ulm university universität ulm

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Diplomarbeit
im Studiengang Informatik

Versionsübergreifende Konsistenzsicherung – Konzeption und Realisierung von Konsistenzanalysen in einer SOA

vorgelegt von

Florian Finkenzeller

März 2011

Erstgutachter: Prof. Dr. Manfred Reichert
Zweitgutachter: Dr. Thomas Bauer, Daimler AG
Durchgeführt bei: Daimler AG

Kurzfassung

Die fachliche Modellierung von Geschäftsprozessen und deren Ausführung mittels technischer Systeme bilden zentrale Aufgaben bei der Realisierung serviceorientierter Informationssysteme.

Um aus fachlichen Geschäftsprozessen ausführbare technische Prozesse zu erzeugen, werden die fachlichen Prozesse vom Fachbereich an den IT-Bereich gegeben, der sie als technische Prozesse umsetzt und implementiert. Da der Fachbereich und der IT-Bereich bei der Spezifizierung von fachlichen und technischen Prozessen unterschiedliche Granularitätsstufen verwenden, entsteht eine Diskrepanz zwischen Fach- und IT-Bereich, die in der Literatur meist als *Business/IT-Gap* bezeichnet wird. Diese Diskrepanz kommt dann zum tragen, wenn kurzfristig zwischen dem Fachbereich und dem IT-Bereich vermittelt werden soll. Falls beispielsweise im technischen Prozess eine Inkonsistenz auftritt, indem ein technischer Service ausfällt oder außerplanmäßig abgeschaltet wird, muss der fachliche Verantwortliche ermittelt werden. Analog dazu können auch Änderungen vom fachlichen Prozess getrieben sein und der technische Verantwortliche muss gesucht werden.

Ziel einer SOA sind die bessere Unterstützung von Anpassungen von Geschäftsprozessen und die wechselseitige Abstimmung von Anforderungen zwischen Fachbereichen und IT-Bereichen zu verbessern. Dies wird auch als *Business/IT-Alignment* bezeichnet. Um die Beziehungen zwischen den fachlichen Prozessen und ihren technischen Realisierungen bidirektional zu dokumentieren und diese Dokumentation auch nach Prozessänderungen weiterverwenden zu können, wird mit dem Systemmodell eine weitere Modellierungsebene zwischen dem Fachprozess und dem technischen Prozess eingeführt, die die notwendigen Umstrukturierungen zwischen beiden Prozessen speichert.

Die vorliegende Arbeit liefert Konzepte und Algorithmen, welche einerseits zur Identifikation von Inkonsistenzen zwischen Fachprozessen und technischen Prozessen dienen, andererseits den Modellierern konkrete Aufgabenlisten für Korrekturen liefern. Dazu werden die an den Prozessen vorgenommen Änderungen ermittelt und nach jeweiligen Auswirkungen gesucht.

Um die Algorithmen auf ihre Realisierbarkeit zu überprüfen werden sie in einer prototypischen Implementierung realisiert.

Vorwort

Mein Dank gilt Herrn Stephan Buchwald, der die Betreuung dieser Arbeit seitens der Daimler AG übernommen hat und der mir von Anfang an stets mit Anregungen und Kritik zur Seite stand.

Weiterhin möchte ich mich bei Frau Karin Steinhauser und meiner Familie bedanken, die durch ausführliche Korrektur der Rechtschreibung und des Satzbaus zum Gelingen dieser Arbeit beigesteuert haben.

Ein besonderer Dank gebührt Herrn Prof. Dr. Manfred Reichert sowie Dr. Thomas Bauer für die Übernahme des Erst- und Zweitgutachtens.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sinngemäße Übernahmen aus anderen Werken sind als solche kenntlich gemacht und mit genauer Quellenangabe (auch aus elektronischen Medien) versehen.

Ulm, den 28.03.2011 Florian Finkenzeller

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Service-orientiete Architektur (SOA) gehört seit einigen Jahren zu den viel diskutierten Themen in der IT-Branche. Unternehmen erhoffen sich dadurch eine leichtere Strukturierung ihrer IT-Systeme und eine einhergehende Wiederverwendung bestehender Funktionalitäten. So sollen trotz stetig wachsener Komplexität der IT die Kosten gesenkt werden und die Produktivität sowie Qualität verbessert werden. Eine SOA ist keine spezifische Technologie oder ein Produkt, dass man kaufen kann, sondern ein Architektur-Denkmuster für Geschäftsprozesse, die über heterogene Systeme verteilt sind. Die Geschäftsprozesse werden zunächst im Fachbereich erfasst und modelliert, wobei die Analyse der benötigten Prozessschritte im Vordergrund steht. Die Spezifikationen und Beschreibungen sind abstrakt und grobgranular gehalten. Aus den fachlichen Prozessen werden vom IT-Bereich technische Prozesse erzeugt, welche feingranularer und detaillierter spezifiziert sind und die genaue technische Umsetzung beschreiben.

Aufgrund der unterschiedlichen Namenskonvention und Granularitätsstufen der fachlichen und der technischen Prozesse entsteht eine Diskrepanz zwischen Fachbereich und IT-Bereich, die meist als *Business/IT-Gap* bezeichnet wird. Durch die Verwendung einer Service-orientierten Architektur [Jos07][BBP09][Erl05] wird ein besserer Abgleich zwischen Fach- und IT-Bereichen erreicht. Eine Möglichkeit dies zu erreichen besteht darin, eine zusätzliche Ebene zwischen den Bereichen einzuführen (vgl. Abbildung 1.1). Das sogenannte *Systemmodell* stellt eine bidirektionale Beziehung zwischen den fachlichen Prozessen technischen Prozessen her. Es dokumentiert, welche Elemente der fachlichen Prozesse in die Elemente der technischen Prozesse überführt werden beziehungsweise aus welchen Elementen der fachlichen Prozesse die Elemente der technischen Prozese generiert wurden. Mit Hilfe des Systemmodells soll garantiert werden, dass bei Änderungen eines fachlichen Prozesses die Verantwortlichen im technischen Prozess unmittelbar ermittelt werden. Analog dazu sollen auch Änderungen technischer Prozesse behandelt werden.

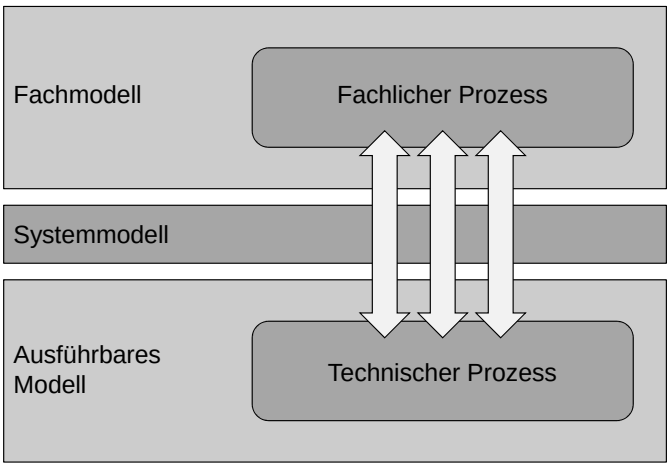

Abbildung 1.1: Ebenen einer durchgängigen Modellierung

Bei Änderungen der Prozesse kann es jedoch zu Inkonsistenzen mit dem Systemmodell kommen, wodurch die Nachvollziehbarkeit nicht mehr gewährleistet werden kann. Aufgrund dieser Problematik müssen Konsistenzchecks durchgeführt werden, die die genauen Fehlerstellen der Inkonsistenzen ermitteln und den Modellierern des fachlichen Prozesses und des technischen Prozesses aufzeigen.

1.1 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, die Konsistenz zwischen den Modellierungsebenen sicherzustellen. Dazu müssen Konzepte entwickelt werden, die neben einfachen Änderungen wie beispielsweise dem Löschen oder Hinzufügen einzelner Aktivitäten im Fachprozess auch zeitliche Aspekte betrachten. Hierbei ist wichtig, dass Informationen über konkrete Zeitstempel in den verschiedenen Modellebenen zugreifbar dokumentiert sind und somit eine Synchronisation modellübergreifend realisiert werden kann. Zudem soll in der Diplomarbeit eine prototypische Implementierung solcher Konsistenzanalysen realisiert werden.

1.2 Aufbau der Arbeit

Die vorliegende Arbeit ist wie folgt gegliedert: In Kapitel 2 werden zunächst die Begrifflichkeiten erläutert, welche zum Verständnis der Arbeit beitragen und die Grundlage für die durchgängige Modellierung von Geschäftsprozessen legen. In diesem Zusammenhang wird insbesondere das Systemmodell ausführlich behandelt, welches die Transformationen zwischen fachlichen und technischen Prozessen ermöglicht. Weiterhin werden Anforderungen an den zu entwickelnden Prototyp spezifiziert.

Kapitel 3 beschreibt Konzepte und Algorithmen zur Identifikation von Änderungen an fachlichen

Prozessen und technischen Prozessen sowie daraus folgende Auswirkungen. Automatismen helfen verantwortlichen Modellierern, Inkonsistenzen zu beheben.

Zu den identifizierten Auswirkungen der Inkonsistenzen wird in Kapitel 4 untersucht, wie die Modellierer durch eine Aufgabenliste in der Inkonsistenzbehebung unterstützt werden können.

In Kapitel 5 wird die Inkonsistenzidentifizierung und -behebung prototypisch umgesetzt. Anschließend werden bestehende wissenschaftliche Ansätze in Kapitel 6 diskutiert, bevor die Arbeit mit einer Zusammenfassung und einem Ausblick schließt.

2

Grundlagen und Anforderungsanalyse

In diesem Kapitel werden grundlegende Begriffe und Rahmenbedingungen erläutert, die zum Verständnis und zur Einordnung der Arbeit notwendig sind.

2.1 Geschäftsprozess

Ein *Geschäftsprozess* (*engl. business process*) ist laut [Jos07] „eine strukturierte Beschreibung der notwendigen *Aktivitäten* oder *Aufgaben* zur Erfüllung einer bestimmten fachlichen Aufgabe. Die Aktivitäten oder Aufgaben können manuelle Schritte (menschliche Interaktion) oder automatisierte Schritte (IT-Schritte) sein. Geschäftsprozesse können verwaltet werden und mit Hilfe von Modellierungsnotationen wie bspw. Business Process Modeling Notation (vgl. [Obj09]) oder Ereignisgesteuerten Prozessketten [Sch01] beschrieben werden.“ Die Verantwortung für die Erstellung und Modellierung sind bei Modellierern des Fachbereichs. Die Beschreibung ist grobgranular und abstrakt gehalten und geht nicht auf jedes Detail ein, um auch für Laien verständlich zu sein.

Abbildung 2.1 zeigt einen Änderungsprozess in der Fahrzeugentwicklung (aus [BBR11]) Der Prozess beginnt, indem ein Fahrzeugentwickler einen Änderungsantrag für ein Bauteil stellt. Danach gibt er die vom Änderungsantrag betroffenen Bauteile an. Darauf folgend detailliert und bewertet der Änderungsverantwortliche den Änderungsantrag, bevor der Baureihenverantwortliche über den Antrag entscheidet. Der Antragsteller wird anschließend durch eine E-Mail über die Entscheidung informiert. Wird der Änderungsantrag genehmigt, wird die Änderung von einem Techniker umgesetzt. Bei einer Ablehnung des Antrags ist der Prozess beendet.

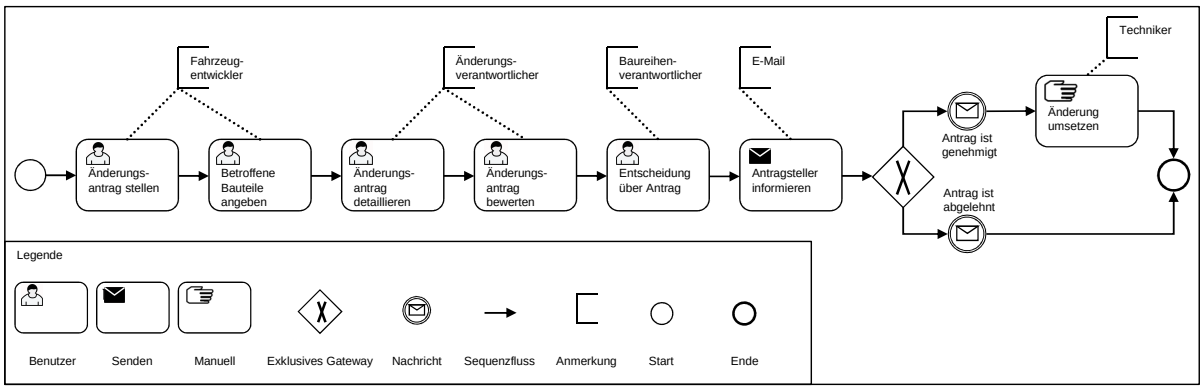

Abbildung 2.1: Fachlicher Geschäftsprozess in BPMN (aus [BBR11])

2.2 Technischer Prozess

Ein technischer Prozess stellt die Realisierung eines Geschäftsprozesses dar, ist also eine ausführbare Implementierung [Wes07]. Im Gegensatz zum Geschäftsprozess ist ein technischer Prozess feingranularer und in seinem Inhalt umfangreicher, seine Aktivitäten sind in ihren Eigenschaften vollständig und formal spezifiziert. Die Abfolge der Aktivitäten kann dabei von der Vorlage des fachlichen Prozesses abweichen. Auch können neue Aktivitäten durch den IT-Bereich eingefügt werden oder die Realisierung einzelner fachlicher Aktivitäten entfallen. Die Modellierungssprache WSBPEL [Web07] ist ein Modellierungssprache von OASIS um Aktivitäten von technischen Prozessen mit Web-Services zu spezifizieren.

Abbildung 2.2 zeigt eine mögliche Realisierung des fachlichen Prozesses aus Abschnitt **??**. Dabei wurden den Aktivitäten englische Namen zugewiesen.

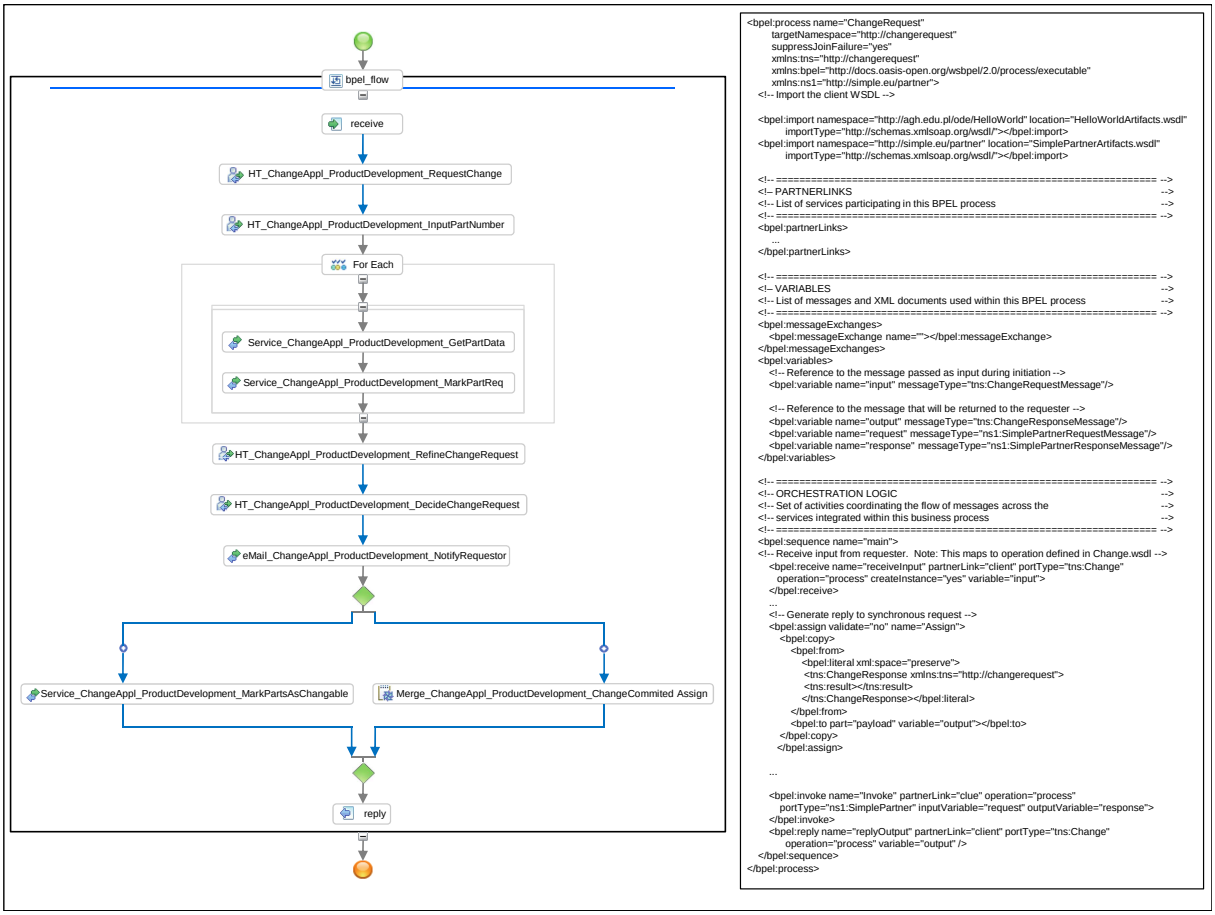

Abbildung 2.2: Technischer Prozess in BPEL [Web07], entwickelt in Websphere Integration Developer [VdPG05] (aus [Tie11])

2.3 Business/IT-Gap

Bei der Erstellung von fachlichen Prozessen und technischen Prozessen sind verschiedene Personengruppen wie zum Beispiel Fachmodellierer, Führungskräfte, Softwareentwickler und Techniker beteiligt. Diese verwenden unterschiedliche Begrifflichkeiten und haben eigene Vorstellungen und Anforderungen an die fachlichen und technischen Prozesse. Bei der Erstellung technischer Prozesse werden diese den fachlichen Prozessen nachempfunden und mit weiteren Details angereichert. Da manche Aktivitäten aus dem Geschäftsprozess jedoch nicht technisch umgesetzt werden können (z.B. *Mit Kunde telefonieren*), sind diese lediglich im Geschäftsprozess dokumentiert. Auch kann es notwendig sein, technische Aktivitäten in den technischen Prozess einzufügen, zu denen keine entsprechende Aktivität im Geschäftsprozess existiert. Es kann vorkommen, dass einzelne Aktivitäten der Geschäftsprozesse in den technischen Prozessen durch mehrere Aktivitäten realisiert werden. Beispielsweise wird ein begleitender Backupschritt eingefügt oder aus technischen Gründen erfolgt eine Aufteilung in zwei Arbeitsschritte. Analog dazu können auch mehrere Aktivitäten der Geschäftsprozesse durch eine einzelne Aktivität

in den technischen Prozessen realisiert werden. Da die Aktivitäten der Geschäftsprozesse häufig sprechende Namen verwenden (z.B. *Änderungsantrag stellen*, vgl. Abb. 2.1), während technische Aktivitäten plattformspezifische Bezeichnungen tragen (z.B. *HT_ChangeAppl_ProductDevelopment_RequestChange*, vgl. Abb. 2.2), ist der Bezug zwischen den Aktivitäten der Geschäftsprozesse und der technischen Prozessen zusätzlich erschwert. Als Ergebnis kann oft nicht erkannt werden, durch welche Umstrukturierungen die technischen Prozesse aus den fachlichen Prozessen entstanden sind.

Um diesem Problem entgegenzuwirken sollen die vorgenommenen Umstrukturierungen in einer durchgängigen Modellierung dokumentiert werden.

2.4 Durchgängige Modellierung

Um das Problem des *Business/IT-Gaps* zu lösen (vgl. Abschnitt 2.3), besteht die Möglichkeit der durchgängigen Modellierung. Zwischen der Modellierungsebene des fachlichen Prozesses und der des technischen Prozesses wird dabei eine weitere Modellebene eingeführt, in der die Umstrukturierungen ihrer Aktivitäten gespeichert werden (siehe Abbildung 2.3).

Das sogenannte *Systemmodell* beinhaltet einen *Systemprozess*, der in Inhalt und Struktur exakt dem technischen Prozess entspricht, das heißt der technische Prozess wird aus dem Systemprozess durch Anreicherung mit plattformspezifischen Informationen erzeugt. Die Verantwortung für die Erstellung und Modellierung des Systemmodells sind bei Modellierern des IT-Bereichs. Änderungen müssen jedoch gemeinsam mit dem Fachbereich besprochen werden.

Der Fachprozess und der Systemprozess weisen die im vorherigen Abschnitt beschriebenen unterschiedlichen Granularitätsstufen auf, die Beschreibung des Fachprozesses ist abstrakter gehalten. Somit kann die Struktur des Fachprozesses nicht direkt in den Systemprozess übernommen werden, sondern muss durch Transformationen umstrukturiert werden. Die Menge aller Transformationen bildet das so genannte *Abbildungsmodell*, welches ebenfalls ein Teil des Systemmodells ist. Jede der Transformationen zeigt für bestimmte Aktivitäten des Fachprozesses auf, wie sie in den Systemprozess überführt werden. Ebenso ist über die Transformationen erkennbar, aus welchen fachlichen Aktivitäten sie entstanden sind. Die Reihenfolge der Aktivitäten des Fachprozesses und des Systemprozesses werden nicht in das Abbildungsmodell übernommen, da der Fokus auf der Dokumentation der Transformationen liegt [BBR09]. Um die Bedeutung des Abbildungsmodells hervorzuheben und die Abbildung zu vereinfachen, wird es in Abbildung 2.3 und folgenden als separate Modellierungsebene dargestellt.

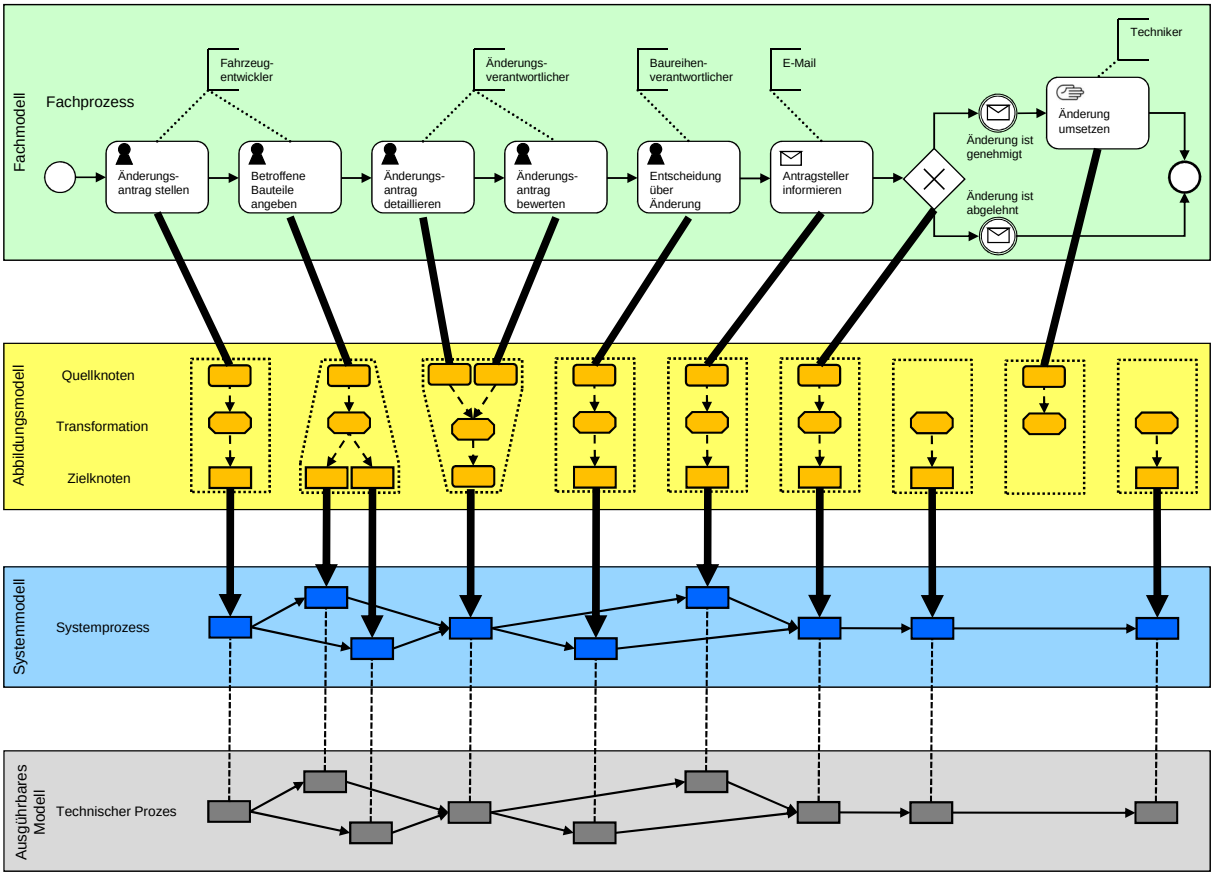

Abbildung 2.3: Ebenen einer durchgängigen Modellierung

Die Transformationen des Abbildungsmodells bilden nicht direkt die Aktivitäten des Fachprozesses auf die Aktivitäten des Systemprozesses ab, sondern sie arbeiten mit Kopien. Die Kopien der Fachprozessaktivitäten bezeichnet man als *Quellknoten*, die Kopien der Systemprozessaktivitäten als *Zielknoten* (vgl. Abbildung 2.4). Die Transformationen können in fünf verschiedene Typen unterschieden werden [BBR10] (vgl. Abbildung 2.4).

Der erste Typ (*Map*), bildet eine Aktivität des Fachprozesses direkt eine Aktivität des Systemprozesses ab. Die Eigenschaften der beidseitigen Aktivitäten dürfen dabei von einander abweichen und unterschiedliche Bezeichnungen tragen oder generell unterschiedliche Detaillierungen aufweisen. Soll eine fachliche Aktivität auf technischer Ebene verfeinert und in mehrere Schritte unterteilt werden, so kann dies durch Transformationen des zweiten Typs (*Split*) realisiert werden. Gründe sind beispielsweise eine verbesserte Parallelisierbarkeit, oder die Unterstützung einer Benutzerinteraktion durch einen Service. Ist es hingegen erwünscht, dass mehrere Aktivitäten des Fachprozesses zu einer Aktivität des Systemprozesses zusammengefasst werden, so wird dies über Transformationen des drittens Typs (*Merge*) realisiert. Ein Beispiel hierfür sind die beiden vom Änderungsverantwortlichen durchgeführten Aktivitäten *Änderungsantrag detaillieren* und *Änderungsantrag bewerten*, des Fachprozesses (vgl. Abbildung 2.3). Falls Aktivitäten des Fachprozesses aufgrund mangelnder technischer Relevanz im Systemprozess weggelassen werden müssen (z.B. *Änderung umsetzen*), so wird dies durch Transformationen des vierten Typs

(*Delete*) realisiert. Falls im umgekehrten Fall Aktivitäten in den Systemprozess aufgenommen werden, die nicht im Fachprozess dokumentiert sind, werden diese über Transformationen des fünften Typs (*Insert*) eingefügt.

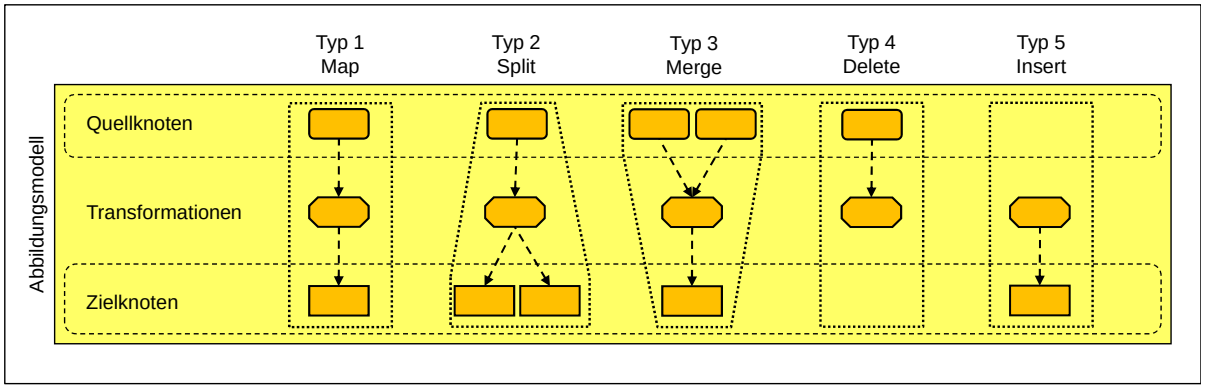

Abbildung 2.4: Abbildungsmodell

Da die Beziehung zwischen den Aktivitäten des Systemprozesses und des ausführbaren technischen Prozesses triviale 1:1 Beziehungen darstellen (siehe Abbildung 2.3), betrachten wir im Folgenden lediglich die Transformationen von Fachprozessen über das Abbildungsmodell in Systemprozesse.

2.5 Anforderungen an Konsitenzalgorithmen

Die durchgängige Modellierung von Geschäftsprozessen führt das Systemmodell als eine zusätzliche Modellebene zwischen dem Fachmodell und dem ausführbaren Modell ein. Im Systemmodell wird zu jedem Element des Fachprozesses über eine Transformation definiert, welchen Elementen des Systemprozesses es zugehörig ist. Die Nachvollziehbarkeit zwischen den Elementen beider Prozesse ist nur dann gegeben, falls die Modelle zueinander konsistent sind. Daraus ergeben sich die folgenden Anforderungen:

Anforderung 2.1 (Konsistenz der Modelle):
Das Fachmodell, das Systemmodell und das Abbildungsmodell müssen konsistent sein. Man muss nachvollziehen können, welche Prozessschritte des Fachmodells in welchen Prozessschritten des Systemmodells realisiert werden. Dies muss sowohl bei der Ersterstellung der Modelle, wie auch nach ihrer Weiterentwicklung gelten.

Anforderung 2.2 (Inkonsistenzerkennung):
Die Konsistenz des Fachmodells, des Systemmodells und des Abbildungsmodells muss automatisiert überprüft werden. Falls die Modelle inkonsistent sind, so muss dies unmittelbar erkannt und den Modellierern mitgeteilt werden, um Korrekturmaßnahmen einzuleiten.

Anforderung 2.3 (Ursachen einer Inkonsistenz):
Zu einer Inkonsistenz der Modelle muss bekannt sein, durch welche Modellelemente sie ausgelöst wird. Insbesondere ist zu ermitteln, ob die entsprechenden Elemente bei der Weiterentwicklung der Modelle neu hinzugefügt wurden, entfernt wurden, oder in ihren Eigenschaften bearbeitet.

Anforderung 2.4 (Auswirkungen einer Inkonsistenz):
Zu jedem im Fachmodell bearbeiteten Element, das eine Inkonsistenz verursacht, müssen diejenigen Elemente des Systemmodells und des Abbildungsmodells bekannt sein, durch deren Anpassung die Inkonsistenz behoben wird. Gleichfalls müssen zu bearbeiteten Elemente des Systemmodells die zu korrigierenden Elemente des Fachmodells und des Abbildungsmodells bekannt sein.

Anforderung 2.5 (Auflösung der Inkonsistenz):
Zur Auflösung der Inkonsistenz benötigen die Modellierer eine Liste der Inkonsistenzursachen, wie auch der Inkonsistenzauswirkungen. Um die Behebung der Inkonsistenz als komfortabel als möglich durchzuführen, muss daraus eine einfach verständliche Aufgabenliste generiert werden, die konstruktive Korrekturmöglichkeiten aufzeigt.

3

Konsistenzsicherung von Modellen

Ziel der durchgängigen Modellierung (vgl. Abschnitt 2.4) ist, zwischen Aktivitäten von Fachprozessen (vgl. Abschnitt 2.1) und Aktivitäten von Systemprozessen (vgl. Abschnitt 2.4) eine bidirektionale Nachvollziehbarkeit herzustellen. Dies wird über das Abbildungsmodell erreicht, welches die vorgenommenen Transformationen enthält.

Verbleiben Fach- und Systemprozess unverändert, so können die jeweils zusammengehörigen Aktivitäten über das Abbildungsmodell ermittelt werden. Wird jedoch der Fachprozess aufgrund neuer fachlicher Anforderungen geändert oder der Systemprozess aufgrund neuer technischer Anforderungen, so können die zusammengehörigen Elemente nicht mehr über das Abbildungsmodell ermittelt werden. Um die Nachvollziehbarkeit wieder herzustellen, müssen die Prozesse aneinander angepasst werden.

Abbildung 3.1 zeigt hierzu ein Beispiel. Ein Fachprozess ① enthält mehrere Prozessschritte, die jeweils über eine Transformation im Abbildungsmodell ② mit Prozessschritten des Systemprozesses ③ verbunden sind. Über das Abbildungsmodell ist somit eine Nachvollziehbarkeit gegeben. Der Fachprozess wird vom Fachmodellierer weiterentwickelt ④ und liegt anschließend mit neuem Inhalt vor ⑤. Für die Modellierer des IT-Bereichs ergibt sich die Frage, inwiefern das Abbildungsmodell wegen möglicher Inkonsistenzen an diese Prozessänderung angepasst werden soll ⑥. Gleichfalls haben sie für den Systemprozess zu entscheiden, inwiefern dieser angepasst werden soll ⑦.

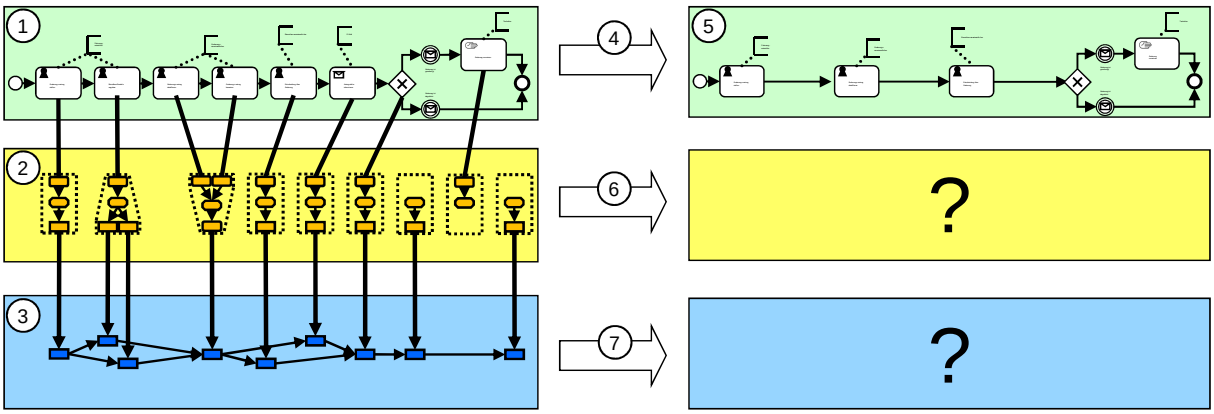

Abbildung 3.1: Ausgangssituation bei Weiterentwicklung des Fachprozesses

Bei Prozessänderungen müssen die Modellierer zunächst überprüfen, ob Änderungsauswirkungen in den bearbeiteten Prozessen selbst existieren. Abbildung 3.2 zeigt als Beispiel einen Fachprozess, dessen drei Aktivitäten *a*, *b* und *c* auf die zwei Datenobjekte *x* und *y* schreibend zugreifen. Die Schreiboperationen stellen eine Abhängigkeit dar, weshalb die Löschung des Datenobjekts *x* ① dazu führt, dass all diejenigen Fachprozessknoten angepasst werden müssen, die auf das Datenobjekt lesend oder schreibend zugreifen. In der Abbildung sind dies die Knoten *a* und *b* ②.

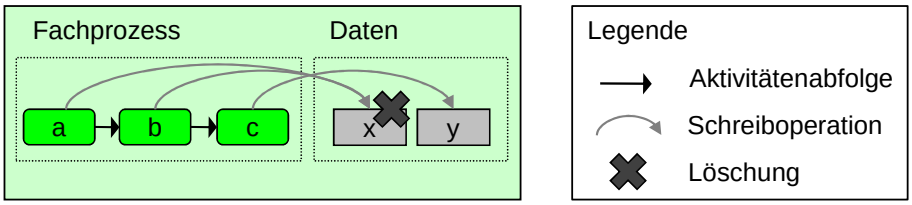

Abbildung 3.2: Abhängigkeiten im Fachprozess

Weitere Überprüfungen der Prozessänderungen sind notwendig, da nicht jede Änderung am Fach- oder Systemprozess zu einer Inkonsistenz zwischen Fachprozess und Systemprozess führt. Dies hängt zum einen davon ab, ob die Folgeversionen der Prozesse (vgl. Abschnitt 3.1) relevante Informationen beinhalten. Das Setzen eines Kommentars ist beispielsweise nicht für die Konsistenz der Prozesse bestimmend. Zum anderen ist die Ursache der Änderung entscheidend, das heißt ob die Änderung vom Fachbereich oder vom IT-Bereich veranlasst wurde. Bei einem geänderten Fachprozess ist nur dann der Systemprozess anzupassen, falls die Änderung initial vom Fachprozess selbst ausgeht, also durch neue fachliche Anforderungen angestoßen wird. Falls die Änderung des Fachprozesses hingegen eine Anpassung an den Systemprozess ist, so benötigt der Systemprozess entsprechend keine Anpassung. Ebenfalls ist bei der Bearbeitung eines Systemprozesses darauf zu achten, ob dies eine Anpassung an den Fachprozess darstellt oder aber initial vorgenommen wird. Ohne genaue Untersuchung dieser Aspekte einer Prozessweiterentwicklung ist für die Modellierer unklar, ob sie eine Anpassung

des korrespondierenden Prozesses durchführen müssen.

Zur Korrektur des korrespondierenden Prozesses benötigen die Modellierer exakte Informationen über diejenigen Prozesselemente, an denen die Inkonsistenz auftritt. Diese müssen sie mit Hilfe der bisher im Abbildungsmodell dokumentierten Beziehungen ermitteln. Wie im bearbeiteten Prozess kann es jedoch auch im korrespondierenden Prozess aufgrund von Abhängigkeiten dazu kommen, dass weitere Objekte von den Modellierern anzupassen sind. Die Modellierer müssen auch indirekt von einer Inkonsistenz betroffene Prozesselemente überprüfen.

Eine neue Ausgangssituation entsteht dann, wenn sowohl der Fachprozess als auch der Systemprozess zeitnah bearbeitet werden, ohne dass zwischenzeitlich eine Anpassung in eine Richtung vorgenommen wird. Dadurch ergibt sich, dass sowohl Änderungen des Fachprozesses in Richtung Systemprozess nachvollzogen werden müssen, als auch Änderungen des Systemprozesses in Richtung Fachprozess. Für die Modellierer stellt sich in einer solchen Situation die Frage, welchem der Prozesse sie den Vorrang einräumen sollen. Wir geben hierzu einige Beispiele und Lösungsansätze in Abschnitt 3.5.

Haben die Modellierer nach diversen Änderungen am Fach- und Systemprozess die beiden Prozesse untereinander angepasst, so bleibt abschließend ein Kontrollschritt zu tätigen. Sie müssen untersuchen, ob die Abgleichung zwischen den Prozessen fehlerfrei und vollständig ablief und die Prozesse somit zueinander konsistent sind. Wir beschreiben in Abschnitt 3.8 ein entsprechendes Kontrollverfahren, welches die Konsistenz der Prozesse anhand ausgewählter Kriterien bewertet.

In diesem Kapitel beschreiben wir in Abschnitt 3.1 die Versionierung von Fach- und Systemprozessen, bevor wir uns mit der Identifikation von Prozessänderungen in Abschnitt 3.2 beschäftigen. In Abschnitt 3.3 wird gezeigt, wie die Richtung einer Prozessanpassung erkannt werden kann. Die Analyse der Auswirkungen von Prozessänderungen in der jeweils korrespondierenden Modellierungsebene wird in Abschnitt 3.4 beschrieben. Abschnitt 3.5 gezeigt, dass sich Fach- und Systemprozesse gegenseitig überschreiben können. Algorithmen, die die Abhängigkeiten im bearbeiteten Prozess selbst identifizieren, werden in Abschnitt 3.6 behandelt. Die Reihenfolge, die Modellierer bei der Anpassung inkonsistenter Prozesse durchführen müssen, wird in Abschnitt 3.7 ausführlich beschrieben. Ob nach erfolgten Korrekturen die Konsistenz der Prozesse gegeben ist, wird in Abschnitt 3.8 überprüft. Den Abschluss bildet eine Zusammenfassung über die im Kapitel besprochenen Algorithmen.

3.1 Versionen von Prozessen

Die Änderungen am Fachprozess und Systemprozess kann man in zwei Gruppen einteilen: Einerseits kann ein Prozess in seinem bisherigen, unveränderten Inhalt beibehalten werden, wobei lediglich der Zeitstempel der letzten Bearbeitung aktualisiert wird. Andererseits kann ein Prozess tatsächlich im Inhalt berührt werden, neue Details erhalten, und ebenfalls mit aktuellem Zeitstempel gespeichet werden. Man bezeichnet jeden neuen Bearbeitungszustand des Fach- und Systemprozesses, der über letztere Art von Änderungen einen neuen Inhalt erhalten hat, als eine *(Prozess-)Version*. Die einzelnen Versionen unterscheidet man, indem man ihnen je

einen eindeutigen Bezeichner zuweist. Man kann die Versionen eines Prozesses miteinander in Beziehung setzen und durch Vergleiche Änderungen erkennen sowie dokumentieren. Wir erklären im Folgenden, wie durch die Änderung eines Prozesses respektive seiner Elemente eine neue *Prozessversion* entsteht.

Einen Fachprozess *fp'* bezeichet man als Folgeversion eines Fachprozesses *fp*, falls *fp'* durch Änderungen an mindestens einem der Elemente von *fp* entstand. Abbildung 3.3 zeigt dies an einem Beispiel. Der Fachprozess enthält in seiner ersten Version *fp* insgesamt drei Aktivitäten. Die am Prozess vorgenommene Bearbeitung ist die das Entfernen der dritten Aktivität. Dadurch entsteht die neue Prozessversion *fp'* mit insgesamt zwei Aktivitäten.

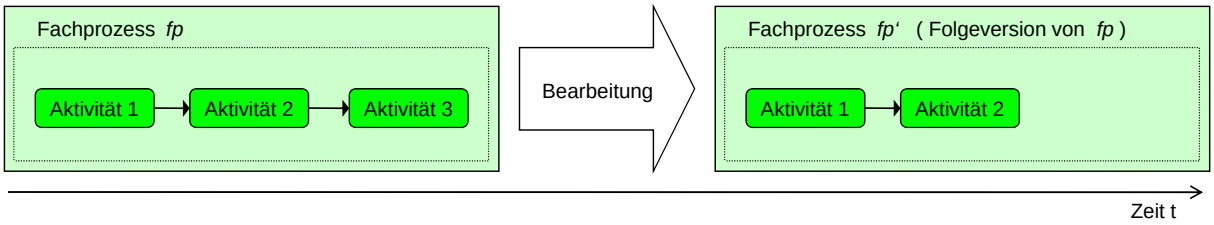

Abbildung 3.3: Folgeversion eines Fachprozesses

Die vom Fachprozess verwalteten weiteren Informationen, wie verwendete Datenobjekte, verwendete Geschäftsregeln und Services, sind mit dieser Bezeichnung nicht mit eingeschlossen. Ist im Folgenden die gesamte Modellierungsebene des Fachprozesses inklusive dieser weiteren Bestandteile gemeint, verwenden wir die Bezeichnung *Fachmodell*. Fachmodelle können ebenfalls in Versionen vorliegen. Zur leichteren Darstellung werden in Abbildungen lediglich der Prozess sowie die Datenobjekte gezeigt. Ein Fachmodell *fm'* bezeichet man als Folgeversion eines Fachmodells *fm*, falls *fm'* durch Änderungen an mindestens einem der Elemente (Prozess, Datenobjekte, Geschäftsregeln und Services) von *fm* entstand. Zur Versionierung eines Fachmodells zeigt Abbildung 3.4 die Versionen *fm* mit seiner Folgeversion *fm'*. Die zwei Versionen entstehen dadurch, dass die Aktivität *c* und das Datenobjekt *d2* gelöscht werden. Die Aktivität *b* wird verändert, indem sie auf ein anderes Datenobjekt schreibt.

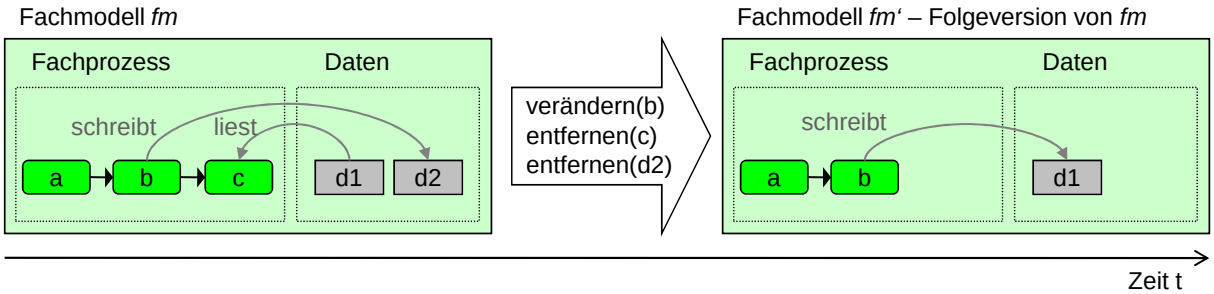

Abbildung 3.4: Folgeversion eines Fachmodells

Analog zur Versionierung der Fachprozess definiert man die Versionierung der Systemprozesse:

Ein Systemprozess *sp'* bezeichnet man als Folgeversion eines Systemprozesses *sp* genau dann, wenn *sp'* durch Veränderungen an den Elementen von *sp* entstand.

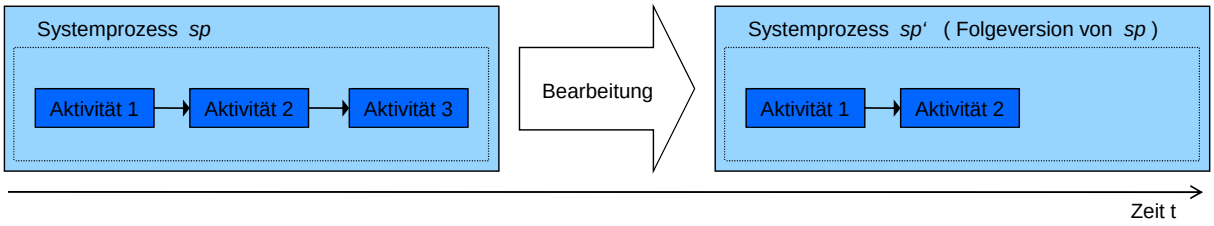

Abbildung 3.5: Folgeversion eines Systemprozesses

Die Modellierungsebene des Systemprozesses beinhaltet analog zum Fachmodell den Systemprozess, die Datenobjekte, Geschäftsregeln und Services. Diese Gesamtheit nennt man ein *Systemmodell*, welches ebenfalls in Versionen vorliegen kann. Ein Systemmodell *sm'* bezeichnet man als Folgeversion eines Systemmodells *sm* genau dann, wenn *sm'* durch Veränderungen an den Elementen von *sm* entstand. Im Folgenden reden wir von Systemmodell, wenn wir die Modellierungsebene des Systemprozesses meinen.

3.2 Identifizieren von Änderungen

Entwicket sich das Fachmodell oder das Systemmodell weiter, so stellt sich für den Modellierer die Frage, ob die Modelle weiterhin zueinander konsistent sind. Um dies beurteilen zu können, benötigt er detaillierte Informationen über die konkret vorgenommenen Änderungen. Wir unterscheiden zwischen drei Arten von Modelländerungen: Dem Hinzufügen neuer Objekte, dem Entfernen von Objekten und dem Bearbeiten der Eigenschaften von Objekten. Um feststellen zu können, welchen dieser drei Kategorien von Änderungen die Elemente eines Modells zugeordnet werden können und welche unverändert beibehalten wurden, vergleichen wir das Modell mit seiner Vorversion (siehe Abschnitt 3.1). Abbildung 3.6 zeigt als Beispiel ein Fachmodell *fm'* mit seiner Vorversion *fm*, sowie ein Systemmodell *sm'* mit seiner Vorversion *sm*. Der Fachmodellierer möchte zum Fachmodell wissen, inwiefern sich dessen Modellversionen unterscheiden. Der Systemmodellierer hingegen möchte wissen, inwiefern sich die Modellversionen des Systemmodells unterscheiden.

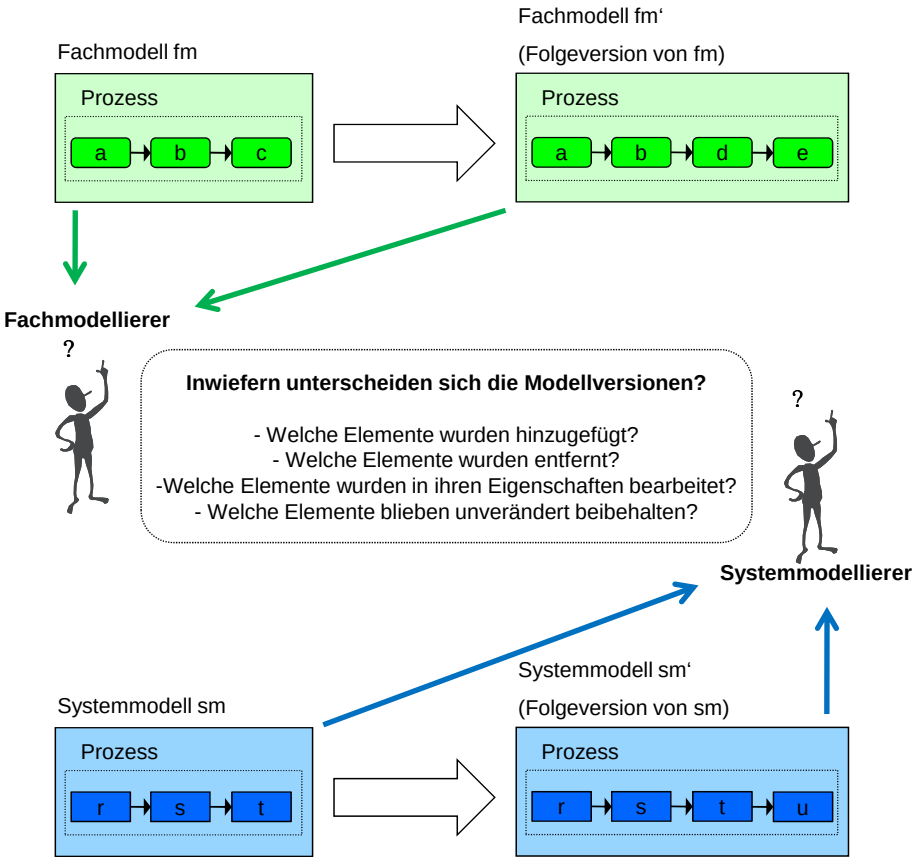

Abbildung 3.6: Motivation zur Identifizierung von Modelländerungen

An der Anzahl der Prozessaktivitäten beider Modellversionen *fm* und *fm'* des Fachmodells sehen wir, dass diesem neue Elemente hinzugefügt wurden. Gleichfalls weist das Systemmodell in seiner Folgeversion *sm'* mehr Prozessaktivitäten auf. Die konkret zu den Modellen hinzugefügten Prozessschritte erhalten wir, indem wir zu jedem Element der Folgeversion untersuchen, ob dieses ebenfalls als Element der Vorversion enthalten ist. Der folgende Algorithmus 3.1 setzt dies um und liefert die Menge der hinzugefügten Prozessschritte in der Menge *Added*.

Algorithmus 3.1 (Identifizierung hinzugefügter Modellelemente)**:**

```
Input:  N               Die Menge der Prozesselemente der Vorversion des
                        untersuchten Modells
Input:  N'              Die Menge der Prozesselemente der Folgeversion des
                        untersuchten Modells
Output: Added           Die Menge der zum Modell hinzugefügten Prozesselemente

for(element:N')         Untersuche zu jedem Prozesselement der Menge N'
  if(element.notIn(N))  ob es in der Menge N der Vorversion enthalten ist
                        Falls nicht,
    Added.add(element)  vermerke, dass es neu erstellt wurde
  fi
end
```

Wendet man Algorithmus 3.1 auf das Fachmodell aus Abbildung 3.6 an, so werden die Prozessaktivitäten *a*, *b*, *d* und *e* der Fachmodellversion *fm'* untersucht, ob diese ebenfalls als Elemente der Modellversion *fm* enthalten sind. Wie aus der Abbildung ersichtlich, trifft dies für *d* und *e* nicht zu. Beide Elemente werden nachweisbar hinzugefügt. Zum Systemmodell aus Abbildung 3.6 werden die Prozessaktivitäten *r*, *s*, *t* und *u* der Systemmodellversion *sm'* untersucht, ob diese ebenfalls als Elemente der Modellversion *sm* enthalten sind. Lediglich das Element *u* ist in der Vorversion nicht enthalten und wird demnach als hinzugefügt identifiziert.

Mit einem ähnlichen Algorithmus werden die entfernten Modellelemente gesucht. Man identifiziert sie, indem man zu jedem Element der Vorversion untersucht, ob dieses nicht mehr als Element der Folgeversion enthalten ist. Die Menge der entfernten Modellelemente wird in Algorithmus 3.2 mit *Removed* bezeichnet.

Algorithmus 3.2 (Identifizierung entfernter Modellelemente)**:**

```
Input:  N               Die Menge der Prozesselemente der Vorversion des
                        untersuchten Modells
Input:  N'              Die Menge der Prozesselemente der Folgeversion des
                        untersuchten Modells
Output: Removed         Die Menge der entfernten Prozesselemente

for(element:N)          Untersuche zu jedem Prozesselement der Menge N
  if(element.notIn(N')) ob es in der Menge N' der Folgeversion enthalten ist
                        Falls nicht,
    Removed.add(element)vermerke, dass es entfernt wurde
  fi
end
```

Zur Fachmodellversion *fm* aus Abbildung 3.6 untersucht der Algorithmus 3.2 die Prozessaktivitäten *a*, *b* und *c*. Die Prozessaktivitäten *a* und *b* sind beide in der Folgeversion *fm'* enthalten, nicht hingegen *c*. Somit erkennt der Algorithmus *c* als ein aus dem Fachmodell entferntes Element. Zum Systemmodell liefert der Algorithmus die leere Menge, da weder *r*, *s*, noch *t* entfernt

werden.

Diejenigen Modellelemente, die weder hinzugefügt noch entfernt wurden, wurden in der Konsequenz beibehalten. Um diese Menge der beibehaltenen Modellelemente unabhängig zu berechnen, kann nach dem Algorithmus 3.3 vorgegangen werden. Die Ergebnismenge wird mit *Reused* bezeichnet.

Algorithmus 3.3 (Identifizierung beibehaltener Modellelemente)**:**

```
Input:  N               Die Menge der Prozesselemente der Vorversion des
                        untersuchten Modells
Input:  N'              Die Menge der Prozesselemente der Folgeversion des
                        untersuchten Modells
Output: Reused          Die Menge der beibehaltenen Prozesselemente

for(element:N)          Untersuche zu jedem Prozesselement der Menge N
  if(element.in(N'))    ob es in der Menge N' der Folgeversion enthalten ist
                        Falls ja,
    Reused.add(element) vermerke, dass es beibehalten wurde
  fi
end
```

Mit obigem Algorithmus angewandt auf das Fachmodell aus Abbildung 3.6 werden die Prozesselemente *a*, *b*, und *c* untersucht, ob sie ebenfalls in der Modellversion *fm'* enthalten sind. Dies trifft für *a* und *b* zu. Im Systemmodell wurden alle drei Modellelemente *r*, *s* und *t* übernommen.

Algorithmus 3.3 ermittelt zu einem Modell die Menge der beibehaltenen Modellelemente. Man erhält jedoch keine Informationen darüber, ob diese in ihren Eigenschaften unbearbeitet bleiben, oder ob diese bearbeitet werden und sich somit von Modellversion zu Modellversion unterscheiden. Ein Beispiel hierfür zeigt Abbildung 3.7 mit einem Fachmodell in der Version *fm* und der Folgeversion *fm'*. In beiden Versionen enthält das Fachmodell die Prozessschritte *a* und *b*. Das Element *a* enthält in beiden Versionen unterschiedliche Eigenschaften, das Element *b* bleibt unbearbeitet.

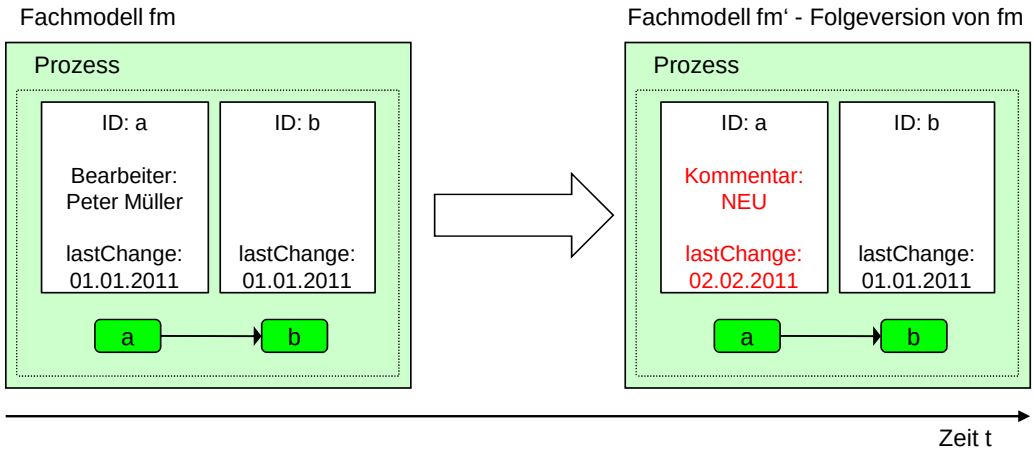

Abbildung 3.7: Bearbeitung der Attribute eines Fachmodell-Elements

Erst wenn man diese Eigenschaften der Modellelemente, auch Attribute genannt, in einem Vergleich der zwei Modellversionen mit einbezieht, kann man die in ihren Eigenschaften bearbeiteten Modellelemente erkennen. Der Algorithmus zur Ermittlung dieser Objekte lautet wie folgt:

Algorithmus 3.4 (Identifizierung bearbeiteter Modellelemente 1)**:**

```
Input:  N               Die Menge der Prozesselemente der Vorversion des
                        untersuchten Modells
Input:  N'              Die Menge der Prozesselemente der Folgeversion des
                        untersuchten Modells
Output: Modified        Die Menge der in ihren Eigenschaften bearbeiteten
                        Prozesselemente

for(element1:N)         Untersuche zu jedem Prozesselement der Menge N
  for(element2:N')      und jedem Prozesselement der Menge N'
                        die den gleichen Bezeichner tragen,
                        ob diese unterschiedliche Attribute aufweisen
                        Falls ja, vermerke das Prozesselement als geändert
    if(element1.getID==element2.getID)
      if(element1.getAttributes!=element2.getAttributes)
        Modified.add(element2)
      fi
    fi
  end
end
```

Obiger Algorithmus, angewandt auf das Fachmodell *fm* der Abbildung 3.7 und seine Folgeversion *fm'*, liefert als Ergebnis das Modellelement *a*, da dieses in der Modellfolgeversion das neue Attribut *Kommentar* aufweist, das Attribut *Bearbeiter* nicht mehr enthält, und zudem einen

jüngeren Zeitstempel trägt.

Da mit jeder Änderung eines Objektattributs ein neuer Zeitpunkt der letzten Bearbeitung vermerkt wird, reicht bereits die Kontrolle dieses einen Attributs zur Identifizierung geänderter Attribute aus. Der Algorithmus dazu lautet wie folgt:

Algorithmus 3.5 (Identifizierung bearbeiteter Modellelemente 2)**:**

```
Input:  N               Die Menge der Prozesselemente der Vorversion des
                        untersuchten Modells
Input:  N'              Die Menge der Prozesselemente der Folgeversion des
                        untersuchten Modells
Output: Modified        Die Menge der in ihren Eigenschaften bearbeiteten
                        Prozesselemente

for(element1:N)         Untersuche zu jedem Prozesselement der Menge N
  for(element2:N')      und jedem Prozesselement der Menge N'
                        die den gleichen Bezeichner tragen,
                        ob diese unterschiedliche Zeitstempel aufweisen
                        Falls ja, vermerke das Prozesselement als geändert
    if(element1.getID==element2.getID)
      if(element1.getTimestamp<element2.getTimestamp)
        Modified.add(element2)
      fi
    fi
  end
end
```

Mit den zwei zuletzt aufgeführten Algorithmen kann man identifizieren, welche Modellelemente in ihren Eigenschaften bearbeitet wurden. Der konkrete Inhalt dieser Änderungen ist jedoch nicht bekannt. Beispielsweise ist durch die Algorithmen nicht bekannt, dass Modellelement *a* aus Abbildung 3.7 ein weiteres Attribut erhält, wie dieses benannt ist, und welchen Wert es beinhaltet. Man benötigt weitere Algorithmen, die diese Information durch einen genauen Vergleich der Attribute liefern.

Bei den Modellelementeattributen gibt es wie bei den Modellelementen drei Arten von Änderungen: Dem Hinzufügen neuer Attribute, dem Entfernen von Attributen und dem Bearbeiten von Attributwerten. Mit dieser Analogie zwischen Modellelementen und ihren Attributen ergibt sich, dass man einige der bereits besprochenen Algorithmen in leicht abgewandelter Form für die Eigenschaften der Modellelemente einsetzen kann. Der Unterschied ist, dass bisherige Algorithmen auf Fach- oder Systemmodelle angewandt werden, die folgenden Algorithmen hingegen die einzelnen Modellelemente als Eingabe erhalten.

Algorithmus 3.6 identifiziert, welche Attribute zu einem Prozessschritt hinzugefügt werden, indem er dessen alten und neuen Zustand als Eingabe erhält und deren Attribute vergleicht.

Algorithmus 3.6 (Identifizierung hinzugefügter Attribute)**:**

```
Input:  element1                Ein Element der Vorversion des untersuchten
                                Modells
Input:  element2                Folgeversion des Elements element1 in der
                                Folgeversion des untersuchten Modells
Output: AttributesAdded         die Menge der zum Element hinzugefügten
                                Attribute

Attributes1=element1.getAttributes      Ermittle die Attribute Attributes1
                                        des Elements element1
Attributes2=element2.getAttributes      Ermittle die Attribute Attributes2
                                        des Elements element2
for(attribute:Attributes2)              Untersuche zu jedem Attribut aus
                                        Attributes2
  if(attribute.notIn(Attributes1))      ob es in den Attributen Attributes1
                                        enthalten ist. Falls nicht,
    AttributesAdded.add(attribute)      vermerke, dass es hinzugefügt wurde
  fi
end
```

Wendet man den Algorithmus 3.6 auf das Modellelement *a* in Abbildung 3.7 an, so erkennt man, dass es um das Attribut *Kommentar* erweitert wird. Analysiert man das Element *b*, so erhält man die leere Menge, da dieses nicht bearbeitet wird. Um zu identifizieren, welche Elementattribute entfernt werden, wendet man folgenden Algorithmus an:

Algorithmus 3.7 (Identifizierung entfernter Attribute)**:**

```
Input:  element1                Ein Element der Vorversion des untersuchten
                                Modells
Input:  element2                Folgeversion des Elements element1 in der
                                Folgeversion des des untersuchten Modells
Output: AttributesRemoved       Die Menge der entfernten Attribute

Attributes1=element1.getAttributes      Ermittle die Attribute Attributes1
                                        des Elements element1
Attributes2=element2.getAttributes      Ermittle die Attribute Attributes2
                                        des Elements element2
for(attribute:Attributes1)              Untersuche zu jedem Attribut aus
                                        Attributes1
  if(attribute.notIn(Attributes2))      ob es in den Attributen Attributes2
                                        enthalten ist. Falls nicht,
    AttributesRemoved.add(attribute)    vermerke, dass es entfernt wurde
  fi
end
```

Dieser Algorithmus, angewandt auf das Element *a* des Fachmodells aus Abbildung 3.7, kontrolliert, ob die Attribute *ID*, *Bearbeiter* und *lastChange* der Vorversion weiterhin in der Folgeversion

enthalten sind. Für *ID* und *lastChange* trifft dies zu. Demnach wird das Attribut *Bearbeiter* entfernt.

Der folgende Algorithmus zeigt auf, wie die in ihren Werten geänderten Attribute identifiziert werden:

Algorithmus 3.8 (Identifizierung eines geänderten Attributwerts)**:**

```
Input:  element1               Ein Element der Vorversion des untersuchten
                               Modells
Input:  element2               Folgeversion des Elements element1 in der
                               Folgeversion des des untersuchten Modells
Output: AttributesEdited       Die Menge der bearbeiteten Attribute

Attributes1=element1.getAttributes      Ermittle die Attribute Attributes1
                                        des Elements element1
Attributes2=element2.getAttributes      Ermittle die Attribute Attributes2
                                        des Elements element2
for(attribute1:Attributes1)             Untersuche zu jedem Attribut aus
                                        Attributes1
  for(attribute2:Attributes2)           und zu jedem Attribut aus Attributes2
                                        die den gleichen Namen tragen,
                                        ob diese unterschiedliche Werte
                                        aufweisen. Falls ja, vermerke, dass
                                        der Attributwert bearbeitet wurde.
    if(element1.getName==element2.getName)
      if(attribute1.getValue=attribute2.getValue)
        AttributesEdited.add(attribute2)
      fi
    fi
  end
end
```

Zum Objekt *a* des Fachmodells aus Abbildung 3.7 liefert der Algorithmus als Ergebnis das Attribut *lastChange*, da dieses in beiden Modellversionen vorkommt, jedoch mit *01.01.2011* und *02.02.2011* zwei unterschiedliche Werte beinhaltet.

Mit den in diesem Abschnitt behandelten Algorithmen identifiziert man, an welchen Elementen der Prozesse des Fachmodells und Systemmodells Änderungen vorgenommen werden. Diese lassen sich in drei Gruppen unterteilen: hinzugefügte Elemente, gelöschte Elemente, und in ihren Eigenschaften geänderte Elemente. Zur Analyse vergleichen die Algorithmen die Anzahl der Elemente, ihre Zeitstempel der letzten Bearbeitung, und ihre Attribute. In gleicher Weise lassen sich auch die weiteren Modellelemente von Fach- und Systemmodellen, die Datenobjekte, die Regeln und die Services, analysieren. Aufbauend auf der Ergebnismenge der Algorithmen wird in den weiteren Abschnitten beleuchtet, welche Auswirkungen Änderungen auf die Konsistenz der Modelle haben.

3.3 Identifizierung der Änderungsrichtung

Stellen die Modellierer fest, dass ein Modell geändert wurde, so benötigen sie Informationen darüber, weshalb die Änderung vorgenommen wurde und ob diese als *Aktion* oder *Reaktion* zu verstehen ist. Eine Änderung am Fachmodell bezeichnet man als *Aktion*, falls diese aufgrund geänderter fachlicher Anforderungen durchgeführt wird. Man bezeichnet sie hingegen als *Reaktion*, falls sie aufgrund eines geänderten Systemmodells an dessen neuen Inhalt angepasst wird. Analog dazu versteht man eine Änderung am Systemmodell bei geänderten technischen Anforderungen als *Aktion* und bei einer Anpassung an das Fachmodell als *Reaktion*. Durch die Zuordnung einer Änderung als *Aktion* oder *Reaktion* können die Modellierer erkennen, ob das korrespondierende Modell inklusive dem Abbildungsmodell angepasst werden muss, was nach einer *Aktion* notwendig ist. Bei *Reaktionen* müssen das korrespondierende Modell und das Abbildungsmodell nicht angepasst werden, da von diesen die Änderung ausgeht.

In Abbildung 3.8 wird ein Fachmodell zur Veranschaulichung dieses Problems dargestellt, in dem drei Modellelemente bearbeitet werden ①. Der Grund für die Bearbeitung des Fachmodells könnte zum einen sein, dass sich fachliche Anforderungen geändert haben ② (*Aktion*). Folgerichtig ist das Systemmodell veraltet und dem neuen Fachmodell anzupassen ③. Zum anderen könnte die Bearbeitung darin begründet sein, dass zuvor das Systemmodell überarbeitet wurde. Die Bearbeitung des Fachmodells stellt eine Anpassung an den neuen Zustand des Systemmodells dar ④ (*Reaktion*). Die Modellierer wissen in dieser Situation nicht, ob eine Anpassung des Systemmodells notwendig ist, da sie durch die bloße Betrachtung des bearbeiteten Fachmodells nicht zwischen einer *Aktion* und einer *Reaktion* unterscheiden können.

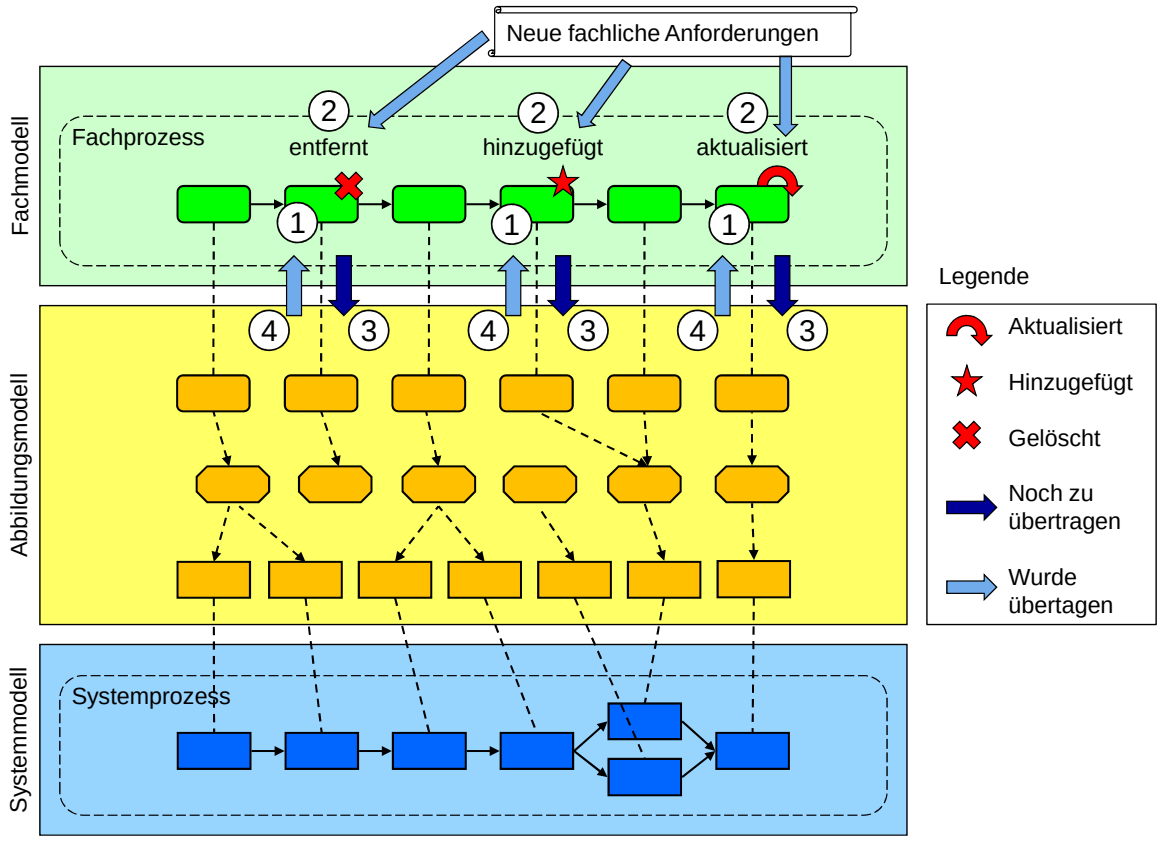

Abbildung 3.8: Änderung im Fachmodell als *Aktion* oder *Reaktion*

Eine Modelländerung kann man als *Aktion* oder *Reaktion* zuordnen, indem man die Abbildungen (Transformationen) zwischen Fachmodell und Systemmodell überprüft, ob sie bereits an die Änderung angepasst wurden. Die Ermittlung gestaltet sich für hinzugefügte Elemente, entfernte Elemente und die in ihren Eigenschaften bearbeiteten Elemente des Prozesses unterschiedlich. Im Folgenden wird behandelt, wie dies bei einem geänderten Fachmodell geschieht.

Bei einem hinzugefügten Prozessknoten des Fachmodells (*bn1* und *bn2* in Abbildung 3.9) untersucht man, ob dieser im Abbildungsmodell in eine Transformation eingebunden ist. Falls eine Transformation existiert (*Transformation 1* in Beispiel ①), so bedeutet dies, dass die zugehörigen Prozessknoten des Systemmodells (*sn1* und *sn2*), wie auch die Transformation bereits zuvor hinzugefügt wurden. Um die Modelle konsistent zu halten, wurde somit der Fachprozessknoten eingefügt (*Reaktion*). Falls keine Transformation zu den hinzugefügten Objekten vorhanden ist ②, so kann das Hinzufügen des Elements keine Anpassung an das Systemmodell beziehungsweise Abbildungsmodell sein und nur aufgrund einer fachlichen Anforderung geschehen sein (*Aktion*). Folglich sind das Abbildungsmodell und das Systemmodell anzupassen.

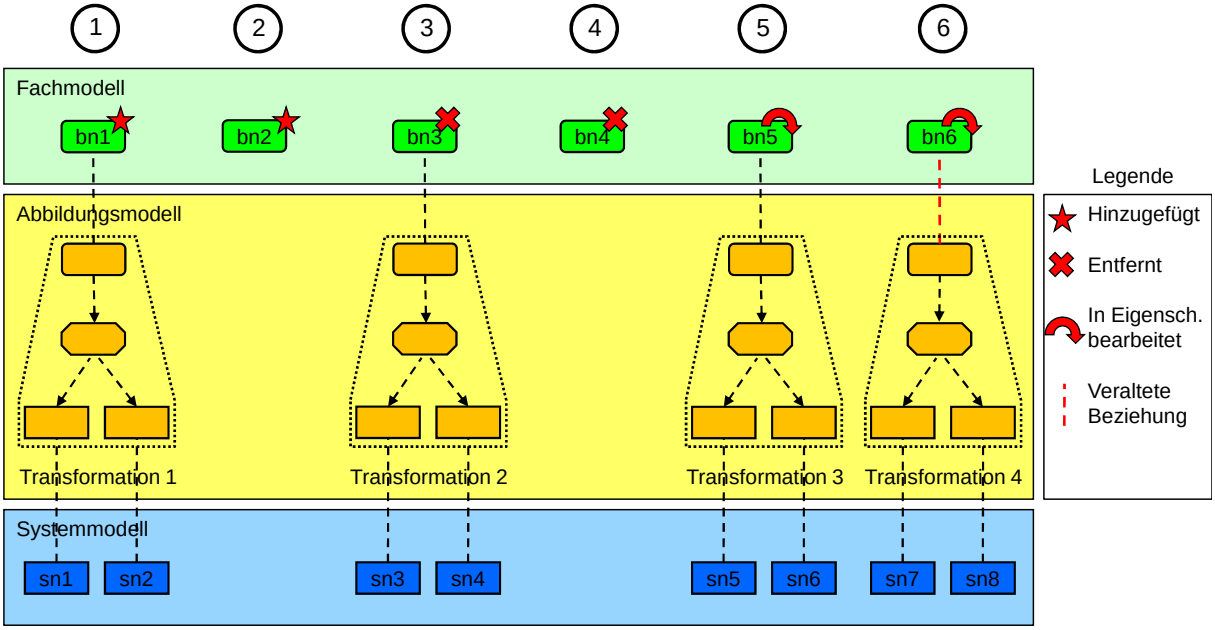

Abbildung 3.9: Bestimmung des Ursprungs einer Änderung am Fachmodell

Bei entfernten Prozessknoten (*bn3* und *bn4* in Abbildung 3.9) kann man die oben beschriebene Analyse leicht abgewandelt vornehmen. Auch hier untersucht man, ob eine entsprechende Abbildung für die Objekte exisitiert. Falls eine solche Abbildung existiert (*Transformation 2* in Beispiel ③), so ist das Entfernen der Objekte noch an das Systemmodell zu übertragen und das Abbildungsmodell mit anzupassen (*Aktion*). In der Abbildung sind dies die Prozessknoten *sn3* und *sn4* sowie die *Transformation 2*. Falls keine Abbildung zu den entfernten Objekten vorhanden ist ④, so ist die Änderung am Fachmodell als Anpassung an das Systemmodell zu werten (*Reaktion*).

Auch für aktualisierte Prozessknoten (*bn5* und *bn6* in Abbildung 3.9) des Fachmodells gilt ein ähnliches Verfahren. Jedoch untersucht man nicht, ob zu dem aktualisierten Objekt eine Transformation existiert, sondern ob sich diese bereits auf das überarbeitete Elemente bezieht. Ist die Transformation bereits an den neuen Zustand des Fachmodells angepasst ⑤, so stellt die Änderung im Fachmodell eine Anpassung an das Systemmodell dar (*Reaktion*). Bezieht sich die Transformation hingegen noch auf den alten Bearbeitungszustand des bearbeiteten Elements ⑥, so kann die Änderung nur fachlich begründet sein (*Aktion*), womit das Abbildungsmodell und das Systemmodell angepasst werden müssen. In der Abbildung sind dies die *Transformation 4* und die Knoten *sn7* und *sn8*.

Die folgenden drei Algorithmen ermitteln durch eine Analyse des Abbildungsmodells, ob mit einer von ihnen zu untersuchenden Veränderung des Fachmodells eine *Aktion* oder *Reaktion* vorliegt. Hierbei verwenden sie zwei Funktionen:

- $\mathrm{QTNodes}(\mathrm{transf})$, mit transf eine Transformation, liefert die Quellknotenmenge *QT* der Transformation transf

- corr_bn(q), mit q ein Quellknoten des Abbildungsmodells, liefert den zu q gehörigen Fachprozessknoten bn

Algorithmus 3.9 zeigt auf, wie zu einem hinzugefügten Knoten (genannt *node*) des Fachprozesses identifiziert wird, ob eine Aktion vorliegt. Als weitere Eingabe dient die Menge der Transformationen des Abbildungsmodells (genannt *Transf*). Das Ergebnis wird als Wahrheitswert angegeben.

Algorithmus 3.9 (Bestätigung des Hinzufügens eines Fachprozessknotens als Aktion)**:**

```
Input:  node            Ein zum Fachprozess hinzugefügter Knoten
Input:  Transf          Die Menge der Transformationen des Abbildungsmodells
                        Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob der Knoten node aufgrund
                        einer fachlichen Anforderung hinzugefügt wurde

for(transf:Transf)      Untersuche zu jeder Transformation des Abbildungsmodells
  QT=QTNodes(transf)    ob aus deren Menge der Quellknoten
  for(q:QT)             ein Quellknoten
    if(corr_bn(q)=node) zu dem Fachprozessknoten node gehört. Falls ja,
      return False      melde, dass eine Reaktion vorliegt
    fi
  end
end
return True             Falls kein solcher Quellknoten gefunden wurde,
                        bestätige eine vorliegende Aktion
```

Algorithmus 3.10 zeigt auf, wie zu einem entfernten Knoten (*node*) des Fachprozesses identifiziert wird, ob die Entfernung als *Aktion* zu deuten ist. Auch hier dient die Menge *Transf* der Transformationen des Abbildungsmodells als Eingabe und das Ergebnis wird als Wahrheitswert angegeben.

Algorithmus 3.10 (Bestätigung der Entfernung eines Fachprozessknotens als Aktion)**:**

```
Input:  node            Ein aus dem Fachprozess entfernter Knoten
Input:  Transf          Die Menge der Transformationen des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob der Knoten node aufgrund
                        einer fachlichen Anforderung entfernt wurde

for(transf:Transf)      Untersuche zu jeder Transformation des Abbildungsmodells
  QT=QTNodes(transf)    ob aus deren Menge der Quellknoten
  for(q:QT)             ein Quellknoten
    if(corr_bn(q)=node) zu dem Fachprozessknoten node gehört. Falls ja,
      return True       bestätige, dass eine Aktion vorliegt
    fi
  end
end
return False            Falls kein solcher Quellknoten gefunden wurde,
                        melde eine vorliegende Reaktion
```

Algorithmus 3.11 zeigt auf, wie zu einem in seinen Eigenschaften bearbeiteten Knoten (*node*) des Fachprozesses identifiziert wird, ob eine Aktion vorliegt. Als weitere Eingabe dient die Menge der Transformationen des Abbildungsmodells (*Transf*). Das Ergebnis wird als Wahrheitswert angegeben.

Der Algorithmus verwendet die weiteren zwei Funktionen:

- *getTimestamp*, angewandt auf einen Knoten des Fachprozesses, liefert dessen Zeitpunkt der letzten Bearbeitung
- *getReferredTimestamp*, angewandt auf einen Quellknoten des Abbildungsmodells, liefert den Zeitpunkt der letzten Bearbeitung des zugehörigen Fachprozessknoten, der zuletzt im Quellknoten vermerkt wurde

Algorithmus 3.11 (Bestätigung des Bearbeitens eines Fachprozessknotens als Aktion)**:**

```
Input:  node            Ein in seinen Attributen bearbeiter Knoten des
                        Fachprozesses
Input:  Transf          Die Menge der Transformationen des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob der Knoten node aufgrund
                        einer fachlichen Anforderung in seinen Attributen
                        bearbeitet wurde

for(transf:Transf)      Untersuche zu jeder Transformation des Abbildungsmodells
  QT=QTNodes(transf)    ob aus deren Menge der Quellknoten
  for(q:QT)             ein Quellknoten
    if(corr_bn(q)=node)         zu dem Fachprozessknoten node gehört. Falls ja,
      t1=q.getReferredTimestamp vergleiche den referenzierten Zeitstempel
      t2=node.getTimestamp      mit dem tatsächlichen Zeitstempel
      if(t1=t2)                 Falls sie gleich sind
        return False            melde eine vorliegende Reaktion
      fi
      if(t1!=t2)                Andernfalls
        return True             melde eine vorliegende Aktion
      fi
    fi
  end
end
```

3.4 Vertikale Auswirkungen

Die Modellierer benötigen nach Änderungen am Fachmodell oder Systemmodell Informationen darüber, an welchen Elementen des jeweils korrespondierenden Modells sie die Änderungen nachvollziehen müssen, beziehungsweise welche Elemente sie auf einen Anpassungsbedarf überprüfen müssen. Die geänderten Objekte des Modells erhalten sie über Vergleiche zwischen zwei Modellversionen (siehe Abschnitt 3.2: *Identifizieren von Änderungen*). Zu diesen ermitteln die Modellierer über die Transformationen des Abbildungsmodells, welche Elemente des korrespondierenden Modells zu kontrollieren sind. Je nachdem ob das Fachmodell oder das Systemmodell bearbeitet wurde, gehen die Modellierer unterschiedlich vor.

Bei bearbeiteten Prozessknoten des Fachmodells (siehe Abbildung 3.10) ist der erste Schritt die Identifikation der zugehörigen Quellknoten im Abbildungsmodell ①. Der zweite Schritt ist die Ermittlung, in welchen Transformationen des Abbildungsmodells diese Quellknoten eingebunden werden ②. Der dritte Schritt ist die Feststellung der in den Transformationen verwendeten Zielknoten ③. In einem vierten und letzten Schritt wird zu jedem dieser Zielknoten ausfindig gemacht, welche Prozessknoten im Systemmodell zugehörig sind ④.

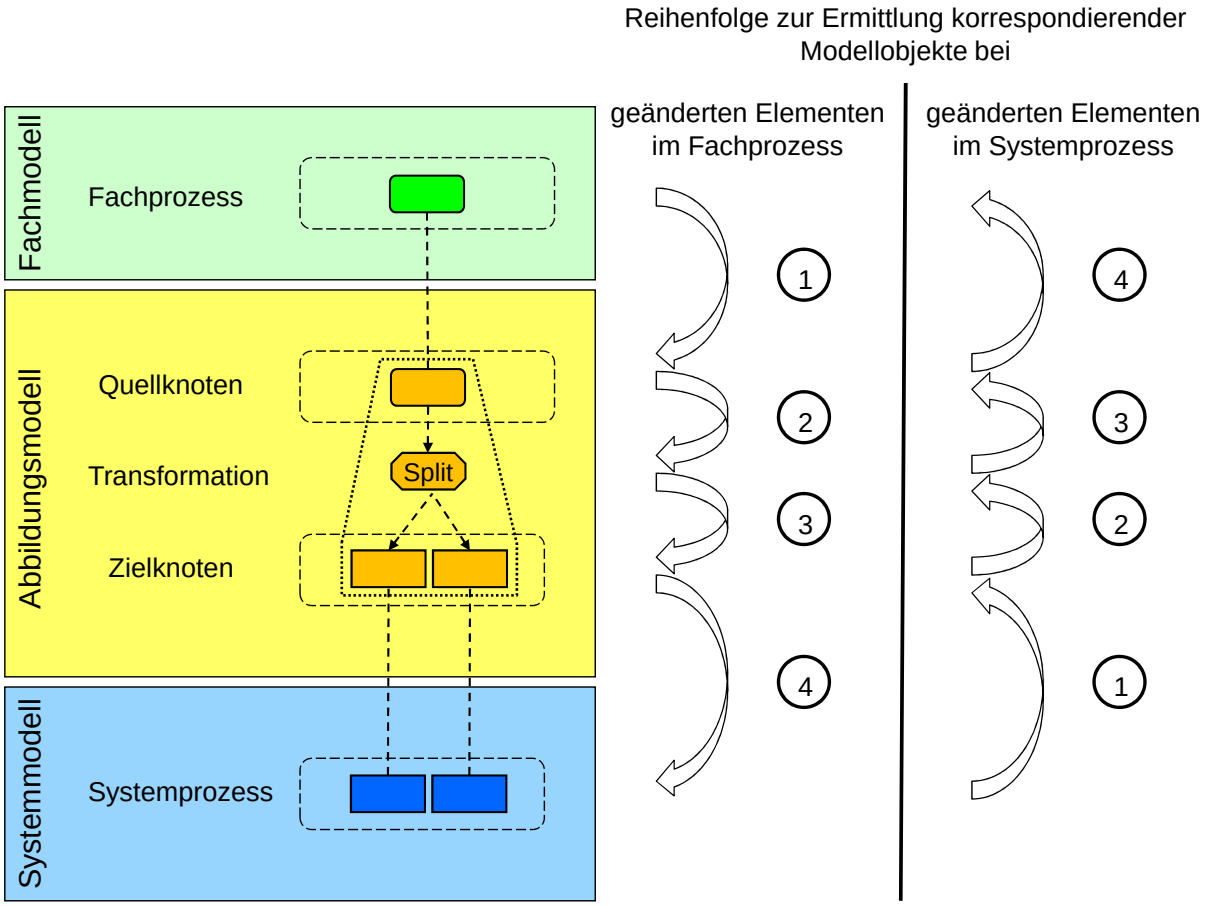

Abbildung 3.10: Die Reihenfolge der Propagierungsschritte

Diesen Vorgang formulieren wir aufgeteilt in zwei Algorithmen unter Benutzung folgender Funktionen:

- $\mathrm{corr\_q(bn)}$, mit bn ein Fachprozessknoten, liefert den zu bn gehörigen Quellknoten q des Abbildungsmodells
- $\mathrm{transf\ (q)}$, mit q ein Quellknoten des Abbildungsmodells, liefert die Transformation transf des Abbildungsmodells, die q als Quellknoten beinhaltet
- $\mathrm{ZTNodes(transf)}$, mit transf eine Transformation, liefert die Zielknotenmenge *ZT* der Transformation transf
- $\mathrm{corr\_sn(z)}$, mit z ein Zielknoten des Abbildungsmodells, liefert den zu z gehörigen Systemprozessknoten sn

Algorithmus 3.12 führt zu einem geänderten Element im Fachprozess, genannt *node*, die oben beschiebenen ersten beiden Schritte ① und ② aus und identifiziert die zugehörige Transformation im Abbildungsmodell, genannt *transf*. Die Transformation wird bei einer Inkonsistenzbehebung mit berücksichtigt und muss deshalb explizit ermittelt werden.

Algorithmus 3.12 (Ermittlung der zum Fachprozessknoten gehörigen Transformation)**:**

```
Input:  node                    Der Prozessknoten des Fachprozesses,
                                zu dem die zugehörige Transformation des
                                Abbildungsmodells gesucht wird
Output: transf                  Die zu node gehörige Transformation

q=corr_q(node)                  Ermittle zum Prozessknoten node den
                                zugehörigen Quellknoten im Abbildungsmodell
transf=transf(q)                und zu diesem die zugehörige Transformation
return transf                   Gebe die Transformation als Ergebnis zurück
```

Zu der durch Algorithmus 3.12 ermittelten Transformation *transf* wendet der Algorithmus 3.13 die letzten beiden Schritte ③ und ④ an und ermittelt diejenigen Aktivitäten im Systemprozess, die über die Zielknoten der Transformation referenziert werden (siehe Abbildung 3.10). Seine Ergebnismenge ist somit die Menge der zum im Fachprozess bearbeiteten Element *node* korrespondierenden Knoten. Im Algorithmus wird die Ergebnismenge als *SNodesAffected* bezeichnet.

Algorithmus 3.13 (Ermittlung der zur Transformation gehörigen Aktivitäten im Systemprozess)**:**

```
Input:  transf                  Die Transformation des Abbildungsmodells,
                                zu der die zugehörigen Prozessschritte
                                im Systemprozess gesucht werden
Output: SNodesAffected          Die Menge der zu transf gehörigen
                                Prozessschritte im Systemprozess

ZT=ZTNodes(transf)              Ermittle die Zielknoten der Transformation
for(z:ZT)                       Ermittle zu jedem dieser Zielknoten
  sn=corr_sn(z)                 den zugehörigen Knoten im Systemprozess
  SNodesAffected.add(sn)        und trage diesen in der Menge der zu transf
                                gehörigen Prozessschritte ein
end
```

Werden die Algorithmen 3.12 und 3.13 nacheinander ausgeführt, so ermitteln sie die zu einem einzigen im Fachprozess bearbeiteten Element *node* korrespondierenden Knoten im Systemprozess. Für eine Menge an Fachprozessknoten, wie sie durch die Algorithmen 3.1, 3.2, oder 3.5 ermittelt werden, wendet man die beiden Algorithmen entsprechend auf jedes einzelne Element an.

Bei bearbeiteten Prozessknoten des Systemmodells gehen die Modellierer zur Identifikation der zugehörigen Elemente im Fachmodell „in entgegengesetzter Richtung“ vor (siehe Abbildung 3.10). Der erste Schritt ist die Ermittlung der zu den Prozessknoten des Systemmodells gehörigen Zielknoten im Abbildungsmodell ①. Der zweite Schritt ist die Ermittlung, in welchen Transformationen des Abbildungsmodells diese Zielknoten eingebunden werden ②. Der dritte Schritt ist die Feststellung der in den Transformationen verwendeten Quellknoten ③. Abschließend

werden in einem vierten Schritt die zu den Quellknoten gehörigen Prozessknoten im Fachmodell ermittelt ④.

Diesen Vorgang formulieren wir aufgeteilt in zwei Algorithmen unter Benutzung folgender Funktionen:

- corr_z(sn), mit sn ein Systemprozessknoten, liefert den zu sn gehörigen Zielknoten z des Abbildungsmodells
- transf (z), mit z ein Zielknoten des Abbildungsmodells, liefert die Transformation transf des Abbildungsmodells, die z als Zielknoten beinhaltet
- QTNodes(transf), mit transf eine Transformation, liefert die Quellknotenmenge *QT* der Transformation transf
- corr_bn(q), mit q ein Quellknoten des Abbildungsmodells, liefert den zu q gehörigen Fachprozessknoten bn

Algorithmus 3.14 führt zu einem geänderten Element im Systemprozess, genannt *node*, die oben beschiebenen ersten beiden Schritte ① und ② aus und identifiziert die zugehörige Transformation im Abbildungsmodell, genannt *transf*.

Algorithmus 3.14 (Ermittlung der zum Systemprozessknoten gehörigen Transformation)**:**

```
Input:  node                  Der Prozessknoten des Systemprozesses,
                              zu dem die zugehörige Transformation des
                              Abbildungsmodells gesucht wird
Output: transf                Die zu node gehörige Transformation

z=corr_z(node)                Ermittle zum Prozessknoten node den
                              zugehörigen Zielknoten im Abbildungsmodell
transf=transf(z)              und zu diesem die zugehörige Transformation
return transf                 Gebe die Transformation als Ergebnis zurück
```

Zu der durch Algorithmus 3.14 ermittelten Transformation *transf* wendet der Algorithmus 3.15 die letzten beiden Schritte ③ und ④ an und ermittelt diejenigen Aktivitäten im Fachprozess, die über die Zielknoten der Transformation referenziert werden (siehe Abbildung 3.10). Seine Ergebnismenge ist somit die Menge der zum im Systemprozess bearbeiteten Element *node* korrespondierenden Knoten. Im Algorithmus wird die Ergebnismenge als *BNodesAffected* bezeichnet.

Algorithmus 3.15 (Ermittlung der zur Transformation gehörigen Aktivitäten im Fachprozess)**:**

```
Input:  transf                  Die Transformation des Abbildungsmodells,
                                zu der die zugehörigen Prozessschritte
                                im Fachprozess gesucht werden
Output: BNodesAffected          Die Menge der zu transf gehörigen
                                Prozessschritte im Fachprozess

QT=QTNodes(transf)              Ermittle die Quellknoten der Transformation
for(q:QT)                       Ermittle zu jedem dieser Quellknoten
  bn=corr_bn(q)                 den zugehörigen Knoten im Fachprozess
  BNodesAffected.add(bn)        und trage diesen in der Menge der zu transf
                                gehörigen Prozessschritte ein
end
```

Zu den bespochenen Algorithmen 3.12, 3.13, 3.14 und 3.15 wird im folgenen Abschnitt gezeigt, dass aus ihren Ergebnismengen ein Prioritätskonflikt erkennbar ist.

3.5 Konkurrierende Prozessänderungen

In diesem Abschnitt wird behandelt, welche Konsequenzen sich dadurch ergeben, dass sowohl der Fachprozess als auch der Systemprozess weiterentwickelt werden und in mehreren Versionen vorliegen (siehe Abschnitt 3.1). Wird ein Prozess bearbeitet, so ist der jeweils korrespondierende Prozess anzupassen. Zum Fachprozess ist dies der Systemprozess, zum Systemprozess ist es der Fachprozess. Welche Elemente des korrespondierenden Prozesses im einzelnen anzupassen sind, wird über die Algorithmen 3.12 und 3.13 beziehungsweise 3.14 und 3.15, ermittelt. Werden jedoch beide Prozesse zeitnah bearbeitet und darauf hin mit diesen Algorithmen die anzupassenden Elemente ermittelt, so dürfen die Modellierer nicht nach diesem einfachen Muster vorgehen und an beiden Prozessesn eine Anpassung vornehmen. Der einfache Grund dafür ist, dass sie ansonst einen Inhalt des einen Prozesses überschreiben könnten, an den der andere Prozess noch anzupassen ist. Somit wäre die vorhergehende Analyse der betroffenen Elemente hinfällig.

Abbildung 3.11 zeigt hierzu ein Beispiel. Das Fachmodell *fm* ① und das Systemmodell *sm* ② sind zum Zeitpunkt t_0 über ein Abbildungsmodell verbunden. Die Transformationen des Abbildungsmodell bilden die Elemente der Modelle auf einander ab und es ist für jedes fachliche Objekt ersichtlich, in welches Objekt des Systemmodells es überführt wird. So wird das Element *a* auf *x* abgebildet, das Element *b* auf *y*. Zum Zeitpunkt t_1 werden neue fachliche und technische Anforderungen bekannt ③, weshalb die Modelle weiter entwickelt werden. Im Fachmodell wird das Element *a* entfernt ④. Im Systemmodell wird das Element *x* inhaltlich überarbeitet, indem es um das Attribut *Text* erweitert wird ⑤. Zum Zeitpunkt t_2 liegt ein neues Fachmodell *fm'* und ein neues Systemmodell *sm'* vor ⑥. Die Bearbeitung des Fachmodells ist eine *Aktion*, das heißt sie wurde durch eine fachliche Anforderung angestoßen (siehe Abschnitt 3.3). Sie bedingt eine Anpassung des Abbildungsmodells und des Systemmodells. Auch die Bearbeitung des

Systemmodells ist eine *Aktion*, da ihrerseits neue technische Anforderungen der Auslöser waren. Als Reaktion ist das Abbildungsmodell und das Fachmodell anzupassen. Für den Modellierer stehen zwei neue Modelle zur Auswahl, zwischen denen er wählen muss ⑦.

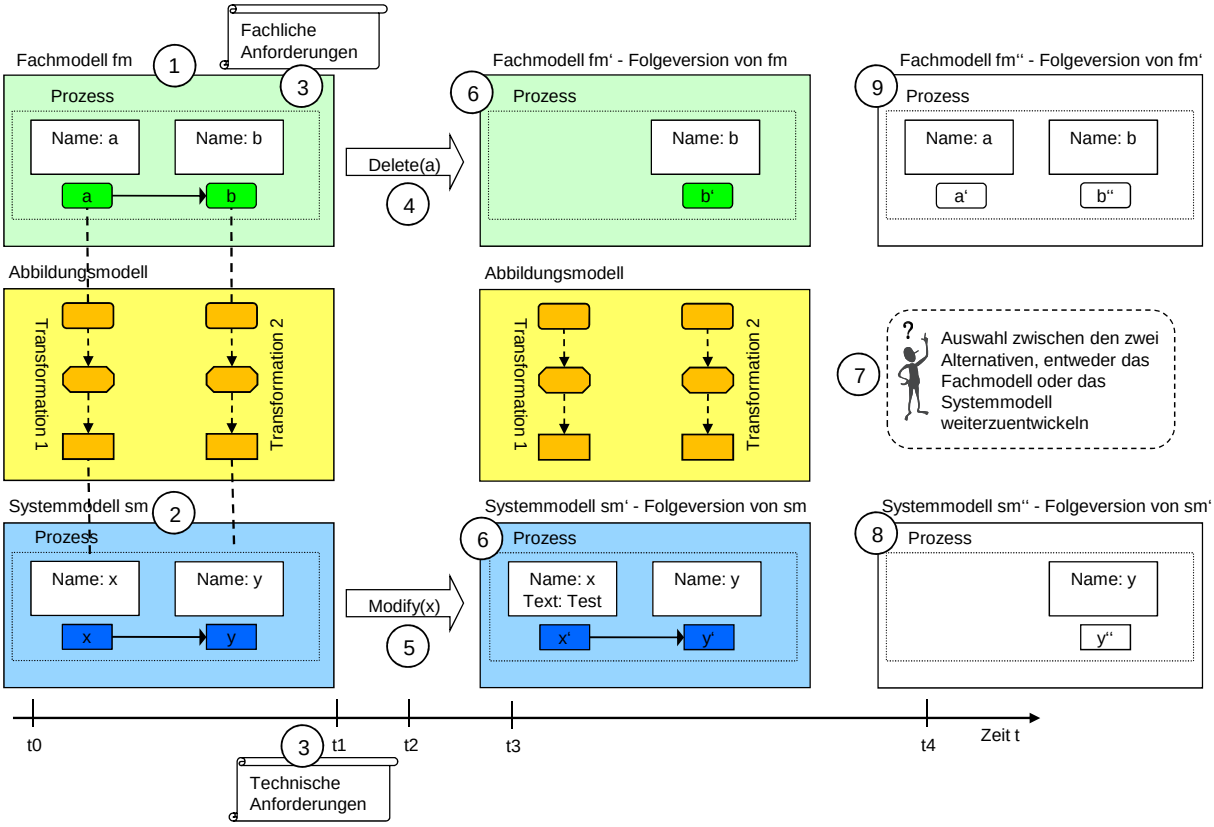

Abbildung 3.11: Entscheidung für Priorität des Fachmodells oder des Systemmodells

Wählt er das Fachmodell *fm'*, so wird das Abbildungsmodell und das Systemmodell an die vorgenommenen Änderungen im Fachmodell angepasst, das heißt die Löschung von *a* bewirkt die Löschung der *Transformation 1* und des Systemprozessknotens *x*. Zum Zeitpunkt t_4 liegen ein Systemmodell *sm''* ⑧ und ein Abbildungsmodell vor, welche dem zum Zeitpunkt t_3 gespeicherten Fachmodell *fm'* in ihrer Struktur entsprechen. Der zum Zeitpunkt t_3 existierende Anpassungsbedarf des Fachmodells *fm'* ist durch das neue Systemmodell *sm''* obsolet geworden. Wählt der Modellierer das Systemmodell *sm'*, so wird die inhaltliche Überarbeitung von *x* auf das Fachmodell übertragen, das heißt es wird ein Objekt *a* eingefügt, welches ebenfalls das Attribut *Text* aufweist. Eine Transformation für die Prozessknoten *a* und *x* wird in das Abbildungsmodell eingetragen. Zum Zeitpunkt t_4 liegen somit ein Fachmodell *fm''* ⑨ und ein Abbildungsmodell vor, welche inhaltlich dem Systemmodell *sm'* entsprechen. Der Anpassungsbedarf des Systemmodells zum Zeitpunkt t_3 ist obsolet geworden. Die Schlussfolgerung dieser zwei Entscheidungsmöglichkeiten und ihrer Auswirkungen ist, dass Fachmodell und Systemmodell je nach Wahl des Modellierers unterschiedliche Strukturen erhalten.

Neu hinzugefügte Knoten besitzen noch keine Transformation mit Objekten im korrespon-

dierenden Modell, wodurch kein Konflikt zwischen Fach- und Systemmodell auftritt. Konflikte treten nur dann auf, wenn Knoten im Fachprozess und Systemprozess gelöscht oder überarbeitet werden, deren Quellknoten und Zielknoten in der selben Transformation des Abbildungsmodells vorkommen. Dies gilt in unserem Beispiel für die *Transformation 1*, die die Elemente *a* und *x* aufeinander abbildet. Das Element *x* wird überarbeitet und das Element *a* gelöscht. Folglich müssen die Modellierer entscheiden, welche dieser Änderungen am anderen Modell nachzuvollziehen ist.

Der Algorithmus zur Identifizierung dieser Konfliktsituation ist wie folgt:

Algorithmus 3.16 (Ermittlung konkurrierender Prozessänderungen)**:**

```
Input:  BnRemovedOrModified     Die Menge der entfernten oder in Attributen
                                bearbeiteten Knoten des Fachprozesses
Input:  SnRemovedOrModified     Die Menge der entfernten oder in Attributen
                                bearbeiteten Knoten des Systemprozesses
Output: Couples                 Tupel von konkurrierenden Knoten beider Prozesse

for(bn:BnRemovedOrModified)     Ermittle zu jedem Knoten aus BnRemovedOrModified
 t1=getTransformation(bn)       die zugehörige Transformation
 for(sn:SnRemovedOrModified)    Ermittle zu jedem Knoten aus SnRemovedOrModified
  t2=getTransformation(sn)      die zugehörige Transformation
  if(t1=t2)                     Falls diese Transformationen identisch sind
   Couples.add(bn,sn)           vermerke die beiden Knoten als konkurrierend
  fi
 end
end
```

Um die Modellierer bei der Priorisierung zu unterstützen, kann man die Transformationen mit weiteren Informationen anreichern. Denkbar wäre ein Flag, dass die Richtung der letzten Priorisierung in der Transformation speichert. Eine Visualisierung ist hier durch eine farbliche Darstellung oder ein Symbol an der Transformation leicht möglich. Ein Beispiel zeigt hierfür Abbildung 3.12: Ohne gespeicherte Prioritätsrichtung verbleibt die Anzeige der Transformationen wie gehabt ①. Nach einer Prioritätsenscheidung wird die Richtung mit angezeigt ②. Transformationen, an denen keine Prioritätsentscheidung vorgenommen werden kann, sind hiervon nicht betroffen ③.

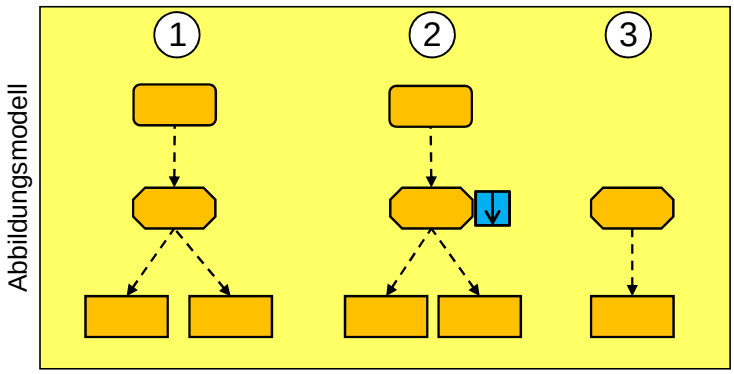

Abbildung 3.12: Anzeige der Priorität an den Transformationen

3.6 Horizontale Auswirkungen

Das Fachmodell und das Systemmodell müssen nach Änderungen an einander angepasst werden, um sie weiterhin zueinander konsistent zu halten. Bevor die Modellierer jedoch eine Anpassung zwischen beiden Modellen vornehmen können, müssen sie die bearbeiteten Fach- und Systemmodelle auf Inkonsistenzen untersuchen und gegebenenfalls korrigieren. Unter Inkonsistenz eines Modells versteht man im Folgenden, dass mindestens eines seiner Elemente in seinem Inhalt veraltet ist oder der korrekte Ablauf des Modellprozesses nicht gewährleistet ist.

Folendes Beispielmodell aus Abbildung 3.13 veranschaulicht, dass durch Änderungen an einem Modell Inkonsistenzen auftreten können: Das Fachmodell *fm* beinhaltet einen Prozess mit den drei sequentiell auszuführenden Knoten *a*, *b* und *c*. Als Daten besitzt es die Objekte *x* und *y*. Ein Pfeil symbolisiert, dass der Knoten *a* schreibend auf Datenobjekt *x* zugreift ①. Der Knoten *c* greift lesend auf das Datenobjekt *y* zu, was ebenfalls durch einen Pfeil dargestellt wird ②. Dieses Fachmodell *fm* wird durch den Modellierer weiterentwickelt. Er nimmt Änderungen an den Eigenschaften des Prozessschritts *a* vor und löscht das Datenobjekt *y* ③. Den neuen Zustand des Modells speichert er in der Version *fm'*. Die Prozessknoten tragen fortan die Bezeichnungen *a'*, *b'* und *c'*, da sie in einem neuen Bearbeitungszustand gespeichert werden. Das Datenobjekt *x* liegt im Zustand *x'* vor. Wie in der Abbildung zu sehen ist, enthält der Prozessschritt *c'* in der Fachmodellversion *fm'* weiterhin einen Eintrag, dass er vom Datenobjekt *y* liest. Dieses ist jedoch nicht mehr im Modell enthalten ④. Die Modellversion *fm'* weist demnach einen klar ersichtlichen Fehler auf, solange ihr Element *c'* nicht vom Modellierer korrigiert wird. Ein weniger offensichtlicher Fehler könnte am Datenobjekt *x'* vorliegen: Da der darauf schreibende Prozessschritt *a* in seinen Attributen bearbeitet wurde, könnte *x'* an dessen neuen Zustand *a'* anzupassen sein.

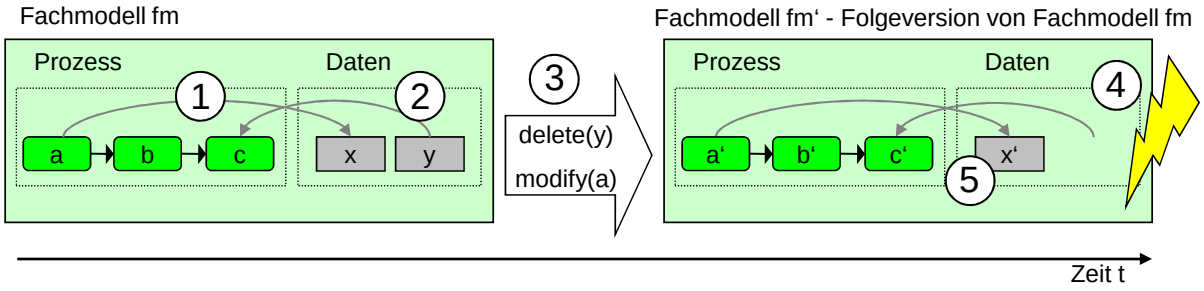

Abbildung 3.13: Inkonsistenz nach Änderung am Fachmodell 1

Wir sehen an diesem Beispiel, dass jede im Modell durchgeführte Änderung eines Elements auch Auswirkungen auf andere Modellelemente haben kann. Diese Modellelemente bezeichnen wir als von der Änderung betroffen. In obiger Abbildung betrifft das Entfernen des Datenobjekts *y* das Prozesselement *c'*, die Änderung der Prozessaktivität *a* das Datenobjekt *x'*. Die betroffenen Modellelemente sind vom Modellierer mit den geänderten Modellelementen abzugleichen und bei Bedarf an dessen neuen Zustand anzupassen. Je nachdem, ob die geänderten Modellelemente Prozessschritte oder Datenobjekte sind, sind jeweils Elemente der anderen Menge betroffen. Zu geänderten Prozessaktivitäten erhält man die Menge der direkt betroffenen Datenobjekte über folgenden Algorithmus:

Algorithmus 3.17 (Ermittlung betroffener Datenobjekte)**:**

```
Input: Modified          Die Menge der im Modell geänderten Prozessaktivitäten
Input: DO                Die Menge der Datenobjekte des Modells in der
                         Folgeversion
Output: Affected         Die Menge der Datenobjekte des Modells,
                         die auf Konsistenz mit den geänderten Prozessaktivitäten
                         zu prüfen ist

for(m:Modified)          Untersuche zu jedem geänderten Prozessschritt m
 for(do:DO)              und zu jedem Datenobjekt do
  if(m.reads(do) or      ob do von m gelesen oder
     m.writes(do))       geschrieben wird. Falls ja,
   Affected.add(do)      vermerke das Datenobjekt do als betroffen
  fi
 end
end
```

Die Menge *Modified* dienst als Eingabe. Sie ist die Vereinigung der Mengen *Added*, *Deleted* und *Modified*, welche die hinzugefügten, entfernten, und in ihren Eigenschaften bearbeiteten Objekte beinhalten. Man erhält sie durch Anwendung der Algorithmen aus Abschnitt 3.2. Als weitere Eingabe dient die Menge der Datenobjekte *DO* der Folgeversion des Modells. Der Algorithmus untersucht, welche Datenobjekte von den in *Modified* enthaltenen Prozessschritten gelesen oder geschrieben werden und gibt diese in der Menge *Affected* aus. Betreffend des Beispiels aus

Abbildung 3.13, in dem die Prozessaktivität *a* in ihren Eigenschaften bearbeitet wird, erhält die Menge *Modified* entsprechend das Element *a'*. Die Menge *DO* besteht aus dem Datenobjekt *x'* der Modellversion *fm'*. Die Ergebnismenge *Affected* liefert *x'*, da dieses von *a'* geschrieben wird.

Zu geänderten Datenobjekten erhält man die Menge von direkt betroffenen Prozessaktivitäten über folgenden Algorithmus:

Algorithmus 3.18 (Ermittlung betroffener Prozessaktivitäten)**:**

```
Input: DO               Die Menge der Datenobjekte, zu denen direkt betroffene
                        Prozessschritte ermittelt werden sollen
Input: N                Die Menge der Prozessaktivitäten des Modells
Output: Affected        Die Menge der Prozessaktivitäten des Modells,
                        die auf Konsistenz mit den Datenobjekten DO
                        zu prüfen ist

for(do:DO)              Untersuche zu jedem Datenobjekt do aus DO
 for(n:N)               und zu jedem Prozessschritt n aus N
  if(n.reads(do) or     ob do von n gelesen oder
     n.writes(do))      geschrieben wird. Falls ja,
   Affected.add(n)      vermerke den Prozessschritt n als betroffen
  fi
 end
end
```

Wendet man diesen Algorithmus auf das Beispiel aus Abbildung 3.13 an, so wird zum gelöschten Datenobjekt *y* untersucht, ob die Prozessaktivitäten *a'*, *b'* und *c'* der Modellversion *fm'* von diesem Datenobjekt lesen oder darauf schreiben. Wie bereits in der Abbildung ersichtlich, trifft dies auf die Aktivität *c'* zu.

Abbildung 3.14 illustriert, dass auch Modellelemente von einer Änderung betroffen sein können, die nicht unmittelbar mit den bearbeiteten Elementen durch einen Lese- oder Schreibzugriff zusammenhängen. Das Fachmodell *fm* beinhaltet die drei Prozessknoten *a*, *b* und *c*. Als Daten liegen die Objekte *x* und *y* vor. Wir sehen durch Pfeile ausgedrückt, dass der Knoten *a* schreibend auf das Datenobjekt *x* zugreift ①, der Knoten *b* hingegen liest von *x* ②. Der Knoten *c* greift lesend auf Datenobjekt *y* zu, ebenfalls durch einen Pfeil dargestellt. Der Modellierer nimmt nun Änderungen an *fm* vor ③ und speichert es in der neuen Version *fm'*. Insbesondere bearbeitet er den Prozessschritt *a*, indem er beispielsweise neue Daten in das Datenobjekt *x* schreiben lässt. Die Prozessknoten tragen nun die Bezeichnungen *a'*, *b'* und *c'*, da sie in einem neuen Bearbeitungszustand gespeichert wurden. Die Datenobjekte liegen im Zustand *x'* und *y'* vor. Wie wir in der neuen Modellversion *fm'* sehen können, hat die Änderung an *a* unmittelbare Auswirkungen auf das Datenobjekt *x'* ④. Der Modellierer muss kontrollieren, ob das Datenobjekt *x'* die durch *a'* geschriebenen neuen Datensätze bereits speichern kann, oder ob es noch anzupassen ist. Die Änderung an *a* hat außerdem noch indirekte Auswirkungen auf den Prozessknoten *b'* ⑤, da dieser die Daten aus dem Datenobjekt *x'* liest, die zuvor durch *a'* dorthin geschrieben werden. Dies bedeutet für den Modellierer, dass er nicht nur das Datenobjekt

x' an die Änderung an *a* anpassen muss, sondern zudem den Prozessschritt *b'*.

Indirekt von einer Änderung betroffene Modellelemente kann es nur zu Prozessknoten geben. Diese ermittelt man, indem man zuerst die zu den geänderten Prozessknoten direkt betroffenen Datenobjekte aufspürt, und zu diesen wiederum die direkt betroffenen Prozessschritte. Die obigen zwei Algorithmen werden für diesen Zweck direkt hintereinander ausgeführt. Die Ergebnismenge *Affected* des ersten Algorithmus dient als Eingabe *DO* des zweiten Algorithmus.

Zu den geänderten Modellelementen beziehungsweise den von den Änderungen betroffenen Elementen sind jeweils die zugehörigen Elemente des korrespondierenden Modells anzupassen (siehe Abschnitt 3.4).

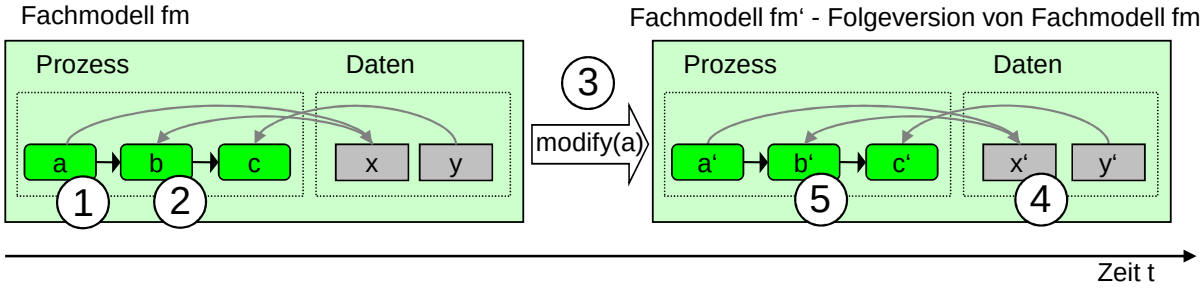

Abbildung 3.14: Inkonsistenz nach Änderung am Fachmodell 2

Abbildung 3.15 zeigt, dass auch bei dieser Anpassung eine horizontale Analyse auf Abhängigkeiten gemacht werden muss. Abgebildet sind ein Fachmodell, ein Abbildungsmodell und ein Systemmodell. Das Fachmodell beinhaltet die zwei Prozessknoten *a* und *b*. Als Daten besitzt das Fachmodell die Objekte *x* und *y*. Wir sehen durch Pfeile ausgedrückt, dass der Knoten *a* schreibend auf das Datenobjekt *x* zugreift, sowie Knoten *b* schreibend auf *y*. Das Systemmodell besteht aus vier Prozessschritten sowie zwei Datenobjekten, zwischen denen ebenfalls Lese- und Schreiboperationen stattfinden. Das Abbildungsmodell besteht aus drei Transformationen, die die Prozessknoten des Fachmodells auf die Prozessknoten des Systemmodells abbilden. Die Datenobjekte des Fachmodells und die Datenobjekte des Systemmodells sind über Mappingregeln verbunden.

Nimmt der Modellierer Änderungen am Prozessschritt *b* ① und Datenobjekt *y* ② vor, so ist nach den besprochenen Algorithmen der Prozessschritt *b* von *y* betroffen, *y* hingegen von *b*. Die weiteren Elemente des Fachmodells, die Aktivität *a* und das Datenobjekt *x*, sind nicht an die beiden anzupassen, da sie nicht zu den direkt Betroffenen zählen. Der Prozessschritt *b* und das Datenobjekt *y* bilden so gewissermaßen eine geschlossene Einheit.

Möchte der Modellierer nun das Systemmodell an diese beiden Änderungen im Fachmodell anpassen, so wendet er das in Abschnitt 3.4 besprochene Verfahren zur Ermittlung der im Systemmodell zugehörigen Elemente an. Zu Prozessknoten *b* wird über die *Transformation 2* ③ identifiziert, dass die Prozessaktivität *s* ④ zugehörig ist. Die *Mappingregel 2* ⑤ zeigt, dass das Datenobjekt *v* ⑥ zum Datenobjekt *y* zugehörig ist. Im Systemmodell sind demnach die Elemente *s* und *v* vom Modellierer an die beiden Änderungen im Fachmodell anzupassen. Kontrolliert

man zu diesen beiden Modellelementen die betroffenen Elemente, so stellt man fest, dass neben den Elementen *s* und *v* noch das Element *t* betroffen ist, welches über die *Transformation 3* in das Systemmodell eingefügt wird. Da die Anpassung derjenigen Elemente im Systemmodell offensichtlich nicht ausreicht, die zu der Menge an geänderten und betroffenen Elementen im Fachmodell zugehörig sind, muss der Modellierer im Systemmodell ebenfalls eine Kontrolle auf betroffene Modellelemente durchführen.

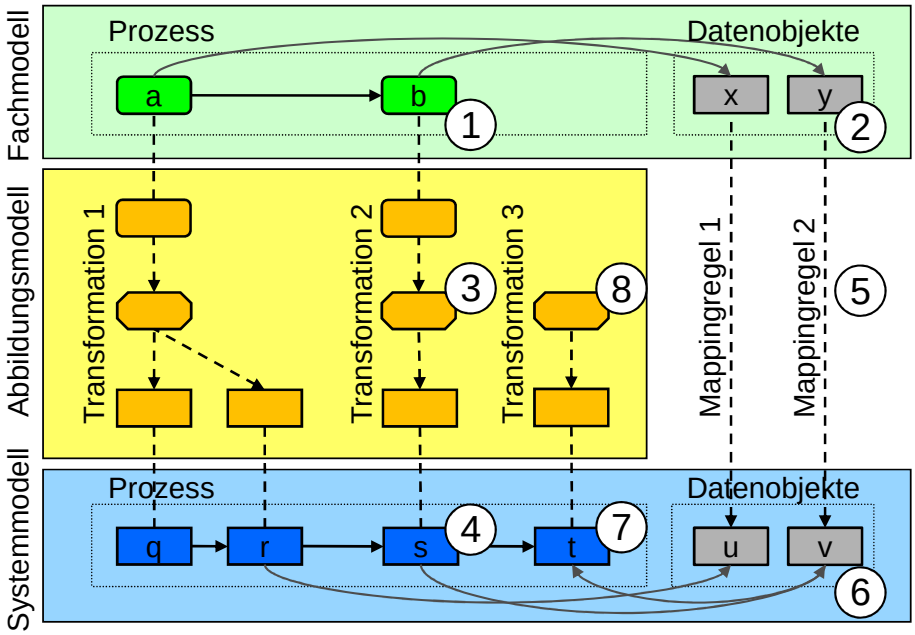

Abbildung 3.15: Indirekte Ermittlung anzupassender Elemente

3.7 Behebung der Inkonsistenzen

Wie bereits in Abschnitt 4.1 erläutert, liegen Fachprozesse und Systemprozesse in mehreren Versionen vor, falls sie geändert werden. In den Prozessen können weitere Aktivitäten hinzugefügt werden, es können Aktivitäten gelöscht werden, oder es können Aktivitäten in ihren Eigenschaften bearbeitet werden (vgl. Abschnitt 4.2). Zu den geänderten Aktivitäten müssen die Modellierer eine Anpassung des Abbildungsmodells und des korrespondierenden Prozesses vornehmen, um die Nachvollziehbarkeit zwischen Fachprozessen und Systemprozessen zu gewährleisten. Die konkrete Vorgehensweise und Entscheidungsmöglichkeiten der Modellierer werden im Folgenden anhand eines geänderten Fachprozesses gezeigt:

Abbildung 3.16 zeigt einen Fachprozess, ein Abbildungsmodell und einen Systemprozess. Einige Aktivitäten des Fachprozesses wurden bearbeitet. Über die Algorithmen aus Abschnitt 3.2 wird ermittelt, dass die Aktivitäten ① und ④ entfernt wurden. Die Aktivität ⑥ wurde hinzugefügt und die Aktivität ⑧ wurde in ihren Eigenschaften bearbeitet.

Zu diesen geänderten Aktivitäten werden über den Algorithmus 3.12 die betroffenen Transformationen berechnet. In einem weiteren Schritt werden zu diesen Transformationen die zugehörigen

Aktivitäten des Systemprozesses ermittelt (vgl. Algorithmus 3.13). Die identifizierten Transformationen und Aktivitäten des Systemprozesses müssen von den Modellierern untersucht werden.

Zu der gelöschten Aktivität ① wird dem Modellierer gemeldet, dass er die Transformation ② untersuchen und anpassen soll. Der Modellierer muss entscheiden, ob die Löschung der Aktivität an den Systemprozess zu übertragen ist. Falls die zugehörigen Aktivitäten des Systemprozesses weiterhin verwendet werden sollen, so muss er lediglich die Transformation anpassen, indem er sie in den Transformationstyp *Insert* umwandelt (vgl. Abschnitt 2.4). Befindet der Modellierer hingegen, dass der Systemprozess anzupassen ist, so werden sowohl die Transformation ② als auch die zugehörigen Aktivitäten im Systemprozess gelöscht ③.

Bei der gelöschten Aktivität ④ stellt sich diese Frage der Anpassung des Systemprozesses nicht, da die zugehörige Transformation des Abbildungsmodells von Typ *Delete* ist ⑤. In diesem Fall kann die Transformation unmittelbar gelöscht werden.

Zu hinzugefügten Aktivitäten wie ⑥ existieren noch keine Transformationen im Abbildungsmodell ⑦. Hier kann der Modellierer entscheiden, ob eine zukünftige technische Umsetzung erfolgen soll. Fügt er eine Transformation des Typs *Delete* ein, so entfällt die technische Umsetzung. Mit Transformationen des Typs *Map* oder *Split* werden hingegen technische Aktivitäten vorgesehen.

Zu der in ihren Eigenschaften bearbeiteten Aktivität ⑧ muss der Modellierer mindestens den Quellknoten der Transformation ⑨ aktualisieren. Danach steht es ihm frei, darüber zu entscheiden, ob die zugehörigen Aktivitäten im Systemprozess ebenfalls aktualisiert und damit angepasst werden sollen.

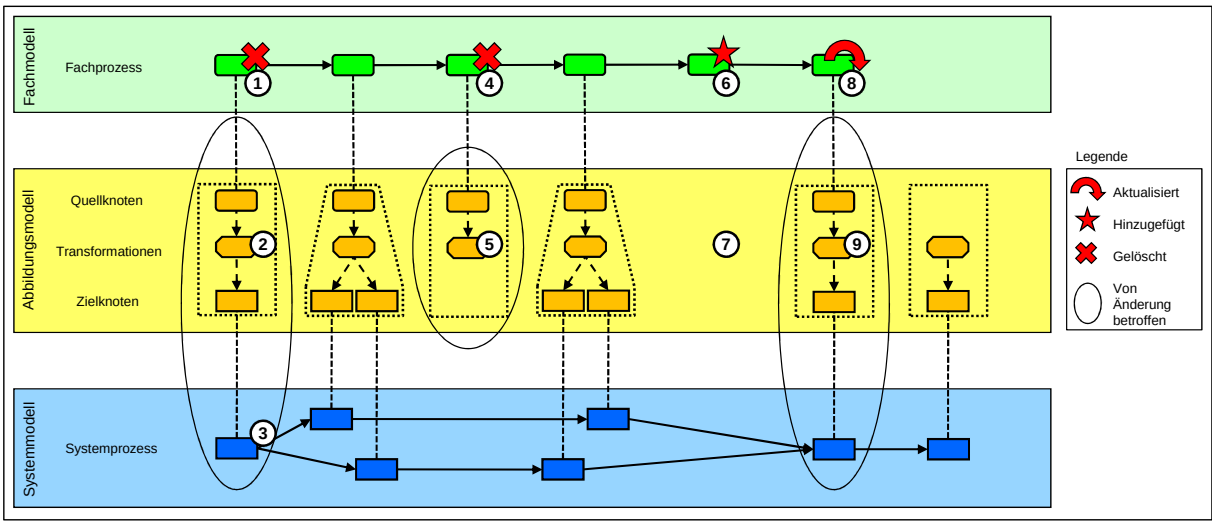

Abbildung 3.16: Zu überprüfende Aktivitäten bei bearbeitetem Fachprozess

Die vorgestellten Handlungsmöglichkeiten zu den Transformationen des Abbildungsmodells stellen nur einen kleinen Ausschnitt aus einer Vielzahl von Variationen dar. Einen tieferen Einblick in Umstrukturierungsmöglichkeiten des Abbildungsmodells bietet [Rec10].

3.8 Konsistenzüberprüfung der Modellebenen

Nachdem die Modellierer den Fachprozess und den Systemprozess untereinander abgeglichen haben, müssen sie einen abschließenden Konsistenzcheck durchführen. Getestet wird auf zwei Arten von Konsistenz: Die *strukturelle Konsistenz* zwischen dem Fachprozess und Abbildungsmodell beziehungsweise Abbildungsmodell und Systemprozess, wie auch die *innere Konsistenz* des Abbildungsmodells.

3.8.1 Test auf strukturelle Konsistenz

Um zu klären, was unter einer strukturellen Konsistenz zu verstehen ist, sei zunächst hier einer Liste an möglichen Inkonsistenzursachen aufgezählt (angelehnt an [BBR11]).

Gegeben seien ein Fachmodell, ein Abbildungsmodell und ein Systemmodell. Es besteht dann zwischen ihnen eine *strukturelle Inkonsistenz*, falls mindestens eine der folgenden Bedingungen erfüllt ist:

- *B1*: Zu einem Fachprozessknoten des Fachmodells gibt es keinen zugehörigen Quellknoten des Abbildungsmodells (vgl. ① in Abb. 3.17)
- *B2*: Zu einem Quellknoten des Abbildungsmodells gibt es keinen zugehörigen Fachprozessknoten des Fachmodells (vgl. ② in Abb. 3.17)
- *B3*: Zu einem Zielknoten des Abbildungsmodells gibt es keinen zugehörigen Systemprozessknoten des Systemmodells (vgl. ③ in Abb. 3.17)
- *B4*: Zu einem Systemprozessknoten des Systemmodells gibt es keinen zugehörigen Zielknoten des Abbildungsmodells (vgl. ④ in Abb. 3.17)

Listing 3.1: Bedingungen struktureller Inkonsistenz

Mit fehlendem Quellknoten beziehungsweise Zielknoten ist jeweils gemeint, dass überhaupt eine Transformation im Abbildungsmodell fehlt, die einen entsprechenden Knoten beinhalten würde.

Das Fachmodell, das Systemmodell und das Abbildungsmodell sind gemäß obiger Definition zu einander *strukturell konsistent*, falls keine der Bedingungen *B1* bis *B4* (vgl. Listing 3.1) erfüllt ist. Mit anderen Worten muss zu jedem Fachprozessknoten über eine Transformation ein Systemprozessknoten zugeordnet sein.

Die folgende Abbildung 3.17 zeigt die Verletzung der Konsistenz durch Erfüllung der Bedingungen *B1* bis *B4* beispielhaft an einem Fachprozess, Systemprozess und Abbildungsmodell auf. Der Fachprozess besitzt die drei Prozessknoten *a*, *b* und *c*, das Abbildungsmodell enthält drei Transformationen und der Systemprozess beinhaltet die drei Prozessschritte *r*, *s* und *t*. Die Prozesse sind zu einander strukturell inkonsistent, da einige der Bedingungen *B1* bis *B4* erfüllt sind (rot gekennzeichnet). Die Bedingung *B1* ist erfüllt, da der Prozessknoten *a* des

Fachprozesses über keinerlei Transformation des Abbildungsmodells auf eine Aktivität im Systemprozess abgebildet wird. Ohne Transformation fehlt auch der zu *a* gehörige Quellknoten ①. Die Bedingung *B2* ist erfüllt, da die *Transformation 1* des Abbildungsmodells einen Quellknoten beinhaltet, der zu einem nicht mehr im Fachprozess vorhandenen Fachprozessknoten zugehörig ist ②. Die Bedingung *B3* ist dadurch erfüllt, dass die *Transformation 2* des Abbildungsmodells einen Zielknoten enthält, der auf einen nicht mehr im Systemprozess vorhandenen Systemprozessknoten verweist ③. Die Bedingung *B4* ist erfüllt, da der Prozessknoten *s* des Systemmodells über keinerlei Transformation des Abbildungsmodells auf eine Aktivität im Fachprozess abgebildet wird. Ohne Transformation fehlt auch der zu *s* gehörige Zielknoten ④. Lediglich die *Transformation 3* mit dem zugehörigen Prozessknoten *c* des Fachprozesses, sowie dem zugehörigen Prozessknoten *t* des Systemprozesses, erfüllen keine der Bedingungen *B1* bis *B4* ⑤. Da jedoch wie aufgezählt bereits mehrere beziehungsweise alle der Bedingungen *B1* bis *B4* erfüllt sind, sind die Modelle *strukturell inkonsistent*.

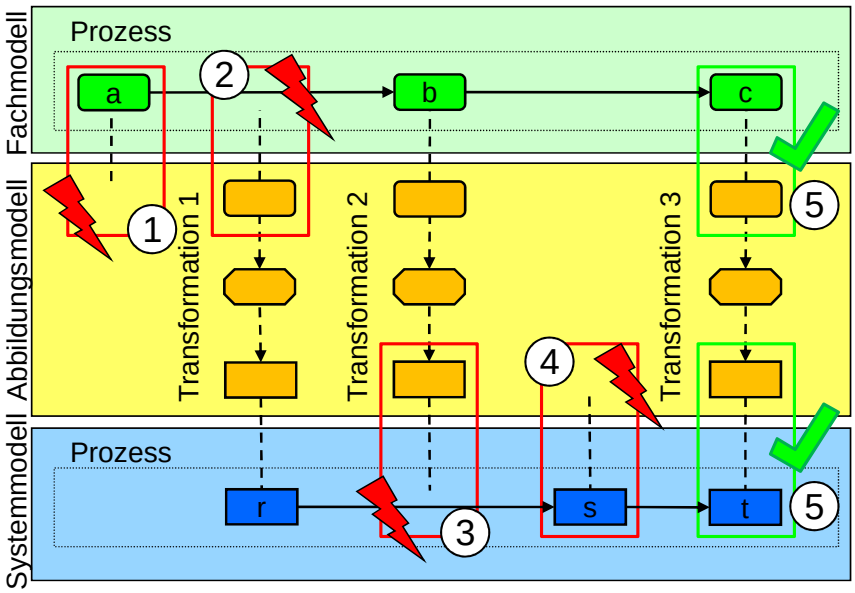

Abbildung 3.17: Strukturelle Inkonsistenz der Modelle

Unter Verwendung folgender zwei Funktionen überprüfen die nächsten vier Algorithmen, ob die Bedingungen *B1* bis *B4* zu einem Fachmodell, einem Systemmodell und einem Abbildungsmodell erfüllt sind.

- corr_bn(q), mit q ein Quellknoten des Abbildungsmodells, liefert den zu q gehörigen Fachprozessknoten bn
- corr_sn(z), mit z ein Zielknoten des Abbildungsmodells, liefert den zu z gehörigen Systemprozessknoten sn

Die Erfüllung der Bedingung *B1* wird über den Algorithmus 3.19 geprüft, der die Prozessknoten des Fachmodells und die Quellknoten des Abbildungsmodells analysiert.

Algorithmus 3.19 (Überprüfung der Inkonsistenz nach Regel *B1*)**:**

```
Input:  BN              Die Menge der Prozessknoten des Fachmodells
Input:  Q               Die Menge der Quellknoten des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Inkonsistenz nach B1
                        gegeben ist

for(bn:BN)              Untersuche zu jedem Prozessknoten bn aus BN
 for(q:Q)               und zu jedem Quellknoten q aus Q
  if(corr_bn(q)=bn)     ob bn der zu q gehörige Prozessknoten ist. Falls dies
                        bestätigt wird,
   return False         gebe als Ergebnis zurück, dass die Regel B1
                        nicht erfüllt wird
  fi
 end
 return True            Falls kein zu bn gehöriger Quellknoten gefunden wurde,
                        gebe als Ergebnis die Erfüllung der Regel B1 bekannt
end
```

Der Algorithmus 3.20 beschreibt, wie die Inkonsistenz nach der Bedingung *B2* getestet wird.

Algorithmus 3.20 (Überprüfung der Inkonsistenz nach Regel *B2*)**:**

```
Input:  BN              Die Menge der Prozessknoten des Fachmodells
Input:  Q               Die Menge der Quellknoten des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Inkonsistenz nach B2
                        gegeben ist

for(q:Q)                Untersuche zu jedem Quellknoten q aus Q
 for(bn:BN)             und zu jedem Prozessknoten bn aus BN
  if(corr_bn(q)=bn)     ob bn der zu q gehörige Prozessknoten ist. Falls dem
                        so ist,
   return False         gebe als Ergebnis zurück, dass die Regel B2 nicht
                        erfüllt ist
  fi
 end
 return True            Falls kein zu q gehöriger Prozessknoten gefunden wurde,
                        gebe als Ergebnis die Erfüllung der Regel B2 bekannt
end
```

Die Erfüllung der Bedingung *B3* wird über den Algorithmus 3.21 geprüft, der die Prozessknoten des Systemmodells und die Zielknoten des Abbildungsmodells analysiert.

Algorithmus 3.21 (Überprüfung der Inkonsistenz nach Regel *B3*)**:**

```
Input:  SN              Die Menge der Prozessknoten des Systemmodells
Input:  Z               Die Menge der Zielknoten des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Inkonsistenz nach B3
                        gegeben ist

for(z:Z)                Untersuche zu jedem Zielknoten z aus Z
 for(sn:SN)             und zu jedem Prozessknoten sn aus SN
  if(corr_sn(z)=sn)     ob sn der zu z gehörige Prozessknoten ist. Falls dem
                        so ist,
   return False         gebe als Ergebnis zurück, dass die Regel B3 nicht
                        erfüllt ist
  fi
 end
 return True            Falls kein zu z gehöriger Prozessknoten gefunden wurde,
                        gebe als Ergebnis die Erfüllung der Regel B3 bekannt
end
```

Der Algorithmus 3.22 beschreibt, wie die Inkonsistenz nach der Bedingung *B4* getestet wird.

Algorithmus 3.22 (Überprüfung der Inkonsistenz nach Regel *B4*)**:**

```
Input:  SN              Die Menge der Prozessknoten des Systemmodells
Input:  Z               Die Menge der Zielknoten des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Inkonsistenz nach B4
                        gegeben ist

for(sn:SN)              Untersuche zu jedem Prozessknoten sn aus SN
 for(z:Z)               und zu jedem Quellknoten z aus Z
  if(corr_sn(z)=sn)     ob sn der zu z gehörige Prozessknoten ist. Falls dem
                        so ist,
   return False         gebe als Ergebnis zurück, dass die Regel B4 nicht
                        erfüllt ist
  fi
 end
 return True            Falls kein zu sn gehöriger Zielknoten gefunden wurde,
                        gebe als Ergebnis die Erfüllung der Regel B4 bekannt
end
```

3.8.2 Überprüfung der Konsistenz des Abbildungsmodells

Ein Test auf *innere Konsistenz* des Abbildungsmodells soll überprüfen, ob die darin formulierten Abbildungen zwischen den Elementen des Fachmodells und den Elementen des Systemmodells Fehler aufweisen. Die *innere Konsistenz* des Abbildungsmodells ist zunächst davon abhängig, ob ihre einzelnen Transformationen in sich konsistent sind (angelehnt an [BBR11]).

Eine Transformation (vgl. Abschnitt 2.4) heißt *konsistent*, falls folgende Bedingungen erfüllt sind:

- *Konsistenz-Regel K 1:* Falls die Transformation vom Typ *Map* ist, dann besitzt sie genau einen Quellknoten und genau einen Zielknoten
- *Konsistenz-Regel K 2:* Falls die Transformation vom Typ *Split* ist, dann besitzt sie genau einen Quellknoten und mehr als einen Zielknoten
- *Konsistenz-Regel K 3:* Falls die Transformation vom Typ *Merge* ist, dann besitzt sie mehr als einen Quellknoten und genau einen Zielknoten
- *Konsistenz-Regel K 4:* Falls die Transformation vom Typ *Remove* ist, dann besitzt sie einen Quellknoten und keinen Zielknoten
- *Konsistenz-Regel K 5:* Falls die Transformation vom Typ *Insert* ist, dann besitzt sie keinen Quellknoten und genau einen Zielknoten

Listing 3.2: Bedingungen für Konsistenz einer Transformation

Zusammengefasst besagen die Konsistenz-Regeln *K1* bis *K5*, dass zu jedem Transformationstyp die Anzahl der Quellknoten und Zielknoten der Transformation passen muss. Die folgende Abbildung 3.18 zeigt zu jeder der Konsistenz-Regeln *K1* bis *K5* (siehe Listing 3.2) eine Transformation, die die zu ihrem Transformationstyp gehörige Regel verletzt.

Die *Transformationen 1* hat den Transformationstyp *Map* und besitzt einen einzigen Quellknoten, jedoch mehr als einen Zielknoten. Dies steht im Widerspruch zu Regel *K1* und die Transformation ist nicht konsistent. Die zweite Transformation der Abbildung weist den Transformationstyp *Split* auf. Um als konsistent zu gelten, muss für die Transformation die Regel *K2* erfüllt sein. Da jedoch weder genau ein einziger Quellknoten, noch mehrere Zielknoten zur Transformation gehören, ist diese Transformation inkonsistent. Die *Transformation 3* enthält ebenfalls nicht die korrekte Anzahl an Quellknoten und Zielknoten. Die Regel *K3* ist verletzt. Zur *Transformation 4* existieren Zielknoten, obwohl dies nach der Regel *K4* nicht zulässig ist. Zur *Transformation 5* des Typs *Insert* sind keine Quellknoten erlaubt. Hier ist die Regel *K5* verletzt. Als einzige Transformation im Abbildungsmodell erfüllt die *Transformation 6* mit der Anzahl ihrer Knoten die den Konsistenz-Regeln und ist konsistent. In ihrem Fall ist die Regel *K1* für den Transformationstyp *Map* zu erfüllen.

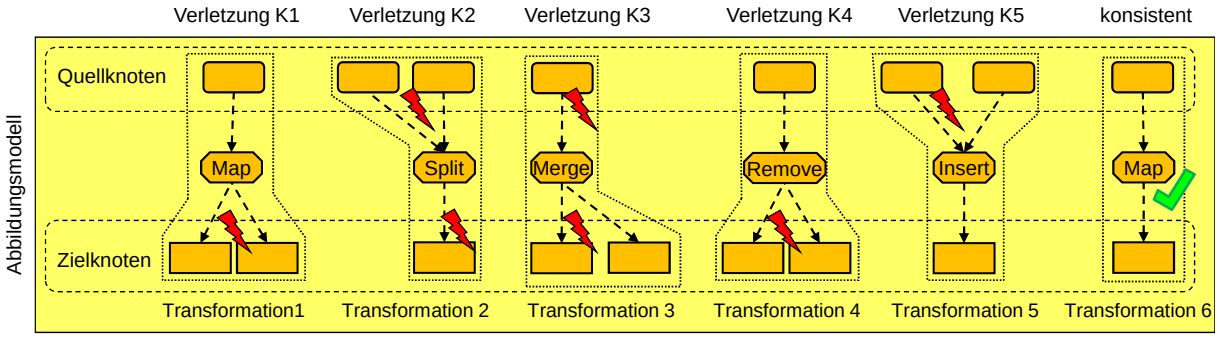

Abbildung 3.18: Verletzung der Konsistenzkriterien durch einzelne Transformationen

Basierend auf der Konsistenz der Transformationen ergibt sich die Definition der *inneren Konsistenz* des Abbildungsmodells (angelehnt an [BBR11]).

Ein Abbildungsmodell (vgl. Abschnitt 2.4) wird als *innerlich konsistent* bezeichnet, wenn folgende Bedingungen erfüllt sind:

- *Konsistenz-Regel K 6:* Alle Transformationen des Abbildungsmodells erfüllen die Bedingungen *K1* bis *K5* (vgl. Listing 3.2)
- *Konsistenz-Regel K 7:* Die Quellknotenmengen aller Transformationen sind paarweise disjunkt
- *Konsistenz-Regel K 8:* Die Zielknotenmengen aller Transformationen sind paarweise disjunkt
- *Konsistenz-Regel K 9:* Die Vereinigung der Quellknoten der einzelnen Transformationen ist identisch mit der Menge der Quellknoten des Abbildungsmodells
- *Konsistenz-Regel K 10:* Die Vereinigung der Zielknoten der einzelnen Transformationen ist identisch mit der Menge der Zielknoten des Abbildungsmodells

Listing 3.3: Bedingungen für Konsistenz des Abbildungsmodells

Falls bereits eine der Regeln *K6* bis *K10* nicht erfüllt ist, so nennt man das Abbildungsmodell *inkonsistent*. Zur Verdeutlichung dieser Regeln hier ein beispielhaftes Abbildungsmodell:

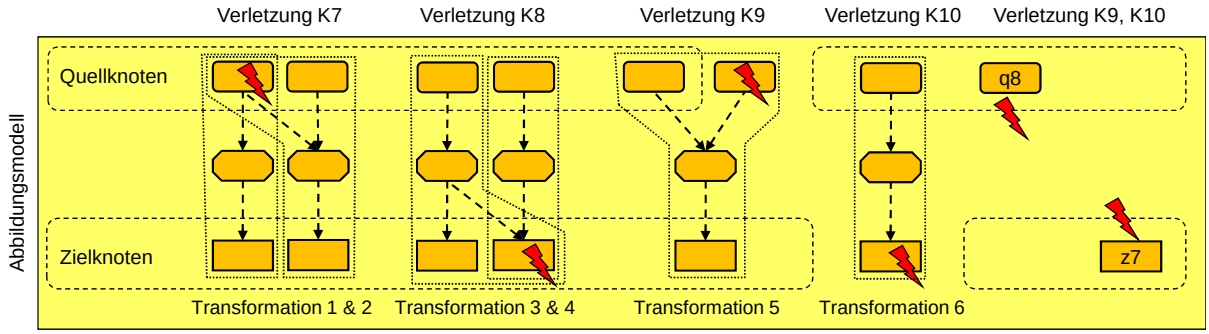

Abbildung 3.19: Inkonsistentes Abbildungsmodell

Die *Transformation 1* und die *Transformation 2* weisen in ihrer Quellknotenmenge einen selben Knoten auf und verletzen damit die Regel *K7*. Mit dieser ersten Verletzung einer der Konsistenzbedingungen gilt das Abbildungsmodell bereits als inkonsistent. Die *Transformation 3* und die *Transformation 4* haben einen gleichen Zielknoten und verletzen damit die Regel *K8*. Die *Transformation 5* weist einen Quellknoten auf, der nicht zur Quellknotenmenge des Abbildungsmodells gehört. Dies bedeutet eine Verletzung der Regel *K9*. Die *Transformation 6* besitzt einen Zielknoten, der nicht zur Zielknotenmenge des Abbildungsmodells gehört. Die Regel *K10* ist verletzt. Der Quellknoten *q8* ist in keine Transformation eingebunden, wodurch die Regel *K9* verletzt ist. Der Zielknoten *z7* wird ebenfalls durch keine Transformation eingebunden. Hier ist die Regel *K10* verletzt. Insgesamt sind in dem Abbildungsmodell die Konsistenzregeln in sechs Punkten verletzt, womit das Modell als inkonsistent gilt.

Folgender Algorithmus bewertet die Konsistenz des Abbildungsmodells nach der Regel *K6*. Dabei wird unterschieden, welchen Transformationstyp die untersuchten Transformationen besitzt.

Algorithmus 3.23 (Überprüfung der Konsistenz nach Regel *K6*)**:**

```
Input:  Q               Die Quellknoten des Abbildungsmodells
Input:  Z               Die Zielknoten des Abbildungsmodells
Input:  Transf          Die Transformationen des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Bedingung K6 erfüllt ist

for(transf:Transf)              Teste zu jeder Transformation transf
                                ob je nach Transformationstyp die Bedingung
                                K1, K2, K3, K4 oder K5 verletzt wird
 if(TransTyp(transf)=Map)
  if(|QTNodes(transf)|!=1 || |ZTNodes(transf)|!=1)
   return False
 if(TransTyp(transf)=Split)
  if(|QTNodes(transf)|!=1 || |ZTNodes(transf)|<=1)
   return False
 if(TransTyp(transf)=Merge)
  if(|QTNodes(transf)|<=1 || |ZTNodes(transf)|!=1)
   return False
 if(TransTyp(transf)=Remove)
  if(|QTNodes(transf)|!=1 || |ZTNodes(transf)|!=0)
   return False
 if(TransTyp(transf)=Insert)
  if(|QTNodes(transf)|!=0 || |ZTNodes(transf)|!=1)
   return False
return True                     Andernfalls ist K6 erfüllt
```

Der folgende Algorithmus 3.24 bewertet die Konsistenz des Abbildungsmodells nach der Regel *K7*:

Algorithmus 3.24 (Überprüfung der Konsistenz nach Regel *K7*)**:**

```
Input:  Q              Die Quellknoten des Abbildungsmodells
Input:  Z              Die Zielknoten des Abbildungsmodells
Input:  Transf         Die Transformationen des Abbildungsmodells
Output: True/False     Ein Wahrheitswert, ob die Bedingung K7 erfüllt ist

for(transf1:Transf)            Untersuche zu jeder Kombinationsmöglichkeit
                               zweier
 for(transf2:Transf)           Transformationenen (transf1 und transf2)
  if(transf1!=transf2)         die ungleich sind
   for(q1:QTNodes(transf1))    ob Elemente ihrer jeweiligen Quellknotenmengen
                               q1
    for(q2:QTNodes(transf2))   und q2
     if(q1==q2)                gleich sind, als von beiden gemeinsam verwendet
                               werden
      return True              Falls dem so ist, vermerke eine Verletzung der
                               Regel K7
return False                   Falls kein solcher Fall aufgetreten ist,
                               bestätige, dass keine Verletzung der Regel K7
                               vorliegt
```

Analog zu Algorithmus 3.24 gestaltet sich auch die Überprüfung der Regel *K8*. Hierzu werden nicht die Quellknoten über die Funktion *QTNodes* ermittelt und verglichen, sondern die Zielknoten über die Funktion *ZTNodes*. Aufgrund dieser Ähnlichkeit wird der Algorithmus hier nicht besprochen.

Der folgende Algorithmus 3.25 bewertet die Konsistenz des Abbildungsmodells nach der Regel *K9*. Um zu überprüfen, dass die Menge der Quellknoten der einzelnen Transformationen identisch mit der Menge der Quellknoten des Abbildungsmodells ist, wird zunächst die Anzahl der Mengenelemente erhoben. Anschließend werden die Elemente der Mengen einzelnen verglichen. Der Algorithmus benutzt die Funktion *Count(M)*, die die Anzahl der Elemente der Menge *M* liefert.

Algorithmus 3.25 (Überprüfung der Konsistenz nach Regel *K9*)**:**

```
Input:  Q               Die Quellknoten des Abbildungsmodells
Input:  Transf          Die Transformationen des Abbildungsmodells
Output: True/False      Ein Wahrheitswert, ob die Bedingung K9 erfüllt ist

countQinTransf=0                Zählvariable für die Menge der Quellknoten
                                in den Transformationen

for(transf:Transf)              Ermittle zu jeder Transformation transf
 QTNodes= QTNodes(transf)       die Menge ihrer Quellknoten
 countQinTransf+=Count(QTNodes) Ermittle die Mächtigkeit der Menge
                                und addiere diese zu der Zählvariablen
end

if(countQinTransf!=Count(Q))    Falls die ermttelte Anzahl der Quellknoten
                                aller Transformationen ungleich der Anzahl
                                der Quellknotendes Abbildungsmodells ist,
                                melde eine Verletzung der Regel K9
 return False


for(q:Q)                        Untersuche zu jedem Quellknoten des
                                Abbildungsmodells

 for(transf:Transf)             ob er bei einer Transformationen
  QTNodes= QTNodes(transf)      in der Menge ihrer Quellknoten
   if(q.notIn(QTNodes))         enthalten ist. Falls nicht,
                                melde eine Verletzung der Regel K9
    return False
   fi
  end
end

return True
```

Analog zu Algorithmus 3.25 gestaltet sich die Überprüfung der Regel *K10*. Hierzu werden anstatt der Quellknoten die Zielknoten verglichen. Aufgrund dieser Ähnlichkeit wird der Algorithmus hier nicht besprochen.

3.9 Zusammenfassung

Fachprozesse und Systemprozesse werden in Versionen verwaltet (vgl. Abschnitt 3.1). Durch Vergleiche zweier Prozessversionen wird identifiziert, welche Prozessschritte hinzugefügt, entfernt, oder in ihren Eigenschaften bearbeitet wurden (vgl. Abschnitt 3.2).

Ob Änderungen eines Fachprozesses durch Änderungen eines Systemprozesses ausgelöst wurden, wird durch Betrachtung der Transformationen des Abbildungsmodells ermittelt (vgl. Abschnitt 3.3).

Zu den geänderten Prozessschritten der Fach- und Systemprozesse werden in Abschnitt 3.4 Algorithmen entworfen, die die zu kontrollierende Transformationen des Abbildungsmodells und weitere betroffene Prozessschritte des korrespondierenden Prozesses ermitteln.

In Abschnitt 3.5 wird gezeigt, woran Bearbeitungen des Fachprozesses und des Systemprozesses identifiziert werden, die sich gegenseitig überschreiben.

Indirekte Abhängigkeiten bei Prozessänderungen werden in Abschnitt 3.6 ermittelt. Diese treten bei gemeinsam genutzten Datenobjekten auf.

Die Vorgehensreihenfolge der Modellierer bei der Anpassung der Prozesse wird in Abschnitt 3.7 geschildert. Dabei werden die Ergebnisse der Algorithmen aus den Abschitten 3.3, 3.4 und 3.6 verwendet.

In abschließend durchgeführten Konsistenzchecks in Abschnitt 3.8 auf strukturelle Konsistenz und innere Konsistenz wird bestätigt, ob die Anpassung der Prozesse zu einem konsistenten Ergebnis geführt hat.

4

Realisierung der Aufgabenliste

Eine Aufgabenliste besteht aus einer Menge von Aufgaben, durch deren Bearbeitung die Fachmodellierer und die Systemmodellierer die Inkonsistenzen der Fachprozesse und der Systemprozesse beheben. Sowohl die Fachmodellierer als auch die Systemmodellierer erhalten jeweils eine eigene Liste, die ausschließlich die zu ihren Prozessen passenden Aufgaben enthalten. Jede der Aufgaben bestehet dabei aus mehreren Komponenten, die mit Hilfe der in Kapitel 3 besprochenen Algorithmen gewonnen werden.

Der erste Bestandteil einer Aufgabe beinhaltet eine Prozessaktivität, die die Inkonsistenz auslöst. Sie wird im Folgenden als *Ursache* bezeichnet. Eine Ursache kann dabei in mehreren Varianten auftreten. Sie ist entweder eine zu den Prozessen hinzugefügte Aktivität, eine entfernte Aktivität, oder eine in ihren Eingenschaften bearbeitete Aktivität. Je nach Variante der Ursache wird sie über den entsprechenden Algorithmus aus Abschnitt 3.2 ermittelt. Für die hinzugefügten Aktivitäten ist dies der Algorithmus 3.1, für entfernte Aktivitäten der Algorithmus 3.2 und für in ihren Eigenschaften bearbeitete Aktivitäten der Algorithmus 3.5.

Zu diesem ersten Bestandteil einer Aufgabe gehört der zweite Teil, die im folgenden genannte *Auswirkung*. Die Auswirkung benennt, welches Objekt an die Ursache anzupassen ist. Zu einer Ursache kann es dabei mehrere Auswirkungen geben. Dies sind zum Einen bestimmte Transformationen des Abbildungsmodells (über die Algorithmen 3.12 und 3.14 ermittelt), zum anderen sind dies die Prozessaktivitäten des korrespondierenden Prozesses. Falls also die Ursache im Fachprozess auftritt, liegt die Auswirkung im Systemprozess (über Algorithmus 3.13 ermittelt), zu Ursachen im Systemprozess liegt die Auswirkung im Fachprozess (über Algorithmus 3.15 ermittelt). Zu den Ursachen werden auch die Objekte mit aufgenommen, die indirekt von einer Ursache betroffen sind (vgl. Abschnitt 3.6).

Einen Sonderfall an Aufgaben für die Aufgabenliste stellen konkurrierende Änderungen am Fachprozess und am Systemprozess dar (vgl. Abschnitt 3.5), da in diesen Fällen nicht feststeht, ob der Fachprozess dem Systemprozess oder der Systemprozess dem Fachprozess angepasst

werden soll. Die Fachmodellierer und die Systemmodellierer entscheiden nach einer Mitteilung über die konkurrierenden Änderungen darüber, welche der Änderungen Vorrang erhält.

4.1 Technische Umsetzung der Aufgabenliste

Die Herausforderung bei der Gestaltung der Aufgabenliste besteht darin, dass der Fachbereich und der IT-Bereich mit unterschiedlichen Modellierungswerkzeugen arbeiten, um den Fachprozess und den Systemprozessl zu gestalten. Aus diesem Grund gilt abzuwägen, ob eine generische, nicht auf die Modellierungswerkzeuge spezialisierte Aufbereitung der Aufgabenliste zu wählen ist, oder eine spezialisierte Lösung mit je einer Variante für eines der zwei Modellierungswerkzeuge. Die generische Lösung wird extern zu den Modellierungswerkzeugen realisiert, das heißt sie ist ein zu diesen unabhängig ausgeführtes Programm. Eine auf die Modellierungswerkzeuge angepasste Umsetzung wird hingegen über Add-Ons realisiert, die deren Funktionsumfang erweitern.

4.1.1 Realisierung als eigenständige Applikation

Eine Lösung als externes Programm bietet den Vorteil, dass dieses nicht an die Modellierungswerkzeuge angepasst werden muss, beziehungsweise die Modellierungswerkzeuge keiner Anpassung bedürfen, mit der sie die Aufgaben importieren können. Die technische Umsetzung und die Gestaltung stehen völlig frei und können unabhängig davon realisiert werden, ob die Modellierungswerkzeuge über Add-ons erweitert werden können. Der Nachteil hieraus ist, dass der Funktionsumfang und die Präsentationsform der Modellierungswerkzeuge nicht nativ im externen Programm gegeben sind. Werden diese benötigt, müssen sie nachgebildet werden. Dadurch, dass die externe Lösung nicht an die Modellierungswerkzeuge angepasst wird, ergibt sich der Vorteil, dass die Modellierer des Fachbereichs auch die Aufgabenliste des IT-Bereichs verstehen können, wie auch die Modellierer des IT-Bereichs die des Fachbereichs. Wechseln sie ihren Aufgabenbereich oder unterstützen ihre Kollegen, bedürfen sie keiner Umgewöhnung. Als Nachteil gilt hingegen, dass die Modellierer bei der Einsicht ihrer Aufgaben im externen Programm keine Möglichkeit haben, sich die betroffenen Objekte im Modellierungswerkzeug unmittelbar und automatisch aufzeigen zu lassen. Die Position der Objekte im Modell muss den Modellierern bereits bekannt sein oder von diesen ermittelt werden. Die Positionsermittlung der zu bearbeitenden Modellelemente kann lediglich dadurch erleichtert werden, dass das Modell ebenfalls im externen Programm dargestellt wird und die Position der Elemente in der Kopie angezeigt wird. Wie bereits erwähnt, ist dies eine unnötige Nachbildung des Funktionsumfangs der Modellierungswerkzeuge.

4.1.2 Integration in Modellierungswerkzeuge

Die Realisierung der Aufgabenlisten als interne Lösung, das heißt eine Erweiterung der Modellierungswerkzeuge um eine Importfunktion und Bearbeitungsfunktion der Listen, ist von

der Erweiterbarkeit der Tools abhängig (Das ARIS Toolset der Software AG beispielsweise unterstützt Visual Basic und Javascript zur Erweiterung des Funktionsumfangs mit Add-Ons). Da der Fachbereich und der IT-Bereich mit unterschiedlichen Programmen arbeiten, muss mit der internen Lösung für jedes dieser Programme ein eigenes Add-on geschrieben werden und es entsteht somit doppelter Arbeitsaufwand. Nur wenn die Modellierungswerkzeuge einheitliche, gemeinsame Schnittstellen (APIs) anbieten, kann eine zu beiden kompatible Realisierung eingesetzt werden. Die Realisierung als interne Lösung bietet den Vorteil, dass auf den Funktionsumfang der Modellierungswerkzeuge zurückgegriffen werden kann. Eine Nachbildung des Funktionsumfangs der Modellierungswerkzeuge, die bei einer externen Umsetzung benötigt wird, findet nicht statt. Bei der Miteinbeziehung des vorhandenen Funktionsumfangs ist insbesondere die Identifizierung und Positionsbestimmung von Modellelementen nützlich. Die Modellierer müssen nicht erst die zu bearbeitenden Objekte in den Modellen suchen, sondern können diese direkt in den Modellen markieren oder in der Sicht zentrieren lassen. Mit Add-ons für die zwei Modellierungswerkzeuge des Fachmodells und des Systemmodells bietet sich die Möglichkeit, die Aufgabenlisten aus Sicht der Modellierer als deren nativer Bestandteil wahrzunehmen, falls die Aufgabenlisten und gewöhnlichen Komponenten der Modellierungswerkzeuge gleich gestaltet sind. Ein weiterer Vorteil der internen Realisierung von Aufgabenlisten für die Modellierer ist, dass für die Betrachtung und Bearbeitung der Liste kein weiteres Programm als das jeweilige Modellierungswerkzeug zu öffnen ist. Ein Datenaustausch über Import/Eport entfällt und die Aufgabenliste kann dynamisch aktualisiert werden, falls ein Modellelement bearbeitet wird.

4.2 Gestaltungsmöglichkeiten der Aufgabenliste

Nach der Entscheidung über eine Integration der Aufgabenliste in die Modellierungswerkzeuge muss eine passende visuelle Präsentation gewählt werden, die die zur Verfügung stehenden technischen Möglichkeiten ausschöpft.

4.2.1 Sortierung der Aufgabenliste

Wie eingangs besprochen bestehen die Aufgaben der Aufgabenliste aus Ursachen und aus Auswirkungen. Eine Ursache hat dabei eine oder mehrere Auswirkungen. Dabei ist nicht ausgeschlossen, dass unterschiediche Ursachen die selbe Auswirkung teilen, das heißt eine Auswirkung kann auch mehrere Ursachen haben. Daras folgt, dass die Aufgabenlisten sowohl primär nach den Ursachen sortiert werden können, als auch primär nach den Auswirkungen. Je nach Wahl werden demnach in den Aufgabenlisten zuerst Informationen über die Ursachen von Inkonsistenzen angezeigt, oder die Auswirkungen. Wichtig ist dabei aufgrund der sequenziellen Auflistung, dass die jeweils anderen Beziehungen nicht angezeigt werden beziehungsweise aus der Liste nicht unmittelbar zu entnehmen sind. Werden die Aufgaben primär nach Ursachen sortiert, ist für die Modellierer nicht ersichtlich, ob die zugehörigen Auswirkungen mit weiteren Ursachen verbunden sind. Werden die Aufgaben hingegen primär nach Auswirkungen sortiert, ist für die Modellierer nicht ersichtlich, in welchen Beziehungen die zugehörigen Ursachen stehen.

Was zunächst als nachteilhafter Informationsverlust anmutet, kann aber auch von Vorteil sein. Durch die sequentielle Anordnung der Aufgaben und Ausblendung der vielen Beziehungen zwischen Ursachen und Wirkungen werden die Modellierer nicht mit Informationen überhäuft, die zur Bearbeitung der einzelnen Aufgaben unrelevant sind, weil zu einem späteren Zeitpunkt einsehbar.

Eine gänzlich andere Vorgehensweise der Aufgabenaufbereitung ist die, das Geflecht aus Beziehungen zwischen Ursachen und Wirkungen nicht sequenziell darstellen zu wollen, sondern diese Informationen mit Hilfe eines Graphen aufzuzeigen. Dies könnte direkt in den Fachprozessen und Systemprozessen realisiert werden.

Ob Aufgabenlisten nach Ursachen oder Auswirkungen sortiert werden oder aber als Graph dargestellt werden, hängt von der technischen Realisierbarkeit und dem Wunsch der Modellierer ab. Werden die Aufgaben in den Modellierungswerkzeugen angezeigt, so so ist die Verwendung und Erweiterbarkeit von deren Funktionsumfang ausschlaggebend.

4.2.2 Darstellung und Präsentation für den Modellierer

Ein einfaches Beispiel für die Umsetzung einer Aufgabenliste ist ihre Präsentation in Textform. Die Aufgaben für die Modellierer im Fach- und Systembereich werden in je einer Textdatei gespeichert, welche durch Textprogramme geöffnet und angezeigt werden. Die Textdarstellung der Aufgabenliste und die Modellierung des Fach- bzw. Systemmodells sind somit durch unterschiedliche Programme realisiert. Die Modellierer des Fach- und IT-Bereichs erhalten und öffnen selbstverständlich nur die Aufgabenliste, die für ihr Modell bestimmt ist.

Der Vorteil der Textform ist, dass die Textdatei der Aufgabenliste zur Bearbeitung von jedem Texteditor geöffnet, beziehungsweise editiert werden kann (zum Beispiel mit Notepad, MS Office-Word, Kate) und plattformunabhängig ist. Als Text fehlt dem Modellierer jedoch eine grafische Darstellung der Aufgaben und der in ihnen erwähnten Modellobjekte, die ihn in seinem Verständnis der Aufgaben unterstützen könnte. Die Selbstbeschreibungsfähigkeit fehlt, womit die Liste nicht intuitiv bearbeitet werden kann. Als weitere Nachteil sind zu nennen, dass die Textform keinen Komfort und Individualisierbarkeit wie beispielsweise eine Sortierfunktion oder Löschfunktion bearbeiteter Aufgaben bietet. Für die Aufbereitung der Aufgaben in Textform gibt es mehrere Möglichkeiten.

Die Informationen, die über die Verfahren zur Änderungserkennung (siehe Abschnitt 3.2) und Betroffenenermittlung (siehe Abschnitt 3.4 und 3.6) ermittelt werden, können in Sätzen ausformuliert werden. Die Modellierer des Fach- und IT-Bereichs können die somit generierte Aufgabenliste unmittelbar beim Lesen verstehen und benötigen keine Schulung über Notationen wie XML[1], JSON[2] oder YAML[3].

Eine Möglichkeit für die Anordnung und Gestaltung der Aufgabenliste mit ausformulierten Sätzen in einer Textdatei zeigt Abbildung 4.1. Die Aufgaben sind untereinander aufgelistet und

[1] Extensible Markup Language
[2] JavaScript Object Notation
[3] YAML Ain't Markup Language

aus Gründen der Übersichtlichkeit jeweils durch eine Leerzeile getrennt. Jede Aufgabe besteht aus mehreren Zeilen. Die erste gibt das jeweils zu bearbeitende Objekt des Modells an (zum Beispiel „HT-Edit“ ①). Jede weitere Zeile formuliert eine zugehörige Ursache (beispielsweise wurde das Datenobjekt „Kunde“ gelöscht ②). Wie in Abschnitt **??** besprochen, kann die Datei der Aufgabenliste bei ihrer Generierung primär nach geänderten Objekten oder betroffenen Objekten sortiert werden.

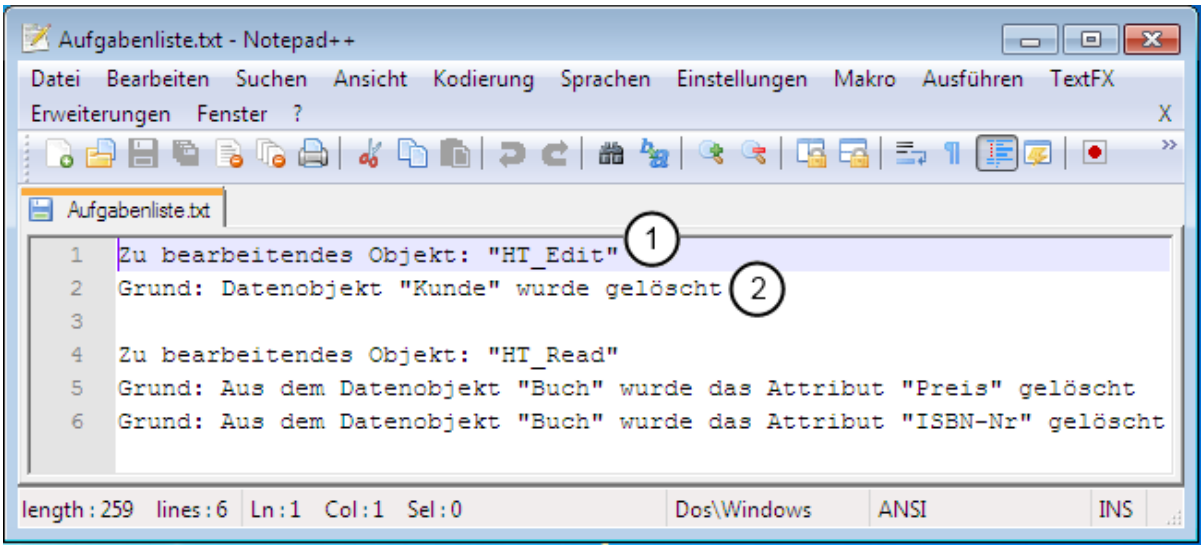

Abbildung 4.1: Aufgabenliste als Textdatei mit ausformulierten Sätzen

Des weiteren kann die Aufgabenliste als strukturierte Markup-Datei, wie zum Beispiel in XML, gespeichert werden. Dies bietet den Vorteil, dass die Informationen im Vergleich zu oberem Ansatz der Ausformulierung strukturiert sind. Je nach Textprogramm können Details der XML-Datei ausgeblendet werden. Der Nachteil hierbei ist, dass insbesondere die Modellierer des Fachbereichs im Verständnis von XML-Dateien geschult werden müssen. Des Weiteren gelten die sonstigen Vor- und Nachteile der Textform auch für XMLs.

Bei der Strukturierung der Aufgabenliste als XML bietet sich an, die Gründe und Auswirkungen in den Tags <Aufgaben>, <Aufgabe>, <Ursachen> und <Ursache> zu sortieren, die in einer Baumstruktur und damit hierarchisch organisiert werden. Das Wurzelelement unseres Beipiels ist <Aufgaben> (siehe Abbildung 4.2 ①), unter dem die Aufgaben jeweils zwischen dem Start-Tag <Aufgabe> und End-Tag </Aufgabe> stehen ②. Die Gründe für die Bearbeitung eines jeden Objekts werden als Kindelement der Aufgaben betrachtet ③. Im Texteditor kann die Baumstruktur der XML in Teilen manuell „zusammengeklappt“ und wieder „ausgeklappt“ werden ④.

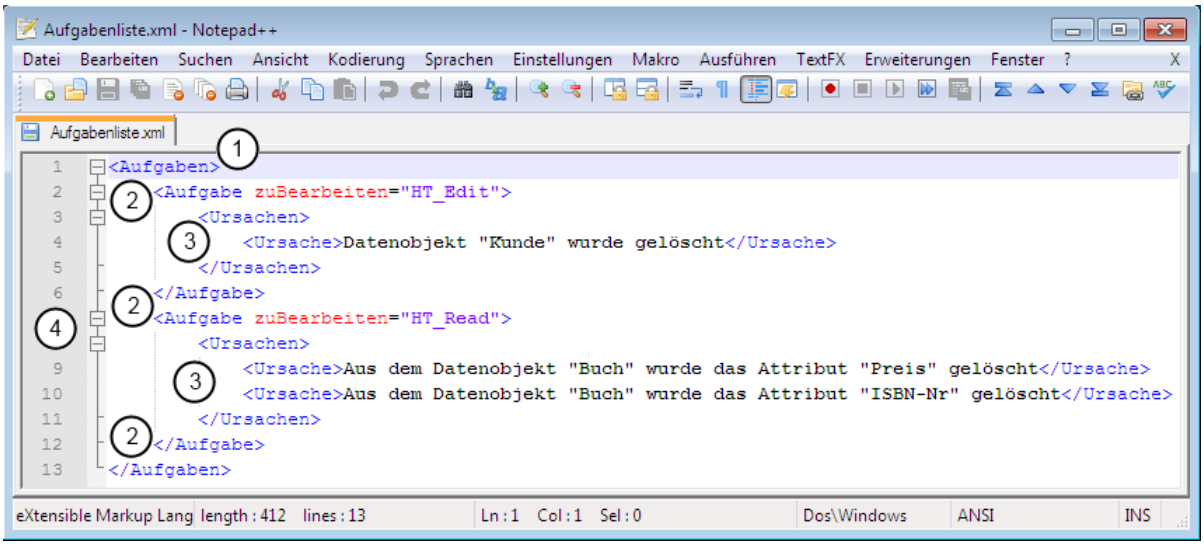

Abbildung 4.2: Aufgabenliste als XML-Datei

Die Struktur der in Abbildung 4.2 gezeigten XML-Datei wird durch folgende DTD[4] beschrieben:

```
1 <!ELEMENT Aufgaben (Aufgabe)*>
2 <!ELEMENT Aufgabe (Ursachen)>
3 <!ELEMENT Ursachen (Ursache)+>
4 <!ELEMENT Ursache (#PCDATA)>
5 <!ATTLIST Aufgabe zuBearbeiten CDATA #REQUIRED>
```

Als weitere Alternative kann die Aufgabenliste als Tabelle gespeichert und dargestellt werden. Damit ist gemeint, dass die Informationen der Aufgaben in Zeilen und Spalten organisiert werden. Technisch geschieht die Spaltensetzung in einer Plain-Text-Datei durch gezieltes Einrücken (etwa mit Space und Tabulator), in einer CSV[5] durch Setzen des Trennzeichens „Semicolon“. Die Spalten geben der Aufgabenliste im Vergleich zur bloßen Ausformulierung mehr Struktur und bieten eine bessere Übersicht. Im Gegensatzt zur XML können jedoch keine Details ausgeblendet werden.

Ein Beipiel für die tabellarische Darstellung zeigt Abbildung 4.3. Die zu bearbeitenden Modellobjekte werden in Spalte A aufgelistet ①, die Ursachen werden in Spalte B genannt ②. Da zu einer Aufgabe mehrere Gründe auftreten können ③, weist die Spalte A Leerräume auf ④. Die zugehörige CSV-Datei wird in Abbildung 4.4 dargestellt. Wie bereits beschrieben werden die Werte der Spalten durch Semicolon getrennt.

[4]Document Type Definition
[5]Comma-Separated Values

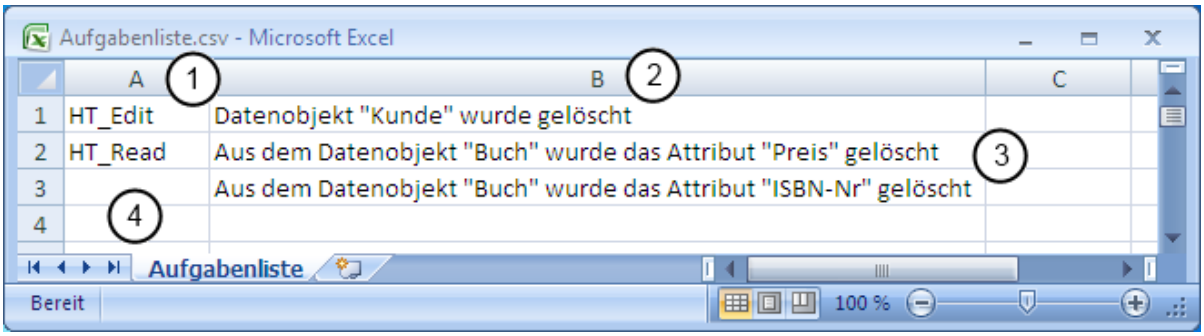

Abbildung 4.3: Aufgabenliste als CSV-Datei - Tabellenansicht

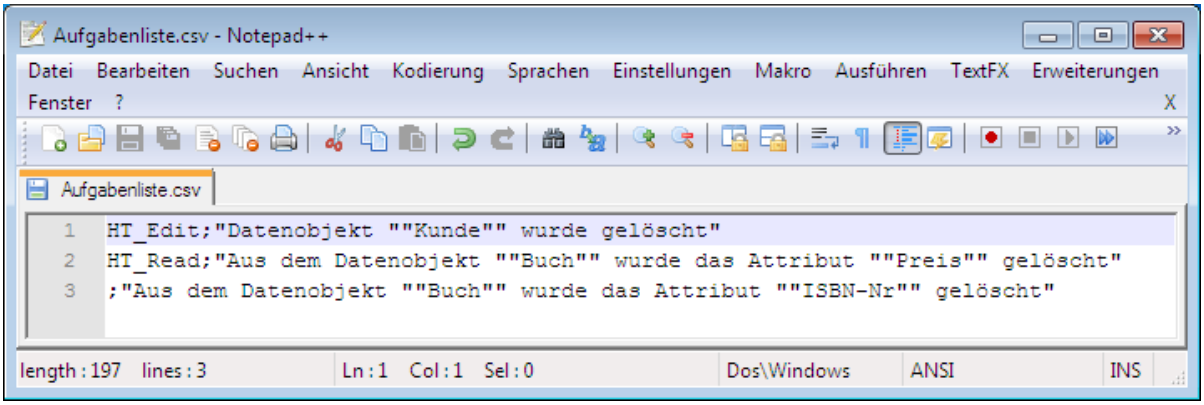

Abbildung 4.4: Aufgabenliste als CSV-Datei - Textansicht

Anhand der vielen Nachteile der Umsetzung einer Aufgabenliste als Textdatei ist ersichtlich, dass eine graphische Benutzeroberfläche als alternative Umsetzung zu diskutieren ist.

Graphische Markierungen helfen den Modellierern zu erkennen, welche Modellelemente von ihnen geändert wurden. Abbildung 4.5 zeigt ein geändertes Fachmodell ①. Über das Abbildungsmodell wird identifiziert, welche Prozessknoten im korrespondierenden Modell betroffen sind. Die betroffenen Transformationen ② und Elemente des Systemmodells ③ werden markiert und zeigen, dass sie noch anzupassen sind.

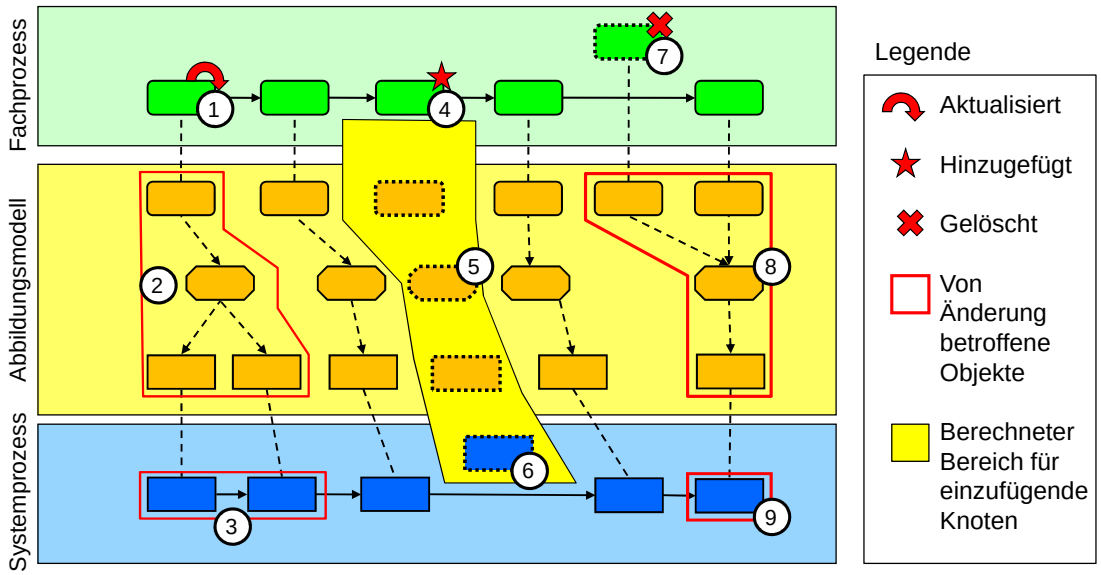

Abbildung 4.5: Markierungen nach Aktualisierung des Fachprozesses

Hinzugefügte Modellelemente werden direkt und ohne einen Platzhalter markiert ④. Da zu diesen noch keine Abbildungen existieren, ist eine Markierung der noch anzupassender Elemente nicht möglich. Statt dessen kann ein Bereich markiert werden ⑤, der über die Position der dem Kontrollfluss zu entnehmenden Vorgänger und Nachfolger berechnet wird. Dieser Bereich zeigt auf, wo die Elemente im korrespondierenden Modell eingefügt werden können ⑥.

Für ein entferntes Modellelement ist eine Markierung nicht möglich. Es kann jedoch ein Platzhalter in der Nähe des ehemaligen Elements eingefügt werden, der an seiner statt markiert wird ⑦. Die Position des Platzhalters kann aus der alten Modellversion entnommen werden.

Graphische Repräsentationen der Aufgaben drohen mit der wachsenden Komplexität der Prozesse eine Unübersichtlichkeit zu erzeugen, falls die Prozessmodelle mit Markierungen angereichert werden. Dabei ist nicht garantiert, dass der Funktionsumfang der Modellierungswerkzeuge eine entsprechende Markierung zulässt.

5 Prototypische Umsetzung

In diesem Kapitel werden die in Kapitel 3 vorgestellten Algorithmen zur Identifizierung von Änderungen und Inkonsistenzen in den Fachprozessen und den Systemprozessen in einem Prototyp umgesetzt. Es handelt sich dabei um eine Applikation, die Aufgabenlisten (vgl. Kapitel 4) für die Modellierer generiert.

Der Aufbau dieses Kapitels ist wie folgt: In Abschnitt 5.1 werden Grundlagen zum Prototyp beschrieben. Danach wird in Abschnitt 5.2 der Datenaustausch mit den Modellierungstools besprochen. In einem weiteren Abschnitt 5.3 wird die Umsetzung der Algorithmen behandelt. In Abschnitt 5.4 wird dargelegt, wie die Berechnungsergebnisse der implementierten Konsistenztests in einer grafischen Benutzeroberfläche aufbereitet werden. Den Abschluss bildet Abschnitt 5.5 mit einer Zusammenfassung.

5.1 Grundlagen

Der Prototyp (*consistency manager*) wurde als eigenständige Cross-platform Anwendung mittels Java [**?**] und der Entwicklungsumgebung Eclipse [**?**] erstellt. Als Datenaustauschformat zwischen dem Prototyp und den Modellierungswerkzeugen der Modellier dienen standartisierte XML-Dateien. In Abbildung 5.1 ist die Architektur des *consistency managers* dargestellt. Ein Fachprozess ①, ein Abbildungsmodell ② und ein Systemprozess ③ werden über eine Schnittstelle als XML importiert. Als weitere Eingabe erhält der consistency manager drei fakultative Eingaben. Dies sind eine Folgeversion des Fachprozesses ④, ein neues Abbildungsmodell ⑤ und eine Folgeversion des Systemprozesses ⑥. Der Prototyp erhält diese fakultativen Eingaben nur dann, wenn der Fachprozess oder der Systemprozess bearbeitet wurden und tatsächlich in einer neuen Version vorliegen. Die Analyse funktioniert unabhängig davon, ob beide Prozesse weiterentwickelt wurden, jedoch muss er mindestens eine Folgeversion als Eingabe erhalten.

Die Eingabe eines überarbeiteten Abbildungsmodells ist grundlegend optional, jedoch wird für die Analysen das jeweils aktuellste Abbildungsmodell verwendet. Zu jeder eingereichten Folgeversion eines Prozesses wird ein Differenzabgleich mit der Vorversion gemacht. Weiter wird untersucht, ob eine Aktion oder Reaktion vorliegt. Dann werden die Auswirkungen auf den korrespondierende Prozess und das Abbildungsmodell errechnet, sowie konkurrierende Elemente identifiziert. Das ganze wird im Aufgabengenerator zu einer Einheit kombiniert und schließlich als XML-Datei exportiert.

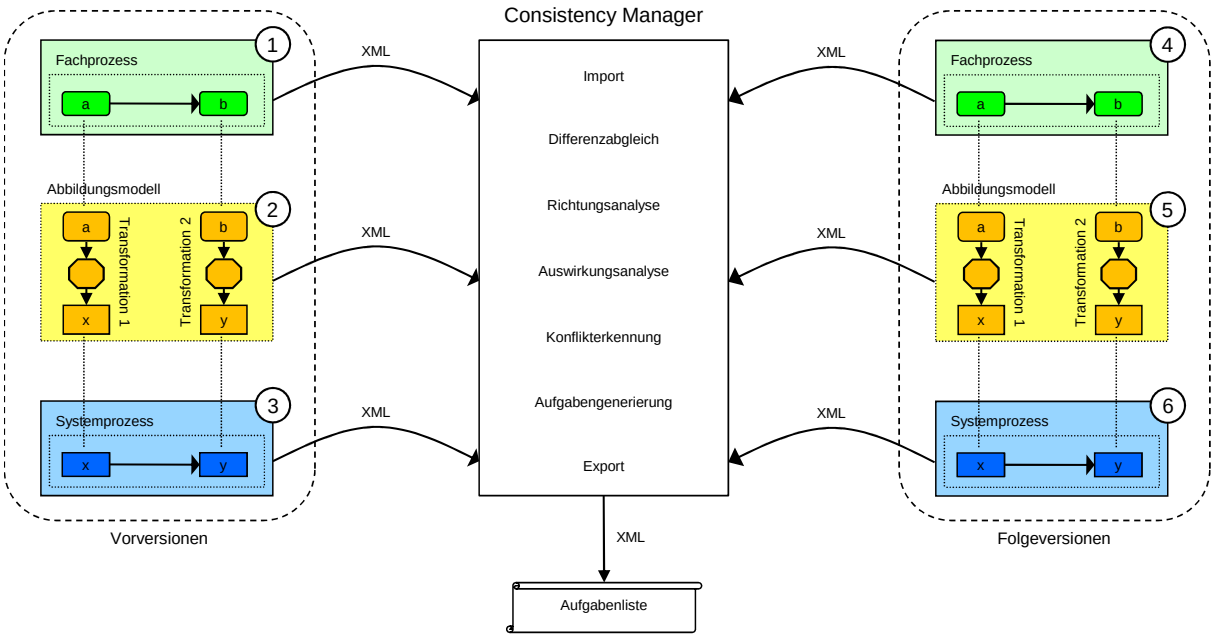

Abbildung 5.1: Funktionen und Schnittstellen des Prototyps

5.2 Umsetzung des Prozessimports

Für den Datenaustausch zwischen dem Prototyp und den Modellierungswerkzeugen werden standartisierte XML-Dateien verwendet, deren Struktur im folgenden gezeigt wird. Falls die als XML exportierenden Prozesse der Modellierungswerkzeuge dieser Struktur nicht entsprechen, können sie über eine XSL-Transformation [**?**] angepasst werden.

Die Struktur der XML für einen eingelesenen Fachprozess zeigt Listing 5.1. In der XML wird die Liste der im Fachprozess vorliegenden Aktivitäten (vgl. Zeile 2) gespeichert. Zu jeder von diesen Aktivitäten liegen mindestens die folgenden drei Attribute vor: Im Attribut *ID* wird gespeichert, über welchen eindeutigen Bezeichner die Fachprozessaktivität identifiziert werden kann (z.B. „*fpa1*“ oder „*fpa99*“, vgl. Zeile 3 und 10). Das Attribut *lastModified* gibt den Zeitpunkt an, zu dem die Aktivität zuletzt bearbeitet wurde. Das Attribut *name* zeigt auf, welchen Namen die Aktivität im Modellierungswerkzeug erhalten hat (z.B. „*Änderungsantrag stellen*“, vgl. Zeile 3). Als weitere Attribute der Aktivität sind ihre Lese- und Schreibzugriffe auf Datenobjekt gespeichert (vgl. Zeile 4, 5 und 11). Diese Zugriffe werden nicht direkt als XML-Attribute in den Entities der

Aktivitäten gespeichert, da mehr als ein Lese- oder Schreibzugriff (vgl. Zeile 4 und 5) möglich sind, Entitities jedoch nicht mehrere Attribute mit dem selben Namen besitzen dürfen. Dadurch wird die Wohlgeformtheit des XML-Dokuments eingehalten. Des weiteren sind in der XML die Liste der Datenobjekte gespeichert (vgl. Zeile 15 bis 21), die von den Prozessaktivitäten gelesen oder geschrieben werden. Auch diese besitzen die drei obligatorischen Attribute *ID*, *lastModified* und *name* (vgl. Zeile 16 und 20). Weitere fakultative Attribute sind im Gegensatz zu den Aktivitäten nicht vorhanden (vgl. Zeile 16 und 20).

```
1  <Fachprozess>
2   <Aktivitaeten>
3    <Aktivitaet ID="fpa1" lastModified="01.01.2011;00:00:00" name="Aenderungsantrag␣stellen">
4     <Zugriff name="reads">fpdo12</Zugriff>
5     <Zugriff name="reads">fpdo34</Zugriff>
6    </Aktivitaet>
7    ...
8    <!--weitere Aktivitaeten-->
9    ...
10   <Aktivitaet ID="fpa99" lastModified="15.04.2011;00:00:00" name="Aenderung␣umsetzen">
11    <Zugriff name="writes">fpdo12</Zugriff>
12   </Aktivitaet>
13  </Aktivitaeten>
14
15  <Datenobjekte>
16   <Datenobjekt ID="fpdo1" lastModified="01.01.2011" name="Aenderungsantrag"/>
17   ...
18   <!--weitere Datenobjekte-->
19   ...
20   <Datenobjekt ID="fpdo67" lastModified="09.09.2011" name="Archiv"/>
21  </Datenobjekte>
22 </Fachprozess>
```

Listing 5.1: XML-Code eines Fachprozesses

Die XML-Dateien der Systemprozesse sind vom Aufbau wie die XML-Dateien für Fachprozesse strukturiert. Auch sie beinhaltet Aktivitäten und Datenobjekte (vgl. Zeile 2 und 13 in Listing 5.2).

```
1  <Systemprozess>
2   <Aktivitaeten>
3    <Aktivitaet ID="spa1" lastModified="01.01.2011;00:00:00" name="Aktivitaet␣1">
4     <Zugriff name="reads">spdo13</Zugriff>
5     <Zugriff name="writes">spod37</Zugriff>
6    </Aktivitaet>
7    ...
8    <!--weitere Aktivitaeten-->
9    ...
10   <Aktivitaet ID="spa73" lastModified="03.09.2011;00:00:00" name="Aktivitaet␣73"/>
11  </Aktivitaeten>
12
13  <Datenobjekte>
14   <Datenobjekt ID="spdo1" lastModified="04.01.2011;00:00:00" name="Datenobjekt␣1"/>
15   ...
16   <!--weitere Datenobjekte-->
17   ...
18   <Datenobjekt ID="spdo75" lastModified="07.04.2011;00:00:00" name="Datenobjekt␣75"/>
19  </Datenobjekte>
20 </Systemprozess>
```

Listing 5.2: XML-Code eines Systemprozesses

Die XML eines Abbildungsmodells enthält zunächst die Menge der Quellknoten (vgl. Zeile 3 bis 9 in Listing 5.3). Zu jedem Quellknoten ist im Attribut *ID* gespeichert, über welchen

eindeutigen Bezeichner er identifiziert werden kann (vgl. Zeile 4, 8 und 12). Jeder Quellknoten weist außerdem die Attribute *refID* und *refLastModified* auf, welche die Attribute *ID* und *lastModified* einer Fachprozessaktivität referenzieren (vgl. Zeile 4 und 8). Die Menge der Zielknoten ist ebenfalls in der XML gespeichert (vgl. Zeile 11 bis 17). Ihre Attribute *refID* und *refLastModified* referenzieren die Attribute *ID* und *lastModified* von Systemprozessaktivitäten (vgl. Zeile 12 und 16). Als weiteren Bestandteil enthält die XML die Menge der Transformationen (vgl. Zeile 19 bis 39), welche ebenfalls einen eindeutigen Bezeichner *ID* aufweisen und ihren Transformationstyp im Attribut *type* angeben (vgl. Zeile 20 und 31). Zu den Transformationen wird gespeichert, welche der vorhergehenden Quellknoten und Zielknoten sie aufeinander abbilden (vgl. Zeile 21 und 26).

```
1  <Abbildungsmodell>
2
3   <Quellknotenmenge>
4    <Quellknoten ID="qk1" refID="fpa1" refLastModified="01.01.2011"/>
5    ...
6    <!--weitere Quellknoten-->
7    ...
8    <Quellknoten ID="qk9" refID="fpa9" refLastModified="09.09.2011"/>
9   </Quellknotenmenge>
10
11  <Zielknotenmenge>
12   <Zielknoten ID="zk1" refID="spa1" refLastModified="01.01.2011"/>
13   ...
14   <!--weitere Zielknoten-->
15   ...
16   <Zielknoten ID="zk9" refID="spa9" refLastModified="09.09.2011"/>
17  </Zielknotenmenge>
18
19  <Transformationen>
20   <Transformation type="Map" ID="t1">
21    <Quellknotenmenge>
22     <Quellknoten ID="qk1"/>
23    </Quellknotenmenge>
24    <Zielknotenmenge>
25     <Zielknoten ID="zk1"/>
26    </Zielknotenmenge>
27   </Transformation>
28   ...
29   <!--weitere Transformationen-->
30   ...
31   <Transformation type="Map" ID="t9">
32    <Quellknotenmenge>
33     <Quellknoten ID="qk9"/>
34    </Quellknotenmenge>
35    <Zielknotenmenge>
36     <Zielknoten ID="zk9"/>
37    </Zielknotenmenge>
38   </Transformation>
39  </Transformationen>
40
41 </Abbildungsmodell>
```

Listing 5.3: XML-Code eines Abbildungsmodells

Zur Veranschaulichung, welche Attribute die Fachprozess- und Systemprozessaktivitäten, die Quellknoten, Zielknoten und Transformationen des Abbildungsmodells in einer exportierten XML besitzen, werden diese in Abbildung 5.2 dargestellt. Insbesondere wird in de Abbildung gezeigt, dass die Quellknotenattribute *refID* und *refLastModified* die Attribute *ID* und *lastModified* der Fachprozessaktivitäten referenzieren. Die Attribute *refID* und *refLastModified* der Zielknoten

referenzieren die Attribute *ID* und *lastModified* der Systemprozessaktivitäten.

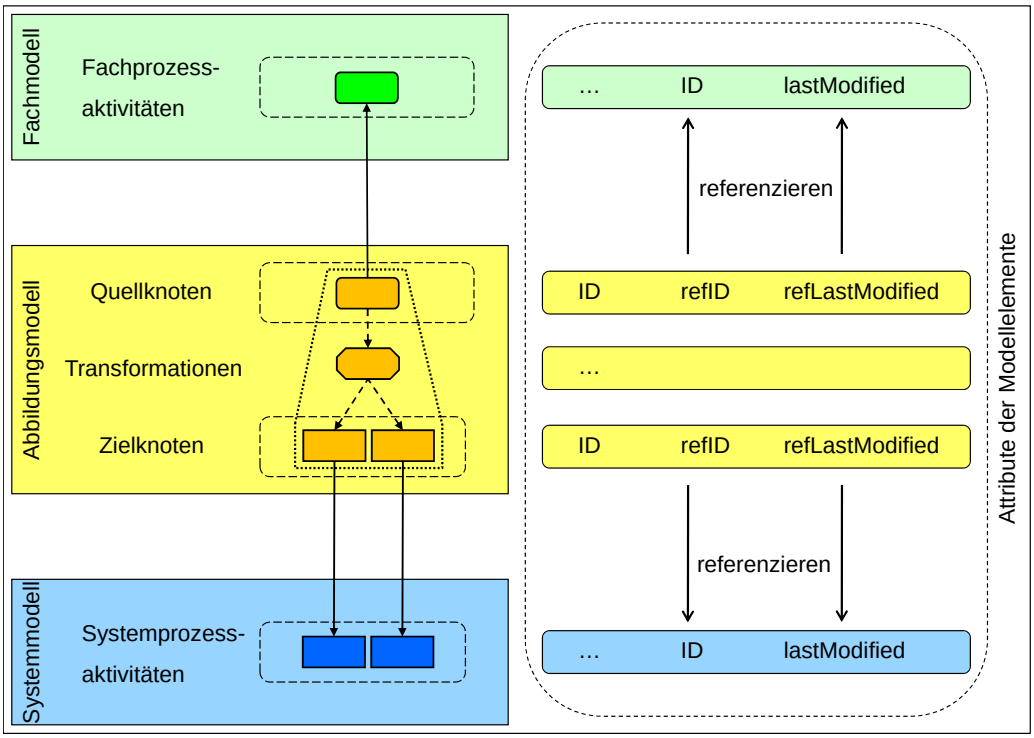

Abbildung 5.2: Attributbeziehungen über Modellgrenzen hinweg

Zum Import der XML-Dateien wird die Klasse SAXBuilder der JDOM-Schnittstelle verwendet, welche die XML-Dokumente als Baum im Hauptspeicher repräsentiert. Listing 5.4 zeigt, wie mit Hilfe des SAXBuilders der Name des Wurzelelements einer XML-Datei ausgelesen und ausgegeben wird. Auf die XML eines Abbildungsmodells (vgl. Listing 5.3) angewandt, wird hierdurch der Name „*Abbildungsmodell*“ ausgegeben, zu einer Systemprozess-XML (vgl. Listing 5.2) wird die Zeichenfolge „*Systemprozess*“ ausgegeben.

```
1 Document fachprozessDokument = new SAXBuilder().build( xmlfile.getAbsolutePath() );
2 Element el= (Element)fachprozessDokument.getContent(0); //get child element at index zero
3 System.out.println(el.getName());
```

Listing 5.4: XML-Code eines Abbildungsmodells

Aus den über den SAXBuilder eingelesenen Dokumenten werden all diejenigen Informationen ausgelesen, die für die Implementierung der Algorithmen 3.1 bis 3.24 in diesem Prototyp benötigt werden. Listing 5.5 zeigt als Beispiel hierfür die Implementierung der Get-Methode *getFachprozessAktivitaetenIDs*, die *IDs* der Aktivitäten eines Fachprozesses liefert. Dazu wird in einem XPATH-Ausdruck angegeben, an welcher Position in der XML-Datei sich die zu ermittelnden *IDs* befinden. Die Rückgabe erfolgt als eine Menge von Zeichenketten (Strings).

```
1 /**
2  * @return Die IDs der Fachprozessaktivitaeten
3  */
4 public static Vector<String> getFachprozessAktivitaetenIDs(){
```

```
5       //Platzhalter fuer zu sammelnde ID-Attribute
6       List<Attribute> ID_Attributes = null;
7       try {
8           //Ermittlung der ID-Attribute
9           ID_Attributes = XPath.selectNodes( fachprozessDokument, "/child::Fachprozess/child::
                Aktivitaeten/child::Aktivitaet/attribute::ID" );
10      } catch (JDOMException e1) {
11          e1.printStackTrace();
12      }
13      //Rueckgabe der Werte der ID-Attribute als eine Menge von String
14      Vector<String> toReturn = new Vector<String>();
15      for(int i = 0; i < ID_Attributes.size(); i++)
16          toReturn.add( ID_Attributes.get(i).getValue() );
17      return toReturn;
18  }
```

Listing 5.5: Ermittlung von Fachprozessaktivitäten-IDs aus einer XML-Datei per XPATH

Analog zu der Get-Methode *getFachprozessAktivitaetenIDs* (vgl. Listing 5.5) sind auch die *IDs* der Datenobjekte auslesbar, sowie die Attribute *lastModified*, *name* und die Zugriffe der Aktivitäten auf die Datenobjekte. Aus Gründen der Vereinfachung werden die entsprechenden Get-Methoden hier nicht dargestellt.

5.3 Umsetzung der Algorithmen

Wie an Listing 5.5 bereits ersichtlich, arbeiten alle Implementierungen der Algorithmen mit den *IDs* als eindeutige Bezeichner der Bestandteile des Fachprozesses, des Systemprozesses und des Abbildungsmodells. Somit werden nicht die Aktivitäten an sich verglichen, sondern ihre IDs, die intern als Zeichenketten verwaltet werden. Listing 5.6 zeigt die Umsetzung des Algorithmus 3.1 aus Kapitel 3, in dem die zu einem Fachprozess hinzugefügten Aktivitäten ermittelt werden. In Zeile 6 und 7 sehen wird den zweimaligen Aufruf der Get-Methode aus Listing 5.5, zuerst für die Vorversion des Fachprozesses und im Anschluss für die Folgeversion des Fachprozesses. Als Ergebnis liefert die Imlementierung eine Menge an *IDs* der zum Fachprozess hinzugefügten Aktivitäten (vgl. Zeile 9 und 15).

```
1   /**
2    * @return Die zum Fachprozess hinzugefuegten Aktivitaeten
3    */
4   public static Vector<String> getNodesAdded(){
5       //Ermittle die alte und neue Menge der Aktivitaeten
6       Vector<String> IDsOld = getOldFachprozessAktivitaetenIDs();
7       Vector<String> IDsNew = getNewFachprozessAktivitaetenIDs();
8       //Vorbereitung der Ergebnismenge
9       Vector<String> toReturn= new Vector<String>();
10      //Ermittlung der hinzugefuegten Aktivitaeten
11      for(int n=0; n<IDsNew.size();n++){
12          if(!isIn(IDsNew.get(n),IDsOld))
13              toReturn.add(IDsNew.get(n));
14      }
15      return toReturn;
16  }
```

Listing 5.6: Identifizierung hinzugefügter Modellelemente. Implementierung des Algorithmus 3.1

Listing 5.7 zeigt die Umsetzung des Algorithmus 3.12. Es wird die zum Fachprozessknoten gehörige Transformation ermittelt.

```
1  /**
2   * @param fpaID Die ID der Fachprozessaktivitaet, zu der die zugehoerige Transformation des
        Abbildungsmodells gesucht wird
3   * @return Die ID der zugehoerigen Transformation
4   */
5  public static String getTransformationAffectedToBusinessActivity(String fpaID){
6      String quellknotenID= getCorrQ(fpaID);
7      String transformationID= getTransf(quellknotenID);
8      return transformationID;
9  }
```

Listing 5.7: Ermittlung der zum Fachprozessknoten gehörigen Transformation. Implementierung des Algorithmus 3.12

Listing 5.8 zeigt die Umsetzung des Algorithmus 3.13, in dem die zu einer Transformation gehörigen Aktivitäten im Systemprozess ermittelt werden. Die ID der Transformation wurde zuvor über Listing 5.7 zu einer geänderten Fachprozessaktivität ermittelt.

```
1  /**
2   * @param transfID Die ID einer Transformation
3   * @return Die zur Transformation (der ID transfID) gehoerigen Aktivitaeten des
        Systemprozesses
4   */
5  public static Vector<String> getSNodesAffected(String transfID){
6      //Vorbereitung eines Datenspeichers fuer die gesuchten IDs
7      Vector<String> SNodesAffected= new Vector<String>();
8      //Ermittlung der Zielknoten der Transformation
9      Vector<String> zIDs= getZielknotenIDsToTransformation(transfID);
10     //Ermittle zu jedem dieser Zielknoten den zugehoerigen Knoten im Systemprozess
11     for(int i=0; i<zIDs.size();i++){
12         String snID= getCorrSnID(zIDs.get(i));
13         SNodesAffected.add(snID);
14     }
15     return SNodesAffected;
16 }
```

Listing 5.8: Ermittlung der zur Transformation gehörigen Aktivitäten im Systemprozess. Implementierung des Algorithmus 3.13

Listing 5.9 zeigt die Implementierung des Algorithmus 3.16. Es werden konkurrierende Prozessänderungen identifiziert und als eine Menge von Tupeln zurückgegeben.

```
1  /**
2   * @param BnRemovedOrModified Die Menge der entfernten oder in Attributen bearbeiteten
        Aktivitaeten des Fachprozesses
3   * @param SnRemovedOrModified Die Menge der entfernten oder in Attributen bearbeiteten
        Aktivitaeten des Systemprozesses
4   * @return Tupel von konkurrierenden Aktivitaeten beider Prozesse
5   */
6  public static Vector<Tuple<String,String>> getCompetingActivities(Vector<String>
       BnRemovedOrModified, Vector<String> SnRemovedOrModified){
7      //Bereite die zurueckzugebende Tupelmenge vor
8      Vector<Tuple<String,String>> Couples= new Vector<Tuple<String,String>>();
9      //Ermittle zu jeder Aktivitaet aus BnRemovedOrModified
10     for(int b=0; b<BnRemovedOrModified.size(); b++){
11         //die zugehoerige Transformation
12         String trID1= getTransformationAffectedToBusinessActivity(BnRemovedOrModified.get(b));
13         //Ermittle zu jeder Aktivitaet aus SnRemovedOrModified
14         for(int s=0; s<SnRemovedOrModified.size(); s++){
15             //die zugehoerige Transformation
```

```
16              String trID2= getTransformationAffectedToSystemActivity(SnRemovedOrModified.get(s)
                    );
17              //Falls diese Transformationen identisch sind
18              if(trID1.equals(trID2))
19                  //vermerke die beiden Aktivitaeten als konkurrierend
20                  Couples.add(new Tuple(trID1,trID2));
21          }
22      }
23      return Couples;
24  }
```

Listing 5.9: Ermittlung konkurrierender Prozessänderungen. Implementierung des Algorithmus 3.16

Die weiteren Implementierungen der Algorithmen gestalten sich ähnlich zu den oben gezeigten und werden hier nicht näher beschrieben.

5.4 Ergebnisaufbereitung

Da wir uns für eine eigenständige Applikation und nicht für ein Add-On zu den Modellierungswerkzeugen entschieden haben und eine grafische Darstellung nach den Betrachtungen aus Kapitel 4 eher für Add-Ons gedacht ist, liefert der Prototyp seine Berechnungsergebnisse in Form einer Liste. Da bereits XML-Dateien für den Datenimport des *Consistency managers* verwendet werden, wird die Aufgabenliste ebenfalls als XML formuliert.

Die Oberfläche der Applikation wird in Abbildung 5.3 dargestellt. Die sechs XML-Dateien werden als Input geladen ①, zuerst die drei obligatorischen ②, dann die drei fakultativen ③. Der Pfad der Dateien wird über das Dateisytem ermittelt ④ und die Konsistenzanalyse auf Benutzereingabe durchgeführt ⑤. Sind die Prozesse inkonsistent und es bedarf einer Korrektur, wird im Programm die Aufgabenliste in Form einer XML angezeigt ⑥. Diese kann auf Wunsch exportiert werden ⑦.

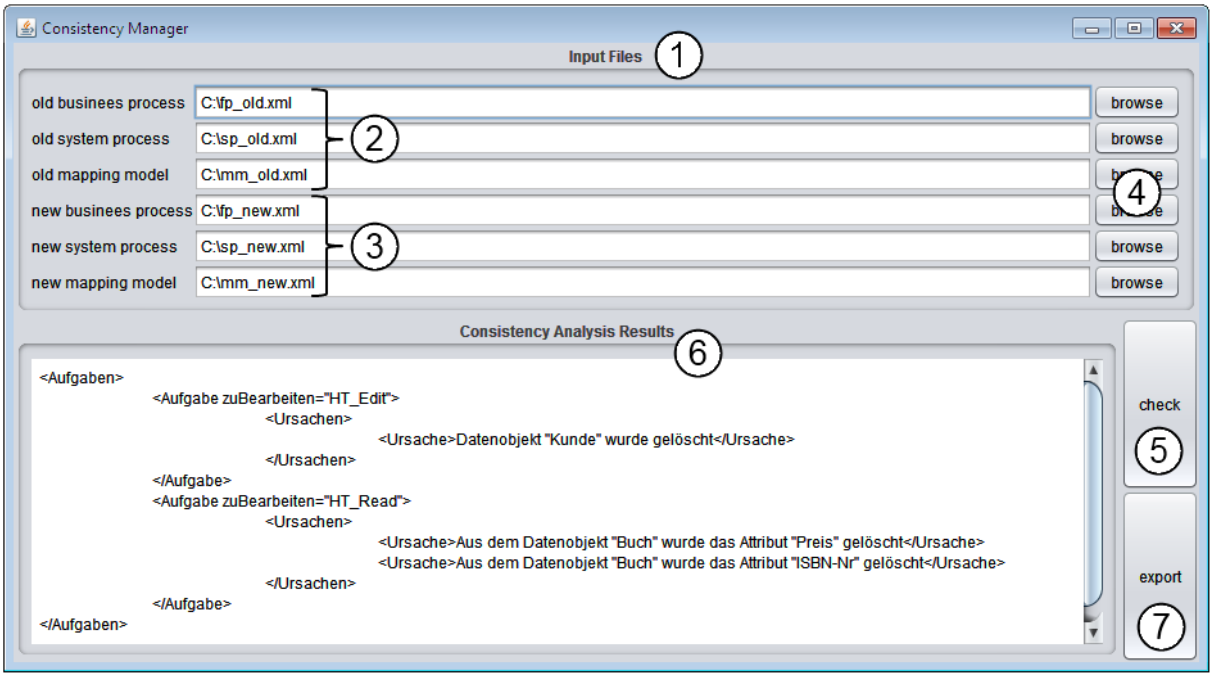

Abbildung 5.3: Die GUI des Prototyps

5.5 Zusammenfassung

In diesem Kapitel wird die Implementierung des Prototypen besprochen. Dazu werden in Abschnitt 5.1 die Grundlagen besprochen. In Abschnitt 5.2 gehen wir auf den Import von Prozessen ein. Folgend wird die Implementierung einzelner Algorithmen besprochen (vgl. Abschnitt 5.3). In Abschnitt 5.4 wird die Darstellung der berechneten Aufgabenlisten dargelegt.

6 Diskussion

In diesem Kapitel werden bestehende wissenschaftliche Ansätze zur durchgängigen Modellierung diskutiert und in Zusammenhang mit dieser Arbeit gebracht. Anschließend werden Schriften zur Erkennung von Modelländerungen besprochen.

Zahlreiche Vorgehensweisen beschreiben eine Transformation fachlicher Geschäftsprozesse in ausführbare technische Prozesse. Die Modellierungsmethodik M3 der Firma MID bietet eine spezielle Variante für die Entwicklung von Applikationen in einer serviceorientierten Architektur [DPRW08], die aus drei Ebenen besteht und damit der durchgängigen Modellierung über eine Systemmodell ähnelt (vgl. Kapitel 2). Auch andere Firmen wie IBM [Ars04] [AGA+08] oder IDS Scheer [IDS05] verwenden zur Transformation eine Zwischenebene.

Zur automatischen Generierung ausführbarer Prozesse aus fachlichen Geschäftsprozessen werden häufig Patterns eingesetzt [All07], die fachliche Prozessbestandteile detaillieren und die Überführung in technische Prozesse beschreiben. Durch eine Wiederverwendung von Patterns in verschiedenen Prozessen sollen Redundanzen vermieden werden und somit der Modellierungsaufwand reduziert werden. Der Einsatz von Patterns ist jedoch nur eingeschränkt zielführend, falls sich die Prozesse der Informationssysteme stark unterscheiden oder in zahlreichen Varianten auftreten (zum Beispiel durch Anpassungen an landesspezifische Gesetze) [Hal10]. Dadurch, dass für jede Prozessvariante eigene Patterns definiert werden müssen, entstehen individuelle Prozesstransformationen. Ein weiterer Nachteil von Patterns ist, dass sie lediglich auf die fachlichen Prozesse angewandt werden, da die Transformationen zwischen den Prozessen nur in Richtung der technischen Prozesse definiert sind. Auf Änderungen technischer Prozesse kann nur eingeschränkt reagiert werden, da die in den fachlichen Prozessen anzupassenden Elemente nicht über die Patterns ermittelt werden können.

Außerdem werden die komplexen Abhängigkeiten zu den Bestandteilen der technischen Prozesse nicht nachvollziehbar dokumentiert. Daraus ergibt sich, dass lediglich die Patterns bei Änderungen der Fachprozesse weiterverwendet werden, die techischen Prozesse jedoch neu

generiert werden müssen. Erst mit Hilfe einer durchgängigen Modellierung können technische Prozesse weiterverwendet werden.

7 Zusammenfassung

Im Rahmen dieser Arbeit werden Konzepte und Algorithmen entworfen, mit deren Hilfe die Modellierungsebenen der durchgängigen Modellierung analysiert werden. Ziel ist es, Inkonsistenzen der Modellierungsebenen durch Automatismen zu erkennen und zielführende Korrekturen anzubieten. Insbesondere werden die genauen Inkonsistenzursachen und ihre Auswirkungen untersucht.

In Kapitel 1 wird die Notwendigkeit der durchgehenden Prozessmodellierung motiviert, die sich aus der Diskrepanz zwischen Fachbereichen und IT-Bereichen ergibt. Weiterhin wird die Zielsetzung der Arbeit formuliert.

Kapitel 2 erläutert Begrifflichkeiten, welche zum Verständnis der Arbeit beitragen. In diesem Zusammenhang wird insbesondere das Systemmodell ausführlich behandelt, welches die Transformationen zwischen fachlichen und technischen Prozessen ermöglicht. Außerdem werden Anforderungen spezifiziert, welche durch die entworfenen Algorithmen umgesetzt werden.

Den Hauptteil dieser Diplomarbeit bildet Kapitel 3, in dem Konzepte und Algorithmen zur Analyse der Fachprozesse und Systemprozesse spezifiziert werden. Ausgehend von der Versionierung von Prozessen (vgl. Abschnitt 3.1). wird in Abschnitt 3.2 mit Hilfe von Algorithmen ermittelt, an welchen Aktivitäten des Fachprozesses oder des Systemprozesses Änderungen vorgenommenen wurden. In Abschnitt 3.3 wird untersucht, ob eine Prozessänderung an den korrespondierenden Prozess zu übertragen ist. Zu den geänderten Aktivitäten werden in Abschnitt 3.4 Algorithmen entworfen, die die zu kontrollierende Transformationen des Abbildungsmodells und weitere betroffene Aktivitäten des korrespondierenden Prozesses ermitteln. Mit Hilfe der Ermittlung der Änderungsauswirkungen wird in Abschnitt 3.5 gezeigt, woran konkurrierende Bearbeitungen des Fachprozesses und des Systemprozesses identifiziert werden können. In Abschitt 3.6 wird beleuchtet, woran indirekt von den Änderungen betroffene Aktivitäten identifiziert werden können. Zu den über die Algorithmen ermittelten Informationen wird in Abschnitt 3.7 geschildert, auf welche Weise die Modellierer des Fachprozesses und des Systemprozesses eine

Anpassung zwischen den Prozessen vornehmen können. Durch abschließend durchgeführte Konsistenzchecks in Abschnitt 3.8 wird auf strukturelle Konsistenz der Prozesse und innere Konsistenz des Abbildungsmodells getestet und somit eine korrekte und vollständige Anpassung zwischen dem Fachprozess und dem Systemprozesses bestätigt.

In Kapitel 4 wird untersucht, wie die Modellierer durch eine Aufgabenliste in der Inkonsistenzbehebung unterstützt werden können. Ziel ist es, dass die Modellierer durch optimale graphische Darstellungen der vorzunehmenden Aufgaben in der Inkonsistenzbehebung unterstützt werden. Dabei wird auf den Unterschied zwischen sequenziellen Listen und visuellen Darstellungen eingegangen und die Bedeutung der Sortierkriterien erklärt.

Um die Algorithmen auf ihre Realisierbarkeit zu überprüfen, werden sie in Kapitel 5 in einem Prototyp realisiert. Zunächst wird die Architektur vorgestellt. Folgend wird auf die Austauschformate eingegangen, bevor zu ausgewählten Algorithmen die Implementierung aufgezeigt wird. Das resultierende Ergebnis ermöglicht, Änderungen zwischen Fachprozessen und deren Implementierung transparent zu halten.

Abschließend werden in Kapitel 6 bestehende Ansätze diskutiert und gegenseitig abgegrenzt. Dabei liegt der Schwerpunkt auf Ansätzen, welche eine durchgängige Modellierung diskutieren, sowie auf Ansätzen, welche sich mit Transformationen zwischen Modellierungssprachen auseinandersetzen.

Literaturverzeichnis

[AGA+08] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley. SOMA: A method for developing service-oriented solutions. *IBM Systems Journal*, 47(3):377–396, 2008.

[All07] T. Allweyer. Erzeugung detaillierter und ausführbarer Geschäftsprozessmodelle durch Modell-zu-Modell-Transformationen. *EPK 2007*, page 23, 2007.

[Ars04] A. Arsanjani. Service-oriented modeling and architecture. *IBM developer works*, 2004.

[BBP09] S. Buchwald, T. Bauer, and R. Pryss. IT-Infrastrukturen für flexible, service-orientierte Anwendungen - ein Rahmenwerk zur Bewertung. In *BTW*, volume 144 of *LNI*, pages 526–543. GI, 2009.

[BBR09] S. Buchwald, T. Bauer, and M. Reichert. Durchgängige Modellierung von Geschäftsprozessen durch Einführung eines Abbildungsmodells: Ansätze, Konzepte, Notationen. Technical report, 2009.

[BBR10] S. Buchwald, T. Bauer, and M. Reichert. Durchgängige Modellierung von Geschäftsprozessen in einer Service-orientierten Architektur. In *Modellierung'10*, number 161 in Lecture Notes in Informatics (LNI), pages 203–211. Koellen-Verlag, March 2010.

[BBR11] S. Buchwald, T. Bauer, and M. Reichert. Bridging the gap between business process models and service composition specifications. In *Int'l Handbook on Service Life Cycle Tools and Technologies: Methods, Trends and Advances*. UNSPECIFIED. (Accepted for Publication), 2011.

[Erl05] T. Erl. *Service-oriented architecture: concepts, technology, and design*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

[Hal10] A. Hallerbach. *Management von Prozessvarianten*. PhD thesis, Universität Ulm. Fakultät für Ingenieurwissenschaften und Informatik, 2010.

[IDS05] IDS Scheer. *Business Process Management: ARIS Value Engineering-Concept*. Whitepaper, 2005.

[Jos07] N.M. Josuttis. *Soa in practice: the art of distributed system Design*. O'Reilly Media, Inc., 2007.

[Obj09] Object Management Group. Business Process Modeling and Notation (BPMN) Specification 2.0. Technical report, 2009.

[Rec10] S. Rechtenbach. Durchgängige Modellierung: Vorgehen und Funktionalität im GPM-Tool, 2010.

[Sch01] A.W. Scheer. *ARIS-Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, 2001.

[Tie11] J. Tiedeken. Konzeption und Realisierung eines logisch zentralen SOA-Repositories: Unterstützung von Impact-Analysen mittels durchgängiger Modellierung. Diplomarbeit, Universität Ulm, 2011.

[VdPG05] G. Van de Putte and L. Gavin. *Technical Overview of WebSphere Process Server and WebSphere Integration Developer*. IBM, 2005.

[Web07] Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, April 2007.

[Wes07] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York Inc, 2007.