, zusätzliche Einschränkungen mithilfe der *Object Constraint Language* (OCL) [OMG06b, WK04] festzulegen. Anhand eines Metamodells können Modellierungskonstrukte und Regeln, wie Elemente eines Modells der darunterliegenden Ebene sich auf Grundlage dieser Modellierungskonstrukte in Beziehung setzen lassen, definiert werden. Prinzipiell ist der Zahl der Ebenen keine Grenze gesetzt, üblicherweise werden jedoch drei bis vier verwendet. Das Verhältnis übereinanderliegender Ebenen ist mit dem Verhältnis von Klasse zu Instanz im Bereich der objektorientierten Programmierung vergleichbar.

MOF-Modelle werden innerhalb eines *MOF-Repositoriums* (MOF Repository) verwaltet, dessen Schnittstellen in der MOF-Spezifikation beschrieben sind. Eine Vorgehensweise zur Implementierung dieser Schnittstellen auf Grundlage der Programmiersprache Java [11] ist in Form einer Spezifikation namens *Java Metadata Interface* (JMI) [JCP02] [12] verfügbar. Zum Austausch von Metadaten auf Grundlage der *Extensible Markup Language* (XML) [W3C06] [13] wurde von der OMG ein Standard namens *XML Metadata Interchange* (XMI) [OMG07a] entwickelt.

2.2. Geschäftsprozessmanagement

Eine der ersten und die wahrscheinlich bekannteste Definition des Begriffs *Geschäftsprozess* stammt von Hammer [Ham90]:

> „We define a business process as a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer."

Definitionen anderer Autoren unterscheiden sich teilweise stark, ein Großteil stimmt jedoch in vielen Punkten überein und versteht unter einem Geschäftsprozess (oft abgekürzt nur Prozess genannt) eine zielgerichtete, inhaltlich abgeschlossene, zeitlich logische Folge von Aktivitäten (auch als Funktionen bezeichnet), an der mehrere Organisationen oder Organisationseinheiten beteiligt sind und die zur Erstellung von Leistungen entsprechend den vorgegebenen, aus der Unternehmensstrategie abgeleiteten Prozesszielen dient, wobei diese Aktivitäten in Form von Informations- und/oder Materialtransformationen erbracht werden [All05, BKR08, Gad10].

Zur rechnergestützten Verarbeitung werden Geschäftsprozesse in Form von *Geschäftsprozessmodellen* oder *Workflow-Modellen* repräsentiert. Geschäftsprozesse und Workflows beschreiben Arbeitsabläufe, unterscheiden sich jedoch hinsichtlich ihres Ziels, ihrer Gestaltungsebene und ihres Detaillierungsgrades [Gad10]. Während erstere auf der fachlich-konzeptionellen Ebene die inhaltlichen Schritte eines Arbeitsablaufs normalerweise auf

Grundlage einer graphischen Notation relativ abstrakt beschrieben werden, werden letztere durch konkrete Anweisungen zur rechnergestützten Ausführung eines Arbeitsablaufs auf der operativen Ebene spezifiziert. Unter einem *Workflow* versteht die *Workflow Management Coalition* [14] einen ganz oder teilweise automatisierten Geschäftsprozess, in dem Dokumente, Informationen oder Aufgaben von einem Teilnehmer an einen anderen zur Ausführung gemäß einer Menge prozeduraler Regeln übergeben werden [Wor99]. Gadatsch sieht einen wichtigen Unterschied zwischen Geschäftsprozessen und Workflows darin, dass erstere auf der fachlich-konzeptionellen Ebene beschreiben, was zu tun ist, um die vorgegebene Geschäftsstrategie umzusetzen, während letztere auf der operativen Ebene beschreiben, wie dies umgesetzt werden soll. Er weist darauf hin, dass ein eindeutiges Unterscheidungsmerkmal die Ausführbarkeit durch einen menschlichen Aufgabenträger (Mitarbeiter) oder ein Computerprogramm sei [Gad10]. Geschäftsprozesse und Workflows werden daher meist auf unterschiedliche Art und Weise repräsentiert, worauf in einem späteren Abschnitt genauer eingegangen wird.

2.2.1. Geschäftsprozess-, Workflow- und Business-Process-Management

Gadatsch [Gad10] unterteilt die Prozessmodellierung in drei Ebenen: die strategische, die fachlich-konzeptionelle und die operative Ebene (siehe Abbildung 2.1).

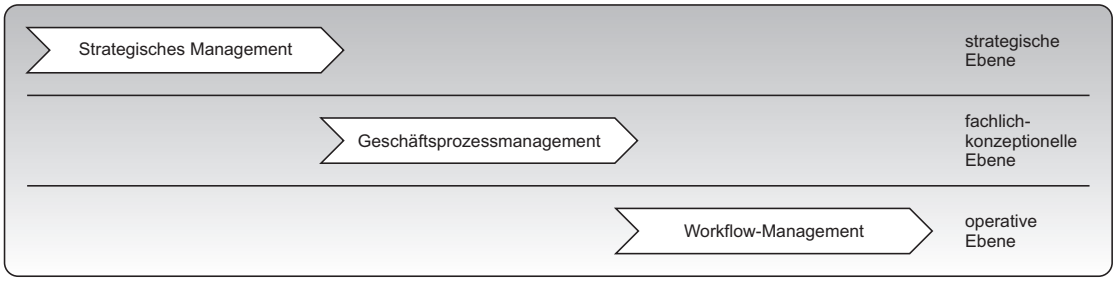


Abbildung 2.1.: Ebenen der Prozessmodellierung

Die Geschäftsfelder eines Unternehmens zeichnen sich durch kritische Erfolgsfaktoren aus. Diese werden im Rahmen des *strategischen Managements* auf der strategischen Ebene betrachtet und beeinflussen die *Unternehmensstrategie*. Das *Geschäftsprozessmanagement* auf der fachlich-konzeptionellen Ebene beschäftigt sich mit der Erfassung, Modellierung und Analyse von Geschäftsprozessen. Ausgehend von den Geschäftsfeldern eines Unternehmens werden dabei zunächst Geschäftsprozesse abgeleitet, modelliert, hinsichtlich vorgegebener Messgrößen für den Prozesserfolg analysiert und eventuell restrukturiert. Die Messgrößen ergeben sich dabei aus den kritischen Erfolgsfaktoren der jeweiligen Geschäftsfelder. Das *Workflow-Management* auf der operativen Ebene beschäftigt sich mit der Modellierung, Ausführung und Überwachung von Workflows. Bei der Workflow-Modellierung wird der im vorherigen Schritt modellierte Geschäftsprozess in eine verfeinerte Spezifikation auf technischer Ebene überführt, die von einem sogenannten *Workflow-Management-System* (WfMS) automatisch ausgeführt werden kann.

Unter einem Workflow-Management-System versteht die Workflow Management Coalition ein zentrales oder verteiltes System, das Workflows durch Verwendung einer Software definiert, erzeugt und verwaltet, das eine Prozessbeschreibung (Workflow-Modell) interpretieren kann, mit anderen Workflow-Teilnehmern interagiert und bei Bedarf Hilfsprogramme und Anwendungen aufruft [Wor99]. Bei der Überwachung eines Workflows wird dessen Ergebnis mit dem erwarteten Ergebnis verglichen. Abhängig vom Grad der Abweichung muss das Modell des Workflows oder des Geschäftsprozesses angepasst werden. Eine Weiterentwicklung der Workflow-Management-Systeme stellen die so genannten *Business-Process-Management-Systeme* (BPMS) dar. Wurde von einigen Autoren das *Business Process Management* (BPM) zunächst nur als Erweiterung des Workflow-Managements angesehen, welche auch die Analyse von Workflows einbezieht und die fachlich-konzeptionelle Ebene explizit ausklammert [WAV04], werden heutzutage zumindest in der englischen Sprache Geschäftsprozess- und Workflow-Management unter diesem Begriff zusammengefasst [Wes07], der jedoch über beide hinausgeht. Business-Process-Management-Systeme eröffnen weitergehende Möglichkeiten zur Entwicklung übergreifender prozessorientierter Anwendungen, da sie im Vergleich zu Workflow-Management-Systemen eine flexible Einbindung heterogener Systeme ermöglichen und damit das Konzept der *Enterprise Application Integration* (EAI) verwirklichen [All05]. BPM geht aber über die Kombination eines Workflow-Management-Systems und des EAI-Konzepts hinaus, da Business-Process-Management-Systeme Komponenten verschiedenster Systeme auf der Basis von *Webdiensten* (Web Services) miteinander kombinieren können, was auch als Orchestrierung bezeichnet wird und zu einer vollständigen Trennung von Prozessbeschreibung und der eigentlichen Ablauflogik führt. Eine derartige Architektur wird als *serviceorientierte Architektur* (SOA) bezeichnet.

2.2.2. Geschäftsprozess- und Workflow-Modellierung

Um die in einem Unternehmen bestehenden (aber nicht oder nicht in geeigneter Form erfassten) oder zukünftigen Geschäftsprozesse zu dokumentieren, zu optimieren oder zu automatisieren, werden diese üblicherweise zunächst mithilfe einer entsprechenden Software auf technischer Ebene modelliert und in digitaler, maschinell verarbeitbarer Form erfasst. Geschäftsprozessmodelle bilden in der Regel die wichtigsten Aspekte von Geschäftsprozessen ab. Kurbel et al. [KNS97] unterscheiden dabei zwischen Prozessschritten und deren Abhängigkeiten, Objekten und Objektflüssen sowie Aufgabenträgern. Zur Spezifikation von Geschäftsprozess- und Workflow-Modellen existieren zahlreiche *Modellierungsmethoden*. Im Rahmen der Modellierung wird dabei meist eine graphische Notation verwendet. Gadatsch [Gad10] spricht dabei von graphischen Methoden und unterscheidet sie von skriptbasierten Methoden mit einer an Programmiersprachen angelehnten formalen Notation, die vor allem bei der Spezifikation von Workflow-Modellen eingesetzt wird. Bei graphischen Methoden unterscheidet er zusätzlich zwischen datenfluss-, kontrollfluss- und objektorientierten Ansätzen. Darüber hinaus können Workflows auch mithilfe von Regeln beschrieben werden. Eine Übersicht historischer und aktueller Modellierungsmethoden ist ebenfalls in [Gad10] zu finden.

Falls einer graphischen Notation ein formales Metamodell zugrunde liegt, spricht man von einer *Modellierungssprache*, die sich aus Modellierungskonstrukten und Regeln, wie diese miteinander in Beziehung gesetzt werden können, zusammensetzt. Ein formales Metamodell bietet viele Vorteile, beispielsweise kann ein darauf basierendes Modell automatisch auf Korrektheit überprüft werden. Die Verwendung einer graphischen Methode setzt jedoch nicht zwingend das Vorhandensein eines formalen Metamodells voraus, da Modellierungsregeln auch fest in ein Modellierungswerkzeug eingebaut werden können, woraus sich offensichtliche Nachteile ergeben, sofern sich diese Regeln ändern.

2.2.3. Business Process Modeling Notation

Ein weitverbreiteter Standard der OMG zur Modellierung von Geschäftsprozessen ist die *Business Process Modeling Notation* (BPMN) [15]. Die erste Version von BPMN wurde im Jahr 2004 veröffentlicht. Im Rahmen dieser Arbeit wurde Version 1.2 [OMG09a] als Grundlage der entwickelten Konzepte verwendet. Die deutschen Übersetzungen der in dieser Arbeit verwendeten Fachausdrücke der BPMN-Spezifikation basieren auf der von Allweyer verwendeten Terminologie [All09] [16]. Wie bereits am Namen abzulesen ist, handelt es sich bei BPMN um eine graphische Notation. Da der Spezifikation dieser Notation kein Metamodell zugrunde liegt, handelt es sich bei BPMN streng genommen nicht um eine Modellierungssprache [Sie07b]. Dieser Nachteil wurde kurz vor Veröffentlichung der vorliegenden Arbeit mit Version 2.0 [OMG11] der Spezifikation behoben, der das *Business Process Definition Metamodel* (BPDM) [OMG08a, OMG08b, Sie07a] zugrunde liegt. Diese Neuerung kommt auch durch die Umbenennung des Standards in *Business Process Model and Notation* zur Geltung. Allerdings basiert auch das in dieser Arbeit verwendete Modellierungswerkzeug auf einem proprietären Metamodell, weswegen es sich bei dieser BPMN-Variante um eine Modellierungssprache handelt.

2.2.3.1. Modellierungskonstrukte

Geschäftsprozesse auf Grundlage von BPMN werden in Form von *Geschäftsprozessdiagrammen* modelliert. Zur graphischen Modellierung stehen mehrere Symbole zur Verfügung. Ein BPMN-Modell in Form eines Diagramms, das die Verwendung der wichtigsten Modellierungskonstrukte von BPMN demonstriert, ist in Abbildung 2.2 dargestellt.

Innerhalb eines BPMN-Diagramms können ein oder mehrere Geschäftsprozesse in Form sogenannter *Pools* modelliert werden. Ein Pool ist immer genau einem einzigen *Teilnehmer* zugeordnet, der die Kontrolle über den Prozessfluss hat (z. B. eine Firma). Ein Pool kann in mehrere Bahnen unterteilt werden, die beispielsweise für unterschiedliche Rollen, Systeme oder Abteilungen stehen können, die an der Ausführung des Prozesses beteiligt sind. Bei der Ausführung eines Prozesses werden Arbeitsschritte ausgeführt. Atomare Arbeitsschritte werden als *Tasks* bezeichnet und können beschriftet werden. Mithilfe des

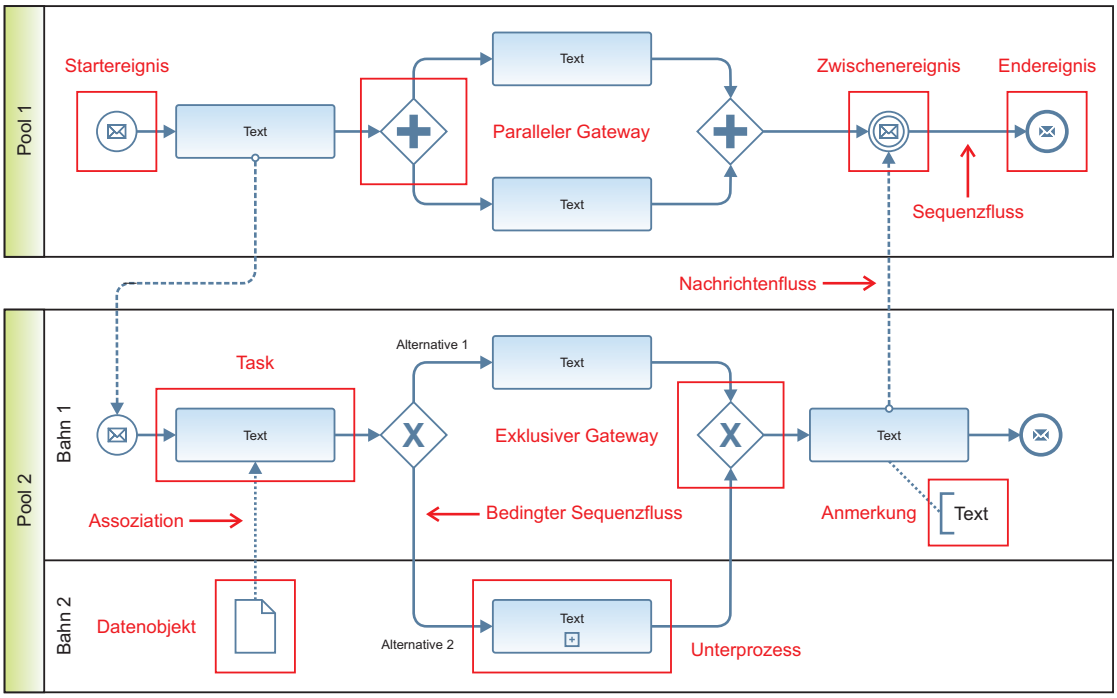


Abbildung 2.2.: Beispielhaftes BPMN-Diagramm

Modellierungskonstrukts *Unterprozess* kann der Aufruf eines weiteren Prozesses veranlasst werden, nach dessen Beendigung die Ausführung des aufrufenden Prozesses fortgesetzt wird. Tasks und Unterprozesse werden als *Aktivitäten* bezeichnet. Bei der Ausführung eines Prozesses können *Ereignisse* eintreten, die auf der einen Seite in Start-, Zwischen- und Endereignisse, auf der anderen Seite in sendende und empfangende Ereignisse unterteilt werden können. Zudem können für ein Ereignis ein oder mehrere *Auslöser* definiert werden, beispielsweise kann wie in der Abbildung dargestellt ein Ereignis den Erhalt einer Nachricht repräsentieren. Darüber hinaus kann der Kontrollfluss eines Prozesses mithilfe sogenannter *Gateways* beeinflusst werden. Bei Aktivitäten, Ereignissen und Gateways handelt es sich um *Flussobjekte*, die zur Modellierung des Kontrollflusses durch *Sequenzflüsse* miteinander verbunden werden. Sequenzflüsse können jedoch nicht dazu verwendet werden, um Flussobjekte innerhalb unterschiedlicher Pools miteinander zu verbinden. Allerdings kann die Kommunikation zwischen Pools mithilfe von *Nachrichtenflüssen* bewerkstelligt werden. Aufeinanderfolgende Aktivitäten, Ereignisse und Sequenzflüsse werden im weiteren Verlauf als *Pfade* bezeichnet.

Bei der Ausführung eines divergierenden *exklusiven Gateways* wird eine Bedingung ausgewertet. Je nach Ergebnis dieser Auswertung wird einer der ausgehenden Sequenzflüsse ausgewählt und der Kontrollfluss an dieser Stelle fortgesetzt. Bei diesen ausgehenden Sequenzflüssen handelt es sich um *bedingte Sequenzflüsse*, die üblicherweise beschriftet werden. Mithilfe eines konvergierenden exklusiven Gateways können mehrere exklusive

Pfade wieder zusammengeführt werden. Bei der Ausführung eines divergierenden *parallelen Gateways* wird der Kontrollfluss aufgespaltet, wodurch eine gleichzeitige Ausführung paralleler Pfade ermöglicht wird, die wiederum mithilfe eines konvergierenden parallelen Gateways zusammengeführt werden können.

Datenobjekte haben keinen direkten Einfluss auf die Semantik eines BPMN-Modells, sondern dienen dem besseren Verständnis des Geschäftsprozesses und können unterschiedlichste Informationen repräsentieren, beispielsweise ein Papierdokument oder einen elektronischen Datensatz. Darüber hinaus stellt BPMN auch *Anmerkungen* zur Verfügung, die beliebig beschriftet werden können. Sowohl Datenobjekte als auch Anmerkungen können über *Assoziationen* mit Flussobjekten verbunden werden. Eine ausführliche Einführung in BPMN ist in [All09, FRH10] zu finden.

2.3. Methoden wissensbasierter Systeme

In diesem Abschnitt wird eine kurze Übersicht über Methoden zur Wissensrepräsentation und zur Verarbeitung von Wissen gegeben, auf deren Grundlage einige der in dieser Arbeit vorgestellten Konzepte beruhen.

2.3.1. Wissensbasierte Systeme und Expertensysteme

Wissensbasierte Systeme sind ein Teilgebiet der künstlichen Intelligenz und werden zur Lösung komplexer Probleme innerhalb spezieller Anwendungsdomänen eingesetzt. Ein wissensbasiertes System ist ein Informationssystem zur Repräsentation und Verarbeitung von Wissen hinsichtlich der Lösung eines Problems mittels Methoden aus dem Bereich der künstlichen Intelligenz. Der wichtigste Aspekt eines wissensbasierten Systems ist die Trennung zwischen der Darstellung des Wissens über den betreffenden Problembereich und der Verarbeitung dieses Wissens [BKI08]. Dieser Ansatz steht im Gegensatz zu klassischen Programmierparadigmen (imperativ, objektorientiert, komponentenorientiert, etc.), bei denen das Wissen zur Lösung eines Problems eng mit Kontrollflussinformationen verzahnt ist. Die wichtigsten Bestandteile eines wissensbasierten Systems sind daher die *Wissensbasis* und die *Inferenzkomponente*. Bei *Expertensystemen* handelt es sich um spezielle wissensbasierte Systeme, deren Wissen von menschlichen Experten stammt. Giarratano bezeichnet ein Expertensystem als Computersystem, das die Fähigkeit zur Entscheidungsfindung eines menschlichen Experten emuliert [GR04]. Die wohl bekannteste Definition des Begriffs Expertensystem stammt von Feigenbaum [CF82]:

> „An intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution."

2.3.2. Formen der Inferenz

Zentraler Bestandteil eines wissensbasierten Systems ist die Fähigkeit, aus vorhandenem Wissen Schlüsse zu ziehen und in Grundzügen zu versuchen, intelligentes Verhalten eines Menschen nachzuahmen. Dies setzt eine formale Repräsentation des Wissens und der Methoden, wie daraus neues Wissen gewonnen werden kann, voraus. Der Begriff *Inferenz* steht dabei für die Relation zwischen vorhandenem Wissen und abzuleitendem Wissen. Peirce [Pei32] unterteilt die Inferenz in *Deduktion*, *Induktion* und *Abduktion*. In der Philosophie steht die Bezeichnung Deduktion für die Ableitung des Besonderen und Einzelnen vom Allgemeinen, in der Logik die Ableitung von Aussagen aus anderen Aussagen mithilfe logischer Schlussregeln [Dud07], wobei dies mit der Vorstellung einhergeht, dass das abgeleitete Wissen stets wahr ist. Bei der Induktion wird das Gegenteil versucht, also vom besonderen Einzelfall auf das Allgemeine, Gesetzmäßige zu schließen, wobei aber nicht mehr davon ausgegangen werden kann, dass das per Induktion abgeleitete Wissen in jedem Fall wahr ist. Die Abduktion sei hier nur der Vollständigkeit halber aufgeführt, da sie sich kategorisch von den vorherigen Verfahren unterscheidet.

2.3.3. Logikbasierte Wissensrepräsentation

Logiken bieten einen Rahmen, Inferenzrelationen zu formalisieren. In einer Logik sind sowohl Syntax als auch Semantik mathematisch präzise definiert. Beispiele sind die Systeme der *Aussagenlogik* oder der *Prädikatenlogik erster Stufe*. *Logische Systeme* bestehen auf syntaktischer Ebene aus einem Vokabular – der *Signatur* – und einer Menge von Formeln, in denen Elemente aus der Signatur mit logischen Verknüpfungsoperatoren – den Junktoren – in Verbindung gesetzt werden können, um die zu repräsentierende Welt auszudrücken. Eine Wissensbasis ist eine Untermenge der Menge aller Formeln, die man in einem logischen System über einer Signatur bilden kann. Auf der semantischen Ebene bestehen logische Systeme aus einer Zuordnung von Signatur und Semantik – den *Interpretationen* – und der *Erfüllungsrelation*. In der Aussagenlogik wird diese Zuordnung auch *Belegung* genannt und ordnet den Elementen der Signatur Wahrheitswerte zu. Die Erfüllungsrelation besagt schließlich, ob eine Formel in einer Interpretation wahr oder falsch ist. Diese vier Komponenten eines logischen Systems ermöglichen es nun beispielsweise, ausgehend von einer Wissensbasis Schlussfolgerungen zu ziehen. Eine ausführliche Einführung in die logikbasierte Wissensrepräsentation ist in [BKI08] zu finden.

2.3.3.1. Semantisches Web

Die Repräsentation der im *World Wide Web* abrufbaren Informationen ist normalerweise auf den Menschen ausgerichtet. Zur maschinellen Erfassung der Bedeutung (Semantik) von Informationen und deren Verarbeitung ist diese Repräsentation daher nicht geeignet. Eine Möglichkeit zur Lösung dieses Problems ist der Ansatz des *semantischen Webs* (Semantic Web) [17, 18], dessen Entwicklung durch das *World Wide Web Consortium*

[19] vorangetrieben wird. Dieser Ansatz zielt darauf ab, die Informationen im World Wide Web in einer Weise zu repräsentieren, die auch von Maschinen verarbeitet werden kann. Zu diesem Zweck wurden in den letzten Jahren mehrere Standards verabschiedet, die auf Konzepten der logikbasierten Wissensrepräsentation basieren: Sprachen zur Spezifikation von Wissensbasen, die in diesem Kontext als *Ontologien* [Gru93] bezeichnet werden, und entsprechende Methoden zur Schlussfolgerung.

2.3.3.2. Resource Description Framework und Web Ontology Language

Das *Resource Description Framework* (RDF) [W3C04b] [20] ist eine Sprache, die ursprünglich zur Repräsentation von Informationen über Ressourcen im World Wide Web (Metadaten) entwickelt wurde, beispielsweise zur Repräsentation des Autors einer Internetseite. Allerdings kann RDF auch zur allgemeinen Repräsentation semantischer Informationen verwendet werden. Mithilfe von RDF werden gerichtete Graphen beschrieben, wobei die Knoten und Kanten eines Graphen normalerweise mit einheitlichen Bezeichnern für Ressourcen beschriftet sind, die im Englischen als *Uniform Resource Identifier* (URI) bezeichnet werden. Eine weitere Spezifikation namens RDF Schema [W3C04c] ermöglicht darüber hinaus die Typisierung von Ressourcen und gestattet deren Unterteilung in Individuen, Beziehungen und Klassen, was weiter reichende Möglichkeiten zur Schlussfolgerung ermöglicht. RDF-Graphen können in Form von RDF/XML-Dokumenten [W3C04d], die auf XML basieren, persistent gespeichert werden.

Zur Darstellung komplexerer Zusammenhänge reichen die Sprachkonstrukte von RDF nicht aus. Aus diesem Grund wurde vom World Wide Web Consortium eine weitere Sprache zur logikbasierten Wissensrepräsentation spezifiziert, die *Web Ontology Language* (OWL) [W3C04a, W3C09]. Die Semantik von OWL lässt sich auf die Prädikatenlogik erster Stufe zurückführen. OWL-Ontologien können in Form von OWL-Dokumenten beschrieben werden, deren Syntax auf RDF basiert. OWL-Ontologien bestehen aus Klassen und Rollen (Beziehungen), im Vergleich zu RDF Schema bietet OWL jedoch eine größere Ausdrucksmächtigkeit. Darüber hinaus können OWL-Ontologien mithilfe der *Semantic Web Rule Language* (SWRL) [W3C04e] um logische Regeln erweitert werden, wodurch eine noch größere Ausdrucksmächtigkeit erreicht wird. Eine ausführliche Einführung in RDF, RDF Schema und OWL ist in [HKRS08] zu finden. Darüber hinaus bietet [SS09] eine Übersicht über aktuelle Anwendungen im Bereich des semantischen Webs.

2.3.3.3. Reasoner und Abfragesprachen

Zur Verwaltung von Ontologien und Schlussfolgerung impliziten Wissens, d. h. nicht explizit in einer Ontologie beschriebenen Wissens, werden entsprechende Werkzeuge bzw. Algorithmen benötigt. Ontologiebasierte Inferenzmaschinen zur Schlussfolgerung impliziten Wissens werden in diesem Kontext als *Reasoner* bezeichnet. Derzeit gibt es mehrere quelloffene und kommerziell vertriebene Reasoner und Werkzeuge zur Verwaltung von Ontologien, beispielsweise *Pellet* [21], *Fact++* [22] und *KAON2/OntoBroker* [23, 24].

Zur Abfrage von Informationen innerhalb einer auf RDF oder OWL basierenden Ontologie existieren entsprechende Sprachen, beispielsweise steht mit der *SPARQL Query Language for RDF* (SPARQL) [W3C08] eine Abfragesprache für RDF-Graphen zur Verfügung, deren Syntax an die *Structured Query Language* (SQL) [ISO08] angelehnt ist. SPARQL unterstützt jedoch weder die Semantik von RDF Schema noch die Semantik von OWL. Allerdings existiert auch zur Abfrage von OWL-Ontologien ein entsprechender Formalismus, die sogenannten *konjunktiven Anfragen*, bei denen es sich im Gegensatz zu SPARQL jedoch nicht um einen offiziellen Standard handelt. Dennoch gibt es trotz unterschiedlicher Beschreibungen der Semantik konjunktiver Anfragen einen allgemeinen Konsens über deren formale Interpretation. Hinsichtlich einer ausführlichen Einführung in SPARQL und konjunktive Anfragen sei ebenfalls auf [HKRS08] verwiesen.

2.3.4. Regelbasierte Systeme

Wissensbasierte Systeme können auch in Form eines *regelbasierten Systems* [HR85] implementiert werden. Bei einem regelbasierten System wird abstraktes Wissen in Form von formalisierten Konditionalsätzen dargestellt, die *Produktionsregeln* oder *Regeln* genannt werden. Im Kontext des Geschäftsprozessmanagements werden Regeln und regelbasierte Systeme auch als *Geschäftsregeln* (Business Rules) und *Geschäftsregel-Managementsysteme* (Business Rule Management Systems) bezeichnet. Aufgrund ihrer Nähe zum menschlichen Denken eignen sich Regeln zur Repräsentation von Expertenwissen und werden in der Form „Wenn P, dann K" verfasst, wobei P als *Prämisse* und K als *Konklusion* bezeichnet wird. Dem abstrakten Wissen gegenüber steht das konkrete Wissen, d. h. der aktuelle Zustand der zu repräsentierenden Welt, welcher in Form von *Fakten* ausgedrückt wird. Der zentrale Bestandteil eines regelbasierten Systems ist die Inferenzkomponente, die auf Grundlage der Fakten zutreffende Regeln anwenden kann. Der Zugriff auf Fakten erfolgt dabei innerhalb der Prämisse von Regeln. Trifft die Prämisse zu, führt die Inferenzkomponente die Konklusion der Regel aus, wobei neue Fakten entstehen oder gewisse Anweisungen ausgeführt werden können. Im Kontext des Geschäftsprozessmanagements wird die Inferenzkomponente im Englischen als *Business Rules Engine* bezeichnet. Trifft in diesem Kontext eine Geschäftsregel zu, kann dies etwa als Eintreten einer bestimmten Situation verstanden werden, auf die entsprechend reagiert werden muss. Regelbasierte Systeme sind Turing-vollständig. Verglichen mit einem Algorithmus, der in einer Turing-mächtigen imperativen Programmiersprache geschriebenen ist, kann durch entsprechende Regeldeklarationen eine äquivalente Berechnung erzielt werden. Im Gegensatz zur logikbasierten Wissensrepräsentation verfolgen regelbasierte Systeme die Repräsentation prozeduralen Wissens.

Die führenden Geschäftsregel-Managementsysteme sind die *WebSphere-ILOG-Produkte* von IBM [25], *FICO Blaze Advisor* von Fair Isaac [26] und *CA Aion Business Rules Expert* von CA Technologies [27]. Ein Beispiel für ein quelloffenes Geschäftsregel-Managementsystem ist die Geschäftslogik-Integrationsplattform *Drools* [Bal09] [28].

3. Szenario: Modellierung und Adaption von Geschäftsprozessmodellen im Rahmen der Flugzeugwartung

In diesem Kapitel werden die in der Einleitung beschriebenen Anforderungen der in dieser Arbeit untersuchten Kategorie und Probleme, die aufgrund von Verletzungen derartiger Anforderungen bei der Modellierung von Geschäftsprozessen und der Adaption bestehender Geschäftsprozessmodelle auftreten können, anhand eines realistischen Szenarios aus der Luftfahrtindustrie (siehe Abschnitt 3.4) beispielhaft veranschaulicht. Im Mittelpunkt dieses Szenarios steht ein Flugzeugwartungsprozess eines mittelgroßen luftfahrttechnischen Betriebs in Deutschland. Dieser realitätsnahe Prozess und die in den nächsten Abschnitten folgende Beschreibung der Organisation und des Ablaufs der Flugzeugwartung sowie die in diesem Zusammenhang dargestellten Maßnahmen zur Verbesserung der Sicherheit basieren auf den Ergebnissen einer Untersuchung, die im Rahmen des europäischen Forschungsprojekts *SToP Tampering of Products* (SToP) [29] vom Verfasser dieser Arbeit durchgeführt wurde. Diese Ergebnisse wurden als Grundlage des nachfolgend skizzierten Szenarios verwendet, das in Zukunft Wirklichkeit werden könnte.

Ausgangspunkt des im weiteren Verlauf noch genauer erläuterten Szenarios ist die Aufgabe, die bisher nur teilweise in Papierform beschriebenen Geschäftsprozesse eines luftfahrttechnischen Betriebs mit einer entsprechenden Software auf technischer Ebene zu modellieren und in digitaler, maschinell verarbeitbarer Form zu erfassen. Die resultierenden Modelle sollen zum einen die Dokumentation dieser Geschäftsprozesse verbessern, zum anderen sollen sie als Grundlage für deren zukünftige Teilautomatisierung dienen. Das zur Modellierung benötigte Wissen ist auf mehrere Personen des Betriebs verteilt und muss daher in mehreren Interviews gesammelt werden. Um größere Fehler bei der Modellierung zu vermeiden, liegen dem Interviewer mehrere Anforderungen, die Aussagen (in natürlicher Sprache) über die notwendige Beschaffenheit der Struktur der zu modellierenden Geschäftsprozesse machen und daher bei der Modellierung beachtet werden müssen, schriftlich vor (siehe Abschnitt 3.4.1). In einer weiteren Phase des Szenarios ändern sich diese Anforderungen an die auf Basis der Interviews erstellten Geschäftsprozessmodelle aufgrund neuer gesetzlicher Rahmenbedingungen im Luftfahrtbereich (siehe Abschnitt 3.4.2). Infolgedessen müssen diese Modelle entsprechend adaptiert werden.

Ausgehend von diesen Eckpfeilern des Szenarios werden am Ende dieses Kapitels Voraussetzungen an eine Softwarelösung formuliert, welche die Modellierung von Geschäftsprozessen und Adaption bestehender Geschäftsprozessmodelle unterstützt, indem diese Lö-

sung eine Möglichkeit bietet, Anforderungen, die bei der Modellierung und Adaption beachtet werden müssen, explizit zu modellieren und automatisch auszuwerten, um auf diese Weise etwaige Modellierungsfehler zu erkennen.

3.1. Organisation und Ablauf der Flugzeugwartung am Beispiel eines mittelgroßen luftfahrttechnischen Betriebs

In Deutschland wird die Instandhaltung von Luftfahrtgerät [BMJ09] (z. B. Flugzeugen) von Instandhaltungsbetrieben oder luftfahrttechnischen Betrieben durchgeführt [BMJ06], die vom *Luftfahrt-Bundesamt* [30] genehmigt und überwacht werden. Jedes Flugzeug unterliegt einem vom Luftfahrt-Bundesamt zugelassenen *Wartungsprogramm*, das auf Grundlage von Vorgaben des Flugzeugherstellers, der Hersteller der verwendeten Flugzeugteile und der Fluglinie erstellt wird. Je nach Nutzung unterliegt ein Flugzeug beispielsweise dem *Standard Maintenance Program* oder dem *Low Utilization Maintenance Program*. Für einen Flugzeugtyp kann es unterschiedliche Wartungsprogramme geben, für ein bestimmtes Flugzeug gibt es jedoch immer nur ein Wartungsprogramm. Im Wartungsprogramm wird beschrieben, wann welche Arbeiten (Prüfpunkte) am Flugzeug durchgeführt werden müssen. Dabei ist das Programm auf Sicherheit und Ökonomie ausgerichtet, indem versucht wird, möglichst viele Arbeitspakete bei einer Wartung abzuarbeiten. Ursachen, die eine Wartung erforderlich machen, sind kalendarische Zyklen, technische Zyklen (z. B. Triebwerk an/aus) und die Anzahl der Flugstunden.

Ein Wartungsprogramm könnte etwa aus einer Zellen- und einer Triebwerkskontrolle bestehen. Im Rahmen der Zellenkontrolle müssen unterschiedliche Arbeitspakete zu verschieden Zeiten durchgeführt werden, beispielsweise jeden Monat und alle drei, sechs, zwölf, 18, 24, 48 und 72 Monate, im Rahmen der Triebwerkskontrolle beispielsweise jeden Monat und alle drei, sechs und zwölf Monate. Folglich müssten an jedem sechsten Monat jeweils drei zusammengefasste Arbeitspakete abgearbeitet werden.

Steht eine Wartung an, werden zunächst die aufgrund fälliger Arbeitspakete zu verrichtenden Aufgaben zusammengefasst und in Form von *Arbeitszetteln* (Maintenance Task Cards) (siehe Abbildung 3.1) samt einer Übersicht ausgedruckt, wobei separate Arbeitszettellisten für die Zellen- und Triebwerkskontrolle erstellt werden. Diese Listen sind nach Kapiteln, die von der *Air Transport Association* (ATA) [31] definiert werden, geordnet und durchnummeriert, so dass die Reihenfolge ersichtlich ist und keine weiteren Arbeitszettel hinzugefügt werden können. Darüber hinaus werden Listen durchzuführender *Lufttüchtigkeitsanweisungen*, *Airworthiness Directives* und *Service Bulletins* angefertigt. Dabei handelt es sich um Beschreibungen zu verrichtender Aufgaben, die außerplanmäßig vom Luftfahrt-Bundesamt, der *Europäischen Agentur für Flugsicherheit* (EASA) [32] bzw. vom Hersteller eines Flugzeugteils herausgegeben werden. Des Weiteren wird das *Bordbuch* bereitgestellt, das Beanstandungen enthalten kann, die während eines Flugs vom Personal dort eingetragen wurden. Schließlich wird der *Inspektionsbericht* ausgedruckt, in den im Verlauf der Wartung folgende Inhalte eingetragen werden müssen:

- Beanstandungen und Erledigungsvermerke
- Teilenummern und Seriennummern der ausgebauten Teile
- Teilenummern und Seriennummern der eingebauten Teile
- Liste durchgeführter Lufttüchtigkeitsanweisungen und Airworthiness Directives
- Liste durchgeführter Service Bulletins
- Offene Beanstandungen (die nicht kritisch sind)

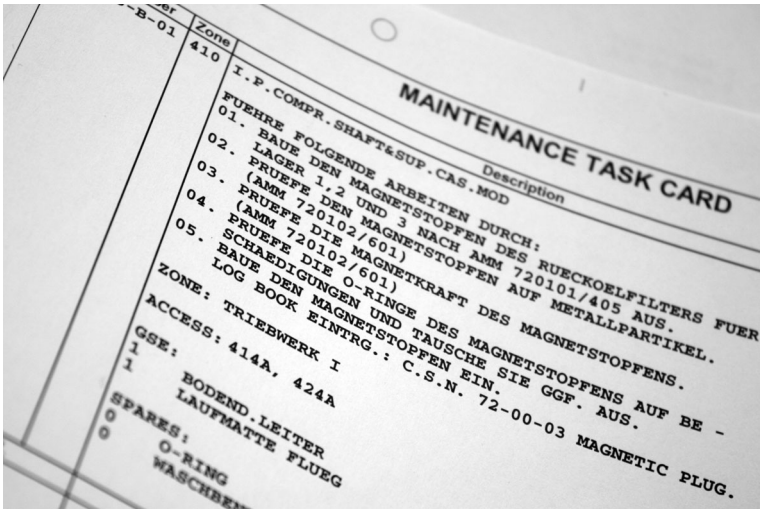


Abbildung 3.1.: Arbeitszettel

Die auf den Arbeitszetteln beschriebenen Prüfpunkte werden von *Fluggerätmechanikern* durchgeführt. Während dieser Arbeit wird auf das *Wartungshandbuch* des Flugzeugs und ein weiteres Dokument zugegriffen, das die einzelnen Teile des Flugzeugs detailliert beschreibt. Wird bei der Abarbeitung eines Arbeitszettels festgestellt, dass ein Flugzeugteil ausgewechselt werden muss, beispielsweise weil es sich um ein sogenanntes *Life-Limited Part* (LLP) handelt, wird es zunächst ausgebaut. Danach wird die dem Ausbau zugrunde liegende Beanstandung sowie die Teile- und Seriennummer des ausgebauten Teils im Inspektionsbericht vermerkt. Hinsichtlich des ausgebauten Teils wird daraufhin der Warenausgangsprozess angestoßen, worauf der Einbau eines Ersatzteils folgt. Falls kein Ersatzteil auf Lager ist, muss zuvor der Wareneingangsprozess angestoßen werden. Sowohl auf den Warenausgangs- als auch auf den Wareneingangsprozess wird in den folgenden Abschnitten noch genauer eingegangen. Im Anschluss an den Einbau des Ersatzteils wird dieser Vorgang im ERP-System (*Enterprise Resource Planning*) des Betriebs registriert sowie die Teile- und Seriennummer des Ersatzteils im Inspektionsbericht vermerkt. Nach Abarbeitung eines Arbeitszettels werden die Erledigung etwaiger Beanstandungen und die Erledigung des Arbeitszettels jeweils durch eine Unterschrift des Mechanikers bestätigt. Ähnlich wird mit Lufttüchtigkeitsanweisungen, Airworthiness Directives, Service

Bulletins und den im Bordbuch eingetragenen Beanstandungen verfahren. Die korrekte Durchführung aller Prüfpunkte wird von einem Prüfer durch Unterschreiben der Arbeitszettel, des Inspektionsberichts und des Bordbuchs bestätigt. Zu diesem Zeitpunkt ist die Luftfahrttüchtigkeit des gewarteten Flugzeugs wieder gewährleistet.

Bevor die Wartung endgültig abgeschlossen ist, müssen noch einige Schritte zur Dokumentation der Wartung durchgeführt werden. Zu diesem Zweck wird zunächst der Inspektionsbericht archiviert. Anschließend werden die im Inspektionsbericht und Bordbuch verzeichneten Teile- und Seriennummern der eingebauten Teile in die *Master Equipment List* (MEL) und das ERP-System eingetragen. Im letzten Schritt wird die *Lebenslaufakte* des Flugzeugs aktualisiert.

Wie bereits erwähnt wird der Warenausgangsprozess angestoßen, falls während der Wartung eines Flugzeugs ein Flugzeugteil ausgewechselt werden muss. Die Auswechslung eines Flugzeugteils ist erforderlich, sofern das Teil ausgefallen ist, es sich bei dem Teil um ein Life-Limited Part mit bestimmten Verbrauchswerten handelt, ein Service Bulletin eine entsprechende Anweisung enthält oder es sich bei dem Teil um ein Leihteil handelt, das an den Verleiher zurückgegeben werden muss. Zu Beginn des Warenausgangsprozesses wird daher zunächst kontrolliert, ob es sich bei dem ausgewechselten Flugzeugteil um ein Leihteil handelt. Falls dem nicht so ist, wird als nächstes überprüft, ob das Teil offensichtlich defekt ist. Falls dies der Fall ist, muss das Teil in das sogenannte *Sperrlager* transportiert und sowohl physisch als auch im Lagerprogramm als defekt gekennzeichnet werden. Parallel dazu wird innerhalb der Logistikabteilung ein Ersatzteilhersteller ausgewählt und eine Bestellung im ERP-System angelegt, die im Anschluss an den Hersteller übermittelt wird. Stellt sich bei der Kontrolle zu Beginn des Warenausgangsprozesses heraus, dass es sich bei dem ausgewechselten Flugzeugteil um ein Leihteil oder ein nicht offensichtlich defektes Teil handelt, wird ein Kurierdienst ausgewählt und die Rückgabe des Teils im ERP-System verbucht bzw. ein Wartungsauftrag im ERP-System angelegt. Anschließend wird das Teil ins Lager transportiert, im Lagerprogramm ausgetragen und zusammen mit den jeweiligen Papieren verpackt. Schließlich wird der Ausgang des Flugzeugteils im ERP-System verbucht und das Teil an den Kurierdienst übergeben.

Bei Anlieferung eines Flugzeugteils per Kurier wird der Wareneingangsprozess des luftfahrttechnischen Betriebs angestoßen. Zu Beginn des Wareneingangsprozesses wird der Eingang der Lieferung zunächst im ERP-System verbucht. Im Anschluss wird eine Sichtkontrolle der Lieferung durchgeführt, auf die eine Überprüfung der beiliegenden Papiere folgt. Daraufhin erfolgt die Qualitätskontrolle des Flugzeugteils. Fällt das gelieferte Flugzeugteil durch eine dieser Kontrollen, wird der Rückversand der Lieferung im ERP-System verbucht und das Teil zurück an den Absender versandt. Besteht das Flugzeugteil hingegen alle Kontrollen, wird es in die Lagerabteilung transportiert. Dort angekommen, wird die Einlagerung des Flugzeugteils im Lagerprogramm vermerkt und dessen Verbrauchswerte aktualisiert. Schließlich wird das Flugzeugteil entweder samt der beiliegenden Papiere eingelagert oder nach einem entsprechenden Vermerk im Lagerprogramm zum Einbau in ein Flugzeug in die Technikabteilung transportiert.

3.2. Teile zweifelhafter Herkunft

Zur Überprüfung der Authentizität eines Flugzeugteils müssen die beiliegenden Papiere auf Plausibilität überprüft werden. Ohne diese sogenannten *Begleitpapiere* darf ein Teil nicht in ein Flugzeug eingebaut werden. Ein Beispiel für ein derartiges Begleitpapier ist das EASA Formblatt 1 (siehe Abbildung A.11 im Anhang). Die in einem Begleitpapier eingetragene Teile- und Seriennummer muss zudem mit den Angaben auf dem Typenschild des Teils übereinstimmen.

Bei einem Teil zweifelhafter Herkunft [FAA95, LBA03] [33], das im Englischen als *Suspected Unapproved Part* (SUP) bezeichnet wird, handelt es sich um eine Baugruppe oder ein Teil oder Material, von dem vermutet wird, dass es nicht gemäß den genehmigten oder anerkannten Verfahren hergestellt oder instand gehalten wurde oder dass es nicht dem zugelassenen Muster oder anzuwendenden Normen oder Standards entspricht. Eigenschaften, die an der Luftfahrttauglichkeit eines Teils zweifeln lassen, sind beispielsweise unvollständige oder veraltete Begleitpapiere. Gemäß der obigen Definition muss es sich bei SUPs nicht zwangsläufig um gefälschte Teile handeln, beispielsweise können auch Originalteile aus verschrotteten Flugzeugen ausgebaut und auf unzulässige Weise wieder in die Lieferkette eingebracht werden. In jedem Fall stellen Teile zweifelhafter Herkunft ein gravierendes Sicherheitsrisiko dar. Vor einigen Jahren schätzte die *Federal Aviation Administration* (FAA), dass Teile zweifelhafter Herkunft zwischen 1973 und 1996 in 174 Fällen eine Rolle bei Abstürzen amerikanischer Flugzeuge oder weniger schwerwiegenden Unfällen spielten, die insgesamt 17 Tote und 39 Verletzte zur Folge hatten [Cla09].

Da Begleitpapiere leicht gefälscht werden können, kann es vorkommen, dass Teile zweifelhafter Herkunft nicht entdeckt und in Flugzeuge eingebaut werden. Eine italienische Firma lieferte beispielsweise über ein Jahrzehnt unerlaubt manipulierte Teile und Teile mit gefälschten Begleitpapieren an Unternehmen in der ganzen Welt [Bev04]. Im Rahmen des SToP-Projekts wurde daher untersucht, wie Teile mithilfe von *Radio Frequency Identification* (RFID) sicher authentifiziert werden können.

3.3. RFID-basierte Authentifikation von Flugzeugteilen

Hinsichtlich einer Lösung zur besseren Erkennung von Teilen zweifelhafter Herkunft wird in der Luftfahrtindustrie seit einigen Jahren der Einsatz von RFID zur Authentifikation von Flugzeugteilen diskutiert. Da zu einer umfassenden Lösung alle am Lebenszyklus eines Flugzeugs beteiligten Unternehmen einbezogen werden müssen, beispielsweise die Flugzeug- und Teilehersteller sowie Wartungs- und Verschrottungsbetriebe, wird deren Umsetzung viele Jahre beanspruchen. Um dennoch die technische Machbarkeit zu demonstrieren, wurde im Rahmen des SToP-Projekts ein prototypisches System zur Authentifikation von Flugzeugteilen mithilfe von RFID entwickelt und in einer realistischen Umgebung getestet.

Das entwickelte System bestand zum einen aus einer Datenbank, auf die über einen Webdienst zugegriffen werden konnte und in der die Identifikationsdaten (z. B. Teile- und Seriennummer) und sicherheitskritische Lebenslaufereignisse (z. B. Informationen über Einbau, Ausbau und Reparatur) von Flugzeugteilen verwaltet wurden. Diese Daten wurden zum anderen in verkürzter Form auf RFID-Transpondern gespeichert, die wiederum an die entsprechenden Flugzeugteile angebracht wurden. Ziel dieser redundanten Datenhaltung war es, die (eingeschränkte) Authentifikation eines Flugzeugteils auch ohne Verbindung zur Datenbank zu ermöglichen, was realistischen Bedingungen entsprach. Die im Rahmen des Projekts verwendeten Transponder arbeiteten im Kurzwellenbereich und verfügten über eine Speicherkapazität von 64 kbit. Um die auf Transpondern gespeicherten Daten vor Manipulation zu schützen, wurde jeder Datensatz mithilfe *elliptischer Kurven* (Elliptic Curve Cryptography) digital signiert. Zu diesem Zweck wurde eine Public-Key-Infrastruktur aufgebaut.

Zur Authentifikation von Flugzeugteilen oder Speicherung von Lebenslaufereignissen wurde eine entsprechende Software für Tablet PCs entwickelt (siehe Abbildung 3.2(a)). Während die Kommunikation zwischen dieser Software und der Datenbank über ein *lokales Funknetz* (Wireless Local Area Network) erfolgte, wurde zum Auslesen und Beschreiben von RFID-Transpondern ein drahtloser Lesestift über *Bluetooth* angesteuert (siehe Abbildung 3.2(b)). Bei einer durch den Benutzer angestoßenen Authentifikation eines RFID-Transponders wurden die darauf gespeicherten Datensätze ausgelesen und deren Integrität anhand der jeweiligen Signatur überprüft. Sofern diese Überprüfung erfolgreich verlief, wurde bei bestehender Verbindung mit der Datenbank zusätzlich ein Vergleich mit den dort gespeicherten Datensätzen durchgeführt, um auf diese Weise geklonte Transponder zu identifizieren. Vor Speicherung eines Datensatzes auf einem Transponder wurde dieser zunächst authentifiziert. Im Erfolgsfall wurde der signierte Datensatz anschließend auf den Transponder geschrieben. Bei bestehender Verbindung wurde der Datensatz darüber hinaus in der Datenbank gespeichert, andernfalls wurde im Rahmen der nächsten Authentifikation bei bestehender Verbindung eine Synchronisation zwischen Transponder und Datenbank durchgeführt. Das entwickelte System wurde von Mitarbeitern eines luftfahrttechnischen Betriebs an einer teilweise mit RFID-Transpondern ausgerüsteten VFW 614 (ein zweistrahliges Kurzstreckenjet) getestet.

3.4. Szenariobeschreibung

Auf Grundlage der in den letzten Abschnitten beschriebenen Organisation der Wartung, der Problematik aufgrund von Teilen zweifelhafter Herkunft und der entwickelten Lösung zur sicheren Authentifikation von Flugzeugteilen wird im Folgenden ein realistisches Szenario beschrieben, das die in der Einleitung beschriebenen Anforderungen der in dieser Arbeit untersuchten Kategorie und Probleme, die aufgrund von Verletzungen dieser Anforderungen bei der Modellierung von Geschäftsprozessen und der Adaption bestehender Geschäftsprozessmodelle auftreten können, veranschaulicht.

(a) Tablet PC mit Authentifikationssoftware

(b) Authentifikation eines RFID-Transponders mithilfe eines drahtlosen Lesestifts

Abbildung 3.2.: RFID-basierte Authentifikation von Flugzeugteilen

Ausgangspunkt des Szenarios ist ein luftfahrttechnischer Betrieb in Deutschland, der sich hauptsächlich mit der Wartung von Flugzeugen beschäftigt. Einer der zentralen Geschäftsprozesse dieses Betriebs ist daher der Wartungsprozess. Innerhalb dieses Prozesses wird auf zwei weitere wichtige Prozesse Bezug genommen: den Warenausgangs- und den Wareneingangsprozess. Alle Prozesse dieses Unternehmens sind bisher nur unzureichend dokumentiert und nur in geringem Umfang automatisiert. Hinsichtlich einer besseren Dokumentation und als Grundlage für eine zukünftige Teilautomatisierung entschließt sich die Geschäftsführung des Unternehmens, Modelle der vorhandenen Prozesse auf technischer Ebene anzufertigen und in digitaler, maschinell verarbeitbarer Form zu erfassen (siehe Abschnitt 2.2.2). Zu diesem Zweck beauftragt die Geschäftsführung ein externes Beratungsunternehmen, das die Modellierung in Zusammenarbeit mit den Mitarbeitern des Betriebs durchführen soll. Da aufgrund eines internationalen Abkommens die Einführung der RFID-Technik zur Authentifikation von Flugzeugteilen längerfristig umgesetzt werden muss, wird das Beratungsunternehmen in einem weiteren Schritt damit betraut, die Prozesse des Unternehmens entsprechend anzupassen. In der ersten Phase müssen während der Modellierung mehrere Anforderungen der betriebswirtschaftlichen Ebene beachtet werden, die zur Vermeidung von Modellierungsfehlern beitragen sollen. In der zweiten Phase ändern sich diese Anforderungen aufgrund gesetzlicher Änderungen auf der betriebswirtschaftlichen Ebene. Beiden Phasen werden im Folgenden genauer erläutert.

3.4.1. Modellierung von Prozessmodellen im Rahmen der Flugzeugwartung

Zur Modellierung des Wartungs-, des Warenausgangs- und des Wareneingangsprozesses führt ein Business-Analyst der beauftragten Beratungsfirma mehrere Befragungen mit den Mitarbeitern durch, die an diesen Prozessen beteiligt sind. Während dieser Befragungen macht sich der Business-Analyst handschriftliche Notizen. Die Befragungen der

Prozessverantwortlichen ergeben darüber hinaus, dass hinsichtlich der geplanten Teilautomatisierung zwar gewisse Änderungen an den Prozessen vorgenommen werden dürfen, bestimmte Anforderungen an diese Prozesse jedoch erfüllt sein und bei der Modellierung beachtet werden müssen. Einige dieser Anforderungen lauten wie folgt:

1. Wartungsprozess: Dieser Prozess muss genau zwei Sequenzen aus jeweils zwei bis drei Arbeitsschritten enthalten, wobei im ersten Schritt ein Flugzeugteil ein- oder ausgebaut wird und im nächsten Schritt die Identifikationsdaten dieses Teils im Inspektionsbericht dokumentiert werden. Dabei darf zwischen diesen Schritten maximal ein weiterer Arbeitsschritt erfolgen.
2. Wartungsprozess: Dieser Prozess muss mindestens eine Sequenz aus zwei Arbeitsschritten enthalten, in denen die Identifikationsdaten eingebauter Flugzeugteile innerhalb der MEL und des ERP-Systems dokumentiert werden. Die Reihenfolge dieser Arbeitsschritte spielt dabei keine Rolle.
3. Wartungsprozess: In diesem Prozess muss jedem Arbeitsschritt, in dem ein Flugzeugteil ausgebaut wird, im weiteren Verlauf ein Schritt folgen, in dem ein Ersatzteil eingebaut wird. Umgekehrt muss jedem Einbau ein Ausbau vorausgehen.
4. Wartungsprozess: Nachdem in diesem Prozess festgestellt wurde, dass ein Flugzeugteil ausgewechselt werden muss, aber kein Ersatzteil auf Lager ist, muss unverzüglich der Wareneingangsprozess angestoßen werden.
5. Warenausgangsprozess: In diesem Prozess muss bei der Auswahl eines Kuriersdiensts und bei der Verpackung des zu versendenden Flugzeugteils zwischen Leihteilen und Nicht-Leihteilen unterschieden werden.
6. Warenausgangsprozess: Nachdem in diesem Prozess festgestellt wurde, dass ein Teil offensichtlich defekt ist, muss es im weiteren Verlauf in das Sperrlager transportiert und im ERP-System als defekt gekennzeichnet werden.
7. Wareneingangsprozess: Nachdem in diesem Prozess festgestellt wurde, dass ein Flugzeugteil die Sicht-, Begleitpapier- oder Qualitätskontrolle nicht bestanden hat, muss im weiteren Verlauf eine Sequenz aus zwei Arbeitsschritten folgen, wobei im ersten Schritt der Rückversand der Lieferung im ERP-System verbucht und im nächsten Schritt die Lieferung zurück an den Lieferanten versandt werden muss.
8. Wareneingangsprozess: Wird in diesem Prozess auf eine Sicht-, Begleitpapier- oder Qualitätskontrolle reagiert, sofern diese bestanden wurde, muss auch immer darauf reagiert werden, sofern diese nicht bestanden wurde (und umgekehrt).

3.4.2. Adaption von Prozessmodellen im Rahmen der Flugzeugwartung

Nachdem Modelle des Wartungs-, Warenausgangs- und Wareneingangsprozesses angefertigt wurden, müssen diese Prozesse im weiteren Verlauf aufgrund der Einführung der RFID-Technik zur Authentifikation von Flugzeugteilen entsprechend angepasst werden.

Im Zuge dieser Veränderung entschließt sich die Geschäftsleitung des Unternehmens, die zur Authentifikation benötigten Tablet PCs auch zur Verwaltung und Nachweisbarkeit der im Zuge einer Wartung durchzuführenden Arbeiten zu verwenden, indem diese aus dem ERP-System abgerufen und auf dem Bildschirm angezeigt werden. Zu diesem Zweck sollen zunächst die betroffenen Geschäftsprozessmodelle adaptiert werden. Auch in diesem Fall muss bei der Modellierung auf die Einhaltung bestimmter Anforderungen geachtet werden. Zum Teil kennzeichnen diese Anforderungen vormals korrektes Prozessverhalten, das aufgrund gesetzlicher Änderungen auf der betriebswirtschaftlichen Ebene seine Gültigkeit verliert, explizit als fehlerhaft, um den Business-Analysten bei der Adaption zu unterstützen. Einige dieser Anforderungen lauten wie folgt:

9. Alle Prozesse: In allen Prozessen muss jedem sicherheitskritischen Arbeitsschritt, der Flugzeugteile betrifft (z. B. Ein- oder Ausbau), unverzüglich ein Arbeitsschritt folgen, in dem ein entsprechender Datensatz, der diesen Sachverhalt dokumentiert, auf den am Flugzeugteil angebrachten RFID-Transponder gespeichert wird. Dieser Arbeitsschritt muss ebenfalls unverzüglich folgen, nachdem in einem Prozess festgestellt wurde, dass eine sicherheitskritische Situation eingetreten ist.
10. Wartungsprozess: In diesem Prozess dürfen keine Arbeitsschritte enthalten sein, in denen Arbeitszettel, Lufttüchtigkeitsanweisungen, Airworthiness Directives oder Service Bulletins gedruckt werden.
11. Wartungsprozess: In diesem Prozess darf einem Ein- oder Ausbau eines Flugzeugteils ein Arbeitsschritt, in dem dieser Sachverhalt schriftlich dokumentiert wird, weder vorausgehen noch nachfolgen.
12. Wareneingangsprozess: In diesem Prozess muss einem Arbeitsschritt, in dem eine Sichtkontrolle eines Flugzeugteils durchgeführt wird, im weiteren Verlauf ein Schritt folgen, in dem der an diesem Teil angebrachte RFID-Transponder authentifiziert wird. Darüber hinaus muss dieser Schritt einem Arbeitsschritt, in dem die Qualitätskontrolle eines Flugzeugteils durchgeführt wird, vorausgehen.

3.5. Probleme durch Verletzung von Anforderungen bei der Modellierung und Adaption

Sowohl bei der Modellierung des Wartungs-, des Warenausgangs- und des Wareneingangsprozesses als auch bei der Adaption der entsprechenden Modelle können durch menschliches Versagen eine oder mehrere der in Abschnitt 3.4.1 und 3.4.2 beschriebenen Anforderungen verletzt werden. Diese Problematik wird in Abbildung 3.3 veranschaulicht, die einen Ausschnitt des vom Business-Analysten angefertigten Wareneingangsprozessmodells darstellt, genauer gesagt eine Beschreibung des Ablauf der verschiedenen Kontrollen nach Erhalt eines Flugzeugteils (siehe Abschnitt 3.1). Dieser Ausschnitt zeigt einen Modellierungsfehler, der dem Business-Analyst beispielsweise aufgrund einer Fehlinterpretation seiner handschriftlichen Notizen unterlaufen ist.

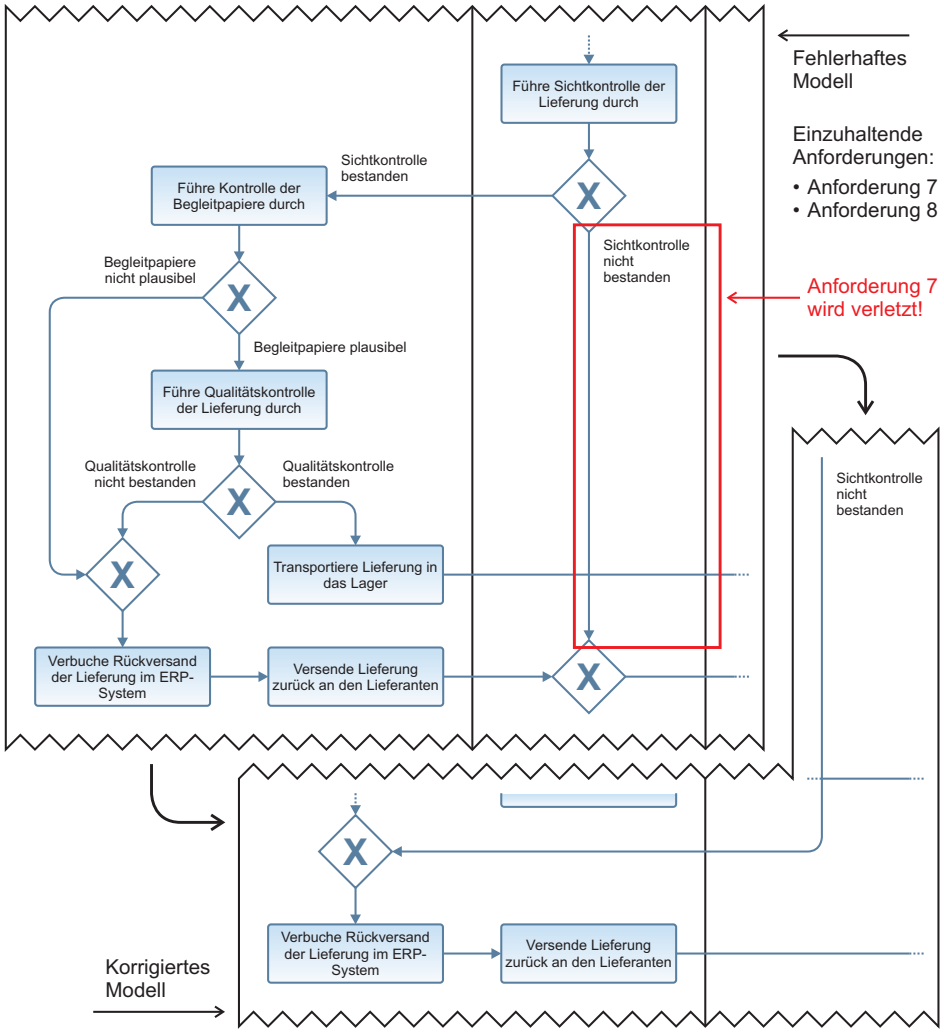


Abbildung 3.3.: Modellierungsfehler innerhalb des Wareneingangsprozessmodells

Laut *Anforderung 7* (siehe Abschnitt 3.4.2) muss, nachdem im Wareneingangsprozess festgestellt wurde, dass ein Flugzeugteil die Sicht-, Begleitpapier- oder Qualitätskontrolle nicht bestanden hat, im weiteren Verlauf eine Sequenz aus zwei Arbeitsschritten folgen, wobei im ersten Schritt der Rückversand der Lieferung im ERP-System verbucht und im nächsten Schritt die Lieferung zurück an den Lieferanten versandt werden muss. Diese Anforderung wird von dem in Abbildung 3.3 dargestellten Ausschnitt des Wareneingangsprozessmodells verletzt, da im Fall einer nicht bestandenen Sichtkontrolle im weiteren Verlauf weder der Rückversand der Lieferung im ERP-System verbucht noch die Lieferung im darauf folgenden Arbeitsschritt zurück an den Lieferanten versandt wird. Abbildung 3.3 zeigt auch, wie der Wareneingangsprozess modelliert sein müsste, um *Anforderung 7* zu erfüllen. Wie in der Einleitung erwähnt, kann die Verletzung derartiger Anforderungen bei der Ausführung zu unterwarteten Ergebnissen oder kritischen

Situationen führen. Der eigentliche Kern des Problems ist die Tatsache, dass derzeit erhältliche Werkzeuge zur Geschäftsprozessmodellierung keine Möglichkeit bieten, Anforderungen der in dieser Arbeit untersuchten Kategorie explizit zu repräsentieren und automatisch auszuwerten, um auf diese Weise Modellierungsfehler zu erkennen und den Benutzer wie in Abbildung 3.3 dargestellt (rotes Rechteck) darauf hinzuweisen.

3.6. Anforderungen an eine Softwarelösung zur Erkennung von Problemen bei der Modellierung und Adaption

Zur automatischen Erkennung von Problemen bei der Modellierung und Adaption, die auf verletzte Anforderungen zurückzuführen sind, ist eine entsprechende Softwarelösung erforderlich. Im Folgenden werden fundamentale Voraussetzungen einer derartigen Softwarelösung auf Grundlage des Szenarios und am Beispiel von *Anforderung 7* an den Wareneingangsprozess (siehe Abschnitt 3.4.1) diskutiert, wobei die inhaltlichen Bestandteile dieser Anforderung in Abbildung 3.4 dargestellt sind.

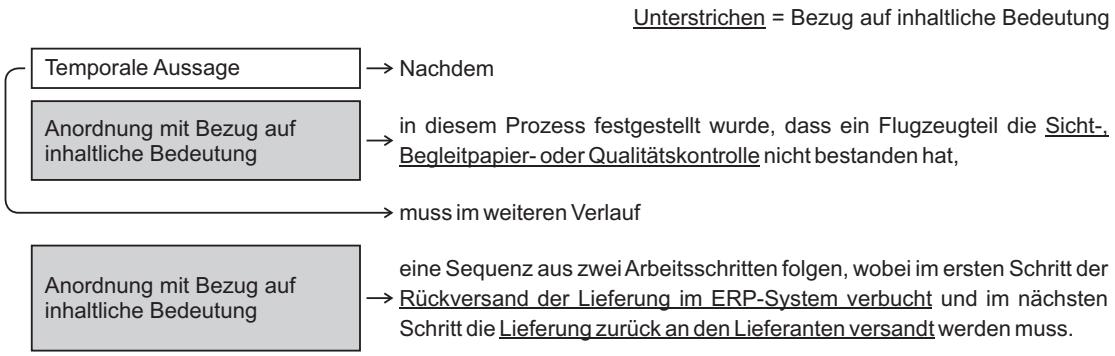


Abbildung 3.4.: Inhaltliche Bestandteile von Anforderung 7

Bei genauerer Betrachtung ihrer inhaltlichen Bestandteile kann *Anforderung 7* als temporale Aussage klassifiziert werden, welche die zeitliche Abfolge zweier Anordnungen von Modellelementen vorschreibt, wobei Bezug auf die inhaltliche Bedeutung dieser Modellelemente genommen wird. Zur Modellierung derartiger Anforderungen an Geschäftsprozessmodelle müssen diese Bestandteile abgebildet werden können. Aus diesem Grund lassen sich folgende Voraussetzungen einer Softwarelösung zur Erkennung von Problemen bei der Modellierung und Adaption ableiten:

- **Voraussetzung 1:** Auf technischer Ebene bezieht sich die in Abbildung 3.4 dargestellte Anforderung auf Anordnungen von Elementen innerhalb eines Geschäftsprozessmodells, die eine bestimmte inhaltliche Bedeutung aufweisen. Die inhaltliche Bedeutung der Elemente eines Geschäftsprozessmodells wird normalerweise durch deren Beschriftung in natürlicher Sprache festgelegt. Derartige Beschriftungen können jedoch zu Mehrdeutigkeiten führen oder Rechtschreibfehler enthalten. Nach-

dem auch die rechnergestützte Verarbeitung natürlicher Sprache aufwendig und fehleranfällig ist, sollte der Benutzer die Möglichkeit haben, die Elemente eines Geschäftsprozessmodells mit eindeutiger maschinenlesbarer Semantik anreichern zu können, auf die bei der Modellierung und automatischen Auswertung von Anforderungen zurückgegriffen werden kann. Allerdings sollte die Modellierung von Anforderungen auch ohne derartige Anreicherungen auskommen.

- **Voraussetzung 2:** Die in Abbildung 3.4 dargestellte Anforderung bezieht sich auf zwei Anordnungen von Elementen innerhalb eines Geschäftsprozessmodells und deren zeitliche Abfolge. Der Benutzer sollte die Möglichkeit haben, sowohl gesuchte Anordnungen von Modellelementen als auch Aussagen existenzieller und temporaler Natur, die sich auf diese Anordnungen beziehen, auf einfache Art und Weise zu spezifizieren und zu verwalten, wobei Bezug auf die inhaltliche Bedeutung von Modellelementen genommen werden können muss.
- **Voraussetzung 3:** Die Softwarelösung muss eine Möglichkeit bieten, sowohl gesuchte Anordnungen von Elementen innerhalb von Geschäftsprozessmodellen zu identifizieren als auch Aussagen, die sich auf diese Anordnungen beziehen, automatisch und innerhalb weniger Sekunden auszuwerten.

Unter dem Aspekt der Praxistauglichkeit wurden die in den folgenden Kapiteln beschriebenen Konzepte zusätzlich zu diesen direkt abgeleiteten Voraussetzungen auf Grundlage einer weiteren Voraussetzung entwickelt:

- **Voraussetzung 4:** Die Softwarelösung sollte möglichst eng mit einem vorhandenen Werkzeug zur Geschäftsprozessmodellierung verzahnt sein und auf anerkannten Industriestandards beruhen.

Wie in Abbildung 3.5 dargestellt, widmen sich die nächsten vier Kapitel den ersten drei Voraussetzungen, während die letzte Voraussetzung in jedem dieser Kapitel separat in Form einer Beschreibung des entwickelten Prototyps behandelt wird. Die in diesen Kapiteln vorgestellten Konzepte werden schließlich in Kapitel 8 auf Grundlage des beschriebenen Szenarios validiert.

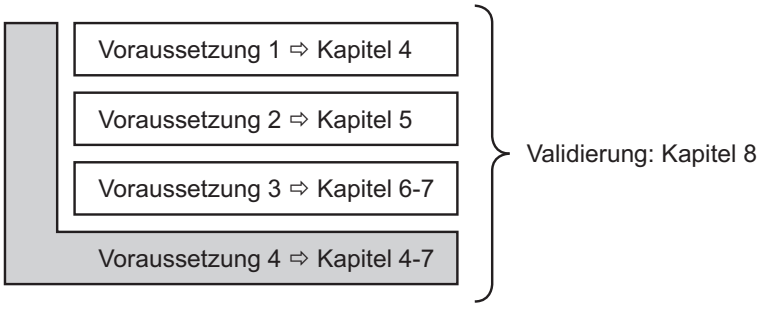


Abbildung 3.5.: Voraussetzungen und entsprechende Kapitel

4. Semantische Geschäftsprozessmodellierung auf Basis von BPMN und MOF

In diesem Kapitel wird beschrieben, wie die Elemente eines Geschäftsprozessmodells mit maschinenlesbarer Semantik angereichert werden können, um mögliche Probleme bei deren Beschriftung in natürlicher Sprache (z. B. Mehrdeutigkeiten, Rechtschreibfehler, etc.) zu vermeiden und eine präzisere Analyse deren inhaltlicher Bedeutung zu ermöglichen, wovon im Rahmen der automatischen Verifikation von Geschäftsprozessmodellen profitiert werden kann. Die Implementierungsgrundlage der in dieser Arbeit vorgestellten Konzepte ist der *Process Composer* der Firma SAP, ein Werkzeug zur Modellierung von Geschäftsprozessen im Kontext der *NetWeaver-Plattform*. Vergleichbare Werkzeuge werden auch von anderen Herstellern angeboten. Zur internen und graphischen Repräsentation von Geschäftsprozessmodellen verwendet der Process Composer MOF bzw. BPMN (siehe Abschnitt 2.1.1 und 2.2.3). Aus diesem Grund orientieren sich auch die in diesem und weiteren Kapiteln entwickelten Konzepte an dieser technischen Grundlage.

Wie in Abschnitt 2.2.2 beschrieben, existieren zahlreiche Ansätze zur Geschäftsprozess- und Workflow-Modellierung. Die Spezifikation von BPMN definiert lediglich eine graphische Notation zur Modellierung von Geschäftsprozessen, nicht jedoch eine Modellierungssprache (siehe Abschnitt 2.2.3). Eine Modellierungssprache ist eine formale Sprache zur Beschreibung von Modellen. Formale Sprachen wiederum können durch Grammatiken beschrieben werden. Das Pendant zur Grammatik findet sich im Kontext der MDA (siehe Abschnitt 2.1) in Form des Metamodells. Ein Metamodell beschreibt also eine Modellierungssprache. Über die Spezifikation hinaus repräsentiert der Process Composer Geschäftsprozessdiagramme intern auf Basis eines proprietären BPMN-Metamodells (siehe Abbildung A.1 im Anhang). Aufgrund dieses Metamodells handelt es sich bei der vom Process Composer verwendeten BPMN-Variante tatsächlich um eine (graphische) Modellierungssprache. In Abbildung 4.1 ist der Zusammenhang zwischen dem Process Composer und der zugrunde liegenden *Modeling Infrastructure* (MOIN) abgebildet. Bei MOIN handelt es sich um eine NetWeaver-spezifische Implementierung der MOF-Spezifikation. Die graphische Darstellung des BPMN-Diagramms beruht auf dessen interner Repräsentation, auf die der Process Composer über die Metadatenschnittstelle von MOIN zugreift. Jedes Element des Diagramms ist normalerweise zwei MOF-Modellelementen zugeordnet, wobei ein Element das eigentliche Geschäftsprozessobjekt (z. B. einen Task) inklusive der zugehörigen Attribute repräsentiert, während das andere dessen graphische Repräsentation beschreibt (z. B. abgerundetes Rechteck).

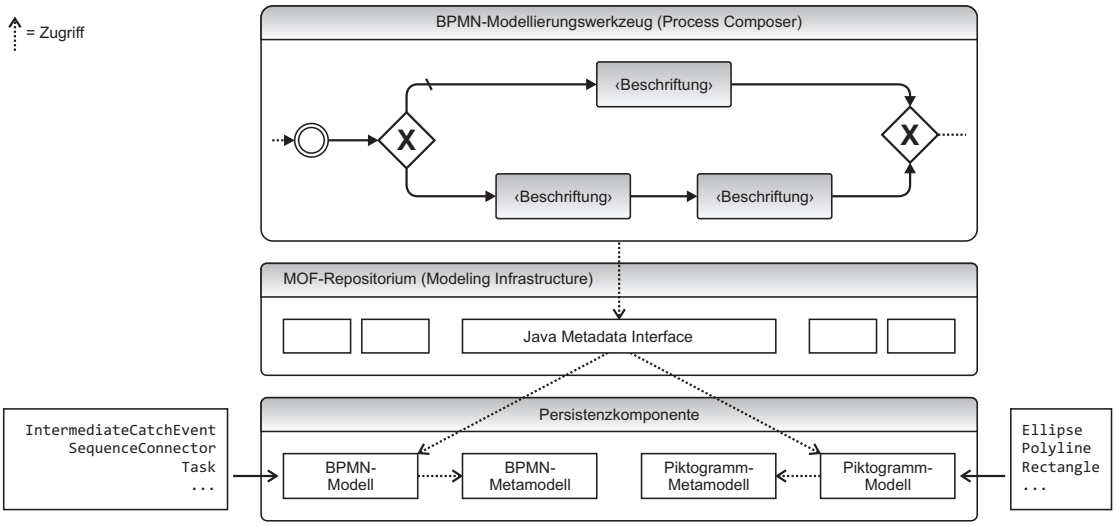


Abbildung 4.1.: Modellierungswerkzeug und Modellierungsinfrastrukur

Aufgrund des proprietären BPMN-Metamodells kann die Einhaltung der in der BPMN-Spezifikation lediglich in natürlicher Sprache beschriebenen Modellierungsregeln erzwungen werden, wodurch sich gewisse Einschränkungen bei der Modellierung ergeben. Im Rahmen von BPMN ist es beispielsweise nicht möglich, einem Endereignis einen ausgehenden Sequenzfluss, der das Ereignis mit einem weiteren Modellelement verbindet, hinzuzufügen, da dies ein entsprechender OCL-Ausdruck verbietet. Bei Verletzung einer derartigen Einschränkung zeigt der Process Composer eine entsprechende Fehlermeldung an. Während also die Syntax bei der Modellierung von Geschäftsprozessen im Modellierungswerkzeug durch das Metamodell vorgegeben ist, basiert die grundlegende Bedeutung der Elemente eines Geschäftsprozessmodells auf deren Beschreibung in der BPMN-Spezifikation, beispielsweise wird eine Aktivität als generische Bezeichnung für die von einer Firma verrichtete Arbeit definiert. Zur eigentlichen Beschreibung dieser Arbeit kann der Benutzer das für Flussobjekte vorgesehene Namensattribut verwenden. Kann diese Information bereits bei verschiedenen Menschen aufgrund von Mehrdeutigkeiten unterschiedliche Vorstellungen wecken, ist sie für Maschinen gänzlich ungeeignet, die eine Repräsentation von Wissen in einer Form benötigen, wie sie in Abschnitt 2.3.3 vorgestellt wurde. Da die vorliegende Arbeit eine automatische Auswertung von Anforderungen an Geschäftsprozessmodelle zum Ziel hat, beispielsweise eine Überprüfung der Aussage, dass ein Task mit bestimmter inhaltlicher Bedeutung innerhalb des Modells enthalten sein muss, ist es von Vorteil, wenn dessen Elemente über eine Semantik verfügen, die maschinell verarbeitet werden kann. Mit dem Process Composer ist es nicht möglich, Modellelemente mit maschinenlesbarer Semantik zu versehen. Auch Werkzeuge zur Modellierung von Geschäftsprozessen anderer Hersteller (z. B. *ARIS Business Architect* [34] oder *WebSphere Business Modeler Advanced* [35]), bieten keine derartige Funktionalität. Zur Lösung dieser Problematik werden im weiteren Verlauf entsprechende Konzepte auf Grundlage von Techniken des semantischen Webs vorgestellt.

Das Grundprinzip dieser Konzepte besteht darin, Wissen (z. B. über den in Abschnitt 3.1 beschriebenen Ablauf der Flugzeugwartung) zunächst in Form einer Ontologie (siehe Abschnitt 2.3.3.1) zu repräsentieren, um im Anschluss Elemente eines BPMN-Modells mit diesem Wissen semantisch anzureichern. Zu diesem Zweck muss das verwendete Modellierungswerkzeug sowohl auf Ontologien zugreifen können als auch die Zuordnung von Elementen einer Ontologie zu Elementen eines BPMN-Modells unterstützen. Zunächst seien in diesem Zusammenhang zwei Begriffe definiert:

Definition 4.1 (**Semantische Anreicherung eines Modellelements**)
Die semantische Anreicherung eines Modellelements ist dessen Zuordnung zu einer oder mehreren Klassen einer Ontologie.

Definition 4.2 (**Zugeordnete Semantik eines Modellelements**)
Die zugeordnete Semantik eines Modellelements ist die Menge der Klassen einer Ontologie, die dem Modellelement zugeordnet sind.

Um die folgenden Betrachtungen in Bezug auf Ontologien zu konkretisieren, wird im weiteren Verlauf von OWL-Ontologien ausgegangen, da die Ontologiesprache OWL auf einem anerkannten Industriestandard beruht. OWL-Ontologien werden üblicherweise mit Werkzeugen wie etwa dem quelloffenen Ontologieeditor *Protégé* [36] oder der kommerziellen SemanticWeb-Middleware OntoBroker erstellt und in Form von RDF/XML-Dokumenten persistent gespeichert. Der Zugriff auf RDF/XML-Dokumente kann mit entsprechenden Programmierschnittstellen bewerkstelligt werden, beispielsweise mithilfe der quelloffenen *OWL API* [37]. Um den Benutzer nicht mit völlig unterschiedlichen Werkzeugen zur Modellierung von Geschäftsprozessen und OWL-Ontologien konfrontieren zu müssen, werden im Folgenden Möglichkeiten zur Repräsentation von Ontologien auf Basis von MOF diskutiert. Aufgrund der augenscheinlichen Ähnlichkeiten zwischen den Konstrukten zur Erstellung von Metamodellen und Ontologien werden im nächsten Abschnitt zunächst Gemeinsamkeiten und Unterschiede beider Konzepte erörtert und die Frage geklärt, ob ein Metamodell eine OWL-Ontologie verkörpern kann. Da dies – um es bereits vorwegzunehmen – nur bis zu einem gewissen Grad möglich ist, wird im Anschluss ein Standard der OMG vorgestellt, der eine Repräsentation von OWL-Ontologien in Form von Modellen auf Basis eines entsprechenden Metamodells ermöglicht.

4.1. Vergleich zwischen (Meta-) Modellen und Ontologien

Die im Rahmen der MOF-Spezifikation enthaltene Sprache zur Definition von Metamodellen und die Ontologiesprache OWL wurden unabhängig voneinander und mit unterschiedlicher Zielsetzung entwickelt: Während MOF wie in Abschnitt 2.1.1 dargelegt vornehmlich für die Automatisierung der Verwaltung und des Austauschs von Metadaten konzipiert wurde, beschäftigen sich OWL und andere Sprachen zur Wissensrepräsentation sowohl mit der Semantik des Inhalts von Metadaten als auch mit der automatischen Schlussfolgerung darauf basierenden Wissens [FHKM04].

Ähnlich wie die UML-Spezifikation ein UML-Modell als Abstraktion eines physischen Systems mit einem bestimmten Zweck definiert, können Modelle allgemein als abstrahierter Ausschnitt der Realität verstanden werden. Ein Modell entsteht, indem die Elemente dieses Ausschnitts vereinfacht und miteinander in Beziehung gesetzt werden. Auf ähnliche Weise werden bei Ontologien Konzepte einer Domäne miteinander verbunden. Sie können daher ebenfalls als eine Art Modell aufgefasst werden.

4.1.1. Gemeinsamkeiten

Bei den Konstrukten zur Erzeugung von MOF-Metamodellen und OWL-Ontologien gibt es auffallende Gemeinsamkeiten. Sowohl bei Metamodellen als auch bei Ontologien stehen Klassen und deren Instanzen bzw. Individuen im Mittelpunkt, die durch Assoziationen bzw. Rollen miteinander in Beziehung stehen, die wiederum durch die Angabe von Multiplizitäten bzw. Kardinalitäten genauer spezifiziert werden. Eine Übersicht vergleichbarer Konstrukte von MOF und OWL ist in Abbildung 4.2 dargestellt.

MOF-Metamodelle	OWL-Ontologien
Modellelement	Ressource
Klasse	Atomare Klasse (`owl:Class`)
Instanz	Individuum
Attribut	Rolle (`owl:DataTypeProperty`, `owl:ObjectProperty`)
Generalisierung	`rdfs:subClassOf`
Assoziation	Rolle (`owl:ObjectProperty`)
Typ eines Assoziationsendes	`rdfs:domain`, `rdfs:range`
Referenz	Rolle (`owl:ObjectProperty`)
Multiplizität	Kardinalität
Datentypen	Datentypen
Enumeration	Abgeschlossene Klasse (`owl:DataRange`, `owl:oneOf`)
Paket	Ontologie

Abbildung 4.2.: Vergleichbare Konstrukte von MOF und OWL

4.1.2. Unterschiede

Ein grundlegender Unterschied zwischen Metamodellen und Ontologien ist, dass erstere lediglich eine Syntax definieren, während letztere zusätzlich über eine formale Semantik verfügen, wodurch die Schlussfolgerung von Wissen ermöglicht wird. Bezogen auf dieses Wesensmerkmal werden Modelle im Englischen auch nur als *semiformal* eingestuft.

Bei den Gemeinsamkeiten zwischen Metamodellen und Ontologien wurden Klassen und Instanzen bzw. Individuen genannt. Ein Unterschied besteht jedoch darin, dass diese bei MOF viel stärker getrennt sind: Während Metamodellklassen und deren Instanzen nicht innerhalb desselben Modells gespeichert werden können, ist es im Gegensatz dazu bei Ontologien möglich, sowohl Klassen als auch deren Individuen innerhalb derselben Ontologie zu erzeugen.

Zwar sind atomare Klassen bei Ontologien mit Metamodellklassen vergleichbar, OWL bietet jedoch weitere Möglichkeiten, Klassen zu beschreiben. Zum einen können atomare Klassen, die durch einen Klassenbezeichner beschrieben werden, mithilfe logischer Konstruktoren ($\cap$, $\cup$, $\neg$) zu *komplexen Klassen* (Complex Classes) verbunden werden, was bei Metamodellen nicht möglich ist. Bei komplexen Klassen handelt es sich aufgrund des fehlenden Klassenbezeichners um *anonyme Klassen*. Zum anderen können anonyme Klassen auch durch *Rolleneinschränkungen* (Property Restrictions) beschrieben werden, die Klassen als Menge der Individuen beschreiben, für die eine bestimmte Rolle immer einen Wert aus einer vorgegebenen Klasse annimmt. Dabei kann innerhalb der Beschreibung mithilfe des Allquantors ($\forall$) oder Existenzquantors ($\exists$) über alle bzw. mindestens einen der Werte etwas ausgesagt werden. Die im Rahmen des Szenarios entwickelte Flugzeugwartungsontologie (siehe Abbildung 4.5) enthält beispielsweise die Klasse `FlugzeugteilAusbau` (siehe Protégé-Notation in Abbildung 4.3), die den Ausbau eines Flugzeugteils beschreibt und eine Rolleneinschränkung enthält.

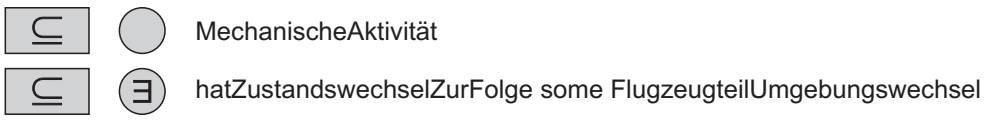


Abbildung 4.3.: Beschreibung der FlugzeugteilAusbau-Klasse

Diese Beschreibung besagt, dass die Klasse `FlugzeugteilAusbau` eine Unterklasse der Klasse `MechanischeAktivität` ist und Individuen dieser Klasse über die Rolle `hatZustandswechselZurFolge` mit einem Individuum der Klasse `FlugzeugteilUmgebungswechsel` in Verbindung gesetzt sind. Rolleneinschränkungen können entweder als *notwendig* (necessary) oder als *notwendig und hinreichend* (necessary and sufficient) definiert werden. Notwendige Rolleneinschränkungen einer Klasse sind so zu verstehen, dass wenn ein Individuum Element dieser Klasse ist, es diese Rolleneinschränkungen erfüllen muss. Eine Klasse, die ausschließlich notwendige Rolleneinschränkungen enthält, wird als *primitive Klasse* (Primitive Class) bezeichnet. Ähnliche Einschränkungen können auch bei Metamodellklassen mithilfe von OCL bewerkstelligt werden. Enthält eine Klasse mindestens eine Rolleneinschränkung notwendiger und hinreichender Natur, so gilt zusätzlich, dass ein Individuum, das diese Rolleneinschränkung erfüllt, Element dieser Klasse ist. Eine Klasse, die mindestens eine notwendige und hinreichende Rolleneinschränkung enthält, wird als *definierte Klasse* (Defined Class) bezeichnet. Ein Beispiel für eine definierte Klasse innerhalb der Flugzeugwartungsontologie ist die Klasse `DokumentationspflichtigeSituation` (siehe Protégé-Notation in Abbildung 4.4), die einen dokumentationspflichtigen Arbeitsschritt oder Zustand beschreibt.

Abbildung 4.4.: Beschreibung der DokumentationspflichtigeSituation-Klasse

Die Beschreibung dieser Klasse enthält eine notwendige und hinreichende Rolleneinschränkung, die besagt, dass ein Individuum, das Element der Klasse `SicherheitskritischeAktivität` oder der Klasse `SicherheitskritischerZustand` ist, auch Element der Klasse `DokumentationspflichtigeSituation` ist.

Es sind diese definierten Klassen, die einen weiteren großen Unterschied zu Metamodellen ausmachen, abgesehen davon, dass es ein zu definierten Klassen vergleichbares Konzept bei Metamodellen nicht gibt. Sofern gilt, dass alle Elemente einer Klasse, beispielsweise die der `FlugzeugteilAusbau`-Klasse, die Bedingungen einer weiteren definierten Klasse erfüllen, beispielsweise die der `DokumentationspflichtigeSituation`-Klasse, kann geschlussfolgert werden, dass die `FlugzeugteilAusbau`-Klasse eine Unterklasse der `DokumentationspflichtigeSituation`-Klasse ist, ohne dass dies in der ursprünglichen Ontologie explizit angegeben wurde. Daher wird im Englischen zwischen der *zugesicherten Hierarchie* (Asserted Hierarchy) und der *inferierten Hierarchie* (Inferred Hierarchy) unterschieden. Die Berechnung der inferierten Hierarchie wird als *Klassifikation der Ontologie* bezeichnet, im Englischen auch als *Subsumption Reasoning*. Im Gegensatz zu einer nicht klassifizierten Ontologie ist die Klassenhierarchie eines Metamodells nach Abschluss der Modellierung unveränderlich. Ein Ausschnitt der Flugzeugwartungsontologie vor und nach deren Klassifikation ist in Abbildung 4.5 dargestellt.

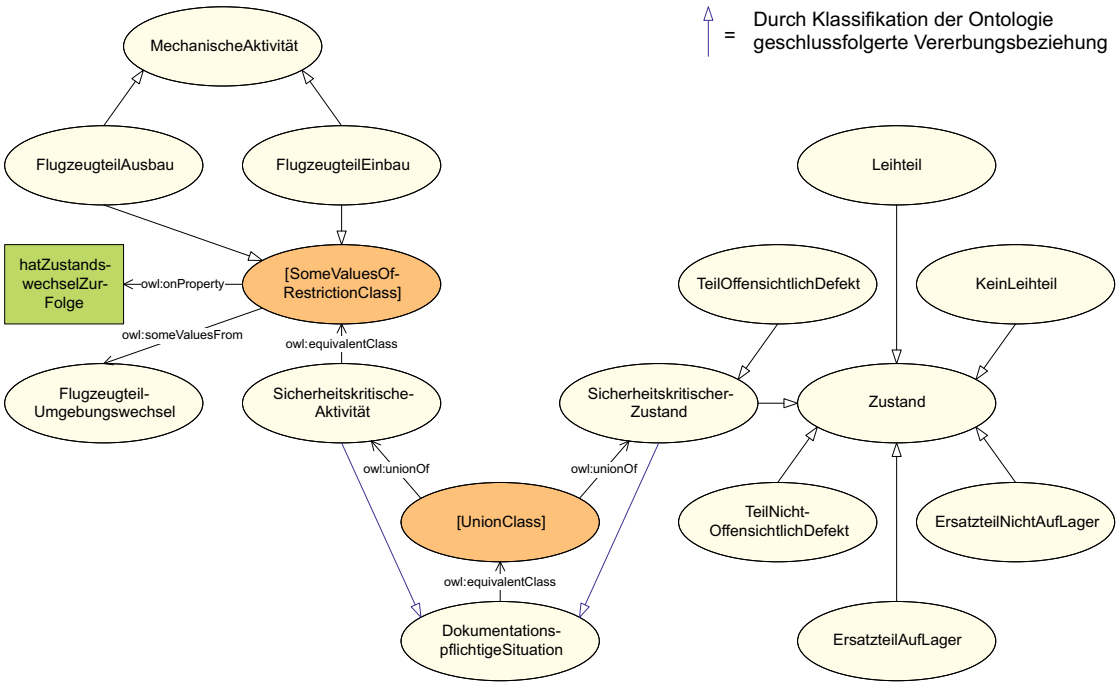


Abbildung 4.5.: Ausschnitt der klassifizierten Flugzeugwartungsontologie

Abgesehen von den Unterschieden zwischen Ontologien und Metamodellen unterscheiden sich Modelle auf deren Grundlage von Ontologien grundsätzlich darin, dass man bei ersteren in der Regel von der *Geschlossenen-Welt-Annahme* (Closed World Assumption)

ausgeht, während letztere grundsätzlich auf der *Offenen-Welt-Annahme* (Open World Assumption) basieren. Bei der Geschlossenen-Welt-Annahme im Kontext von Wissensbasen geht man davon aus, dass sämtliche relevanten Fakten in einer Wissensbasis vorhanden sind, während bei der Offenen-Welt-Annahme davon ausgegangen wird, dass eine Wissensbasis immer potentiell unvollständig ist [HKRS08]. Hinsichtlich der Repräsentation von physischen Systemen unterscheiden Aßmann et al. dabei zwischen Modellen, die Systeme beschreiben und solchen, die das Verhalten von Systemen vorschreiben. Ihrer Meinung nach sind Ontologien aufgrund der Offenen-Welt-Annahme immer beschreibender Natur und können daher etwa im Rahmen der Softwaretechnik nicht zur Spezifikation des Verhaltens eines Systems verwendet werden [AZW06]. Im Gegensatz dazu bezeichnet Devedžić [Dev02] Ontologien explizit als Metamodelle zur Erstellung von Modellen und schließt somit deren Verwendung zur Spezifikation von Systemen nicht aus.

Aufgrund ihrer verschiedenartigen Wesensmerkmale wird deutlich, dass eine bidirektionale Abbildung zwischen Ontologien und Metamodellen nicht möglich ist. Dennoch führen ihre Gemeinsamkeiten in letzter Zeit dazu, dass sich die Forschung auf beiden Gebieten einander annähert. In [PSW07] werden diesbezügliche Ansätze in vier verschiedene Kategorien unterteilt: Prüfung der internen Konsistenz von Klassendefinitionen durch deren Transformation in eine Ontologie und logisches Schlussfolgern, Abbildungen zwischen Modellen mithilfe semantischer Techniken, Modellierung von Ontologien auf Grundlage graphischer Modellierungssprachen sowie Mischansätze. Für diese Arbeit ist die dritte Kategorie besonders interessant. Die Arbeit auf diesem Gebiet beschäftigt sich damit, wie Ontologien trotz der in diesem Abschnitt beschriebenen Unterschiede auf Basis von MOF – nämlich in Form von Modellen anstelle von Metamodellen – repräsentiert werden können.

4.2. Repräsentation von Ontologien auf Basis von MOF

Derzeit gibt es für die Repräsentation von Ontologien auf Basis von MOF einen von der OMG veröffentlichten Standard, der in dieser Arbeit als Grundlage für die Entwicklung eines eigenen Metamodells für OWL-Ontologien verwendet wurde und im folgenden Abschnitt vorgestellt wird. Im Anschluss daran wird erläutert, was beim Zugriff auf Modelle auf Basis dieses Metamodells beachtet werden muss.

4.2.1. Ontology Definition Metamodel

Das im Mai 2009 von der OMG veröffentlichte *Ontology Definition Metamodel* (ODM) [OMG09b] zielt darauf ab, die Entwicklung von Ontologien auf Basis der modellgetriebenen Architektur zu ermöglichen. Die ODM-Spezifikation beschreibt vier Metamodelle, welche die Wissensrepräsentationssprachen RDF, OWL, Common Logic [ISO07] und

Topic Maps [ISO03] auf Grundlage von MOF definieren. Darüber hinaus werden zugehörige UML-Profile und Abbildungsvorschriften zwischen diesen Metamodellen spezifiziert. Die Motivation zur Entwicklung der Spezifikation war die Annahme, dass semantische Techniken einem größeren Publikum zugänglich gemacht werden könnten, wenn es möglich wäre, Ontologien auf Grundlage der graphischen Modellierungssprache UML modellieren zu können, deren Verwendung im Bereich der softwarebasierten Systemmodellierung dominierend ist. Mithilfe des in der ODM-Spezifikation beschriebenen OWL-Metamodells können OWL-Ontologien auf diese Weise als Modell anstelle eines RDF/XML-Dokuments repräsentiert werden. Allerdings konnte das ODM im Rahmen dieser Arbeit nicht verwendet werden, da es auf Version 2.0 der MOF-Spezifikation basiert, während die verwendete Modellierungsinfrastruktur lediglich Version 1.4 unterstützt. Aus diesem Grund wurde ein eigenes Metamodell für OWL auf Basis von Version 1.4 konzipiert, das sich an das ODM anlehnt, allerdings nur soweit, wie es für die vorliegende Arbeit von Nutzen war (siehe Abschnitt A.2 im Anhang).

4.2.2. Zugriff auf OWL-Ontologien im Process Composer

Soll innerhalb des Process Composers ein BPMN-Modellelement (z. B. ein Task) mit maschinenlesbarer Semantik angereichert werden, muss der Benutzer zunächst die Möglichkeit haben, aus den verfügbaren Ontologiemodellen eine oder mehrere Klassen auszuwählen, welche die Semantik des Modellelements beschreiben. Die Klassenhierarchie, die dem Benutzer zu diesem Zweck angezeigt werden soll, ist dabei durch die Assoziation `ClassGeneralization` des OWL-Metamodells (siehe Abbildung A.2 im Anhang) vorgegeben. Allerdings kann mithilfe dieser Assoziation nur auf die zugesicherte Hierarchie zugegriffen werden. Diese Tatsache ist jedoch unbefriedigend, da die inferierte Hierarchie wie in Abschnitt 4.1.2 erläutert aufgrund definierter Klassen davon abweichen kann. Um diese Problematik zu lösen, könnte die Anforderung an Ontologien gestellt werden, dass sich ihre Klassenhierarchie durch deren Klassifikation nicht ändern darf und damit die zugesicherte Hierarchie mit der inferierten Hierarchie übereinstimmen muss. Da diese Auflage jedoch sehr einschränkend wäre, muss die Klassifikation der Ontologie innerhalb des Modellierungswerkzeugs vorgenommen werden, wodurch zwangsweise zusätzliche Software benötigt wird, genauer gesagt ein Reasoner (siehe Abschnitt 2.3.3.3).

Der im Rahmen dieser Arbeit erweiterte Process Composer ist in der Programmiersprache Java implementiert. Zur Klassifikation von Ontologien wurde aus diesem Grund der in Abschnitt 2.3.3.3 erwähnte quelloffene OWL-Reasoner Pellet verwendet, der ebenfalls auf Java basiert. Pellet unterstützt die OWL API und akzeptiert zur Klassifikation einer Ontologie als Eingabe eine Menge aus Objekten, welche eine bestimmte Schnittstelle der OWL API implementieren müssen. Nach Abschluss der Klassifikation entsprechen die Vererbungsbeziehungen zwischen den Klassen innerhalb der übergebenen Menge der inferierten Hierarchie. Um ein Objektmodell einer Ontologie auf Basis des OWL-Metamodells in die zur Klassifikation benötigte Repräsentation in Form eines Objektmodells auf Grundlage der OWL API überführen zu können, wurde ein entsprechen-

der Algorithmus konzipiert. Dieser Algorithmus durchläuft während der Transformation die folgenden Phasen:

1. Zunächst werden alle atomaren Klassen erzeugt.
2. Danach werden alle Rollen erzeugt, wobei Rollenbeziehungen (`owl:inverseOf`, `rdfs:subPropertyOf`) und Rolleneigenschaften (funktional, invers funktional, symmetrisch, transitiv, `rdfs:domain`, `rdfs:range`) berücksichtigt werden.
3. Schließlich werden alle einfachen Klassenbeziehungen erzeugt (`owl:disjointWith`, `owl:equivalentClass`, `rdfs:subClassOf`). Dieser Vorgang erfolgt rekursiv, da es sich bei den dabei in Beziehung gesetzten Klassen um anonyme Klassen handeln kann (`owl:oneOf`, `owl:Restriction`, `owl:intersectionOf`, `owl:unionOf`, `owl:complementOf`), die zunächst erzeugt werden müssen und wiederum selbst mit anonymen Klassen in Beziehung stehen können.

Die Aufteilung des Algorithmus in Phasen ist darauf zurückzuführen, dass während der Transformation Konstrukte unterschiedlichen Typs erzeugt werden, die aufeinander aufbauen. Die Wahl der obigen Reihenfolge beruht auf der Tatsache, dass zur Erzeugung von Konstrukten in der zweiten und dritten Phase auf Konstrukte der ersten bzw. zweiten Phase zugegriffen werden muss.

4.3. Interne Repräsentation semantischer Anreicherungen

Nachdem im letzten Abschnitt erläutert wurde, wie der Process Composer auf OWL-Ontologien und die darin enthaltenen Klassenbeschreibungen zugreifen kann, ist noch zu klären, wie eine semantische Anreicherung eines BPMN-Modellelements mit einer oder mehreren dieser Klassen innerhalb des Modellierungswerkzeugs umgesetzt werden kann. Hierfür gibt es verschiedene Möglichkeiten: Zum einen wäre es möglich, die Zuordnung zu Klassen einer Ontologie ohne Veränderung des BPMN-Metamodells mit vorhandenen Mitteln im Geschäftsprozessmodell unterzubringen, zum anderen könnte das BPMN-Metamodell um diese Zuordnung gezielt erweitert werden. Ohne Veränderung des BPMN-Metamodells könnte beispielsweise das bei Flussobjekten vorhandene Dokumentationsfeld verwendet werden, indem die URIs der zu diesem Modellelement zugeordneten Klassen einer Ontologie als kommaseparierte Zeichenkette dort abgelegt werden. Allerdings wird dadurch dessen Verwendung zu anderen Zwecken ausgeschlossen. Eine weitere Möglichkeit wäre, die URIs der zugeordneten Klassen mithilfe von Anmerkungen direkt im Diagramm unterzubringen, wie es in [FT08] vorgeschlagen wird.

Derartige Vorgehensweisen haben allerdings einen eklatanten Nachteil, da hier keine enge Kopplung zwischen der im Geschäftsprozessmodell in Textform angegebenen URI und der zugehörigen OWL-Klasse besteht – unabhängig davon, ob letztere innerhalb eines RDF/XML-Dokuments als XML-Element oder innerhalb eines Modells repräsentiert ist. Wird eine OWL-Klasse beispielsweise gelöscht, sollte dies im Idealfall dazu führen,

dass Geschäftsprozesselemente, die mit dieser Klasse semantisch angereicht wurden, entsprechend angepasst werden, indem auch die semantische Anreicherung entfernt wird. Aufgrund der losen Kopplung bleibt die Zuordnung zur URI der gelöschten Klasse jedoch bestehen und könnte bei einer darauffolgenden Analyse zu Fehlern führen.

Aus diesem Grund wird in dieser Arbeit ein neuer Ansatz vorgeschlagen, der eine enge Kopplung zwischen den Elementen eines Geschäftsprozessmodells und solchen eines Ontologiemodells ermöglicht. Dieser Ansatz sieht ein drittes Metamodell vor, das mit dem BPMN-Metamodell und dem OWL-Metamodell *föderiert* wird. Unter einer Metamodellföderation versteht man den Prozess der Wiederverwendung eines Metamodells innerhalb eines anderen Metamodells. Innerhalb dieses dritten Metamodells, dem *ExtendedBPMN-Metamodell* (siehe Abschnitt A.3 im Anhang), wurde eine Unterklasse jeder MOF-Klasse des BPMN-Metamodells erzeugt, deren Instanzen semantisch angereichert werden können (z. B. eine Unterklasse der Metamodellklasse `Task` des BPMN-Metamodells). Ein Ausschnitt des Metamodells ist in Abbildung 4.6 dargestellt.

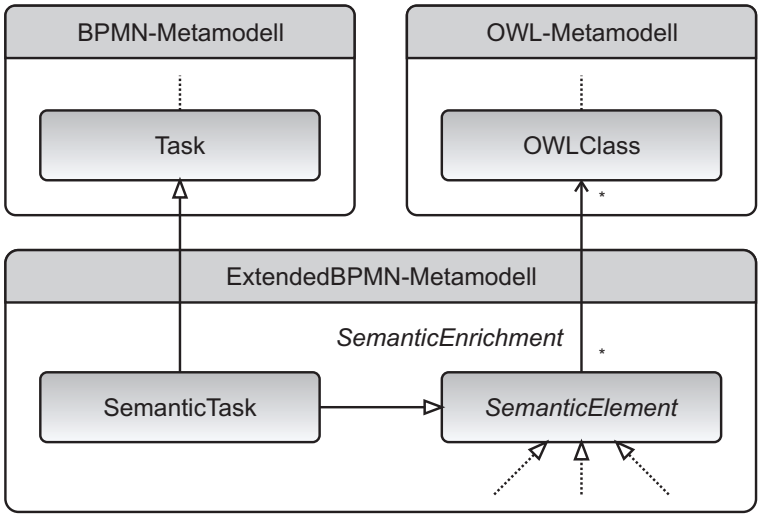


Abbildung 4.6.: Ausschnitt des ExtendedBPMN-Metamodells

Die zentrale Klasse dieses Metamodells ist die abstrakte Klasse `SemanticElement`. Aufgrund der Assoziation `SemanticEnrichment` können Instanzen einer Unterklasse dieser Klasse mit einer beliebigen Anzahl von Instanzen der Klasse `OWLClass` verbunden werden. Die exemplarisch abgebildete Klasse `SemanticTask` ist eine Unterklasse der Klasse `Task` des BPMN-Metamodells und der Klasse `SemanticElement`. Instanzen dieser Klasse erben dadurch alle Eigenschaften normaler Tasks und können so vom Modellierungswerkzeug problemlos verarbeitet werden, ermöglichen nun aber auch eine enge Kopplung mit OWL-Klassen innerhalb eines Ontologiemodells. Wird nun eine referenzierte OWL-Klasse gelöscht, entfernt die zugrunde liegende Modellierungsinfrastruktur auch automatisch Referenzen, die sich auf diese Klasse beziehen.

4.4. Implementierung

Ein Ziel bei der Implementierung der in diesem Kapitel vorgestellten Konzepte war deren Integration in den Process Composer und dessen Umgebung. Der Process Composer ist ein Bestandteil der Entwicklungsumgebung *SAP NetWeaver Developer Studio* [38], mit der Anwendungen für die SOA-Plattform *SAP NetWeaver Composition Environment* [39] entwickelt werden können. Die Entwicklungsumgebung selbst basiert auf der quelloffenen *Eclipse-Plattform* [40]. Der Process Composer wie auch die anderen Bestandteile des SAP NetWeaver Developer Studios liegen wie bei Eclipse vorgesehen in Form sogenannter *Eclipse Plug-ins* vor. Aus diesem Grund wurde auch der im Rahmen dieser Arbeit entwickelte Prototyp auf diese Weise implementiert. Eine Übersicht der entwickelten und modifizierten Eclipse Plug-ins, bei denen es sich um eine Implementierung der in diesem Kapitel vorgestellten Konzepte handelt und die im Folgenden näher beschrieben werden, ist in Abbildung 4.7 dargestellt.

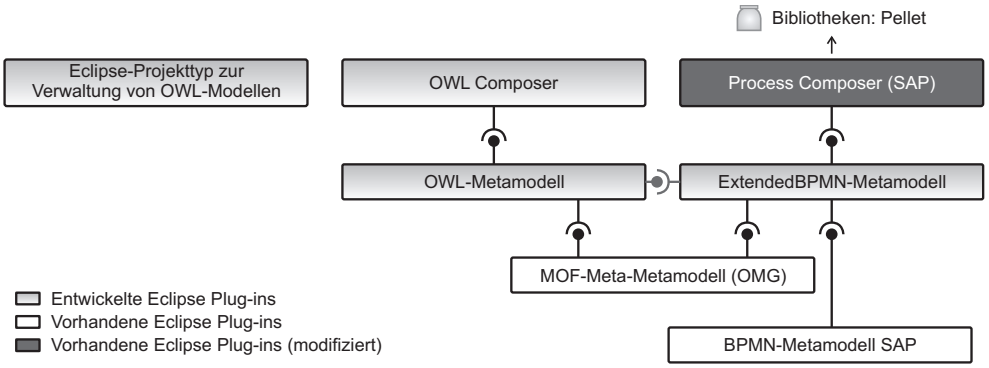


Abbildung 4.7.: Entwickelte und modifizierte Eclipse Plug-ins

4.4.1. OWL- und ExtendedBPMN-Metamodell

Ein Metamodell wird mithilfe eines speziellen Eclipse-Projekttyps erzeugt, dem *Metamodel Project*, in dem mit entsprechenden Werkzeugen und der zugrunde liegenden Modellierungsinfrastruktur Metamodellelemente erstellt werden können. Um jedoch mit einem Metamodell programmatisch arbeiten zu können, genauer gesagt auf Basis von JMI (siehe Abschnitt 2.1.1), muss darüber hinaus mithilfe eines dafür vorgesehenen Assistenten ein Plug-in-Projekt erzeugt werden, in dem die mit dem Metamodell korrespondierenden JMI-Klassen automatisch generiert werden. Schließlich muss dieses Plug-in exportiert und beim Start der Eclipse-Plattform geladen werden. Diese Vorgehensweise wurde folglich auch auf das OWL- und das ExtendedBPMN-Metamodell angewandt.

4.4.2. Graphisches Modellierungswerkzeug für OWL-Ontologien

Zur einfachen Modellierung von OWL-Ontologien auf Basis des OWL-Metamodells wurde mit dem *OWL Composer* ein entsprechendes graphisches Modellierungswerkzeug entwickelt, das im Vergleich zu anderen Ontologie-Editoren lediglich eine rudimentäre Funktionalität bietet. Der Beweggrund für die Entwicklung dieses Werkzeugs war jedoch das Ziel, mit diesem Werkzeug auf schnelle Art und Weise Ontologiemodelle anfertigen zu können, auf die der Process Composer direkt zugreifen kann.

Zur Verringerung des Entwicklungsaufwands von graphischen Modellierungswerkzeugen bietet das NetWeaver Developer Studio ein Graphik-Framework an, das eine Abstraktionsschicht zwischen MOF-Modellen sowie deren graphischer Darstellung bietet und auf dem *Graphical Editing Framework* (GEF) [41] basiert. Graphische Modellierungswerkzeuge auf Grundlage dieses Frameworks – im Folgenden auch Editoren genannt – werden als Eclipse-Plug-ins realisiert. Bei der Initialisierung eines Editors muss für jedes Metamodellelement, das in der Werkzeugleiste des Editors zur Auswahl stehen soll, spezifiziert werden, wie sich dessen graphische Darstellung zusammensetzt. Diese Spezifikation erfolgt durch Erzeugung von Modellelementen auf Basis des *Piktogramm-Metamodells* des Graphik-Frameworks. Die Modellelemente dieses Metamodells repräsentieren einfache geometrische Formen, für deren Darstellung das Framework verantwortlich ist. Auf diese Weise wurde auch der OWL Composer entwickelt, der in Abbildung 4.8 dargestellt ist. Darüber hinaus wurde ein eigener Eclipse-Projekttyp zur Speicherung von Ontologiemodellen eingeführt und der Projektexplorer erweitert, um innerhalb von Projekten dieses Typs Ontologiemodelle komfortabel verwalten zu können.

4.4.3. Modifikation des Modellierungswerkzeugs

Nach dieser Vorarbeit wurde schließlich der Process Composer um eine Funktion erweitert, die es erlaubt, BPMN-Modellelemente semantisch anzureichern. Zunächst wurde die Werkzeugleiste des Modellierungswerkzeugs um zusätzliche Modellierungskonstrukte erweitert, beispielsweise um den *semantischen Task*, hinter dem sich die Klasse `SemanticTask` des ExtendedBPMN-Metamodells verbirgt. Zusätzlich wurde die Eigenschaftsansicht, die bei der Auswahl eines Tasks angezeigt wird, um einen weiteren Reiter erweitert (siehe Abbildung 4.9). Dieser wird jedoch nur angezeigt, wenn es sich bei einem ausgewählten Task um einen semantischen Task handelt.

Auf dem zur Eigenschaftsansicht hinzugefügten Reiter sind die OWL-Klassen aufgelistet, mit denen der Task semantisch angereichert wurde, wobei jeweils die URI-Referenz der OWL-Klasse, das Ontologiemodell, in dem die Klasse enthalten ist, sowie das zugehörige Eclipse-Projekt angezeigt wird. Ausgehend von diesem Reiter kann ein Dialog zur Zuordnung von OWL-Klassen geöffnet werden, der in Abbildung 4.10 dargestellt ist. In der oberen Liste auf der linken Seite werden alle Ontologieprojekte im Eclipse-Arbeitsbereich angezeigt. Bei Auswahl eines Projekts werden in der oberen Liste auf der rechten Seite

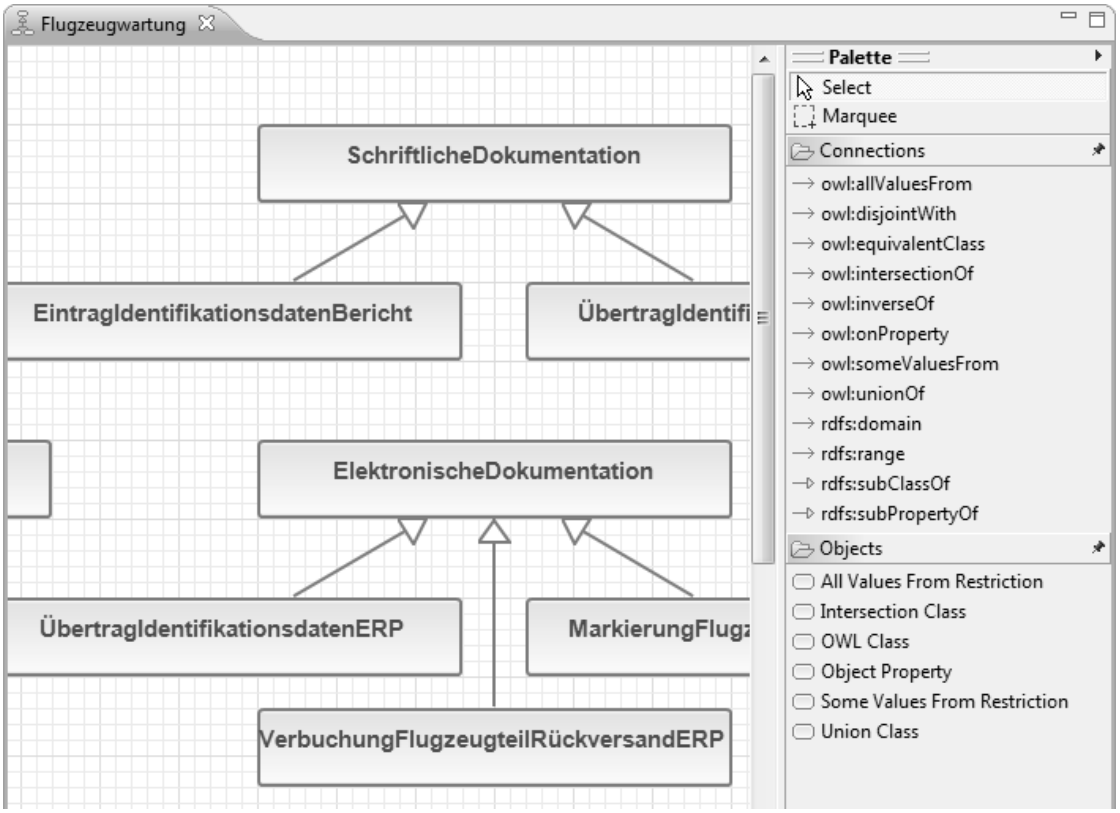


Abbildung 4.8.: OWL Composer

alle darin enthaltenen Ontologiemodelle aufgelistet. Nach der Selektion eines Ontologiemodells wird dieses zunächst mithilfe des in Abschnitt 4.2.2 vorgestellten Algorithmus in ein Objektmodell auf Basis der OWL API transformiert. Nach Abschluss der darauffolgenden Klassifikation der Ontologie werden deren Klassen in einer Baumstruktur angezeigt, die der vom Reasoner berechneten inferierten Klassenhierarchie entspricht. Wird der Dialog geschlossen, wird der im Process Composer ausgewählte Task mit den in der Baumstruktur markierten Klassen verknüpft und verfügt damit über eine eindeutige Semantik, die maschinell verarbeitet werden kann, unabhängig von ihrer Beschriftung in natürlicher Sprache.

4.5. Stand der Wissenschaft und Technik

Das in diesem Kapitel vorgestellte Konzept zur Anreicherung von BPMN-Modellen mit zusätzlicher Semantik auf MOF-Ebene kann in den Bereich des *semantischen Geschäftsprozessmanagements* (Semantic Business Process Management) [HLD+05, HR07] eingeordnet werden. Das semantische Geschäftsprozessmanagement beschäftigt sich mit der

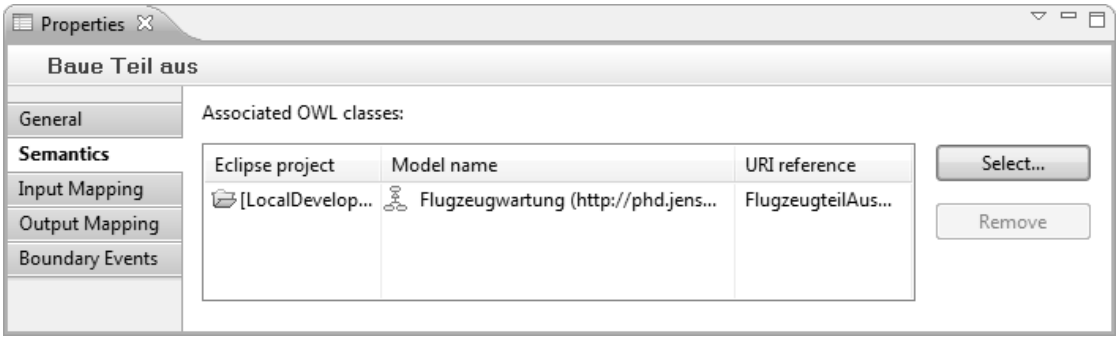


Abbildung 4.9.: Eigenschaftsansicht eines semantischen Tasks

Verbindung von Business-Process-Management-Systemen und Techniken aus dem Bereich des semantischen Webs. Ziel des semantischen Geschäftsprozessmanagements ist es, den Automatisierungsgrad bei der bidirektionalen Übersetzung von Anforderungen der betriebswirtschaftlichen Ebene an die Geschäftsprozesse eines Unternehmens und den korrespondierenden ausführbaren Geschäftsprozessmodellen auf der technischen Ebene zu erhöhen. Zu diesem Zweck werden die verschiedenen Ebenen eines Unternehmens im Rahmen des semantischen Geschäftsprozessmanagements in Form von Ontologien und semantischen Webdiensten repräsentiert. Auf diese Weise kann die Übersetzung mithilfe von Reasonern automatisiert werden.

Im Zusammenhang des semantischen Geschäftsprozessmanagements wird vorgeschlagen, die funktionalen Aspekte von Prozessen und darin enthaltener Tasks zu ontologisieren (d. h. in Form von Ontologien zu repräsentieren), indem beispielsweise die domänenspezifische Funktion eines Tasks in Form eines Konzepts beschrieben und diesem Task zugeordnet wird [WMF+07]. Ein ähnlicher Ansatz wird in [Lin08] vorgestellt. In beiden Fällen wird dieser Vorgang als *semantische Annotation* bezeichnet. In [LB07] wird eine Übersicht über Ansätze zur semantischen Annotation von Geschäftsprozess- und Workflow-Modellen präsentiert, wobei in dieser Arbeit Annotationen unter dem Aspekt der automatischen Komposition von Geschäftsprozess- oder Workflow-Modellen betrachtet werden. Im Rahmen des SUPER-Projekts [42] wurden mehrere Ontologien [AFKK07, NWv07, FKS09] auf Grundlage der *Web Service Modeling Language* (WSML) und zugehörige Modellierungswerkzeuge [BDW07, DSSK07, BHK+08] entwickelt, um Geschäftsprozess- und Workflow-Modelle auf Basis von BPMN, *Ereignisgesteuerten Prozessketten* (EPKs) oder auf Basis der *Business Process Execution Language* (BPEL) abbilden zu können. Diese Ontologien erlauben auch die Annotation von Instanzen der darin beschriebenen Konzepte (z. B. Tasks) mit Konzepten innerhalb von Domänenontologien. Ein vergleichbarer Ansatz auf Basis Ereignisgesteuerter Prozessketten wird in [TF06, TF07a, TF07b] beschrieben, ein vergleichbarer Ansatz auf Grundlage von BPMN in [FT08, FGR+08, Fra08]. In [FG09] wird argumentiert, dass die Verwendung terminologischer Industriestandards, die in Ontologien überführt wurden, eine ideale Grundlage zur Annotation von Geschäftsprozessmodellen darstellt, da der Aufwand zur Erstellung

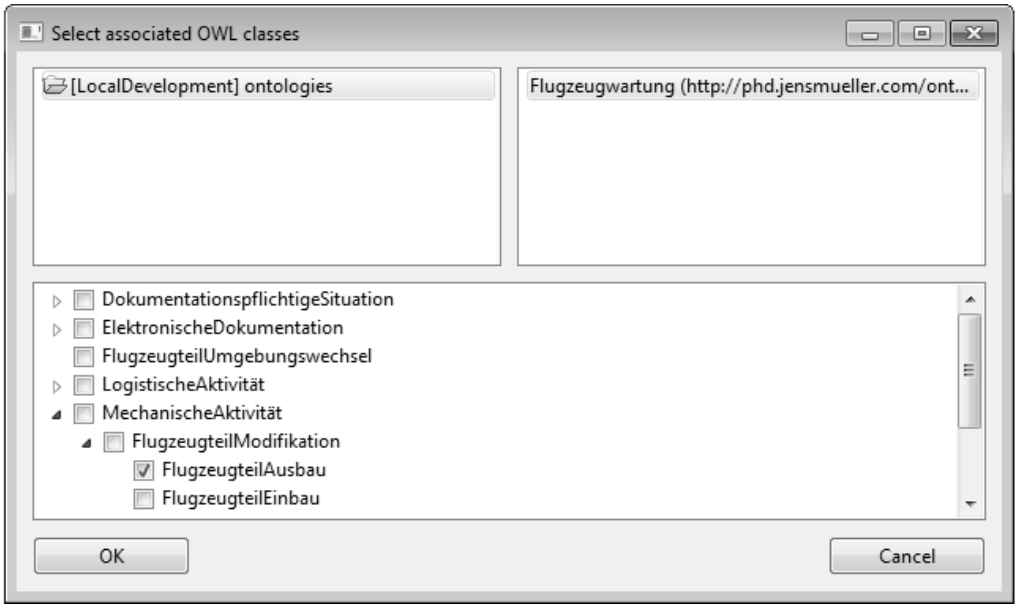


Abbildung 4.10.: Dialog zur Auswahl von OWL-Klassen

dieser Ontologien relativ gering ist. Allerdings sind in vielen Industrien derartige Standards nicht vorhanden. Um in solchen Fällen den Aufwand zur Erstellung von Ontologien zu reduzieren, wird in [FR09] ein Ansatz vorgeschlagen, der diesen Vorgang unterstützt. In diesem Ansatz werden Domänenontologien automatisch auf Grundlage von Geschäftsprozessmodellen erzeugt, indem für jeden Namen der darin enthaltenen Tasks eine Ontologieklasse erzeugt wird. Im nächsten Schritt werden die Klassen unterschiedlicher Ontologien automatisch miteinander verglichen und mögliche semantische Beziehungen vorgeschlagen. Auf Grundlage dieser automatisch erzeugten Ontologien können Benutzer Korrekturen vornehmen und weitere Beziehungen hinzufügen. Die auf diese Weise extrahierten Beziehungen können dann die Beschriftung von Tasks bei der Modellierung von Geschäftsprozessen unterstützen.

Insgesamt wird deutlich, dass Ontologien sowohl zur Repräsentation von Geschäftsprozessmodellen als auch zur expliziten Abbildung des über diese Modelle hinaus vorhandenen Wissens geeignet sind. Die genannten Ansätze haben gemeinsam, dass zur Verwaltung von Geschäftsprozessdiagrammen und Ontologien und zur Repräsentation semantischer Anreicherungen ausschließlich Techniken des semantischen Webs verwendet werden, beispielsweise indem ontologisierte BPMN-Diagramme inklusive semantischer Annotationen in Form von RDF/XML-Dateien gespeichert werden. Die in diesem Kapitel vorgestellte Technik zur semantischen Anreicherung von Elementen eines Geschäftsprozessmodells ist mit den zuvor genannten Ansätzen vergleichbar. Allerdings unterscheidet sich die in dieser Arbeit vorgeschlagene Vorgehensweise darin, dass sowohl die Verwaltung von Geschäftsprozessdiagrammen und Ontologien als auch die Repräsentation semantischer Anreicherungen mithilfe modellgetriebener Techniken bewerkstelligt

wird und Ontologiemodelle nur bei Bedarf in das vom verwendeten Reasoner benötigte Objektmodell transformiert werden.

Aus der Verwendung modellgetriebener Techniken ergeben sich in diesem Zusammenhang mehrere Vorteile. Die Vorzüge eines MOF-Repositoriums liegen darin, dass es grundlegende Dienste zur Verwaltung und zum Austausch von Metadaten bietet. Hinsichtlich der Verwaltung übernimmt ein MOF-Repositorium üblicherweise die Konsistenzüberprüfung von Modellen, ist für deren Persistenz und Versionierung verantwortlich und stellt standardisierte Schnittstellen für den Modellzugriff zur Verfügung. Bei der Konsistenzüberprüfung wird beispielsweise überprüft, ob ein Modell die durch das zugehörige Metamodell vorgegebenen Regeln und Einschränkungen einhält. Da vergleichbare Regeln und Einschränkungen mit den Sprachkonstrukten von OWL nur ansatzweise abgebildet werden können, kann ein Reasoner fehlerhafte BPMN-Ontologien eventuell nicht erkennen; allerdings existiert eine proprietäre Erweiterung zur Spezifikation von Integritätsbedingungen für RDF-Graphen in Form von OWL-Ontologien [43].

Zur Realisierung semantischer Annotationen auf Ontologieebene wurden mehrere Vorgehensweise vorgeschlagen [Sei08, S. 31-34]. In [TF06, TF07a, TF07b] wird beispielsweise eine entsprechende Rolle verwendet, die Individuen, die innerhalb der *Geschäftsprozessontologie* enthalten sind und deren Klasse dem jeweiligen Element der graphischen Notation entspricht, mit Individuen verbindet, welche die annotierten Domänenkonzepte repräsentieren. Mit dem Begriff Geschäftsprozessontologie wird die Ontologie bezeichnet, die einem Geschäftsprozessdiagramm zugrunde liegt. In [FT08] wird zusätzlich zur Geschäftsprozessontologie und den Domänenontologien eine weitere Ontologie erzeugt, die für jede Kombination (ohne Duplikate) aus Elementen des Geschäftsprozessdiagramms und den damit annotierten Klassen aus den Domänenontologien eine Klasse enthält, die als Unterklasse der entsprechenden Klassen definiert ist. Diese Klassen dienen als Grundlage der Individuen innerhalb der Geschäftsprozessontologie.

Die in dieser Arbeit vorgeschlagene Vorgehensweise zur semantischen Anreicherung auf Modellebene hat gegenüber den obigen Ansätzen mehrere Vorteile. Zum einen gibt es eine klare Trennung zwischen BPMN-Metamodell und den als MOF-Modellen vorliegenden Domänenontologien. Da das ExtendedBPMN-Metamodell eine Erweiterung des BPMN-Metamodells darstellt, musste letzteres nicht modifiziert werden. Zum anderen profitieren Benutzer auch bei der semantischen Anreicherung von den Diensten eines MOF-Repositoriums, beispielsweise von der festen Kopplung zwischen den Elementen eines BPMN-Modells und den ihnen zugeordneten Klassen eines Ontologiemodells. Wird eine dieser Klassen innerhalb eines Ontologiemodells gelöscht, stellt das MOF-Repositorium sicher, dass auch etwaige Zuordnungen zu den Elementen von BPMN-Modellen gelöscht und damit Inkonsistenzen verhindert werden. Darüber hinaus werden manuell verursachte Inkonsistenzen automatisch entdeckt. Ein Nachteil dieses Ansatzes ist, dass zur Auswertung der zugeordneten Semantik eines BPMN-Modells die Ontologiemodelle zunächst in das vom verwendeten Reasoner benötigte Objektmodell transformiert werden müssen, was während der Modellierung allerdings nur bei sehr großen Ontologien zu wahrnehmbaren Verzögerungen führen dürfte.

5. Modellierung von Anforderungen an BPMN-Modelle

Bei der Modellierung von Geschäftsprozessen mit dem Process Composer existieren bedingt durch das zugrunde liegende Metamodell und die darin enthaltenen OCL-Ausdrücke gewisse Einschränkungen. Diese Einschränkungen legen fest, wie Instanzen der zur Verfügung stehenden Modellierungskonstrukte des Metamodells verwendet und miteinander in Beziehung gesetzt werden dürfen. Wie bereits in Kapitel 2.2.2 erwähnt, werden Geschäftsprozesse üblicherweise modelliert, um sie zu dokumentieren, zu optimieren oder sie hinsichtlich ihrer Automatisierung in eine maschinenlesbare Form zu überführen.

Wie am Beispiel des Szenarios in Kapitel 3 veranschaulicht, müssen bei der Modellierung von Geschäftsprozessen über die syntaktischen Einschränkungen hinaus jedoch meist zusätzliche Anforderungen der betriebswirtschaftlichen Ebene berücksichtigt werden, die in Form von Aussagen (in natürlicher Sprache) über die notwendige Beschaffenheit der Struktur von Geschäftsprozessmodellen formuliert werden können. Die in Abbildung 5.1 dargestellte *Anforderung 1* des Szenarios (siehe Abschnitt 3.4.1) macht beispielsweise eine Aussage über die Existenz einer bestimmten Anordnung von Modellelementen mit Bezug auf deren inhaltliche Bedeutung, indem gefordert wird, dass das Wartungsprozessmodell genau zwei Sequenzen aus jeweils zwei bis drei Arbeitsschritten enthält, wobei im ersten Schritt ein Flugzeugteil ein- oder ausgebaut wird und im nächsten Schritt die Identifikationsdaten dieses Teils im Inspektionsbericht dokumentiert werden. Darüber hinaus darf zwischen diesen Schritten maximal ein weiterer Arbeitsschritt erfolgen.

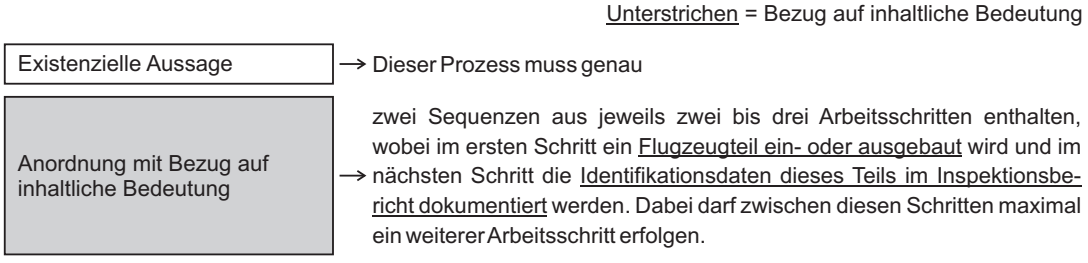


Abbildung 5.1.: Inhaltliche Bestandteile von Anforderung 1 des Szenarios

Im Gegensatz dazu macht die in Abbildung 5.2 dargestellte und bereits diskutierte *Anforderung 7* des Szenarios (siehe Abschnitt 3.4.1) eine Aussage über die temporale Beziehung zweier Anordnungen von Modellelementen, indem im Rahmen des Wareneingangsprozessmodells gefordert wird, dass nach Feststellung einer nicht bestandenen

Sicht-, Begleitpapier- oder Qualitätskontrolle eines Flugzeugteils im weiteren Verlauf eine Sequenz aus zwei Arbeitsschritten folgt, wobei im ersten Schritt der Rückversand der Lieferung im ERP-System verbucht und im nächsten Schritt die Lieferung zurück an den Lieferanten versandt wird.

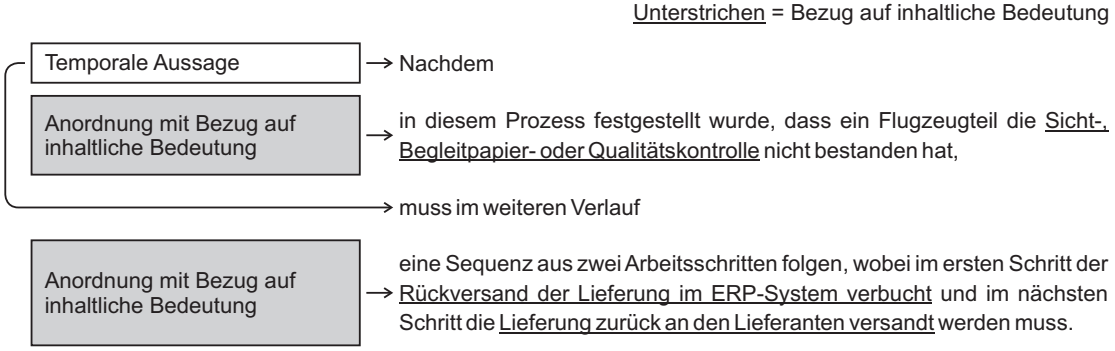


Abbildung 5.2.: Inhaltliche Bestandteile von Anforderung 7 des Szenarios

Im Rahmen der Modellierung kann die Nichtbeachtung derartiger Anforderungen negative Auswirkungen haben, sofern ein aus diesem Grund fehlerhaft modellierter Geschäftsprozess in der Realität entsprechend gelebt oder maschinell ausgeführt wird (siehe Abschnitt 3.5). Probleme können jedoch auch bei bestehenden Geschäftsprozessmodellen auftreten, falls es zu Änderungen auf der betriebswirtschaftlichen Ebene kommt, aus der sich entsprechende Anforderungen ergeben, die jedoch von den betroffenen Modellen nicht erfüllt werden, beispielsweise weil eine zu ändernde Stelle innerhalb eines Modells vom Modellierer übersehen wurde. Es wäre daher wünschenswert, Verletzungen dieser Art auf der einen Seite bereits während der Modellierung feststellen und den Benutzer darauf hinweisen zu können. Auf der anderen Seite sollten bestehende Geschäftsprozessmodelle bei Veränderung des Anforderungsprofils automatisch auf Korrektheit überprüft werden können. In den zwei folgenden Abschnitten wird daher zunächst die Frage erörtert, wie Anforderungen der in dieser Arbeit untersuchten Kategorie definiert werden können. Anschließend wird eine Methode zur Formalisierung derartiger Anforderungen hinsichtlich ihrer maschinellen Verarbeitung vorgestellt.

5.1. Methode zur Modellierung von Anforderungen

Wie im letzten Abschnitt veranschaulicht, handelt es sich bei den in natürlicher Sprache vorliegenden Anforderungen im Rahmen des Szenarios um existenzielle und temporale Aussagen, die sich auf gewisse Anordnungen von Modellelementen beziehen. Eine der Anordnungen, auf die sich beispielsweise *Anforderung 7* (siehe Abbildung 5.2) bezieht („[...] eine Sequenz aus zwei Arbeitsschritten [...]"), beschreibt Modellelemente eines bestimmten Typs („Arbeitsschritte" = Tasks), deren Struktur („Sequenz") und deren inhaltliche Bedeutung (Verbuchung und Versand). Weiterhin fällt auf, dass etwas über

alle derartigen Anordnungen ausgesagt wird, nämlich dass diese Sequenz nach Eintreten einer bestimmten Situation („[...] Sicht-, Begleitpapier- oder Qualitätskontrolle nicht bestanden [...]"), die innerhalb eines BPMN-Modells durch einen bedingten Sequenzfluss ausgedrückt werden kann (siehe Abschnitt 2.2.3), im weiteren Verlauf folgen muss. Auch die sonstigen Anforderungen im Rahmen des Szenarios machen vergleichbare Aussagen über die Existenz gewisser Anordnungen oder Aussagen über die temporale Beziehung zwischen verschiedenen Anordnungen innerhalb bestimmter Geschäftsprozessmodelle.

Existenzielle und temporale Aussagen, die sich auf gewisse Anordnungen von Modellelementen beziehen, gehen über die strukturellen Einschränkungsmöglichkeiten, die im Rahmen eines Metamodells spezifiziert werden können, hinaus. Des Weiteren beziehen sich die untersuchten Aussagen auf Anordnungen von Modellelementen, die eine bestimmte inhaltliche Bedeutung aufweisen. Bei einem Task bezeichnet der Begriff Bedeutung beispielsweise dessen Beschriftung und dessen zugeordnete Semantik (siehe Definition 4.2), bei einem *automatisierten Task* darüber hinaus die URI der von diesem Task aufgerufenen Webdienstmethode. Auch derartige Bezüge inhaltlicher Natur können nicht mithilfe eines Metamodells ausgedrückt werden. Hinsichtlich der automatischen Auswertung von Anforderungen der in dieser Arbeit untersuchten Kategorie müssen folglich sowohl die Anordnungen von Modellelementen, auf die sich existenzielle und temporale Aussagen beziehen, als auch diese Aussagen selbst auf eine andere Art und Weise in maschinenlesbarer Form repräsentiert werden. Zu diesem Zweck wurde im Rahmen dieser Arbeit eine Modellierungsmethode konzipiert, die zwei entsprechende graphische Modellierungssprachen zur Verfügung stellt, die *Process Pattern Modeling Language* (PPML) und die *Process Constraint Modeling Language* (PCML). PPML-Modelle und PCML-Modelle werden auch als *strukturelle Muster* bzw. *Bedingungsausdrücke* bezeichnet. Darüber hinaus wurden zugehörige Modellierungswerkzeuge als Erweiterung der für das Geschäftsprozessmanagement zuständigen Komponente der NetWeaver-Plattform entwickelt: der *Pattern Composer* und der *Constraint Composer*.

5.1.1. Modellierung struktureller Muster

Anordnungen von Modellelementen, auf die sich Anforderungen in Form existenzieller und temporaler Aussagen beziehen, werden zunächst auf Grundlage von PPML mithilfe struktureller Muster beschrieben. Dieser Vorgang ist in Abbildung 5.3 anhand von *Anforderung 7* veranschaulicht. Auf die Details von PPML und der darauf basierenden graphischen Notation wird im weiteren Verlauf noch genauer eingegangen. Ein ebenfalls im späteren Verlauf vorgestelltes Verfahren zur *musterbasierten Suche* (Pattern Matching) ermöglicht es, Geschäftsprozessmodelle nach Anordnungen von Elementen zu durchsuchen, die mit einem strukturellen Muster übereinstimmen, beispielsweise das in Abbildung 5.3 dargestellte Muster, das eine Sequenz aus zwei Arbeitsschritten mit bestimmter zugeordneter Semantik beschreibt. Anordnungen von Modellelementen innerhalb von Geschäftsprozessmodellen, die mit einem strukturellen Muster übereinstimmen, werden als *Instanzen* dieses strukturellen Musters bezeichnet.

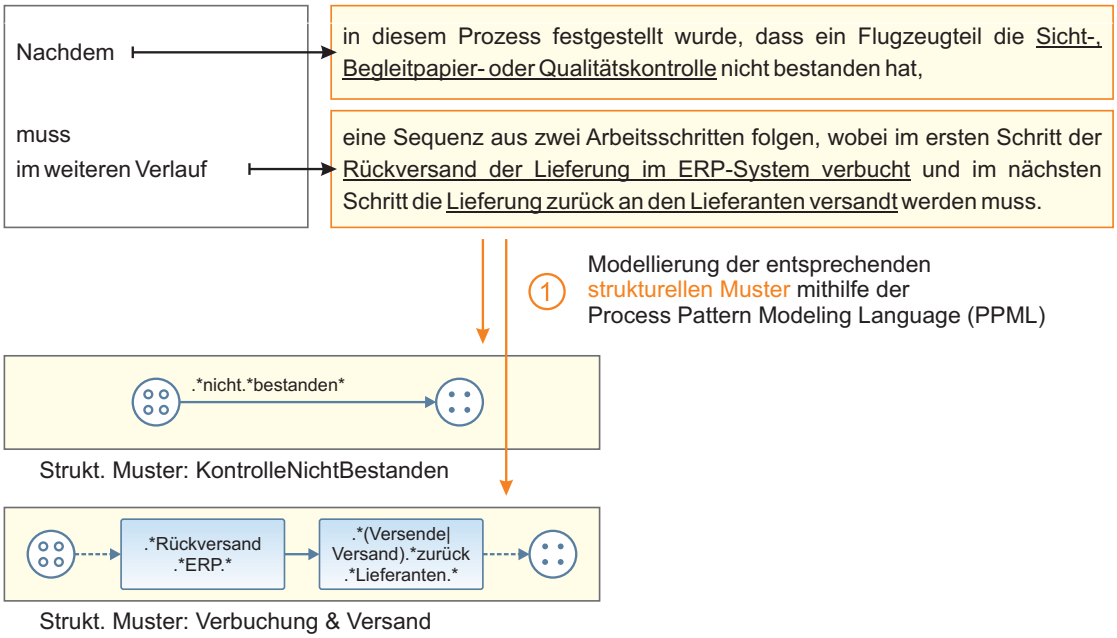


Abbildung 5.3.: Modellierung struktureller Muster

5.1.2. Modellierung von Bedingungsausdrücken

Hinsichtlich struktureller Muster stellt sich die Frage, welche sinnvollen Aussagen über sie gemacht werden können. Zunächst einmal können Aussagen über die Existenz oder Nichtexistenz einer oder mehrerer Instanzen eines strukturellen Musters innerhalb von Geschäftsprozessmodellen gemacht werden, beispielsweise fordert *Anforderung 1* (siehe Abbildung 5.1), dass das Wartungsprozessmodell „[...] genau zwei Sequenzen aus jeweils zwei bis drei Arbeitsschritten enthalten [muss]", während *Anforderung 9* (siehe Abschnitt 3.4.2) fordert, dass es bestimmte Arbeitsschritte nicht enthalten darf. Da beliebig viele Anordnungen von Elementen beschrieben werden können, die nicht innerhalb eines bestimmten Geschäftsprozessmodells enthalten sein dürfen (beispielsweise Arbeitsschritte, die nichts mit dem korrespondierenden Geschäftsprozess zu tun haben), ist die Spezifikation negierter Aussagen normalerweise nicht sinnvoll. Wie im Szenario dargestellt (siehe Abschnitt 3.4.2), können Änderungen auf betriebswirtschaftlicher Ebene jedoch dazu führen, dass Instanzen bestimmter struktureller Muster nicht mehr innerhalb von Geschäftsprozessmodellen enthalten sein dürfen, obwohl sie zuvor erwünscht waren. In diesem Fall könnte eine entsprechende Anforderung die Beseitigung eventuell vorhandener Instanzen unterstützen, indem sie deren Existenz explizit verbietet. Derartige Anforderungen beziehen sich jeweils auf die Existenz von Instanzen einzelner struktureller Muster. Darüber hinaus sind auch Anforderungen denkbar, die eine existenzielle Abhängigkeit zwischen Instanzen verschiedener Muster innerhalb von Geschäftsprozessmodellen fordern, beispielsweise dass die Existenz einer Instanz eines strukturellen Musters die Existenz mindestens einer Instanz eines anderen Musters erforderlich macht. Des

Weiteren können Aussagen über die temporale Beziehung zwischen Instanzen verschiedener Muster gemacht werden, beispielsweise fordert *Anforderung 7* (siehe Abbildung 5.2), dass einer Instanz eines bestimmten Musters im weiteren Verlauf eine Instanz eines weiteren Musters folgen und daher eine entsprechende Folge von Flussobjekten und Sequenzflüssen innerhalb des Geschäftsprozessmodells enthalten sein muss.

Existenzielle und temporale Aussagen, die sich auf Anordnungen von Modellelementen und damit auf Instanzen struktureller Muster beziehen, werden auf Grundlage von PCML in Form sogenannter *musterbasierter Bedingungen* innerhalb von Bedingungsausdrücken modelliert. Dieser Vorgang ist in Abbildung 5.4 anhand von *Anforderung 7* veranschaulicht. Der in dieser Abbildung dargestellte Bedingungsausdruck enthält eine musterbasierte Bedingung, die der von *Anforderung 7* gemachten temporalen Aussage entspricht und die sich auf die zuvor modellierten strukturellen Muster bezieht. Um festzulegen, dass ein Geschäftsprozessmodell eine Anforderung erfüllen muss, wird der entsprechende Bedingungsausdruck mit dem Modell verknüpft (siehe Abbildung 5.4). Im Folgenden wird auf die Details von PPML und PCML und die darauf basierenden graphischen Notationen genauer eingegangen.

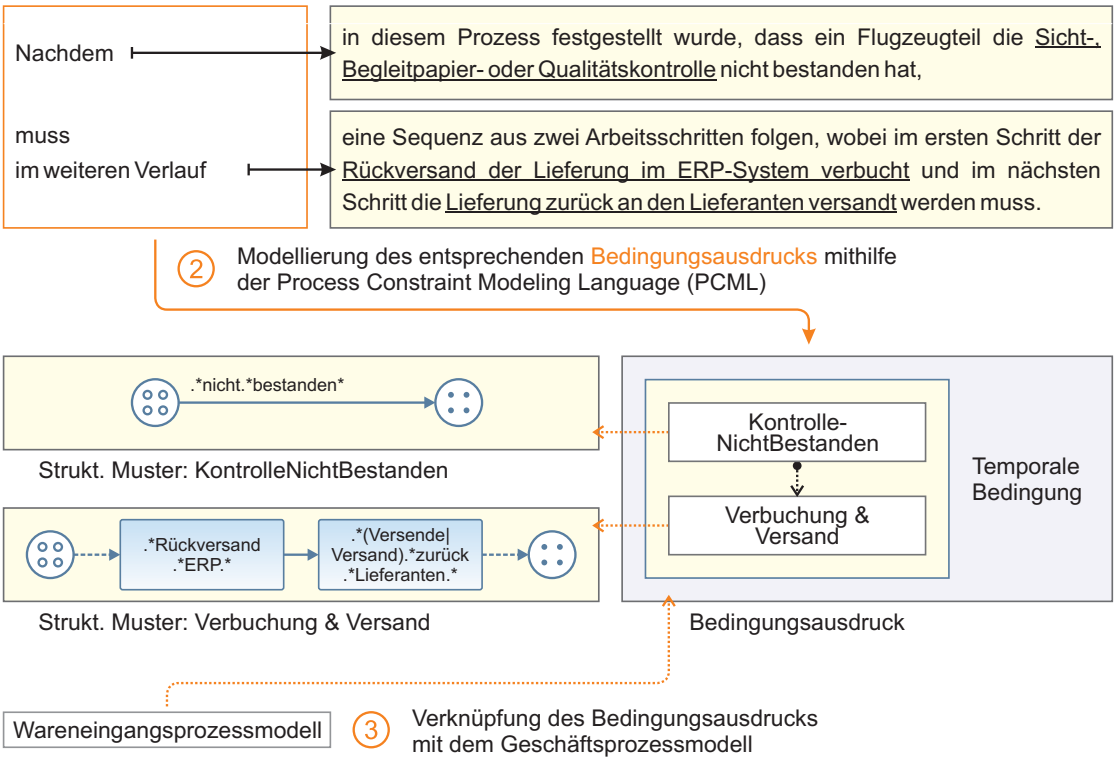


Abbildung 5.4.: Modellierung und Verknüpfung eines Bedingungsausdrucks

5.2. Process Pattern Modeling Language

Die *Process Pattern Modeling Language* ist eine graphische Modellierungssprache zur Spezifikation struktureller Muster, deren Definition auf einem MOF-Metamodell beruht, dem *Process Constraint Definition Metamodel* (PCDM) (siehe Anhang A.4). Über das Metamodell hinaus wurde eine Notation entwickelt, die mithilfe eines entsprechenden Werkzeugs eine graphische Modellierung struktureller Muster ermöglicht. Die Notation von PPML ist teilweise an BPMN angelehnt, wofür es zwei Gründe gibt. Für die generelle Verwendung einer graphischen Notation spricht die Tatsache, dass strukturelle Muster Anordnungen von Elementen innerhalb von Geschäftsprozessmodellen beschreiben und daher ähnliche Wesensmerkmale aufweisen, wobei die graphische Darstellung einer der grundlegenden Wesenszüge der vorliegenden Geschäftsprozessmodelle ist. Für die Verwendung einer an BPMN angelehnten Notation spricht die Tatsache, dass es sich bei BPMN um einen anerkannten Industriestandard handelt, mit dem viele Benutzer vertraut sind, was ihnen das Erlernen einer ähnlichen Notation erleichtert. Die PPML-Modellierungskonstrukte orientieren sich an den in Abschnitt 3.6 auf Grundlage des Szenarios abgeleiteten Anforderungen und sind in Abbildung 5.5 dargestellt, wobei die Konstrukte in Muster- und Verbindungsobjekte unterteilt sind.

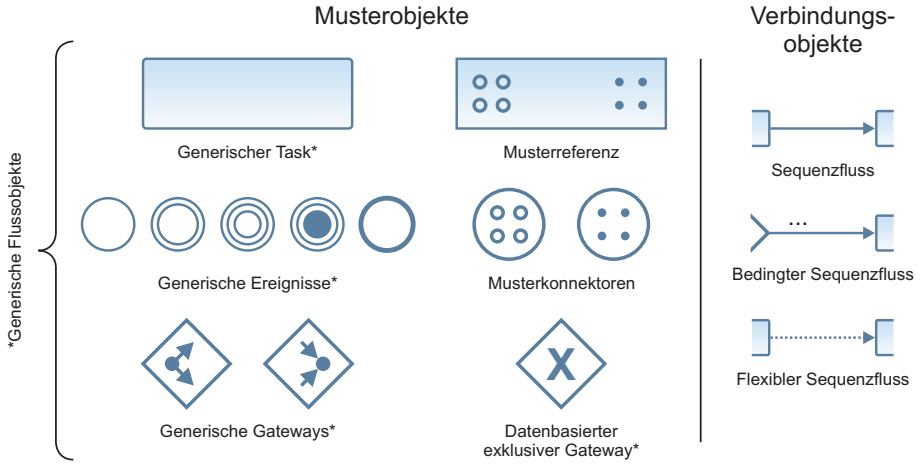


Abbildung 5.5.: PPML-Modellierungskonstrukte

Musterobjekte werden zur Spezifikation gesuchter Elemente innerhalb von BPMN-Modellen verwendet, während Verbindungsobjekte zur Spezifikation ihrer sequentiellen Anordnung dienen. Eine Unterkategorie der Musterobjekte bilden die generischen Flussobjekte, die sich aus generischen Tasks, generischen Ereignissen und mehreren Gateways zusammensetzen. Diese Modellierungskonstrukte werden zur Identifikation von Tasks, Ereignissen bzw. Gateways eingesetzt, wobei zusätzliche Kriterien spezifiziert werden können, die erfüllt sein müssen, damit eine Übereinstimmung vorliegt. Das Modellierungskonstrukt Musterreferenz ermöglicht die modulare Spezifikation struktureller Muster. Mithilfe dieses Konstrukts kann ein strukturelles Muster aus bereits bestehenden PPML-

Modellen bausteinartig zusammengesetzt werden. Die Kategorie der Verbindungsobjekte enthält drei Modellierungskonstrukte: den Sequenzfluss, den bedingten Sequenzfluss und den flexiblen Sequenzfluss, die zur Identifikation bestimmter Anordnungen von Flussobjekten innerhalb von BPMN-Modellen verwendet werden. Flexible Sequenzflüsse ermöglichen es, bei der Spezifikation gesuchter Anordnungen von Flussobjekten einen gewissen Spielraum zuzulassen. Im Folgenden werden die PPML-Modellierungskonstrukte detailliert beschrieben.

5.2.1. Generische Tasks

Jeder Task innerhalb eines BPMN-Modells hat eine bestimmte inhaltliche Bedeutung. Die Bedeutung bezeichnet dabei die Gesamtheit der folgenden Eigenschaften eines Tasks:

- Name (normalerweise in natürlicher Sprache)
- Zugeordnete Semantik (siehe Abschnitt 4.2)
- Aufgerufener Webdienst und dessen Methode (automatisierte Tasks)

Das PPML-Modellierungskonstrukt *Generischer Task* (MOF-Klasse: `GenericTask`) ermöglicht es, Tasks mit einer bestimmten inhaltlichen Bedeutung innerhalb von BPMN-Modellen zu identifizieren (siehe Abbildung 5.6).

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Generischer Task	GenericTask	operations : Collection<Operation> regularExpression : String semanticExpression : SemanticExpression	‹Regulärer Ausdruck›

Abbildung 5.6.: PPML-Modellierungskonstrukt: Generischer Task

Generische Tasks verfügen über drei Attribute, die zur Beschreibung der gesuchten Bedeutung von BPMN-Tasks dienen. Um BPMN-Tasks mit einem bestimmten Namen zu identifizieren, muss dem Attribut `regularExpression` ein entsprechender Wert zugewiesen werden. Bei diesem Wert handelt es sich um eine Zeichenkette, die einen regulären Ausdruck repräsentiert, dessen Syntax in [44] definiert ist. Muss bei der Modellierung generischer Tasks auf den Namen von BPMN-Tasks zurückgegriffen werden, bieten reguläre Ausdrücke einen gewissen Grad an Flexibilität. Die Verwendung regulärer Ausdrücke könnte beispielsweise notwendig sein, weil den Tasks innerhalb von BPMN-Modellen, die auf das Vorhandensein von Instanzen struktureller Muster untersucht werden sollen, keine Semantik zugeordnet ist. Enthält ein derartiges BPMN-Modell Tasks, die denselben Vorgang in der Realität repräsentieren, jedoch unterschiedlich benannt wurden, können diese Tasks bei nicht allzu unterschiedlicher Benennung mithilfe eines regulären Ausdrucks dennoch mittels eines einzelnen generischen Tasks identifiziert werden. Beispielsweise könnten die unterschiedlichen Zeichenketten `Bauteil` und `Teil` durch die regulären Ausdrücke `(Bauteil|Teil)` oder `(Baut|T)eil` identifiziert werden.

Zur Identifikation von BPMN-Tasks mit einer bestimmten zugeordneten Semantik muss dem Attribut `semanticExpression` eines generischen Tasks ein entsprechender *semantischer Ausdruck* zugewiesen werden. Der semantische Ausdruck eines generischen Tasks beschreibt, welche Ontologieklassen einem BPMN-Task zugeordnet sein müssen, damit er durch den generischen Task identifiziert wird. Semantische Ausdrücke können in Form einer kontextfreien Grammatik mithilfe der *Backus-Naur-Form* (BNF) beschrieben werden, die in Listing 5.1 dargestellt ist. Das Nichtterminalsymbol ⟨`uri`⟩ bezieht sich dabei auf die BNF einer URI [BLFM05].

```
<expression> ::= <uri> |
                 "¬" <expression> |
                 "(" <expression> <binary-operator> <expression> ")"

<binary-operator> ::= " ∧ " | " ∨ "
```

Listing 5.1: BNF für semantische Ausdrücke

Semantische Ausdrücke setzen sich also aus URIs und den logischen Operatoren Konjunktion (∧), Disjunktion (∨) und Negation (¬) zusammen. Die Auswertung eines semantischen Ausdrucks eines generischen Tasks bezüglich eines BPMN-Tasks erfolgt auf Basis der zugeordneten Ontologieklassen des BPMN-Tasks. Für jede URI innerhalb des semantischen Ausdrucks wird überprüft, ob die entsprechende Ontologieklasse oder eine ihrer Unterklassen dem BPMN-Task zugeordnet ist. Falls eine derartige Zuordnung besteht, wird die entsprechende URI im semantischen Ausdruck durch `true` ersetzt, andernfalls durch `false`. Diese Art der Auswertung basiert auf der Geschlossenen-Welt-Annahme und steht im Gegensatz zur Offenen-Welt-Annahme von OWL, aufgrund derer ein Reasoner nur darauf schließen kann, dass eine Individuum nicht zu einer bestimmten Klasse gehört, wenn dies explizit deklariert ist. Dennoch wird bei der Auswertung von der Geschlossenen-Welt-Annahme ausgegangen, da sie in diesem Kontext intuitiver erscheint. Nach Abschluss dieses Vorgangs resultiert aus dem semantischen Ausdruck ein aussagenlogischer Ausdruck, der wahr oder falsch ist. Ist dieser Ausdruck wahr, so identifiziert der generische Task den BPMN-Task. Abbildung 5.7 zeigt beispielsweise auf der linken Seite einen BPMN-Task, der den Transport eines Teils in das Sperrlager im Rahmen der Flugzeugwartung repräsentiert. Diesem Task sind zwei Ontologieklassen zugeordnet: `wartung:Transport` und `wartung:Sperrlager`. Der generische Task auf der rechten Seite der Abbildung beschreibt Tasks, die den Transport eines Teils in ein Lager repräsentieren, wobei es sich bei diesem Lager nicht um ein Sperrlager handelt. Die Auswertung des zugehörigen semantischen Ausdrucks auf Grundlage der Ontologieklassen, die dem BPMN-Task zugeordnet sind, führt in diesem Fall zu einer falschen Aussage, wodurch der BPMN-Task nicht durch den generischen Task identifiziert wird.

Um automatisierte Tasks innerhalb eines BPMN-Modells zu identifizieren, die eine bestimmte Webdienstmethode aufrufen, muss dem Attribut `operations` eines generischen Tasks eine Liste aus Webdienstmethoden zugewiesen werden. Die Identifikation eines automatisierten Tasks erfolgt, falls er eine Webdienstmethode aufruft, die in der Liste des

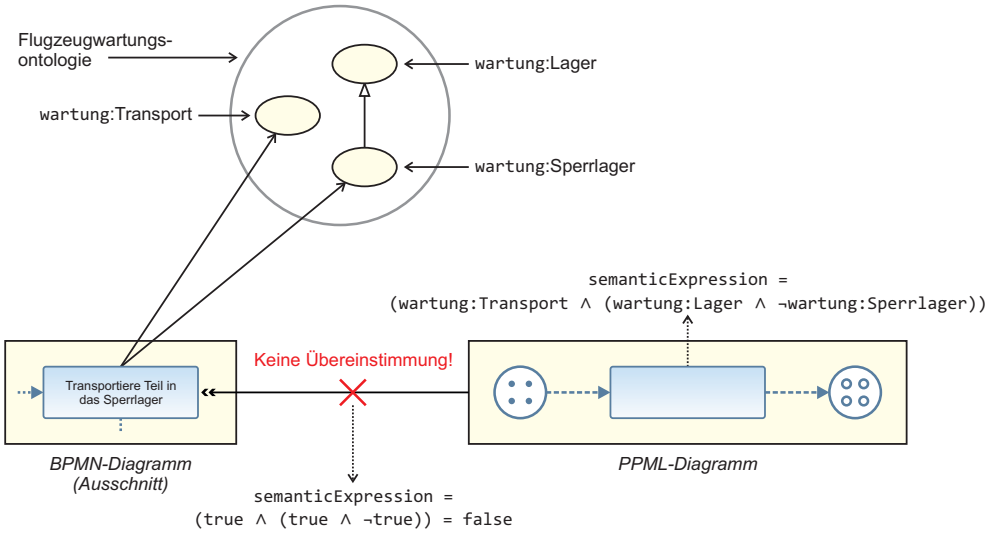


Abbildung 5.7.: Auswertung eines semantischen Ausdrucks eines generischen Tasks

generischen Tasks enthalten ist. Wird mehr als einem der beschriebenen Attribute eines generischen Tasks ein Wert zugewiesen, müssen alle daraus resultierenden Bedingungen von einem BPMN-Task erfüllt werden, damit eine Übereinstimmung vorliegt.

5.2.2. Generische Ereignisse

PPML stellt mehrere Modellierungskonstrukte zur Verfügung, die eine Identifikation von BPMN-Ereignissen ermöglichen (siehe Abbildung 5.8). BPMN verfügt über drei grundlegende Ereignistypen (Start-, Zwischen- und Endereignis) und neun Auslöser, die Ereignisse genauer beschreiben und gemäß der BPMN-Spezifikation mit Ereignissen eines bestimmten Typs kombiniert werden können. Zwischenereignisse können ferner dahingehend unterschieden werden, ob sie ein Ereignis senden oder ein Ereignis empfangen, das an anderer Stelle gesendet wird.

Start-, Zwischen- und Endereignisse werden mithilfe der PPML-Modellierungskonstrukte *Generisches Startereignis* (MOF-Klasse: `GenericStartEvent`), *Generisches Zwischenereignis* (MOF-Klasse: `GenericIntermediateEvent`) bzw. *Generisches Endereignis* (MOF-Klasse: `GenericEndEvent`) identifiziert. Die Modellierungskonstrukte *generisches sendendes Zwischenereignis* (MOF-Klasse: `GenericIntermediateCatchEvent`) und *generisches empfangendes Zwischenereignis* (MOF-Klasse: `GenericIntermediateThrowEvent`) ermöglichen eine Differenzierung von BPMN-Zwischenereignissen. Je nach Konstrukt kann darüber hinaus durch Setzen eines entsprechenden Attributs (z. B. `matchMessageTriggers`) angegeben werden, welcher Auslöser einem BPMN-Ereignis zugeordnet sein muss, damit eine Identifikation erfolgt. Werden mehrere Attribute gesetzt, muss dem BPMN-Ereignis nur einer der zugehörigen Auslöser zugewiesen sein.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
		regularExpression : String semanticExpression : SemanticExpression	
Generisches Startereignis	**GenericStartEvent**	matchConditionalTriggers : boolean matchMessageTriggers : boolean matchSignalTriggers : boolean matchTimerTriggers : boolean	
Generisches Zwischenereignis	**GenericIntermediateEvent**		
Generisches sendendes Zwischenereignis	**Generic-IntermediateCatchEvent**	matchCancelTriggers : boolean matchCompensationTriggers : boolean matchConditionalTriggers : boolean matchErrorTriggers : boolean matchLinkTriggers : boolean matchMessageTriggers : boolean matchSignalTriggers : boolean matchTimerTriggers : boolean	
Generisches empfangendes Zwischenereignis	**Generic-IntermediateThrowEvent**	matchCompensationTriggers : boolean matchLinkTriggers : boolean matchMessageTriggers : boolean matchSignalTriggers : boolean	
Generisches Endereignis	**GenericEndEvent**	matchCancelTriggers : boolean matchCompensationTriggers: boolean matchErrorTriggers : boolean matchMessageTriggers : boolean matchSignalTriggers : boolean matchTerminationTriggers : boolean	

Abbildung 5.8.: PPML-Modellierungskonstrukt: Generische Ereignisse

5.2.3. Generische Gateways

Die PPML-Modellierungskonstrukte *Divergierender Gateway* (MOF-Klasse: `DivergingGateway`) und *Konvergierender Gateway* (MOF-Klasse: `ConvergingGateway`) (siehe Abbildung 5.9) ermöglichen die Identifikation von BPMN-Gateways. Im Gegensatz zur BPMN-Spezifikation differenziert der Process Composer zwischen divergierenden und konvergierenden Gateways. Zur Spezifikation der Bedeutung eines Gateways stellt BPMN fünf verschiedene Typen bereit: datenbasiert exklusiv, ereignisbasiert exklusiv, inklusiv, komplex und parallel. Vergleichbar mit den Modellierungskonstrukten zur Identifikation von BPMN-Ereignissen verfügen die Konstrukte zur Identifikation von Gateways über fünf Attribute, durch deren Setzen festgelegt werden kann, welche BPMN-Gateways von einem generischen Gateway identifiziert werden.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Divergierender Gateway	**DivergingGateway**	matchExactly : boolean matchExclusiveDataBasedGateways : boolean matchExclusiveEventBasedGateways : boolean matchInclusiveGateways : boolean matchComplexGateways : boolean matchParallelGateways : boolean regularExpression : String semanticExpression : SemanticExpression	
Konvergierender Gateway	**ConvergingGateway**		

Abbildung 5.9.: PPML-Modellierungskonstrukt: Generische Gateways

Sofern das Attribut `matchExactly` eines divergierenden oder konvergierenden Gateways gesetzt ist (Gleichheitszeichen innerhalb der graphischen Darstellung), wird ein divergierender bzw. konvergierender BPMN-Gateway innerhalb eines BPMN-Modells nur dann identifiziert, sofern die Anzahl seiner aus- bzw. eingehenden Sequenzflüsse mit der des generischen Gateways übereinstimmt, andernfalls wird ein BPMN-Gateway identifiziert, sofern die Anzahl seiner aus- bzw. eingehenden Sequenzflüsse größer oder gleich der Anzahl der aus- bzw. eingehenden Sequenzflüsse des generischen Gateways ist.

5.2.4. Divergierender exklusiver Gateway

Das PPML-Modellierungskonstrukt *Divergierender exklusiver Gateway* (MOF-Klasse: `DivergingExclusiveGateway`) (siehe Abbildung 5.10) ermöglicht die Identifikation exklusiver BPMN-Gateways und stellt einen Spezialfall des PPML-Modellierungskonstrukts *Divergierender Gateway* dar. Dieses Modellierungskonstrukt dient als Quelle für bedingte Sequenzflüsse, auf die im nächsten Abschnitt näher eingegangen wird.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Divergierender exklusiver Gateway	Diverging-ExclusiveGateway	matchExactly : boolean regularExpression : String semanticExpression : SemanticExpression	

Abbildung 5.10.: PPML-Modellierungskonstrukt: Divergierender exklusiver Gateway

5.2.5. Verbindungsobjekte

5.2.5.1. Sequenzflüsse

Zur Spezifikation des gesuchten kausalen Zusammenhangs bestimmter BPMN-Elemente bietet PPML mehrere Modellierungskonstrukte. Eines dieser Modellierungskonstrukte ist der *Sequenzfluss* (MOF-Klasse: `SequenceConnector`) (siehe Abbildung 5.11). PPML-Sequenzflüsse werden dazu verwendet, die Anordnung gesuchter Modellelemente innerhalb von BPMN-Modellen zu beschreiben und letztendlich BPMN-Sequenzflüsse zu identifizieren. Zu diesem Zweck können Musterobjekte mithilfe von PPML-Sequenzflüssen miteinander verbunden werden. Ein PPML-Sequenzfluss, der zwei Musterobjekte verbindet, identifiziert einen BPMN-Sequenzfluss, der zwei Flussobjekte verbindet, sofern diese Flussobjekte von den korrespondierenden Musterobjekten identifiziert werden.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Sequenzfluss	SequenceConnector	source : FlowObject target : FlowObject	

Abbildung 5.11.: PPML-Modellierungskonstrukt: Sequenzfluss

Die Anzahl erlaubter ein- und ausgehender Sequenzflüsse hängt vom jeweiligen Musterobjekt ab. Generische Tasks und generische Zwischenereignisse müssen genau mit einem ein- und einem ausgehenden Sequenzfluss verbunden sein. Startereignisse und eingehende Musterkonnektoren dürfen über keinen eingehenden Sequenzfluss verfügen. Im Gegensatz dazu dürfen Endereignisse oder ausgehende Musterkonnektoren über keinen ausgehenden Sequenzfluss verfügen. Divergierende und konvergierende Gateways dürfen mit mehreren aus- bzw. eingehenden Sequenzflüssen verbunden sein. Eine Übersicht dieser Regeln ist in Abbildung 5.12 aufgelistet.

Modellierungskonstrukt	Erlaubte Anzahl eingehender Sequenzflüsse		Erlaubte Anzahl ausgehender Sequenzflüsse	
	Sequenzfluss	Flexibler Sequenzfluss	Sequenzfluss	Flexibler Sequenzfluss
Generischer Task	1..1	1..1	1..1	1..1
Generisches Start-Ereignis	0	0	1..1	1..1
Generisches Zwischen-Ereignis	1..1	1..1	1..1	1..1
Generisches End-Ereignis	1..1	1..1	0	0
Divergierender Gateway	1..1	1..1	1..*	1..*
Konvergierender Gateway	1..*	1..*	1..1	1..1
Divergierender exkl. Gateway	1..1	1..1	0	1..*
Eingehender Musterkonnektor	0	0	1..1	0
Ausgehender Musterkonnektor	1..1	0	0	0
Musterreferenz	1..1	1..1	1..1	1..1
	0	0	1..1	1..1
	1..1	1..1	0	0
		XOR		XOR

Abbildung 5.12.: PPML-Modellierungskonstrukt: Sequenzfluss (Verbindungsmatrix)

5.2.5.2. Bedingte Sequenzflüsse

Das PPML-Modellierungskonstrukt *Bedingter Sequenzfluss* (MOF-Klasse: `ConditionalSequenceConnector`) dient zur Identifikation bedingter Sequenzflüsse innerhalb von BPMN-Modellen (siehe Abbildung 5.13). Zur Spezifikation der gesuchten inhaltlichen Bedeutung verfügt dieses Modellierungskonstrukt wie generische Flussobjekte über Attribute, welche die Zuweisung eines regulären und eines semantischen Ausdrucks erlauben. Bei der Quelle eines bedingten Sequenzflusses muss es sich um einen exklusiven Gateway oder einen eingehenden Musterkonnektor handeln, als Ziel sind alle Musterobjekte außer Musterreferenzen zulässig. Wie im weiteren Verlauf noch gezeigt wird, eignen sich bedingte Sequenzflüsse dazu, Bedingungen zu modellieren, die sich auf bestimmte Pfade beziehen, die von einem exklusiven Gateway ausgehen.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation

Bedingter Sequenzfluss	**Conditional-SequenceConnector**	`source : FlowObject`
		`target : FlowObject`
		`semanticExpression : SemanticExpression`
		`regularExpression : String`	

Abbildung 5.13.: PPML-Modellierungskonstrukt: Bedingter Sequenzfluss

5.2.5.3. Flexible Sequenzflüsse

Über Sequenzflüsse und bedingte Sequenzflüsse hinaus können Musterobjekte auch durch *flexible Sequenzflüsse* (MOF-Klasse: `FlexibleSequenceConnector`) (siehe Abbildung 5.14) miteinander verbunden werden, falls eine gewisse Flexibilität bei der Beschreibung der Anordnung gesuchter Modellelemente gewährleistet sein soll. Allerdings unterscheidet sich die Semantik flexibler Sequenzflüsse von der Semantik der anderen Verbindungsobjekte erheblich. Während die durch einen (bedingten) Sequenzfluss miteinander verbundenen Musterobjekte mit Flussobjekten innerhalb eines BPMN-Modells korrespondieren, die ebenfalls direkt miteinander verbunden sind, kann bei der Verwendung eines flexiblen Sequenzflusses angegeben werden, wie viele Tasks und Ereignisse zwischen zwei Flussobjekten (F_1 und F_2), die mit den durch den flexiblen Sequenzfluss miteinander verbundenen Musterobjekten korrespondieren, liegen dürfen. Zu diesem Zweck können durch Setzen der Attribute `minimumNumberOfActivities` und `minimumNumberOfEvents` entsprechende Untergrenzen, durch Setzen der Attribute `maximumNumberOfActivities` und `maximumNumberOfEvents` entsprechende Obergrenzen gesetzt werden, die jeweils zwischen null und unendlich liegen dürfen. Die Anzahl erlaubter ein- und ausgehender flexibler Sequenzflüsse hängt wie bei regulären Sequenzflüssen vom Typ des jeweiligen Musterobjekts ab (siehe Abbildung 5.12).

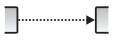
Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Flexibler Sequenzfluss	FlexibleSequenceConnector	maximumNumberOfActivities : int maximumNumberOfEvents : int minimumNumberOfActivities : int minimumNumberOfEvents : int source : FlowObject target : FlowObject	

Abbildung 5.14.: PPML-Modellierungskonstrukt: Flexibler Sequenzfluss

Flexible Sequenzflüsse können dazu verwendet werden, Modellelemente zu beschreiben, deren genaue Anordnung unbekannt ist. Auch unterschiedliche Anordnungen von Modellelementen, die jedoch gemeinsame Charakteristiken aufweisen, können mithilfe flexibler Sequenzflüsse durch ein einzelnes strukturelles Muster beschrieben werden. In Abbildung 5.15 ist ein Beispiel dargestellt, das die Semantik eines flexiblen Sequenzflusses verdeutlicht. In diesem Beispiel wird mithilfe des strukturellen Musters M eine bestimmte Anordnung zweier Tasks beschrieben, wobei ersterer den Namen A und letzterer den Namen D trägt. Weiterhin sagt die Beschreibung aus, dass diese Tasks entweder direkt miteinander verbunden sind oder maximal ein weiterer Task dazwischenliegt. Diese Beschreibung wird in M durch einen flexiblen Sequenzfluss ausgedrückt, dessen Attribute entsprechend gesetzt sind. Im unteren Teil von Abbildung 5.15 sind drei Ausschnitte eines BPMN-Modells (A_1, A_2 und A_3) abgebildet, die aufeinanderfolgende BPMN-Tasks zeigen. Die Suche nach Instanzen des Musters M liefert in diesem Fall ein Resultat innerhalb von A_1 und A_2. Die mit A und D beschrifteten BPMN-Tasks innerhalb von A_3 stimmen dagegen nicht mit der durch M beschriebenen Anordnung von Modellelementen überein, da zwischen A und D zwei weitere BPMN-Tasks liegen.

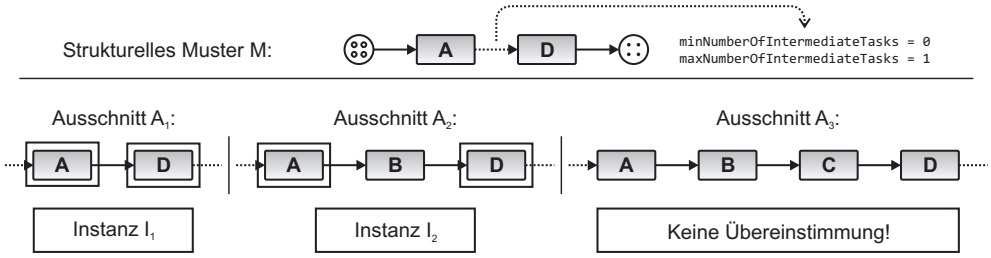


Abbildung 5.15.: PPML-Modellierungskonstrukt: Flexibler Sequenzfluss (Beispiel)

Die Semantik eines flexiblen Sequenzflusses ist vom Wert der Ober- und Untergrenzen abhängig. Dürfen zwischen F_1 und F_2 beliebig viele Tasks und Ereignisse liegen, ist die Auswertung des flexiblen Sequenzflusses erfolgreich, sofern innerhalb des zu durchsuchenden BPMN-Modells ein Pfad, d. h. eine Folge von Flussobjekten und Sequenzflüssen, in Kontrollflussrichtung zwischen F_1 und F_2 existiert. Dabei werden Zyklen innerhalb des Modells berücksichtigt. In diesem Fall reicht ein relativ einfacher Algorithmus zur Auswertung aus, da keine Zählung durchgeführt werden muss.

Ist nur eine bestimmte Anzahl dazwischenliegender Tasks oder Ereignisse zulässig, muss bei der Auswertung des flexiblen Sequenzflusses im Gegensatz dazu eine Zählung durchgeführt werden. Aufgrund der Komplexität einer entsprechenden Berechnung kann in diesem Fall hinsichtlich der Semantik zwischen einem struktur- und einem laufzeitbasierten Verfahren gewählt werden (siehe Abbildung 5.16), die unterschiedliche Vor- und Nachteile aufweisen und auf die im nächsten Kapitel näher eingegangen wird. Bei Verwendung des strukturbasierten Verfahrens ist die Auswertung erfolgreich, sofern innerhalb des zu durchsuchenden BPMN-Modells ein Pfad zwischen F_1 und F_2 existiert, der die Kriterien des flexiblen Sequenzflusses erfüllt. Dabei bricht die Auswertung jedoch bei Erkennung eines Zyklus ab (ein BPMN-Modell, das einen Zyklus enthält, ist in Abbildung 7.2 dargestellt), was in der Praxis meist ausreichen dürfte. Zu diesem Zweck reicht ebenfalls ein relativ einfacher Algorithmus aus. Bei Verwendung des laufzeitbasierten Verfahrens wird dagegen nicht die vom Benutzer vorgegebene Struktur des BPMN-Modells, sondern dessen Laufzeitverhalten untersucht. Zwar werden bei diesem Verfahren Zyklen berücksichtigt, das Ergebnis dieses Verfahrens kann jedoch vom strukturbasierten Verfahren abweichen. Der Nachteil dieses Verfahrens ist, dass zu dessen Durchführung eine aufwendige *Modellprüfung* (Model Checking) erforderlich ist.

Aufgrund der Verwendung flexibler Sequenzflüsse innerhalb eines strukturellen Musters handelt es sich bei den Instanzen dieses Musters um sogenannte *fragmentierte Instanzen*. Da flexible Sequenzflüsse die Anordnung von zwei gegebenen Flussobjekten überprüfen, sind diese bei erfolgreicher Auswertung zwar Teil einer gefundenen Instanz des zugehörigen Musters, nicht jedoch die Modellelemente, die sich zwischen den beiden Flussobjekten befinden. Daher besteht eine solche Instanz aus mehreren Fragmenten des zugrunde liegenden BPMN-Modells. Die in Abbildung 5.15 dargestellte Instanz I_2 enthält beispielsweise zwei isolierte Tasks.

Auswertungsgrundlage	Vorgehensweise	Zählung	Zyklen	Übereinstimmungskriterium
Struktur	Spezieller Algorithmus (1)	nein	ja	Das Modell enthält einen Pfad zwischen F_1 und F_2.
Struktur	Spezieller Algorithmus (2)	ja	nein	Das Modell enthält einen Pfad zwischen F_1 und F_2, der die Kriterien des flexiblen Sequenzflusses erfüllt.
Laufzeitzeitverhalten	Aufwendige Modellprüfung	ja	ja	Die Ausführung des Modells kann dazu führen, dass der Ausführung von F_1 eine Ausführung von F_2 folgt, wobei die Folge der zwischen F_1 und F_2 ausgeführten Tasks und Ereignisse die Kriterien des flexiblen Sequenzflusses erfüllt.

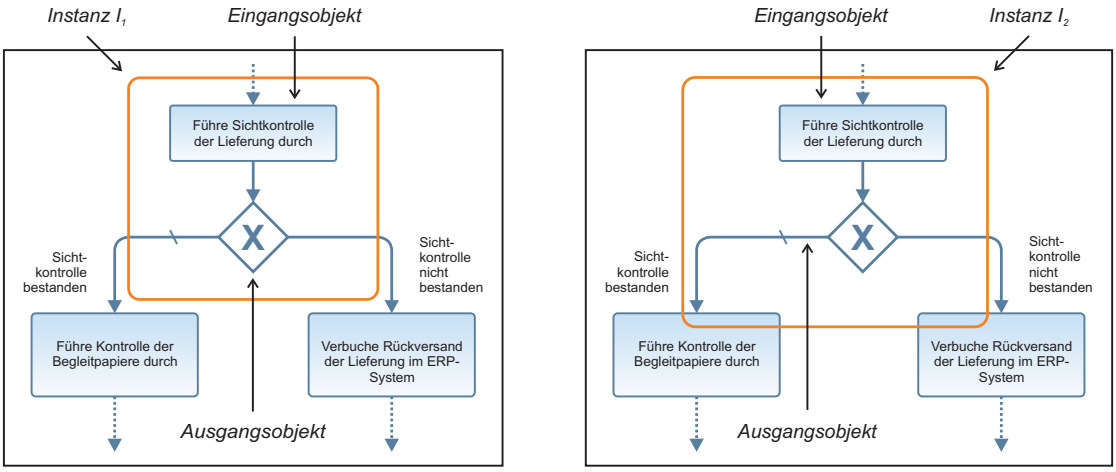
Abbildung 5.16.: PPML-Modellierungskonstrukt: Flexibler Sequenzfluss (Semantik)

5.2.6. Ein- und ausgehende Musterkonnektoren

Strukturelle Muster dienen zur Beschreibung bestimmter Anordnungen von Elementen innerhalb von BPMN-Modellen. Da es sich bei Instanzen struktureller Muster, d. h. Anordnungen von Modellelementen, die mit einem Muster übereinstimmen, um Fragmente von BPMN-Modellen handelt, können sie über mehrere *Ein- und Ausgangsobjekte* verfügen. Als Ein- und Ausgangsobjekte einer Instanz eines strukturellen Musters werden Flussobjekte (Tasks, Ereignisse und Gateways) bezeichnet, die mit keinem vorhergehenden bzw. nachfolgenden Flussobjekt in Beziehung stehen, das ebenfalls Teil der Instanz ist. Wie im Folgenden gezeigt wird, können unter bestimmten Umständen auch bedingte Sequenzflüsse Ein- oder Ausgangsobjekte darstellen. In Abbildung 5.17(a) und Abbildung 5.17(b) ist jeweils ein Ausschnitt eines BPMN-Modells dargestellt und eine Instanz eines strukturellen Musters innerhalb des jeweiligen Modells hervorgehoben. Die in Abbildung 5.17(a) dargestellte Instanz I_1 besteht aus einem Task, dem ein exklusiver Gateway folgt. Die in Abbildung 5.17(b) dargestellte Instanz I_2 enthält im Vergleich zu I_1 zusätzlich einen der bedingten Sequenzflüsse, die dem exklusiven Gateway folgen. In beiden Fällen enthält die Instanz jeweils ein Eingangs- und ein Ausgangsobjekt.

Strukturelle Muster enthalten Modellelemente, die mit den Ein- und Ausgangsobjekten von Instanzen struktureller Muster korrespondieren. Bei der Modellierung struktureller Muster müssen diese Modellelemente explizit abgegrenzt werden, sofern es sich nicht um generische Start- oder Endereignisse handelt. Zur Abgrenzung stehen die PPML-Modellierungskonstrukte *Eingehender Musterkonnektor* (MOF-Klasse: `IncomingPatternConnector`) und *Ausgehender Musterkonnektor* (MOF-Klasse: `OutgoingPatternConnector`) zur Verfügung (siehe Abbildung 5.18).

Musterkonnektoren werden mithilfe von Sequenzflüssen oder bedingten Sequenzflüssen mit anderen Musterobjekten verknüpft. Diese Verbindungsobjekte werden normalerweise nicht in die Suche nach Instanzen struktureller Muster einbezogen, beispielsweise *Sequenzfluss 1* und *Sequenzfluss 2* in Abbildung 5.19, die jeweils mit einem eingehenden Musterkonnektor und einem generischen Task verbunden sind. Da derartige Sequenzflüsse nicht in das Ergebnis der Suche einfließen, werden sie gestrichelt dargestellt.



(a) BPMN-Modell (Ausschnitt) (b) BPMN-Modell (Ausschnitt)

Abbildung 5.17.: Eingangs- und Ausgangsobjekte

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Eingehender Musterkonnektor	**IncomingPatternConnector**	`name : String`	
Ausgehender Musterkonnektor	**OutgoingPatternConnector**		

Abbildung 5.18.: PPML-Modellierungskonstrukte: Musterkonnektoren

Eine Ausnahme dieser Regel besteht, falls ein Sequenzfluss einen generischen Gateway mit einem Musterkonnektor oder ein bedingter Sequenzfluss einen exklusiven Gateway oder eingehenden Musterkonnektor mit einem ausgehenden Musterkonnektor verbindet. Sequenzflüsse, die einen generischen Gateway mit einem Musterkonnektor verbinden, werden zwar in die Suche nach Instanzen struktureller Muster einbezogen, fließen jedoch nicht in deren Ergebnis ein. Dieses Verhalten zielt darauf ab, Gateways mit einer bestimmten Anzahl oder Mindestanzahl ein- oder ausgehender Sequenzflüsse zu identifizieren, beispielsweise korrespondiert das in Abbildung 5.19 dargestellte Muster M_1 mit der in Abbildung 5.17(a) hervorgehobenen Instanz I_1, da aufgrund des generischen Gateways in Abbildung 5.19, dessen Attribut `matchExactly` gesetzt ist und der über zwei ausgehende Sequenzflüsse verfügt, alle divergierenden Gateways innerhalb eines BPMN-Modells identifiziert werden, die über genau zwei ausgehende Sequenzflüsse verfügen. Bedingte Sequenzflüsse, die einen exklusiven Gateway mit einem ausgehenden Musterkonnektor verbinden werden ebenfalls in die Suche nach Instanzen struktureller Muster einbezogen, fließen allerdings auch in deren Ergebnis ein. Da bedingte Sequenzflüsse in das Ergebnis der Suche einfließen, korrespondiert das in Abbildung 5.19 dargestellte Muster M_2 in diesem Fall mit der in Abbildung 5.17(b) hervorgehobenen Instanz I_2.

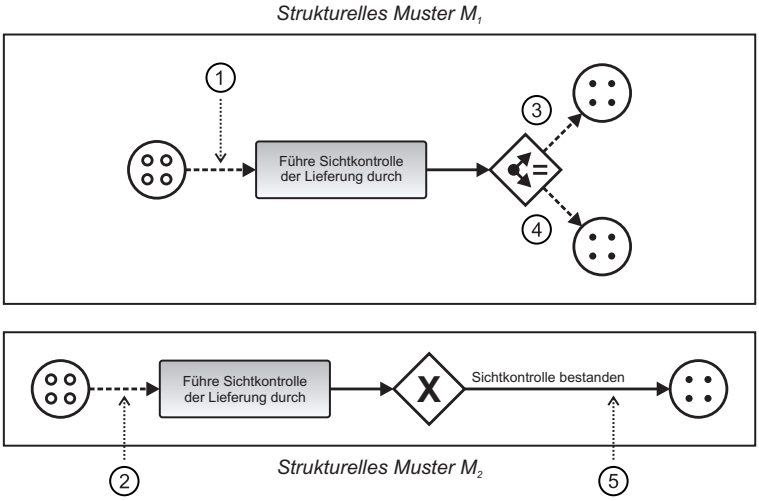


Abbildung 5.19.: Strukturelle Muster mit ein- und ausgehenden Musterkonnektoren

5.2.7. Musterreferenz

Das PPML-Modellierungskonstrukt *Musterreferenz* (MOF-Klasse: `PatternReference`) (siehe Abbildung 5.20) wird bei der Modellierung eines strukturellen Musters zur Integration bestehender PPML-Modelle in dieses Muster verwendet. Musterreferenzen ermöglichen damit eine Modularisierung struktureller Muster, die zur besseren Verständlichkeit und Verringerung der Redundanz beiträgt. Vor der Suche nach Instanzen struktureller Muster müssen alle Musterreferenzen innerhalb eines Musters aufgelöst werden, d. h. durch die entsprechenden Muster ersetzt werden. Musterreferenzen ähneln in Bezug auf den Modularitätsaspekt dem BPMN-Modellierungskonstrukt *Unterprozess*.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation

Musterreferenz	`PatternReference`	`ipc : IncomingPatternConnector`
		`opc : OutgoingPatternConnector`
		`structuralPattern : StructuralPattern`
			

Abbildung 5.20.: PPML-Modellierungskonstrukt: Musterreferenz

Bei Verwendung einer Musterreferenz muss dem Attribut `structuralPattern` das strukturelle Muster zugewiesen werden, das an dieser Stelle integriert werden soll. Musterreferenzen können je nach gewünschter Lage mit einem eingehenden, einem ausgehenden oder einem ein- und einem ausgehenden (flexiblen) Sequenzfluss verbunden werden. Da ein Muster über mehrere ein- und ausgehende Musterkonnektoren verfügen kann, muss darüber hinaus angegeben werden, auf welche Musterkonnektoren sich die Musterreferenz bezieht. Ist eine Musterreferenz mit einem eingehenden Sequenzfluss verbunden, muss dem Attribut `ipc` der Musterreferenz ein eingehender Musterkonnektor des referenzierten Musters zugewiesen werden. Falls die Musterreferenz mit einem ausgehenden Sequenzfluss verbunden ist, muss dem Attribut `opc` analog dazu ein ausgehender

Musterkonnektor zugewiesen werden. In Abbildung 5.21 wird die Funktionsweise einer Musterreferenz erklärt. Im oberen Teil der Abbildung sind die strukturellen Muster M_1 und M_2 dargestellt, wobei M_2 eine Musterreferenz enthält, die auf M_1 verweist. Da die Musterreferenz sowohl mit einem eingehenden als auch einem ausgehenden Sequenzfluss verbunden ist, verweist sie zusätzlich auf einen eingehenden (E_2) und auf einen ausgehenden Musterkonnektor (A_1) von M_1. Aufgrund dieser Zuordnungen kann die Musterreferenz eindeutig aufgelöst werden. Das im unteren Teil der Abbildung dargestellte struktureller Muster M_2' entsteht durch Auflösung der Musterreferenz in M_2.

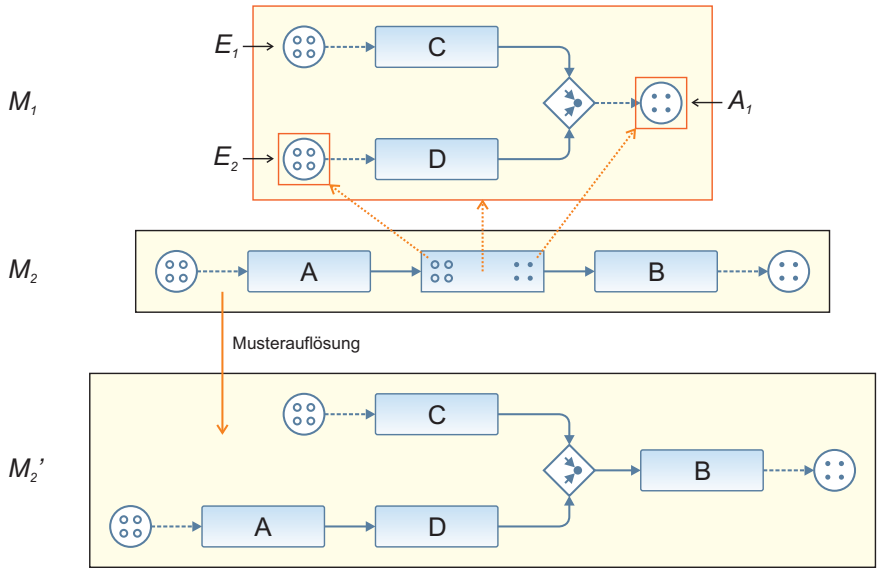


Abbildung 5.21.: PPML-Modellierungskonstrukt: Musterreferenz (Beispiel)

5.3. Process Constraint Modelling Language

Analog zu PPML ist die Process Constraint Modeling Language eine graphische Modellierungssprache zur Spezifikation von Bedingungsausdrücken, deren Definition ebenfalls auf dem Process Constraint Definition Metamodel beruht. Auch in diesem Fall wurde eine einfache Notation entwickelt, die mithilfe eines entsprechenden Werkzeugs die graphische Modellierung von Bedingungsausdrücken ermöglicht. Bedingungsausdrücke bestehen aus musterbasierten Bedingungen und logischen Operatoren, mit deren Hilfe musterbasierte Bedingungen verknüpft werden können. Auf diese Weise kann beispielweise ausgedrückt werden, dass ein BPMN-Modell mindestens eine von zwei Bedingungen erfüllen muss. Ein Spezialfall ist ein Bedingungsausdruck, der lediglich aus einer einzelnen musterbasierten Bedingung besteht. Bedingungsausdrücke können einem oder mehreren BPMN-Modellen zugeordnet werden.

Die PCML-Modellierungskonstrukte sind in Abbildung 5.22 dargestellt und sind in zwei Kategorien unterteilt: Bedingungsausdrucks- und Verbindungsobjekte. Die zentralen Modellierungskonstrukte stellen *existenzielle* und *temporale Bedingungen* dar, die sich jeweils auf ein oder zwei strukturelle Muster beziehen und daher auch in *unäre* und *binäre* Bedingungen eingeteilt werden. Bedingungen können wiederum mithilfe logischer Operatoren (Konjunktion, Disjunktion) miteinander verknüpft werden. Die verschiedenen Bedingungstypen weisen eine ähnliche graphische Notation auf, sind aber anhand eines eindeutigen Symbols voneinander unterscheidbar. Diese Bedingungstypen basieren größtenteils auf den in [DAC98] vorgeschlagenen Entwurfsmustern zur Spezifikation von Eigenschaften zur Überprüfung durch endliche Automaten. Weitere Bedingungstypen und die verwendete Notation wurden von dem in [AP06b, AP06c] vorgestellten Verfahren zur deklarativen Modellierung automatisierter Geschäftsprozessmodelle inspiriert.

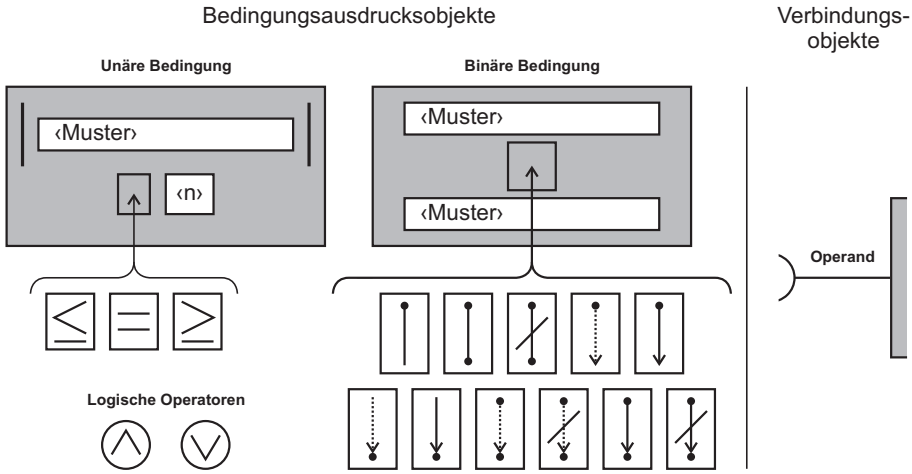


Abbildung 5.22.: PCML-Modellierungskonstrukte

Ein graphischer Bedingungsausdruck kann auch in Form einer kontextfreien Grammatik beschrieben werden, die in Listing 5.2 dargestellt ist. Zur Auswertung eines Bedingungsausdrucks, der einem BPMN-Modell zugeordnet ist, wird zunächst jede Bedingung innerhalb des Bedingungsausdrucks ausgewertet. Falls eine Bedingung erfüllt ist, wird sie im textuellen Bedingungsausdruck durch `true` ersetzt, andernfalls durch `false`. Nach Abschluss dieses Vorgangs resultiert aus dem textuellen Bedingungsausdruck ein aussagenlogischer Ausdruck, der wahr oder falsch ist. Ist dieser Ausdruck wahr, so erfüllt das BPMN-Modell den Bedingungsausdruck, andernfalls wird der Ausdruck verletzt. Im Folgenden werden die PCML-Modellierungskonstrukte detailliert beschrieben.

5.3.1. Existenzielle Bedingungen

Mithilfe existenzieller Bedingungen können Anforderungen modelliert werden, die Aussagen über das Vorhandensein von Instanzen struktureller Muster innerhalb von BPMN-Modellen machen. Abbildung 5.23 veranschaulicht, wie *Anforderung 1* des Szenarios

```
<condition>  ::= <unary-condition-name>  "(" <pattern-name> ")" ||
                 <binary-condition-name> "(" <pattern-name> ", "
                                             <pattern-name> ")"
<expression> ::= <condition> |
                 "(" <expression> <binary-operator> <expression> ")"

<binary-operator> ::= " ∧ " | " ∨ "
```

Listing 5.2: BNF für Bedingungsausdrücke

(siehe Abschnitt 3.4.1), die eine existenzielle Aussage über eine gewisse Anordnung von Modellelementen macht, in Form einer existenziellen Bedingung innerhalb eines Bedingungsausdrucks modelliert werden kann. Die existenzielle Bedingung bezieht sich dabei auf ein zuvor modelliertes strukturelles Muster, das die Anordnung beschreibt, auf die sich die existenzielle Aussage bezieht.

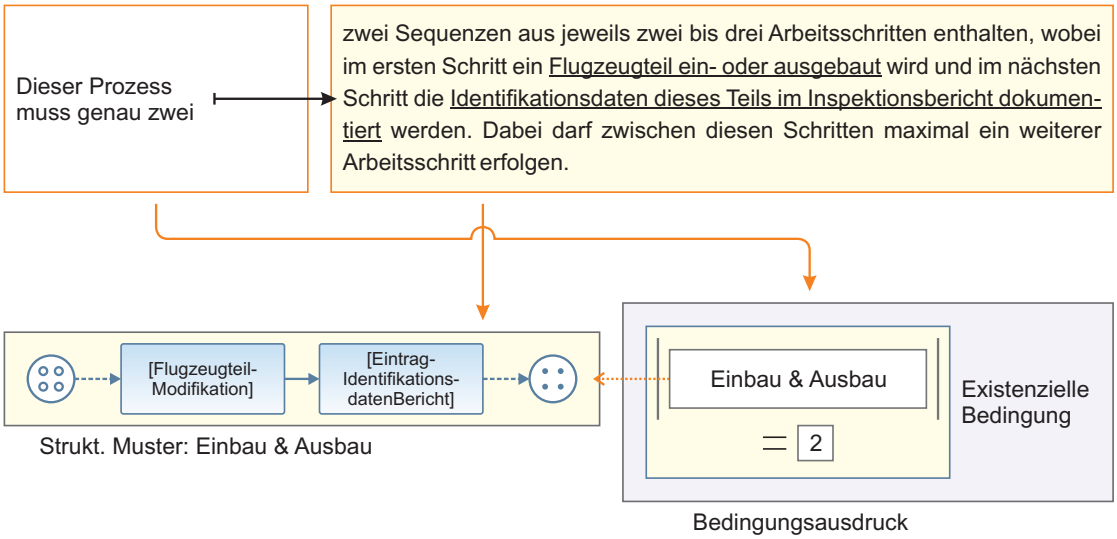


Abbildung 5.23.: Modellierung einer existenziellen Bedingung

PCML ermöglicht die Modellierung unärer und binärer existenzieller Bedingungen (siehe Abbildung 5.24). Zur Modellierung unärer existenzieller Bedingungen stehen drei PCML-Modellierungskonstrukte zur Verfügung: *Abwesenheit* (MOF-Klasse: `AbsenceCondition`), *Exaktheit* (MOF-Klasse: `ExactnessCondition`) und *Existenz* (MOF-Klasse: `ExistenceCondition`). Mit einer Abwesenheits-Bedingung kann ausgedrückt werden, dass höchstens eine bestimmte Anzahl von Instanzen des angegebenen strukturellen Musters innerhalb eines BPMN-Modells enthalten sein darf. Mit einer Exaktheits-Bedingung und einer Existenz-Bedingung kann im Gegensatz dazu festgelegt werden, dass eine genaue Anzahl bzw. mindestens eine bestimmte Anzahl von Instanzen enthalten sein muss.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Abwesenheit	AbsenceCondition	amount : int structuralPattern : StructuralPattern	
Exaktheit	ExactnessCondition		
Existenz	ExistenceCondition		
Erwiderte Existenz	Responded-ExistenceCondition	matchPairwise : bool structuralPattern1 : StructuralPattern structuralPattern2 : StructuralPattern	
Koexistenz	CoexistenceCondition		
Negierte Koexistenz	Negated-CoexistenceCondition	structuralPattern1 : StructuralPattern structuralPattern2 : StructuralPattern	

Abbildung 5.24.: PCML-Modellierungskonstrukt: Existenzielle Bedingungen

Binäre existenzielle Bedingungen können mithilfe folgender PCML-Modellierungskonstrukte modelliert werden: *Erwiderte Existenz* (MOF-Klasse: `RespondedExistenceCondition`), *Koexistenz* (MOF-Klasse: `CoexistenceCondition`) und *Negierte Koexistenz* (MOF-Klasse: `NegatedCoexistenceCondition`). Mithilfe einer Bedingung vom Typ Erwiderte Existenz kann ausgedrückt werden, dass bei Vorhandensein einer Instanz von M_1 innerhalb eines BPMN-Modells auch mindestens eine Instanz von M_2 enthalten sein muss. Eine gegenseitige Beziehung existenzieller Natur zwischen strukturellen Mustern kann mit einer Koexistenz-Bedingung festgelegt werden. Eine Koexistenz-Bedingung drückt aus, dass bei Vorhandensein einer Instanz von M_1 oder M_2 auch mindestens eine Instanz von M_2 bzw. M_1 enthalten sein muss. Im Gegensatz dazu besagt eine negierte Koexistenz-Bedingung, dass bei Vorhandensein einer Instanz von M_1 oder M_2 keine Instanz von M_2 bzw. M_1 enthalten sein darf. Im Gegensatz zu den anderen existenziellen Bedingungen verfügen Erwiderte-Existenz- und Koexistenz-Bedingungen über ein boolesches Attribut namens `matchPairwise`. Wenn dieses Attribut den Wert `true` aufweist, muss zur Erfüllung dieser Bedingungen zusätzlich die Anzahl gefundener Instanzen von M_1 mit der Anzahl gefundener Instanzen von M_2 übereinstimmen.

5.3.2. Temporale Bedingungen

Mithilfe temporaler Bedingungen können Anforderungen modelliert werden, die Aussagen über die temporale Beziehung zwischen Instanzen verschiedener struktureller Muster innerhalb von BPMN-Modellen machen. Abbildung 5.25 veranschaulicht erneut, wie *Anforderung 7* des Szenarios (siehe Abschnitt 3.4.1), die eine temporale Aussage über gewisse Anordnungen von Modellelementen macht, in Form einer temporalen Bedingung innerhalb eines Bedingungsausdrucks modelliert werden kann. Die temporale Bedingung

bezieht sich dabei auf zwei zuvor modellierte strukturelle Muster, welche die Anordnungen beschreiben, auf die sich die temporale Aussage bezieht.

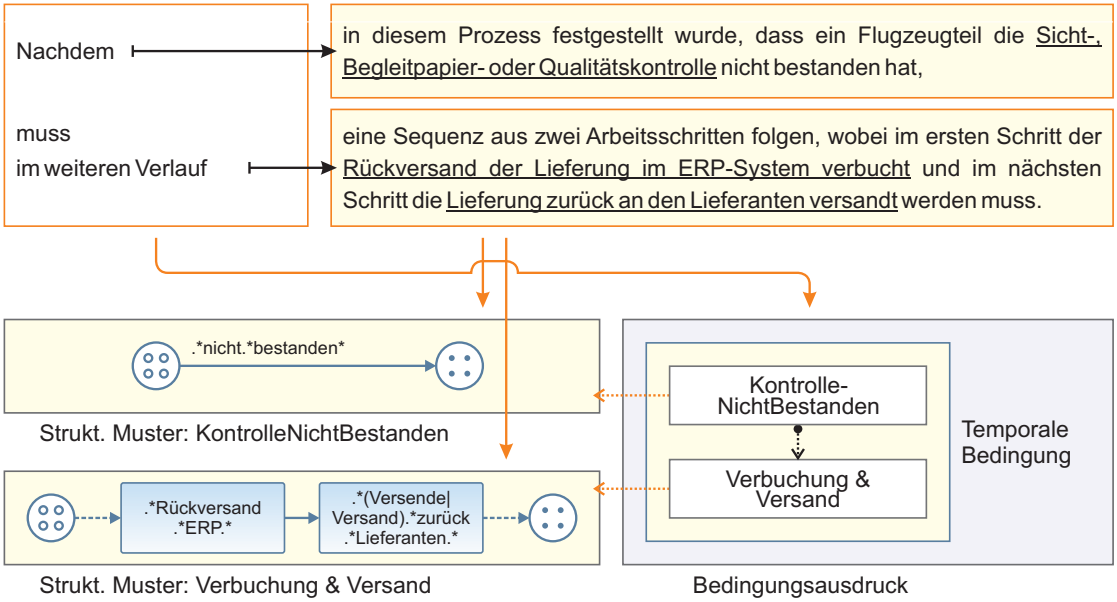


Abbildung 5.25.: Modellierung einer temporalen Bedingung

Bevor auf die verschiedenen PCML-Modellierungskonstrukte zur Spezifikation temporaler Bedingungen eingegangen wird, wird im Folgenden zunächst definiert, was unter der temporalen Beziehung zweier Instanzen struktureller Muster innerhalb eines BPMN-Modells zu verstehen ist:

Definition 5.1 (Temp. Beziehung zwischen Instanzen struktureller Muster) *Innerhalb eines BPMN-Modells besteht eine temporale Beziehung zwischen einer Instanz I_1 eines strukturellen Musters M_1 und einer Instanz I_2 eines strukturellen Musters M_2, falls zur Laufzeit die Ausführung eines Ausgangsobjekts von I_1 die Ausführung eines Eingangsobjekts von I_2 zur Folge haben kann.*

Wie an dieser Definition zu sehen ist, werden bei temporalen Bedingungen strukturelle Aspekte eines BPMN-Modells mit Laufzeitaspekten in Verbindung gebracht. Ziel derartiger Bedingungen ist die Überprüfung von Anforderungen bezüglich der modellierten Reihenfolge von Modellelementanordnungen innerhalb eines BPMN-Modells. Bei der Bestimmung der Reihenfolge kommt es im vorliegenden Ansatz nicht darauf an, ob zwischen diesen Anordnungen Zyklen auftreten, was sich in der Definition widerspiegelt („[. . .] zur Folge haben kann."). Diese Lockerung ist auf die Annahme zurückzuführen, dass bei der Ausführung von Geschäftsprozessmodellen normalerweise keine Endlosschleife auftritt, worauf im weiteren Verlauf noch genauer eingegangen wird.

Zur Spezifikation von Anforderungen an die temporale Beziehung zwischen Instanzen struktureller Muster stehen acht Modellierungskonstrukte zur Verfügung (siehe Abbil-

dung 5.26). Mithilfe von Bedingungen vom Typ *Nachfolger* (MOF-Klasse: `ResponseCondition`) und *Direktnachfolger* (MOF-Klasse: `ChainResponseCondition`) kann auf der einen Seite ausgedrückt werden, dass auf jedes Ausgangsobjekt einer Instanz von M_1 im weiteren Verlauf bzw. direkt ein Eingangsobjekt einer Instanz von M_2 folgen muss. Auf der anderen Seite kann mithilfe von Bedingungen vom Typ *Vorgänger* (MOF-Klasse: `PrecedenceCondition`) und *Direktvorgänger* (MOF-Klasse: `ChainPrecedenceCondition`) ausgedrückt werden, dass jedem Eingangsobjekt einer Instanz von M_2 ein Ausgangsobjekt von M_1 im vorherigen Verlauf bzw. direkt vorausgehen muss. Das Modellierungskonstrukt *Abfolge* (MOF-Klasse: `SuccessionCondition`) fasst die Semantik der Nachfolger- und Vorgänger-Modellierungskonstrukte zusammen und stellt daher syntaktischen Zucker dar. Ebenso fasst das Modellierungskonstrukt *Direktabfolge* (MOF-Klasse: `ChainSuccessionCondition`) die Semantik der Direktnachfolger- und Direktvorgänger-Modellierungskonstrukte zusammen.

Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Nachfolger	ResponseCondition	structuralPattern1 : StructuralPattern structuralPattern2 : StructuralPattern	
Direktnachfolger	ChainResponseCondition		
Vorgänger	PrecedenceCondition		
Direktvorgänger	ChainPrecedenceCondition		
Abfolge	SuccessionCondition		
Direktabfolge	ChainSuccessionCondition		
Negierte Abfolge	Negated-SuccessionCondition		
Negierte Direktabfolge	Negated-ChainSuccessionCondition		

Abbildung 5.26.: PCML-Modellierungskonstrukt: Temporale Bedingungen

Um auszudrücken, dass auf kein Ausgangsobjekt einer Instanz von M_1 im weiteren Verlauf ein Eingangsobjekt einer Instanz von M_2 folgen und umgekehrt keinem Eingangsobjekt einer Instanz von M_2 ein Ausgangsobjekt von M_1 im vorherigen Verlauf vorausgehen darf, kann eine Bedingung vom Typ *Negierte Abfolge* (MOF-Klasse: `NegatedSuccessionCondition`) verwendet werden. Ebenso kann eine Bedingung vom Typ *Negierte Direktabfolge* (MOF-Klasse: `NegatedChainSuccessionCondition`) verwendet werden, um auszudrücken, dass auf kein Ausgangsobjekt einer Instanz von M_1 direkt ein Eingangsobjekt einer Instanz von M_2 folgen und umgekehrt keinem Eingangsobjekt einer Instanz von M_2 ein Ausgangsobjekt von M_1 direkt vorausgehen darf.

5.3.3. Logische Operatoren

Drei Modellierungskonstrukte ermöglichen die Spezifikation zusammengesetzter Bedingungsausdrücke (siehe Abbildung 5.27). Zusammengesetzte Bedingungsausdrücke bestehen aus musterbasierten Bedingungen und logischen Operatoren. Die Modellierungskonstrukte *Bedingungskonjunktion* (MOF-Klasse: `ConstraintConjunction`) und *Inklusive Bedingungsdisjunktion* (MOF-Klasse: `ConstraintInclusiveDisjunction`) können mithilfe des Modellierungskonstrukts *Operand* (MOF-Assoziation: `OperandsForConstraintExpression`) mit Bedingungen und anderen logischen Operatoren verbunden werden, wobei mindestens zwei Verbindungen vorhanden sein müssen. Eine Bedingungskonjunktion fordert, dass bei der Überprüfung eines BPMN-Modells alle Bedingungen erfüllt sein müssen, mit denen sie verbunden ist. Eine Bedingungsdisjunktion fordert, dass mindestens eine der Bedingungen erfüllt sein muss, mit denen sie verbunden ist.

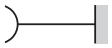
Modellierungskonstrukt	MOF-Klassenname	Attribute	Graphische Notation
Bedingungskonjunktion	ConstraintConjunction	children : Collection<ConstraintExpression>	
Inklusive Bedingungsdisjunktion	Constraint-InclusiveDisjunction		
Operand	– (Assoziation)		

Abbildung 5.27.: PCML-Modellierungskonstrukt: Logische Operatoren

5.4. Implementierung

Zur graphischen Modellierung struktureller Muster mithilfe von PPML und zur graphischen Modellierung von Bedingungsausdrücken mithilfe von PCML wurden mit dem *Pattern Composer* bzw. *Constraint Composer* zwei entsprechende graphische Modellierungswerkzeuge entwickelt, über welche die nächsten zwei Abschnitten einen kurzen Überblick geben. Analog zum OWL Composer (siehe Abschnitt 4.4.2) basieren beide Modellierungswerkzeuge auf dem Graphics Framework des SAP NetWeaver Developer Studios und liegen in Form von Eclipse Plug-ins vor. Eine Übersicht der entwickelten Eclipse Plug-ins ist in Abbildung 5.28 dargestellt.

5.4.1. Pattern Composer

Ein Screenshot des Pattern Composers ist in Abbildung 5.29 dargestellt. Zur Verwaltung struktureller Muster wurde ein eigener Eclipse-Projekttyp eingeführt. Jedes strukturelle Muster wird intern durch zwei MOF-Modelle repräsentiert: ein Modell auf Basis des PCDM, das die Elemente des Musters enthält, sowie ein Modell auf Basis des Piktogramm-Metamodells, das die graphische Darstellung des Musters beschreibt. Auf

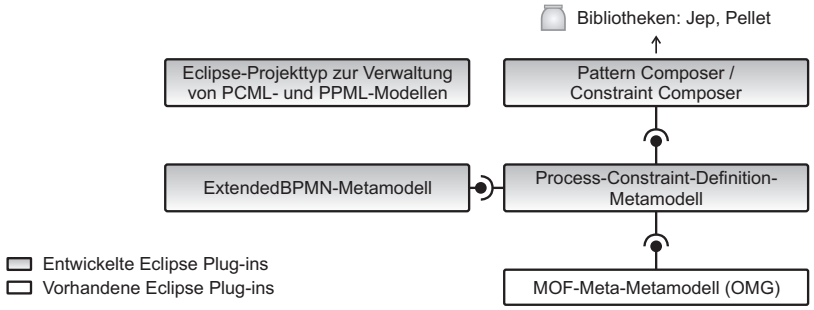


Abbildung 5.28.: Entwickelte Eclipse Plug-ins

der linken Seite der Abbildung ist der graphische Editor für strukturelle Muster dargestellt. Die innerhalb dieses Editors zur Verfügung stehenden PPML-Modellierungskonstrukte sind in der Werkzeugleiste auf der rechten Seite der Abbildung aufgelistet.

Wird ein Modellelement innerhalb des graphischen Editors selektiert, werden je nach Typ des Modellelements dessen Attribute und ihr aktueller Wert auf der Eigenschaftsansicht von Eclipse angezeigt und können dort vom Benutzer verändert werden. Die umfangreichsten Einstellungsmöglichkeiten bieten generische Tasks. Auf der entsprechenden Eigenschaftsansicht kann einem generischen Task ein regulärer Ausdruck, ein semantischer Ausdruck und eine Liste aus Webdienstmethoden zugewiesen werden. Ein semantischer Ausdruck wird in Form eines aussagenlogischen Ausdrucks spezifiziert (siehe Abbildung 5.30). Dabei können OWL-Klassen, die als atomare Formeln behandelt werden, über einen entsprechenden Dialog ausgewählt und mit Junktoren verknüpft werden. Die Überprüfung der Korrektheit eines aussagenlogischen Ausdrucks wird mithilfe der Jep-Bibliothek [45] bewerkstelligt, die das Parsen und Auswerten mathematischer Ausdrücke ermöglicht. Vor der Überprüfung muss ein semantischer Ausdruck mithilfe eines entsprechenden Algorithmus in das von dieser Bibliothek benötigte Objektmodell transformiert werden.

Bei Auswahl eines generischen Ereignisses wird auf der Eigenschaftsansicht eine Liste aus Kontrollkästchen angezeigt, die mit den für dieses Ereignis zulässigen Auslösern beschriftet sind. Da der Process Composer nicht alle zulässigen Kombinationen aus Ereignistyp und Auslöser unterstützt, sind die entsprechenden Kontrollkästchen deaktiviert. Durch Auswahl von Kontrollkästchen kann ein Benutzer festlegen, welche Ereignisse in BPMN-Modellen durch das generische Ereignis identifiziert werden. Auf vergleichbare Weise wird bei Auswahl eines generischen Gateways eine Liste aus Kontrollkästchen angezeigt, die mit den in BPMN verfügbaren Gateway-Typen beschriftet sind. Abbildung 5.31 zeigt zwei Screenshots dieser Eigenschaftsansichten.

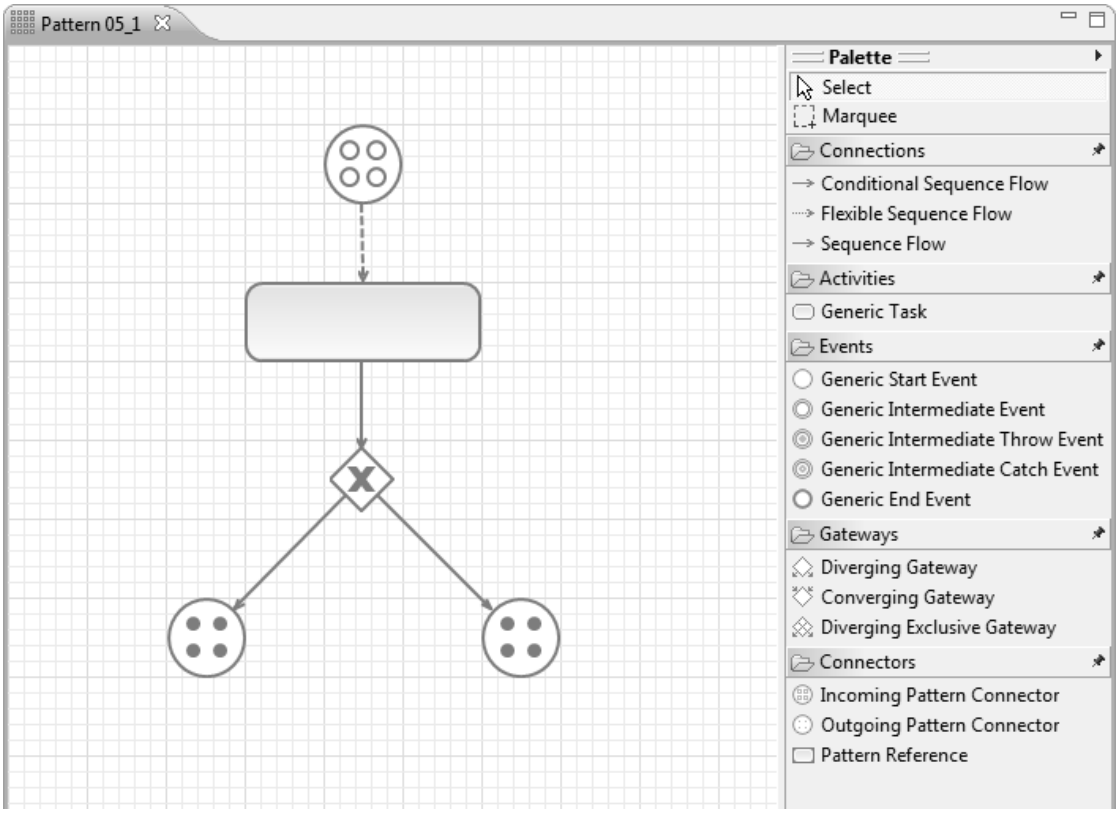


Abbildung 5.29.: Pattern Composer

5.4.2. Constraint Composer

Abbildung 5.32 zeigt einen Screenshot des Constraint Composers. Zur Verwaltung musterbasierter Bedingungen dient derselbe Eclipse-Projekttyp, der zur Verwaltung struktureller Muster eingeführt wurde. Analog zu strukturellen Mustern wird jeder Bedingungsausdruck intern durch zwei MOF-Modelle repräsentiert: ein Modell auf Basis des PCDM, das die Elemente des Bedingungsausdrucks enthält, und ein Modell auf Basis des Piktogramm-Metamodells, das die graphische Darstellung des Bedingungsausdrucks beschreibt. Auf der linken Seite der Abbildung ist wiederum der graphische Editor für Bedingungsausdrücke dargestellt. Die innerhalb des Editors zur Verfügung stehenden PCML-Modellierungskonstrukte sind in der Werkzeugleiste auf der rechten Seite der Abbildung aufgelistet.

Auch innerhalb des Constraint Composers wird bei Auswahl eines Modellelements je nach Typ eine entsprechende Eigenschaftsansicht angezeigt. Bei binären Bedingungen, die eine Anforderung an die existenzielle oder temporale Beziehung zwischen Instanzen zweier struktureller Muster ausdrücken, müssen beispielsweise zwei PPML-Modelle ausgewählt werden, die sich innerhalb desselben Projekts befinden müssen.

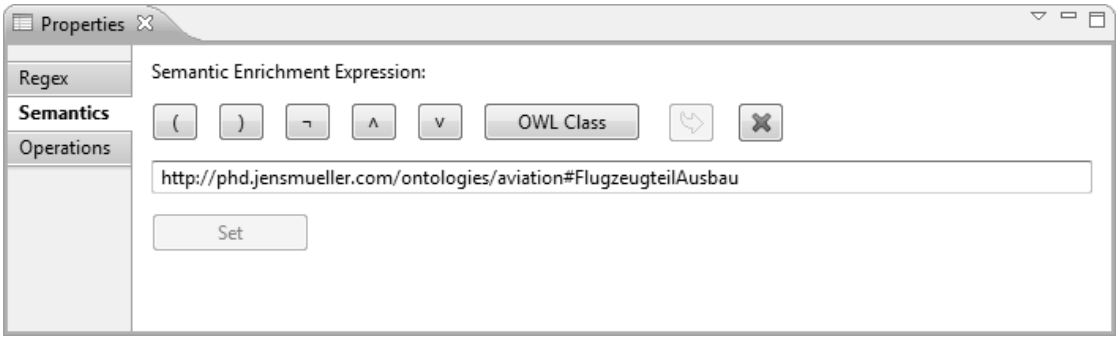


Abbildung 5.30.: Eigenschaftsansicht: Generischer Task

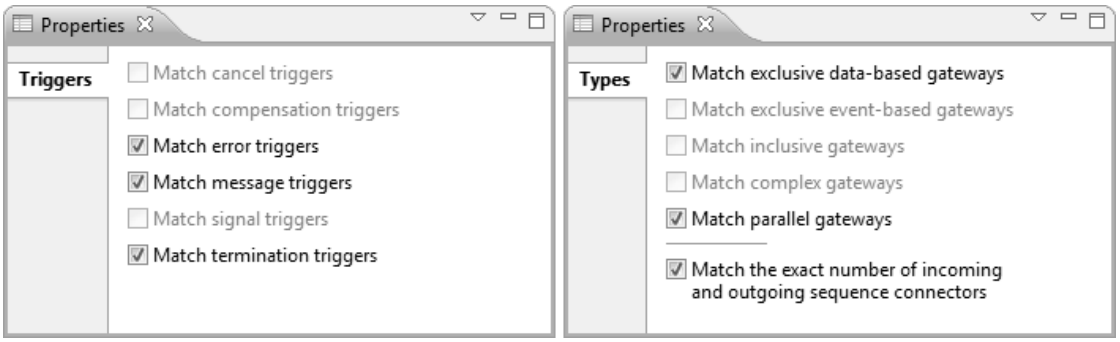


Abbildung 5.31.: Eigenschaftsansicht: Gen. Endereignis und gen. Gateway

5.4.3. Verknüpfung von BPMN-Modellen mit Bedingungsausdrücken

Nachdem in Abschnitt 5.4.1 und 5.4.2 Werkzeuge zur Modellierung struktureller Muster und zur Modellierung von Bedingungsausdrücken vorgestellt wurden, musste noch eine Möglichkeit geschaffen werden, um BPMN-Modelle mit Bedingungsausdrücken innerhalb des Process Composers verknüpfen zu können. Zu diesem Zweck wurde zunächst das ExtendedBPMN-Metamodell (siehe Abschnitt A.3 im Anhang) um eine Klasse namens `ExtendedPool` erweitert, bei der es sich um eine Unterklasse der Klasse `Pool` des BPMN-Metamodells handelt. Darüber hinaus wurde in diesem Metamodell eine Assoziation namens `ConstraintsForPool` definiert, die eine Verknüpfung von Instanzen der Klasse `ExtendedPool` mit Bedingungsausdrücken erlaubt. Auf Grundlage dieser Änderungen wurde der Process Composer dahingehend modifiziert, dass eine Instanz dieser Klasse bei Erstellung eines neuen BPMN-Modells verwendet wird. Schließlich wurde die Eigenschaftsansicht, die bei Auswahl eines Pools angezeigt wird, um einen Reiter mit der Beschriftung *Constraints* erweitert, auf dem die Verknüpfung eines Pools mit Bedingungsausdrücken vorgenommen werden kann und auf dem eine Liste aller zugeordneten Bedingungsausdrücke angezeigt wird (siehe Abbildung 5.33).

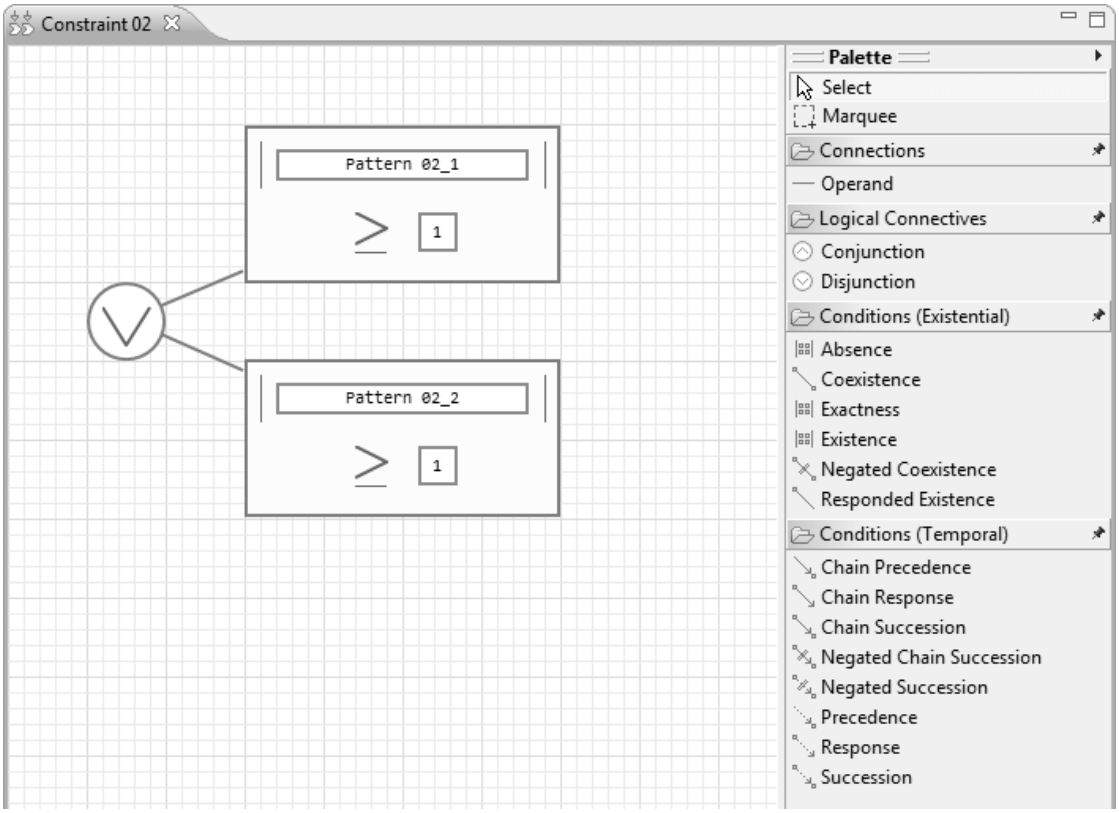


Abbildung 5.32.: Constraint Composer

5.5. Stand der Wissenschaft und Technik

Das in diesem Kapitel vorgestellte Konzept des strukturellen Musters ist nicht mit dem Konzept des Entwurfsmusters zu verwechseln, wenngleich beide Konzepte gewisse Parallelen aufweisen. Der Begriff des Entwurfsmusters stammt aus dem Bereich der objektorientierten Softwareentwicklung und wurde von Gamma et al. [GHJV09] geprägt. Er beschreibt eine Vorgehensweise zur Lösung eines wiederkehrenden Problems im Rahmen des Softwareentwurfs. Der Begriff des Entwurfsmusters kann auch auf Sprachen zur Modellierung von Geschäftsprozessen übertragen werden. Die *Workflow Patterns Initiative* stellt auf ihrer Internetseite [46] beispielsweise Beschreibungen von Workflow-Mustern zur Verfügung, welche die Bereiche Kontrollfluss, Daten, Ressourcen und Ausnahmebehandlung abdecken. Diese Workflow-Muster beschreiben wiederkehrende Anforderungen, die an Werkzeuge zur Geschäftsprozessmodellierung gestellt werden.

In [FE03] wird erläutert, dass es sinnvoll sein kann, Entwurfsmuster sowohl bei der Modellierung von Geschäftsprozessen zu verwenden, als auch zu überprüfen, ob ein bereits vorhandenes Geschäftsprozessmodell ein bestimmtes Entwurfsmuster implementiert. Auf Grundlage dieser Idee wird in [FES05, FESS06] eine auf UML basierende Sprache zur gra-

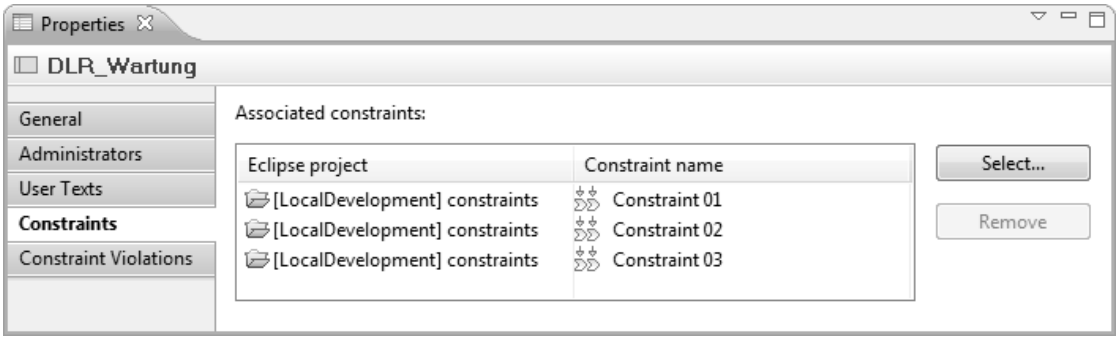


Abbildung 5.33.: Pool-Eigenschaftsansicht im Process Composer

phischen Modellierung von Entwurfsmustern für Geschäftsprozessmodelle namens *Process Pattern Specification Language* vorgestellt, die am Beispiel von Qualitätsanforderungen veranschaulicht wird. Entwurfsmuster, die auf Grundlage dieser Sprache modelliert wurden, können als bausteinartige Vorlage bei der Modellierung von Geschäftsprozessmodellen verwendet werden. Die Process Pattern Specification Language enthält alle Modellierungskonstrukte zur Modellierung von UML-Aktivitätsdiagrammen, ist jedoch um zusätzliche Modellierungskonstrukte erweitert. Einige der in diesem Kapitel vorgestellten PPML- und PCML-Modellierungskonstrukte ähneln diesen hinzugefügten Modellierungskonstrukten, beispielsweise Vorgänger- und Nachfolger-Bedingungen.

Im Gegensatz zur Process Pattern Specification Language handelt es sich bei strukturellen Mustern nicht zwangsläufig um Entwurfsmuster, die im Rahmen der Modellierung von Geschäftsprozessen als bausteinartige Vorlage verwendet werden. Auch werden bei der Process Pattern Specification Language Modellelemente und temporale Bedingungen, die sich auf diese Elemente beziehen, mithilfe derselben graphischen Notation ausgedrückt, während der in diesem Kapitel vorgestellte Ansatz eine strikte Trennung zwischen strukturellen Mustern und darauf basierenden Bedingungen zieht. Ein weiterer Unterschied zwischen beiden Ansätzen besteht in der unterschiedlichen Semantik vergleichbarer Modellierungskonstrukte, da sich die Überprüfung, ob ein Geschäftsprozessmodell ein auf Grundlage der Process Pattern Specification Language spezifiziertes Entwurfsmuster implementiert, an der Laufzeitsemantik dieses Modells orientiert, während sich ein strukturelles Muster auf dessen Struktur bezieht.

In mehreren Veröffentlichungen werden Ansätze vorgeschlagen, die es ermöglichen, Geschäftsprozessmodelle mittels einer graphischen Notation abzufragen. Diese Ansätze ähneln der in diesem Kapitel vorgeschlagenen Process Pattern Modeling Language, da zum einen eine graphische Notation verwendet wird, die auf der Notation der zugrunde liegenden Geschäftsprozessmodelle basiert, zum anderen die Struktur dieser Modelle Gegenstand von Abfragen ist. In [BEKM05, BEKM06, BEKM08] wird mit *BP-QL* eine Notation zur Abfrage von BPEL-Prozessen präsentiert, deren Semantik jedoch nicht auf BPMN übertragbar ist. Parallel zur Entwicklung von PPML wurden zwei weitere Ansätze zur Abfrage von BPMN mittels einer graphischen Notation vorgestellt: *BPMN-Q*

[Awa07] und die *BPMN Visual Query Language* [FT08, Fra08, FT09]. Teilweise weisen die Modellierungskonstrukte von PPML Übereinstimmungen mit den Modellierungskonstrukten der jeweiligen Notation dieser Ansätze auf. Allerdings führen die von PPML zusätzlich zur Verfügung gestellten Modellierungskonstrukte (z. B. flexible Sequenzflüsse mit Kardinalitäten, bedingte Sequenzflüsse, Musterreferenzen, etc.) zu einer insgesamt höheren Ausdrucksmächtigkeit. BPMN-Q bietet zudem keine Möglichkeit, die Semantik von Modellelementen zu spezifizieren und sich auf diese Semantik innerhalb von Anfragen zu beziehen. Darüber hinaus basiert die Repräsentation von PPML- und PCML-Modellen auf einem anerkannten Industriestandard (MOF). Abgesehen davon zielt keiner dieser Ansätze darauf ab, Geschäftsprozessmodelle auf Grundlage von Abfrageergebnissen auf die Einhaltung existenzieller oder temporaler Bedingungen zu verifizieren.

Zahlreiche Veröffentlichungen beschäftigen sich mit der graphischen Spezifikation von Eigenschaften, die Geschäftsprozessmodelle zu erfüllen haben. Eine graphische Notation auf Basis von BPMN zur Modellierung temporaler Eigenschaften prozessbasierter Webanwendungen wird in [Bra05, BDSV05] vorgestellt, graphische Verfahren zur deklarativen Modellierung automatisierter Geschäftsprozessmodelle (*ConDec* und *DecSerFlow*) in [AP06a, AP06b, AP06c]. Das in [FF08] präsentierte *Temporal Logics Visualization Framework* bietet eine Möglichkeit zur graphischen Modellierung temporaler Eigenschaften ereignisorientierter Prozessketten. Trotz bestehender Parallelen zu PCML beziehen sich die mittels der genannten Ansätze modellierten Eigenschaften immer auf die Laufzeitsemantik von prozessbasierten Anwendungen und Geschäftsprozessmodellen, im Gegensatz zu PCML jedoch nicht auf deren Struktur.

6. Suche nach Instanzen struktureller Muster in BPMN-Modellen

Zur Spezifikation betriebswirtschaftlicher Anforderungen in Form struktureller Muster und darauf basierender Bedingungsausdrücke wurden in Kapitel 5 zwei graphische Modellierungssprachen beschrieben: PPML (siehe Abschnitt 5.2) und PCML (siehe Abschnitt 5.3). Zur Verifikation eines BPMN-Modells in Bezug auf einen Bedingungsausdruck, der mit dem Modell verknüpft ist, wurde eine dreistufige Vorgehensweise konzipiert. Im Rahmen dieser Vorgehensweise wird das BPMN-Modell zunächst nach Instanzen struktureller Muster durchsucht (im Folgenden auch verkürzt als *musterbasierte Suche* bezeichnet), die von musterbasierten Bedingungen innerhalb des Bedingungsausdrucks referenziert werden. Im nächsten Kapitel wird beschrieben, wie die Bedingungen innerhalb eines Bedingungsausdrucks und der Ausdruck selbst auf Grundlage des Resultats der musterbasierten Suche ausgewertet werden. Die Bestandteile eines Bedingungsausdrucks sind noch einmal beispielhaft in Abbildung 6.1 dargestellt. Der darin abgebildete *Bedingungsausdruck 12* (siehe Abschnitt 3.4.2) verknüpft die Bedingungen B_1 und B_2 mit dem $\wedge$-Operator, d. h. der Bedingungsausdruck ist erfüllt, sofern B_1 und B_2 erfüllt sind. Zusammen referenzieren beide Bedingungen drei strukturelle Muster.

Abbildung 6.1.: Bedingungsausdruck (Beispiel)

Zur Durchführung der musterbasierten Suche wird ein entsprechender Algorithmus benötigt. Die Eingabe dieses Algorithmus muss zum einen aus einem strukturellen Muster, zum anderen aus dem BPMN-Modell, das nach Instanzen dieses Musters durchsucht werden soll, bestehen. Im mathematischen Sinn handelt es sich dabei um zwei Mengen aus Modellelementen: eine Menge aus Modellelementen auf Basis des PCDM-Metamodells und eine Menge aus Modellelementen auf Basis des BPMN-Metamodells. Das Ergebnis des Algorithmus ist mathematisch betrachtetet eine Menge aus Teilmengen der Menge, die das Geschäftsprozessmodell repräsentiert. Bei diesen Teilmengen handelt es sich um gefundene Instanzen des strukturellen Musters. Der Algorithmus kann als Ergebnis auch die leere Menge zurückliefern. In diesem Fall enthält das Geschäftsprozessmodell keine Instanz des strukturellen Musters.

Die wesentliche Aufgabe eines Algorithmus zur musterbasierten Suche ist die Identifikation von Elementen innerhalb eines BPMN-Modells anhand eines strukturellen Musters. Bei dieser Aufgabe handelt es sich um ein kombinatorisches Problem, da für jedes Element des PPML-Modells überprüft werden muss, ob es mit einem oder mehreren Elementen des untersuchten BPMN-Modells korrespondiert. Zur Identifikation einer Instanz eines strukturellen Musters muss jedes Element des PPML-Modells mit einem Element des BPMN-Modells korrespondieren. Darüber hinaus muss ein Algorithmus in der Lage sein, alle Instanzen eines strukturellen Musters zu finden. Das Geschäftsprozessmodell G in Abbildung 6.2 enthält beispielsweise zwei Instanzen des strukturellen Musters M.

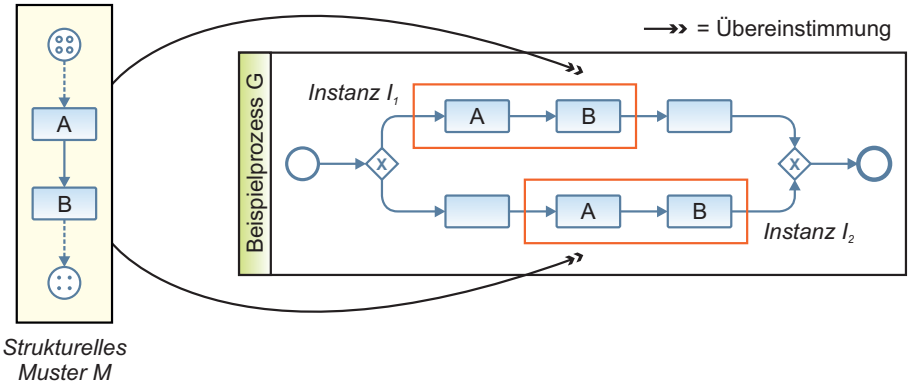


Abbildung 6.2.: Identifikation von Instanzen eines strukturellen Musters

Hinsichtlich eines derartigen Suchverfahrens wurde in der vorliegenden Arbeit auf die Integration bestehender Werkzeuge gesetzt, deren Mechanismen zur musterbasierten Suche geeignet sind. Für diese Entscheidung sprach, dass die in dieser Arbeit verwendeten Werkzeuge und deren Mechanismen ausgereift und optimiert sind. Wie im weiteren Verlauf noch gezeigt wird, arbeiten die auf Grundlage dieser Werkzeuge entwickelten Suchverfahren sehr effizient.

Bei den Werkzeugen, die in der vorliegenden Arbeit verwendet wurden, handelt es sich um Abfrageprozessoren, regelbasierte Systeme und Reasoner, die als Eingabe Abfragen oder Produktionsregeln entgegennehmen, auf gewissen Daten (z. B. auf einem Modell, das als Menge aus Modellelementen aufgefasst wird) operieren und schließlich ein Ergebnis (z. B. eine Menge aus Teilmengen dieses Modells) zurückgeben. Abfragen und Regeln werden dabei in einer entsprechenden Sprache spezifiziert. Die Besonderheit der verwendeten Werkzeuge liegt darin, dass die jeweilige Sprache zur Spezifikation der Eingabedaten einen deklarativen Ansatz verfolgt. Eine verbreitete deklarative Sprache zur Definition, Abfrage und Manipulation von Daten in relationalen Datenbanken ist beispielsweise SQL. Bei der Verwendung deklarativer Programmiersprachen steht im Gegensatz zu imperativen Programmiersprachen die Beschreibung eines Problems im Vordergrund, nicht dessen Lösung. Die Lösung eines deklarativ formulierten Problems wird dann von spezialisierten Algorithmen durchgeführt, im Fall von SQL beispielsweise von einem Abfrageprozessor, der Teil eines jeden Datenbankmanagementsystems ist.

Die in dieser Arbeit entwickelten musterbasierten Suchverfahren basieren auf einer gemeinsamen Vorgehensweise. Kern dieser Vorgehensweise ist die Übersetzung von PPML-Modellen in die benötigte Eingaberepräsentation des integrierten Werkzeugs, das zur Lösung des Suchproblems verwendet wird. Die gemeinsame Vorgehensweise untergliedert sich in vier Phasen (siehe Abbildung 6.3):

1. Um ein BPMN-Modell auf das Vorhandensein von Instanzen eines strukturellen Musters zu überprüfen, wird das PPML-Modell in ein deklarativ formuliertes Suchproblem überführt, wobei die Sprache zur Spezifikation von Eingabedaten des integrierten Werkzeugs verwendet wird.

2. Falls das integrierte Werkzeug nicht direkt auf das BPMN-Modell zugreifen kann, wird es in die vom Werkzeug benötigte Repräsentation des Suchraums überführt.

3. Das integrierte Werkzeug wird zur Lösung des Suchproblems verwendet.

4. Nach Abschluss der Suche wird die Ausgabe des integrierten Werkzeugs in eine zur Weiterverarbeitung geeignete Form überführt.

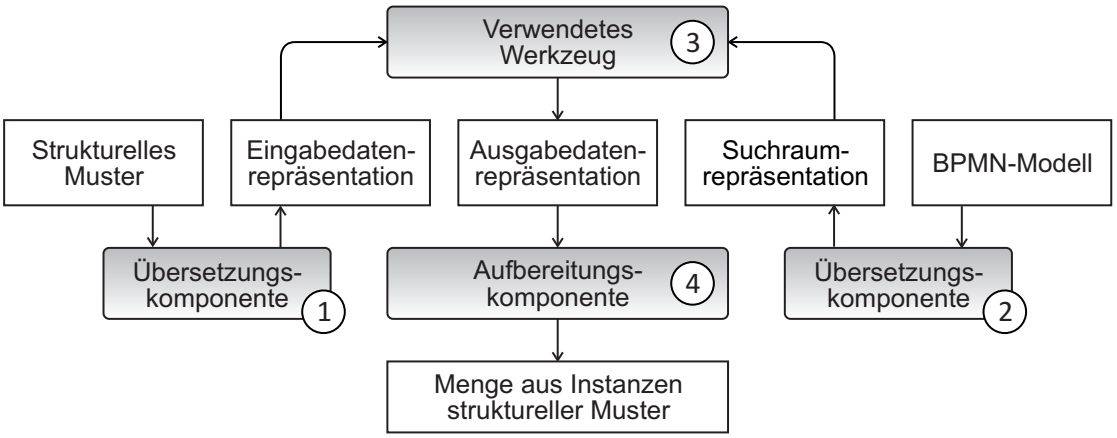


Abbildung 6.3.: Ablauf der musterbasierten Suche

Abgesehen von den verschiedenen Werkzeugen unterscheiden sich die in den nächsten Abschnitten vorgestellten musterbasierten Suchverfahren darüber hinaus fundamental voneinander, was die Repräsentation des Suchraums betrifft. Während einige Werkzeuge direkt auf MOF-Modelle zugreifen können, operieren Reasoner auf Grundlage von Ontologien. Wird als Suchraum eine Ontologie benötigt, muss das zu überprüfende BPMN-Modell zunächst entsprechend transformiert werden. Reasoner unterscheiden sich auch bei der Suche stark von anderen Werkzeugtypen (z. B. regelbasierten Systemen), worauf im späteren Verlauf näher eingegangen wird.

6.1. Verwendung der MOIN Query Language

Zunächst wurde im Rahmen der vorliegenden Arbeit versucht, das Problem der musterbasierten Suche mit der von MOIN zur Verfügung gestellten Funktionalität zu lösen. Zur Verwaltung von Modellen unterstützt MOIN die JMI-Schnittstelle (siehe Abschnitt 2.1.1). Darüber hinaus gibt es die Möglichkeit, Modelle mithilfe der *MOIN Query Language* (MQL) abzufragen. Bei MQL handelt es sich um eine deklarative Sprache, deren Syntax an SQL angelehnt ist und die zur Abfrage von Modellen verwendet wird. Zwar ist MQL im Vergleich zu JMI weniger ausdrucksmächtig, allerdings werden MQL-Abfragen in Bezug auf Speicherverbrauch und Leistung effizienter verarbeitet. Die Verarbeitung wird dabei vom MQL-Abfrageprozessor durchgeführt. Ein vergleichbarer Abfrageprozessor existiert in Form der *Model-Query-Komponente* innerhalb des *Eclipse Modeling Frameworks* (EMF) [47], das Teil der Eclipse-Plattform ist.

Im Vergleich zu SQL arbeitet MQL nicht auf Grundlage von Tabellen innerhalb eines Datenbankschemas, sondern auf Basis von Modellen und der in den zugrunde liegenden Metamodellen definierten Klassen. Bei der Verarbeitung einer MQL-Abfrage wird zunächst das kartesische Produkt der angegebenen Typen berechnet. Werden beispielsweise bei einer Abfrage die Metamodellklassen R und S angegeben, so resultiert aus der Berechnung des kartesischen Produkts $R \times S$ die Menge aller Kombinationen der Instanzen von R und S innerhalb der in die Abfrage einbezogenen Modelle, d. h. jede Instanz von R wird mit jeder Instanz von S kombiniert. Darüber hinaus können wie bei SQL Einschränkungen spezifiziert werden, die zu einer Filterung dieser Menge führen. Eine MQL-Abfrage besteht aus dem obligatorischen `select`-Teil, dem obligatorischen `from`-Teil und dem optionalen `where`-Teil (siehe Listing 6.1). Im `from`-Teil werden die Metamodellklassen angegeben, die in die Berechnung des kartesischen Produkts einfließen sollen. Im `where`-Teil können Filterkriterien auf Basis von Attributen der Metamodellklassen innerhalb des `from`-Teils spezifiziert werden. Im `select`-Teil werden Modellelemente oder Attribute primitiven Typs von Modellelementen angegeben, die innerhalb der Ergebnismenge zurückgeliefert werden sollen.

```
<mql-query> ::= select <select-clause>
                from <from-clause>
                (where <where-clause>)*
```

Listing 6.1: BNF für MQL-Abfragen (Ausschnitt)

Zur Suche nach Instanzen eines strukturellen Musters mithilfe des MQL-Abfrageprozessors muss das PPML-Modell in die benötigte Eingabedatenrepräsentation transformiert werden, d. h. in eine MQL-Abfrage. Diese Transformation kann bewerkstelligt werden, indem für jedes Element des PPML-Modells entsprechende Fragmente der resultierenden MQL-Abfrage erzeugt werden. Listing 6.2 zeigt eine MQL-Abfrage, die mit dem strukturellen Muster in Abbildung 6.2 korrespondiert. Dieses Muster beschreibt eine Anordnung von zwei Tasks, die durch einen Sequenzfluss miteinander verbunden sind. Für jedes die-

ser Elemente wird im `from`-Teil der MQL-Abfrage die entsprechende Klasse des BPMN-Metamodells eingetragen und mithilfe des Schlüsselworts `as` mit einem eindeutigen Alias versehen (`task_1`, `sequence_connector_1`, `task_2`). Für jeden Task muss darüber hinaus eine weitere Klasse in die Berechnung des kartesischen Produkts einbezogen werden, deren Instanzen die Beschriftung von Tasks repräsentieren (`ModelElementNameText`). Im `where`-Teil der Abfrage werden die Kombinationen aus Instanzen gefiltert, die den gewünschten Kriterien entsprechen, wobei in diesem Fall die durch `task_1` referenzierten Tasks mit der Beschriftung *A* und die durch `task_2` referenzierten Tasks mit der Beschriftung *B* von Interesse sind, die durch `sequence_connector_1` referenzierte Sequenzflüsse miteinander verbunden sind. Aufgrund der `select`-Anweisung enthält die Ergebnismenge dieser Abfrage eine Liste aus Tripeln, die jeweils aus zwei Tasks und einem Sequenzfluss bestehen, welche die gesuchte Anordnung aufweisen.

```
1  select task_1, sequence_connector_1, task_2
2
3  from Galaxy::Workflow::Task as task_1,
4       Galaxy::Text::ModelElementNameText as model_element_name_text_1,
5       Galaxy::Workflow::SequenceConnector as sequence_connector_1,
6       Galaxy::Workflow::Task as task_2,
7       Galaxy::Text::ModelElementNameText as model_element_name_text_2
8
9  where model_element_name_text_1.originalText like 'A'
10 where task_1.name = model_element_name_text_1
11 where sequence_connector_1.source = task_1
12 where sequence_connector_1.target = task_2
13 where model_element_name_text_2.originalText like 'B'
14 where task_2.name = model_element_name_text_2
```

Listing 6.2: MQL-Abfrage zur Suche nach Instanzen eines strukturellen Musters

Leider ist es innerhalb einer MQL-Abfrage, genauer gesagt im `where`-Teil einer MQL-Abfrage, nicht möglich, benutzerdefinierte Funktionen aufzurufen, die komplexere Filterungsmechanismen ermöglichen würden. Die Notwendigkeit benutzerdefinierter Funktionen soll am Beispiel der Auswertung eines semantischen Ausdrucks (siehe Abschnitt 5.2.1) aufgezeigt werden. Um einen semantischen Ausdruck auszuwerten, muss für jeden Task innerhalb eines BPMN-Modells und jede URI innerhalb des Ausdrucks überprüft werden, ob der Task zu einer OWL-Klasse mit dieser URI oder einer ihrer Unterklassen zugeordnet ist. Allerdings kann eine Überprüfung der Zuordnung zu Unterklassen nicht mit den vorhandenen Mechanismen von MQL bewerkstelligt werden. Der Grund dafür ist, dass die Ontologie, zu der die Klasse, die dem Task zugeordnet ist, zunächst mithilfe eines Reasoners klassifiziert werden muss, da die Klassifizierung Vererbungsbeziehungen zum Vorschein bringen kann, die vom Benutzer nicht explizit angegeben wurden (siehe Abschnitt 4.1.2). Allerdings ist es nicht möglich, diesen Vorgang während der Ausführung einer MQL-Abfrage durchzuführen, geschweige denn auf das Ergebnis zurückzugreifen. Selbst wenn eine Klassifizierung nicht erforderlich wäre, müsste für jede Klasse, die dem Task zugeordnet ist, rekursiv überprüft werden, ob ihre URI oder die URI einer ihrer

Oberklassen mit der URI einer Klasse innerhalb des semantischen Ausdrucks übereinstimmt, was mit den Mechanismen von MQL ebenfalls nicht möglich ist. Dennoch ist eine aufwendige Lösung denkbar, um zumindest semantische Ausdrücke mithilfe von MQL auszuwerten, die keine logischen Operatoren enthalten, also lediglich aus einer einzelnen URI bestehen. Vor der Transformation eines strukturellen Musters in eine MQL-Abfrage müssten im Zuge dieser Lösung folgende Schritte zur Laufzeit durchgeführt werden, die in Abbildung 6.4 dargestellt sind:

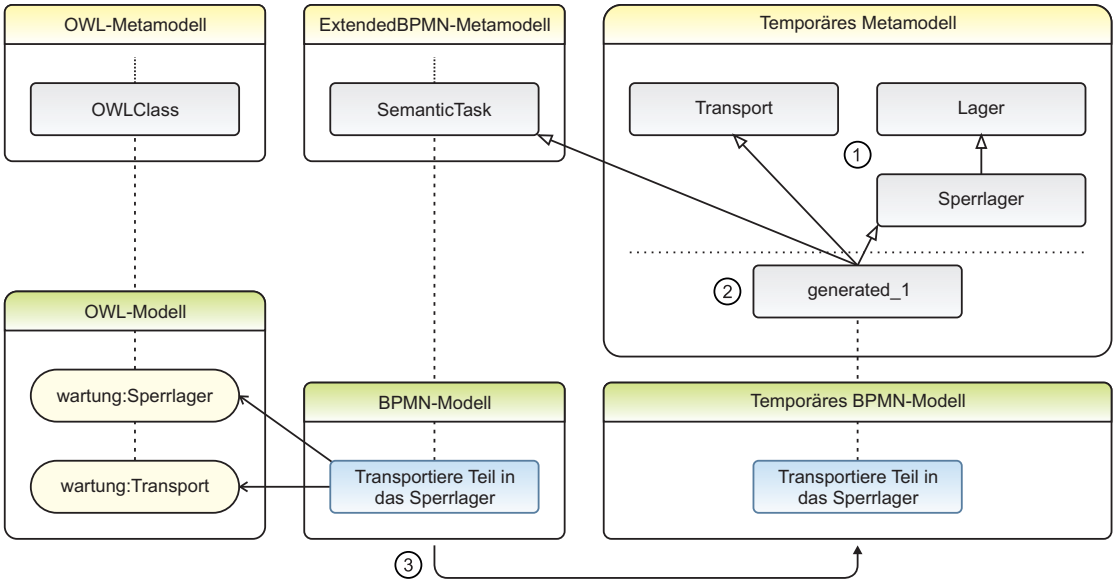


Abbildung 6.4.: Erzeugung eines temporären Metamodells

1. Zunächst wird ein temporäres Metamodell erzeugt. In diesem Metamodell wird für jede OWL-Klasse, die einer Instanz von `SemanticTask` innerhalb des zu überprüfenden BPMN-Modells zugeordnet ist, und für jede ihrer Oberklassen (auf Ontologieebene) eine Metamodellklasse generiert. Darüber hinaus werden entsprechende Vererbungsbeziehungen zwischen diesen Metamodellklassen angelegt.

2. Im Anschluss wird für jede Kombination aus OWL-Klassen, die einer Instanz von `SemanticTask` innerhalb des BPMN-Modells zugeordnet sind, eine weitere Metamodellklasse erzeugt. Diese Klasse wird sowohl als Unterklasse der Klassen, die im ersten Schritt erzeugt wurden und mit den OWL-Klassen korrespondieren, die der Instanz zugeordnet sind, als auch als Unterklasse von `SemanticTask` aus dem ExtendedBPMN-Metamodell angelegt.

3. Schließlich wird ein temporäres Modell erzeugt, das bis auf Instanzen von `SemanticTask` Kopien der Elemente des BPMN-Modells enthält. Anstelle der Instanzen von `SemanticTask` wird jeweils eine Instanz der Klasse des temporären Metamodells generiert, die im zweiten Schritt erzeugt wurde und mit der Kombination aus OWL-Klassen korrespondiert, die der Instanz zugeordnet sind.

Nach diesen vorbereitenden Maßnahmen kann ein PPML-Modell in eine entsprechende MQL-Abfrage transformiert werden. Enthält das Modell einen generischen Task, dem ein semantischer Ausdruck in Form einer einzelnen URI zugeordnet ist, muss bei der Transformation dieses Tasks im `from`-Teil der Abfrage der Name der generierten Metamodellklasse angegeben werden. Sollen innerhalb eines BPMN-Modell Tasks gesucht werden, denen die OWL-Klasse `wartung:Lager` oder eine ihrer Unterklassen zugeordnet ist, würde die zugehörige MQL-Abfrage auf Basis des in Abbildung 6.4 dargestellten Metamodells folgendermaßen laufen:

```
select task from Generated::Lager as task
```

Listing 6.3: MQL-Abfrage zur Suche semantisch angereichter Tasks

Wie dieses Beispiel zeigt, ist die Tatsache, dass MQL keine Möglichkeit bietet, benutzerdefinierte Funktionen innerhalb von MQL-Ausdrücken zu verwenden, bei der Auswertung semantischer Ausdrücke problematisch. Zwar kann diese Auswertung wie beschrieben auch auf andere Weise realisiert werden, allerdings ist dieser Lösungsweg unbefriedigend, da nur einfache Ausdrücke ausgewertet werden können und die Erzeugung von Metamodellen zur Laufzeit sehr aufwendig ist. Darüber hinaus können andere PPML-Modellierungskonstrukte überhaupt nicht auf MQL abgebildet werden können, beispielsweise flexible Sequenzflüsse.

6.2. Verwendung regelbasierter Systeme

Aufgrund der Unzulänglichkeiten von MQL wurden zwei weitere Werkzeugtypen und entsprechende Produkte analysiert und im Rahmen der Implementierung erfolgreich integriert (siehe Kapitel 8): Regelbasierte Systeme (siehe Abschnitt 2.3.4) und Reasoner (siehe Abschnitt 2.3.3.3). Regelbasierte Systeme stellen einen weiteren Werkzeugtyp dar, der zur musterbasierten Suche geeignet ist, da auch Regeln einen deklarativen Ansatz zur Beschreibung von Suchproblemen verfolgen. Eines der in Abschnitt 2.3.4 erwähnten Geschäftsregel-Managementsysteme ist die quelloffene Geschäftslogik-Integrationsplattform Drools, die als Java-Bibliothek vorliegt. Drools verwendet zur Auswertung von Regeln eine erweiterte und optimierte Variante des *Rete-Algorithmus* [For79] namens *ReteOO*. Regeln werden mithilfe der *Drools Rule Language* (DRL) oder in XML auf Basis eines entsprechenden Schemas formuliert. Beliebige Java-Objekte können als Fakten in den Arbeitsspeicher der Inferenzmaschine von Drools eingefügt werden. Dies stellt gegenüber anderen Verfahren einen Vorteil dar, da zur Suche nach Instanzen eines strukturellen Musters das zu überprüfende Geschäftsprozessmodell nicht in eine andere Repräsentation transformiert werden muss, lediglich die einzelnen Elemente des Modells müssen der Inferenzmaschine zur Verfügung gestellt werden. Ein weiterer Vorteil von Drools ist die Tatsache, dass innerhalb von Regeln benutzerdefinierte Funktionen aufgerufen werden können, genauer gesagt kann innerhalb einer Regel auf statische Methoden

einer Klasse oder auf Objekte zugegriffen werden, die vor Aktivierung der Inferenzmaschine registriert werden müssen. Aus diesen Gründen bietet Drools die besten Voraussetzungen zur Verwendung im Rahmen der musterbasierten Suche. Zu diesem Zweck muss ähnlich wie beim MQL-Ansatz verfahren werden und das strukturelle Muster in eine entsprechende Drools-Regel auf Basis der DRL-Syntax transformiert werden.

Listing 6.4 zeigt eine Drools-Regel auf Basis der DRL-Syntax, die mit dem strukturellen Muster in Abbildung 6.2 korrespondiert. Die Regel beginnt mit der Deklaration eines Paketnamens, der mithilfe des Schlüsselworts `package` spezifiziert wird. Anweisungen, die mit dem Schlüsselwort `import` oder dem Schlüsselwort `global` beginnen, deklarieren referenzierte Java-Klassen bzw. globale Variablen. Innerhalb eines Pakets können mehrere Regeln spezifiziert werden, wobei der Anfang und das Ende einer Regel durch die Schlüsselwörter `rule` und `end` gekennzeichnet wird. Die Prämisse und Konklusion einer Regel werden durch das Schlüsselwort `when` bzw. `end` eingeleitet.

```
1  package com.jensmueller.phd.rules
2
3  import (...) global (...)
4
5  rule "Structural Pattern M"
6
7    when
8      modelElementNameText1 : ModelElementNameText(originalText matches "A")
9      modelElementNameText2 : ModelElementNameText(originalText matches "B")
10     task1 : Task(name == modelElementNameText1)
11     task2 : Task(name == modelElementNameText2)
12     sequenceConnector1 : SequenceConnector
13                          (source == task1, target == task2)
14   then (...)
15
16 end
```

Listing 6.4: Prämisse einer Drools-Regel

Die Prämisse der in Listing 6.4 dargestellten Regel weist auffällige Parallelen zur äquivalenten MQL-Abfrage (siehe Listing 6.2) auf. Auch hier werden für jedes gesuchte Modellelement eine sogenannte *Typeinschränkung* (z. B. `ModelElementNameText`) und optional eine oder mehrere sogenannte *Feldeinschränkungen* angegeben (z. B. `originalText matches "A"`). Eine Typeinschränkung und zugehörige Feldeinschränkungen werden bei Drools als *Muster* (Pattern) bezeichnet, werden aber in diesem Kontext aufgrund der begrifflichen Ähnlichkeit zu strukturellen Mustern im Folgenden als *Regelbedingungen* bezeichnet. Wenn eine Instanz innerhalb des Arbeitsspeichers eine Regelbedingung erfüllt, liegt eine Übereinstimmung vor. Um eine Instanz, die mit einer Regelbedingung übereinstimmt, in anderen Regelbedingungen oder der Konklusion referenzieren zu können, kann eine Variable angegeben werden (z. B. `modelElementNameText1`), an welche die Instanz gebunden wird. Falls Drools eine Kombination aus Instanzen findet, die zu einer Erfüllung aller Regelbedingungen der Prämisse führt, wird die Konklusion der Re-

gel ausgeführt. Für die musterbasierte Suche bedeutet dies, dass das zu durchsuchende BPMN-Modell eine Instanz des korrespondierenden strukturellen Musters enthält.

6.2.1. Transformation von PPML-Modellen in Drools-Regeln

Zur Suche nach Instanzen eines strukturellen Musters mithilfe von Drools wird das jeweilige PPML-Modell (siehe Abschnitt 5.2) wie im letzten Abschnitt erwähnt in eine Drools-Regel auf Basis der DRL-Syntax transformiert. Dazu wird jedes Element des Modells einzeln in einen entsprechenden Teil der Regel übersetzt. Die bei der Übersetzung von Modellelementen erzeugten Teile der resultierenden Regel werden zunächst innerhalb einer entsprechenden Datenstruktur abgelegt (siehe Abbildung 6.5). Erst nachdem alle Modellelemente übersetzt sind, wird aus der internen Repräsentation der Regel zunächst eine Zeichenkette auf Basis der DRL-Syntax erzeugt (1). Anschließend wird mithilfe der Drools API ein `Package`-Objekt erzeugt, das wiederum auf ein `Rule`-Objekt verweist, das dieser Zeichenkette entspricht (2). Schließlich wird das `Package`-Objekt in Form einer Datei mithilfe der *Java Serialization API* [48] serialisiert (3).

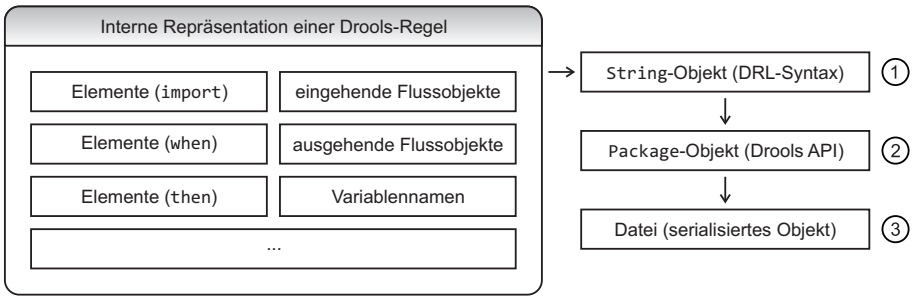


Abbildung 6.5.: Interne Repräsentation und Erzeugung einer Drools-Regel

Bei der Erzeugung der internen Repräsentation von Drools-Regeln ist zu beachten, dass PPML-Sequenzflüsse erst übersetzt werden können, wenn die generischen Flussobjekte, die ein Sequenzfluss miteinander verbindet, bereits übersetzt sind. Die Erzeugung der internen Repräsentation wird von dem in Listing 6.5 dargestellten Pseudocode-Algorithmus gesteuert, wobei die eigentliche Erzeugung durch Aufruf der Prozedur `Create` angestoßen wird. Im Folgenden wird für jedes PPML-Modellierungskonstrukt beschrieben, wie Modellelemente dieses Typs in entsprechende Teile der resultierenden Drools-Regel auf Grundlage der internen Repräsentation übersetzt werden.

6.2.1.1. Ein- und ausgehende Musterkonnektoren

Musterkonnektoren (siehe Abschnitt 5.2.6) werden bei der Transformation von PPML-Modellen ignoriert, da sie lediglich die Ein- und Ausgänge eines PPML-Modells markieren, allerdings fließen sie in die Transformation anderer Modellierungskonstrukte und in die Erzeugung der Konklusion von Drools-Regeln ein.

```
1  procedure CreateRuleData(element, traversed, ruleData):
2
3  if not traversed contains element then
4      traversed ← element
5
6      if element instanceof SequenceConnector then
7          CreateRuleData(GetSource(element), traversed, ruleData)
8          CreateRuleData(GetTarget(element), traversed, ruleData)
9
10     Create(element, ruleData)
```

Listing 6.5: Pseudocode-Prozedur: CreateRuleData

6.2.1.2. Generische Tasks

Die Übersetzung generischer Tasks (siehe Abschnitt 5.2.1) hängt davon ab, welche Attribute der generischen Tasks gesetzt sind. Abbildung 6.6 zeigt Ausschnitte zweier PPML-Diagramme M_1 und M_2, in denen jeweils ein generischer Task dargestellt ist.

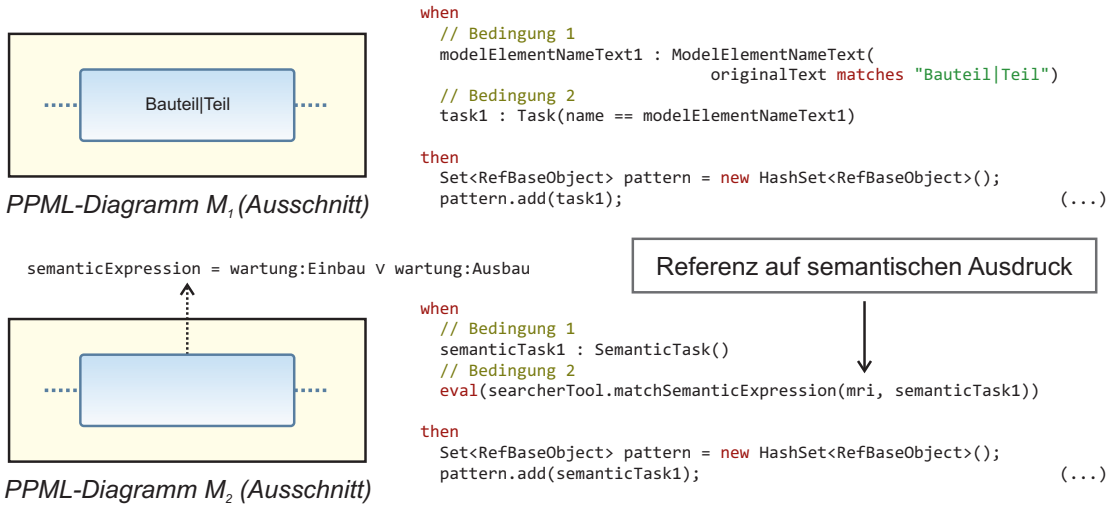


Abbildung 6.6.: Übersetzung generischer Tasks

Der Ausschnitt von M_1 zeigt einen generischen Task, dem ein regulärer Ausdruck, aber kein semantischer Ausdruck zugewiesen ist. Bei der Übersetzung dieses Tasks werden zwei Regelbedingungen erzeugt. Diese Bedingungen werden erfüllt, falls der Arbeitsspeicher von Drools einen BPMN-Task enthält, dessen Beschriftung durch den regulären Ausdruck identifiziert wird. Die Auswertung des regulären Ausdrucks wird mithilfe des Drools-Operators `matches` durchgeführt. Der Ausschnitt von M_2 zeigt im Gegensatz dazu einen generischen Task, dem ein semantischer Ausdruck, aber kein regulärer Ausdruck, zugewiesen ist. Auch bei der Übersetzung dieses Tasks werden zwei Bedingungen erzeugt. Die erste Bedingung ist erfüllt, falls der Arbeitsspeicher von Drools einen se-

mantischen Task enthält. In diesem Fall wird der semantische Task an die Variable `semanticTask1` gebunden. Die zweite Bedingung wertet den semantischen Ausdruck des generischen Tasks auf Grundlage der Ontologieklassen aus, die `semanticTask1` zugeordnet sind. Da die Auswertung des semantischen Ausdrucks mit den Sprachkonstrukten von Drools nicht bewerkstelligt werden kann, wird eine benutzerdefinierte Hilfsmethode aufgerufen (`matchSemanticExpression`), die diese Aufgabe übernimmt. Die zweite Bedingung ist erfüllt, falls diese Methode `true` zurückliefert.

Die Hilfsmethode `matchSemanticExpression` hat zwei Parameter. Der erste Parameter (`mri`) ist eine Referenz auf den semantischen Ausdruck des generischen Tasks innerhalb des zu transformierenden PPML-Modells. Im Gegensatz zum regulären Ausdruck eines generischen Tasks, der als Zeichenkette (z. B. `Bauteil|Teil`) in die Regel übernommen wird, muss bei einem semantischen Ausdruck auf das PPML-Modell zurückgegriffen werden, da es sich bei einem semantischen Ausdruck um eine Reihe von Modellelementen handelt, die nicht innerhalb einer Drools-Regel unabhängig vom ursprünglichen Modell repräsentiert werden können. Als zweiter Parameter muss der Hilfsmethode der zu überprüfende semantische Task übergeben werden. Die Auswertung des semantischen Ausdrucks wird wie in Abschnitt 5.2.1 beschrieben durchgeführt.

6.2.1.3. Generische Ereignisse und generische Gateways

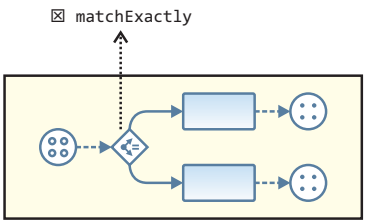
Die Übersetzung generischer Ereignisse (siehe Abschnitt 5.2.2) und Gateways (siehe Abschnitt 5.2.3) ist weniger aufwendig als die Übersetzung generischer Tasks. Generische Ereignisse werden in eine oder mehrere Regelbedingungen übersetzt. Abbildung 6.7 zeigt ein PPML-Diagramm, das ein generisches Endereignis enthält, welches mit einem eingehenden Musterkonnektor verbunden ist. Aufgrund seiner Attributswerte identifiziert das generische Endereignis lediglich BPMN-Endereignisse, bei denen es sich um Nachrichten-Ereignisse handelt. Diese Einschränkung führt in der resultierenden Regel zu entsprechenden Feldeinschränkungen. Eine vollständige Übersicht der bei der Übersetzung generischer Ereignisse und Gateways erzeugten Bedingungen ist in Abbildung A.5 im Anhang dargestellt. Auf die Übersetzung generischer Gateways, deren Attribut `matchExactly` den Wert `true` aufweist, wird im nächsten Abschnitt eingegangen.

Abbildung 6.7.: Übersetzung generischer Ereignisse

6.2.1.4. Sequenzflüsse

Vor der Übersetzung von Sequenzflüssen (siehe Abschnitt 5.2.5.1) muss überprüft werden, ob die Modellelemente, die durch einen Sequenzfluss verbunden sind, bereits übersetzt sind. Falls dem nicht so ist, wird zunächst deren Übersetzung durchgeführt, da die dabei erzeugten Variablennamen in die Übersetzung des Sequenzflusses einfließen (siehe Listing 6.5). Die bei der Übersetzung von Sequenzflüssen erzeugten Regelbedingungen hängen davon ab, welche Modellelemente die Sequenzflüsse miteinander verbinden. Handelt es sich bei der Quelle oder dem Ziel eines Sequenzflusses um einen eingehenden bzw. ausgehenden Musterkonnektor, werden beispielsweise überhaupt keine Bedingungen erzeugt (siehe Abbildung 6.7). In diesem Fall markieren die Musterkonnektoren lediglich die Ein- und Ausgänge eines PPML-Modells.

Abbildung 6.8 zeigt ein PPML-Diagramm, das aus einem divergierenden generischen Gateway besteht, der über einen eingehenden Sequenzfluss mit einem Musterkonnektor und über zwei ausgehende Sequenzflüsse jeweils mit einem generischen Task verbunden ist. Generische Gateways, die über mehr als einen ein- oder ausgehenden Sequenzfluss verfügen, stellen bei der Übersetzung einen Sonderfall dar. Im Fall des in Abbildung 6.8 dargestellten PPML-Diagramms muss zusätzlich zu den zwei Bedingungen, die bei der Übersetzung der ausgehenden Sequenzflüsse des generischen Gateways erzeugt werden (Bedingung 4 und 5), eine weitere Bedingung generiert werden (Bedingung 6). Diese Bedingung stellt sicher, dass die Prämisse der resultierenden Regel nur erfüllt ist, wenn es sich bei den BPMN-Sequenzflüssen, die bei der musterbasierten Suche von Bedingung 4 und 5 identifiziert werden, um unterschiedliche Modellelemente handelt. Ohne diese Bedingung könnte das Ergebnis der Suche fehlerhaft sein. Da das Attribut `matchExactly` des abgebildeten generischen Gateways den Wert `true` aufweist, wird schließlich eine Bedingung erzeugt (Bedingung 7), die sicherstellt, dass nur Gateways mit genau zwei ausgehenden Sequenzflüssen identifiziert werden.



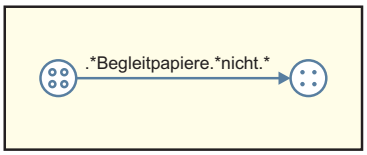
PPML-Diagramm

```
when
  // Bedingung 1
  task1 : Task()
  // Bedingung 2
  task2 : Task()
  // Bedingung 3
  divergingGateway1 : DivergingGateway()
  // Bedingung 4
  sequenceConnector2 : SequenceConnector(source == divergingGateway1,
                                         target == task1)
  // Bedingung 5
  sequenceConnector3 : SequenceConnector(source == divergingGateway1,
                                         target == task2)
  // Bedingung 6
  eval(sequenceConnector2 != sequenceConnector3)
  // Bedingung 7
  eval(divergingGateway1.getOutgoingConnectors().size() == 2)     (...)
```

Abbildung 6.8.: Übersetzung von Sequenzflüssen

6.2.1.5. Bedingte Sequenzflüsse

Die Übersetzung bedingter Sequenzflüsse (siehe Abschnitt 5.2.5.2) unterscheidet sich nur geringfügig von der Übersetzung regulärer Sequenzflüsse. Abbildung 6.9 zeigt ein PPML-Diagramm, das aus einem bedingten Sequenzfluss besteht, der mit zwei Musterkonnektoren verbunden ist. Da dem bedingten Sequenzfluss ein regulärer Ausdruck zugeordnet ist, werden zwei Regelbedingungen erzeugt. Diese Bedingungen werden erfüllt, falls der Arbeitsspeicher von Drools einen bedingten Sequenzfluss enthält, dessen Beschriftung durch den regulären Ausdruck identifiziert wird. Da bedingte Sequenzflüsse in das Ergebnis einfließen, wird ein entsprechender Eintrag in der Konklusion generiert.



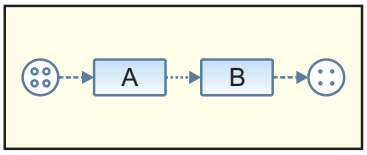
PPML-Diagramm

```
when
  // Bedingung 1
  modelElementNameText1 : ModelElementNameText(
      originalText matches ".*Begleitpapiere.*nicht.*")
  // Bedingung 2
  conditionalSequenceConnector1 :
      ConditionalSequenceConnector(name == modelElementNameText1)

then
  Set<RefBaseObject> pattern = new HashSet<RefBaseObject>();
  pattern.add(conditionalSequenceConnector1);                    (...)
```

Abbildung 6.9.: Übersetzung von bedingten Sequenzflüssen

6.2.1.6. Flexible Sequenzflüsse

Flexible Sequenzflüsse (siehe Abschnitt 5.2.5.3) werden in eine einzige Regelbedingung übersetzt, die den Aufruf der Hilfsmethode `matchFlexibleSequenceConnector` auswertet (siehe Abbildung 6.10). Diese Hilfsmethode wertet einen flexiblen Sequenzfluss bezüglich zweier Flussobjekte (F_1, F_2) aus und hat drei Parameter. Der erste Parameter ist eine Referenz auf den flexiblen Sequenzfluss innerhalb des zu transformierenden PPML-Modells. Eine Alternative wäre gewesen, die minimale und maximale Anzahl von Tasks und Ereignissen, die zwischen F_1 und F_2 liegen dürfen, der Methode direkt zu übergeben. Bei den beiden anderen Parametern handelt es sich um die jeweiligen Flussobjekte.



PPML-Diagramm

```
when
  // Bedingung 1
  modelElementNameText1 :
      ModelElementNameText(originalText matches "A")
  // Bedingung 2
  modelElementNameText2 :
      ModelElementNameText(originalText matches "B")
  // Bedingung 3
  task1 : Task(name == modelElementNameText1)
  // Bedingung 4
  task2 : Task(name == modelElementNameText2)
  // Bedingung 5
  eval(searcherTool.matchFlexibleSequenceConnector(mri, task1, task2));

                                                                 (...)
```

Abbildung 6.10.: Übersetzung von flexiblen Sequenzflüssen

Wie in Abschnitt 5.2.5.3 beschrieben, hängt die Semantik eines flexiblen Sequenzflusses vom Wert seiner Attribute ab. Dürfen zwischen F_1 und F_2 beliebig viele Tasks und Ereignisse liegen, reicht zur Auswertung ein Algorithmus, der durch Traversierung des BPMN-Modells überprüft, ob ein Pfad zwischen F_1 und F_2 in Kontrollflussrichtung existiert, wobei Zyklen berücksichtigt werden. Da bei der Überprüfung eines BPMN-Modells unter Umständen mehrere flexible Sequenzflüsse ausgewertet werden müssen, wird vor der eigentlichen Überprüfung mithilfe des *Algorithmus von Floyd und Warshall* [Flo62, War62] eine Erreichbarkeitsmatrix berechnet, die für jede Kombination aus Flussobjekten des BPMN-Modells speichert, ob diese durch einen Pfad miteinander verbunden sind. Zur Auswertung eines flexiblen Sequenzflusses wird der entsprechende Eintrag in dieser Matrix ausgelesen.

Muss bei der Auswertung dagegen eine Zählung der zwischen F_1 und F_2 liegenden Tasks und Ereignisse durchgeführt werden, kann wie in Abschnitt 5.2.5.3 erläutert zwischen einem struktur- und einer laufzeitbasierten Verfahren gewählt werden. Auch beim strukturbasierten Verfahren wird vor der eigentlichen Überprüfung eine Erreichbarkeitsmatrix berechnet. Zu diesem Zweck wurde ein entsprechender Algorithmus entwickelt. Jeder Eintrag dieser Matrix enthält eine Liste, die für jeden Pfad zwischen F_1 und F_2 ein Paar aus numerischen Werten enthält, wobei Zyklen nicht berücksichtigt werden. Die Elemente eines Paars stehen für die Anzahl der Tasks und Ereignisse, die entlang dieses Pfades auftreten. Bei der Auswertung eines flexiblen Sequenzflusses wird überprüft, ob einer dieser Pfade die Kriterien des flexiblen Sequenzflusses erfüllt.

Sollen bei der Zählung der zwischen F_1 und F_2 liegenden Tasks und Ereignisse Zyklen berücksichtigt werden, müssen im ungünstigsten Fall alle Pfade zwischen F_1 und F_2 berechnet werden. Aufgrund von Verzweigungen innerhalb des BPMN-Modells kann diese Berechnung mit sehr hohem Aufwand verbunden sein. Zur Bewältigung dieser Aufgabe führt das laufzeitbasierte Verfahren eine Modellprüfung durch. Zu diesem Zweck muss das BPMN-Modell in eine Systembeschreibung transformiert werden. Zur Auswertung des flexiblen Sequenzflusses wird die Systembeschreibung so erzeugt, dass deren Verifikation erfolgreich ist, sobald eine Folge der zwischen F_1 und F_2 ausgeführten Tasks und Ereignisse die Kriterien des flexiblen Sequenzflusses erfüllt. Da die Transformation eines BPMN-Modells in eine Systembeschreibung auch für die Auswertung bestimmter musterbasierter Bedingungen erforderlich ist, werden die entwickelten Transformationsalgorithmen ausführlich in Kapitel 7 vorgestellt. Im Gegensatz zum strukturbasierten Verfahren, das die statische Struktur des BPMN-Modells analysiert, wird bei einer Modellprüfung das Laufzeitverhalten des Modells untersucht, was im Vergleich zum strukturbasierten Verfahren zu unterschiedlichen Ergebnissen führen kann (z. B. aufgrund parallel ausführbarer Pfade). Aufgrund dieses Aufwands sollte hinsichtlich der Auswertungsdauer musterbasierter Bedingungen mit der Verwendung flexibler Sequenzflüsse, zu deren Auswertung dieses Verfahren benötigt wird, sparsam umgegangen werden.

6.2.1.7. Erzeugung der Konklusion von Drools-Regeln

Nachdem alle Elemente eines PPML-Modells in Regelbedingungen der Prämisse einer Drools-Regel übersetzt sind, wird die Konklusion der Regel erzeugt. Listing 6.6 zeigt die Konklusion der Drools-Regel, die mit dem strukturellen Muster in Abbildung 6.2 korrespondiert. Innerhalb der Konklusion werden zunächst zwei Mengen erzeugt, die zur Auswertung bestimmter musterbasierter Bedingungen benötigt werden: `incomingScopeObjects` und `outgoingScopeObjects`. Darüber hinaus werden Anweisungen erzeugt, die bei der Ausführung der Konklusion bewirken, dass die Ein- und Ausgangsobjekte der gefundenen Instanz des strukturellen Musters in die Menge `incomingScopeObjects` bzw. `outgoingScopeObjects` eingefügt werden. In Listing 6.6 wird beispielsweise jeweils ein Flussobjekt eingefügt (`task1` und `task2`). Die Entscheidung, ob ein Flussobjekt, das innerhalb einer Bedingung der Prämisse einer Drools-Regel an eine Variable gebunden wird, einer dieser Mengen zugeordnet werden muss, wird bereits bei der Übersetzung generischer Flussobjekte gefällt, indem überprüft wird, ob das zu übersetzende generische Flussobjekt mit einem ein- oder ausgehenden Musterkonnektor verbunden ist. Des Weiteren werden innerhalb der Konklusion sowohl eine dritte Menge namens `pattern` als auch Anweisungen erzeugt, die bei der Ausführung der Konklusion bewirken, dass die Elementen der Instanz des strukturellen Musters in diese Menge eingefügt werden. Schließlich wird bei Ausführung der Konklusion eine Rückrufmethode aufgerufen (`patternFound`), der eine Referenz auf das strukturelle Muster und eine Instanz der Klasse `PatternInstance`, welche die erzeugten Mengen kapselt, übergeben werden. Auf diese Weise kann Drools der aufrufenden Komponente mitteilen, dass ein Muster gefunden wurde und die zur Weiterverarbeitung benötigten Daten zurückliefern.

```
1   (...)
2
3   then
4     Set<RefBaseObject> incomingScopeObjects = new HashSet<RefBaseObject>();
5     incomingScopeObjects.add(task1);
6
7     Set<RefBaseObject> outgoingScopeObjects = new HashSet<RefBaseObject>();
8     outgoingScopeObjects.add(task2);
9
10    Set<RefBaseObject> pattern = new HashSet<RefBaseObject>();
11    pattern.add(task1);
12    pattern.add(task2);
13    pattern.add(sequenceConnector1);
14
15    patternFoundListener.patternFound(mri, new PatternInstance(
16      pattern, incomingScopeObjects, outgoingScopeObjects));
17
18  end
```

Listing 6.6: Konklusion einer Drools-Regel

6.2.2. Ablauf der musterbasierten Suche

Bei der Suche nach Instanzen struktureller Muster auf Basis von Drools werden zunächst die korrespondierenden Regeln der jeweiligen strukturellen Muster deserialisiert und in die Regelbasis von Drools eingefügt. Des Weiteren werden eine Instanz (`patternFoundListener`) der Klasse, welche die in der Konklusion von Regeln aufgerufene Rückrufmethode exponiert, und eine Instanz (`searcherTool`) der Klasse, welche Hilfsmethoden enthält, die innerhalb von Regelbedingungen aufgerufen werden, bei Drools registriert. Anschließend werden die Elemente des BPMN-Modells als Fakten in den Arbeitsspeicher von Drools eingefügt. Schließlich werden alle Regeln aktiviert und damit die Suche nach Instanzen struktureller Muster innerhalb des BPMN-Modells angestoßen.

6.3. Verwendung von Techniken aus dem Bereich des semantischen Webs

Einen dritten Werkzeugtyp, der zur Suche nach Instanzen struktureller Muster verwendet werden kann, stellen Reasoner in Verbindung mit einer vom Reasoner unterstützten Sprache zur Abfrage von Ontologien dar. Die musterbasierte Suche kann mithilfe eines Reasoners bewerkstelligt werden, indem PPML-Modelle in entsprechende Abfragen transformiert werden, beispielsweise in SPARQL-Abfragen oder konjunktive Anfragen (siehe Abschnitt 2.3.3.3). Da Reasoner auf Grundlage von Ontologien arbeiten, muss das zu durchsuchende BPMN-Modell darüber hinaus in eine Ontologie transformiert werden. In dieser Arbeit wurde die Eignung eines Reasoners in Verbindung mit konjunktiven Anfragen zur musterbasierten Suche überprüft. Ziel dieser Untersuchung war zum einen die Frage, ob es überhaupt möglich ist, Instanzen struktureller Muster mithilfe semantischer Techniken zu suchen. Zum anderen sollte durch die Entwicklung eines weiteren Ansatzes zur musterbasierten Suche eine Vergleichsmöglichkeit bezüglich des auf Drools basierenden Ansatzes geschaffen werden. Die Ergebnisse dieser Untersuchung werden im Folgenden zusammengefasst.

6.3.1. Transformation von BPMN-Modellen in OWL-Ontologien

Der Einsatz semantischer Techniken zur musterbasierten Suche setzt voraus, dass das zu durchsuchende BPMN-Modell in Form einer Ontologie auf Grundlage des vom verwendeten Reasoner benötigen Objektmodells (z. B. OWL API) vorliegt, die im Folgenden als *Suchontologie* bezeichnet wird. Zu diesem Zweck muss eine entsprechende Transformation durchgeführt werden, wozu eine weitere Ontologie benötigt wird, die eine formale Repräsentation von BPMN-Diagrammen ermöglicht und mit dem BPMN-Metamodell vergleichbar ist. Eine derartige Ontologie auf Basis von OWL wird in [GRS08] vorgestellt. Allerdings ist diese Ontologie sehr komplex, da sie fast alle BPMN-Modellierungskonstrukte und -regeln abbildet. Der Einfachheit halber wurde daher in dieser Arbeit eine

BPMN-Ontologie auf Basis von OWL entwickelt, die sich auf die Teilmenge der vom Process Composer unterstützten Modellierungskonstrukte und deren Eigenschaften, auf die sich strukturelle Muster beziehen können, beschränkt. Einen weiteren Vorteil dieser Beschränkung stellt die erhöhte Leistung von Reasoning-Vorgängen aufgrund der verminderten Anzahl von Axiomen dar. Die innerhalb dieser Ontologie spezifizierten OWL-Klassen sind in Abbildung 6.11 dargestellt.

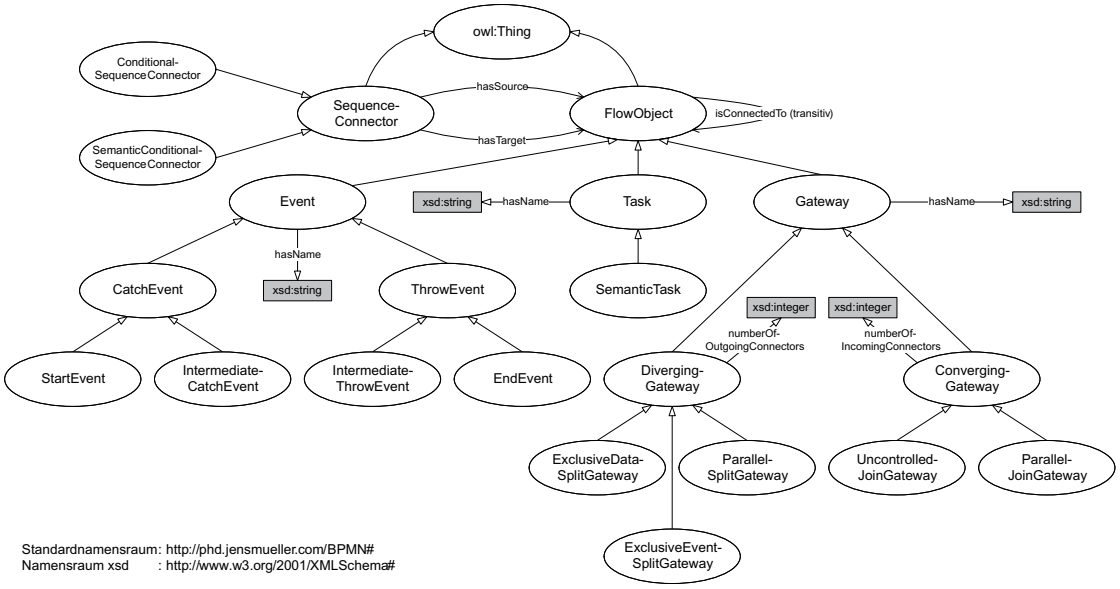


Abbildung 6.11.: BPMN-Ontologie

Bei der Transformation eines BPMN-Modells in eine Suchontologie wird für jedes Modellelement ein Individuum auf Grundlage der entsprechenden OWL-Klasse innerhalb der BPMN-Ontologie erzeugt. Die URI dieses Individuums basiert auf dem Ressourcenbezeichner des zu übersetzenden Modellelements, der im Englischen als *Model Element Resource Identifier* (MRI) bezeichnet wird, so dass es möglich ist, anhand des Individuums das korrespondierende Modellelement zu ermitteln, was zur Auswertung des Resultats der musterbasierten Suche erforderlich ist. Handelt es sich bei dem zu übersetzenden Modellelement um einen Task, wird dem erzeugten Individuum über die konkrete Rolle `hasName` der Name des Tasks zugewiesen. Die Übersetzung der zugeordneten Semantik eines Tasks gestaltet sich einfach, da für jede OWL-Klasse, die dem Task zugeordnet ist, das erzeugte Individuum lediglich als Instanz dieser Klasse deklariert wird. Da die Ontologie, die diese Klasse enthält, als Modell vorliegt, müssen dessen Elemente zuvor in einen entsprechenden Teil der Suchontologie auf Basis der vom Reasoner benötigten Repräsentation überführt werden (z. B. mithilfe des in Abschnitt 4.2.2 beschriebenen Algorithmus). Handelt es sich bei dem zu übersetzenden Modellelement um einen Sequenzfluss, wird das erzeugte Individuum über die abstrakten Rollen `hasSource` und `hasTarget` mit den Individuen verknüpft, die mit den Flussobjekten innerhalb des BPMN-Modells korrespondieren, die der zu übersetzende Sequenzfluss miteinander ver-

bindet. Darüber hinaus werden diese Individuen auch untereinander durch die transitive abstrakte Rolle `isConnectedTo` verknüpft, die zur Auswertung flexibler Sequenzflüsse im Rahmen der musterbasierten Suche benötigt wird. Im Fall von Sequenzflüssen, die mit demselben generischen Gateway verbunden sind, müssen die korrespondierenden Individuen explizit als paarweise verschieden deklariert werden (`owl:AllDifferent`), um die sogenannte *Unique Names Assumption* [49] zu erzwingen.

6.3.2. Transformation von PPML-Modellen in konjunktive Anfragen

Zur Suche nach Instanzen struktureller Muster mithilfe eines Reasoners wird das jeweilige PPML-Modell in eine konjunktive Anfrage transformiert. Listing 6.7 zeigt eine konjunktive Anfrage, die mit dem strukturellen Muster in Abbildung 6.2 korrespondiert. Da es für konjunktive Anfragen keinen Standard und damit keine normative Syntax gibt, basiert die Darstellung dieser Anfrage auf der in der SWRL-Spezifikation (siehe Abschnitt 2.3.3.2) verwendeten Syntax, die auch in nachfolgenden Beispielen verwendet wird. Die Anfrage besteht aus mehreren Atomen, die durch Konjunktionen ($\wedge$) miteinander verknüpft sind. Atome der Form *C(x)* (z. B. `bpmn:Task(?task1)`) sind wahr, falls die Variable x mit einem Individuum der Klasse C belegt werden kann. Atome der Form *R(x, y)* (z. B. `bpmn:hasName(?task1, ?task1Name)`) sind wahr, falls x und y mit Individuen belegt werden können, die über die Rolle R miteinander verknüpft sind. Bei der Übersetzung werden für jedes gesuchte Modellelement ein oder mehrere Atome erzeugt. Findet der Reasoner für alle Variablen eine Belegung mit Individuen, durch die alle Atome erfüllt sind, wird diese Belegung der Ergebnismenge hinzugefügt. Für die musterbasierte Suche bedeutet dies, dass das zu durchsuchende BPMN-Modell eine Instanz des korrespondierenden strukturellen Musters enthält.

```
1  bpmn:Task(?task1) ∧
2  bpmn:hasName(?task1, ?task1Name) ∧
3  bpmn:Task(?task2) ∧
4  bpmn:hasName(?task2, ?task2Name) ∧
5
6  bpmn:SequenceConnector(?sequenceConnector1) ∧
7  bpmn:hasSource(?sequenceConnector1, ?task1) ∧
8  bpmn:hasTarget(?sequenceConnector1, ?task2) ∧
9
10 kaon2:ifTrue("matchRegularExpression", "A", ?task1Name) ∧
11 kaon2:ifTrue("matchRegularExpression", "B", ?task2Name)
```

Listing 6.7: Konjunktive Anfrage

Konjunktive Anfragen werden von Pellet und dem ebenfalls auf Java basierenden Reasoner KAON2 unterstützt. Eine konjunktive Anfrage wird beispielsweise mithilfe der Programmierschnittstelle von KAON2 wie der Rumpf einer SWRL-Regel erzeugt, wobei alle Sprachkonstrukte von SWRL verwendet werden können. Da diese Sprachkonstrukte zur Übersetzung bestimmter Elemente von PPML-Modellen nicht ausreichen, ist

die Unterstützung benutzerdefinierter Funktionen innerhalb konjunktiver Anfragen eine zwingende Voraussetzung, die von KAON2 erfüllt wird. Zu diesem Zweck stellt KAON2 entsprechende *Built-ins* zur Verfügung, mit deren Hilfe Methoden von Java-Objekten aufgerufen werden können, die zuvor beim Reasoner registriert wurden. Der ebenfalls im Rahmen dieser Arbeit verwendete Reasoner Pellet bietet keine derartige Möglichkeit und konnte daher nicht zur musterbasierten Suche eingesetzt werden.

Zur Verwaltung von OWL-Ontologien und konjunktiven Anfragen bietet KAON2 eine entsprechende Programmierschnittstelle, die allerdings keine Implementierung der OWL API darstellt. Bei der Transformation eines PPML-Modells in eine konjunktive Anfrage muss ein Objektmodell mithilfe dieser Programmierschnittstelle erzeugt werden. Wie beim Drools-Ansatz muss dabei jedes Element des Modells in einen entsprechenden Teil des Objektmodells übersetzt werden. Auch in diesem Fall werden die bei der Übersetzung von Modellelementen erzeugten Teile des resultierenden Objektmodells innerhalb einer entsprechenden Datenstruktur abgelegt (siehe Abbildung 6.12). Nach Abschluss aller Übersetzungen wird zunächst mithilfe der Programmierschnittstelle aus der internen Repräsentation ein `QueryDefinition`-Objekt erzeugt (1). Zwar können Instanzen dieser Klasse aus technischen Gründen nicht serialisiert werden, allerdings erlaubt die Programmierschnittstelle eine Abbildung von Regeln auf Zeichenketten, denen eine proprietäre Syntax zugrunde liegt (2). Diese Zeichenkette wird schließlich in Form einer Datei serialisiert (3).

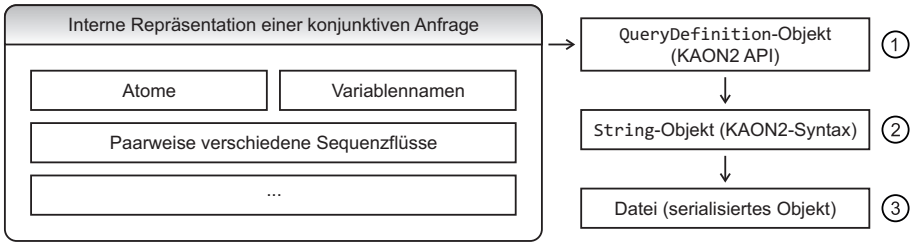


Abbildung 6.12.: Interne Repräsentation und Erzeugung einer konjunktiven Anfrage

Die Erzeugung der internen Repräsentation einer konjunktiven Anfrage ähnelt vom Ablauf her der Erzeugung der internen Repräsentation von Drools-Regeln (siehe Listing 6.5). Im Folgenden wird für die wichtigsten PPML-Modellierungskonstrukte beschrieben, wie deren Instanzen in entsprechende Teile des resultierenden `QueryDefinition`-Objekts auf Grundlage der internen Repräsentation übersetzt werden, wobei zur Darstellung die bereits erwähnte SWRL-Syntax verwendet wird.

6.3.2.1. Generische Tasks

Abbildung 6.13 zeigt Ausschnitte zweier PPML-Diagramme M_1 und M_2 (vgl. Abbildung 6.6). Der in M_1 abgebildete generische Task mit gesetztem regulärem Ausdruck wird in drei Atome übersetzt. Die ersten beiden Atome fordern das Vorhandensein eines

Individuums der OWL-Klasse `Task`, das über die konkrete Rolle `hasName` mit einem Literal vom Typ `xsd:string` verbunden ist. Findet der Reasoner eine passende Belegung, bindet er die die entsprechenden Individuen an die Variablen `task1` und `task1Name`. Zur Auswertung des regulären Ausdrucks wird mithilfe des Built-ins `kaon2:ifTrue` die benutzerdefinierte Hilfsfunktion `matchRegularExpression` aufgerufen. Zwar können reguläre Ausdrücke auch mit dem Built-in `swrlb:matches` ausgewertet werden, allerdings muss in diesem Fall eine andere Syntax [50] verwendet werden, die sich von der PPML-Syntax für reguläre Ausdrücke unterscheidet.

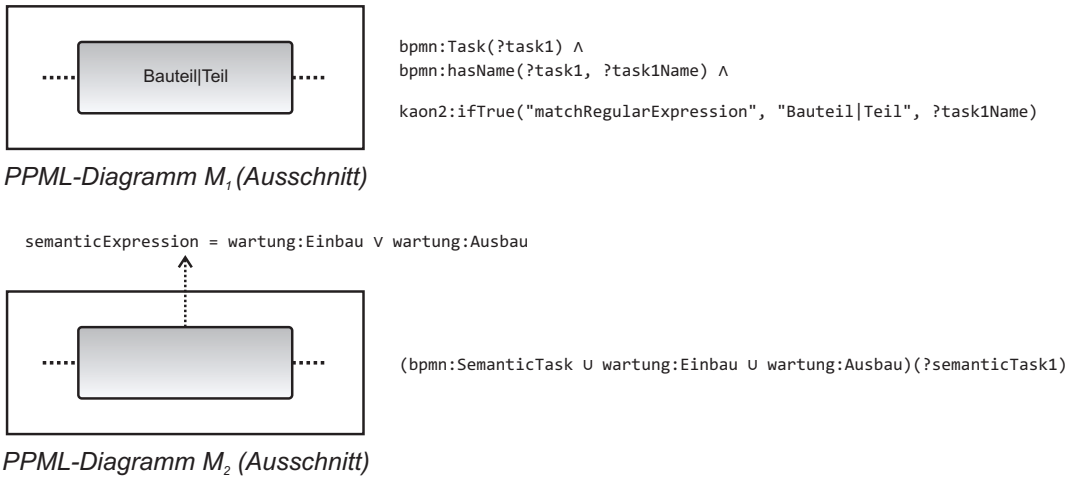
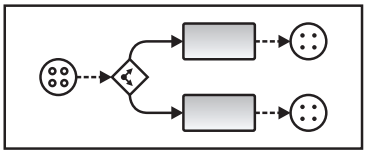


Abbildung 6.13.: Übersetzung generischer Tasks (Konjunktive Anfrage)

Zur Übersetzung des in M_2 abgebildeten generischen Tasks mit gesetztem semantischem Ausdruck reicht die Erzeugung eines einzelnen Atoms. Da die Individuen innerhalb der Suchontologie, die mit den Tasks innerhalb des zu durchsuchenden BPMN-Modells korrespondieren, als Unterklasse der dem Task zugeordneten OWL-Klassen deklariert sind, kann der semantische Ausdruck direkt durch ein Atom der Form *C(x)* ausgedrückt werden. Dieses Atom fordert das Vorhandensein eines Individuums, das sowohl eine Instanz der Klasse `SemanticTask` als auch eine Instanz der beiden innerhalb des semantischen Ausdrucks referenzierten Klassen der Domänenontologie ist. Die innerhalb des semantischen Ausdrucks verwendeten aussagenlogischen Junktoren ∧ und ∨ werden dabei durch die mengentheoretischen Operatoren ∩ bzw. ∪ ersetzt, deren Semantik den OWL-Sprachkonstrukten `owl:intersectionOf` bzw. `owl:unionOf` entspricht. Die Übersetzung des logischen Operators ¬ innerhalb eines semantischen Ausdrucks stellt jedoch ein Problem dar, da dessen Auswertung von der Geschlossenen-Welt-Annahme ausgeht (siehe Abschnitt 5.2.1), wohingegen OWL auf der Offenen-Welt-Annahme basiert (`owl:complementOf`). Allerdings kann dieses Problem mithilfe von KAON2 umgangen werden, da KAON2 auch die Auswertung negierter Atome auf Grundlage der Geschlossenen-Welt-Annahme unterstützt (KAON2-Schnittstelle: `DefaultNegation`).

6.3.2.2. Sequenzflüsse und flexible Sequenzflüsse

Die Steuerung der Übersetzung von Sequenzflüssen in den entsprechenden Teil einer konjunktiven Anfrage entspricht der Steuerung beim Drools-Ansatz (siehe Abschnitt 6.2.1.4). Abbildung 6.14 zeigt ein PPML-Diagramm und die korrespondierende konjunktive Anfrage (vgl. Abbildung 6.8). Bei der Übersetzung von Sequenzflüssen werden deren Attribute (`source`, `target`) in Atome der Form *R(x, y)* übersetzt, wobei sich *R* auf die abstrakten Rollen `hasSource` und `hasTarget` bezieht. Bei der Übersetzung von Sequenzflüssen, die im PPML-Modell mit demselben generischen Gateway verbunden sind, muss darüber hinaus sichergestellt werden, dass die Variablen der korrespondierenden Atome nur mit paarweise verschiedenen Individuen belegt werden. Andernfalls könnte das Ergebnis der musterbasierten Suche fehlerhaft sein. Um dies zu gewährleisten, wird jeweils ein Atom der Form *differentFrom(x, y)* erzeugt.



PPML-Diagramm

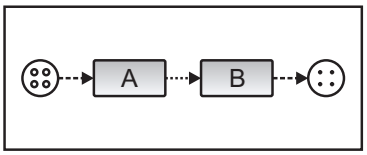
```
bpmn:Task(?task1_o) ∧
bpmn:Task(?task2_o) ∧

bpmn:DivergingGateway(?divergingGateway1_i) ∧

bpmn:SequenceConnector(?sequenceConnector2) ∧
bpmn:hasSource(?sequenceConnector2, ?divergingGateway1_i) ∧
bpmn:hasTarget(?sequenceConnector2, ?task1_o) ∧
bpmn:SequenceConnector(?sequenceConnector3) ∧
bpmn:hasSource(?sequenceConnector3, ?divergingGateway1_i) ∧
bpmn:hasTarget(?sequenceConnector3, ?task2_o) ∧

differentFrom(?sequenceConnector2, ?sequenceConnector3)
```

Abbildung 6.14.: Übersetzung von Sequenzflüssen (Konjunktive Anfrage)

Flexible Sequenzflüsse werden in ein einzelnes Atom übersetzt (siehe Abbildung 6.15). Dürfen zwischen den Flussobjekten, deren Anordnung mithilfe eines flexiblen Sequenzflusses ausgewertet werden soll, beliebig viele Tasks und Ereignisse liegen, kann die Auswertung im Gegensatz zum Drools-Ansatz ohne Verwendung einer benutzerdefinierten Funktion durchgeführt werden. In diesem Fall muss lediglich überprüft werden, ob die Individuen, die mit den Flussobjekten korrespondieren, über die transitive abstrakte Rolle `isConnectedTo` miteinander verknüpft sind. Aufgrund der Transitivität dieser Rolle kann mithilfe des Reasoners überprüft werden, ob es einen Pfad zwischen diesen Flussobjekten gibt. Muss bei der Auswertung dagegen eine Zählung der zwischen den Flussobjekten liegenden Tasks und Ereignisse durchgeführt werden, wird zur Auswertung des flexiblen Sequenzflusses mithilfe des Built-ins `kaon2:ifTrue` die benutzerdefinierte Funktion `matchFlexibleSequenceConnector` aufgerufen (siehe Abschnitt 6.2.1.6).

6.3.3. Bestimmung von Ein- und Ausgangsobjekten

Wie beim Drools-Ansatz müssen die Ein- und Ausgangsobjekte der Instanzen eines strukturellen Musters während der Übersetzung in eine konjunktive Anfrage bestimmt und



PPML-Diagramm

```
bpmn:Task(?task1_i) ∧
bpmn:hasName(?task1_i, ?task1Name) ∧
bpmn:Task(?task2_o) ∧
bpmn:hasName(?task2_o, ?task2Name) ∧

bpmn:isConnectedTo(?task1_i, ?task2_o) ∧

kaon2:ifTrue("matchRegularExpression", "A", ?task1Name) ∧
kaon2:ifTrue("matchRegularExpression", "B", ?task2Name)
```

Abbildung 6.15.: Übersetzung von flexiblen Sequenzflüssen (Konjunktive Anfrage)

das Ergebnis dieser Bestimmung für nachfolgende Schritte festgehalten werden. Zu diesem Zweck wird jeder Variablen, die mit einem generischen Flussobjekt innerhalb des strukturellen Musters korrespondiert, das mit einem ein- oder ausgehenden Musterkonnektor verbunden ist, das Suffix `_i` bzw. `_o` angehängt. Sofern eine Variable mit einem generischen Flussobjekt korrespondiert, das sowohl mit einem ein- als auch einen ausgehenden Musterkonnektor verbunden ist, wird das Suffix `_io` an die Variable angehängt. Da bei der Auswertung eines Tupels, das als Teil des Resultats einer konjunktiven Anfrage zurückgeliefert wird, die Namen der Variablen, die mit den Individuen des Tupels korrespondieren, abgefragt werden können, kann auch in dieser Phase die Klassifikation eines Individuums (z. B. `task1_i`) als Ein- oder Ausgangsobjekts erfolgen.

6.3.4. Ablauf der musterbasierten Suche

Bei der musterbasierten Suche mithilfe konjunktiver Anfragen wird zunächst die Suchontologie erzeugt. Dabei werden die BPMN-Ontologie und die verwendeten Domänenontologien importiert. Des Weiteren werden die benutzerdefinierten Funktionen `matchFlexibleSequenceConnector` und `matchRegularExpression`, die innerhalb konjunktiver Anfragen in Form von Built-ins aufgerufen werden können, beim Reasoner registriert. Anschließend wird für jedes der jeweiligen strukturellen Muster die entsprechende konjunktive Anfrage deserialisiert und ausgeführt. Schließlich werden die Elemente des überprüften BPMN-Modells, die mit den Individuen eines Ergebnistupels korrespondieren, bestimmt und als Ein- oder Ausgangsobjekt klassifiziert.

6.4. Implementierung: Mustertransformatoren und Mustersucher

Die in den vorangehenden Abschnitten vorgestellten Suchverfahren auf Grundlage eines regelbasierten Systems und auf Grundlage eines Reasoners wurden im Rahmen dieser Arbeit vollständig in Form von Eclipse Plug-ins implementiert, die in Abbildung 6.16 dargestellt sind. Zunächst wurde die Logik der Suchverfahren jeweils auf einen *Mustertransformator* und einen *Mustersucher* aufgeteilt. Die Aufgabe eines Mustertransformators ist die Transformation eines PPML-Modells in die Eingabedatenrepräsentation des

verwendeten Werkzeugs zur Durchführung der Suche nach Instanzen struktureller Muster und die Serialisierung dieser Eingabedatenrepräsentation in eine Datei. Die Aufgabe eines Mustersuchers ist die Steuerung der musterbasierte Suche mithilfe des jeweiligen Werkzeugs auf Grundlage eines BPMN-Modells und der deserialisierten Eingabedatenrepräsentation. Die Aufteilung in Mustertransformatoren und Mustersucher ist auf die Annahme zurückzuführen, dass ein PPML nur einmal modelliert, aber dessen korrespondierende Eingabedatenrepräsentation vom jeweiligen Werkzeug im Rahmen der Auswertung musterbasierter Bedingungen mehrmals verwendet wird.

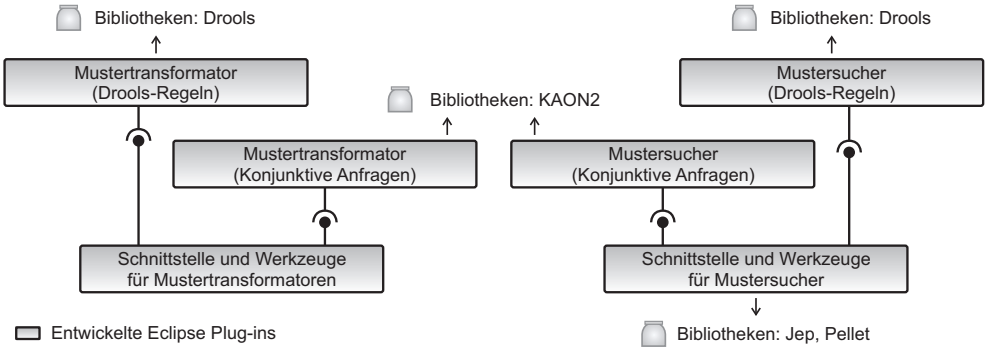


Abbildung 6.16.: Entwickelte Eclipse Plug-ins

Um einen einheitlichen Zugriff auf Mustertransformatoren und Mustersucher zu gewährleisten, wurden zwei Plug-ins entwickelt, die entsprechende Schnittstellen enthalten. Diese Schnittstellen müssen von Mustertransformatoren und Mustersucher implementiert werden. Darüber hinaus stellen beide Plug-ins einen sogenannten *Extension Point* zur Verfügung, über den sich Mustertransformatoren und Mustersucher innerhalb von Eclipse als solche registrieren können.

6.5. Stand der Wissenschaft und Technik

Die Verwendung eines regelbasierten Systems oder die Verwendung konjunktiver Anfragen zur Suche nach Instanzen struktureller Muster innerhalb von Geschäftsprozessmodellen wurde bisher noch nicht erforscht und wird in der vorliegenden Arbeit zum ersten Mal vorgeschlagen. Dennoch wurden bereits vergleichbare Ansätze vorgeschlagen, die sich mit der Identifikation von Unterstrukturen innerhalb von Strukturen auf Grundlage eines vorgegebenen Musters befassen.

Auf dem Gebiet der Graphentheorie existiert eine Vielzahl von Algorithmen [RC77, Gat79], die berechnen, ob ein Graph G zu einem oder mehreren Untergraphen eines Graphs H isomorph ist. Die zugehörige Aufgabenstellung wird im Englischen als *Subgraph Isomorphism Problem* bezeichnet. Eine Anwendung hierfür ist die Suche nach molekularen Unterstrukturen innerhalb von Molekülen, die auf Grundlage der textuellen Sprache *Simplified Molecular Input Line Entry System* [51] spezifiziert wurden. Zu

diesem Zweck muss wiederum ein entsprechendes Muster auf Grundlage der textuellen Sprache SMARTS [52] spezifiziert werden. Auch wenn ein BPMN-Modell als gerichteter Graph aufgefasst wird, können derartige Verfahren dennoch nicht zur musterbasierten Suche verwendet werden, da es sich bei Instanzen struktureller Muster nicht zwangsläufig um Graphen handeln muss, beispielsweise falls die musterbasierte Suche aufgrund flexibler Sequenzflüsse fragmentierte Instanzen zu Tage fördert (siehe Abschnitt 5.2.5.3).

In Abschnitt 5.5 wurde bereits eine graphische Notation zur Abfrage von BPEL-Prozessen erwähnt: BP-QL [BEKM05, BEKM06, BEKM08]. In diesem Ansatz werden BPEL-Prozesse intern auf Grundlage von *Active XML* [53] repräsentiert. Zur Auswertung einer graphischen BP-QL-Abfrage wird diese in eine Repräsentation überführt, die an *XQuery* [W3C07], eine Sprache zur Abfrage von XML-Datenbanken, angelehnt ist und zur Abfrage von Dokumenten auf Grundlage von Active XML verwendet werden kann. Allerdings ist diese Vorgehensweise auf BPEL zurechtgeschnitten und kann nicht ohne Weiteres auf BPMN-Modelle übertragen werden.

Die ebenfalls in Abschnitt 5.5 erwähnten Ansätze zur Abfrage von BPMN-Modellen – BPMN-Q [Awa07] und die BPMN Visual Query Language [FT08, Fra08, FT09] – verfolgen unterschiedliche Auswertungsstrategien. Im BPMN-Q-Ansatz werden BPMN-Modelle innerhalb einer Datenbank auf Grundlage eines entsprechenden Schemas gespeichert. Zur Auswertung einer graphischen BPMN-Q-Abfrage wird ein speziell zu diesem Zweck entwickelter Algorithmus verwendet. Der auf der BPMN Visual Query Language basierende Ansatz kommt den in diesem Kapitel vorgestellten Verfahren zur Suche nach Instanzen struktureller Muster am nächsten, insbesondere dem musterbasierten Suchverfahren auf Grundlage konjunktiver Anfragen. Wie bereits in Abschnitt 4.5 beschrieben werden im Fall der BPMN Visual Query Language semantisch annotierte BPMN-Modelle in Form von OWL-Ontologien repräsentiert. Zur Auswertung einer graphischen Abfrage wird diese in eine entsprechende SPARQL-Abfrage transformiert, die im Anschluss von einem Reasoner verarbeitet wird. Auch in diesem Ansatz wird also eine deklarative Vorgehensweise verfolgt. Allerdings ist es nicht möglich, benutzerdefinierte Funktionen innerhalb von SPARQL-Abfragen zu verwenden. Deren Verwendung ist jedoch zur Auswertung bestimmter PPML-Modellierungskonstrukte unerlässlich.

Insgesamt erfüllt keiner der bestehenden Ansätze alle Anforderungen zur Suche nach Instanzen struktureller Muster, wenngleich eine gewisse Verwandtschaft zur Auswertung von Diagrammen auf Grundlage der BPMN Visual Query Language im Vergleich zur musterbasierten Suche mithilfe konjunktiver Anfragen besteht. Darüber hinaus zielt keiner der in diesem Abschnitt erwähnten Ansätze darauf ab, Geschäftsprozessmodelle auf die Erfüllung von Anforderungen zu überprüfen, die sich auf Anordnungen von Modellelementen mit bestimmter inhaltlicher Bedeutung beziehen. Aus diesem Grund bietet auch keiner dieser Ansätze die Möglichkeit, die Ein- und Ausgangsobjekte gefundener Instanzen struktureller Muster zu bestimmen. Diese Anforderung ist für die Auswertung musterbasierter Bedingungen jedoch eine zwingende Voraussetzung.

7. Auswertung musterbasierter Bedingungen an BPMN-Modelle

Zur Verifikation eines BPMN-Modells in Bezug auf einen Bedingungsausdruck, der mit dem Modell verknüpft ist, wurde im Rahmen der vorliegenden Arbeit eine dreistufige Vorgehensweise konzipiert. Nachdem in Kapitel 6 zwei Verfahren vorgestellt wurden, die ein BPMN-Modell im Zuge dieser Vorgehensweise zunächst nach Instanzen struktureller Muster durchsuchen (siehe Abschnitt 6.2 und 6.3), beschreibt dieses Kapitel, wie musterbasierte Bedingungen innerhalb eines Bedingungsausdrucks und der Ausdruck selbst auf Grundlage des Resultats der musterbasierten Suche ausgewertet werden. Bei der Auswertung musterbasierter Bedingungen wird zwischen existenziellen (siehe Abschnitt 5.3.1) und temporalen (siehe Abschnitt 5.3.2) Bedingungen unterschieden. Während zur Auswertung existenzieller Bedingungen relativ einfache Algorithmen ausreichen, ist zur Auswertung temporaler Bedingungen normalerweise eine Modellprüfung erforderlich, wobei im Rahmen dieser Arbeit der Modellprüfer *Spin* verwendet wurde. Die zur Durchführung einer Modellprüfung benötigten Transformationsschritte werden im weiteren Verlauf ausführlich beschrieben. Anhand des Prototyps wird schließlich gezeigt, wie der gesamte Ablauf der Verifikation eines BPMN-Modells vonstattengeht.

7.1. Auswertung existenzieller Bedingungen

Wie in Abschnitt 5.3 beschrieben, können mithilfe von PCML unäre und binäre Bedingungen modelliert werden. Unäre und binäre Bedingungen, die eine Forderung an die Existenz von Instanzen struktureller Muster innerhalb eines BPMN-Modells stellen, können mithilfe einfacher Algorithmen ausgewertet werden. Beispielsweise muss bei einer Existenz-Bedingung nach Abschluss der musterbasierten Suche lediglich überprüft werden, ob eine bestimmte Anzahl von Instanzen eines strukturellen Musters innerhalb des zu verifizierenden BPMN-Modells enthalten ist oder nicht. Abbildung 7.1 zeigt eine Übersicht der Konstrukte zur Modellierung existenzieller Bedingungen und den entsprechenden Algorithmus zu deren Auswertung in Form von Pseudocode. Innerhalb dieser Algorithmen wird die musterbasierte Suche durch Aufruf der Prozedur `Search` angestoßen, die als Eingabe ein strukturelles Muster benötigt und eine Menge aus Mengen, die wiederum Instanzen des strukturellen Musters enthalten, zurückgibt. Jeder dieser Algorithmen liefert ein boolesches Ergebnis zurück.

<table>
<tr><th>Modellierunskonstrukt</th><th>Eingabe</th><th>Auswertung</th></tr>
<tr><td colspan="2">Allen Algorithmen gemeinsame Schritte</td><td><pre>patternInstances ← Search(structuralPattern)
size ← |patternInstances|</pre></td></tr>
<tr><td>Abwesenheit</td><td rowspan="3"><pre>amount : int
structuralPattern : StructuralPattern</pre></td><td><pre>return size <= amount</pre></td></tr>
<tr><td>Exaktheit</td><td><pre>return size = amount</pre></td></tr>
<tr><td>Existenz</td><td><pre>return size >= amount</pre></td></tr>
<tr><td colspan="2">Allen Algorithmen gemeinsame Schritte</td><td><pre>patternInstances1 ← Search(structuralPattern1)
patternInstances2 ← Search(structuralPattern2)
size1 ← |patternInstances1|
size2 ← |patternInstances2|</pre></td></tr>
<tr><td>Erwiderte Existenz</td><td rowspan="3"><pre>structuralPattern1 : StructuralPattern
structuralPattern2 : StructuralPattern</pre></td><td><pre>result ← true

if size1 != 0 then result ← size2 != 0

return result</pre></td></tr>
<tr><td>Koexistenz</td><td><pre>result1 ← true
result2 ← true

if size1 != 0 then result1 ← size2 != 0
if size2 != 0 then result2 ← size1 != 0

return result1 and result2</pre></td></tr>
<tr><td>Negierte Koexistenz</td><td><pre>result1 ← true
result2 ← true

if size1 = 0 then result1 ← size2 == 0
if size2 = 0 then result2 ← size1 == 0

return result1 and result2</pre></td></tr>
</table>

Abbildung 7.1.: Auswertung existenzieller Bedingungen

7.2. Auswertung temporaler Bedingungen

Im Gegensatz zur Auswertung existenzieller Bedingungen kann es sehr aufwendig sein, die temporale Beziehung zwischen Instanzen zweier struktureller Muster zu überprüfen, da alle Pfade des BPMN-Modells berechnet und analysiert werden müssen. Zur Lösung derartiger Probleme bietet sich die Durchführung einer Modellprüfung [BK08] des BPMN-Modells an. Um eine Bedingung an die temporale Beziehung zwischen Instanzen struktureller Muster mithilfe eines Modellprüfers auszuwerten, muss auf der einen Seite das zu überprüfende BPMN-Modell in eine Systembeschreibung übersetzt werden, deren Repräsentation vom verwendeten Modellprüfer abhängt. Auf der anderen Seite muss die Bedingung in eine Spezifikation übersetzt werden. Aufgabe des Modellprüfers ist es, die Systembeschreibung auf Einhaltung dieser Spezifikation zu verifizieren, was im Kontext dieser Arbeit der Auswertung der Bedingung bezüglich des BPMN-Modells entspricht.

In dieser Arbeit wurde zur Auswertung temporaler Bedingungen der Modellprüfer *Spin* [Hol04, BA08] [54] verwendet. Spin ist ein Werkzeug zur Analyse der logischen Konsistenz nebenläufiger Systeme (z. B. Protokolle zur Datenkommunikation). Die Entwicklung von Spin reicht bis in das Jahr 1980 zurück und wird bis heute fortgesetzt (In dieser Arbeit wurde die Version 5.2.5 verwendet). Systembeschreibungen, die von Spin verifiziert

werden sollen, müssen in Form von Programmen auf Grundlage der Programmiersprache *Process Meta Language* (PROMELA) vorliegen. Folglich muss zur Durchführung einer Modellprüfung das zu verifizierende BPMN-Modell in ein PROMELA-Programm übersetzt werden. Auf diese Weise wird dem Modell eine formale Semantik zugewiesen.

Zur Erstellung einer Spezifikation bietet Spin zwei unterschiedliche Mechanismen. Zum einen können Bedingungen an den Zustand eines PROMELA-Programms direkt im Quelltext durch *Invarianten* ausgedrückt werden. Eine Invariante wird in Form einer *Zusicherung* (Assertion) [Flo67] mithilfe des Schlüsselworts `assert` in Verbindung mit einem Ausdruck spezifiziert. Stellt Spin während der Modellprüfung fest, dass eine Invariante verletzt wird, d. h. die Auswertung des zugehörigen Ausdrucks `false` ergibt, bricht die Modellprüfung mit einem Fehler ab. Zum anderen kann das geforderte Verhalten eines Programms mithilfe von LTL-Formeln (*Lineare temporale Logik*) spezifiziert werden. Zur Auswertung einer musterbasierten Bedingung wird daher eine entsprechende LTL-Formel erzeugt. Darüber hinaus wird die Generierung des PROMELA-Programms auf diese Formel abgestimmt. Sofern die Modellprüfung ergibt, dass das Programm die Formel in keinem Zustand verletzt, erfüllt das korrespondierende Geschäftsprozessmodell die Bedingung. In den zwei folgenden Abschnitten wird sowohl die Transformation von BPMN-Modellen in PROMELA-Modelle als auch die Transformation musterbasierter Bedingungen in LTL-Formeln ausführlich beschrieben.

7.2.1. Transformation von BPMN-Modellen in PROMELA-Programme

PROMELA ist eine einfache prozedurale Programmiersprache, deren Syntax an C [ISO99] angelehnt ist. PROMELA erlaubt die dynamische (d. h. zur Laufzeit) Erzeugung nebenläufiger Prozesse, die durch *Nachrichtenkanäle* synchron oder asynchron miteinander kommunizieren können. Darüber hinaus können globale oder lokale Variablen deklariert werden. Zur Verifikation eines PROMELA-Programms erzeugt Spin daraus ein C-Programm, das nach dessen Kompilierung die Durchführung einer Modellprüfung ermöglicht. Auf diese Weise wird für jede Systembeschreibung ein passgenauer Modellprüfer generiert, der sehr effizient ist. Zur Auswertung einer musterbasierten Bedingung muss das zu überprüfende BPMN-Modell in ein PROMELA-Programm übersetzt werden. Zu diesem Zweck wurde ein entsprechender Algorithmus entwickelt, der teilweise von den in [VF07] vorgestellten Konzepten inspiriert ist, allerdings nur die im Rahmen dieser Arbeit verwendeten Modellierungskonstrukte berücksichtigt. Tiefergehende Ansätze zur Formalisierung von BPMN werden in [WG08, DDO08] beschrieben.

Bei der Übersetzung eines BPMN-Modells wird ein PROMELA-Programm in Form einer Datei erzeugt. Dabei fließt jedes Element des Modells in das resultierende Programm ein. Zur Übersetzung wird das BPMN-Modell ausgehend von dessen Startereignis traversiert. Dieser Vorgang endet nach abgeschlossener Verarbeitung aller Endereignisse des Modells. Für jedes Flussobjekt wird eine *Prozessdeklaration* generiert, welche die Laufzeitsemantik dieses Modellelements gemäß der BPMN-Spezifikation nachbildet. Der von den Sequenzflüssen innerhalb des Modells vorgegebene Kontrollfluss wird durch den Versand

und Empfang von Nachrichten innerhalb von Prozessen implementiert. Zu diesem Zweck müssen entsprechende Nachrichtenkanäle deklariert werden. Im Folgenden wird für einen Großteil der vom Process Composer unterstützten BPMN-Modellierungskonstrukte erläutert, wie Modellelemente des entsprechenden Typs übersetzt werden. Nachrichtenflüsse werden bei der Übersetzung ignoriert, da PPML derzeit keine Möglichkeit bietet, struktureller Muster zu modellieren, die Nachrichtenflüsse identifizieren, was sich jedoch in einer zukünftigen Version ändern ließe.

7.2.1.1. Sequenzflüsse

Wie bereits erwähnt werden bei der Übersetzung eines BPMN-Modells in ein PROMELA-Programm Sequenzflüsse durch Nachrichtenkanäle, mithilfe derer Prozesse miteinander kommunizieren können, implementiert. PROMELA unterstützt zwei Arten von Nachrichtenkanälen: *Rendezvous-Kanäle* und *gepufferte Kanäle*. Rendezvous-Kanäle können keine Nachrichten zwischenspeichern und können daher nur zur synchronen Kommunikation verwenden werden. Dies bedeutet, dass eine Anweisung, die eine Nachricht über einen Rendezvous-Kanal versendet, nur ausgeführt werden kann, wenn ein anderer Prozess bereit ist, eine Nachricht von diesem Kanal in Empfang zu nehmen. Im Gegensatz dazu können gepufferte Kanäle eine bestimmte Anzahl von Nachrichten zwischenspeichern und eignen sich daher zur Modellierung asynchroner Kommunikation, allerdings erhöhen Sie die Komplexität von Systembeschreibungen. Zur Übersetzung von Sequenzflüssen reichen Rendezvous-Kanäle jedoch aus. Zu diesem Zweck wird am Anfang des PROMELA-Programms ein globales Array aus Rendezvous-Kanälen generiert, dessen Größe (im Beispiel 12) vom jeweiligen BPMN-Modell abhängt:

```
1 chan rendezvous_channels[12] = [0] of { bit }
```

Zur Verbesserung der Lesbarkeit des generierten Programmcodes werden zwei Makros verwendet (siehe Listing 7.1). Innerhalb von PROMELA-Programmen werden Makro-Aufrufe vor der Kompilierung ersetzt. Während das Makro `send` für eine Anweisung steht, die einen Wert (`token`) über einen Nachrichtenkanal (`channel`) versendet, steht das Makro `receive` für eine Anweisung, die einen Wert von einem Nachrichtenkanal (`channel`) in Empfang nimmt und ihn an eine Variable (`token`) bindet.

```
1 inline send(channel, token) {
2     channel ! token
3 }
4
5 inline receive(channel, token) {
6     channel ? token
7 }
```

Listing 7.1: PROMELA-Makros zum Versand und Empfang von Nachrichten

7.2.1.2. Tasks und Ereignisse

Für Tasks und Ereignisse innerhalb des zu übersetzenden BPMN-Modells wird mithilfe des Schlüsselworts `proctype` jeweils eine eigene Prozessdeklaration generiert. Das Schlüsselwort `active` definiert, dass ein auf dieser Deklaration basierender Prozess im Startzustand des PROMELA-Programms ausgeführt wird. Für Tasks und Zwischenereignisse werden normalerweise zwei Anweisungen erzeugt: eine Anweisung zum Empfang einer Nachricht (`receive`) von dem Nachrichtenkanal, der mit dem eingehenden Sequenzfluss des Modellelements korrespondiert und eine Anweisung zum Versand einer Nachricht (`send`) über den Kanal, der mit dem ausgehenden Sequenzfluss korrespondiert (siehe Listing 7.2). Der innerhalb des `receive`-Makros verwendete Unterstrich ist eine vordefinierte globale Variable, die zur Speicherung von Nachrichten verwendet werden kann, deren Inhalt für die weitere Programmausführung irrelevant ist, was in diesem Kontext der Fall ist, da der Empfang einer Nachricht (Bit mit Wert 1) lediglich zur Aktivierung des Prozesses dient und es nicht auf deren Inhalt ankommt.

```
1 active proctype Task⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2 {
3 end:
4     receive(rendezvous_channels[⟨ID⟩], _);
5     send(rendezvous_channels[⟨ID⟩], 1)
6 }
```

Listing 7.2: Übersetzung von Tasks (PROMELA)

Damit bei der Verifikation eines PROMELA-Programms keine Fehlermeldung erzeugt wird, müssen sich alle Prozesse des Programms letztendlich in einem gültigen Endzustand befinden. Ein Prozess erreicht einen Endzustand, sobald er das Ende der Prozessdeklaration erreicht. Allerdings kann es sein, dass der in Listing 7.2 dargestellte Prozess nie ausgeführt wird (z. B. aufgrund eines vorherigen exklusiven Gateways). In diesem Fall kommt die Ausführung des Prozesses beim Aufruf des `receive`-Makros zum Stillstand. Um auszudrücken, dass es sich in diesem Fall um einen gültigen Endzustand handelt, wird daher vor dem Aufruf eine Sprungmarke erzeugt (`end`). Anhand von Sprungmarken, die mit der Zeichenkette `end` beginnen, wird Spin mitgeteilt, dass sich der Prozess vor Ausführung der nachfolgenden Anweisung in einem Endzustand befindet. Handelt es sich bei dem zu übersetzenden Flussobjekt um ein Start- oder Endereignis entfällt der Aufruf des `receive`- bzw. `send`-Makros.

7.2.1.3. Exklusive Gateways

Bei der Übersetzung exklusiver Gateways wird wie bei Tasks und Ereignissen zunächst ein Aufruf des `receive`-Makros generiert. Gemäß der BPMN-Spezifikation aktiviert ein exklusiver Gateway genau eines der nachfolgenden Flussobjekte. Listing 7.3 stellt dar,

wie das Verhalten exklusiver Gateways mit zwei ausgehenden Sequenzflüssen mithilfe der `if`-Anweisung nachgebildet wird.

```
1  active proctype ExclusiveDataSplitGateway⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2  {
3  end:
4      receive(rendezvous_channels[⟨ID⟩], _);
5
6      if
7      :: send(rendezvous_channels[⟨ID⟩], 1)
8      :: send(rendezvous_channels[⟨ID⟩], 1)
9      fi
10 }
```

Listing 7.3: Übersetzung exklusiver Gateways (PROMELA)

Eine `if`-Anweisung wird von den Schlüsselwörtern `if` und `fi` umrahmt. Zwischen diesen Schlüsselwörtern stehen eine oder mehrere *Alternativen*. Eine Alternative wird mit zwei aufeinanderfolgenden Doppelpunkten eingeleitet, denen eine als *Guard* bezeichnete Anweisung folgt. Auf jede dieser Guard-Anweisungen können weitere Anweisungen folgen. Die `if`-Anweisung von PROMELA unterscheidet sich stark von vergleichbaren Konstrukten anderer Programmiersprachen. Die Entscheidung, welche der Guard-Anweisungen ausgeführt wird, hängt von ihrer *Ausführbarkeit* (Executability) ab. Eine Anweisung ist ausführbar, sofern deren Auswertung `true` ergibt. Die PROMELA-Spezifikation definiert für jedes Sprachkonstrukt, wann es ausführbar ist. Falls die Auswertung mehrerer Guard-Anweisungen `true` ergibt, wird im Rahmen einer Simulation eine dieser Alternativen per Zufall gewählt. Im Gegensatz dazu werden im Rahmen einer Verifikation alle Möglichkeiten in Betracht gezogen. Sofern keine Guard-Anweisung zu `true` ausgewertet werden kann, wird die Ausführung des Prozesses solange ausgesetzt, bis sich an diesem Zustand etwas ändert. Dieses Verhalten wird bei der Übersetzung exklusiver Gateways ausgenutzt, indem für jeden ausgehenden Sequenzfluss eine Guard-Anweisung erzeugt wird, die das `send`-Makro mit dem entsprechenden Nachrichtenkanal aufruft. Die Anweisung, durch die dieses Makro ersetzt wird (`channel ! token`), ist ausführbar, sofern ein anderer Prozess zum Empfang einer Nachricht von diesem Kanal bereit ist, was letztendlich immer der Fall ist, da alle Prozesse bei der Ausführung des Programms gestartet werden (Schlüsselwort `active`). Auf diese Weise werden bei der Verifikation alle Möglichkeiten zur Fortsetzung des Kontrollflusses berücksichtigt.

7.2.1.4. Zusammenführungen (exklusiver Pfade)

Die Übersetzung einer Zusammenführung exklusiver Pfade liefert ein ähnliches Ergebnis wie die Übersetzung von Tasks und Zwischenereignissen. Auch in diesem Fall werden zwei Anweisungen zum Empfang und Versand einer Nachricht generiert. Allerdings wird darüber hinaus eine Sprunganweisung (`goto`) erzeugt, um die Ausführung des Prozesses

nach dem Versand der Nachricht wieder mit der ersten Anweisung des Prozesses fortzusetzen (siehe Listing 7.4). Auch in diesem Fall wird die zur Markierung der Rücksprungadresse verwendete Sprungmarke mit dem Präfix `end` versehen, um einen gültigen Endzustand im Fall einer Aussetzung der Prozessausführung aufgrund des `receive`-Makros auszudrücken. Die auf diese Weise erzeugte Endlosschleife wird benötigt, da mithilfe von Zusammenführungen exklusiver Pfade Zyklen modelliert werden können, wodurch diese Zusammenführungen und alle nachfolgenden Flussobjekte mehrmals ausgeführt werden können. Aus diesem Grund werden bei der Übersetzung dieser Flussobjekte nach dem erstmaligen Auftreten einer Zusammenführung exklusiver Pfade ebenfalls Sprungmarken und entsprechende Sprunganweisungen generiert.

```
1  active proctype UncontrolledJoinGateway⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2  {
3  end_loop:
4      receive(rendezvous_channels[⟨ID⟩], _);
5      send(rendezvous_channels[⟨ID⟩], 1);
6
7      goto end_loop
8  }
```

Listing 7.4: Übersetzung von Zusammenführungen exklusiver Pfade (PROMELA)

7.2.1.5. Parallele Gateways

Gemäß der BPMN-Spezifikation aktiviert ein paralleler Gateway alle nachfolgenden Flussobjekte. Dieses Verhalten wird innerhalb des korrespondierenden PROMELA-Prozesses mithilfe des `parallel_split`-Makros nachgebildet (siehe Listing 7.5).

```
1  inline parallel_split(channels, channels_size)
2  {
3      byte i;
4
5      atomic
6      {
7          i = 0;
8
9          do
10         :: i == channels_size -> break
11         :: else -> send(channels[i], 1); i++
12         od
13     }
14 }
```

Listing 7.5: PROMELA-Makro für parallele Gateways

Das `parallel_split`-Makro benötigt zwei Parameter: ein Array aus Nachrichtenkanälen und dessen Größe. Das zentrale Element dieses Makros ist eine `do`-Schleife, die von

den Schlüsselwörtern `do` und `od` umrahmt wird. Wie bei einer `if`-Anweisung stehen zwischen diesen Schlüsselwörtern eine oder mehrere Alternativen. Der einzige Unterschied bezüglich der Semantik zwischen einer `if`-Anweisung und einer `do`-Schleife ist der, dass nach Ausführung einer Guard-Anweisung und eventuell darauffolgender Anweisungen die Auswertung der Alternativen von vorne beginnt. Ein Verlassen der `do`-Schleife kann mithilfe des Schlüsselworts `break` erreicht werden. Das Schlüsselwort `else` steht für eine vordefinierte bedingte Anweisung, die dafür bestimmt ist, als Guard-Anweisung verwendet zu werden und nur dann ausführbar ist, sofern alle anderen Guard-Anweisungen nicht ausführbar sind. Innerhalb der `do`-Schleife des Makros wird an jeden Kanal des als Parameter übergebenen Arrays eine Nachricht gesendet.

Listing 7.6 stellt dar, wie ein paralleler Gateway mit zwei ausgehenden Sequenzflüssen mithilfe des `parallel_split`-Makros übersetzt wird. Nach der Generierung eines Aufrufs des `receive`-Makros wird im Anschluss die Deklaration eines lokalen Arrays generiert, das die Nachrichtenkanäle beinhaltet, die mit den eingehenden Sequenzflüssen der nachfolgenden Flussobjekte korrespondieren. Schließlich wird ein Aufruf des `parallel_split`-Makros generiert.

```
1  active proctype ParallelSplitGateway⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2  {
3  end:
4      receive(rendezvous_channels[⟨ID⟩], _);
5
6      atomic
7      {
8          chan channels[2] = [0] of { bit };
9          channels[0] = rendezvous_channels[⟨ID⟩];
10         channels[1] = rendezvous_channels[⟨ID⟩];
11     }
12
13     parallel_split(channels, 2)
14 }
```

Listing 7.6: Übersetzung paralleler Gateways (PROMELA)

7.2.1.6. Zusammenführungen (paralleler Pfade)

In Bezug auf Ihre Übersetzung unterscheiden sich Zusammenführungen paralleler Pfade von Zusammenführungen exklusiver Pfade dahingehend, dass nicht nur eine Nachricht, sondern alle Nachrichten empfangen werden müssen, die seitens des parallelen Gateways versandt wurden, der mit der zu übersetzenden Zusammenführung korrespondiert, bevor die Ausführung fortgesetzt werden kann. Zu diesem Zweck werden gepufferte Nachrichtenkanäle verwendet, wobei für jede Zusammenführung die Deklaration eines gepufferten Nachrichtenkanals erzeugt wird, dessen Größe der Anzahl erwarteter Nachrichten entspricht, die mit der Anzahl eingehender Sequenzflüsse übereinstimmt:

```
chan buffered_channel⟨ID⟩ = [⟨size⟩] of { bit } /* ⟨...⟩ = Platzhalter */
```

Zur Nachbildung des Verhaltens von Zusammenführungen paralleler Pfade wird das `synchronization`-Makro verwendet (siehe Listing 7.7), das einen gepufferten Nachrichtenkanal und die Anzahl erwarteter Nachrichten als Parameter erwartet. Auch in diesem Fall ist das zentrale Element dieses Makros eine `do`-Schleife. Innerhalb dieser Schleife werden solange Nachrichten vom übergebenen Nachrichtenkanal gelesen, bis die Anzahl erhaltener Nachrichten mit der Anzahl erwarteter Nachrichten übereinstimmt.

```
1  inline synchronization(channel, number_of_messages)
2  {
3      byte i;
4      i = 0;
5
6      do
7      :: i == number_of_messages -> break
8      :: else -> end: channel ? _; i++
9      od
10 }
```

Listing 7.7: PROMELA-Makro für Zusammenführungen paralleler Pfade

Die Übersetzung einer Zusammenführung zweier paralleler Pfade resultiert in der Generierung eines Aufrufs des `synchronization`-Makros und eines Aufrufs des `send`-Makros (siehe Listing 7.8). Die Sprungmarke zur Markierung eines gültigen Endzustands befindet sich in diesem Fall innerhalb des `synchronization`-Makros (siehe Listing 7.7).

```
1  active proctype ParallelJoinGateway⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2  {
3      synchronization(buffered_channel⟨ID⟩, 2);
4      send(rendezvous_channels[⟨ID⟩], 1)
5  }
```

Listing 7.8: Übersetzung von Zusammenführungen paralleler Pfade (PROMELA)

7.2.2. Transformation musterbasierter Bedingungen in LTL-Formeln

Die Auswertung musterbasierter Bedingungen bezüglich eines BPMN-Modells kann auf Grundlage des mit diesem Modell korrespondierenden PROMELA-Programms in Kombination mit einer entsprechenden LTL-Formel durchgeführt werden. LTL-Formeln bestehen aus atomaren Aussagen, Operatoren der Aussagenlogik ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) und temporalen Operatoren ($\Box$, $\Diamond$, $\mathcal{U}$, $\mathcal{W}$). Atomare Aussagen können von Spin in jedem Zustand eines PROMELA-Programms ausgewertet werden. Bei atomaren Aussagen handelt es sich um Bezeichner global definierter Variablen oder Makros, die boolesche Ausdrücke

repräsentieren. Bei der Auswertung musterbasierter Bedingungen werden Variablen je nach Bedingung dazu verwendet, die Ausführung eines Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters zu signalisieren. Aus diesem Grund fließt der Typ der auszuwertenden Bedingung in die Transformation des BPMN-Modells ein. Bei der Transformation wird für jedes strukturelle Muster, auf das sich die auszuwertende Bedingung bezieht, die Deklaration einer globalen Variablen generiert, deren initialer Wert auf `false` gesetzt wird:

```
1 bool pattern1 = false
2 bool pattern2 = false
3 (...)
```

Zur Signalisierung der Aktivierung eines Prozesses, der mit einem Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters korrespondiert, werden bei der Transformation innerhalb der Deklaration dieses Prozesses Wertzuweisungen der entsprechenden Variablen generiert. Dieser Vorgang wird im Folgenden am Beispiel einer Nachfolger-Bedingung (siehe Abschnitt 5.3.2) erklärt. Eine Nachfolger-Bedingung, die sich auf Instanzen der strukturellen Muster M_1 und M_2 bezieht, erfordert, dass einem Ausgangsobjekt einer Instanz von M_1 im weiteren Verlauf ein Eingangsobjekt einer Instanz von M_2 folgen muss. Die LTL-Formel, die diesem Bedingungstyp entspricht, lautet wie folgt:

```
□(pattern1 → ◇pattern2)
```

Vor der Transformation des zu überprüfenden BPMN-Modells wird bestimmt, innerhalb welcher Prozessdeklarationen im korrespondierenden PROMELA-Programm bestimmte Wertzuweisungen erzeugt werden müssen. Dazu wird auf das Ergebnis der musterbasierte Suche zurückgegriffen (siehe Abschnitt 6.2.1.7 und Abschnitt 6.3.3). Handelt es sich beispielsweise bei einem zu übersetzenden Task um ein Ausgangsobjekt einer Instanz von M_1, wird eine Deklaration mit folgender Struktur generiert:

```
1  active proctype Task⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2  {
3  end:
4      receive(rendezvous_channels[⟨ID⟩], _);
5
6      atomic
7      {
8          pattern2 = false; /* = Instanz von M1 noch nicht ausgeführt */
9          pattern1 = true;  /* = Instanz von M2 ausgeführt */
10     }
11
12     send(rendezvous_channels[⟨ID⟩], 1)
13 }
```

Handelt es sich bei dem zu übersetzenden Task dagegen um ein Eingangsobjekt einer Instanz von M_2, sieht die Struktur der generierten Deklaration folgendermaßen aus:

```
1 active proctype Task⟨ID⟩() /* ⟨...⟩ = Platzhalter */
2 {
3 end:
4     receive(rendezvous_channels[⟨ID⟩], _);
5
6     pattern2 = true; /* = Instanz von M2 ausgeführt */
7
8     send(rendezvous_channels[⟨ID⟩], 1)
9 }
```

Um die Aktivierung eines Prozesses zu signalisieren, der mit einem Ausgangsobjekt einer Instanz von M_1 korrespondiert, wird nach Erhalt einer Nachricht der Wert der globalen Variable `pattern1` auf `true` gesetzt. Allerdings muss zuvor die Variable `pattern2` auf `false` gesetzt werden, da eventuell schon vor Aktivierung dieses Prozesses ein Prozess aktiviert worden ist, der mit einem Eingangsobjekt von M_2 korrespondiert und den Wert der globalen Variablen `pattern2` auf `true` gesetzt hat. In diesem Fall stellt die Wertzuweisung sicher, dass nur nachfolgende Aktivierungen von Prozessen, die mit Eingangsobjekten von M_2 korrespondieren, eine Auswirkung auf die Auswertung der LTL-Formel haben. Allerdings kann obige LTL-Formel zur Auswertung von Nachfolger-Bedingungen zu unerwarteten Ergebnissen führen, sofern das zu verifizierende BPMN-Modell Zyklen enthält. Dieser Sachverhalt ist in Abbildung 7.2 veranschaulicht.

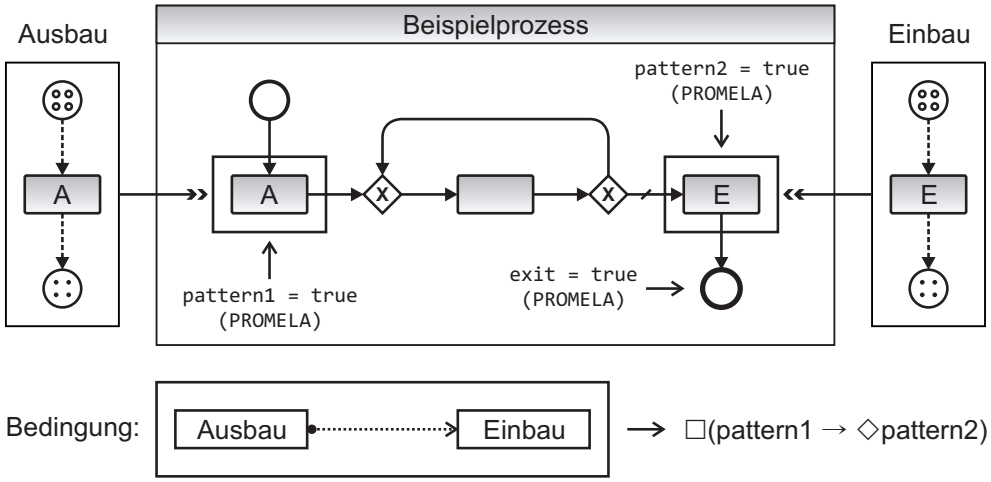


Abbildung 7.2.: Problem bei der Auswertung einer Nachfolger-Bedingung

Das in dieser Abbildung dargestellte BPMN-Modell enthält sowohl eine Instanz des auf der linken Seite abgebildeten strukturellen Musters *Ausbau* als auch eine Instanz des auf der rechten Seite abgebildeten Musters *Einbau*. Die am unteren Rand der Abbildung dargestellte Nachfolger-Bedingung soll ausdrücken, dass innerhalb von BPMN-Modellen, die mit dieser Bedingung verknüpft sind, auf jede darin enthaltene Instanz des Musters *Ausbau* eine Instanz des Musters *Einbau* folgen muss. Wird dieses BPMN-Modell in ein

PROMELA-Programm übersetzt, führt die Verifikation auf Basis obiger LTL-Formel zu einem Fehler, da Spin die Möglichkeit einer Endlosschleife aufgrund des innerhalb des BPMN-Modells enthaltenen Zyklus entdeckt. Da Zyklen auf die Bestimmung der temporalen Beziehung zwischen Instanzen struktureller Muster gemäß Definition 5.1 keine Auswirkung haben sollen, müssen geringfügige Änderungen bei der Erzeugung des PROMELA-Modells und an der LTL-Formel vorgenommen werden, indem am Anfang des Programms zunächst eine weitere globale Variable namens `exit` deklariert wird. Diese Variable wird gesetzt, sobald ein Prozess, der mit einem Endereignis des übersetzten BPMN-Modells korrespondiert, eine Nachricht empfängt. Dadurch wird signalisiert, dass die Ausführung des vom PROMELA-Programm repräsentierten BPMN-Modells beendet ist. Mithilfe dieser Variablen und der folgenden LTL-Formel kann nun ausgeschlossen werden, dass Zyklen wie im Fall des in Abbildung 7.2 dargestellten BPMN-Modells zur Verletzung einer Nachfolger-Bedingung führen, da die Überprüfung, ob auf ein Ausgangsobjekt von M_1 ein Eingangsobjekt von M_2 folgt, erst am Ende der Ausführung des PROMELA-Programms durchgeführt wird:

```
□(pattern1 → ◇(exit → pattern2))
```

Abbildung 7.3 listet die zu jedem Bedingungstyp gehörige LTL-Formel auf, die zur Auswertung darauf basierender Bedingungen verwendet wird. Darüber hinaus ist für jeden Bedingungstyp angegeben, welche Wertzuweisungen innerhalb von Prozessdeklarationen generiert werden, die mit den Ein- und Ausgangsobjekten der referenzierten strukturellen Muster korrespondieren. Dabei ist anzumerken, dass für Bedingungen vom Typ Vorgänger zwei Formeln angegeben sind. Zwar reicht die zuerst angegebene LTL-Formel zur Auswertung derartiger Bedingungen aus, allerdings bietet Spin keine Unterstützung für den darin enthaltenen $\mathcal{W}$-Operator (weak until). Mithilfe des von Spin unterstützen $\mathcal{U}$-Operators (strong until) kann jedoch eine äquivalente Formel angegeben werden. Die LTL-Formel zur Auswertung von Abfolge-Bedingungen besteht aus den LTL-Formeln zur Auswertung von Vorgänger- und Nachfolger-Bedingungen, die durch den logischen Operator $\wedge$ miteinander verknüpft sind.

Bedingungstyp	LTL-Formel	Wertzuweisungen (Ausgangsobjekte / M_1)	Wertzuweisungen (Eingangsobjekte / M_2)
Nachfolger	□(p1 → ◇(exit → p2))	`p2 = false` `p1 = true`	`p2 = true`
Vorgänger (1)	¬p2 $\mathcal{W}$ p1	`p1 = true`	`p2 = true`
Vorgänger (2)	□¬p2 ∨ (¬p2 $\mathcal{U}$ p1)	`p1 = true`	`p2 = true`
Abfolge	□(p1 → ◇(exit → p2a)) ∧ (□¬p2b ∨ (¬p2b $\mathcal{U}$ p1))	`p2a = false` `p1 = true`	`p2a = true` `p2b = true`
Negierte Abfolge	◇exit → □(p1 → ¬p2)	`p2 = false` `p1 = true`	`p2 = true`

Abbildung 7.3.: LTL-Formeln zur Auswertung temporaler Bedingungen

7.2.3. Direktnachfolger, Direktvorgänger und (negierte) Direktabfolge

Die temporalen Bedingungen Direktnachfolger-, Direktvorgänger, Direktabfolge und negierte Direktabfolge könnten ebenfalls mithilfe einer Modellprüfung ausgewertet werden. Da bei diesen Bedingungen überprüft wird, ob zwischen den Ein- und Ausgangsobjekten von Instanzen zweier struktureller Muster eine direkte Verbindung vorhanden ist oder nicht, also ob sie durch maximal einen Sequenzfluss voneinander getrennt sind oder nicht, und nur die Struktur von BPMN-Modellen (und nicht dessen Laufzeitverhalten) untersucht wird, reicht zu deren Auswertung ein spezieller Algorithmus aus. Direktnachfolger-Bedingungen können beispielsweise mit dem in Listing 7.9 dargestellten Pseudocode-Algorithmus ausgewertet werden.

```
1  function EvalChainResponseCondition(p1inst, p2inst)
2
3  outgoingObjects ← GetOutgoingObjects(p1inst)
4  incomingObjects ← GetIncomingObjects(p2inst)
5
6  for i ← 0 to GetSize(outgoingObjects) - 1 do
7      outgoingObject ← outgoingObjects[i]
8
9      if outgoingObject instanceof ConditionalSequenceConnector then
10         target ← GetTarget(outgoingObject)
11
12         if not incomingObjects contains target then return false
13
14     else
15         outgoingConnectors ← GetOutgoingConnectors(outgoingObject)
16
17         for j ← 0 to GetSize(outgoingConnectors) - 1 do
18             outgoingConnector ← outgoingConnectors[j]
19             target ← GetTarget(outgoingConnector)
20
21             if not incomingObjects contains target then return false
22
23 return true
```

Listing 7.9: Pseudocode-Funktion: EvalChainResponseCondition

7.2.4. Optimierungsmaßnahmen bei der Generierung von PROMELA-Programmen

Da bei einer Modellprüfung im ungünstigsten Fall der gesamte Zustandsraum des zu verifizierenden Systems durchlaufen werden muss, sollte bei der Modellierung der Systembeschreibung darauf geachtet werden, dass die Komplexität des beschriebenen Systems auf ein Minimum reduziert wird, ohne dass die zu überprüfenden Eigenschaften des Systems in Mitleidenschaft gezogen werden. Die in Abschnitt 7.2.1 beschriebene Transformation von BPMN-Modellen in PROMELA-Programme übersetzt die Elemente von

BPMN-Modellen bereits teilweise unter diesem Aspekt, jedoch sind weitere Optimierungsmaßnahmen möglich. Eine Möglichkeit zur Verringerung der Komplexität besteht darin, mehrere Flussobjekte innerhalb eines BPMN-Modells bei der Transformation zu einer Prozessdeklaration zusammenzufassen. Dabei spielt es in der Regel keine Rolle, ob es sich bei diesen Flussobjekten um Ein- oder Ausgangsobjekte von Instanzen struktureller Muster handelt, da Wertzuweisungen der entsprechenden Variablen zur Signalisierung der Ausführung ebenfalls innerhalb dieser Prozessdeklaration gruppiert werden können. Im Folgenden sind die Regeln aufgelistet, die bei der optimierten Transformation eines BPMN-Modells beachtet werden müssen, wobei die zuletzt erzeugte Prozessdeklaration als *aktive Prozessdeklaration* bezeichnet wird:

- **Regel 1:** Für jedes Startereignis wird eine entsprechende Prozessdeklaration erzeugt und aktiviert. Anschließend wird der ausgehende Sequenzfluss traversiert.
- **Regel 2:** Stößt der Algorithmus während der Traversierung auf einen Task oder ein Ereignis, wird keine neue Prozessdeklaration erzeugt. Handelt es sich bei dem traversierten Task oder Ereignis um ein Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters, wird der aktiven Prozessdeklaration eine entsprechende Wertzuweisung hinzugefügt.
- **Regel 3:** Stößt der Algorithmus während der Traversierung auf einen divergierenden Gateway, wird keine neue Prozessdeklaration erzeugt. Handelt es sich bei dem traversierten Gateway um ein Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters, wird der aktiven Prozessdeklaration eine entsprechende Wertzuweisung hinzugefügt. Danach wird der für den Gateway spezifische Code generiert. Anschließend wird für jeden ausgehenden Sequenzfluss eine neue Prozessdeklaration erzeugt, sofern der Sequenzfluss den divergierenden Gateway nicht direkt mit einem konvergierenden Gateway verbindet und kein Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters darstellt. Schließlich wird jeder der ausgehenden Sequenzflüsse traversiert, wobei zuvor die jeweils korrespondierende Prozessdeklaration aktiviert wird, sofern eine Prozessdeklaration erzeugt wurde.
- **Regel 4:** Stößt der Algorithmus während der Traversierung auf einen bedingten Sequenzfluss, bei dem es sich um ein Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters handelt, wird der aktiven Prozessdeklaration eine entsprechende Wertzuweisung hinzugefügt.
- **Regel 5:** Stößt der Algorithmus während der Traversierung auf einen konvergierenden Gateway, wird eine neue Prozessdeklaration erzeugt und aktiviert. Handelt es sich bei dem traversierten Gateway um ein Ein- oder Ausgangsobjekt einer Instanz eines strukturellen Musters, wird der aktiven Prozessdeklaration eine entsprechende Wertzuweisung hinzugefügt. Schließlich wird der für den Gateway spezifische Code generiert und der ausgehende Sequenzfluss traversiert.

Die Regeln zur optimierten Transformation von BPMN-Modellen sind in Abbildung 7.4 veranschaulicht. Bei der Transformation des in dieser Abbildung dargestellten BPMN-

Modells werden mehrere Elemente des Modells in Form von Prozessdeklarationen zusammengefasst. Die zusammengefassten Modellelemente sind innerhalb der Abbildung durch rote Rechtecke hervorgehoben. Darüber hinaus ist jedes dieser Rechtecke mit den in diesem Zusammenhang stehenden Regeln beschriftet. Dabei ist jeweils die Regel unterstrichen, auf welche die Erzeugung der jeweiligen Zusammenfassung zurückzuführen ist. Da die Ausführung eines bedingten Sequenzflusses lediglich in einem Fall eine Wertzuweisung einer Variablen zur Folge hat, wird *Regel 4* nur einmal angewandt.

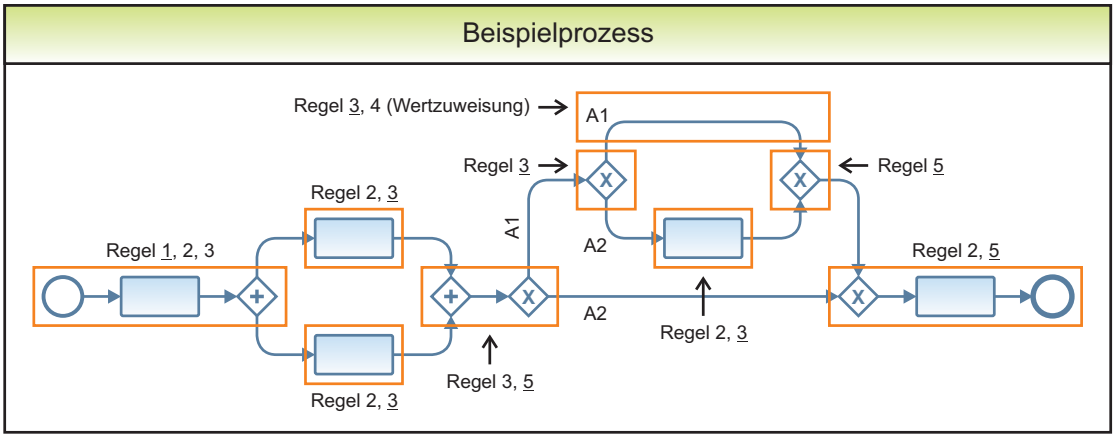


Abbildung 7.4.: BPMN-Modell und korrespondierende Prozessdeklarationen

7.3. Auswertung von Bedingungsausdrücken

Zur Auswertung eines Bedingungsausdrucks, der in Form eines Modells vorliegt und eine Baumstruktur aufweist, wird der Bedingungsausdruck in einen aussagenlogischen Ausdruck übersetzt, der zum besseren Verständnis als Zeichenkette aufgefasst werden kann. Zu diesem Zweck wird das Modell ausgehend vom Wurzelelement traversiert. Wird bei der Traversierung eine Bedingung angetroffen, wird diese wie in den vorangehenden Abschnitten beschrieben ausgewertet und dem resultierenden aussagenlogischen Ausdruck je nach Ergebnis `true` oder `false` hinzugefügt. Wird dagegen ein logischer Operator angetroffen, werden zunächst alle Kindknoten ausgewertet. Anschließend werden die Ergebnisse dieser Auswertungen mit dem entsprechenden aussagenlogischen Operator ($\wedge$ oder $\vee$) verknüpft und ebenfalls dem resultierenden aussagenlogischen Ausdruck hinzugefügt. Nach Beendigung dieses Vorgangs wird der aussagenlogische Ausdruck selbst ausgewertet. Ist das Ergebnis dieser Berechnung `true`, erfüllt das untersuchte BPMN-Modell den korrespondierenden Bedingungsausdruck. Falls diese Berechnung `false` zurückliefert, liegt eine Verletzung des Bedingungsausdrucks und damit eine Verletzung einer oder mehrerer Bedingungen innerhalb dieses Ausdrucks vor.

7.4. Implementierung: Constraint Checker

Zur automatischen Auswertung von Bedingungsausdrücken wurden zwei Eclipse Plug-ins entwickelt (siehe Abbildung 7.5). Eines dieser Plug-ins – der *Constraint Checker* – steuert alle Phasen der Auswertung eines Bedingungsausdrucks. Bedingungen innerhalb eines Bedingungsausdrucks, für die keine Modellprüfung erforderlich ist, werden direkt innerhalb dieses Plug-ins ausgewertet. Zur Durchführung einer Modellprüfung wird das zweite in diesem Kontext entwickelte Plug-in verwendet. Dieses Plug-in ist für die Übersetzung von BPMN-Modellen in PROMELA-Programme und die Ausführung des externen Modellprüfers Spin verantwortlich. In den folgenden Abschnitten wird auf die Funktionsweise dieser Plug-ins näher eingegangen.

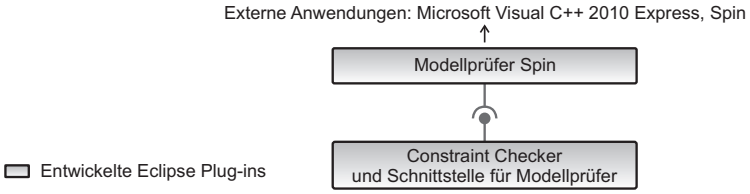


Abbildung 7.5.: Entwickelte Eclipse Plug-ins

7.4.1. Komponente: Constraint Checker

Der Constraint Checker ist die zentrale Komponente des entwickelten Prototyps und steuert die Auswertung von Bedingungsausdrücken, die mit einem BPMN-Modell verknüpft sind. Zu diesem Zweck werden mehrere Phasen durchlaufen, die in Abbildung 7.6 dargestellt sind. Die Auswertung wird entweder automatisch durch Speicherung eines modifizierten BPMN-Modells oder manuell durch eine Interaktion des Benutzers ausgelöst. Um dies zu ermöglichen, wurde der Process Composer entsprechend angepasst.

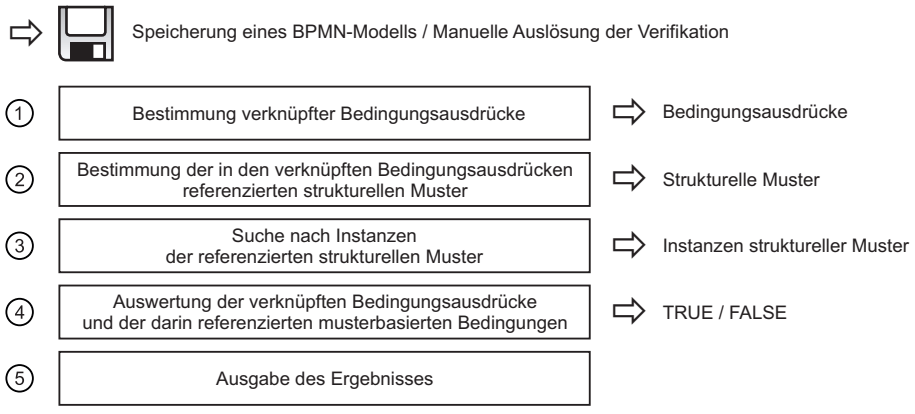


Abbildung 7.6.: Phasen der Auswertung von Bedingungsausdrücken

Zunächst überprüft der Constraint Checker, welche Bedingungsausdrücke mit dem zu verifizierenden BPMN-Modell verknüpft sind (siehe Abschnitt 5.4.3). Anschließend werden alle strukturellen Muster bestimmt, die von musterbasierten Bedingungen innerhalb dieser Bedingungsausdrücke referenziert werden. Dies kann durch Traversierung des entsprechenden Objektmodells oder mithilfe einer MQL-Abfrage erreicht werden. Auf Grundlage dieses Resultats wird eine Suche nach Instanzen dieser Muster durchgeführt, indem der vom Benutzer bevorzugte Mustersucher aufgerufen wird (siehe Abschnitt 6.2.2 und Abschnitt 6.3.4). Dabei werden dem Mustersucher Referenzen auf das BPMN-Modell und die entsprechenden PPML-Modelle übergeben. Statt nach Instanzen aller referenzierten Muster auf einmal zu suchen, könnten auch dedizierte Suchvorgänge für Instanzen einzelner Muster im Rahmen der Auswertung der jeweils referenzierenden Bedingung durchgeführt werden. Allerdings stellte sich bei der Implementierung heraus, dass diese Alternative beim Drools-Ansatz weniger effizient ist.

Das Ergebnis der Suche nach Instanzen struktureller Muster ist eine Datenstruktur, die jedem strukturellen Muster eine Menge gefundener Instanzen zuordnet. Mithilfe dieser Datenstruktur wird nun die Auswertung der Bedingungsausdrücke durchgeführt (siehe Abschnitt 7.3). Der zu diesem Zweck entwickelte Algorithmus ist in Form von Pseudocode in Listing 7.10 dargestellt. Stößt dieser Algorithmus auf einen aussagenlogischen Operator, werden zunächst alle Operanden ausgewertet. An dieser Stelle hätte der Algorithmus dahingehend optimiert werden können, dass im Fall einer Bedingungskonjunktion (siehe Abschnitt 5.3.3) die Berechnung weiterer Operanden abgebrochen wird, sofern die vorhergehende Auswertung eines Operanden `false` ergibt. Um dem Benutzer am Ende der Auswertung eines Bedingungsausdrucks über alle Verletzungen musterbasierter Bedingungen informieren zu können, wurde von dieser Maßnahme jedoch abgesehen. Findet der Algorithmus eine musterbasierte Bedingung vor, wird deren Auswertung je nach Bedingungstyp an den vom Benutzer bevorzugten Modellprüfer oder einen der in Abschnitt 7.1 und Abschnitt 7.2.3 beschriebenen Algorithmen delegiert. Im Fall einer Modellprüfung muss der Modellprüfer zuvor über den vom Constraint Checker Plug-in zur Verfügung gestellten Extension Point registriert werden. Falls der in Listing 7.10 dargestellte Algorithmus `false` zurückliefert, verletzt das BPMN-Modell den ausgewerteten Bedingungsausdruck. In diesem Fall muss zunächst bestimmt werden, welche der innerhalb des Bedingungsausdrucks referenzierten Bedingungen diese Verletzung verursachen. Zu diesem Zweck muss der Bedingungsausdruck traversiert und eine Menge entsprechender Bedingungen zurückliefert werden.

Nachdem feststeht, welche Bedingungsausdrücke ein BPMN-Modell verletzt und welche Bedingungen innerhalb dieser Ausdrücke die jeweilige Verletzung verursachen, werden diese Fehlerinformationen zunächst im Modell selbst gespeichert. Zu diesem Zweck wurde das ExtendedBPMN-Metamodell um entsprechende Klassen und Assoziationen erweitert (siehe Abschnitt A.3 im Anhang). Anschließend werden dem Benutzer eine oder mehrere entsprechende Fehlermeldungen innerhalb der Problemansicht angezeigt. Darüber hinaus wurde die Eigenschaftsansicht für Pools um einen Reiter mit der Beschriftung *Constraint Violations* erweitert, der auf die innerhalb des Modells gespeicherten Fehlerinformationen zugreift und in drei Listen unterteilt ist (siehe Abbildung 7.7).

```
function EvalConstraintExpression(expression)

result ← true

if expression instanceof ConstraintConjunction then
    children ← GetChildren(expression)

    for i ← 0 to GetSize(children) - 1 do
        child ← children[i]
        result ← EvalConstraintExpression(child) and result

elseif expression instanceof ConstraintInclusiveDisjunction then
    children ← GetChildren(expression)

    for i ← 0 to GetSize(children) - 1 do
        child ← children[i]
        result ← EvalConstraintExpression(child) or result

elseif expression instanceof ConditionReference then
    condition ← GetCondition(expression)
    result ← CheckCondition(condition)

return result
```

Listing 7.10: Pseudocode-Funktion: EvalConstraintExpression

In der ersten Spalte sind die Bedingungsausdrücke aufgelistet, die mit dem selektieren Pool verknüpft sind und von diesem Pool verletzt werden. Bei Auswahl eines dieser Bedingungsausdrücke werden in der zweiten Spalte die Bedingungen angezeigt, welche die Verletzung des Bedingungsausdrucks verursachen. Wird eine dieser Bedingungen selektiert, wird in der dritten Spalte je nach Bedingung (siehe Abbildung 7.8) eine Liste aus Instanzen oder Instanzpaaren des strukturellen Musters bzw. der strukturellen Muster aufgeführt, die wiederum für die Verletzung der Bedingung verantwortlich sind. Nach Auswahl einer Instanz oder eines Instanzpaars kann der Benutzer schließlich durch Betätigung einer der auf der rechten Seite angezeigten Schaltflächen die Instanz oder eine der Instanzen des Instanzpaars innerhalb des BPMN-Modells hervorheben, sodass der Grund für die Verletzung der Bedingung genau analysiert werden kann.

7.4.2. Komponente: Modellprüfer (Spin)

Die Modellprüfer-Komponente wird vom Constraint Checker aufgerufen, sofern zur Auswertung einer musterbasierten Bedingung eine Modellprüfung erforderlich ist. Zu diesem Zweck wird der Modellprüfer-Komponente eine entsprechende Datenstruktur übergeben, die in Abbildung 7.9 dargestellt ist. Diese Datenstruktur enthält eine Referenz auf das BPMN-Modell und eine LTL-Formel in Form einer Zeichenkette. Darüber hinaus ist in dieser Datenstruktur festgehalten, für welche Elemente des zu verifizieren-

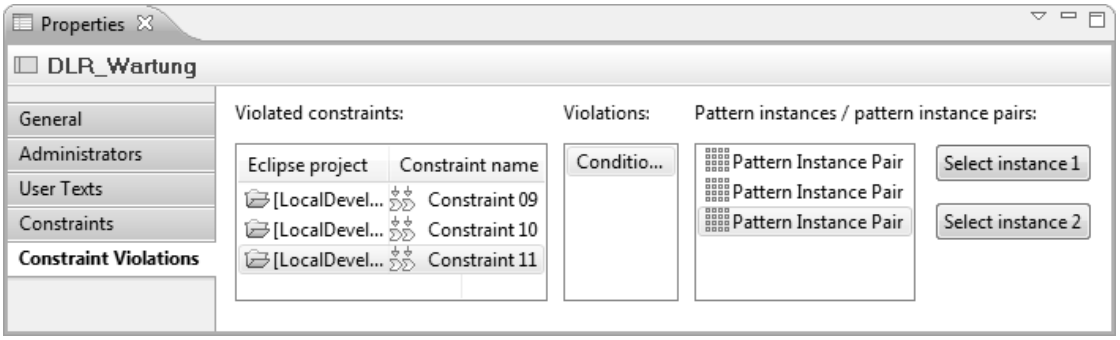


Abbildung 7.7.: Eigenschaftsansicht: Übersicht verletzter Bedingungsausdrücke

Modellierungskonstrukt	Rückgabewert der Auswertung	Modellierungskonstrukt	Rückgabewert der Auswertung
Abwesenheit	Liste aus Instanzen eines strukturellen Musters	Direktnachfolger	Liste aus Instanzen eines strukturellen Musters
Exaktheit		Vorgänger	
Existenz		Direktvorgänger	
Erwiderte Existenz		Abfolge	
Koexistenz		Direktabfolge	
Negierte Koexistenz		Negierte Abfolge	Liste aus Paaren von Instanzen struktureller Muster
Nachfolger		Negierte Direktabfolge	

Abbildung 7.8.: Rückgabewert bei der Auswertung musterbasierter Bedingungen

den BPMN-Modells Wertzuweisungen in den korrespondierenden Prozessdeklarationen des PROMELA-Programms generiert werden müssen. Des Weiteren kann mithilfe eines Flags angegeben werden, ob bei Ausführung von Endereignissen eine entsprechende Wertzuweisung der Variablen `exit` erfolgen soll (siehe Abschnitt 7.2.2). Mithilfe eines weiteren Flags kann bestimmt werden, ob sich der Modellprüfer bei Feststellung einer Verletzung auf die Rückgabe des zuerst gefundenen Fehlers beschränken oder möglichst viele Fehler suchen soll. Schließlich enthält die Datenstruktur Informationen, die zur detaillierten Analyse gefundener Fehler benötigt werden.

Nach Aufruf der Modellprüfer-Komponente wird zunächst das übergebene BPMN-Modell auf Grundlage eines Algorithmus, der auf den in Abschnitt 7.2.1 vorgestellten Konzepten basiert, in ein PROMELA-Programm in Form einer `pml`-Datei transformiert. Im nächsten Schritt wird Spin aufgerufen, um auf Grundlage dieses Programms den Quellcode für einen speziell daran angepassten Modellprüfer in Form eines C-Programms zu erzeugen, wobei die LTL-Formel als Parameter übergeben wird:

```
spin -a -f ⟨LTL-Formel⟩ model.pml
```

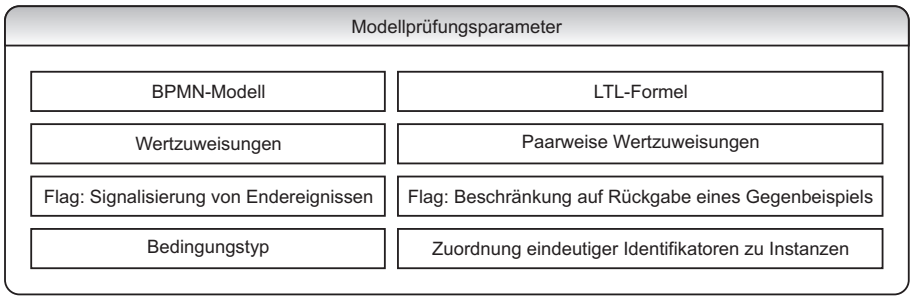


Abbildung 7.9.: Modellprüfungsparameter

Anschließend wird ein C-Compiler aufgerufen, um aus dem erzeugten Quellcode ein ausführbares Programm zu erzeugen, dessen Name standardmäßig *pan* lautet. Im Rahmen dieser Arbeit wurde der C-Compiler der Firma *Microsoft* verwendet, der im Produkt *Visual C++ 2010 Express* [55] enthalten ist. Nach Fertigstellung der Kompilierung wird das resultierende Programm aufgerufen. Um festzustellen, ob das BPMN-Modell eine musterbasierte Bedingung mehrfach verletzt (d. h. möglichst viele Fehler innerhalb des PROMELA-Programms finden soll), müssen weitere Parameter hinzugefügt werden:

```
pan -a -n [-c0 -e]
```

An dieser Stelle sei angemerkt, dass es nicht gelang, das PROMELA-Programm bei der Transformation so zu erzeugen, dass Spin immer alle Fehler im Rahmen einer einzigen Modellprüfung findet. Werden jedoch mehrere Modellprüfungen durchgeführt und jeweils im Anschluss die von Spin gefundenen Fehler beseitigt, findet Spin letztendlich auch zuvor nicht erkannte Fehler. Ein Nachteil von Spin und anderen frei verfügbaren Modellprüfern ist, dass das Resultat der Modellprüfung nicht anhand einer Programmierschnittstelle abgefragt werden kann. Stattdessen wird das Ergebnis, das über etwaige Fehler Auskunft gibt, als Zeichenkette in den Standardausgabestrom geschrieben, beispielsweise wie in Abbildung 7.10 dargestellt. Aus diesem Grund muss diese Zeichenkette, der eine proprietäre Syntax zugrunde liegt, zunächst lexikalisch analysiert werden.

Sofern sich auf Grundlage der lexikalischen Analyse herausstellt, dass das PROMELA-Programm einen oder mehrere Fehler enthält, lässt sich daraus jedoch nicht ableiten, warum die entsprechende musterbasierte Bedingung verletzt wird und welche Instanzen struktureller Muster in diesem Zusammenhang eine Rolle spielen. Diese Informationen werden jedoch benötigt, sofern dem Benutzer detaillierte Fehlerinformationen präsentiert werden sollen (siehe Abbildung 7.7). Aus diesem Grund wurde ein Verfahren entwickelt, dass die Extraktion dieser Informationen erlaubt. Dieses Verfahren basiert auf mehreren Funktionen, die Spin zur Verfügung stellt. Zum einen erstellt Spin für jeden Fehler, der während der Verifikation entdeckt wird, eine sogenannte *Trail-Datei*. Mithilfe dieser Datei kann Spin im Simulationsmodus die Befehlsfolge wiederholen, die während der Verifikation zu einem Fehler geführt hat. Zum anderen kann innerhalb von PROMELA-

```
pan: acceptance cycle (at depth 103)

(Spin Version 5.2.5 -- 17 April 2010)
Warning: Search not completed
       + Partial Order Reduction

Full statespace search for:
       never claim               +
       assertion violations      + (if within scope of claim)
       acceptance   cycles       + (fairness disabled)
       invalid end states  - (disabled by never claim)

State-vector 200 byte, depth reached 104, errors: 1
       92 states, stored
        6 states, matched
       98 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

    2.501    memory usage (Mbyte)
```

Anzahl gefundener Fehler

Abbildung 7.10.: Ausgabe des Resultats einer Modellprüfung in der Standardausgabe

Programmen der Befehl `printf` verwendet werden, um eine Zeichenkette in den Standardausgabestrom zu schreiben. Allerdings werden diese Ausgaben bei der Verifikation sinnigerweise ignoriert und finden nur im Simulationsmodus Beachtung, also auch bei Wiederholung einer Befehlsfolge anhand einer Trail-Datei.

Erfolgt während der Transformation eines BPMN-Modells in ein PROMELA-Programm die Generierung einer Wertzuweisung einer Variablen, welche die Ausführung eines Ein- oder Ausgangsobjekts einer Instanz eines strukturellen Musters signalisiert, wird zusätzlich ein `printf`-Befehl generiert, der sowohl ausgibt, welche Variable auf welchen Wert gesetzt wurde, als auch ausgibt, um welche Instanz es sich handelt, was mithilfe eines intern verwalteten Identifikators bewerkstelligt wird, beispielsweise folgendermaßen:

```
+++ variable set: ID=23 | p1=true +++
```

Stellt sich bei der Verifikation eines BPMN-Modells heraus, dass dieses Modell eine musterbasierte Bedingung verletzt, wird Spin im Simulationsmodus aufgerufen, um anhand der Trail-Datei die entsprechende Befehlsfolge zu wiederholen. Wie zuvor wird auch in diesem Fall der Standardausgabestrom lexikalisch analysiert, indem Ausgaben, die der obigen Syntax entsprechen, in der Reihenfolge ihres Auftretens in einer Liste gespeichert werden. Je nach ausgewerteter Bedingung wird diese Liste unterschiedlich analysiert, um auf diese Weise detaillierte Fehlerinformationen zu erhalten.

Zur Auswertung einer Vorgänger-Bedingung (jedem Eingangsobjekt einer Instanz von M_2 muss ein Ausgangsobjekt von M_1 im vorherigen Verlauf bzw. direkt vorausgehen) wird über diese Liste von vorne bis hinten iteriert. Sofern bei der Iteration eine Zeichenkette gefunden wird, welche die Teilzeichenfolge `p1=true` (Auftreten eines Ausgangs-

objekts einer Instanz von M_1) enthält, wird die Iteration abgebrochen. Wird hingegen eine Zeichenkette gefunden wird, welche die Teilzeichenfolge `p2=true` (Auftreten eines Eingangsobjekts einer Instanz von M_2) enthält, wird die entsprechende Instanz des strukturellen Musters anhand des Identifikators innerhalb dieser Zeichenkette bestimmt und der Ergebnismenge hinzugefügt. Dieser Vorgang ist darauf zurückzuführen, dass bei der untersuchten Befehlsfolge ein Eingangsobjekt einer Instanz von M_2, jedoch bisher kein Ausgangsobjekt einer Instanz von M_1 vorgefunden wurde und damit die geforderte temporale Beziehung zwischen Instanzen dieser Muster verletzt wird. Am Ende der Auswertung wird die Ergebnismenge zurückgegeben. Die jeweiligen Algorithmen für musterbasierte Bedingungen vom Typ Nachfolger, Vorgänger, Abfolge und negierte Abfolge sind in Form von Pseudocode in Abbildung 7.11 dargestellt.

7.5. Stand der Wissenschaft und Technik

Zahlreiche Ansätze beschäftigen sich mit der Überprüfung von Eigenschaften von Geschäftsprozessmodellen zur Entwurfszeit. Diese Ansätze lassen sich in zwei Kategorien unterteilen: Ansätze, die ein Modell auf *Korrektheit* überprüfen und solche, die ein Modell auf die Einhaltung gesetzlicher, vertraglicher oder unternehmensinterner Richtlinien überprüfen (im Englischen als *Compliance* bezeichnet). Zwar wurde in Abschnitt 2.2.2 beschrieben, dass ein Metamodell die Möglichkeit bietet, ein darauf basierendes Modell automatisch auf Korrektheit zu überprüfen, allerdings können mithilfe eines Metamodells nur grundlegende Merkmale eines System spezifiziert werden. Eine komplexere Anforderung an ein BPMN-Modell, die im Rahmen der BPMN-Spezifikation beschrieben wird und nicht mithilfe eines Metamodells ausgedrückt werden kann, bezieht sich beispielsweise auf die korrekte Verwendung konvergierender exklusiver und paralleler Gateways [OMG09a, S. 76]. Der in der vorliegenden Arbeit präsentierte Ansatz ist jedoch nicht in diese, sondern in die zweite Kategorie einzuordnen.

In den letzten Jahren wurden mehrere Ansätze zur Überprüfung von Geschäftsprozessmodellen auf die Einhaltung gesetzlicher, vertraglicher oder unternehmensinterner Richtlinien vorgeschlagen. Bei der überwiegenden Mehrzahl dieser Ansätze basiert diese Überprüfung darauf, den Raum aller möglichen Zustandsfolgen, die bei Ausführung eines Prozessmodells auftreten können, zu analysieren. Zu diesem Zweck wird in allen Fällen eine Modellprüfung durchgeführt, wozu das Prozessmodell zuvor in die vom verwendeten Modellprüfer benötigte Repräsentation transformiert wird, bei der es sich letztendlich um einen endlichen Automaten handelt. Zur Überprüfung von Richtlinien werden diese üblicherweise als Formeln einer temporalen Logik ausgedrückt, beispielsweise auf Grundlage von LTL oder *Computation Tree Logic* (CTL). Die verfügbaren Ansätze lassen sich anhand der jeweils zugrunde liegenden Modellierungsmethode unterteilen: Ansätze auf Grundlage von EPKs [Rum99, FF08], BPMN [Bra05, BDSV05, ADW08], UML-Aktivitätsdiagrammen [FESS07] oder BPEL [LMX07]. Die in [AP06a, AP06b, AP06c] vorgestellten Methoden zur deklarativen Modellierung automatisierter Geschäftsprozessmodelle ConDec und DecSerFlow basieren ebenfalls auf diesen Prinzipien. Ein von diesen

Modellierungskonstrukt	Auswertung
	`Eingabe: lines (Liste aus Zeichenketten = Ausgabe von Spin im Simulationsmodus)`

Nachfolger	`result ← <Menge aus Instanzen struktureller Muster>`
	
	`for i ← GetSize(lines) - 1 to 0 do`
	`  line ← lines[i]`
	
	`  if Contains(line, "p1=true") then`
	`  patternInstance ← ExtractPatternInstance(line)`
	`  result ← patternInstance`
	
	`  else if Contains(line, "p2=true") then return result`

Vorgänger	`result ← <Menge aus Instanzen struktureller Muster>`
	
	`for i ← 0 to GetSize(lines) - 1 do`
	`  line ← lines[i]`
	
	`  if Contains(line, "p2=true") then`
	`  patternInstance ← ExtractPatternInstance(line)`
	`  result ← patternInstance`
	
	`  else if Contains(line, "p1=true") then return result`

Abfolge	`flag ← false`
	`instances ← <Menge aus Instanzen struktureller Muster>`
	`result ← <Menge aus Instanzen struktureller Muster>`
	
	`for i ← 0 to GetSize(lines) - 1 do`
	`  line ← lines[i]`
	
	`  if Contains(line, "p1=true") then`
	`  flag ← true`
	`  patternInstance ← ExtractPatternInstance(line)`
	`  instances ← patternInstance`
	
	`  else if Contains(line, "p2a=true") then Clear(instances)`
	
	`  else if Contains(line, "p2b=true") and not flag then`
	`  patternInstance ← ExtractPatternInstance(line)`
	`  result ← patternInstance`
	
	`result ← instances`

Negierte Abfolge	`flag ← false`
	`instances ← <Menge aus Instanzen struktureller Muster>`
	`result ← <Menge aus Paaren aus Instanzen struktureller Muster>`
	
	`for i ← 0 to GetSize(lines) - 1 do`
	`  line ← lines[i]`
	
	`  if Contains(line, "p1=true") then`
	`  flag ← true`
	`  patternInstance_p1 ← ExtractPatternInstance(line)`
	`  instances ← patternInstance_p1`
	
	`  else if Contains(line, "p2=true") and flag then`
	`  patternInstance_p2 ← ExtractPatternInstance(line)`
	
	`  for j ← 0 to GetSize(instances) - 1 do`
	`  patternInstance_p1 ← instances[j]`
	`  result ← (patternInstance_p1, patternInstance_p2)`

	`return result`

Abbildung 7.11.: Algorithmen zur Extraktion detaillierter Fehlerinformationen

Prinzipien abweichender, aber in diesem Zusammenhang erwähnenswerter Ansatz ist die in [SM06] beschriebene Vorgehensweise zur Verifikation Ereignisgesteuerter Prozessketten, um verbotenes Verhalten zu erkennen.

Die in dieser Arbeit vorgestellte Vorgehensweise zur Auswertung musterbasierter Bedingungen unterscheidet sich in einem Punkt fundamental von den bisherigen Ansätzen. Während letztere den Raum aller möglichen Zustandsfolgen auswerten, wird bei der Auswertung musterbasierter Bedingungen, die gesetzliche, vertragliche oder unternehmensinterne Richtlinien repräsentieren können, zunächst das Vorhandensein gewisser Anordnungen von Modellelementen innerhalb des zu verifizierenden Geschäftsprozessmodells überprüft, also dessen vom Benutzer vorgegebene Struktur durchsucht. Beispielsweise ist es mithilfe laufzeitbasierter Verfahren nicht möglich, Instanzen des in Abbildung 5.19 dargestellten Musters M_1 innerhalb eines Geschäftsprozessmodells zu suchen, da es sich um eine *zweidimensionale* Struktur, nicht jedoch um eine *eindimensionale* Zustandsfolge handelt. Dieser Sachverhalt wird in Abbildung 7.12 verdeutlicht, die ein strukturelles Muster M und ein Geschäftsprozessmodell G darstellt, wobei M eine Sequenz zweier Tasks beschreibt, die mit A und B beschriftet sind. Obwohl keine Instanz von M innerhalb von G enthalten ist, kann die Ausführung von G dazu führen, dass *Task B* direkt nach *Task A* ausgeführt wird.

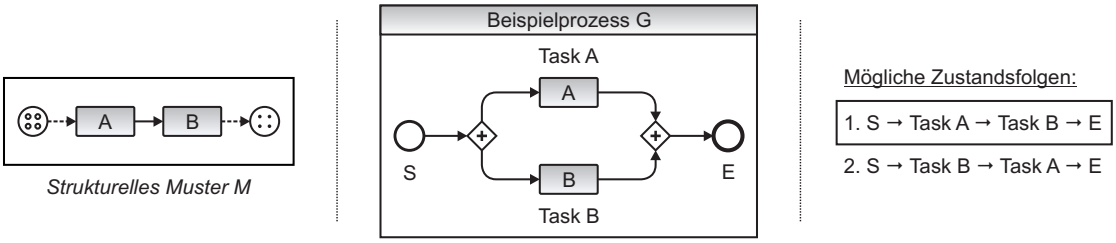


Abbildung 7.12.: BPMN-Modell und mögliche Zustandsfolgen

Insgesamt handelt es sich bei struktur- und laufzeitbasierten Verfahren um unterschiedliche Sichtweisen, die beide ihre Daseinsberechtigung haben. Neuartig am vorliegenden Ansatz und im Unterschied zu anderen strukturbasierten Suchverfahren (siehe Abschnitt 6.5), ist die Möglichkeit, Bedingungen auf Grundlage struktureller Muster spezifizieren und auswerten zu können, wobei temporale Bedingungen auch Laufzeitaspekte berücksichtigen und damit beide Verfahrensweisen verknüpft werden. Als einziger in Grundzügen vergleichbarer Ansatz ist die in [WKH08] vorgestellte *Business Process Compliance Language* zu nennen, die auf OCL basiert. Kommerzielle BPM-Werkzeuge unterstützen in der Regel nur rudimentäre Verfahren zur Auswertung benutzerdefinierter Anforderungen an Geschäftsprozessmodelle, beispielsweise können mithilfe des *ARIS Business Architect* sogenannte *Semantic Checks* [DB07, S. 337] durchgeführt werden.

8. Validierung

Auf Grundlage der in Kapitel 3.6 formulierten Anforderungen an eine Softwarelösung zur Modellierung und automatischen Auswertung musterbasierter Bedingungen wurden in den vier vorangehenden Kapiteln entsprechende Konzepte und ein darauf basierender Prototyp vorgestellt. Diese Konzepte orientieren sich auf der einen Seite an den spezifischen Problemen des in Abschnitt 3.4 beschriebenen Szenarios, sind jedoch auf der anderen Seite so allgemein gehalten, dass sie auf andere Problembereiche übertragen werden können. In diesem Kapitel werden die entwickelten Konzepte validiert, indem der Prototyp zur Lösung der im Rahmen des Szenarios aufgezeigten Probleme verwendet wird. Darüber hinaus wird bei der Validierung überprüft, ob der Prototyp die in Kapitel 3.6 formulierten funktionalen und nichtfunktionalen Anforderungen erfüllt, wobei in diesem Zusammenhang auch Leistungsmessungen präsentiert werden. Am Ende des Kapitels wird schließlich auf die externe Veröffentlichung und interne Verwertung der im Rahmen dieser Arbeit erzielten Ergebnisse eingegangen.

8.1. Anwendung der entwickelten Konzepte im Rahmen des Szenarios

Im Folgenden wird die Beschreibung des in Abschnitt 3.4 dargestellten Szenarios fortgesetzt. Ausgangspunkt der anschließenden Ausführungen ist die Aufgabe des Business-Analysten, die zuvor handschriftlich dokumentierten Arbeitsschritte im Rahmen des Wartungs-, des Warenausgangs- und des Wareneingangsprozesses (siehe Abschnitt 3.1) unter Berücksichtigung der dabei zu beachtenden Anforderungen zu modellieren. Zu diesem Zweck verwendet der Business-Analyst den Process Composer der Firma SAP. Zur Vermeidung von Modellierungsfehlern, die auf verletzte Anforderungen zurückzuführen sind, setzt der Business-Analyst die in dieser Arbeit präsentierte Erweiterung des Process Composers ein. Da im Zuge der Dokumentation und Teilautomatisierung von Geschäftsprozessen des luftfahrttechnischen Betriebs Konzepte des semantischen Geschäftsprozessmanagements eingesetzt werden sollen, modelliert der Business-Analyst zunächst eine domänenspezifische Ontologie, welche verschiedene Aspekte der Flugzeugwartung bis zu einer gewissen Abstraktionsebene beschreibt. Im nächsten Schritt modelliert der Business-Analyst die zu beachtenden Anforderungen in Form musterbasierter Bedingungen. Schließlich erfolgt die eigentliche Spezifikation der Geschäftsprozessmodelle unter Verwendung der zuvor modellierten Domänenontologie.

8.1.1. Erstellung einer Ontologie zur Beschreibung von Konzepten im Bereich der Flugzeugwartung

Da zur Modellierung der Geschäftsprozesse des luftfahrttechnischen Betriebs eine Entscheidung zugunsten der Verwendung von Konzepten des semantischen Geschäftsprozessmanagements gefällt wurde, modelliert der Business-Analyst mithilfe des Ontology Composers zunächst eine Domänenontologie, welche die verschiedenen Arbeitsschritte im Bereich der Flugzeugwartung beschreibt. Ein Ausschnitt dieser Ontologie ist in Abbildung 8.2 dargestellt. Diese Ontologie enthält größtenteils atomare Klassen, die nach inhaltlicher Bedeutung gruppiert (z. B. mechanische und logistische Arbeitsschritte, Zustände, etc.) und aufgrund von Vererbungsbeziehungen hierarchisch angeordnet sind. Der Beschreibung sicherheitskritischer Aktivitäten und dokumentationspflichtiger Situationen liegen komplexe Klassen zugrunde. Anstelle von Beschriftungen in natürlicher Sprache werden die in dieser Ontologie beschriebenen Klassen im späteren Verlauf zur semantischen Anreicherung von BPMN-Elementen verwendet.

8.1.2. Modellierung musterbasierter Bedingungen an Geschäftsprozessmodelle im Bereich der Flugzeugwartung

Nach Abschluss der Anfertigung der Domänenontologie modelliert der Business-Analyst für jede der in Abschnitt 3.4.1 beschriebenen Anforderungen jeweils einen Bedingungsausdruck und zugehörige strukturelle Muster mithilfe des Constraint bzw. Pattern Composers. Abbildung 8.1 zeigt für jede Anforderung den korrespondierenden Bedingungsausdruck auf Grundlage der in Listing 5.2 dargestellten Syntax und die mit diesem Ausdruck verknüpften Geschäftsprozessmodelle. Die konkreten Bedingungsausdrücke und strukturellen Muster sind in Abbildung 8.3 und Abbildung 8.4 dargestellt.

Anforderung	Korrespondierender Bedingungsausdruck	Verknüpfte Modelle
Anforderung 1	`Exaktheit(M1-1)`	Wartung
Anforderung 2	`Existenz(M2-1) ∨ Existenz(M2-2)`	Wartung
Anforderung 3	`Abfolge(M3-1, M3-2)`	Wartung
Anforderung 4	`Direktnachfolger(M4-1, M4-2)`	Wartung
Anforderung 5	`Exaktheit(M5-1)`	Warenausgang
Anforderung 6	`Nachfolger(M6-1, M6-2)`	Warenausgang
Anforderung 7	`Nachfolger(M7-1, M7-1)`	Wareneingang
Anforderung 8	`Koexistenz(M8-1, M8-2)`	Wareneingang

Abbildung 8.1.: Anforderungen und korrespondierende Bedingungsausdrücke

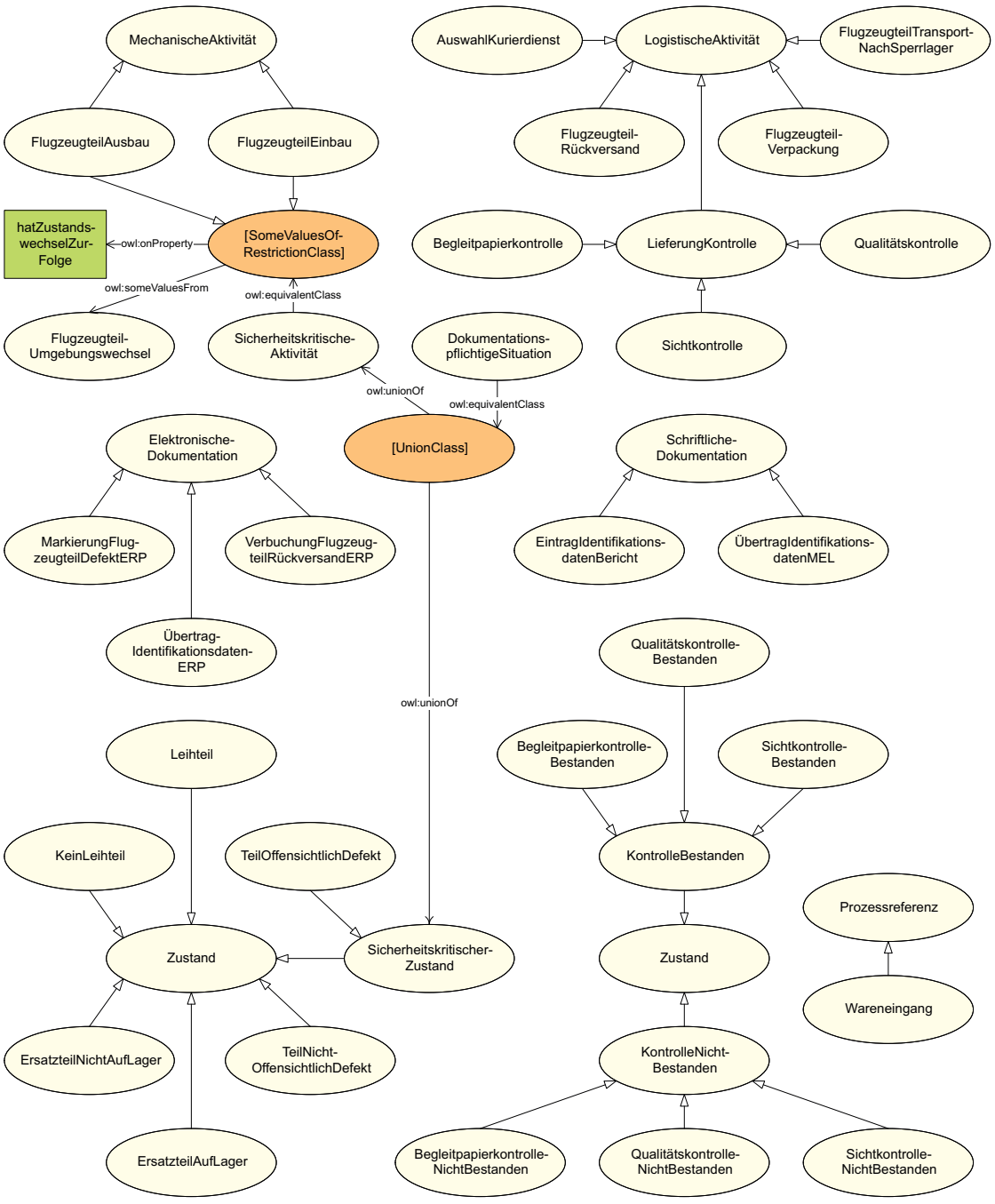


Abbildung 8.2.: Wartungsprozessontologie (Ausschnitt)

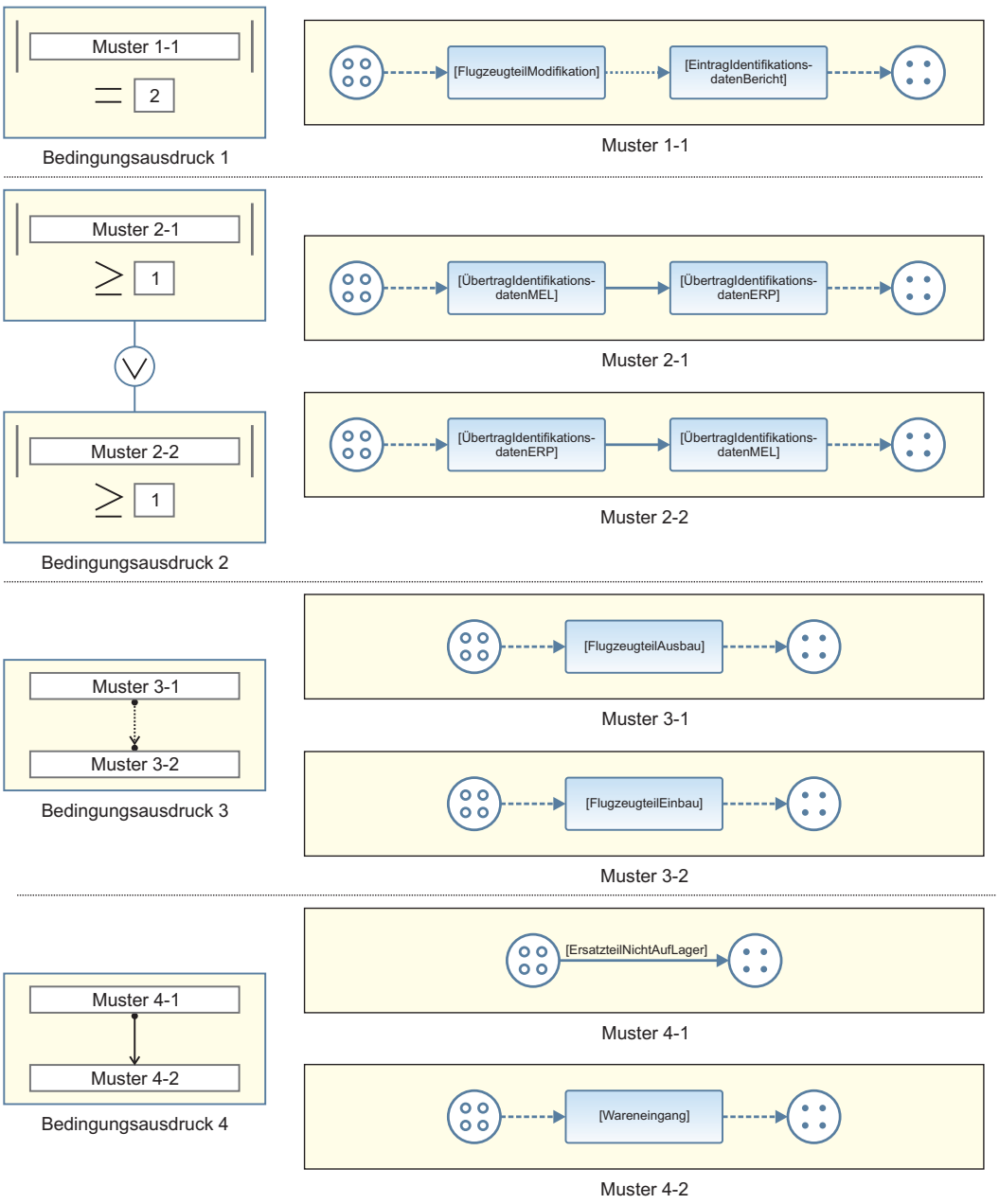


Abbildung 8.3.: Bedingungsausdrücke

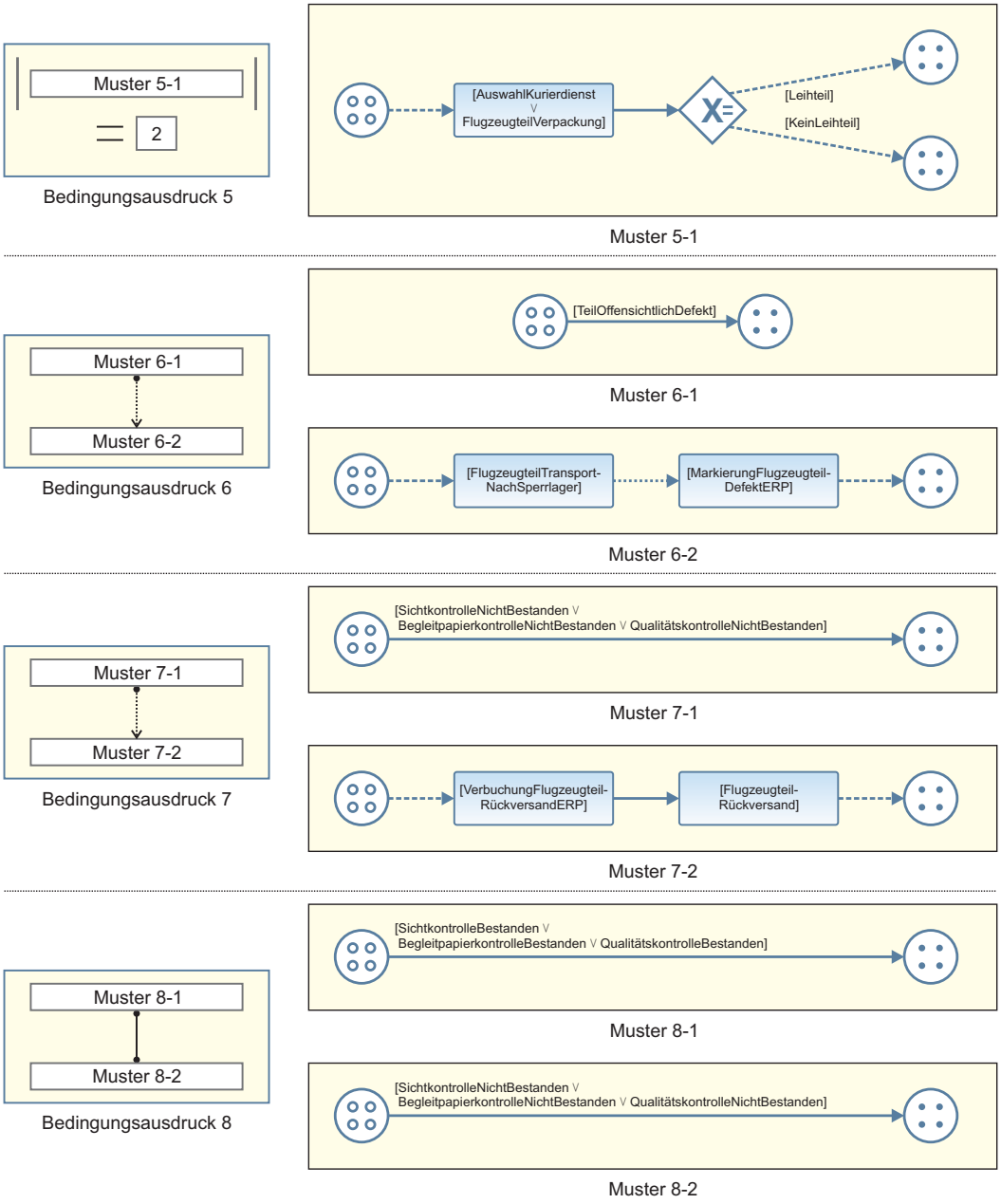


Abbildung 8.4.: Bedingungsausdrücke (Fortsetzung)

8.1.3. Modellierung und automatische Verifikation von Geschäftsprozessmodellen im Bereich der Flugzeugwartung

Im Anschluss an die Modellierung der Bedingungsausdrücke und strukturellen Muster modelliert der Business-Analyst schließlich die eigentlichen Geschäftsprozessmodelle des luftfahrttechnischen Betriebs: den Wartungsprozess (siehe Abbildung 8.6 und 8.7), den Warenausgangsprozess (siehe Abbildung 8.8) und den Wareneingangsprozess (siehe Abbildung 8.9). Nach Abschluss der Modellierung löst der Business-Analyst deren automatische Verifikation durch den Constraint Checker aus. Dabei stellt sich heraus, dass alle drei Modelle Modellierungsfehler aufweisen (siehe Abbildung 8.5).

BPMN-Modell	Verletzter Bedingungsausdruck	Grund
Wartungsprozess	Bedingungsausdruck 1	Das Modell enthält nur eine Instanz von Muster 1-1.
Wartungsprozess	Bedingungsausdruck 2	Das Modell enthält weder eine Instanz von Muster 2-1 noch eine Instanz von Muster 2-2.
Warenausgangsprozess	Bedingungsausdruck 5	Das Modell enthält nur eine Instanz von Muster 5-1.
Warenausgangsprozess	Bedingungsausdruck 6	Das Modell enthält eine Instanz von Muster 6-1, auf die keine Instanz von Muster 6-2 folgt.
Wareneingangsprozess	Bedingungsausdruck 7	Das Modell enthält eine Instanz von Muster 7-1, auf die keine Instanz von Muster 7-2 folgt.

Abbildung 8.5.: Ergebnis der Verifikation

Das Wartungsprozessmodell (siehe Abbildung 8.6 und 8.7) verletzt zwei Bedingungsausdrücke. *Bedingungsausdruck 1* wird verletzt, da zwischen dem Task, in dem ein Flugzeugteil ausgebaut wird und dem Task, in dem die Identifikationsdaten dieses Teils im Inspektionsbericht dokumentiert werden, zwei Tasks erfolgen, obwohl nur maximal ein Schritt zulässig ist, und damit nur eine Instanz von *Muster 1-1* innerhalb des Modells enthalten ist. *Bedingungsausdruck 2* wird verletzt, da der Business-Analyst vergessen hat, einen Task zu modellieren, in dem die Identifikationsdaten eingebauter Flugzeugteile innerhalb der MEL dokumentiert werden, und damit weder eine Instanz von *Muster 2-1* noch eine Instanz von *Muster 2-2* innerhalb des Modells enthalten ist.

Das Warenausgangsprozessmodell (siehe Abbildung 8.8) verletzt ebenfalls zwei Bedingungsausdrücke. *Bedingungsausdruck 5* wird verletzt, da der Business-Analyst übersehen hat, dass bei der Verpackung eines zu versendenden Flugzeugteils zwischen Leihteilen und Nicht-Leihteilen unterschieden werden muss, und damit nur eine Instanz von *Muster 5-1* innerhalb des Modells enthalten ist. *Bedingungsausdruck 6* wird verletzt, da der Business-Analyst versäumt hat, einen Task zu modellieren, in dem ein offensichtlich defektes Teil im ERP-System als solches gekennzeichnet wird. Daher enthält das Modell zwar eine Instanz von *Muster 6-1*, auf die jedoch keine Instanz von *Muster 6-2* folgt.

Schließlich verletzt das Wareneingangsprozessmodell (siehe Abbildung 8.9) *Bedingungsausdruck 7*, da auf eine der drei Instanzen von *Muster 7-1* keine Instanz von *Muster 7-2* folgt. Auf Grundlage dieses Resultats korrigiert der Business-Analyst die fehlerhaften Geschäftsprozessmodelle (siehe Abbildung 8.10, 8.11, 8.12 und 8.13).

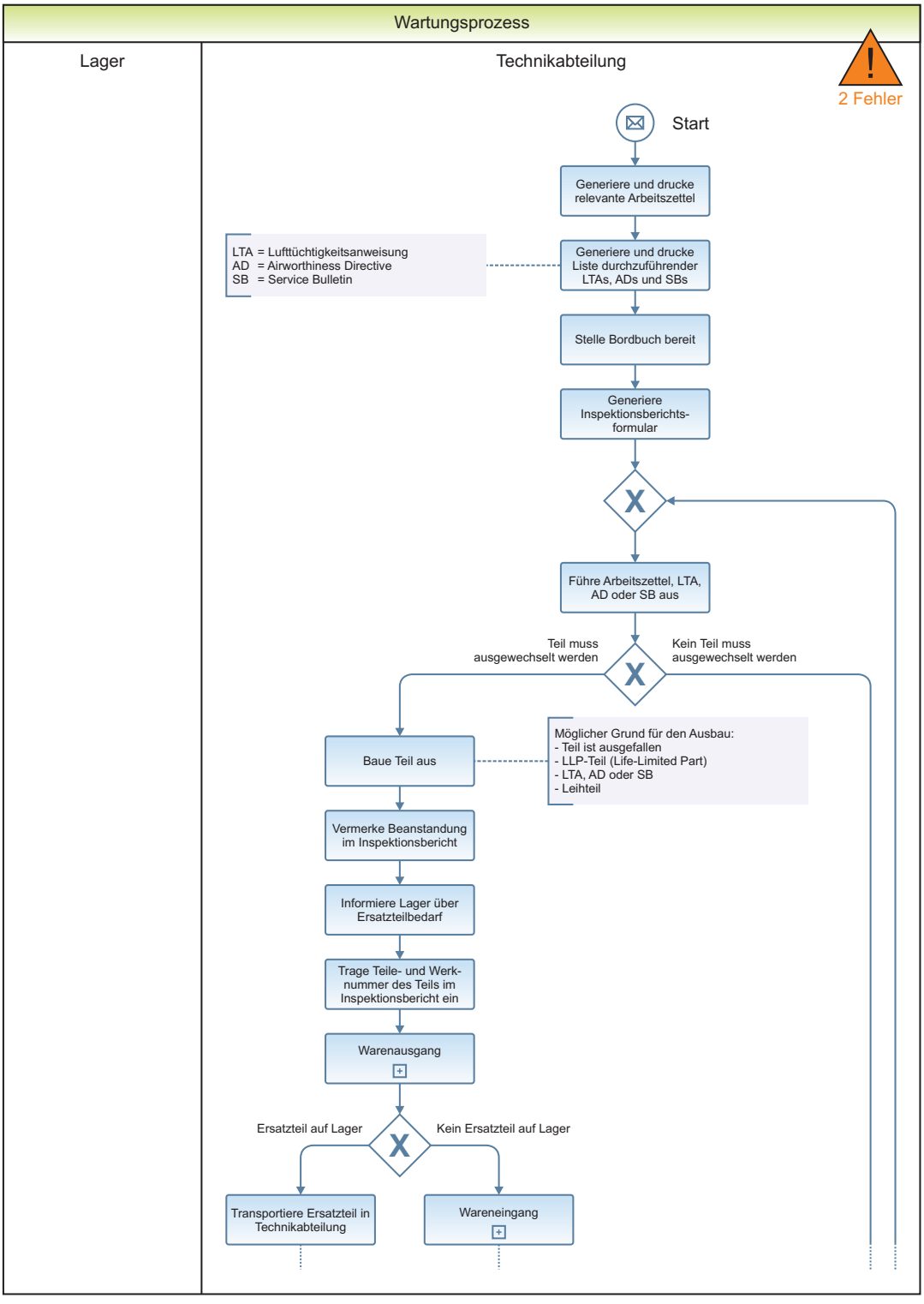


Abbildung 8.6.: Wartungsprozess

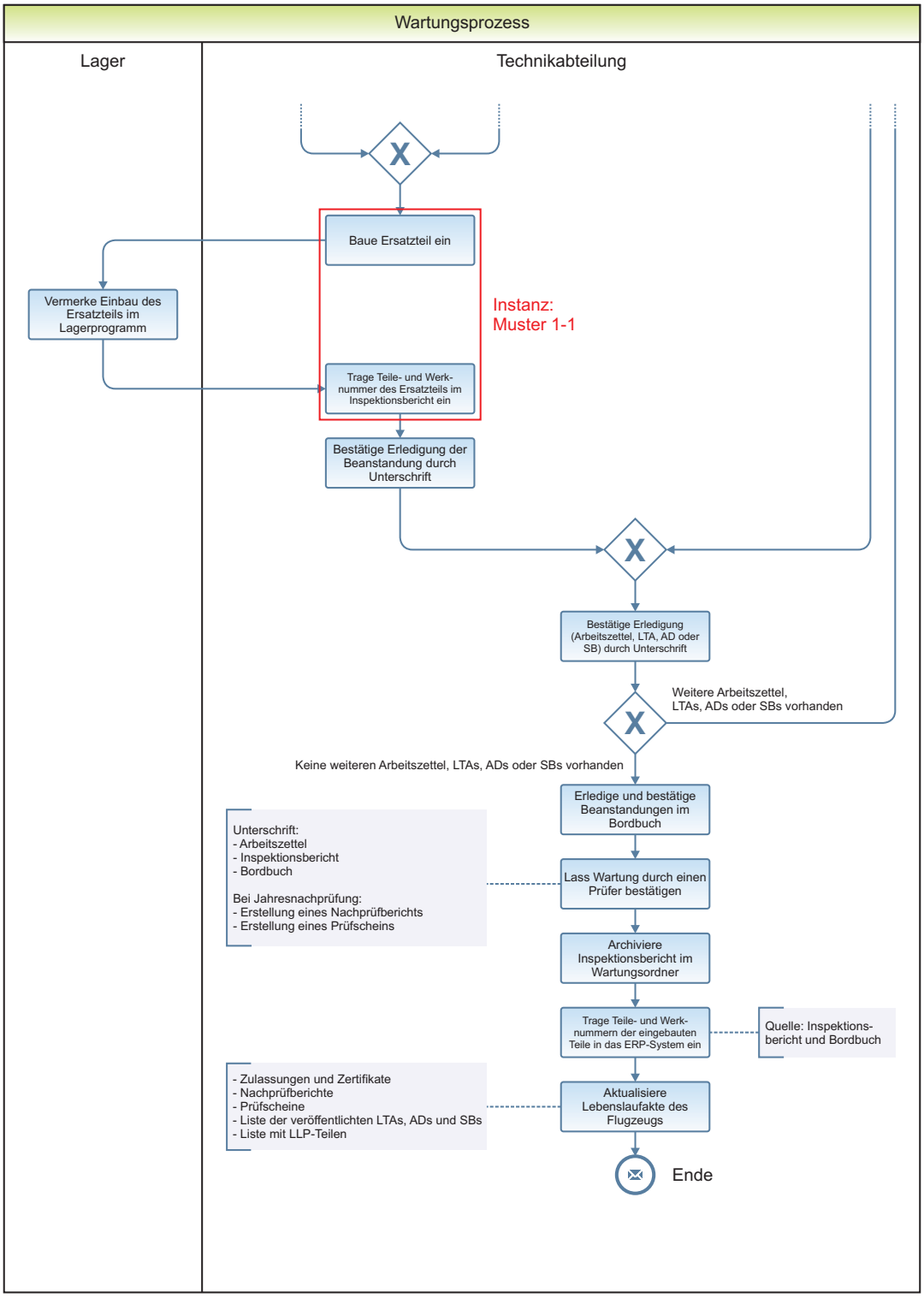


Abbildung 8.7.: Wartungsprozess (Fortsetzung)

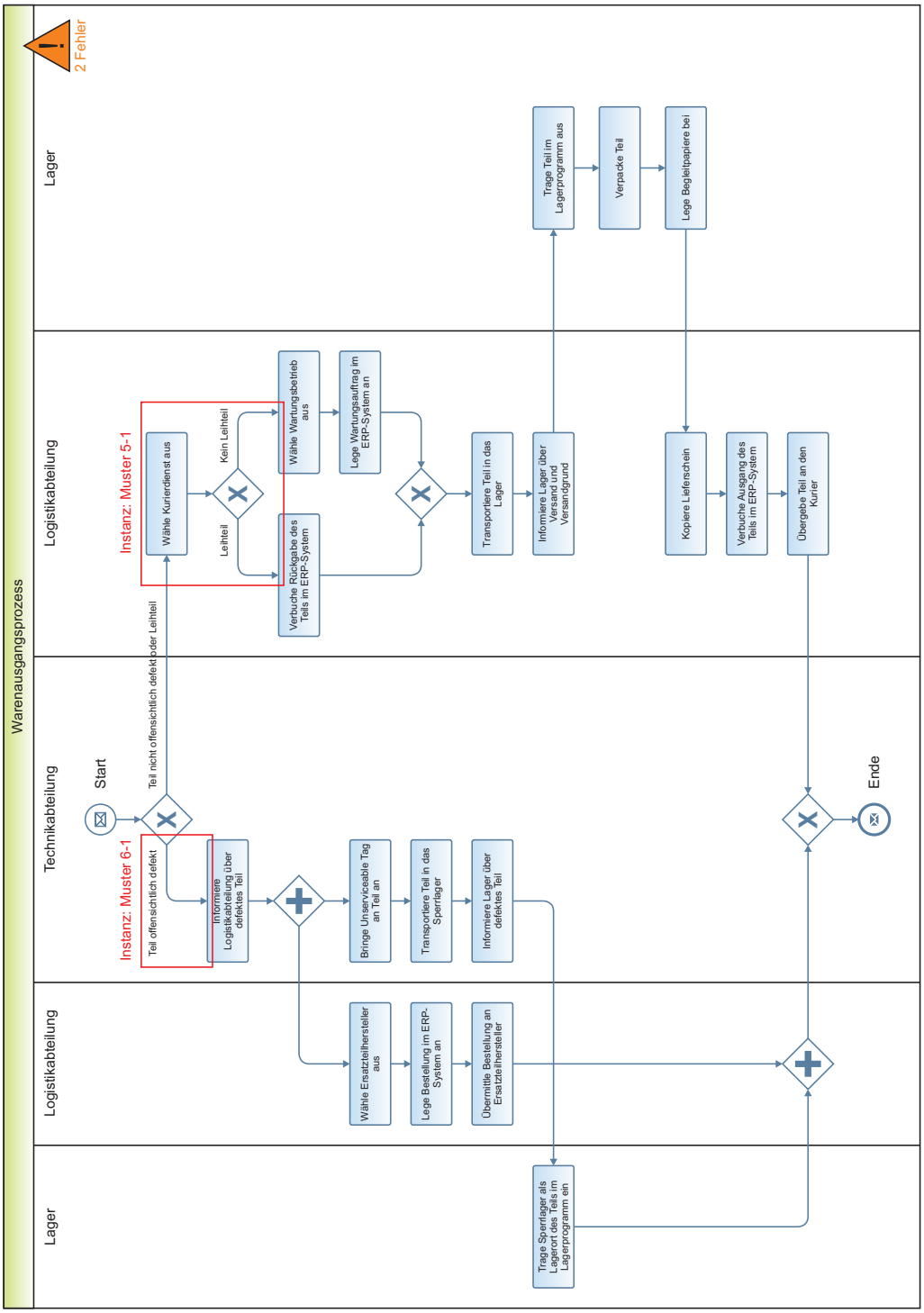


Abbildung 8.8.: Warenausgangsprozess

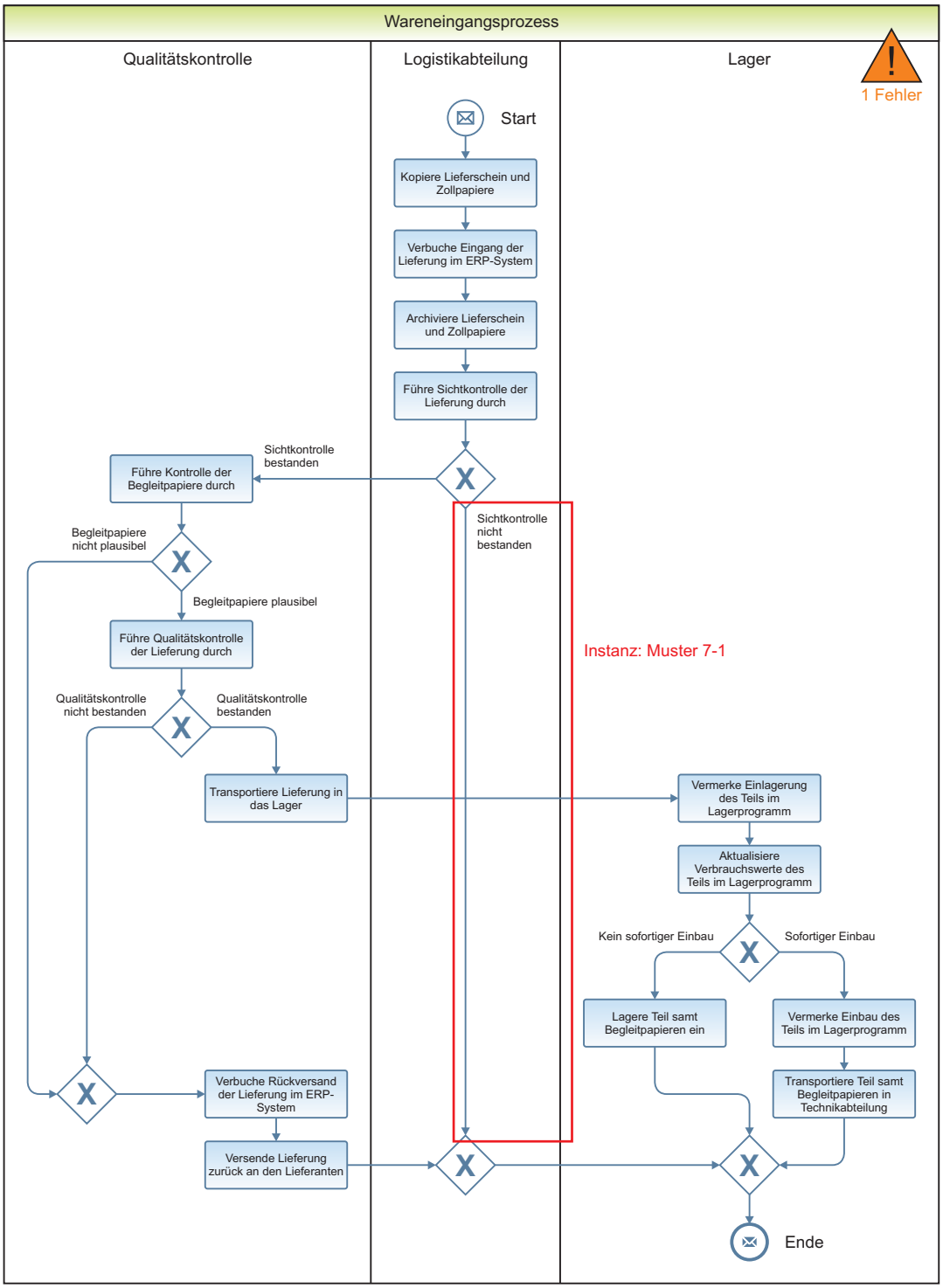


Abbildung 8.9.: Wareneingangsprozess

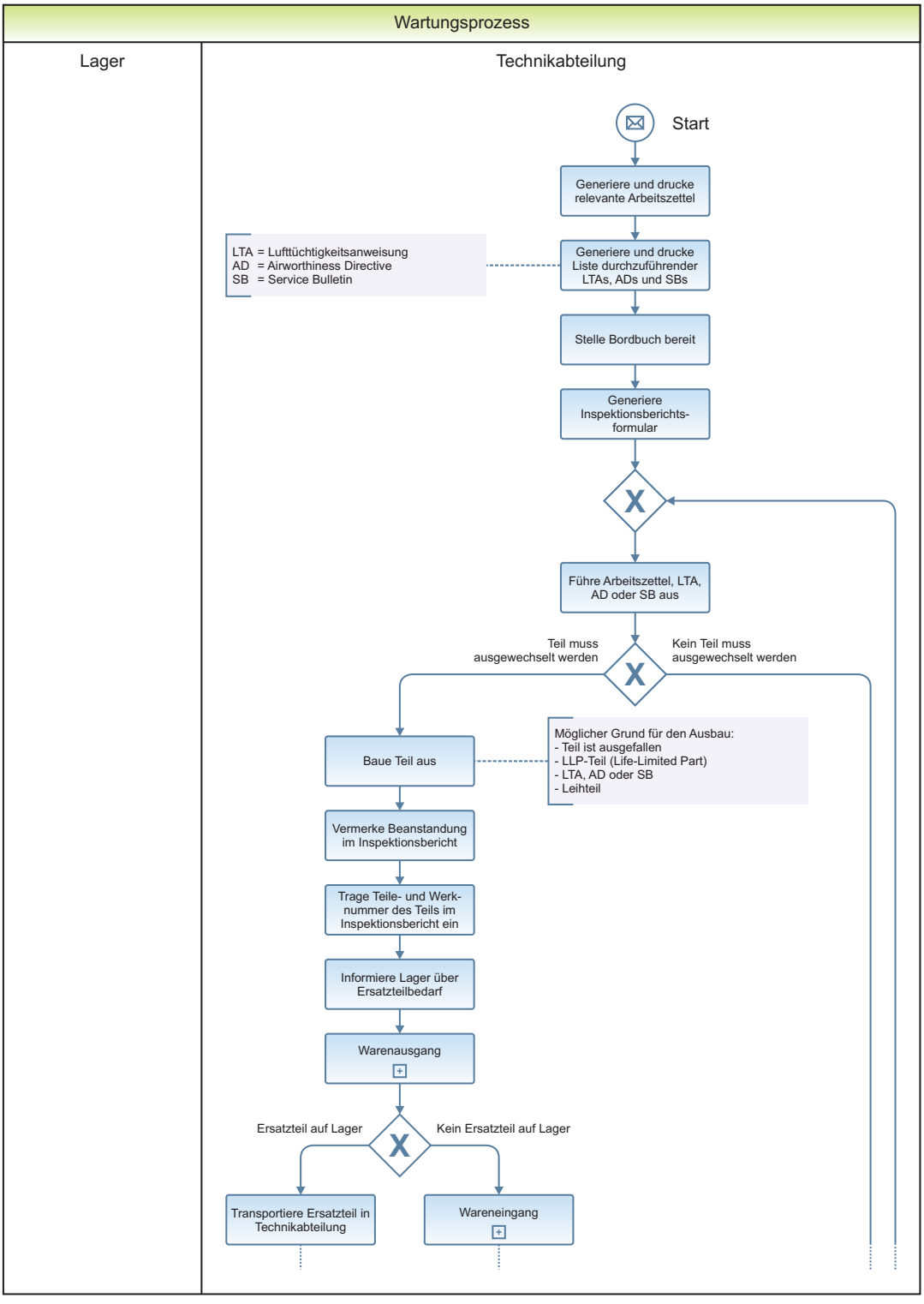


Abbildung 8.10.: Korrigierter Wartungsprozess

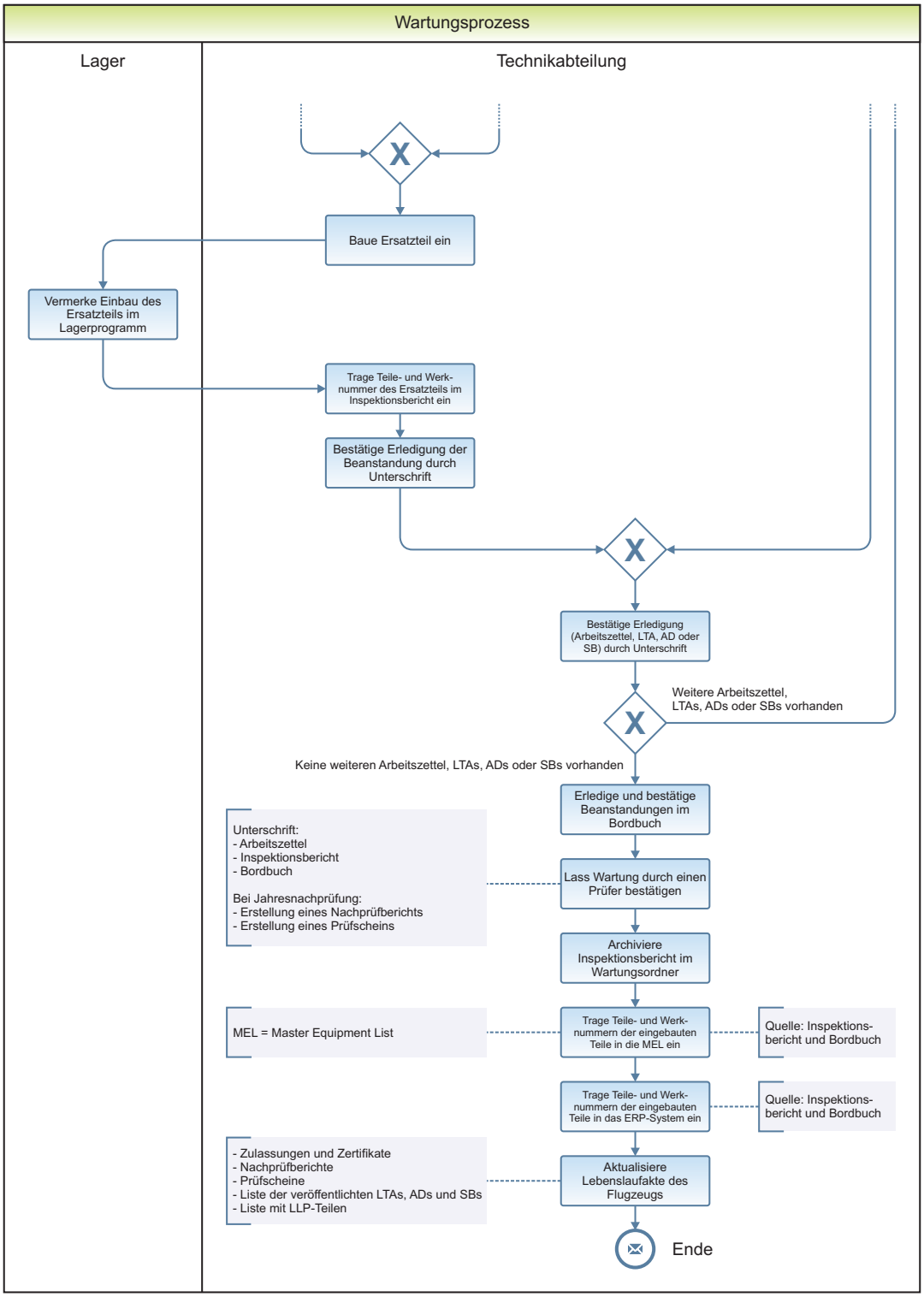


Abbildung 8.11.: Korrigierter Wartungsprozess (Fortsetzung)

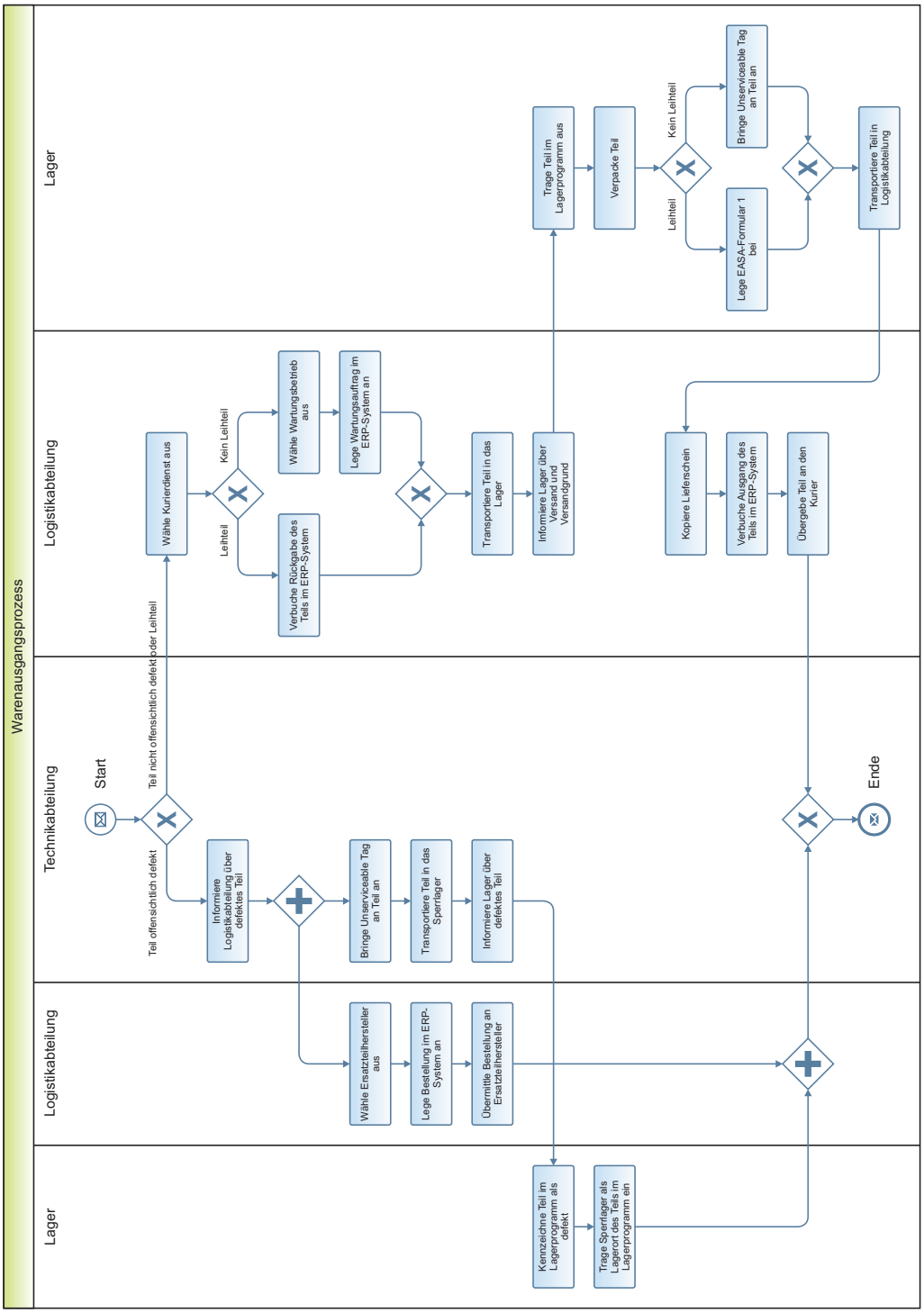


Abbildung 8.12.: Korrigierter Warenausgangsprozess

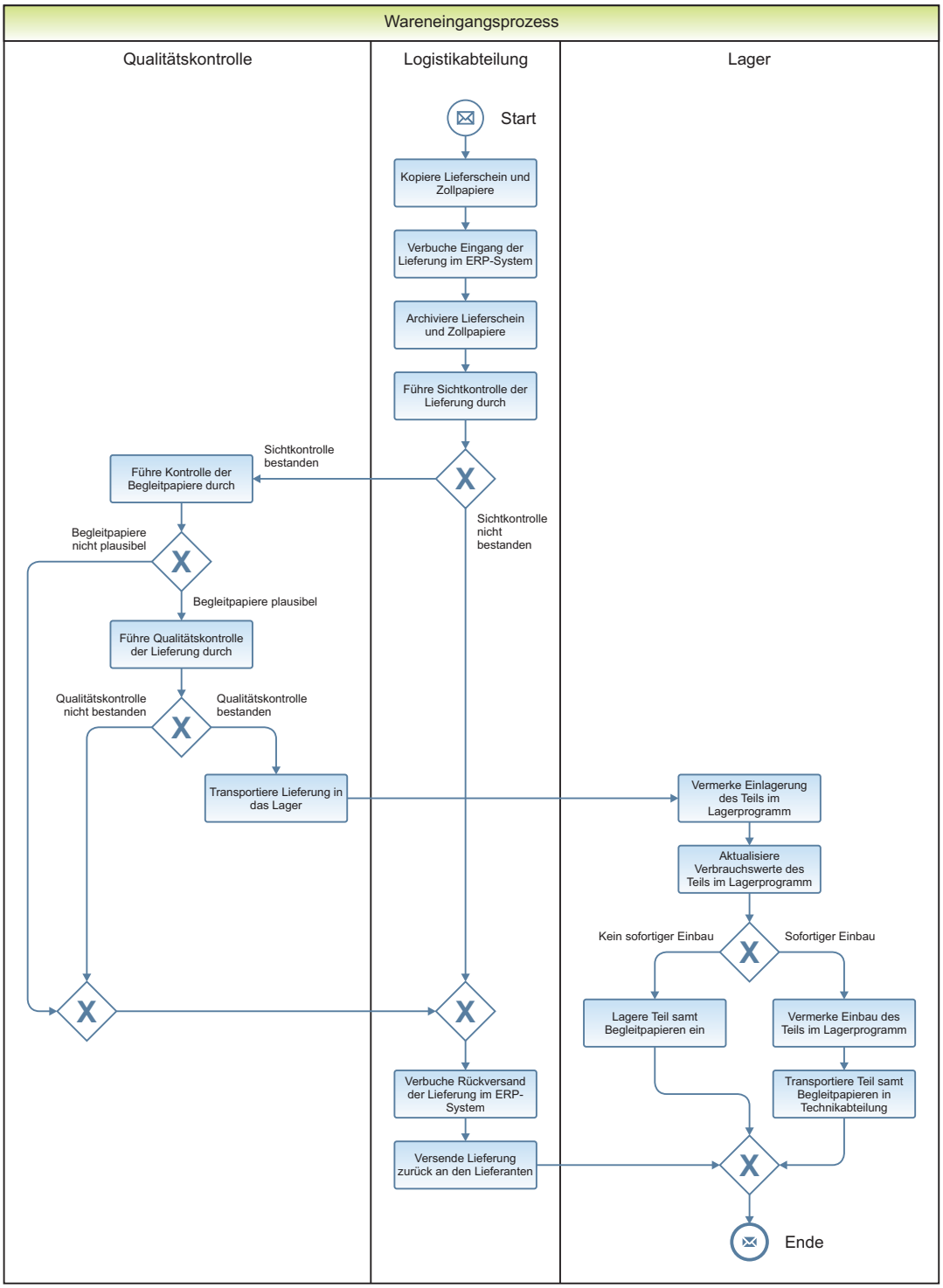


Abbildung 8.13.: Korrigierter Wareneingangsprozess

8.1.4. Adaption von Geschäftsprozessmodellen im Bereich der Flugzeugwartung

Nachdem die Einführung der RFID-Technik zur Authentifikation von Flugzeugteilen aufgrund eines internationalen Abkommens von der Geschäftsführung des luftfahrttechnischen Betriebs beschlossen wurde, muss der Business-Analyst die betroffenen Geschäftsprozessmodelle des Unternehmens entsprechend anpassen. Um die im Rahmen dieser Anpassung hinzugefügten Arbeitsschritte auf der Ontologieebene zu beschreiben, fügt der Business-Analyst der zuvor modellierten Domänenontologie (siehe Abbildung 8.2) entsprechende Klassen und Vererbungsbeziehungen hinzu (siehe Abbildung 8.14).

Abbildung 8.14.: Zusätzliche Ontologieklassen

Im Anschluss modelliert der Business-Analyst auch in diesem Fall für jede der in Abschnitt 3.4.2 beschriebenen Anforderungen jeweils einen Bedingungsausdruck und zugehörige strukturelle Muster, die sich auf die zugeordnete Semantik von Geschäftsprozessmodellen beziehen. Abbildung 8.15 zeigt für jede Anforderung den korrespondierenden Bedingungsausdruck auf Grundlage der in Listing 5.2 dargestellten BNF und die mit diesem Bedingungsausdruck verknüpften Geschäftsprozessmodelle. Die konkreten Bedingungsausdrücke und strukturellen Muster sind in Abbildung 8.16 dargestellt.

Anforderung	Korrespondierender Bedingungsausdruck	Verknüpfte Modelle
Anforderung 9	`Direktnachfolger(M9-1, M9-3) ∧ Direktnachfolger(M9-2, M9-3)`	Alle Geschäftsprozessmodelle
Anforderung 10	`Abwesenheit(M10-1)`	Wartung
Anforderung 11	`NegierteAbfolge(M11-1, M11-2)`	Wartung
Anforderung 12	`Nachfolger(M12-1, M12-3) ∧ Vorgänger(M12-3, M12-2)`	Wareneingang

Abbildung 8.15.: Anforderungen und korrespondierende Bedingungsausdrücke

Bei der Adaption von Geschäftsprozessmodellen aufgrund von Änderungen auf betriebswirtschaftlicher Ebene erfüllen Bedingungsausdrücke die Funktion, den Benutzer darauf hinzuweisen, an welchen Stellen innerhalb eines Modells entsprechende Modifikationen vorgenommen werden müssen. Zu diesem Zweck kann auf unterschiedliche Weise vorgegangen werden, beispielsweise indem mithilfe nichtnegierter Bedingungen hinzuzufügendes Prozessverhalten (siehe *Bedingungsausdruck 9* und *Bedingungsausdruck 12*) oder mithilfe negierter Bedingungen zu entfernendes Prozessverhalten (siehe *Bedingungsausdruck 10* und *Bedingungsausdruck 11*) ausgedrückt wird. Nach Abschluss der Adaption sollten Bedingungsausdrücke, die negierte Bedingungen enthalten, wieder entfernt werden, da sie zukünftig keine Funktion mehr erfüllen.

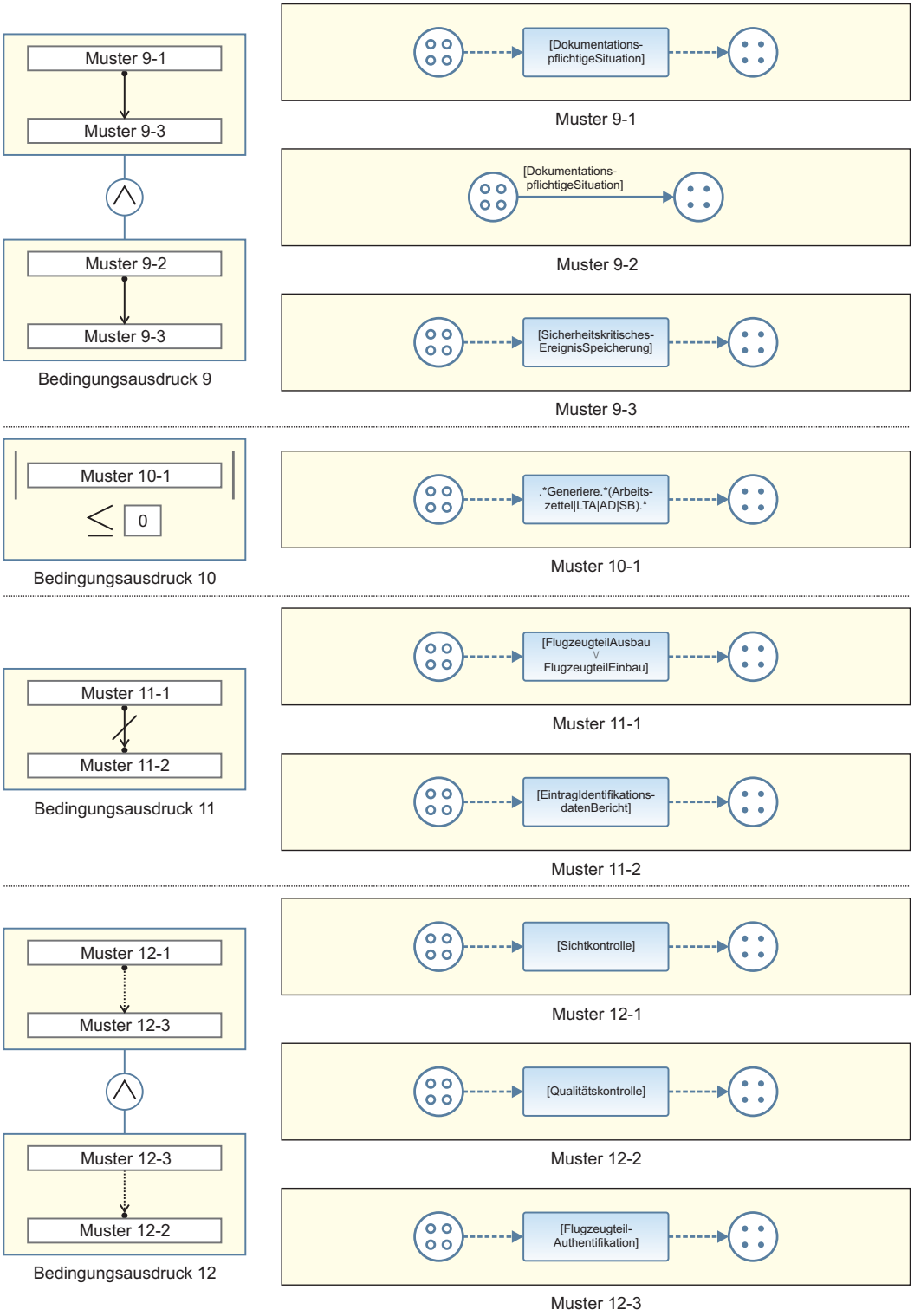


Abbildung 8.16.: Hinzugefügte Bedingungsausdrücke

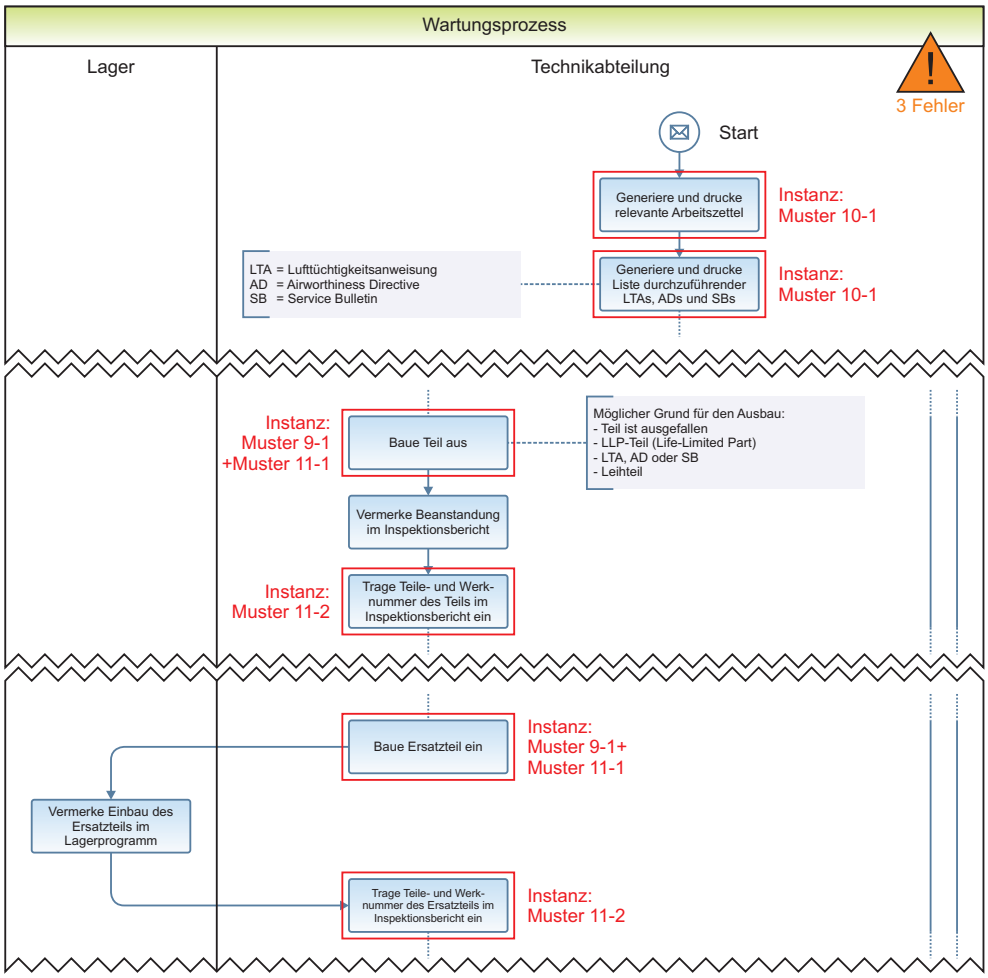


Abbildung 8.17.: Wartungsprozess (Ausschnitt)

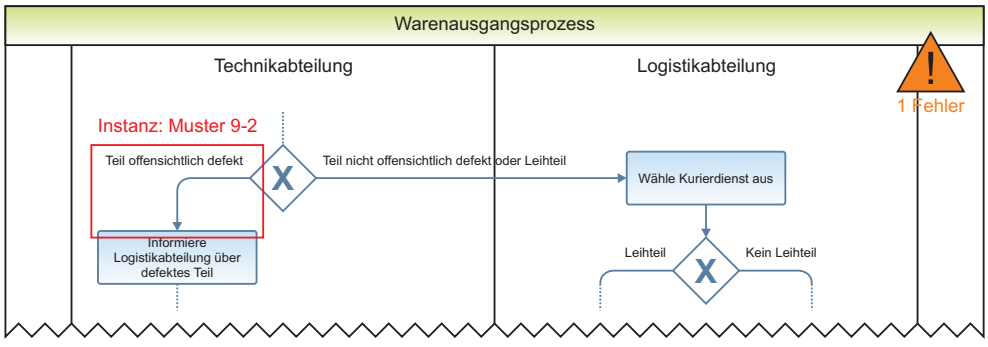


Abbildung 8.18.: Warenausgangsprozess (Ausschnitt)

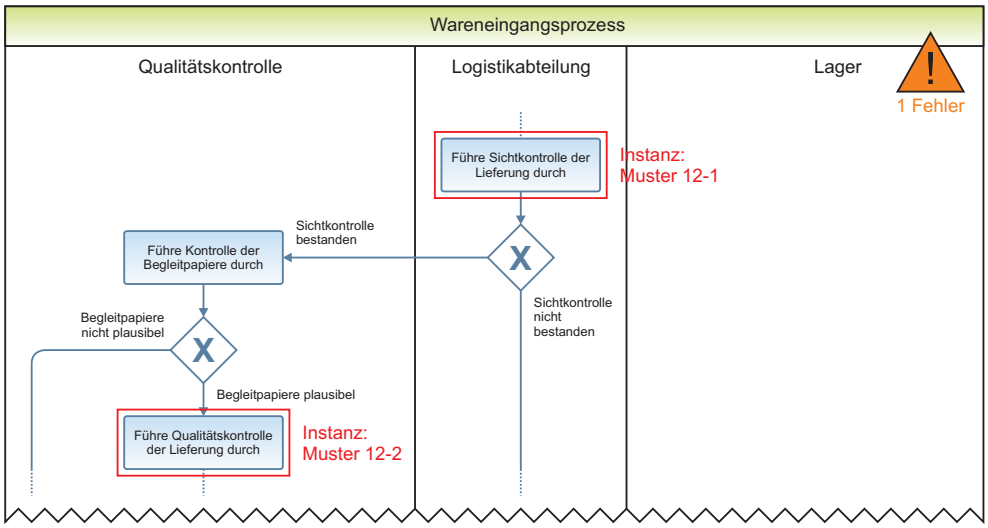


Abbildung 8.19.: Wareneingangsprozess (Ausschnitt)

Nach Verknüpfung des Wartungs-, Warenausgangs- und Wareneingangsprozessmodells mit den entsprechenden Bedingungsausdrücken wird die automatische Verifikation dieser Modelle erneut durch deren Speicherung ausgelöst. Erwartungsgemäß werden alle hinzugefügten Bedingungsausdrücke verletzt (siehe Abbildung 8.20).

BPMN-Modell	Verletzter Bedingungsausdruck	Grund
Wartungsprozess	Bedingungsausdruck 9	Das Modell enthält zwei Instanzen von Muster 9-1, auf die jeweils keine Instanz von Muster 9-3 direkt folgt.
Wartungsprozess	Bedingungsausdruck 10	Das Modell enthält zwei Instanzen von Muster 10-1.
Wartungsprozess	Bedingungsausdruck 11	Das Modell enthält drei geordnete Paare aus Instanzen von Muster 11-1 und 11-2, wobei letztere jeweils auf erstere folgen.
Warenausgangsprozess	Bedingungsausdruck 9	Das Modell enthält eine Instanz von Muster 9-2, auf die keine Instanz von Muster 9-3 direkt folgt.
Wareneingangsprozess	Bedingungsausdruck 12	Das Modell enthält jeweils eine Instanz von Muster 12-1 und Muster 12-2, auf die keine Instanz von Muster 12-3 folgt bzw. der keine Instanz von Muster 12-3 vorausgeht.

Abbildung 8.20.: Ergebnis der Verifikation

Das Wartungsprozessmodell (siehe Abbildung 8.17) verletzt drei Bedingungsausdrücke. *Bedingungsausdruck 9* wird verletzt, da den Tasks, in denen ein Flugzeugteil aus- oder eingebaut wird (Instanzen von *Muster 9-1*), kein Task folgt (d. h. keine Instanz von *Muster 9-3*), in dem ein entsprechender Datensatz, der diesen Sachverhalt dokumentiert, auf den am Flugzeugteil angebrachten RFID-Transponder gespeichert wird. *Bedingungsausdruck 10* wird verletzt, da das Modell zwei Tasks (zwei Instanzen von *Muster 10-1*) enthält, in denen Arbeitszettel, Lufttüchtigkeitsanweisungen, Airworthiness Directives und Service Bulletins gedruckt werden. *Bedingungsausdruck 11* wird verletzt, da innerhalb des Modells sowohl nach einem Ein- als auch nach einem Ausbau eines Flugzeugteils

ein Task folgt, in dem dieser Sachverhalt schriftlich dokumentiert wird. Insgesamt enthält das Modell drei geordnete Paare aus Instanzen von *Muster 11-1* und *Muster 11-2*, wobei einer Instanz von *Muster 11-1* jeweils eine Instanz von *Muster 11-2* folgt.

Auch das Warenausgangsprozessmodell (siehe Abbildung 8.18) verletzt *Bedingungsausdruck 9*, da innerhalb des Modells nach der Feststellung, dass ein Teil offensichtlich defekt ist (Instanz von *Muster 9-2*), ebenfalls kein Task folgt (d. h. keine Instanz von *Muster 9-3*), in dem ein entsprechender Datensatz, der diesen Sachverhalt dokumentiert, auf den am Flugzeugteil angebrachten RFID-Transponder gespeichert wird.

Schließlich verletzt das Wareneingangsprozessmodell (siehe Abbildung 8.19) *Bedingungsausdruck 12*, da innerhalb des Modells zwischen der Sichtkontrolle eines Flugzeugteils (Instanz von *Muster 12-1*) und dessen Qualitätskontrolle (Instanz von *Muster 12-2*) kein Task enthalten ist (d. h. keine Instanz von *Muster 12-3*), in dem der an diesem Teil angebrachte RFID-Transponder authentifiziert wird.

Anhand dieser Ergebnisse und mithilfe des modifizierten Process Composers sieht der Business-Analyst sofort, an welchen Positionen innerhalb der verifizierten Geschäftsprozessmodelle Änderungen vorgenommen werden müssen. An dieser Stelle wäre es möglich, dem Benutzer anhand verletzter Bedingungen konkrete Änderungsvorschläge zu unterbreiten, was im Rahmen dieser Arbeit allerdings nicht weiter verfolgt wurde.

8.2. Leistungsmessung

Hinsichtlich der Praxistauglichkeit einer Softwarelösung zur Unterstützung der Modellierung und Adaption von Geschäftsprozessmodellen wurde in Abschnitt 3.6 eine Anforderung formuliert, die besagt, dass eine derartige Lösung die Möglichkeit bieten muss, sowohl gesuchte Anordnungen von Modellelementen innerhalb von Geschäftsprozessmodellen zu identifizieren als auch Bedingungen, die sich auf diese Anordnungen beziehen, automatisch und innerhalb weniger Sekunden auszuwerten. In diesem Abschnitt wird daher abschließend auf den letzten Teil dieser Anforderung eingegangen und die Leistungsfähigkeit des entwickelten Prototyps validiert.

8.2.1. Vorgehensweise

Zur Messung der Leistung des entwickelten Prototyps wurde getestet, wie viel Zeit ein Bürocomputer der gehobenen Mittelklasse für die Verifikation der mit Bedingungsausdrücken verknüpften Geschäftsprozessmodelle des Szenarios benötigt. Zu diesem Zweck wurde ein Testprogramm entwickelt, das die Verifikation des Wartungs-, des Warenausgangs- und des Wareneingangsprozessmodells anstößt, mehrere Zeiten misst und diesen Vorgang entsprechend einer vorgegebenen Anzahl wiederholt, um die durchschnittliche Verifikationsdauer mithilfe des arithmetischen Mittels zu berechnen. Dabei wurden diese

Messungen für das Wartungs-, das Warenausgangs- und das Wareneingangsprozessmodell nach deren Modellierung (siehe Abbildung 8.6-8.7, Abbildung 8.8 und Abbildung 8.9) bei Verknüpfung mit Bedingungsausdruck 1 bis 8 und nach deren Korrektur (siehe Abbildung 8.10-8.11, Abbildung 8.12 und Abbildung 8.13) bei Verknüpfung mit Bedingungsausdruck 9 bis 12 durchgeführt, wobei jeweils Drools und KAON2 zur musterbasierten Suche verwendet wurden, um diese Ansätze miteinander zu vergleichen. Insgesamt wurden dementsprechend zwölf Testdurchläufe mit jeweils 20 Wiederholungen durchgeführt. Pro Durchlauf wurde jeweils die Zeit zur Suche nach Instanzen struktureller Muster und die Zeit zur Auswertung der verknüpften Bedingungsausdrücke gemessen. Bei Verwendung konjunktiver Anfragen zur musterbasierten Suche wurde darüber hinaus die Zeit zur Transformation des zu verifizierenden Geschäftsprozessmodells in eine OWL-Ontologie ermittelt. Da die Speicherung von Fehlerinformationen am Ende einer Verifikation (siehe Abschnitt 7.4.1) zu unregelmäßigen Ergebnissen führte, wurde diese Funktion im Rahmen der Leistungsmessung deaktiviert.

8.2.2. Testsystem

Die nachfolgenden Ergebnisse der Leistungsmessung wurden auf einem Testsystem durchgeführt, dessen technische Daten zum Zeitpunkt der Fertigstellung der vorliegenden Arbeit von der Rechenleistung her den Daten eines Bürocomputers der gehobenen Mittelklasse entsprachen (siehe Abbildung 8.21).

Hard-/Softwarekomponente	
Hardware: Prozessor	Intel Core 2 Duo E8500 (3,16 GHz)
Hardware: Arbeitsspeicher	Kingston KHX6400D2K2/4G (4 GiB, DDR2-800)
Hardware: Mainboard	Gigabyte GA-EP35-DS4
Hardware: Festplatte	Intel X25-M SATA Solid State Drive (160 GB)
Software: Betriebssystem	Windows 7 Professional (64-Bit-Version)
Software: Java-Version	JDK 6 Update 20
Software: IDE	SAP NetWeaver Developer Studio 7.1 (SAP EHP 1 for SAP NetWeaver Developer Studio 7.1)
Sonstige Software	Drools 4.0.7, KAON2, Microsoft Visual C++ 2010 Express, Pellet 2.0.0, Spin 5.2.5

Abbildung 8.21.: Hard- und Softwarekomponenten des Testsystems

8.2.3. Ergebnisse

Zur besseren Erklärung der Ergebnisse der Leistungsmessung zeigt Abbildung 8.22 eine Übersicht, die für jedes untersuchte Geschäftsprozessmodell die Anzahl struktureller

Muster, die von musterbasierten Bedingungen innerhalb der jeweils verknüpften Bedingungsausdrücke referenziert werden, die Anzahl gefundener Instanzen dieser strukturellen Muster, die Anzahl auszuwertender Bedingungen und die Anzahl durchzuführender Modellprüfungen darstellt. Beim nicht korrigierten Warenausgangsprozessmodell und beim korrigierten Wareneingangsprozessmodell ist anzumerken, dass trotz auszuwertender Nachfolger- und Vorgänger-Bedingungen keine Modellprüfung erforderlich ist, da aufgrund fehlender Instanzen der in diesem Zusammenhang relevanten strukturellen Muster bereits im Vorfeld klar ist, dass diese Bedingungen verletzt werden.

Geschäftsprozessmodell	Anzahl referenzierter struktureller Muster	Anz. gefundener Instanzen struktureller Muster	Anz. auszuwertender musterbasierter Bedingungen	Anz. durchzuführender Modellprüfungen
Wartung	6	5	4	1
Warenausgang	3	2	2	0
Wareneingang	4	10	2	1
Wartung (korrigiert)	6	8	3	1
Warenausgang (korrigiert)	3	1	1	0
Wareneingang (korrigiert)	6	2	2	0

Abbildung 8.22.: Verifikationsdetails

Die Verifikation des Wartungsprozessmodells (siehe Abbildung 8.23), des Warenausgangsprozessmodells (siehe Abbildung 8.24) und des Wareneingangsprozessmodells (siehe Abbildung 8.25) dauerte bei Verwendung des Drools-Ansatzes zwischen 0,34 s und 0,54 s, bei Verwendung konjunktiver Anfragen zwischen 0,07 s und 0,35 s. Die Verifikation des korrigierten Wartungsprozessmodells (siehe Abbildung 8.26), des korrigierten Warenausgangsprozessmodells (siehe Abbildung 8.27) und des korrigierten Wareneingangsprozessmodells (siehe Abbildung 8.28) dauerte bei Verwendung des Drools-Ansatzes zwischen 0,10 s und 0,50 s, bei Verwendung konjunktiver Anfragen zwischen 0,09 s und 0,39 s. In Bezug auf die Dauer der musterbasierten Suche schneidet die Verwendung konjunktiver Anfragen (KAON2) trotz der notwendigen Transformation des zu durchsuchenden BPMN-Modells in eine OWL-Ontologie durchgehend besser ab, wenngleich teilweise nur geringfügig. Die relativ lange Dauer der musterbasierten Suche beim Drools-Ansatz im sechsten Durchlauf der Verifikation des Warenausgangsprozessmodells und bei Verwendung konjunktiver Anfragen im zwölften Durchlauf der Verifikation des korrigierten Warenausgangsprozessmodells ist vermutlich auf eine *automatische Speicherbereinigung* (Garbage Collection) zurückzuführen. Anhand der Diagramme ist auch deutlich zu erkennen, wann eine Modellprüfung durchgeführt wurde (orangefarbene Säulen).

Anhand dieser Ergebnisse wird deutlich, dass die Auswertung von Bedingungsausdrücken mithilfe des im Rahmen dieser Arbeit entwickelten Prototyps trotz der vielen zu diesem Zweck benötigten Schritte keine nennenswerten Wartezeiten bei der Modellierung von Geschäftsprozessmodellen verursacht, was ein wichtiges Kriterium für die Akzeptanz einer derartigen Lösung darstellt.

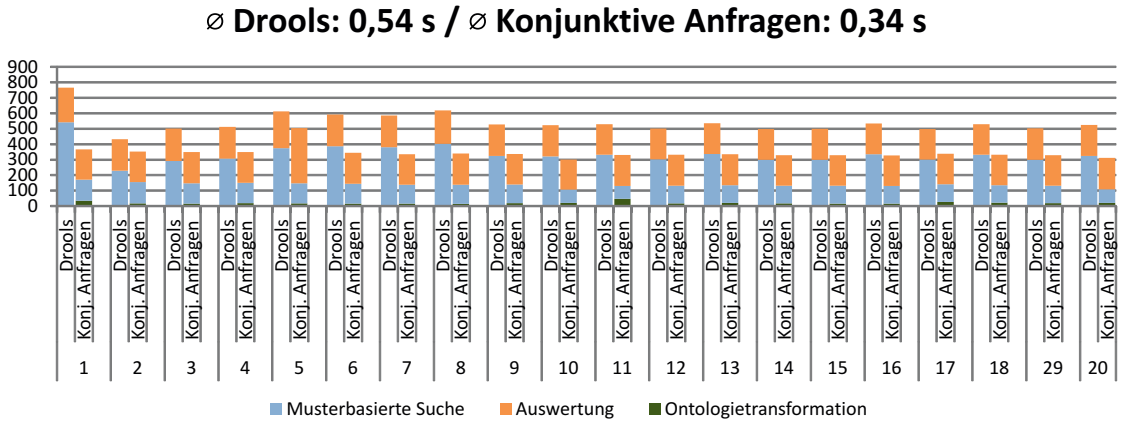


Abbildung 8.23.: Verifikationsdauer des Wartungsprozessmodells

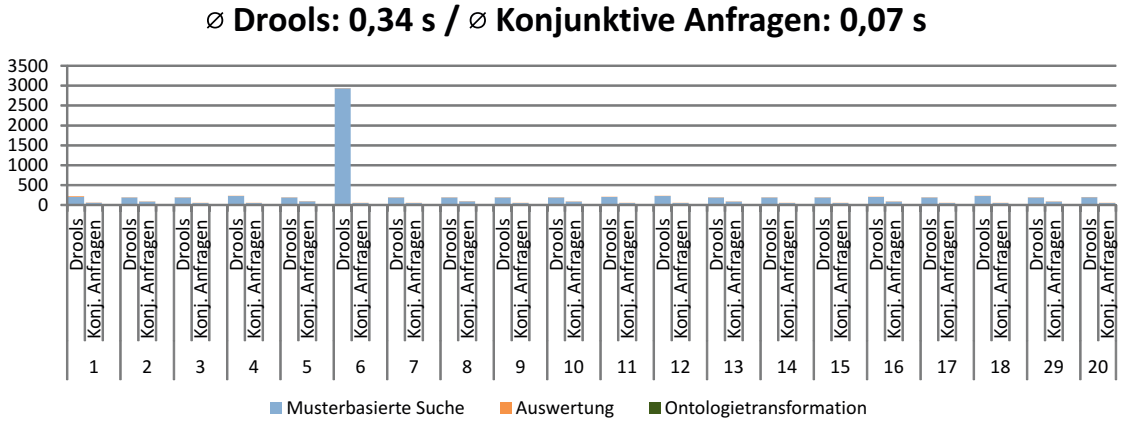


Abbildung 8.24.: Verifikationsdauer des Warenausgangsprozessmodells

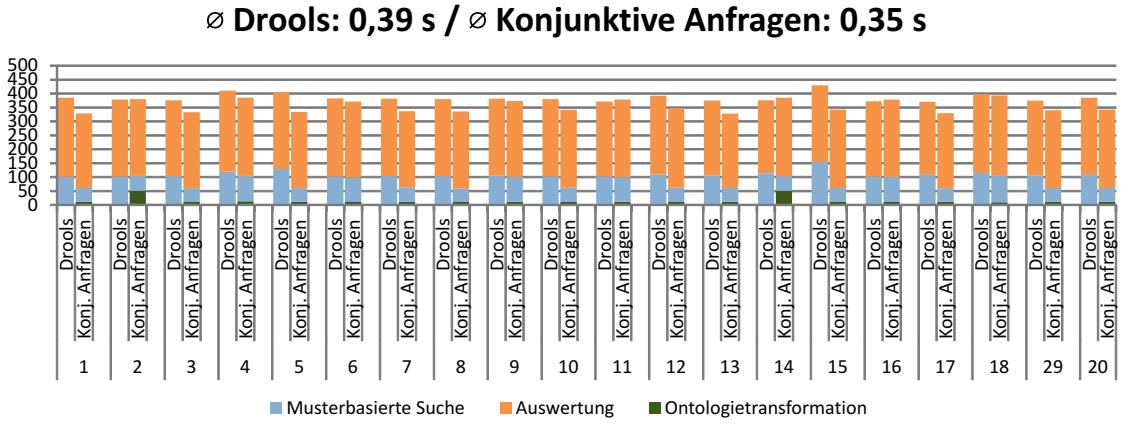


Abbildung 8.25.: Verifikationsdauer des Wareneingangsprozessmodells

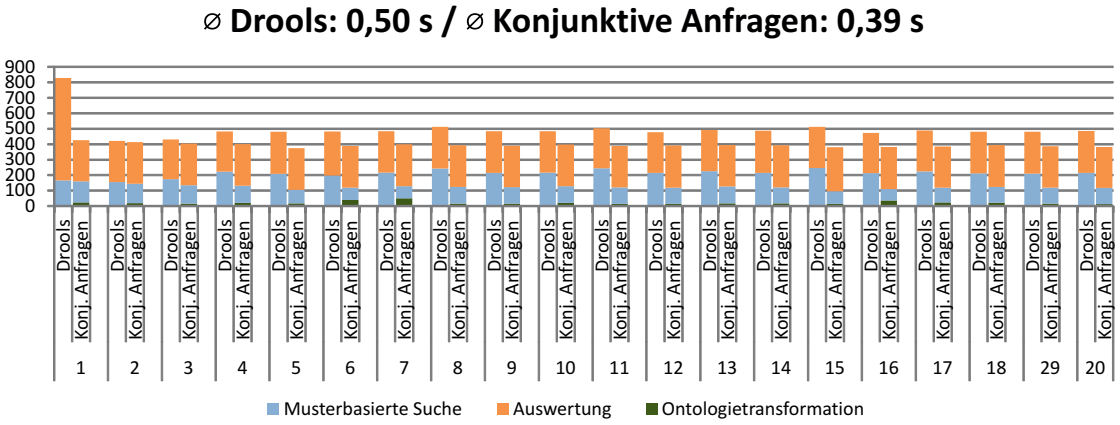


Abbildung 8.26.: Verifikationsdauer des korrigierten Wartungsprozessmodells

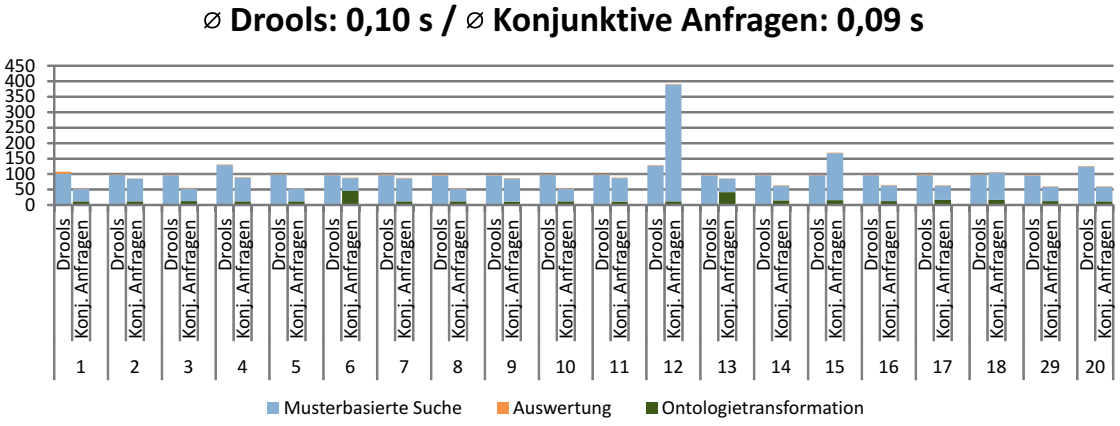


Abbildung 8.27.: Verifikationsdauer des korrigierten Warenausgangsprozessmodells

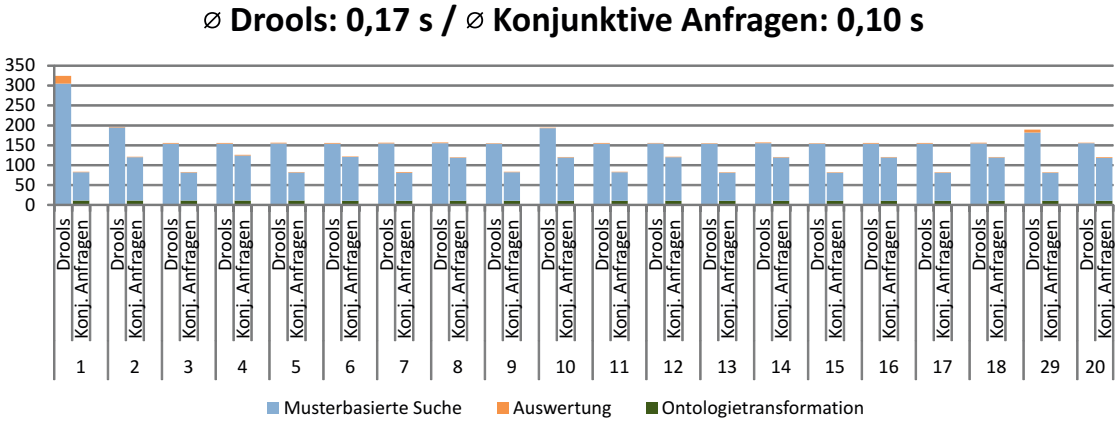


Abbildung 8.28.: Verifikationsdauer des korrigierten Wareneingangsprozessmodells

8.3. Externe Veröffentlichung und interne Verwertung der Ergebnisse

Ausgewählte Ergebnisse der vorliegenden Arbeit wurden in Form wissenschaftlicher Veröffentlichungen publiziert [Mül09a, Mül09b] und auf wissenschaftlichen Konferenzen präsentiert. Darüber hinaus wurden Teilergebnisse dieser Arbeit durch Studenten, die vom Verfasser dieser Arbeit betreut wurden, in Form einer Diplomarbeit [Sch08] (Note: 1,0) und einer Bachelorarbeit [Gra09] (Note: 1,2) weiter vertieft.

Innerhalb der Firma SAP findet der Wissensaustausch zwischen der Forschungsabteilung und den jeweiligen Produktgruppen im Rahmen interner Transferprojekte statt. Die im Rahmen dieser Projekte geleistete Arbeit der Forschungsabteilung wird entsprechend abgerechnet. Ein Transferprojekt besteht normalerweise aus mehreren Arbeitspaketen. Zu Beginn eines Projekts werden zunächst die innerhalb der Arbeitspakete zu verrichtenden Aufgaben, Meilensteine und der Zeitplan festgelegt. Das Ziel eines Transferprojekts ist beispielsweise eine lauffähige Implementierung innovativer Konzepte inklusive deren schriftlicher Dokumentation. Am Ende eines Transferprojekts werden die Ergebnisse der Forschungsabteilung von der jeweiligen Produktgruppe bewertet.

Auch die im Rahmen der vorliegenden Arbeit beschriebene Erweiterung des Process Composers wurde im Rahmen eines Transferprojekts in Kooperation mit der für die Geschäftsprozessmanagement-Komponente der NetWeaver-Plattform verantwortlichen Produktgruppe entwickelt und umfasst mehr als 25.000 Zeilen Quellcode. Darüber hinaus wurde gemäß unternehmensinterner Richtlinien eine englischsprachige Dokumentation der in der vorliegenden Arbeit vorgestellten Konzepte, die der entwickelten Erweiterung zugrunde liegen, erstellt. Die entwickelte Erweiterung bietet zum ersten Mal die Möglichkeit, Geschäftsprozessmodelle auf Einhaltung betriebswirtschaftlicher Anforderungen der in dieser Arbeit untersuchten Kategorie auf Grundlage eines kommerziellen Modellierungswerkzeugs zu verifizieren.

Am Ende des Transferprojekts wurden die entwickelten Konzepte und die darauf basierende Erweiterung dem Management der Produktgruppe präsentiert. Insgesamt wurde das Transferprojekt, das mehrere Arbeitspakete umfasste und an dem auch andere Kollegen beteiligt waren, mit der Bestnote ausgezeichnet.

Über diese Aktivitäten hinaus wurde ein Großteil der in dieser Arbeit vorgestellten Konzepte in den Vereinigten Staaten von Amerika von der Firma SAP zum Patent angemeldet („Evaluating Pattern-Based Constraints on Business Process Models", Application Serial Number 12/939,026). Zum Zeitpunkt der Einreichung der vorliegenden Arbeit stand die Entscheidung noch aus, diese Konzepte auch in Deutschland und weiteren europäischen Ländern als Patent anzumelden.

9. Fazit

In diesem Kapitel werden die im Rahmen dieser Arbeit erzielten Ergebnisse und deren wissenschaftlicher Beitrag zusammengefasst. Schließlich wird auf offene Forschungsfragen hingewiesen, die im Kontext dieser Arbeit nicht bearbeitet werden konnten.

9.1. Zusammenfassung und wissenschaftlicher Beitrag

Das Ziel der vorliegenden Arbeit war die Entwicklung von Konzepten und eines darauf basierenden Prototyps zur Modellierung und automatischen Auswertung betriebswirtschaftlicher Anforderungen, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung machen. Die grundsätzlichen Voraussetzungen an eine derartige Lösung wurden auf Grundlage eines Szenarios aus der Luftfahrtindustrie abgeleitet. Die entwickelten Konzepte wurden verallgemeinert und sind daher nicht an eine spezifische Anwendungsdomäne gebunden. Die Implementierung der entwickelten Konzepte stellt eine Erweiterung der für das Geschäftsprozessmanagement zuständigen Komponente der NetWeaver-Plattform der Firma SAP dar.

Die in dieser Arbeit vorgeschlagene Lösung setzt bereits bei der Modellierung von Geschäftsprozessen an. In diesem Kontext wurden Konzepte erarbeitet, welche die Anreicherung der Elemente eines auf MOF basierenden BPMN-Modells mit maschinenlesbarer Semantik erlauben und eine präzisere Analyse des Modells ermöglichen, wovon im Rahmen der automatischen Verifikation von BPMN-Modellen profitiert werden kann. Zunächst wurde aufgrund augenscheinlicher Ähnlichkeiten der Frage nachgegangen, ob ein MOF-Metamodell eine OWL-Ontologie verkörpern kann. Nachdem gezeigt wurde, dass dies nicht möglich ist, wurde unter Verwendung des Ontology-Definition-Metamodells eine Vorgehensweise zur semantischen Anreicherung vorgeschlagen, die im Vergleich zu bisherigen Ansätzen vollständig auf Grundlage von MOF realisiert ist. Die Vorteile dieser Vorgehensweise bestehen zum einen in der klaren Trennung zwischen Geschäftsprozessmodellen und den als MOF-Modellen vorliegenden Domänenontologien, zum anderen in den von MOF-Repositorien zur Verfügung gestellten Diensten. Darüber hinaus wurde ein Algorithmus skizziert, der eine Transformation von Ontologien auf Basis des OWL-Metamodells in ein Objektmodell auf Grundlage der OWL API ermöglicht, das der zur Klassifikation der Ontologie verwendete Reasoner benötigt.

Anforderungen an Geschäftsprozessmodelle, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung

machen, werden im Rahmen der in dieser Arbeit vorgeschlagenen Lösung in Form struktureller Muster und musterbasierter Bedingungen innerhalb von Bedingungsausdrücken modelliert. Zu diesem Zweck wurden zwei entsprechende graphische Modellierungssprachen auf Grundlage von MOF vorgeschlagen: PPML und PCML. Die Entwicklung von Modellierungssprachen auf Grundlage einer *graphischen* Notation zielte darauf ab, eine Modellierungsmethode zur Verfügung zu stellen, die auch weniger technisch versierten Benutzern zugänglich ist. Während anhand struktureller Muster gesuchte Anordnungen von Modellelementen mit einer bestimmten Semantik beschrieben werden können, können mithilfe musterbasierter Bedingungen Aussagen über die Existenz gesuchter Anordnungen oder Aussagen über die temporale Beziehung zwischen verschiedenen Anordnungen modelliert werden. Bedingungen können wiederum in Form von Bedingungsausdrücken logisch verknüpft werden. Die Modellierungskonstrukte von PPML und PCML orientieren sich an den Bedürfnissen des Szenarios, erheben jedoch keinen Anspruch auf Vollständigkeit und sind als Ausgangspunkt für zukünftige Erweiterungen anzusehen. Bei der abschließenden Gegenüberstellung der entwickelten Modellierungssprachen mit vergleichbaren Ansätzen wurden strukturelle Muster zunächst vom Begriff des Entwurfsmusters abgegrenzt. Als Vorteile von PPML im Vergleich zu Ansätzen, die es ermöglichen, Geschäftsprozessmodelle mittels einer graphischen Notation abzufragen, wurden die höhere Ausdrucksmächtigkeit und die zugrunde liegenden Industriestandards angeführt. Zudem wurde darauf hingewiesen, dass keiner der bestehenden Ansätze darauf abzielt, strukturelle Muster als Grundlage der Spezifikation von Bedingungen zu verwenden. In Bezug auf PCML wurde verdeutlicht, dass sich vergleichbare Ansätze zur graphischen Spezifikation von Eigenschaften auf die Laufzeitsemantik von Geschäftsprozessmodellen beziehen, im Gegensatz zu PCML jedoch nicht auf deren Struktur.

Zur Verifikation eines BPMN-Modells in Bezug auf einen Bedingungsausdruck wurde eine dreistufige Vorgehensweise konzipiert. In diesem Kontext wird das BPMN-Modell zunächst nach Instanzen struktureller Muster durchsucht, die von musterbasierten Bedingungen innerhalb des Bedingungsausdrucks referenziert werden. Zu diesem Zweck wurde eine grundsätzliche Methode präsentiert, die auf einer Integration bestehender Werkzeuge beruht, deren Mechanismen zur musterbasierten Suche geeignet sind und deren Besonderheit darin liegt, dass die jeweilige Sprache zur Spezifikation der Eingabedaten einen deklarativen Ansatz verfolgt. Auf Grundlage dieses gemeinsamen Merkmals wurden zwei bisher noch nicht erforschte Verfahren vorgestellt, die ein PPML-Modell in die vom jeweiligen Werkzeug benötigte Eingabedatenrepräsentation transformieren. Zur deklarativen Repräsentation des Suchproblems und zur musterbasierten Suche verwendet das erste Verfahren Produktionsregeln bzw. das regelbasierte System Drools, das zweite Verfahren konjunktive Anfragen bzw. den Reasoner KAON2. In beiden Fällen wurde gezeigt, wie Instanzen der verschiedenen PPML-Modellierungskonstrukte in den entsprechenden Teil einer Produktionsregel oder Abfrage transformiert werden. Gegenüber vergleichbaren Ansätzen wurde hervorgehoben, dass keiner dieser Ansätze eine Abbildung aller PPML-Modellierungskonstrukte ermöglicht und die Voraussetzungen zur Auswertung musterbasierter Bedingungen auf Grundlage struktureller Muster erfüllt.

Nach Abschluss der musterbasierten Suche werden die Bedingungen innerhalb des Bedingungsausdrucks und der Bedingungsausdruck selbst ausgewertet. In diesem Zusammenhang wurden einige einfache Algorithmen zur Auswertung existenzieller Bedingungen präsentiert. Im Gegensatz dazu wurde zur Auswertung temporaler Bedingungen ein aufwendigeres Verfahren auf Grundlage des Modellprüfers Spin vorgestellt. Unter Berücksichtigung der auszuwertenden Bedingung und des Resultats der musterbasierten Suche wird im Rahmen dieses Verfahrens das zu verifizierende Geschäftsprozessmodell in eine Systembeschreibung transformiert, die in Form eines PROMELA-Programms vorliegt. Zur Erläuterung der Funktionsweise des zu diesem Zweck entwickelten Algorithmus wurde gezeigt, wie Instanzen der im Rahmen dieser Arbeit berücksichtigten BPMN-Modellierungskonstrukte in den entsprechenden Teil eines PROMELA-Programms transformiert werden. Des Weiteren wurde für jeden Bedingungstyp eine entsprechende LTL-Formel angegeben. Darüber hinaus wurden Optimierungsmaßnahmen bei der Generierung von PROMELA-Programmen erläutert. Schließlich wurde das vorgestellte Verfahren von rein laufzeitbasierten Ansätzen zur Überprüfung gewisser Eigenschaften von Geschäftsprozessmodellen abgegrenzt und herausgestellt, das bisher kein Verfahren zur Überprüfung von Geschäftsprozessmodellen auf die Einhaltung gesetzlicher, vertraglicher oder unternehmensinterner Richtlinien existiert, das strukturelle Aspekte mit Laufzeitaspekten in Verbindung bringt.

Zu guter Letzt wurden die in dieser Arbeit vorgestellten Konzepte auf Grundlage des am Anfang beschriebenen Szenarios aus der Luftfahrtindustrie validiert. Zu diesem Zweck wurde gezeigt, dass die auf diesen Konzepten basierende Implementierung in der Lage ist, die Anforderungen des Szenarios an eine Softwarelösung zur Unterstützung der Modellierung und Adaption von Geschäftsprozessmodellen zu erfüllen. Im Rahmen der Validierung wurden zunächst drei Geschäftsprozesse modelliert. Anschließend wurden die entsprechenden Geschäftsprozessmodelle aufgrund von Änderungen auf betriebswirtschaftlicher Ebene adaptiert. In beiden Fällen wurden diese Modelle mithilfe der entwickelten Softwarelösung und den jeweils verknüpften Bedingungsausdrücken automatisch verifiziert und darin enthaltene Fehler ausfindig gemacht. Darüber hinaus wurde der entwickelte Prototyp einer Leistungsmessung unterzogen. Das Ergebnis dieser Messung ergab, dass die zur automatischen Verifikation benötigte Zeit vernachlässigbar ist.

Abschließend ist zu sagen, dass die automatische Erkennung von Modellierungsfehlern der in dieser Arbeit untersuchten Kategorie naturgemäß von der Qualität der Geschäftsprozessmodelle selbst als auch von der Qualität modellierter musterbasierter Bedingungen und struktureller Muster abhängt. Dazu trägt die Verwendung einer standardisierten Domänenontologie bei, zumindest aber eine einheitliche Benennung von Tasks bei Verzicht auf die semantische Anreicherung von Modellelementen. Was den Nutzen der automatischen Verifikation von Geschäftsprozessmodellen auf Grundlage musterbasierter Bedingungen und struktureller Muster im Verhältnis zum Modellierungsaufwand betrifft, kommt es auf die Anzahl und Größe der zu verifizierenden Geschäftsprozessmodelle an. Dabei dürfte sich der Modellierungsaufwand ab einer gewissen Anzahl oder Größe amortisieren.

9.2. Ausblick

Während der Entwicklung der in den vorangehenden Kapiteln vorgestellten Konzepte tauchten mehrere Fragen auf, deren Beantwortung den Rahmen dieser Arbeit gesprengt hätte, die jedoch für zukünftige Forschungsvorhaben interessant sein könnten. Diese Fragen zielen auf der einen Seite direkt auf eine Verbesserung der in dieser Arbeit entwickelten Konzepte ab, beschäftigen sich auf der anderen Seite aber auch mit Verbesserungsmöglichkeiten im Kontext dieser Konzepte.

Wie bereits erwähnt sollen die Modellierungskonstrukte der Process Pattern Modeling Language und der Process Constraint Modeling Language nicht als absolut angesehen werden, sondern als Ausgangspunkt für eine Weiterentwicklung beider Sprachen dienen. Hinsichtlich zukünftiger Erweiterungen wäre es nützlich, auf die Erfahrung von Anwendern im Bereich der Geschäftsprozessmodellierung, die mit gesetzlichen, vertraglichen oder internen Richtlinien von Unternehmen unterschiedlicher Branchen vertraut sind, zurückzugreifen. Aufgrund des deklarativen Suchansatzes können neue Modellierungskonstrukte ohne größere Änderungen der Architektur hinzugefügt werden.

Aus technischer Sicht wäre zu überlegen, ob es sinnvoll wäre, die graphischen Notationen von PPML und PCML zu einer einzigen Notation zu vereinen. Diese Möglichkeit wurde in der vorliegenden Arbeit der Einfachheit halber nicht weiter verfolgt, eine gelungene Synthese der beiden graphischen Modellierungssprachen soll dadurch jedoch nicht ausgeschlossen werden. Des Weiteren wurde an mehreren Stellen innerhalb der vorangehenden Kapitel auf vergleichbare Ansätze hingewiesen, die es ermöglichen, das Laufzeitverhalten von Geschäftsprozessmodellen auf gewisse Eigenschaften zu überprüfen. Der in der vorliegenden Arbeit vorgeschlagene Ansatz könnte um vergleichbare Konzepte erweitert werden, die neben der Modellierung struktureller Muster auch die Modellierung von Laufzeitmustern ermöglichen. Eine derartige Erweiterung würde zu einem ganzheitlichen Ansatz führen, mit dessen Hilfe bei der Verifikation von Geschäftsprozessmodellen sowohl strukturelle Aspekte als auch Laufzeitaspekte berücksichtigt werden könnten.

Über diese Erweiterungsmöglichkeiten hinaus wären zwei weitere Verbesserungen wünschenswert, die eine bessere Verknüpfung von Techniken aus dem Bereich der modellgetriebenen Softwareentwicklung und des semantischen Webs ermöglichen. Zum einen wäre es erfreulich, wenn eine zukünftige Version von MQL (siehe Abschnitt 6.1) die Verwendung benutzerdefinierter Funktionen innerhalb von Abfragen erlauben würde, da die Verwendung von MQL als Grundlage der musterbasierten Suche deren Leistung noch erhöhen dürfte. Zum anderen wäre die Entwicklung eines Reasoners wünschenswert, der direkt auf Ontologien in Form von Modellen auf Grundlage des Ontology-Definition-Metamodells zugreifen kann und die derzeit benötigten Transformationsschritte entfallen. Insgesamt bleibt zu hoffen, dass zukünftige Werkzeuge zur Geschäftsprozessmodellierung weiter reichende Möglichkeiten zur Auswertung von Anforderungen bieten, die Aussagen über die notwendige Beschaffenheit der Struktur von Geschäftsprozessen mit Bezug auf deren inhaltliche Bedeutung machen. Die vorliegende Arbeit sollte dafür wertvolle Impulse liefern können.

A. Zusätzliche Abbildungen

A.1. BPMN-Metamodell

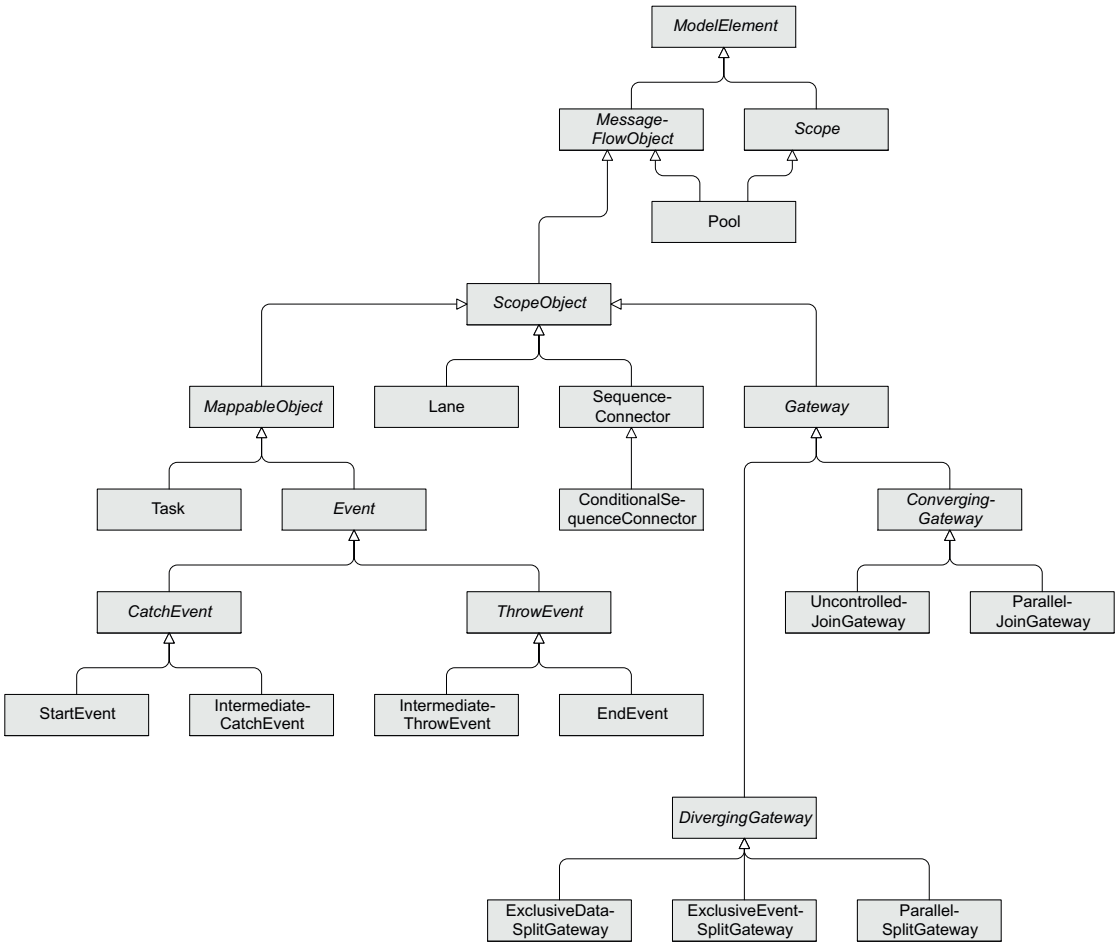


Abbildung A.1.: BPMN-Metamodell (Ausschnitt)

A.2. OWL-Metamodell

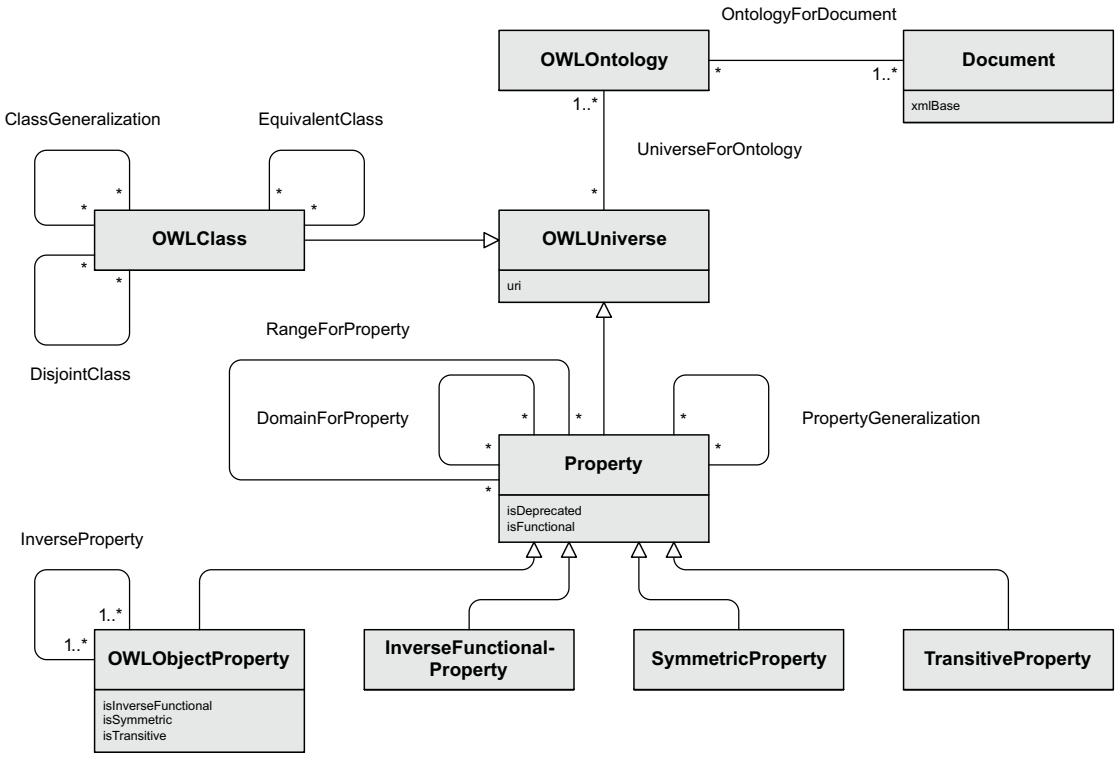


Abbildung A.2.: OWL-Metamodell

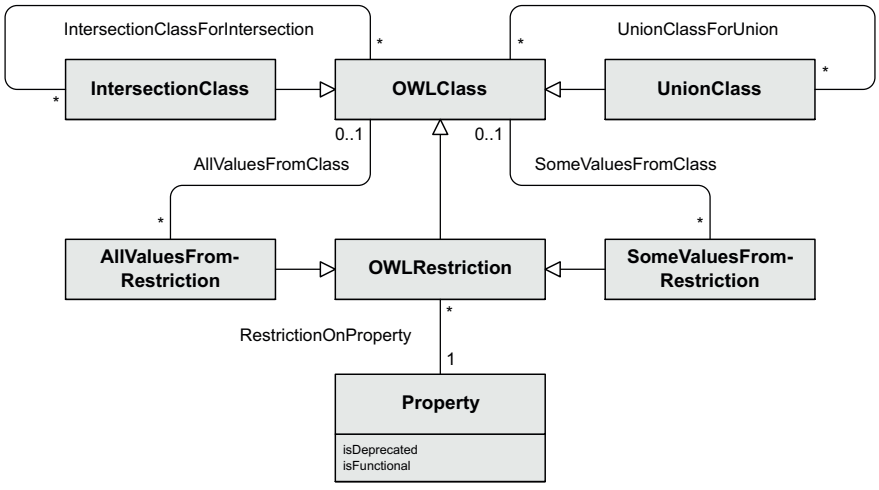


Abbildung A.3.: OWL-Metamodell (Fortsetzung)

A.3. ExtendedBPMN-Metamodell

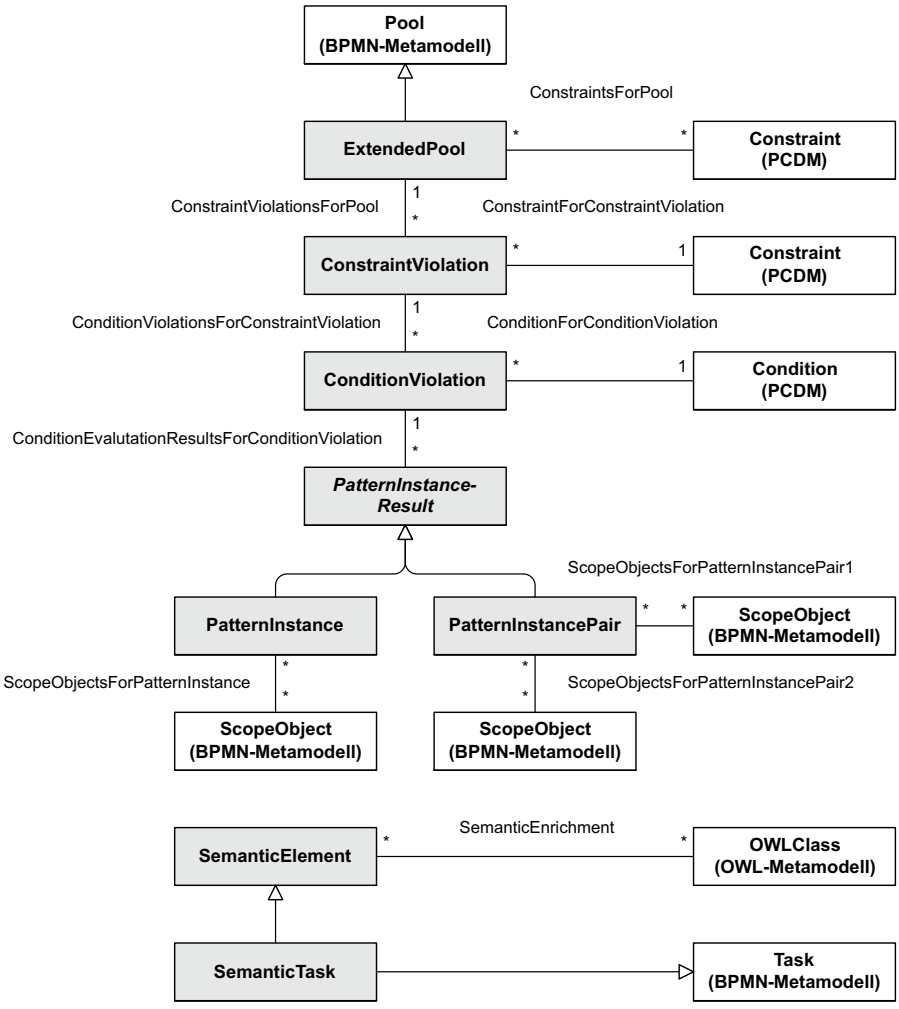


Abbildung A.4.: ExtendedBPMN-Metamodell

A.4. Process Constraint Definition Metamodel

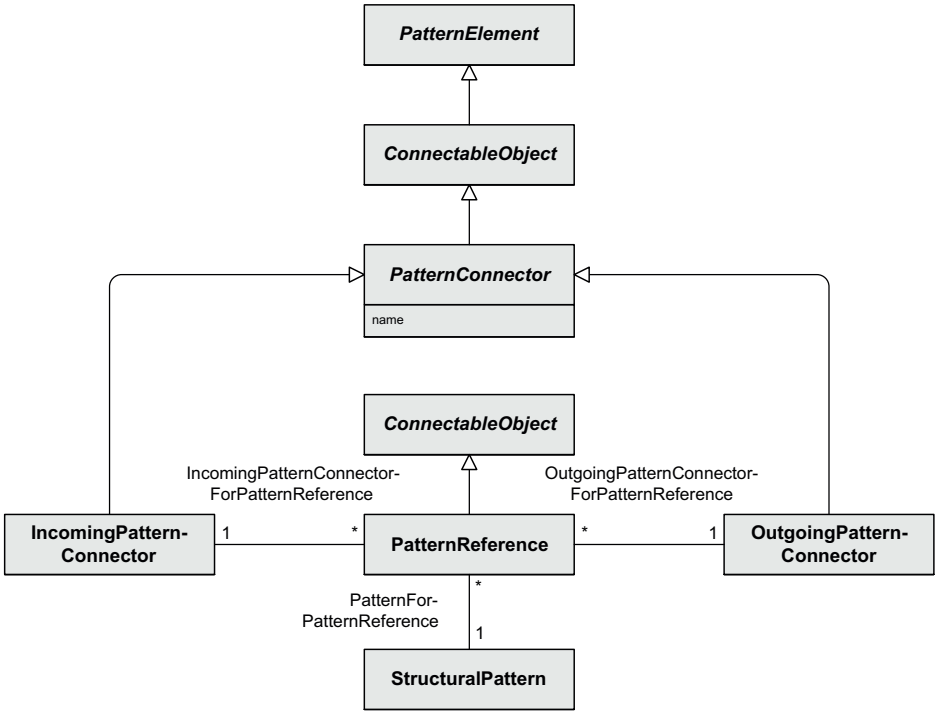


Abbildung A.5.: PCDM (Musterkonnektoren und Musterreferenz)

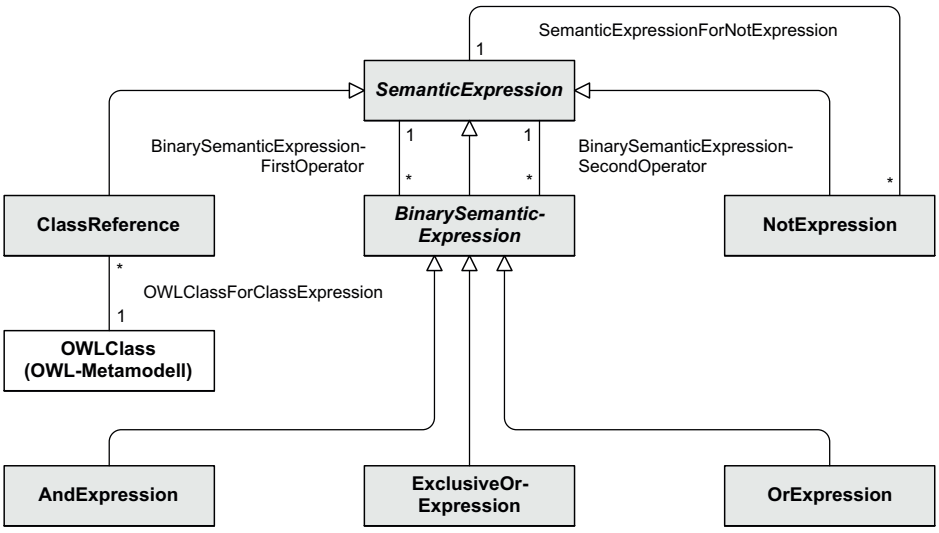


Abbildung A.6.: PCDM (Semantische Ausdrücke)

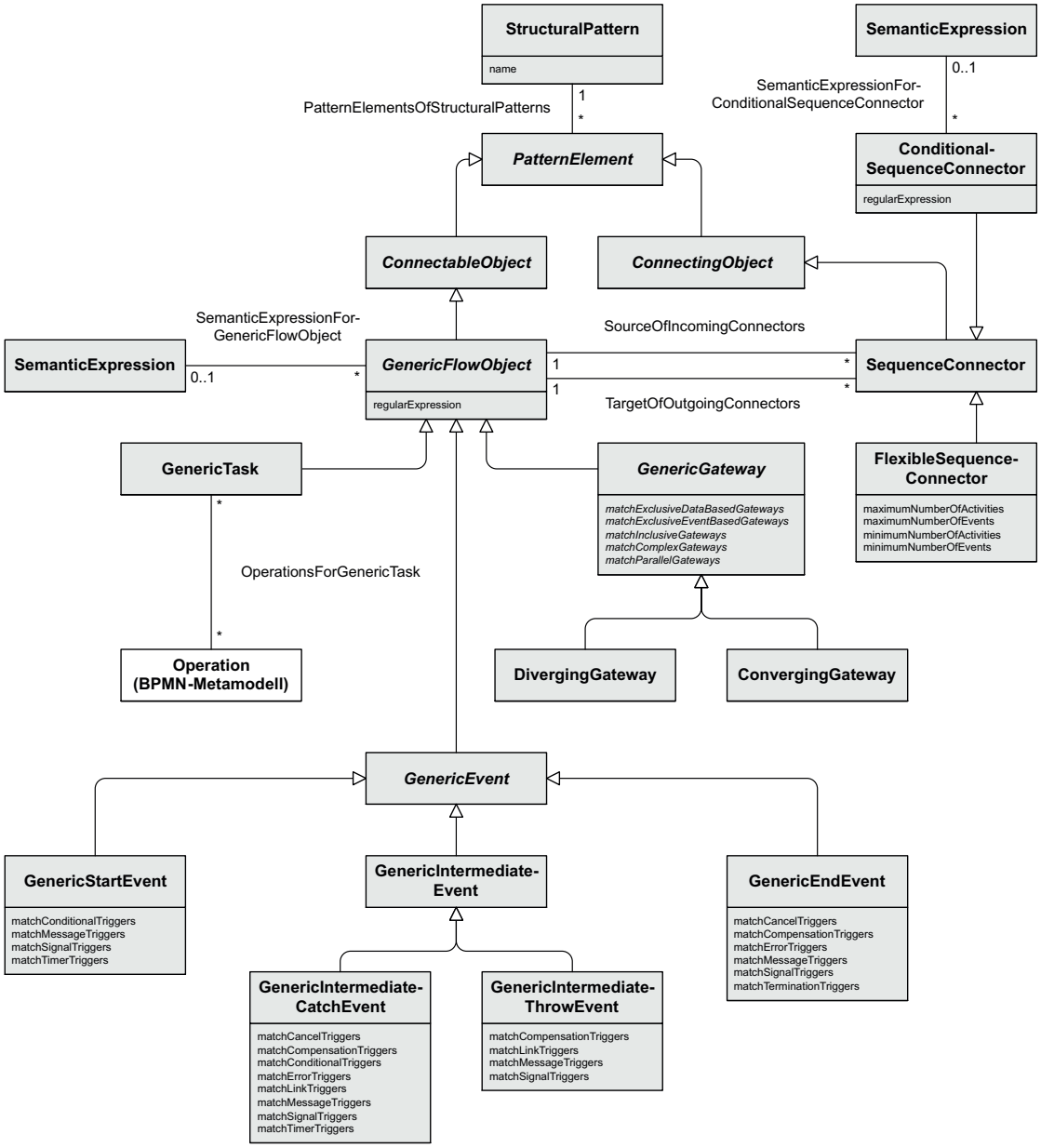


Abbildung A.7.: PCDM (Generische Fluss- und Verbindungsobjekte)

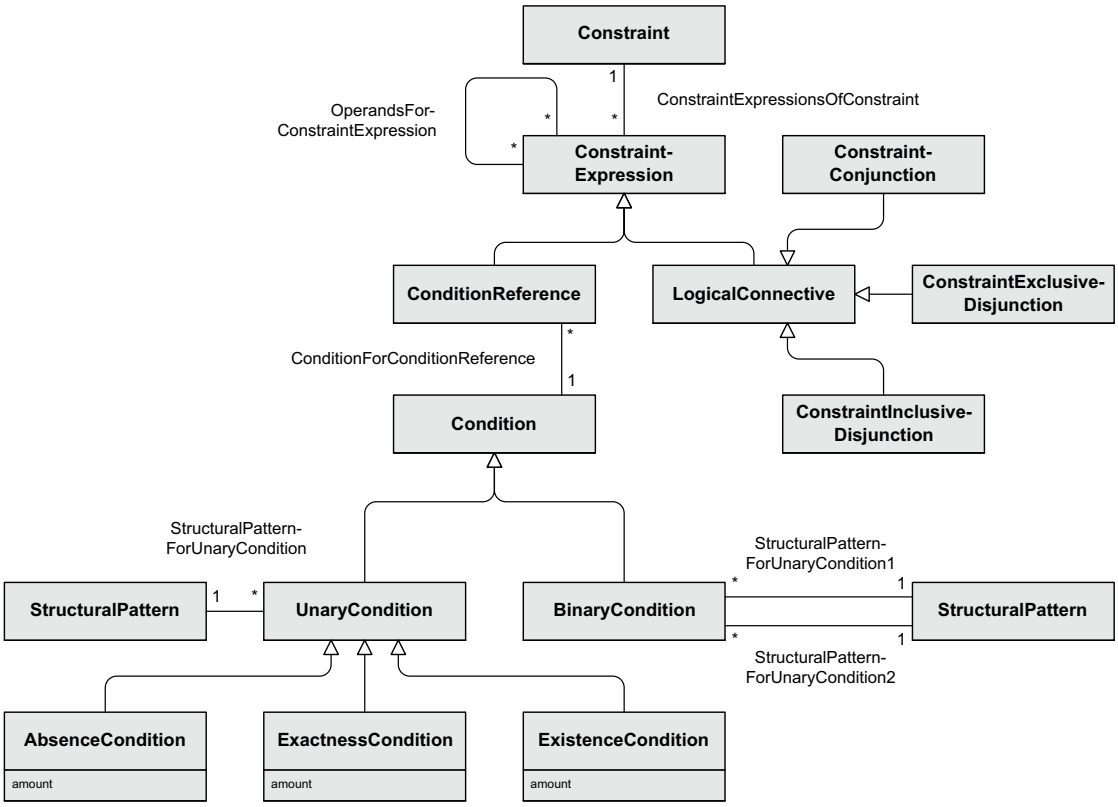


Abbildung A.8.: PCDM (Bedingungsausdrucks- und Verbindungsobjekte)

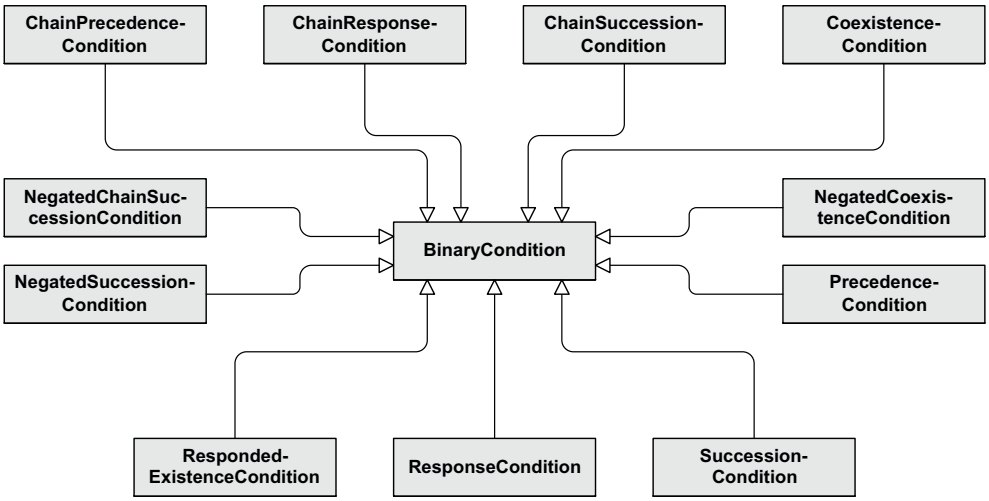


Abbildung A.9.: PCDM (Binäre musterbasierte Bedingungen)

A.5. Übersetzung generischer Ereignisse und Gateways

Modellierungskonstrukt	Gesetzte Attribute	Drools-Regel: Erzeugte Bedingungen
Generisches Startereignis		`startEvent : StartEvent()`
Generisches Zwischenereignis		`intermediateEvent : Event()` `eval((intermediateEvent instanceof IntermediateCatchEvent) || (intermediateEvent instanceof IntermediateThrowEvent))`
Gen. send. Zwischenereignis		`intermediateEvent : IntermediateThrowEvent()`
Gen. empf. Zwischenereignis		`intermediateEvent : IntermediateCatchEvent()`
Generisches Endereignis	☒ matchErrorTriggers ☒ matchMessageTriggers ☒ matchTerminationTriggers	`endEvent : EndEvent()`
Generisches Endereignis	☒ matchErrorTriggers	`eventDefinition : ErrorEventDefinition()` `endEvent : EndEvent(eventDefinition == eventDefinition)`
Generisches Endereignis	☒ matchMessageTriggers	`eventDefinition : MessageEventDefinition()` `endEvent : EndEvent(eventDefinition == eventDefinition, terminate == false)`
Generisches Endereignis	☒ matchTerminationTriggers	`endEvent : EndEvent(terminate == true)`
Generisches Endereignis	☒ matchErrorTriggers ☒ matchMessageTriggers	`endEvent : EndEvent()` `eval((endEvent.getEventDefinition() instanceof ErrorEventDefinition) || (endEvent.getEventDefinition() instanceof MessageEventDefinition))`
Generisches Endereignis	☒ matchErrorTriggers ☒ matchTerminationTriggers	`endEvent : EndEvent()` `eval((endEvent.getEventDefinition() instanceof ErrorEventDefinition) || endEvent.isTerminate())`
Generisches Endereignis	☒ matchMessageTriggers ☒ matchTerminationTriggers	`endEvent : EndEvent()` `eval((endEvent.getEventDefinition() instanceof MessageEventDefinition) || endEvent.isTerminate())`
Divergierender Gateway	☒ matchExclusiveDataBasedGateways ☒ matchParallelGateways	`divergingGateway : DivergingGateway()`
Divergierender Gateway	☒ matchExclusiveDataBasedGateways	`divergingGateway : ExclusiveDataSplitGateway()`
Divergierender Gateway	☒ matchParallelGateways	`divergingGateway : ParallelSplitGateway()`
Konvergierender Gateway	☒ matchExclusiveDataBasedGateways ☒ matchParallelGateways	`convergingGateway : ConvergingGateway()`
Konvergierender Gateway	☒ matchExclusiveDataBasedGateways	`convergingGateway : UncontrolledJoinGateway()`
Konvergierender Gateway	☒ matchParallelGateways	`convergingGateway : ParallelJoinGateway()`

Abbildung A.10.: Übersetzung generischer Ereignisse und Gateways

A.6. Begleitpapiere

1. Approving Competent Authority/Country Zuständige Genehmigungsbehörde / Staat Luftfahrt-Bundesamt / Germany	2. AUTHORISED RELEASE CERTIFICATE Autorisierte Freigabebescheinigung EASA FORM 1 EASA Formblatt 1					3. Form Tracking Number Lfd. Formularnummer
4. Approved Organisation Name and Address: Name und Anschrift des genehmigten Betriebes						5. Work Order/Contract/Invoice Arbeitsauftrag/Vertrag/Lieferschein
6. Item Lfd. Nr./Position	7. Description Beschreibung	8. Part No. Teile-Nr.	9. Eligibility(*) Verwendbarkeit(*)	10. Quantity Anzahl/Menge	11. Serial/Batch-No. Werk-/Los-Nr.	12. Status/Work Zustand/Arbeiten
13.Remarks Bemerkungen						
14. Certifies that the items identified above were manufactured in conformity to: Es wird bescheinigt, dass die oben aufgeführten Artikel hergestellt wurden in Übereinstimmung mit: ☐ approved design data and are in condition for safe operation genehmigten Konstruktionsdaten und in einem Zustand für einen sicheren Betrieb sind. ☐ non-approved design data specified in block 13 in Feld 13 aufgeführten nicht genehmigten Konstruktionsdaten.			19. ☐ Part-145.A.50 Release to Service Freigabebescheinigung gemäß Teil -145 A.50 Certifies that unless otherwise specified in block 13, the work identified in block 12 and described in block 13, was accomplished in accordance with Part-145 and in respect to that work the items are considered ready for release to service Es wird bescheinigt, dass, sofern in Feld 13 nichts anderes bestimmt ist, die in Feld 12 aufgeführten und in Feld 13 beschriebenen Arbeiten in Übereinstimmung mit Teil-145 durchgeführt wurden, und dass die Artikel im Hinblick auf diese Arbeiten als geeignet für die Freigabe zum Betrieb betrachtet werden.		☐ Other Regulation specified in block 13 Freigabebescheinigung gemäß anderer in Feld 13 aufgeführter Vorschrift	
15. Authorised Signature Unterschrift der berechtigten Person	16. Approval/Authorisation Number Genehmigungs-/Zulassungsnummer		20. Authorised Signature Unterschrift der berechtigten Person		21. Certificate/Approval Ref. No. Bescheinigungs-/Genehmigungsnummer	
17. Name Name	18. Date (d/m/y) Datum (T/M/J)		22. Name Name		23. Date (d/m/y) Datum (T/M/J)	

EASA Form 1 – Issue 1
EASA Formblatt 1 – Ausgabe 1

(*) Installer must cross-check eligibility with applicable technical data
(*) Der Verwender/einbauende Betrieb ist verpflichtet, die Verwendbarkeit anhand der geltenden technischen Unterlagen zu überprüfen.

Abbildung A.11.: EASA Formblatt 1

Akronyme

ATA	Air Transport Association
BNF	Backus-Naur-Form
BPDM	Business Process Definition Metamodel
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modeling Notation
BPMS	Business-Process-Management-System
CTL	Computation Tree Logic
DRL	Drools Rule Language
EAI	Enterprise Application Integration
EASA	Europäische Agentur für Flugsicherheit
EMF	Eclipse Modeling Framework
EPK	Ereignisgesteuerte Prozesskette
ERP	Enterprise Resource Planning
FAA	Federal Aviation Administration
GEF	Graphical Editing Framework
JMI	Java Metadata Interface
LLP	Life-Limited Part
LTL	Lineare temporale Logik
MDA	Model Driven Architecture

MEL . . . . . Master Equipment List
MOF . . . . . Meta Object Facility
MOIN . . . . . Modeling Infrastructure
MQL . . . . . MOIN Query Language
MRI . . . . . Model Element Resource Identifier

OCL . . . . . Object Constraint Language
ODM . . . . . Ontology Definition Metamodel
OMG . . . . . Object Management Group
OWL . . . . . Web Ontology Language

PCDM . . . . . Process Constraint Definition Metamodel
PCML . . . . . Process Constraint Modeling Language
PPML . . . . . Process Pattern Modeling Language
PROMELA . . . . . Process Meta Language

RDF . . . . . Resource Description Framework
RFID . . . . . Radio Frequency Identification

SOA . . . . . serviceorientierte Architektur
SPARQL . . . . . SPARQL Query Language for RDF
SQL . . . . . Structured Query Language
SUP . . . . . Suspected Unapproved Part
SWRL . . . . . Semantic Web Rule Language

UML . . . . . Unified Modeling Language
URI . . . . . Uniform Resource Identifier

WfMS . . . . . Workflow-Management-System
WSML . . . . . Web Service Modeling Language

XMI . . . . . XML Metadata Interchange
XML . . . . . Extensible Markup Language

Literaturverzeichnis

[ADW08] Awad, Ahmed ; Decker, Gero ; Weske, Mathias: Efficient Compliance Checking using BPMN-Q and Temporal Logic. In: *Business Process Management* Bd. 5240. Berlin Heidelberg : Springer-Verlag, 2008 (Lecture Notes in Computer Science), S. 326–341. – DOI 10.1007/978-3-540-85758-7

[AFKK07] Abramowicz, Witold ; Filipowska, Agata ; Kaczmarek, Monika ; Kaczmarek, Tomasz: Semantically Enhanced Business Process Modelling Notation. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 88–91

[All05] Allweyer, Thomas: *Geschäftsprozessmanagement – Strategie, Entwurf, Implementierung, Controlling.* Herdecke Bochum : W3L-Verlag, 2005

[All09] Allweyer, Thomas: *BPMN 2.0 – Business Process Model and Notation: Einführung in den Standard für die Geschäftsprozessmodellierung.* 2. Auflage. Norderstedt : Books on Demand, 2009

[AP06a] Aalst, Willibrordus Martinus Pancratius van der ; Pešić, Maja: A Declarative Approach for Flexible Business Processes Management. In: Eder, Johann (Hrsg.) ; Dustdar, Schahram (Hrsg.): *Business Process Management Workshops* Bd. 4103. Berlin Heidelberg : Springer-Verlag, 2006 (Lecture Notes in Computer Science), S. 169–180. – DOI 10.1007/11837862_18

[AP06b] Aalst, Willibrordus Martinus Pancratius van der ; Pešić, Maja: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, Mario (Hrsg.) ; Núñez, Manuel (Hrsg.) ; Zavattaro, Gianluigi (Hrsg.): *Web Services and Formal Methods* Bd. 4184. Berlin Heidelberg : Springer-Verlag, 2006 (Lecture Notes in Computer Science), S. 1–23. – DOI 10.1007/11841197_1

[AP06c] Aalst, Willibrordus Martinus Pancratius van der ; Pešić, Maja: Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware / BPM Center. 2006 (BPM-06-09). – Forschungsbericht

[Awa07] Awad, Ahmed: BPMN-Q: A Language to Query Business Processes. In: Reichert, Manfred (Hrsg.) ; Strecker, Stefan (Hrsg.) ; Turowski, Klaus (Hrsg.): *Enterprise Modelling and Information Systems Architectures: Concepts and Applications* Bd. P-119. Bonn : Gesellschaft für Informatik, 2007 (Lecture Notes in Informatics), S. 115–128

[AZW06] Assmann, Uwe ; Zschaler, Steffen ; Wagner, Gerd: Ontologies, Metamodels, and the Model-Driven Paradigm. In: *Ontologies for Software Engineering and Technology*. Berlin Heidelberg : Springer-Verlag, 2006, S. 249–274. – DOI 10.1007/3-540-34518-3_9

[BA08] Ben-Ari, Mordechai: *Principles of the Spin Model Checker*. Berlin Heidelberg : Springer-Verlag, 2008

[Bal09] Bali, Michal: *Drools JBoss Rules 5.0 Developer's Guide*. Birmingham : Packt Publishing, 2009

[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: Lowe, David (Hrsg.) ; Gaedke, Martin (Hrsg.): *Web Engineering* Bd. 3579. Berlin Heidelberg : Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 557–568. – DOI 10.1007/11531371_70

[BDW07] Born, Matthias ; Dörr, Florian ; Weber, Ingo: User-friendly Semantic Annotation in Business Process Modeling. In: Weske, Mathias (Hrsg.) ; Hacid, Mohand-Saïd (Hrsg.) ; Godart, Claude (Hrsg.): *Web Information Systems Engineering – WISE 2007 Workshops* Bd. 4832. Berlin Heidelberg : Springer-Verlag, 2007 (Lecture Notes in Computer Science), S. 260–271. – DOI 10.1007/978-3-540-77010-7_25

[BEKM05] Beeri, C. ; Eyal, A. ; Kamenkovich, S. ; Milo, T.: Querying Business Processes with BP-QL. In: *Proceedings of the 31st International Conference on Very Large Data Bases*, 2005, S. 1255–1258

[BEKM06] Beeri, C. ; Eyal, A. ; Kamenkovich, S. ; Milo, T.: Querying Business Processes. In: *Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006, S. 343–354

[BEKM08] Beeri, Catriel ; Eyal, Anat ; Kamenkovich, Simon ; Milo, Tova: Querying Business Processes with BP-QL. In: *Information Systems* 33 (2008), Nr. 6, S. 477–507. – DOI 10.1016/j.is.2008.02.005

[Bev04] Beveren, Tim van: Tickende Zeitbomben – im Visier der Fahnder. In: *VdL-Nachrichten* 4 (2004), S. 6–14

[BHK+08] Born, Matthias ; Hoffmann, Jörg ; Kaczmarek, Tomasz ; Kowalkiewicz, Marek ; Markovic, Ivan ; Scicluna, James ; Weber, Ingo ; Zhou, Xuan: Semantic Annotation and Composition of Business Processes with Maestro. In: Bechhofer, Sean (Hrsg.) ; Hauswirth, Manfred (Hrsg.) ; Hoffmann, Jörg (Hrsg.) ; Koubarakis, Manolis (Hrsg.): *The Semantic Web: Research and Applications* Bd. 5021. Berlin Heidelberg : Springer-Verlag, 2008 (Lecture Notes in Computer Science), S. 772–776. – DOI 10.1007/978-3-540-68234-9_56

[BK08] Baier, Christel ; Katoen, Joost-Pieter: *Principles of Model Checking*. MIT Press, 2008

[BKI08] Beierle, Christoph ; Kern-Isberner, Gabriele: *Methoden wissensbasierter Systeme*. 4. Aufl. Wiesbaden : Vieweg+Teubner, 2008

[BKR08] Becker, Jörg (Hrsg.) ; Kugeler, Martin (Hrsg.) ; Rosemann, Michael (Hrsg.): *Prozessmanagement : Ein Leitfaden zur prozessoptimierten Organisationsgestaltung*. 6. Auflage. Berlin Heidelberg : Springer-Verlag, 2008

[BLFM05] Berners-Lee, Tim ; Fielding, Roy T. ; Masinter, Larry: *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax*. 2005

[BMJ06] BMJ (Bundesministeriums der Justiz): *Verordnung zur Prüfung von Luftfahrtgerät (Artikel 2 der Verordnung zur Änderung luftrechtlicher Vorschriften über die Entwicklung, Zulassung, Herstellung und Instandhaltung von Luftfahrtgerät)*. 2006

[BMJ09] BMJ (Bundesministeriums der Justiz): *Luftverkehrs-Zulassungs-Ordnung (LuftVZO)*. 2009

[Bra05] Brambilla, Marco: LTL Formalization of BPML Semantics and Visual Notation for Linear Temporal Logic / Politecnico di Milano. 2005. – Forschungsbericht

[CF82] Cohen, Paul R. (Hrsg.) ; Feigenbaum, Edward A. (Hrsg.): *The Handbook of Artificial Intelligence*. Bd. 3. Los Altos : William Kaufmann, 1982

[Cla09] Clark, Jim: New Technology Could Protect Against Aircraft Parts Counterfeiting. In: *Aviation Maintenance* 28 (2009), Januar, Nr. 1, S. 8

[DAC98] Dwyer, Matthew B. ; Avrunin, George S. ; Corbett, James C.: Property Specification Patterns for Finite-State Verification. In: *Proceedings of the Second Workshop on Formal Methods in Software Practice*. New York : ACM Press, 1998, S. 7–15. – DOI 10.1145/298595.298598

[DB07] Davis, Rob ; Brabänder, Eric: *ARIS Design Platform: Getting Started with BPM*. London : Springer-Verlag, 2007

[DDO08] Dijkman, Remco M. ; Dumas, Marlon ; Ouyang, Chun: Semantics and Analysis of Business Process Models in BPMN. In: *Information and Software Technology* 50 (2008), Nr. 12, S. 1281–1294. – DOI 10.1016/j.infsof.2008.02.006

[Dev02] Devedžić, Vladan: Understanding Ontological Engineering. In: *Communications of the ACM* 45 (2002), Nr. 4, S. 136–144. – DOI 10.1145/505248.506002

[DSSK07] Dimitrov, Marin ; Simov, Alex ; Stein, Sebastian ; Konstantinov, Mihail: A BPMO Based Semantic Business Process Modelling Environment. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 101–104

[Dud07] Dudenredaktion (Hrsg.): *Duden: Das Fremdwörterbuch.* 9. Aufl. Mannheim : Bibliographisches Institut & F. A. Brockhaus, 2007

[FAA95] FAA (Federal Aviation Administration): *Suspected 'Unapproved Parts' Program Plan.* 1995

[FE03] Förster, Alexander ; Engels, Gregor: Quality Ensuring Development of Software Processes. In: Oquendo, Flavio (Hrsg.): *Software Process Technology* Bd. 2786. Berlin Heidelberg : Springer-Verlag, 2003 (Lecture Notes in Computer Science), S. 62–73. – DOI 10.1007/b12019

[FES05] Förster, Alexander ; Engels, Gregor ; Schattkowsky, Tim: Activity Diagram Patterns for Modeling Quality Constraints in Business Processes. In: Briand, Lionel (Hrsg.) ; Williams, Clay (Hrsg.): *Model Driven Engineering Languages and Systems* Bd. 3713. Berlin Heidelberg : Springer-Verlag, 2005 (Lecture Notes in Computer Science), S. 2–16. – DOI 10.1007/11557432_2

[FESS06] Förster, Alexander ; Engels, Gregor ; Schattkowsky, Tim ; Straeten, Ragnhild van d.: A Pattern-driven Development Process for Quality Standard-conforming Business Process Models. In: *2006 IEEE Symposium on Visual Languages and Human-Centric Computing.* Los Alamitos : IEEE Computer Society Press, 2006, S. 135–142. – DOI 10.1109/VLHCC.2006.5

[FESS07] Förster, Alexander ; Engels, Gregor ; Schattkowsky, Tim ; Straeten, Ragnhild van d.: Verification of Business Process Quality Constraints Based on Visual Process Patterns. In: *TASE 2007: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering*, IEEE Computer Society Press, 2007, S. 197–208. – DOI 10.1109/TASE.2007.56

[FF08] Feja, Sven ; Fötsch, Daniel: Model Checking with Graphical Validation Rules. In: Bustard, David W. (Hrsg.) ; Sterritt, Roy (Hrsg.): *Engineering of Computer Based Systems*. Los Alamitos : IEEE Computer Society, 2008, S. 117–125. – DOI 10.1109/ECBS.2008.45

[FG09] Francisco, David de ; Grenon, Pierre: Enhancing Telecommunication Business Process Representation and Integration with Ontologised Industry Standards. In: *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2009)*, 2009

[FGR+08] Francescomarino, Chiara di ; Ghidini, Chiara ; Rospocher, Marco ; Serafini, Luciano ; Tonella, Paolo: Reasoning on Semantically Annotated Processes. In: Bouguettaya, Athman (Hrsg.) ; Krueger, Ingolf (Hrsg.) ; Margaria, Tiziana (Hrsg.): *Service-Oriented Computing – ICSOC 2008* Bd. 5364. Berlin Heidelberg : Springer-Verlag, 2008 (Lecture Notes in Computer Science), S. 132–146. – DOI 10.1007/978-3-540-89652-4_13

[FHKM04] Frankel, David ; Hayes, Pat ; Kendall, Elisa ; McGuinness, Deborah: The Model Driven Semantic Web. In: *Proceedings of the 1st International Workshop on the Model-Driven Semantic Web*, 2004

[FKS09] Filipowska, Agata ; Kaczmarek, Monika ; Stein, Sebastian: Semantically Annotated EPC within Semantic Business Process Management. In: Ardagna, Danilo (Hrsg.) ; Mecella, Massimo (Hrsg.) ; Yang, Jian (Hrsg.): *Business Process Management Workshops* Bd. 17. Berlin Heidelberg : Springer-Verlag, 2009 (Lecture Notes in Business Information Processing), S. 486–497. – DOI 10.1007/978-3-642-00328-8_49

[Flo62] Floyd, Robert W.: Algorithm 97: Shortest path. In: *Communications of the ACM* 5 (1962), Nr. 6, S. 345. – DOI 10.1145/367766.368168

[Flo67] Floyd, Robert W.: Assigning Meanings to Programs. In: Schwarz, Jacob T. (Hrsg.): *Mathematical Aspects of Computer Science* Bd. 19. Providence : American Mathematical Society, 1967, S. 19–32

[For79] Forgy, Charles: *On the Efficient Implementation of Production Systems*, Carnegie Mellon University, Diss., 1979

[FR09] Fengel, Janina ; Rebstock, Michael: Model-Based Domain Ontology Engineering. In: *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2009)*, 2009

[Fra08] Francescomarino, Chiara di: Supporting Documentation and Evolution of Crosscutting Concerns in Business Processes. In: Motahari-Nezhad, Hamid R. (Hrsg.) ; Toumani, Farouk (Hrsg.) ; Velegrakis, Yannis (Hrsg.): *ICSOC PhD Symposium 2008* Bd. 421, 2008 (CEUR Workshop Proceedings), S. 23–28

[FRH10] Freund, Jakob ; Rücker, Bernd ; Henninger, Thomas: *Praxishandbuch BPMN*. München : Carl Hanser Verlag, 2010

[FT08] Francescomarino, Chiara di ; Tonella, Paolo: Business Process Concern Documentation And Evolution / Fondazione Bruno Kessler. 2008 (200806006). – Forschungsbericht

[FT09] Francescomarino, Chiara di ; Tonella, Paolo: Crosscutting Concern Documentation by Visual Query of Business Processes. In: Ardagna, Danilo (Hrsg.) ; Mecella, Massimo (Hrsg.) ; Yang, Jian (Hrsg.): *Business Process Management Workshops* Bd. 17. Berlin Heidelberg : Springer-Verlag, 2009 (Lecture Notes in Business Information Processing), S. 18–31. – DOI 10.1007/978-3-642-00328-8_3

[Gad10] Gadatsch, Andreas: *Grundkurs Geschäftsprozess-Management : Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. 6. Auflage. Wiesbaden : Vieweg+Teubner Verlag, 2010

[Gat79] Gati, Georg: Further Annotated Bibliography on the Isomorphism Disease. In: *Journal of Graph Theory* 3 (1979), Nr. 2, S. 95–109. – DOI 10.1002/jgt.3190030202

[GHJV09] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software*. 5. Auflage. München : Addison-Wesley, 2009

[GR04] Giarratano, Joseph C. ; Riley, Gary D.: *Expert Systems: Principles and Programming*. Course Technology, 2004

[Gra09] Graf, Mario: *Automatisierte Mustersuche in semantisch angereicherten BPMN-Graphen zur Auffindung von Schwachstellen*, Duale Hochschule Baden-Württemberg Karlsruhe, Bachelorarbeit, 2009

[GRS08] Ghidini, Chiara ; Rospocher, Marco ; Serafini, Luciano: A Formalisation of BPMN in Description Logics / Fondazione Bruno Kessler. 2008 (200806004). – Forschungsbericht

[Gru93] Gruber, Thomas R.: A Translation Approach to Portable Ontology Specifications. In: *Knowledge Acquisition* 5 (1993), Nr. 2, S. 199–220. – DOI 10.1006/knac.1993.1008

[Ham90] Hammer, Michael: Reengineering Work: Don't Automate, Obliterate. In: *Harvard Business Review* 68 (1990), Nr. 4, S. 104–112

[HKRS08] Hitzler, Pascal ; Krötsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web*. Berlin Heidelberg : Springer-Verlag, 2008

[HLD+05] Hepp, Martin ; Leymann, Frank ; Domingue, John ; Wahler, Alexander ; Fensel, Dieter: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: *Proceedings of the IEEE International Conference on e-Business Engineering (ICEBE 2005)*. Los Alamitos : IEEE Computer Society, 2005, S. 535–540. – DOI 10.1109/ICEBE.2005.110

[Hol04] Holzmann, Gerard J.: *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2004

[HR85] Hayes-Roth, Frederick: Rule-based Systems. In: *Communications of the ACM* 28 (1985), Nr. 9, S. 921–932. – DOI 10.1145/4284.4286

[HR07] Hepp, Martin ; Roman, Dumitru: An Ontology Framework for Semantic Business Process Management. In: Oberweis, Andreas (Hrsg.) ; Weinhardt, Christof (Hrsg.) ; Gimpel, Henner (Hrsg.) ; Koschmider, Agnes (Hrsg.) ; Pankratius, Victor (Hrsg.) ; Schnizler, Björn (Hrsg.): *eOrganisation: Service, Prozess-, Market-Engineering : 8. Internationale Tagung Wirtschaftsinformatik*. Karlsruhe : Universitätsverlag Karlsruhe, 2007, S. 423–440

[ISO99] ISO (Internationale Organisation für Normung): *ISO/IEC 9899:1999 : Programming languages – C*. 1999

[ISO03] ISO (Internationale Organisation für Normung): *ISO/IEC 13250:2003 : Information technology – SGML applications – Topic maps*. 2003

[ISO07] ISO (Internationale Organisation für Normung): *ISO/IEC 24707:2007 : Information technology – Common Logic (CL): a framework for a family of logic-based languages*. 2007

[ISO08] ISO (Internationale Organisation für Normung): *ISO/IEC 9075-1:2008 : Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. 2008

[JCP02] JCP (Java Community Process): *Java Metadata Interface (JMI) Specification*. 2002

[KNS92] Keller, Gerhard ; Nüttgens, Markus ; Scheer, August-Wilhelm: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. In: *Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes* 89 (1992), S. 2–30

[KNS97] Kurbel, Karl ; Nenoglu, Georg ; Schwarz, Christian: Von der Geschäftsprozessmodellierung zur Workflowspezifikation – Zur Kompatibilität von Modellen und Werkzeugen. In: *HMD – Praxis der Wirtschaftsinformatik* 34 (1997), November, Nr. 198, S. 66–82

[LB07] Lautenbacher, Florian ; Bauer, Bernhard: A Survey on Workflow Annotation & Composition Approaches. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 12–23

[LBA03] LBA (Luftfahrt-Bundesamt): *Rundschreiben Nr. 18-01/03-2.* Mai 2003

[Lin08] Lin, Yun: *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability*, Norwegian University of Science and Technology, Diss., 2008

[LMX07] Liu, Ying ; Müller, Samuel ; Xu, Ke: A Static Compliance-Checking Framework for Business Process Models. In: *IBM Systems Journal* 46 (2007), Nr. 2, S. 335–361. – DOI 10.1147/sj.462.0335

[Mül09a] Müller, Jens: A Rule-Based Approach to Match Structural Patterns with Business Process Models. In: Governatori, Guido (Hrsg.) ; Hall, John (Hrsg.) ; Paschke, Adrian (Hrsg.): *Rule Interchange and Applications* Bd. 5858. Berlin Heidelberg : Springer-Verlag, 2009 (Lecture Notes in Computer Science), S. 208–215. – DOI 10.1007/978-3-642-04985-9_20

[Mül09b] Müller, Jens: Supporting Change in Business Process Models Using Pattern-Based Constraints. In: Halpin, Terry (Hrsg.) ; Krogstie, John (Hrsg.) ; Nurcan, Selmin (Hrsg.): *Enterprise, Business-Process and Information Systems Modeling* Bd. 29. Berlin Heidelberg : Springer-Verlag, 2009 (Lecture Notes in Business Information Processing), S. 27–32. – DOI 10.1007/978-3-642-01862-6_3

[NWv07] Nitzsche, Jörg ; Wutke, Daniel ; van Lessen, Tammo: An Ontology for Executable Business Processes. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 52–63

[OAS07] OASIS: *Web Services Business Process Execution Language Version 2.0.* 2007

[OMG03] OMG (Object Management Group): *MDA Guide Version 1.0.1.* 2003

[OMG06a] OMG (Object Management Group): *Meta Object Facility (MOF) Core Specification: Version 2.0.* 2006

[OMG06b] OMG (Object Management Group): *Object Constraint Language Version 2.0.* 2006

[OMG07a] OMG (Object Management Group): *MOF 2.0/XMI Mapping, Version 2.1.1.* 2007

[OMG07b] OMG (Object Management Group): *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2.* 2007

[OMG08a] OMG (Object Management Group): *Business Process Definition Metamodel : Volume I: Common Infrastructure.* 2008

[OMG08b] OMG (Object Management Group): *Business Process Definition Metamodel : Volume II: Process Definitions.* 2008

[OMG09a] OMG (Object Management Group): *Business Process Modeling Notation: Version 1.2.* 2009

[OMG09b] OMG (Object Management Group): *Ontology Definition Metamodel: Version 1.0.* 2009

[OMG11] OMG (Object Management Group): *Business Process Model and Notation (BPMN) Version 2.0.* 2011

[Pei32] Peirce, Charles S. ; Hartshorne, Charles (Hrsg.) ; Weiss, Paul (Hrsg.): *Collected Papers of Charles Sanders Peirce, Volumes I and II: Principles of Philosophy and Elements of Logic.* Harvard University Press, 1932

[Pnu77] Pnueli, Amir: The Temporal Logic of Programs. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977).* New York, NY : Institute of Electrical and Electronics Engineers, 1977, S. 46–57. – DOI 10.1109/SFCS.1977.32

[PSW07] Parreiras, Fernando S. ; Staab, Steffen ; Winter, Andreas: On Marrying Ontological and Metamodeling Technical Spaces. In: *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering.* New York, NY, USA : ACM Press, 2007, S. 439–448. – DOI 10.1145/1287624.1287687

[RC77] Read, Ronald C. ; Corneil, Derek G.: The Graph Isomorphism Disease. In: *Journal of Graph Theory* 1 (1977), Nr. 4, S. 339–363. – DOI 10.1002/jgt.3190010410

[Rum99] Rump, Frank J.: *Geschäftsprozessmanagement auf Basis ereignisgesteuerter Prozessketten.* Stuttgart : B. G. Teubner, 1999

[Sch08] Schneider, Thorsten: *Graphische und formale Modellierung und Analyse von Schwachstellen in Geschäftsprozessen am Beispiel der Luftfahrtindustrie*, Universität Karlsruhe (TH), Diplomarbeit, 2008

[Sei08] Seitz, Christian: Patterns for Semantic Business Process Modeling / Institut für Informatik, Universität Augsburg. 2008 (2008-07). – Forschungsbericht

[Sie07a] Siegel, J.: BPDM: Die OMG-Spezifikationen zur Geschäftsprozessmodellierung. In: *OBJEKTspektrum* 14 (2007), Nr. 6, S. 12–13

[Sie07b] Siegel, J.: BPMN und BPDM: Die OMG-Spezifikationen zur Modellierung von Geschäftsprozessen. In: *OBJEKTspektrum* 14 (2007), Nr. 5, S. 10–11

[SM06] Simon, Carlo ; Mendling, Jan: Verification of Forbidden Behavior in EPCs. In: Mayr, Heinrich C. (Hrsg.) ; Breu, Ruth (Hrsg.): *Modellierung 2006* Bd. P-82. Bonn : Gesellschaft für Informatik, 2006 (Lecture Notes in Informatics), S. 233–244

[SS09] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. 2. Aufl. Berlin Heidelberg : Springer-Verlag, 2009 (International Handbooks on Information Systems)

[TF06] Thomas, Oliver ; Fellmann, Michael: Semantic Event-Driven Process Chains. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantics for Business Process Management (SBPM 2006)*, 2006

[TF07a] Thomas, Oliver ; Fellmann, Michael: Semantic Business Process Management: Ontology-Based Process Modeling Using Event-Driven Process Chains. In: *International Journal of Interoperability in Business Information Systems* 2 (2007), Nr. 1, S. 29–44

[TF07b] Thomas, Oliver ; Fellmann, Michael: Semantic EPC: Enhancing Process Modeling Using Ontology Languages. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 64–75

[VF07] Vaz, Cátia ; Ferreira, Carla: Formal Verification of Workflow Patterns with SPIN / INESC-ID. 2007 (12/2007). – Forschungsbericht

[W3C04a] W3C (World Wide Web Consortium): *OWL Web Ontology Language – Overview*. 2004

[W3C04b] W3C (World Wide Web Consortium): *RDF Primer*. Februar 2004

[W3C04c] W3C (World Wide Web Consortium): *RDF Vocabulary Description Language 1.0: RDF Schema*. 2004

[W3C04d] W3C (World Wide Web Consortium): *RDF/XML Syntax Specification (Revised)*. 2004

[W3C04e] W3C (World Wide Web Consortium): *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. 2004

[W3C06] W3C (World Wide Web Consortium): *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. 2006

[W3C07] W3C (World Wide Web Consortium): *XQuery 1.0: An XML Query Language*. 2007

[W3C08] W3C (World Wide Web Consortium): *SPARQL Query Language for RDF*. 2008

[W3C09] W3C (World Wide Web Consortium): *OWL 2 Web Ontology Language Document Overview*. 2009

[War62] Warshall, Stephen: A Theorem on Boolean Matrices. In: *Journal of the ACM* 9 (1962), Nr. 1, S. 11–12. – DOI 10.1145/321105.321107

[WAV04] Weske, Mathias ; Aalst, Willibrordus Martinus Pancratius van der ; Verbeek, H. M. W.: Advances in Business Process Management. In: *Data & Knowledge Engineering* 50 (2004), S. 1–8. – DOI 10.1016/j.datak.2004.01.001

[Wes07] Weske, Mathias: *Business Process Management: Concepts, Languages, Architectures*. Berlin Heidelberg : Springer-Verlag, 2007

[WG08] Wong, Peter Y. H. ; Gibbons, Jeremy: A Process Semantics for BPMN. In: Liu, Shaoying (Hrsg.) ; Maibaum, Tom (Hrsg.) ; Araki, Keijiro (Hrsg.): *Formal Methods and Software Engineering* Bd. 5256. Berlin Heidelberg : Springer-Verlag, 2008 (Lecture Notes in Computer Science), S. 355–374. – DOI 10.1007/978-3-540-88194-0_22

[WK04] Warmer, Jos ; Kleppe, Anneke: *Object Constraint Language 2.0*. Bonn : mitp-Verlag, 2004

[WKH08] Wörzberger, René ; Kurpick, Thomas ; Heer, Thomas: Checking Correctness and Compliance of Integrated Process Models. In: Negru, Viorel (Hrsg.) ; Jebelean, Tudor (Hrsg.) ; Petcu, Dana (Hrsg.) ; Zaharie, Daniela (Hrsg.): *SYNASC 2008*. Los Alamitos : IEEE Computer Society, 2008, S. 576–583. – DOI 10.1109/SYNASC.2008.10

[WMF+07] Wetzstein, Branimir ; Ma, Zhilei ; Filipowska, Agata ; Kaczmarek, Monika ; Bhiri, Sami ; Losada, Silvestre ; Lopez-Cobo, Jose-Manuel ; Cicurel, Laurent: Semantic Business Process Management: A Lifecycle Based Requirements Analysis. In: Hepp, Martin (Hrsg.) ; Hinkelmann, Knut (Hrsg.) ; Karagiannis, Dimitris (Hrsg.) ; Klein, Rüdiger (Hrsg.) ; Stojanovic, Nenad (Hrsg.): *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SBPM 2007)* Bd. 251, 2007 (CEUR Workshop Proceedings), S. 1–11

[Wor99] Workflow Management Coalition: *Workflow Management Coalition – Terminology & Glossary*. Februar 1999

Internetseitenverzeichnis

[1] SAP AG: *SAP - SAP NetWeaver Business Process Management.* http://www.sap.com/platform/netweaver/components/sapnetweaverbpm/

[2] SAP AG: *SAP - SAP NetWeaver: Your Foundation for Enabling and Managing Change.* http://www.sap.com/platform/netweaver/

[3] SAP AG: *SAP Deutschland - Geschäftsanwendungen, Unternehmenssoftware, Services und Support.* http://www.sap.com/germany/

[4] SAP AG: *Modeling Processes with Process Composer.* http://help.sap.com/saphelp_nwce72/helpdata/en/ce/19dc55105b46a0b498af9d840a93a8/frameset.htm

[5] IDS Scheer AG: *IDS Scheer AG: ARIS Platform.* http://www.ids-scheer.de/de/ARIS_ARIS_Platform/7796.html

[6] IDS Scheer: *IDS Scheer AG: Ereignisgesteuerte Prozesskette (EPK).* http://www.ids-scheer.de/de/ARIS/ARIS_Modellierungsstandards/EPK/79890.html

[7] OMG (Object Management Group): *Object Management Group.* http://www.omg.org/

[8] OMG (Object Management Group): *MDA.* http://www.omg.org/mda/

[9] OMG (Object Management Group): *Object Management Group - UML.* http://www.uml.org/

[10] OMG (Object Management Group): *OMG's MetaObject Facility (MOF) Home Page.* http://www.omg.org/mof/

[11] Oracle Corporation: *Oracle and Java | Technologies.* http://www.oracle.com/us/technologies/java/

[12] Oracle Corporation: *Java Metadata Interface (JMI).* http://java.sun.com/products/jmi/

[13] W3C (World Wide Web Consortium): *Extensible Markup Language (XML).* http://www.w3.org/XML/

[14] Workflow Management Coalition: *Home.* http://www.wfmc.org/

[15] OMG (Object Management Group): *BPMN Information Home.* http://www.bpmn.org/

[16] Allweyer, Thomas: *BPMN 2.0 Business Process Model and Notation - Das Buch zum Standard für die Geschäftsprozessmodellierung.* http://www.bpmn-buch.de/BPMNDeutsch.html

[17] W3C (World Wide Web Consortium): *Semantic Web - W3C.* http://www.w3.org/standards/semanticweb/

[18] W3C (World Wide Web Consortium): *W3C Semantic Web Activity.* http://www.w3.org/2001/sw/

[19] W3C (World Wide Web Consortium): *World Wide Web Consortium (W3C).* http://www.w3.org/

[20] W3C (World Wide Web Consortium): *Resource Description Framework (RDF) / W3C Semantic Web Activity.* http://www.w3.org/RDF/

[21] Clark & Parsia, LLC: *Pellet: The Open Source OWL DL Reasoner.* http://clarkparsia.com/pellet/

[22] University of Manchester: *OWL : FaCT++.* http://owl.man.ac.uk/factplusplus/

[23] Forschungszentrum Informatik: *KAON2 – Ontology Management for the Semantic Web.* http://kaon2.semanticweb.org/

[24] ontoprise GmbH: *ontoprise: OntoBroker.* http://www.ontoprise.de/de/produkte/ontobroker/

[25] IBM (International Business Machines): *IBM - WebSphere ILOG Business Rule Management Systems.* http://www.ibm.com/software/websphere/products/business-rule-management/

[26] Fair Isaac Corporation: *Tools für Entscheidungsmanagement.* http://www.fico.com/de/Loesungen/EntscheidungsmanagementTools/Seiten/standard.aspx#Geschäftsregeln

[27] CA Technologies: *CA Aion Business Rules Expert - CA Technologies.* http://www.ca.com/products/product.aspx?id=250

[28] JBoss Community: *Drools.* http://www.jboss.org/drools/

[29] SToP Consortium: *SToP - Stop Tampering of Products.* http://www.stop-project.eu/

[30] LBA (Luftfahrt-Bundesamt): *Luftfahrt Bundesamt - Homepage.* http://www.lba.de/

[31] Air Transport Association of America, Inc.: *Air Transport Association.* http://www.airlines.org/

[32] Europäische Agentur für Flugsicherheit: *European Aviation Safety Agency - easa.europa.eu.* http://www.easa.europa.eu/

[33] LBA (Luftfahrt-Bundesamt): *Luftfahrt Bundesamt - Homepage - Teile zweifelhafter Herkunft.* http://www.lba.de/cln_011/DE/Technik/Fachthemen/Teile_zweifelhafter_Herkunft/Teile_zweifelhafter_Herkunft.html

[34] IDS Scheer AG: *IDS Scheer AG: Der Web-basierte Standard für die unternehmensweite Modellierung von Prozessen und die Geschäftsprozessanalyse.* http://www.ids-scheer.com/de/ARIS/ARIS_Platform/ARIS_Business_Architect/7772.html

[35] IBM (International Business Machines): *IBM - WebSphere Business Modeler Advanced - Features and benefits.* http://www.ibm.com/software/integration/wbimodeler/advanced/features/

[36] Stanford Center for Biomedical Informatics Research: *The Protégé Ontology Editor and Knowledge Acquisition System.* http://protege.stanford.edu/

[37] University of Manchester: *OWL API.* http://owlapi.sourceforge.net/

[38] SAP AG: *SAP - Components & Tools of SAP NetWeaver: SAP NetWeaver Developer Studio.* http://www.sap.com/platform/netweaver/components/developerstudio/

[39] SAP AG: *SAP - Components & Tools of SAP NetWeaver: SAP NetWeaver Composition Environment.* http://www.sap.com/platform/netweaver/components/ce/

[40] Eclipse Foundation, Inc.: *Eclipse Project.* http://www.eclipse.org/eclipse/

[41] Eclipse Foundation, Inc.: *Eclipse Graphical Editing Framework (GEF).* http://www.eclipse.org/gef/

[42] SUPER Consortium: *SUPER Integrated Project - Home.* http://www.ip-super.org/

[43] Clark & Parsia, LLC: *Pellet Integrity Constraint Validator.* http://www.clarkparsia.com/pellet/icv

[44] Oracle Corporation: *Pattern (Java Platform SE 6).* http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html#sum

[45] Singular Systems: *Jep - Java Math Expression Parser - Singular Systems.* http://www.singularsys.com/jep/

[46] Workflow Patterns Initiative: *Workflow Patterns Home Page.* http://www.workflowpatterns.com/

[47] Eclipse Foundation, Inc.: *Eclipse Modeling - EMF - Home.* http://www.eclipse.org/modeling/emf/

[48] Oracle Corporation: *Serializable (Java Platform SE 6).* http://download.oracle.com/javase/6/docs/api/java/io/Serializable.html

[49] W3C (World Wide Web Consortium): *OWL Web Ontology Language Overview.* http://www.w3.org/TR/2004/REC-owl-features-20040210/#AllDifferent

[50] W3C (World Wide Web Consortium): *XQuery 1.0 and XPath 2.0 Functions and Operators.* http://www.w3.org/TR/xpath-functions/#regex-syntax

[51] Daylight Chemical Information Systems, Inc.: *Daylight>Cheminformatics.* http://www.daylight.com/smiles/

[52] Daylight Chemical Information Systems, Inc.: *Daylight Theory: SMARTS - A Language for Describing Molecular Patterns.* http://www.daylight.com/dayhtml/doc/theory/theory.smarts.html

[53] INRIA (Institut National de Recherche en Informatique): *Active XML.* http://www.activexml.net/

[54] Holzmann, Gerard J.: *Spin - Formal Verification.* http://www.spinroot.com/

[55] Microsoft Corporation: *Microsoft Visual Studio 2010 Express.* http://www.microsoft.com/germany/express/