Diplomarbeit an der Universität Ulm
- Fakultät für Informatik -
Abteilung Datenbanken und Informationssysteme

Prozessdatenintegration und -transformation für die systemübergreifende Visualisierung von Arbeitsabläufen

vorgelegt von: Tiberius Mihalca

Erstgutachter:	Prof. Dr. Peter Dadam
Zweitgutachter:	Dr. Manfred Reichert
Betreuer:	Ralph Bobrik

Ulm, November 2005

Kurzfassung

Die integrierte Visualisierung systemübergreifender Prozesse wird für Unternehmen immer wichtiger. Voraussetzung ist, dass die Informationen zu Prozessen von allen Systemen einheitlich und in integrierter Form vorliegen. Aufgrund der Verteilung von Prozessdaten auf mehrere Systeme ergeben sich zumeist syntaktische und semantische Heterogenitäten, weshalb sich für die Prozessdatenintegration zahlreiche Herausforderungen stellen.

Diese Arbeit identifiziert und kategorisiert auftretende Probleme und zeigt Lösungsansätze auf. Darauf basierend wird eine Architektur für eine Komponente entwickelt, die die Lösungen umsetzt und es wird an einem konkreten Beispiel die Transformation eines Prozessmodells demonstriert.

Ausgangspunkt für die Zusammenführung der Prozessdaten, an der sich die Problemidentifikation und -kategorisierung orientiert, sind heterogene Datenquellen. Deren Daten müssen zunächst angepasst werden, bevor die Prozesse der einzelnen Systeme auf ein vorgegebenes Metamodell abgebildet und die so entstandenen Prozessfragmente in einen Gesamtprozess eingegliedert werden. Nachdem das statische Prozessmodell erstellt ist, müssen noch Laufzeitdaten integriert werden. Dabei ist zu beachten, dass sich durch die vorangegangenen Schritte das Prozessmodell geändert haben kann.

Nach Identifizierung der wesentlichen Probleme und Aufgaben wird eine Architektur entwickelt, die berücksichtigt, dass nicht alle Systeme gleich viele Daten bereitstellen und somit mehrere Vorgehensweisen für die Abbildung der Prozessfragmente auf das vorgegebene Metamodell nötig sind.

Zum Schluss werden als Beispiel zwei Modelle des ARIS-Toolsets auf ein vorgegebenes Metamodell abgebildet und es wird gezeigt, wie man die Modelle automatisch mit Hilfe impliziter Informationen aus dem Prozessmodell semantisch anreichern kann.

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Der verschärfte Wettbewerbsdruck auf den Märkten und der damit verbundene Zwang zu Einsparungen haben zu strategischen Veränderungen in den Unternehmen geführt. Eine dieser Strategien ist Outsourcing. Um sich auf die Kerngeschäfte konzentrieren zu können, reduzieren viele Unternehmen ihre Fertigungstiefe und produzieren Vorerzeugnisse nicht mehr selber. Stattdessen beziehen sie diese bei Zulieferern, die auf diese Produkte spezialisiert sind und sie aufgrund höherer Stückzahlen günstiger herstellen können.

Eine andere Strategie sind Fusionen. Anstatt sich mit Konkurrenten zu bekämpfen, schließen sich konkurrierende Unternehmen zusammen. So erlangt man mehr Marktanteile und kann die Vorteile der Massenproduktion nutzen.

Durch diese Entwicklungen besteht der gesamte Entstehungsprozess eines Produktes nicht mehr aus einem einzigen Prozess eines Unternehmens, sondern aus Teilprozessen, die auf mehrere Unternehmen und Abteilungen verteilt sind. Diese Teilprozesse wiederum können ein breites Spektrum an Prozessarten abdecken, wie es beispielsweise in der Automobilindustrie der Fall ist. Hier gibt es Prozesse u.a. für die Fahrzeugentwicklung, für Änderungsanträge (Change Management) oder für das Supply Chain Management. Diese haben unterschiedliche Komplexitätsgrade und Dauer und betreffen neben mehreren Unternehmen oder Abteilungen oft auch mehrere organisatorische Gruppen (z.B. Manager, Ingenieure, Techniker).

Um einen Überblick über den gesamten Prozess zu erhalten und somit den aktuellen Stand erfahren, Optimierungen vornehmen oder in Ausnahmefällen schneller reagieren zu können, ist eine einheitliche und gemeinsame Visualisierung aller Teilprozesse wünschenswert. Das ist allerdings in dem angesprochenen Umfeld durchaus problematisch, da gerade in der Automobilindustrie die Informationssysteme sehr komplex sind und oft aus verteilten, heterogenen Anwendungssystemen bestehen [BRB05]. Zudem sind diese oft über Jahrzehnte gewachsen, was zu enormen technologischen Unterschieden geführt hat.

Durch die Vielfalt der Systeme und der daraus resultierenden unterschiedlichen Beschreibung und Speicherung der Daten entstehen verschiedene Heterogenitäten auf mehreren Ebenen.

Will man eine integrierte Visualisierung der Teilprozesse erreichen, so müssen die angesprochenen Heterogenitäten beseitigt und die zum Prozess gehörenden Daten aus den

verschiedenen Quellen zusammengeführt werden. Diese Daten bilden dann eine Basis für beliebige Visualisierungen.

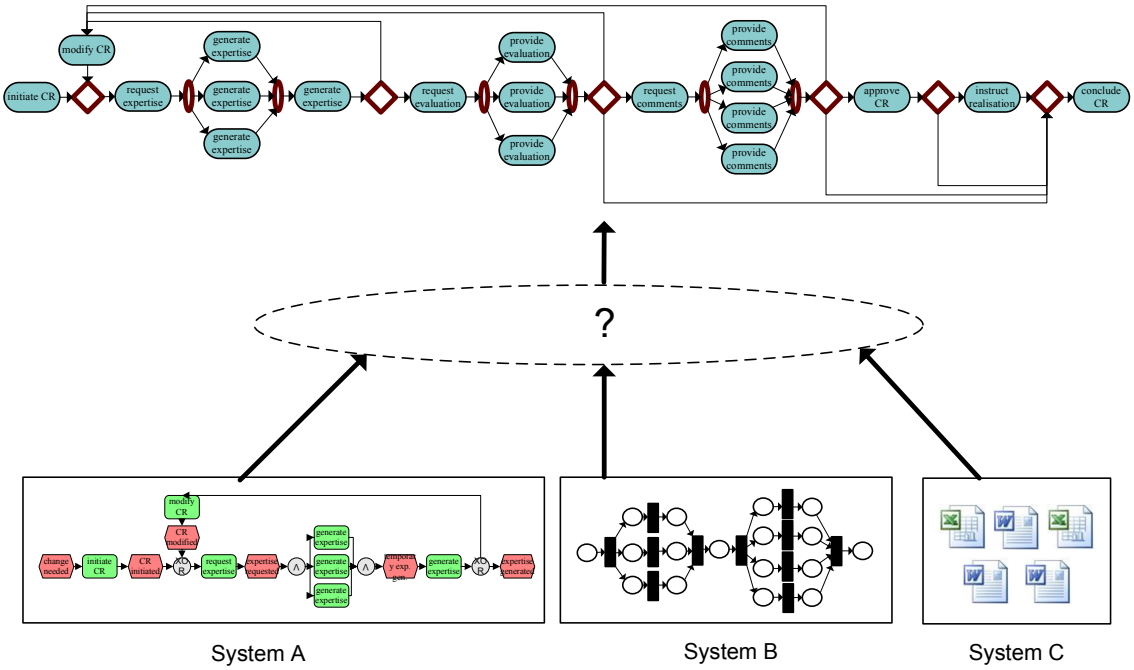


Abbildung 1.1 Visualisierung der Problemstellung

Die angesprochenen Punkte sind in Abbildung 1.1 angedeutet. Diese zeigt den Ablauf der Bearbeitung eines Änderungsantrages in der Automobilindustrie, im Folgenden *ChangeManagement*-Prozess genannt. Eine genaue Abbildung dieses Prozesses findet sich im Anhang A.

Man kann sehen, dass der Gesamtprozess aus mehreren Teilprozessen besteht, die auf unterschiedlichen Systemen ablaufen. Ob die Systeme zu unterschiedlichen Unternehmen oder Abteilungen oder zu einer Abteilung gehören, ist dabei unerheblich. Entscheidend ist, dass die Teilprozesse, auch Prozessfragmente genannt, von den einzelnen Systemen unterschiedlich abgebildet und gespeichert werden. So werden die Prozessfragmente von den ersten beiden Systemen in unterschiedlichen Prozess-Beschreibungssprachen beschrieben, während das letzte Prozessfragment nur aus den bearbeiteten Daten des Systems besteht und keine Prozessinformationen enthält.

Um solche unterschiedliche Prozessfragmente einheitlich auf einen Gesamtprozess abzubilden, müssen zunächst die Daten der einzelnen Systeme auf ein gemeinsames Modell abgebildet werden. Dafür müssen Prozessdaten transformiert werden und in den Gesamtprozess integriert werden. Dieses Vorgehen wird im Folgenden Mapping genannt.

1.2 Ziele und Aufgabenstellung der Arbeit

In dieser Arbeit werden bei der Transformation und Integration von Prozessdaten auftretende Konflikte und Problemfälle identifiziert und systematisch klassifiziert. Anschließend werden für ausgewählte Problemfälle beispielhaft Konzepte und Architekturvarianten entwickelt, um Konflikte beim Mapping möglichst automatisch aufzulösen.

Für das Mapping existieren unterschiedliche Realisierungsvarianten, die in dieser Arbeit ebenfalls recherchiert werden. Eine Fragestellung hierbei ist, welche Informationen für das Mapping benötigt werden und ob bzw. wie diese zur Verfügung gestellt werden können.

Nach den vorhergehenden Überlegungen und Konzepten wird zudem an einem Beispiel die Umsetzung demonstriert. Dabei wird untersucht, inwieweit beim Mapping unterschiedlicher Metamodelle das Modell, das weniger explizite Informationen enthält, automatisch aus Eigenschaften des Prozessgraphen semantisch aufgewertet werden kann.

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich wie folgt: in Kapitel 2 werden Grundlagen vorgestellt, die für das Verständnis der Arbeit wichtig sind. Danach werden zunächst Probleme identifiziert, die beim Mapping von Prozessmodellen auftreten können. Da diese Probleme in mehreren Bereichen und in unterschiedlicher Form auftreten können, werden sie kategorisiert und getrennt betrachtet. Die Kategorisierung der Probleme richtet sich nach den Schritten, die für das Mapping notwendig sind. Um die Problemkategorisierung und somit auch den Aufbau der Arbeit verständlicher zu machen, zeigt Abbildung 1.2 die benötigten Schritte.

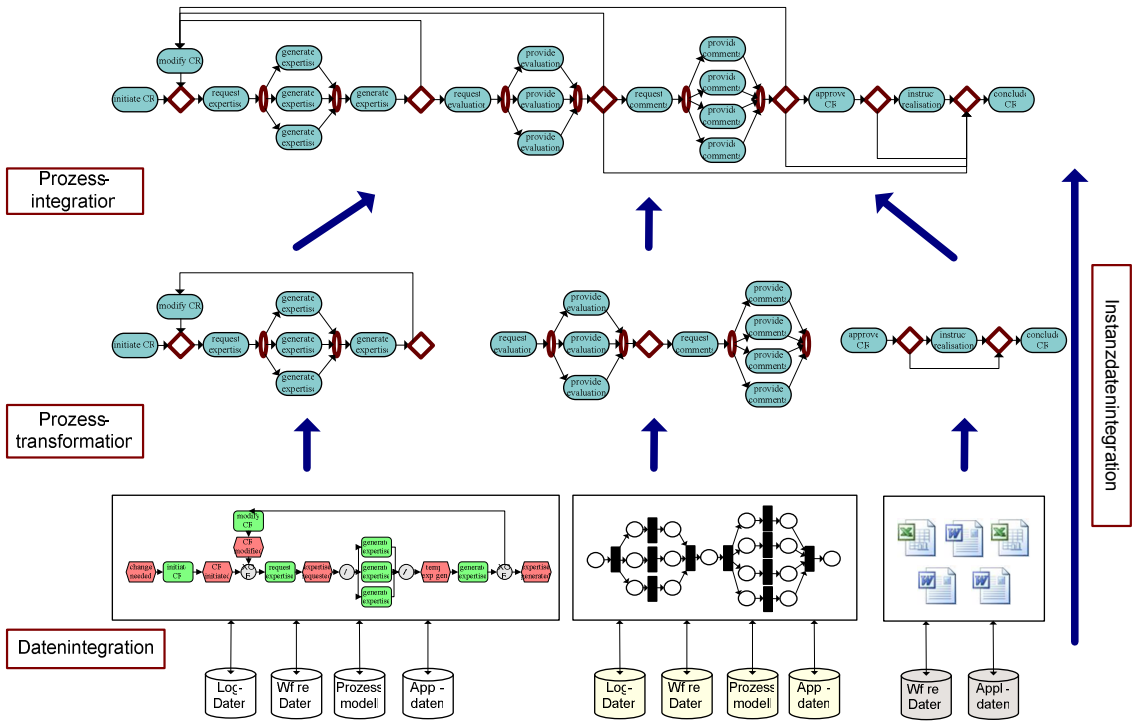


Abbildung 1.2 Kategorisierung der Probleme beim Mapping

Die Probleme beginnen, wenn Daten heterogener Systeme, die in unterschiedlichen Formaten vorliegen können, gelesen werden müssen. Diese Daten auf eine einheitliche Datenbasis zu bringen, nennt sich Datenintegration. Nach der Integration der Daten liegen die Prozessfragmente der einzelnen Systeme zumeist in unterschiedlicher Form vor, da sie von den Systemen unterschiedlich dargestellt werden. Somit müssen diese transformiert und auf ein einheitliches Metamodell abgebildet werden. Das passiert bei der Prozesstransformation. Nachdem die Prozessfragmente in eine einheitliche Form gebracht sind, werden sie bei der Prozessintegration zu einem Gesamtprozess integriert. Abschließend, wenn das statische Prozessmodell vorliegt, werden bei der Instanzdatenintegration noch die dynamischen Daten der laufenden Prozessinstanzen integriert.

Die vier Problemkategorien sind also Datenintegration, Prozesstransformation, Prozessintegration und Instanzdatenintegration. Diese werden in den Kapiteln 3-6 in der oben angegebenen Reihenfolge behandelt, wobei die Probleme der Datenintegration nur kurz anhand von Systemen, die diese bereits gelöst haben, betrachtet werden.

Nach der Identifikation aller Probleme wird in Kapitel 7 eine Architektur für eine Mapping-Komponente entwickelt, mit deren Hilfe die identifizierten Probleme gelöst werden können. Kapitel 8 demonstriert beispielhaft die Transformation eines Prozessmodells auf ein anderes Metamodell anhand ausgewählter Probleme. Abschließend, in Kapitel 9 und 10, werden noch Ansätze verwandter Arbeiten und mögliche weiterreichende Lösungen diskutiert.

2 Grundlagen

Dieses Kapitel stellt einige Grundlagen vor, die für das Verständnis der Arbeit wichtig sind. Dabei werden zunächst Systeme, auf denen die Prozessfragmente ausgeführt werden können, eingeführt und klassifiziert, bevor auf mögliche Mappingvarianten eingegangen wird und diese bewertet werden.

2.1 Betrachtete Systeme

Unter dem Begriff System sind in dieser Arbeit alle Arten von Unternehmenssoftware zu verstehen. Damit sind sowohl einfache Anwendungen wie beispielsweise MS Word, als auch komplexe Datenbanksysteme und modernste Workflow Management Systeme gemeint.

Diese Systeme haben unterschiedliche Einsatzzwecke und sind unter Umständen über mehrere Jahrzehnte gewachsen. Dadurch basieren sie auf unterschiedlichen Technologien und Architekturen. Aufgrund der Vielzahl der Systeme, ist es für die kommenden Diskussionen sinnlos sie nach ihrem Zweck oder ihrer Technologie zu unterscheiden. Deswegen werden wir uns im Folgenden nur auf die für diese Arbeit relevanten Teile der Systeme konzentrieren. Das sind die von den Systemen bereitgestellten Daten. Diese werden in Abschnitt 2.1.1 vorgestellt. In Abschnitt 2.1.2 werden wir die Systeme anhand dieser Daten klassifizieren und somit durch diese Abstraktion die Vielzahl der Systeme reduzieren. In Abschnitt 2.1.3 werden wir zwei System-Beispiele angeben, die im Kontext dieser Arbeit sehr gegensätzlich sind. Eines davon erzeugt besonders viele Probleme, das andere dagegen eignet sich optimal für die Abbildung auf ein gemeinsames Modell.

2.1.1 Bereitgestellte Daten der Systeme

Bevor wir die Systeme anhand ihrer bereitgestellten Daten klassifizieren, werden diese zunächst erklärt. Von den betrachteten Daten sind Applikations-, Log- und Workflow relevante Daten Laufzeitdaten (Instanzdaten). Das sind Daten, die erst zur Laufzeit des Prozesses entstehen. Modell- und Metamodelldaten dagegen gibt es schon bei der Modellierung des Prozesses, falls dies vom jeweiligen System unterstützt wird.

2.1.1.1 Applikationsdaten

Applikationsdaten können auf Applikationssystemen erzeugt und von diesen auch direkt manipuliert werden. Sie sind das eigentliche Produkt bei der Arbeit mit solchen Systemen.

Beispiele für Applikationsdaten sind:

- CAD-Bild
- Word-Dokument
- Powerpoint-Präsentation

2.1.1.2 Log-Daten

Log-Daten sind Daten, die im Verlauf der Bearbeitung einer Aufgabe oder eines ganzen Prozesses protokolliert werden.

Im Prozesskontext stellen sie eine historische Aufzeichnung des Fortschritts einer Prozessinstanz vom Anfang bis zum Ende dar [WfMC99a]. Bei einfachen Applikationen, ohne Prozesskontext, dienen sie ebenfalls dazu, gewisse Vorgänge zu erfassen, um sie später beispielsweise für Analyse- oder Kontrollzwecke zu nutzen.

Mögliche Log-Daten sind:

- Zeiten:
  - Anfang/Ende einer Aufgabe oder eines Prozesses
  - Dauer einer Aufgabe
- Bearbeiter der Aufgabe
- Typ des Prozesses oder der Aktivität
- Verantwortlicher des Prozesses oder der Aufgabe

2.1.1.3 Workflow relevante Daten

Daten, die vom Workflow Management System für Zustandsübergänge von Workflowinstanzen, Weiterschaltungsbedingungen oder Benutzerzuordnungen benutzt werden. Diese Daten können sowohl von Workflow-Anwendungen als auch von der Workflow-Engine manipuliert werden [WfMC99a].

Mögliche Workflow relevante Daten sind:

- Daten für die Weglenkung
- Schleifenzähler
- Bearbeiter einer bestimmten Aufgabe (z.B. „Workflow-Initiator“)

2.1.1.4 Modelldaten

Modelldaten sind alle Daten, die zur Beschreibung eines Prozessmodells dienen.

Ein Prozessmodell ist eine abstrakte Abbildung eines Geschäftsprozesses nach einem bestimmten Gesichtspunkt (z.B. Kontrollfluss), mit dem Ziel eine Ordnung zu schaffen, die greifbar, analysierbar und in ihrer Gesamtheit optimierbar wird [WfMC99a].

Mögliche Modelldaten sind:

- Struktur des Modells: z.B. „Aktivität A folgt direkt auf Aktivität B“ oder „A und B können parallel ausgeführt werden“.
- Bearbeiterzuordnungen: z.B. Aktivität A muss von Bearbeiter X ausgeführt werden oder B muss vom gleichen Bearbeiter ausgeführt werden, der A ausgeführt hat.

2.1.1.5 Metamodell

Ein Metamodell ist ein Modell, das vorgibt wie andere Modelle gebaut werden dürfen und wie diese zu interpretieren sind. D.h. es beschreibt die Syntax und die Semantik der Modelle, die auf diesem Metamodell basieren [WfMC99a].

Mögliche Metamodell-Daten sind:

- Informationen über Schleifen: „Sind Schleifen erlaubt oder nicht?“
- Konnektorregel: „Welche Elemente dürfen mit einem Konnektor verbunden werden. Darf man beispielsweise nur Ereignisse und Funktionen oder auch zwei Funktionen miteinander verbinden?“
- Anzahl Start-/Endknoten.

2.1.2 Systemklassifikation

Wie bereits erwähnt, ist bei der Vielzahl der Systeme eine Klassifikation anhand der bereitgestellten Daten enorm wichtig zur Reduzierung der Komplexität und somit zur einfacheren Betrachtung vieler Sachverhalte. Würde man allerdings bei der Klassifikation alle möglichen Permutationen von den beschriebenen Daten berücksichtigen, dann ergeben sich $\sum_{1}^{5}\binom{5}{n} = 31$ Klassen, die weitere Diskussionen kaum erleichtern würden. Doch wenn man die Daten genau betrachtet, stellt man fest, dass gar nicht alle Kombinationen möglich bzw. sinnvoll sind. Beschränkt man sich auf die sinnvollen Kombinationen, dann reduziert sich die Anzahl der Klassen enorm. Um solche Kombinationen zu ermitteln, wird zunächst untersucht, welche Daten notwendigerweise vorhanden sein müssen (s. Abschnitt 2.1.2.1). Danach wird, auf Basis dieser Erkenntnisse, in Abschnitt 2.1.2.2 die eigentliche Klassifikation vorgenommen.

2.1.2.1 Notwendigkeit der Daten

Applikationsdaten sind unerlässlich und in irgendeiner Form immer vorhanden, wenn ein Prozess oder eine Aufgabe am System ausgeführt wird. Wären keine Applikationsdaten vorhanden, dann gäbe es auch kein Arbeitsergebnis.

Workflow relevante Daten sind in den betrieblichen Systemen nicht notwendigerweise vorhanden. Sie werden allerdings für Entscheidungen oder für die Steuerung von Aufgaben in Workflow Management Systemen benötigt. Dabei kann es sich um eigentliche Applikationsdaten handeln, die für die Steuerung bzw. für Entscheidungen benutzt werden.

Log-Daten sind für die Ausführung eines Prozesses oder einer Aufgabe nicht unbedingt notwendig, werden aber normalerweise in betriebswirtschaftlichen Systemen, vor allem aus organisatorischen Gründen, aufgezeichnet und bereitgestellt.

Modelldaten sind nicht immer vorhanden, sind aber Voraussetzung für einen vordefinierten Ablauf von Prozessen. Theoretisch kann aber eine Aufgabe auch ohne diese Daten bearbeitet werden.

Daten zum Metamodell sind in den einzelnen Systemen nicht unbedingt notwendig. Man braucht sie aber, wenn beispielsweise neue Prozesse modelliert oder Korrektheitsüberprüfungen ausgeführt werden sollen. Gibt es kein Metamodell und ist die Modellierung deswegen nicht möglich, dann bieten einige Systeme eine Menge vordefinierter Prozesse an, aus der man den benötigten Prozess auswählen kann.

Werden Daten zum Metamodell aus anderen Gründen benötigt, dann können diese an der benötigten Stelle auch von außen eingebracht werden. Ein Metamodell eines bestimmten Typs muss nämlich bei allen Systemen gleich sein. Somit müssen die Systeme diese Daten nicht unbedingt bereitstellen. Es reicht, wenn man weiß, auf welchem Metamodell die Prozessmodelle basieren. Aus diesem Grund werden die Metamodell-Daten bei der Klassifikation nicht mehr betrachtet.

2.1.2.2 Klassifikation

Klassifiziert man nun die Systeme nach den bereitgestellten Daten, aber unter Berücksichtigung ihrer Notwendigkeit, dann kann man den Großteil der 31 möglichen Kombinationen eliminieren. So kommen wir auf drei sinnvolle Klassen, die im Folgenden beschrieben werden.

- Klasse 1: Systeme dieser Klasse stellen Applikations- und Workflow relevante Daten bereit. Die Applikationsdaten müssen von jedem System in irgendeiner Form bereitgestellt werden und da die Workflow relevanten Daten normalerweise auf diesen basieren, werden sie von dieser Klasse ebenfalls bereitgestellt.

- Klasse 2: Neben den Daten von Klasse-1-Systemen, stellen diese Systeme zusätzlich Log-Daten bereit. Das ist für die meisten Systeme in Unternehmen typisch, denn so wird die Protokollierung, Kontrolle oder Nachvollziehbarkeit von Ereignissen oder Aktionen erleichtert bzw. ermöglicht. Die Art der Log-Daten kann sich allerdings von System zu System unterscheiden.
- Klasse 3: Neben den Daten von Klasse-2-Systemen, stellen diese Systeme zusätzlich Modelldaten bereit. Dadurch wird der Verlauf des Prozesses offen gelegt und dessen Visualisierung somit vereinfacht. Systeme dieser Klassen stellen alle Daten bereit, die für die Abbildung ihrer Prozesse auf ein gemeinsames Modell nötig sind.

Abbildung 2.1 zeigt eine Übersicht der drei Klassen inklusive der, durch die Systeme, bereitgestellten Daten.

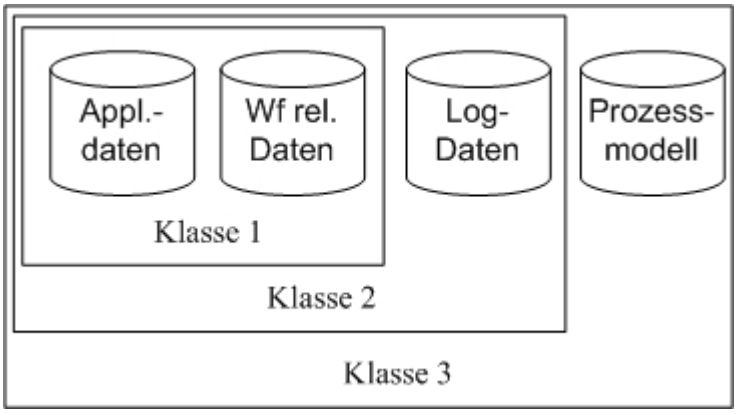


Abbildung 2.1 Klassenübersicht mit zugehörigen Daten

2.1.3 Systembeispiele

In diesem Abschnitt werden beispielhaft zwei Arten von Systemen genannt, die sich vor allem im Kontext dieser Arbeit sehr stark unterscheiden. Das ist zum einen ein Beispiel für Systeme der Klasse 1, die bei der Abbildung auf ein anderes Modell besonders viele Probleme bereiten können und zum anderen ein Beispiel für Systeme der Klasse 3, deren Datenbasis entschieden weniger Probleme verursacht.

2.1.3.1 Workflow Management Systeme

Workflow Management Systeme (WfMSe) sind typische Klasse-3-Systeme. Sie ermöglichen es, Geschäftsprozesse zu definieren, zu erstellen, ihre Ausführung zu überwachen und zu regeln. Der große Vorteil im Vergleich zu anderen Systemen ist, dass WfMSe Prozess- und Applikationslogik voneinander trennen. Das bedeutet, dass der Ablauf des Prozesses nicht in den Applikationen fest codiert ist, sondern im WfMS durch ein Pro-

zessmodell repräsentiert wird. So ein Prozessmodell kann beispielsweise ein Graph oder ein Regelwerk sein. Durch diese Trennung ist es möglich, den Ablauf des Prozesses auf einem hohem Abstraktionsniveau zu ändern, ohne dafür in den Applikationscode eingreifen zu müssen.

Um betrieblichen Anforderungen gerecht zu werden, protokollieren WfMSe relevante Ereignisse, wie z.B. Bearbeiter einer Aufgabe, Zeitpunkt und Dauer der Bearbeitung usw., und stellen diese Daten zu Analyse- und Kontrollzwecken zur Verfügung.

Die benötigten Applikationsdaten werden nicht direkt von oder auf dem WfMS bearbeitet. Dieses ruft nur die benötigten Applikationen zur richtigen Zeit, am richtigen Ort und für den richtigen Bearbeiter auf und liefert die nötigen Informationen für die Bearbeitung der Aufgaben. Die so erzeugten Applikationsdaten werden oft auch für die Steuerung des Prozesses benutzt.

Somit stellen WfMSe alle von uns benötigten Daten bereit und erleichtern dadurch die einheitliche Abbildung von Prozessen.

2.1.3.2 Legacy Systeme

Legacy Systeme (englisch *legacy* für Altlast, Erbe) sind historisch gewachsene Applikationen im Bereich der Unternehmenssoftware. Diese Applikationen sind technisch meistens veraltet, können aber oft aus organisatorischen Gründen nicht ohne Probleme ausgetauscht werden. Sie zeichnen sich durch mangelnde Dokumentation und fehlende oder proprietäre Schnittstellen aus. Oft stellen sie nur die Applikationsdaten in einem eigenen, speziellen Format bereit und sind somit typische Klasse-1-Systeme. Ihre integrierte Abbildung in einen systemübergreifenden Prozess bereitet enorme Schwierigkeiten.

2.2 Mapping von Prozessmodellen

Unter Mapping versteht man in diesem Zusammenhang die Abbildung von einem Prozessmodell auf ein anderes. Das Modell, das abgebildet werden soll, ist das Quellmodell, das andere auf das die Abbildung erfolgen soll, ist das Zielmodell. Je nach Metamodell, auf dem die einzelnen Prozessmodelle basieren und je nach bereitgestellten Daten durch die Systeme, auf denen die Prozesse ablaufen, gibt es beim Mapping unterschiedliche Möglichkeiten, Probleme und weitere Aspekte, die zu beachten sind. Diese werden im Folgenden ebenfalls aufgezeigt.

2.2.1 Möglichkeiten für das Mapping

Optimalerweise sollte das Mapping automatisch oder zumindest semi-automatisch ablaufen. Hierfür gibt es mehrere Varianten, die von einigen Faktoren abhängen. Je nach

Systemklasse hat man unterschiedliche Voraussetzungen. Im schlimmsten Fall muss das Mapping manuell vorgenommen werden.

Die unterschiedlichen Varianten für das Mapping sind in Abbildung 2.2 visualisiert und werden im Folgenden detailliert beschrieben.

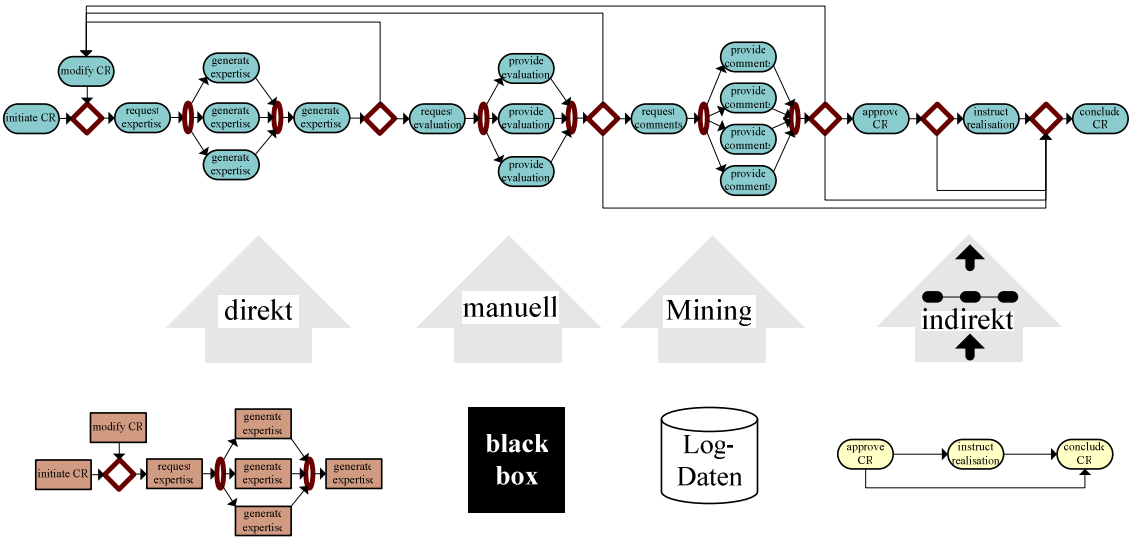


Abbildung 2.2 Mappingvarianten

2.2.1.1 Indirektes Mapping

Beim indirekten Mapping werden die Daten der verschiedenen Systeme zunächst auf ein gemeinsames Zwischenmodell abgebildet, das als Grundlage für die weiteren Visualisierungsschritte dient. Dieses Vorgehen läuft nach dem ETL-Prozess ab, der aus dem Bereich der *Data-Warehouse-Systeme* (s. Abschnitt 3.2) bekannt ist. ETL ist die Abkürzung für Extract, Transform, Load und steht für die drei gleichnamigen Phasen des Prozesses.

In der Extraktionsphase werden die relevanten Daten aus den Quellen ausgewählt und in einen Arbeitsbereich übertragen, wo sie weiter bearbeitet werden können. Die Zeitpunkte der Extraktion hängen bei Data-Warehouse-Systemen entscheidend von der Semantik der Daten bzw. von den auf diesen Daten durchzuführenden Auswertungen ab. In unserem Fall ist es ebenfalls entscheidend, wie aktuell der visualisierte Prozess sein muss. Abhängig davon kommen prinzipiell vier Strategien in Frage:

- *periodische Extraktion*, wobei die Periodendauer von der geforderten Mindestqualität der Daten bzw. von der Aktualität des visualisierten Prozesses abhängt
- *Extraktion auf Anfrage*

- *ereignisgesteuerte Extraktion*, z.B. bei Erreichen einer a priori festgelegten Anzahl von Änderungen
- *sofortige Extraktion* bei Änderungen.

In der Transformationsphase werden die heterogenen Daten aus den unterschiedlichen Quellen, die sich nun in einem gemeinsamen Arbeitsbereich befinden, in ein gemeinsames Format gebracht. Dies betrifft sowohl syntaktische und semantische Aspekte der Datenbasis, als auch die strukturelle Anpassung der Prozesse, die u.U. auf unterschiedlichen Metamodellen basieren [BaGu01].

Nach Abschluss der Transformationsphase werden in der Ladephase die transformierten Daten aus dem Arbeitsbereich in einen anderen Bereich geladen, wo sie weiter bearbeitet werden [BaGu01].

2.2.1.2 Direktes Mapping

Beim direkten Mapping wird das Quellmodell, ohne Zwischenmodell, direkt auf das Zielmodell abgebildet. Das funktioniert prinzipiell wie das Mapping vom Quell- auf das Zwischenmodell, nur mit dem Unterschied, dass das Mapping damit abgeschlossen ist. Auf den ersten Blick erscheint diese Variante einfacher. Doch bei näherer Betrachtung stellt man fest, dass sie einen gravierenden Nachteil gegenüber dem indirekten Mapping hat.

Unser Ziel ist es einer Visualisierungskomponente eine einheitliche Darstellung eines systemübergreifenden Prozesses bereitzustellen, die als Basis für die Visualisierung dienen soll. Die Visualisierungskomponente soll den Prozess in unterschiedlichen Formen, je nach Anwender oder Bedarf, anzeigen. Das bedeutet, dass es mehrere Zielmodelle geben muss. Will man von allen Quellmodellen auf alle Zielmodelle abbilden können, so ergeben sich bei n Ziel- und n Quellmodellen $n*n$ Mappings. Das ist eine Komplexität von $O(n^2)$. Beim Mapping über ein Zwischenmodell sind es dagegen nur $n+n$ Mappings, also eine Komplexität von $O(n)$. Deswegen wird das direkte Mapping hier nicht benutzt und auch nicht genauer behandelt.

An dieser Stelle sei aber noch erwähnt, dass beide bisher vorgestellten Mappingvarianten, sowohl die Modelldaten als auch alle Instanzdaten benötigen. Sind nicht alle Daten gegeben, dann muss man die folgenden Varianten benutzen.

2.2.1.3 Process Mining

Process Mining ermöglicht es, aus gesammelten Log-Daten Prozessinformationen zu gewinnen. Dazu sind lediglich die eindeutige Identifikation der Prozess-Instanzen, der Aktivitäten und die zeitliche Abfolge der Aktivitäten nötig [AaWe04]. Für die Abbildung der Organisationsstrukturen werden zusätzlich der Ausführungszeitpunkt der Aktivitäten sowie der Bearbeiter benötigt [MWA04]. Es gibt allerdings auch noch Prob-

lemfälle, wie z.B. einelementige Schleifen, die automatisch nicht erzeugt werden können [AaWe04]. An solchen Stellen ist ein manueller Eingriff notwendig.

Process Mining kann sinnvoll eingesetzt werden, wenn die zu integrierenden Systeme keine Daten zum Prozessmodell, aber die oben erwähnten nötigen Log-Daten bereitstellen. Werden die nötigen Log-Daten von den jeweiligen Systemen nicht bereitgestellt, so muss mit Hilfe von Werkzeugen versucht werden diese zu sammeln. Solche Werkzeuge sind beispielsweise *Event-Listener*, die Ereignisse in der Datenbasis beobachten. Diese Ereignisse kann man entweder protokollieren oder man kann aus ihnen Rückschlüsse ziehen, aus denen man die nötigen Log-Daten generieren kann. Man nennt diese Werkzeuge auch *Wrapper*, weil sie die Systeme kapseln bzw. umhüllen und so alle ihre Aktivitäten beobachten.

Mit den Process Mining-Werkzeugen kann man mittlerweile gute Ergebnisse erzielen und für die Log-Daten einiger Systeme, wie z.B. Staffware gibt es bereits Plug-Ins, die die Log-Daten in die benötigte Form bringen, wodurch keine Vorarbeit mit diesen Daten notwendig ist. Die Werkzeuge erzeugen dann automatisch ein Prozessmodell. Basiert das erzeugte Modell auf dem Metamodell des Zwischenmodells, so sind keine weiteren Transformationen notwendig. Ansonsten dient es zusammen mit den Instanzdaten lediglich als Grundlage für weitere Transformationen.

2.2.1.4 Manuelle Modellierung

Sind weder indirektes Mapping noch Process Mining möglich, weil die benötigten Daten nicht vorhanden und auch nicht zu beschaffen sind, dann muss man den Prozess manuell nachmodellieren. Das kann vor allem bei nicht-automatisierten Teilen des Prozesses nötig sein. Eine manuelle Modellierung ist allerdings soweit wie möglich zu vermeiden, da sie zum einen fehlerträchtig ist und zum anderen enorm aufwendig werden kann. Trotzdem muss für den Fall, dass es keine andere Möglichkeit gibt, ein Modellierungswerkzeug bereitgestellt werden. Das Metamodell des Modellierungswerkzeugs sollte mit dem des Zwischenmodells übereinstimmen um auch hier unnötige Transformationen zu vermeiden.

2.2.1.5 Zusammenfassung

Von den vier vorgestellten Mappingvarianten können also drei wirklich gebraucht werden. Die vierte Variante, das direkte Mapping, wird nicht benutzt, da mit dem indirekten Mapping auf effizientere Weise das Gleiche erreicht werden kann. Indirektes Mapping, Process Mining und manuelle Modellierung schließen sich gegenseitig nicht aus, denn, je nachdem welche Daten von den einzelnen Systemen bereitgestellt werden, können sich diese sinnvoll ergänzen. So kann manuelle Modellierung immer zur Ergänzung oder Korrektur benutzt werden. Process Mining kann als Grundlage für das indirekte

Mapping eingesetzt werden, wenn die benötigten Daten nicht von vornherein bereitgestellt werden. Ansonsten wird nur das indirekte Mapping benutzt.

2.2.2 Zwischenmodell für das indirekte Mapping

Beim indirekten Mapping stellt sich noch die Frage nach dem Zwischenmodell. Auch hier gibt es mehrere Möglichkeiten, die alle ihre Vor- und Nachteile haben. In diesem Abschnitt werden diese abgewogen und es wird überlegt welches Modell in unserem Fall das Beste ist. Untersucht werden ein minimales, ein maximales, ein kanonisches Modell und Variationen dieser.

2.2.2.1 Minimales Modell

Ein minimales Modell besteht aus der Schnittmenge der Elemente aller anderen Modelle, also nur aus den Elementen, die in allen Modellen vorkommen. Definition: Sei E(i) die Elementmenge von System i, wobei $0<i\leq n$ und n=Anzahl der Systeme. Dann ist die Menge der Elemente des minimalen Modells: $E(m)= \bigcap_{i=1..n} E(i)$.

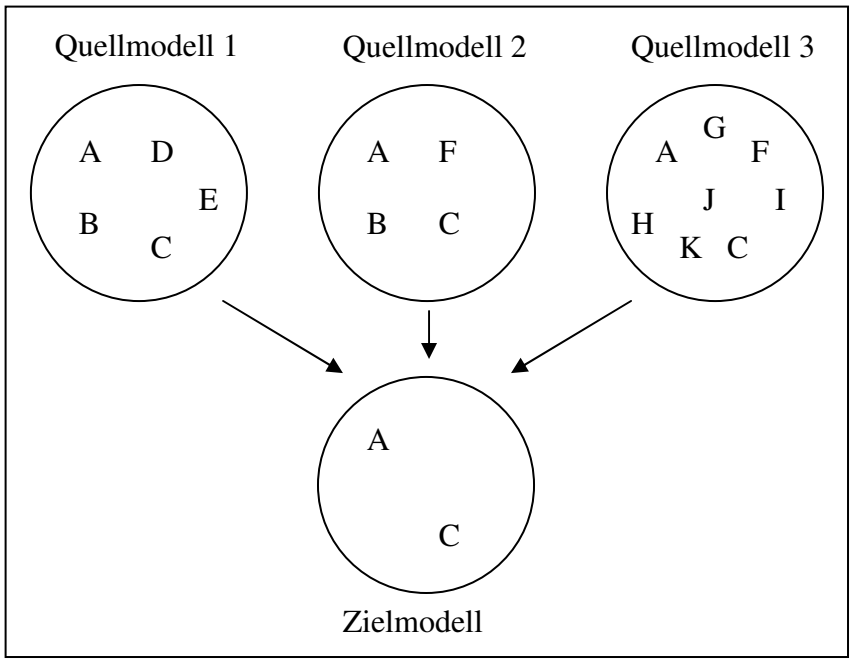


Abbildung 2.3 Elemente eines minimalen Zielmodells

Vorteile dieses Modells sind zum Einen die einfache Abbildung von Elementen des Quellmodells auf vorhandene Elemente des Zielmodells, da diese laut Definition identisch sein müssen. Zum Anderen ist eine übersichtliche Visualisierung möglich, da die Menge der zu visualisierenden Elemente im Vergleich zu anderen Modellen geringer ist und das Modell dadurch nicht mit zu vielen Elementen überladen wird.

Dabei wird allerdings auch gleich der große Nachteil dieses Modells offensichtlich: das Modell ist höchstens so ausdrucksmächtig wie das ausdrucksschwächste Quellmodell. Elemente, die auch nur in einem einzigen Modell nicht vorkommen, können auch nicht im Zwischenmodell vorkommen und somit auch nicht abgebildet werden. Stattdessen müssen diese Elemente, wenn möglich, auf ähnliche Elemente abgebildet werden. Dadurch verändert sich allerdings die Qualität des Mappings. (siehe Abschnitt 2.2.3)

2.2.2.2 Maximales Modell

Ein maximales Modell besteht aus der Vereinigungsmenge der Elemente aller anderen Modelle, also aus allen Elementen, die in mindestens einem Modell vorkommen.

Definition: Sei E(i) die Elementmenge von System i, wobei $0<i\leq n$ und n=Anzahl der Systeme. Dann ist die Menge der betrachteten Elemente $E(M)= \bigcup_{i=1..n} E(i)$.

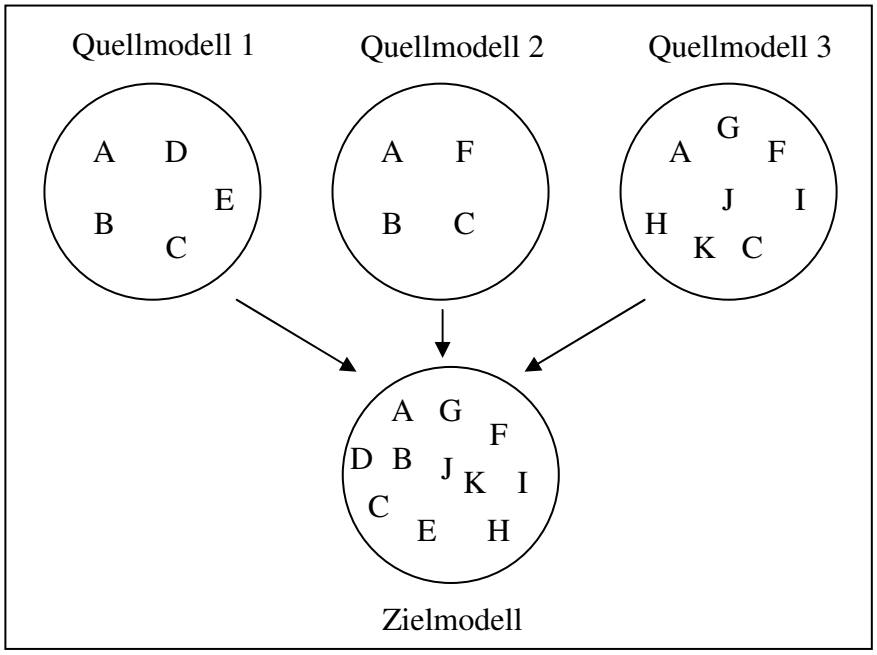


Abbildung 2.4 Elemente eines maximalen Zielmodells

Dieses Modell bietet den Vorteil, dass es eine enorm hohe Ausdrucksmächtigkeit hat und zudem kann man sehr leicht jedes Element auf ein anderes abbilden, weil es laut Definition zu jedem Element der Quellmodelle ein identisches Element im Zwischenmodell gibt.

Der Nachteil ist allerdings, dass durch die vielen unterschiedlichen Elemente, das Modell überladen und unübersichtlich wird, obwohl vielleicht viele Elemente eine durchaus ähnliche oder gleiche Bedeutung haben. Zudem muss die Visualisierungskomponente aufwendiger gestaltet werden, da sehr viele unterschiedliche Elemente angezeigt werden müssen.

2.2.2.3 Kanonisches Modell

Ein kanonisches Modell besteht aus einer vorgegebenen Menge von Elementen. Diese Menge ist nicht, wie die bisherigen Modelle, von den Elementen der abzubildenden Modelle abhängig.

Definition: Sei E(i) die Elementmenge von System i, wobei $0<i\leq n$ und n=Anzahl der Systeme. Dann ist die Menge der betrachteten Elemente E(k) = vorgegeben. E(k) ist also unabhängig von E(i), $\forall$ i.

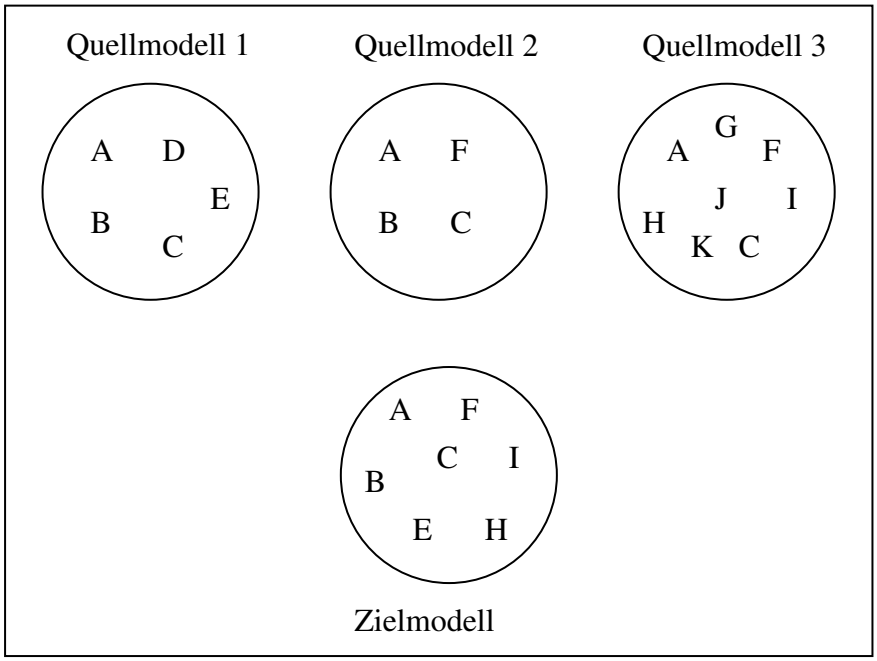


Abbildung 2.5 Elemente eines kanonischen Zielmodells

Vorteil dieses Modells ist zunächst mal die Übersichtlichkeit, weil die Elemente unterschiedlicher Systeme nicht gemischt werden. Dazu kommt, dass durch die vorgegebene Menge an Elementen die Ausdrucksmächtigkeit des Modells nicht eingeschränkt wird. Die Ausdrucksmächtigkeit der einzelnen Systeme wird zwar unter Umständen eingeschränkt, aber nur bis zu der des kanonischen Modells. Diese kann aber durch die Wahl entsprechender Elemente für das Modell so gestaltet werden, dass die Anforderungen an die Ausdrucksmächtigkeit immer erfüllt werden.

Der Nachteil ist allerdings eine erschwerte Abbildung der Quell- auf die Zielelemente. Dadurch, dass die Elemente vorgegeben sind und nicht aus den Quellmodellen stammen, sind keine identischen Abbildungen zwischen Quell- und Zielelementen gegeben. Die sich daraus ergebenden Probleme werden in Kapitel 4 diskutiert.

2.2.2.4 Modellvarianten

Denkbar wären auch Varianten der bisher genannten Modelle, die deren Vorteile größtenteils erhalten und gleichzeitig ihre Nachteile lindern.

So könnte man das minimale Modell dahingehend erweitern, dass das Zielmodell bei *n* Quellmodellen nicht nur diejenigen Elemente enthält, die in allen Quellmodellen, sondern auch die, die in mindestens *m* Modellen vorhanden sind. Dadurch würde man die Ausdrucksmächtigkeit nicht so stark einschränken wie beim minimalen Modell.

Analog könnte man auch vom maximalen Modell ausgehen und dieses dermaßen ändern, dass nur diejenigen Elemente aufgenommen werden, die in mehr als einem Quellmodell enthalten sind, also in mindestens $m>1$ Quellmodellen, wobei $m<n$. Dadurch würde man das Modell nicht unnötig mit zu vielen Elementen überladen.

2.2.2.5 Zusammenfassung

Da die Nachteile der minimalen und maximalen Modelle für eine vernünftige Abbildung nicht hinnehmbar sind, werden wir für das Mapping ein kanonisches Modell benutzen. Dieses ist selbst den verbesserten Varianten der beiden anderen Modelle vorzuziehen, da bei einem kanonischen Modell die Ausdrucksmächtigkeit fest gegeben ist. Bei den Modellen aus Abschnitt 2.2.2.4 variiert die Ausdrucksmächtigkeit je nach gewähltem *m* und nach den Quellmodellen und ist somit schwer einzuschätzen.

2.2.3 Qualität des Mappings

Definition: Die Qualität des Mappings sei ein Maß dafür, wie genau ein Quellelement einem Zielelement entspricht. Dabei bedeutet ein Wert von „1“ eine genaue Übereinstimmung, ein Wert kleiner „1“ eine semantische Verfälschung oder Abwertung. Analog bedeutet ein Wert über „1“ eine semantische Aufwertung.

Bei der Visualisierung sollte jede Abweichung vom optimalen Wert angezeigt werden. Die Art dieser Anzeige ist dabei nebensächlich. Wichtig ist nur, dass erkennbar ist, um welche Abweichung es sich handelt. Dafür könnte man beispielsweise den jeweiligen Wert anzeigen oder wenn der Grad der Abweichung nebensächlich ist, ein grünes Zeichen bei positiver Abweichung und ein rotes bei negativer benutzen. Um so eine Anzeige zu unterstützen müssen beim Mapping die dafür nötigen Daten erzeugt werden.

2.3 Zusammenfassung

Dieses Kapitel hat die wichtigsten Begriffe für diese Arbeit eingeführt. Systeme sind dabei ganz wichtig, denn diese stellen die Daten bereit, die transformiert und integriert werden müssen. Um die Vielfalt der Systeme einzuschränken und später sinnvollere Betrachtungen machen zu können, wurden diese anhand ihrer bereitgestellten Daten in zuvor definierte Klassen eingeteilt. Nach der Klassifikation wurden die Möglichkeiten des Mappings eingeführt und diskutiert. Dabei wurde festgestellt, dass die sinnvollste Variante das indirekte Mapping ist, bei dem zunächst auf ein Zwischenmodell und danach erst auf das eigentliche Zielmodell abgebildet wird. Mögliche Zwischenmodelle

wurden dabei auch diskutiert, wobei für weitere Betrachtungen in der Arbeit ein kanonisches Modell gewählt wurde.

Nachdem nun die wichtigsten Begriffe eingeführt sind, können wir nun mit der eigentlichen Prozessdatentransformation und -integration anfangen. Die Basis dafür bilden die Daten der unterschiedlichen Systeme, die zunächst jede Menge Heterogenitäten aufweisen. Diese werden im folgenden Kapitel behandelt.

3 Verwandte Probleme und Arbeiten

Bevor die speziellen Probleme bei der Transformation und Integration von Prozessen und ihrer Instanzdaten betrachtet werden, werden in diesem Kapitel drei Systeme vorgestellt, deren Grundprobleme die gleichen sind wie diejenigen, die in dieser Arbeit behandelt werden. Alle haben sie gemeinsam, dass sie Daten mehrerer Systeme integrieren sollen, unabhängig vom Zweck und von der Bedeutung dieser Daten.

Zunächst werden die Probleme anhand der föderierten Datenbanken aufgezeigt. Danach werden die systemspezifischen Lösungsansätze der föderierten Datenbanken, Data-Warehouse-Systeme und EAI-Systeme (*Enterprise Application Integration*) kurz vorgestellt. Dabei sollen nicht die Details im Vordergrund stehen, sondern die Tatsache, dass die Probleme bei der Integration heterogener Daten bekannt sind, und dass es bereits unterschiedliche Möglichkeiten gibt diese zu beheben. Aufgrund dieser Tatsache können wir nach diesem Kapitel von den vorgestellten Problemen abstrahieren und eine einheitliche Datenbasis annehmen. Somit können wir uns auf die Probleme konzentrieren, die noch nicht gelöst sind.

3.1 Föderierte Datenbanken

Föderierte Datenbanken sind eine spezielle Form verteilter Datenbank-Management-Systeme (vDBMS) [Dad96], bei der typischerweise eine nachträgliche Integration bereits existierender, bislang dezentral organisierter Informationssysteme vorliegt. Dabei wird ein Teil der lokalen Daten der einzelnen Informationssysteme in einen globalen Verbund integriert. Bei der Integration wird ein globales Schema erzeugt, das eine integrierte Sicht auf die Daten der lokalen DBMSe bietet. Diese bleiben zusammen mit ihren lokalen Schemata unverändert an ihrem ursprünglichen Ort gespeichert und können von lokalen Anwendungsprogrammen weiterhin benutzt werden [Dad96].

Bei der Integration der lokalen Daten in den globalen Verbund kann es aus vielen Gründen zu diversen Problemen kommen. Da all diese Probleme auch bei der Integration von Prozessmodellen auftreten können, werden sie zusammen mit ihren Lösungen im Folgenden genauer betrachtet. Auf weitere Aspekte und Probleme bei vDBMS, wie z.B. Autonomie, wird nicht eingegangen, da sie für die vorliegende Arbeit nicht relevant sind.

Erste Probleme haben ihren Ursprung in der Natur der föderierten Datenbanken. Da bei diesen die zu integrierenden Informationssysteme normalerweise bereits vorliegen, hat man es oft mit heterogenen Systemen zu tun, die zum einen unterschiedliche Einsatzzwecke haben und zum anderen oft über mehrere Jahrzehnte gewachsen sind. „Gewach-

sen“ bedeutet in diesem Zusammenhang, dass mit der Zeit und mit dem Wachstum des Unternehmens das bestehende System um weitere Teilsysteme oder Komponenten ergänzt wurde. Bei solchen Systemen kann es, bedingt durch deren Entwicklung, große Unterschiede geben. So haben sich beispielsweise seit den 60er Jahren die DBMSe von den hierarchischen und Netzwerk DBMSen über die relationalen bis hin zu den objekt-relationalen DBMSen entwickelt [SpDa02]. All diese Systeme haben unterschiedliche Datenmodelle, die bei einer Integration vereinheitlicht werden müssen.

Doch selbst wenn die Systeme homogen sind, kann es zu Heterogenitäten kommen, da die lokalen Schemata der Systeme normalerweise unabhängig voneinander entstanden sind und somit semantisch äquivalente Informationen strukturell oft unterschiedlich abgelegt sind. Im Folgenden werden zunächst die Heterogenitäten behandelt, die bei allen Systemen auftreten können, bevor kurz auf die speziellen Probleme der heterogenen Systeme eingegangen wird.

3.1.1 Koexistenzproblematik

Die zu integrierenden Daten werden normalerweise von Anwendungsprogrammen verwendet, die oft seit vielen Jahren benutzt werden und nicht einfach über Nacht ausgetauscht werden können. Das bedeutet, dass das alte Schema der lokalen Datenbanken für diese Programme weiterhin erhalten bleiben muss. Dies wird erreicht, indem zwei verschiedene Arten von externen Schemata bereitgestellt werden. Zum einen die alten lokalen externen Schemata für die lokalen Anwendungsprogramme und zum anderen ein globales externes Schema für die globalen Anwendungsprogramme [Dad96].

Die Transformation von globalen in lokale und damit auf den Datenbanken ausführbare Anfragen wird schrittweise über die Abbildung auf mehrere Schemata erreicht. Die genaue konzeptuelle und technische Umsetzung kann beispielsweise in [Dad96] oder [Con97] nachgelesen werden. Die grobe Vorgehensweise und die dabei auftretenden Probleme werden in Kapitel 3.1.2 beschrieben.

Die Koexistenzproblematik ist auch bei der Integration der Prozessdaten durchaus relevant, weil auch dort der Zugriff der lokalen Systeme auf ihre Daten weiterhin gewährleistet werden muss und zusätzlich in mindestens einem weiteren System ein globaler Zugriff auf die Daten erforderlich ist. Für die Transformationen der Daten der Prozessmodelle kann der oben beschriebene Ansatz verwendet werden. Dabei tritt, anders als bei föderierten Datenbanken, kein Problem bei Updates auf, da lediglich ein lesender Zugriff auf den Datenbestand nötig ist.

3.1.2 Schema-Integration

Bei der Schema-Integration geht es um die Abbildung der lokalen Schemata auf ein einheitliches globales Schema. Dabei muss beachtet werden, dass die lokalen Schemata

eventuell unabhängig voneinander entworfen wurden. Dadurch können einige Probleme auftreten, auf die im Laufe dieses Kapitels genau eingegangen wird, da sie bei der Integration der Prozessmodelle ebenso auftreten können.

Die Schema-Integration besteht aus vier Phasen:

In der Prä-Integrationsphase wird die Vorarbeit für die Integration gemacht. Zunächst einmal wird die Ausgangssituation überprüft. Dabei wird festgestellt, welche Art von Systemen und welche Art von Informationen integriert werden müssen. Danach werden Regeln für die Vorgehensweise festgelegt, wie z.B. binäre oder n-stellige Integration. Zudem werden noch die jeweiligen Entitäten (Dateneinheiten), Beziehungen und deren Schlüssel bestimmt [Dad96].

In der Vergleichsphase werden Konflikte zwischen den zu integrierenden Schemata ermittelt. Hier kann es zu Namens-, Struktur- und funktionalen Konflikten sowie zu Ambiguitätsproblemen kommen [Dad96]. Auf diese wird in Kapitel 3.1.3 genauer eingegangen.

In der Vereinheitlichungsphase werden Lösungsmöglichkeiten für die, in der vorherigen Phase erkannten Konflikte und Probleme, gesucht [Dad96]. Auf diese wird ebenfalls in Kapitel 3.1.3 eingegangen.

In der Restrukturierungs- und Zusammenfassungsphase wird, basierend auf den nun konfliktfreien Ausgangsschemata, entschieden wie das globale Schema, das Zielschema, festgelegt werden soll. Bei der Wahl des Zielschemas sind Vollständigkeit, Minimalität und Verständlichkeit wichtige Kriterien [Dad96].

3.1.3 Konflikte und Probleme

3.1.3.1 Namenskonflikte

Namenskonflikte können als Synonyme oder als Homonyme auftreten. Synonyme sind verschiedene Benennungen für denselben Sachverhalt, Homonyme dagegen sind gleiche Bezeichnungen für unterschiedliche Sachverhalte [Dad96].

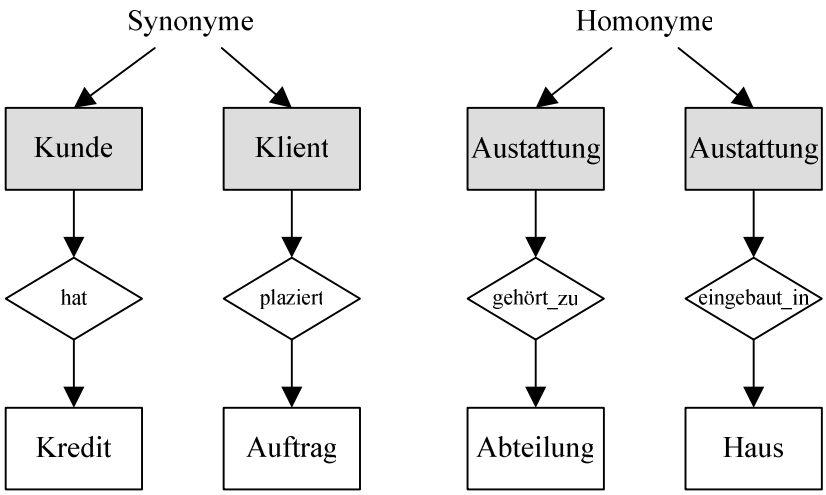


Abbildung 3.1 Namenskonflikte

Namenskonflikte lassen sich durch Umbenennungen einfach auflösen. Treten z.B. auf Attributebene Homonyme auf, so kann man den Entitätstyp als Präfix zum Namen hinzufügen [Dad96]. Bei Synonymen muss man nur die unterschiedlichen Benennungen vereinheitlichen, also durch eine der Benennungen ersetzen.

3.1.3.2 Strukturelle Konflikte

Strukturelle Konflikte sind Konflikte, die die Struktur der Schemata betreffen. Man kann hier Typkonflikte, Beziehungskonflikte, Schlüsselkonflikte und Verhaltenskonflikte unterscheiden.

Wurde beispielsweise in einem Schema ein Sachverhalt als Attribut statt als Entität modelliert, so ist das ein Typkonflikt. Wurden für die gleiche Beziehung unterschiedliche Kardinalitäten verwendet, so ist das ein Beziehungskonflikt. Schlüsselkonflikte treten auf, wenn in den lokalen Schemata unterschiedliche Schlüssel verwendet wurden. Verhaltenskonflikte betreffen Aktionen, die durch das DBMS durchgeführt werden, wie z.B. das Löschen, wo sowohl manuelles als auch kaskadierendes Löschen möglich ist.

Eine Auflösung von Strukturkonflikten ist beispielsweise durch die Umformung von Entitäten in Attribute oder Beziehungen oder durch analoge Umformungen möglich. Durch Änderung der Schlüssel und des Löschverhaltens können weitere Konflikte aufgelöst werden [Dad96].

Weitere strukturelle Konflikte sind in folgenden Bereichen zu finden [BaGu01]:

- Kodierung: Unterschiedliche Kodierungen der Daten möglich. Bsp. Farbcode gespeichert als numerischer Wert oder als Zeichenkette. (001 vs. „schwarz“)
- Zeichensätze: Unterschiedliche Zeichensätze wie z.B. ASCII und UNICODE.

- Zeichenketten: Unterschiedliche Verkettung von Daten wie z.B. „Kurt Meyer“ und „Meyer, Kurt“
- Datumsformat: Unterschiedliche Datumsangaben wie z.B. „MM-DD-YYYY“ und „DD.MM.YYYY“
- Maßeinheiten: Unterschiedliche Maßeinheiten entweder wegen länderspezifischen Einheiten (z.B. „inch“ und „cm“) oder einfach wegen getrennter Modellierung der Schemata (z.B. „m“ und „km“).

Diese Konflikte können meistens durch einfache Transformationen beseitigt werden. Bei den Maßeinheiten bedarf es einer Umrechnung in eine andere Maßeinheit. Bei den anderen Konflikten ist normalerweise lediglich eine einfache Transformation von Werten und Zeichenketten in eine andere Form nötig.

3.1.3.3 Ambiguitäts-Probleme auf Instanzebene

Ambiguitäts-Probleme entstehen auf Instanzebene bei der Mehrfachspeicherung derselben Entität in den Datenbanken der lokalen Systeme mit unterschiedlichen Schlüsseln oder bei der Verwendung desselben Schlüsselwertes für unterschiedliche Entitäten. Abbildung 3.2 veranschaulicht das Problem [Dad96].

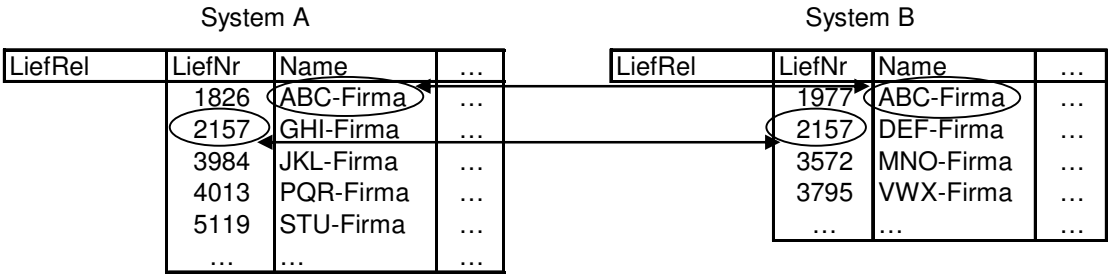


Abbildung 3.2 Ambiguitäts-Probleme [Dad96]

Ambiguitäts-Probleme kann man durch die Einführung eines neuen, global eindeutigen Schlüssels oder durch die globale Verwendung von erweiterten lokalen Schlüsseln lösen. Bei der ersten Variante erfolgt die Umsetzung über eine Schlüsselumsetzungstabelle, die den globalen Schlüssel auf den lokalen Schlüssel abbildet und umgekehrt [Dad96].

3.1.3.4 Probleme bei heterogenen DBMSen

Die bisher beschriebenen Probleme, Konflikte und Vorgehensweisen gelten sowohl für die Integration homogener als auch heterogener DBMSe. Bei den heterogenen Systemen können noch weitere Probleme hinzukommen.

Durch die Verwendung unterschiedlicher Datenmodelle, wie z.B. Netzwerk- und relationales Datenmodell, müssen Anfragen auf die jeweiligen Datenmodelle transformiert werden.

Selbst bei Verwendung der gleichen Datenmodelle kann es zwischen diesen Heterogenitäten geben. Und je nach verwendetem Modell können sich auch die Heterogenitäten stark unterscheiden. Denn bei Verwendung beispielsweise des Netzwerkdatenmodells entstehen durch die fehlende Standardisierung ähnliche Probleme, wie bei heterogenen Datenmodellen, wodurch auch ähnliche Integrationsmechanismen erforderlich sind.

Bei Verwendung des relationalen Datenmodells entstehen dagegen nur kleinere Probleme. Durch unterschiedliche Hersteller kann es zu verschiedenen SQL-Dialekten oder zu speziellen Befehlen mancher Hersteller kommen. Solche Probleme lassen sich allerdings relativ leicht lösen, indem man nur die Grundkonstrukte benutzt, die von allen beteiligten SQL-Systemen unterstützt werden, oder indem man eine allgemein akzeptierte offene Datenbankschnittstelle benutzt, wie z.B. den ISO/OSI RDA-Standard (remote database access) [Dad96].

3.2 Data Warehouse Systeme

Der Begriff Data Warehouse stammt aus zwei Bereichen, aus der Informatik und aus der Betriebswirtschaft. Grund hierfür sind die dualen Ziele der Data Warehouse Systeme. Aus technischer Sicht integrieren sie Daten aus verschiedenen Datenquellen und stellen diese, aus betriebswirtschaftlicher Sicht, dem Anwender zu Analysezwecken zur Verfügung.

Dabei müssen Data Warehouse Systeme ähnliche Probleme lösen wie föderierte Datenbanken und somit auch ähnliche wie wir bei der Integration der Daten heterogener Systeme. Im Folgenden werden deswegen nur die entscheidenden Unterschiede zu den föderierten Datenbanken aufgezeigt.

Ein Data Warehouse ist eine physische Datenbank, die eine integrierte Sicht auf beliebige Daten ermöglicht [BaGu01]. Dadurch entstehen Probleme, wie bei der Integration der föderierten Datenbanksysteme. Weitere Probleme entstehen durch den Analyseaspekt des Data Warehouses. Durch Unterschiede bei transaktionalen Anfragen, wie bei üblichen Datenbanken, und analytischen Anfragen, wie bei den Data Warehouses, wird ein anderer Modellierungsansatz für die Daten benötigt. Als adäquater Ansatz wird das multidimensionale Modell vorgeschlagen [BaGu01].

Ein weiterer Unterschied zu föderierten Datenbanken ist, dass die Daten normalerweise nicht modifiziert werden. Es können neue Daten aufgenommen werden, diese überschreiben aber die alten nicht. Außerdem dient das Schema des Data Warehouses einem speziellen Analysezweck und die Daten werden lokal und global redundant gehalten.

3.3 Enterprise Application Integration (EAI)

EAI stellt einen Ansatz zur Integration unabhängiger Anwendungssysteme dar. Sowohl Daten- als auch Funktionsintegration werden ermöglicht, unabhängig von der Plattform der verschiedenen Systeme. Im Kontext dieser Arbeit ist allerdings lediglich die Datenintegration interessant.

Das Ziel von EAI ist dabei, die Schnittstellen zwischen den verschiedenen Systemen zu reduzieren, indem es als gemeinsame Middleware zwischen den beteiligten Systemen fungiert. Unter Middleware versteht man allgemein Technologien, die Dienstleistungen zur Vermittlung zwischen Anwendungen anbieten, so dass die Komplexität der zugrundeliegenden Anwendungen und Technologien verborgen bleibt [Wik05].

Für die Integration der Daten werden zwei primäre Integrationsmethoden eingesetzt. Die wohl häufiger benutzte ist nachrichtenorientiert. Bei dieser Methode werden immer nur von einem Sender Nachrichten an einen Empfänger geschickt. Die Nachrichten enthalten Metadaten, die den Aufbau der Nachricht beschreiben und die eigentlichen Nutzdaten, wie z.B. Anweisungen oder angeforderte Daten. Zwischen Sender und Empfänger befindet sich die Middleware, deren Message Broker [Pry05] bei Bedarf Anpassungen an der Nachricht vornimmt, bevor diese an den Empfänger weitergeleitet wird. Somit müssen alle beteiligten Anwendungen nur die Schnittstelle kennen, die für das Verschicken einer Nachricht notwendig ist.

Bei der anderen Integrationsmethode läuft die Kommunikation über anwendungsspezifische Schnittstellen, ähnlich wie bei traditionellen Programmstrukturen mit Prozeduraufrufen usw. Der Unterschied ist allerdings, dass die Schnittstellen nach außen hin sichtbar sind und von anderen Anwendungen genutzt werden können.

Unabhängig von der Integrationsmethode werden noch Konnektoren (auch Adapter genannt) als Zugangspunkt zu den Systemen benötigt. Diese sind Logikbausteine mit dem Zweck strukturierten Zugang zu den Daten oder zur Logik der Systeme zu ermöglichen. Mit Hilfe dieser Adapter kann auch auf andere Datenquellen als nur auf Datenbanken zugegriffen werden, was im Kontext dieser Arbeit enorm wichtig ist.

3.4 Zusammenfassung

In diesem Kapitel wurden Konflikte und Probleme behandelt, die bei der Integration von Daten mehrerer Systeme auftreten können und es wurden Technologien zur Lösung dieser Probleme aufgezeigt. Anhand der föderierten Datenbanken wurden u.a. die Koexistenzproblematik, Probleme bei der Integration unterschiedlicher Schemata und Datenmodelle, Namens- und strukturelle Konflikte, sowie ihre Lösungen vorgestellt. Danach wurden mit Data-Warehouse-Systemen und EAI-Systemen zwei weitere Systeme kurz vorgestellt, die analoge Probleme lösen. Dabei wurden allerdings nur die Unter-

schiede zwischen den drei Systemen deutlich gemacht. Während föderierte Datenbanken mehrere DBMSe über globale Schemata integrieren und die Daten unverändert in den lokalen Datenquellen bleiben, transformieren Data-Warehouse-Systeme die Daten und legen sie in einem eigenen Arbeitsbereich ab. EAI-Systeme integrieren die Daten nachrichtenbasiert über Message Broker oder über anwendungsspezifische Schnittstellen.

Die Details sind dabei nebensächlich. Entscheidend an dieser Stelle ist lediglich, dass es Technologien gibt, die einen einheitlichen Zugriff auf heterogene Daten und Datenquellen ermöglichen. Somit können wir uns im nächsten Kapitel auf die eigentlichen Probleme der Prozesstransformation konzentrieren.

4 Prozesstransformation

Nachdem Technologien eingeführt wurden, die eine einheitliche Datenbasis ermöglichen, können wir uns in diesem Kapitel auf die Probleme konzentrieren, die bei Transformation von Prozessfragmenten in Fragmente des gleichen Metamodells auftreten. Das ist der zweite Schritt im Mapping-Prozess und somit auch die zweite Problemkategorie.

Abbildung 4.1 veranschaulicht, was bei dieser Transformation passiert. Die Prozesse der einzelnen Systeme werden alle auf ein einheitliches Metamodell abgebildet. Die dabei neu entstandenen Modellfragmente, dienen als Basis für die Prozessintegration, die im nächsten Kapitel behandelt wird.

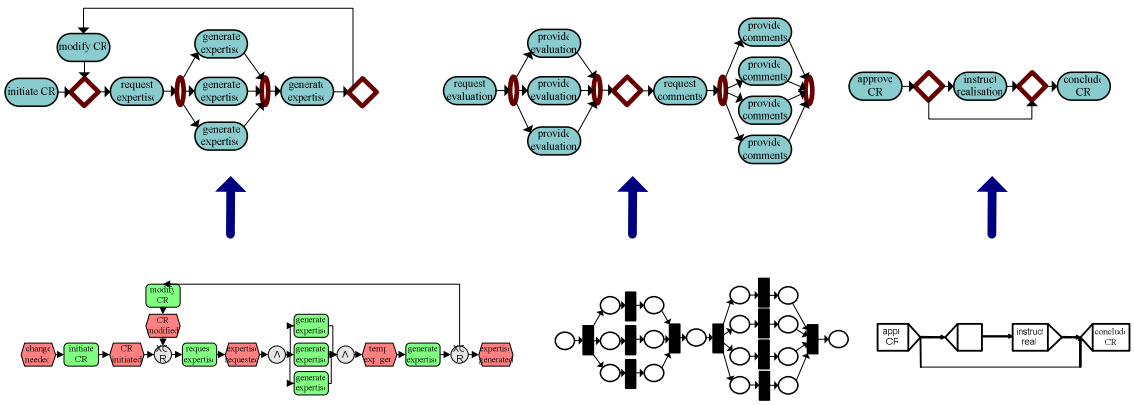


Abbildung 4.1 Transformation von Prozessfragmenten

Bei der Transformation treten jede Menge Probleme und Konflikte auf. Wir werden zunächst in Abschnitt 4.1 Modellkonflikte behandeln, bevor wir in den Abschnitten 4.2 und 4.3 Namenskonflikte und Probleme, die durch fehlende Daten entstehen können, aufzeigen. Am Ende dieses Kapitels, in Abschnitt 4.4 wird anhand von Modellierungskonstrukten auf einige speziellere Aspekte im Bereich des Kontrollflusses eingegangen.

4.1 Modellkonflikte

Modellkonflikte entstehen wegen Heterogenitäten zwischen den Metamodellen der einzelnen Systeme und weil selbst bei gleichem Metamodell die Umsetzung der Modellierungskonstrukte durch die Systeme unterschiedlich sein kann.

Durch die Vielzahl der Metamodelle und der tatsächlich umgesetzten Konstrukte durch die Systeme ist eine allgemeine Betrachtung allerdings nicht möglich. Deswegen wird in den folgenden Abschnitten lediglich anhand von Beispielen aufgezeigt in welchen

Bereichen und in welcher Form Konflikte auftreten können. Die betrachteten Bereiche sind der Kontroll- und Datenfluss, sowie das Organisationsmodell. Dabei liegt der Fokus auf dem Kontrollfluss.

4.1.1 Kontrollfluss

Der Kontrollfluss beschreibt die Struktur und den Ablauf eines Prozesses. Betrachtet man zwei Prozesse, deren Kontrollflüsse auf unterschiedlichen Metamodellen basieren, so sieht man gleich einen visuellen Unterschied. Abbildung 4.2 und Abbildung 4.3 zeigen beide den ersten Teil des *ChangeManagement*-Prozesses, einmal als ereignisgesteuerte Prozesskette (EPK) in ARIS modelliert und einmal als Aktivitätennetz in Staffware.

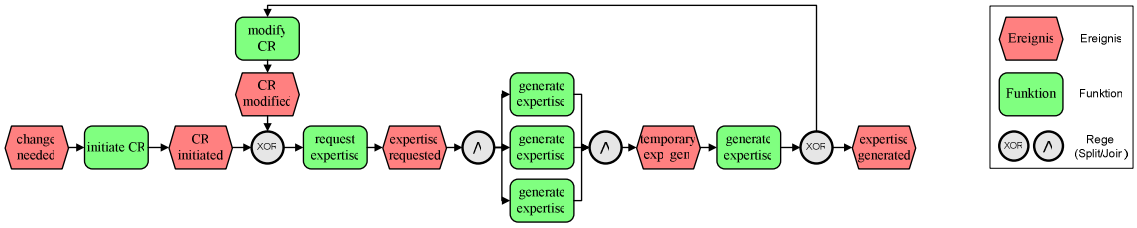


Abbildung 4.2 ARIS – Change Management erster Teil

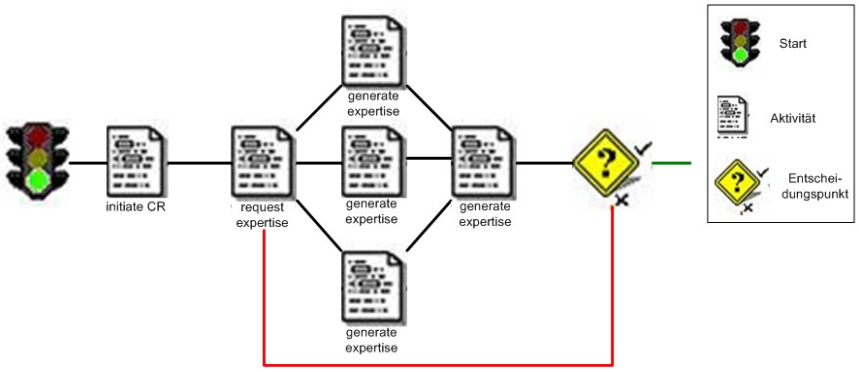


Abbildung 4.3 Staffware – ChangeManagement erster Teil

Vergleicht man die beiden Metamodelle miteinander, so stellt man fest, dass bei EPKs jede Aktivität von einem Ereignis gefolgt ist, bei Aktivitätennetzen dagegen Aktivitäten auf Aktivitäten folgen. Das Eintreten eines Ereignisses löst bei EPKs die nachfolgende Aktivität aus. So ein Ereignis kann beispielsweise das Ende der vorhergehenden Aktivität sein. Bei Aktivitätennetzen dagegen gibt es gar keine Ereignisse. Das Ereignis „Ende einer Aktivität“ ist hier nur implizit gegeben, indem beim Beenden einer Aktivität die nachfolgende Aktivität gestartet wird.

Zudem ist es in Aktivitätennetzen eigentlich nicht möglich Schleifen zu modellieren, weil diese zu einer Verklemmung des Prozesses führen. Somit wäre eine Modellierung des obigen Prozesses gar nicht erlaubt. Dass es in Staffware dennoch geht, liegt daran, dass Staffware keine Synchronisation mehrerer, in eine Aktivität eingehender Kanten ermöglicht. Verklemmungen werden dadurch ausgeschlossen. In EPKs ist die Modellierung von Schleifen kein Problem.

Sind Schleifen bei den Systemen modellierbar, so stellt sich noch die Frage, wie diese umgesetzt werden. So gibt es Systeme, die Schleifen explizit modellieren, wie z.B. ADEPT und solche, die Schleifen implizit modellieren, wie z.B. MQSeries Workflow. In MQSeries werden dazu mehrere Aktivitäten in Blöcke zusammengefasst und der Block wird so oft wiederholt, bis eine Austrittsbedingung zutrifft.

Ähnlich gibt es auch große Unterschiede zwischen den Aktivitäten, die unterstützt werden. So kann man beispielsweise in ARIS für automatische Aktivitäten „Systemfunktion“, „SAP-Funktion“, „IS-Funktion“ oder „DV-Funktion“ wählen [IDS04]. In Staffware dagegen sind nur „automatischer Schritt“ oder „Skript“ möglich [Sta00].

Große Unterschiede gibt es ebenso in der Weglenkung. In ARIS gibt es Regeln, die sowohl für eingehende als auch für ausgehende Kanten eine „XOR“-, „AND“- oder „OR“-Auswertung ermöglichen.

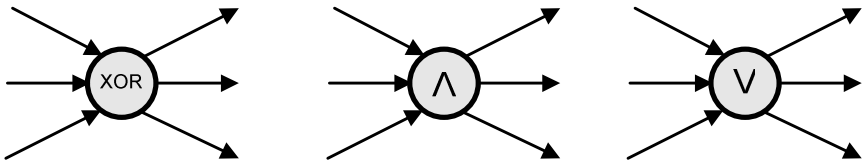


Abbildung 4.4 Splits und Joins in ARIS

In Staffware gibt es viel weniger Möglichkeiten. So gibt es ein „XOR-Split“ als „Entscheidungspunkt“ nur für zwei ausgehende Kanten. Die „AND-Regel“ gibt es lediglich als „AND-Split“ und wird implizit umgesetzt, indem mehrere Kontrollkanten von einer Aktivität ausgehen. Ein „AND-Join“ ist wegen des oben angesprochenen Problems der Verklemmung nicht möglich. Bei mehreren in eine Aktivität eingehenden Kanten wird somit implizit ein „XOR-Join“ umgesetzt. Ein „OR-Join“ ist in Staffware nicht möglich und ein „OR-Split“ ist auch nur indirekt über mehrere „XOR-Regel“ modellierbar.

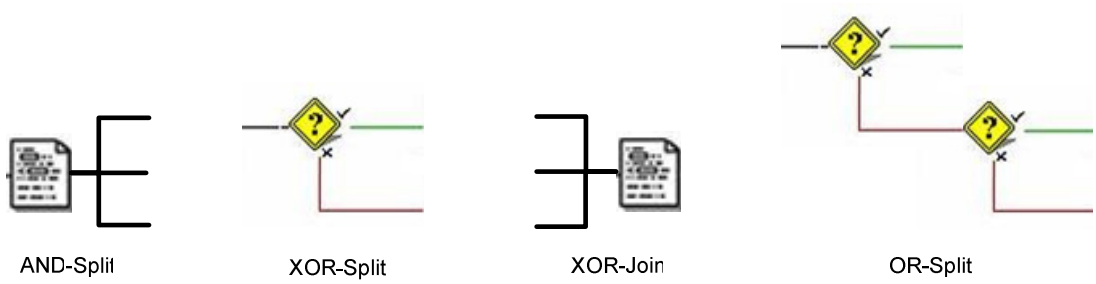


Abbildung 4.5 Splits und Joins in Staffware

4.1.2 Datenfluss

Auch im Datenfluss gibt es Unterschiede. In ARIS wird der Datenfluss explizit in speziellen Datenflussdiagrammen modelliert. In Staffware dagegen wird der Datenfluss implizit über globale Variablen geregelt.

4.1.3 Organisationsmodell

Im Organisationsmodell der beiden Systeme gibt es ebenfalls Heterogenitäten. So ist es beispielsweise in Staffware nicht möglich Hierarchien zwischen den Organisationseinheiten zu modellieren. Dazu müssen externe Tools benutzt werden. In ARIS dagegen ist die Modellierung von Hierarchien kein Problem.

Auch bei der Bearbeiterzuordnung gibt es Unterschiede. In ARIS werden diese explizit modelliert, in Staffware dagegen nur implizit.

4.2 Namenskonflikte

Allgemeine Namenskonflikte wurden bereits in Kapitel 3 beschrieben. Neben diesen allgemeinen Namenskonflikten, die bei allen möglichen Daten vorkommen können, gibt es noch Namenskonflikte, die speziell bei Prozessmodellen auftreten können. Bei diesen unterscheidet man ebenfalls zwischen Synonymen und Homonymen.

4.2.1 Synonyme

Werden in Prozessmodellen verschiedene Bezeichnungen für dasselbe Modellierungskonstrukt benutzt, so sind das Synonyme. So wird z.B. in ARIS „Funktion“ für das gleiche Konstrukt wie Aktivität in ADEPT benutzt (s. Abbildung 4.6).

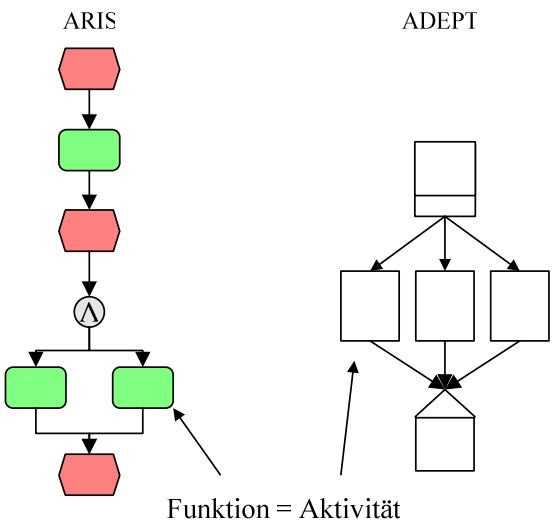


Abbildung 4.6 Synonyme bei Modellierungskonstrukten

4.2.2 Homonyme

Wird die gleiche Bezeichnung für Konstrukte mit unterschiedlicher Bedeutung benutzt, so spricht man von Homonymen. So kann man beispielsweise sowohl im Kontext des Kontrollflusses als auch eines Organisationsmodells von einer Funktion sprechen, allerdings mit unterschiedlicher Bedeutung (s. Abbildung 4.7).

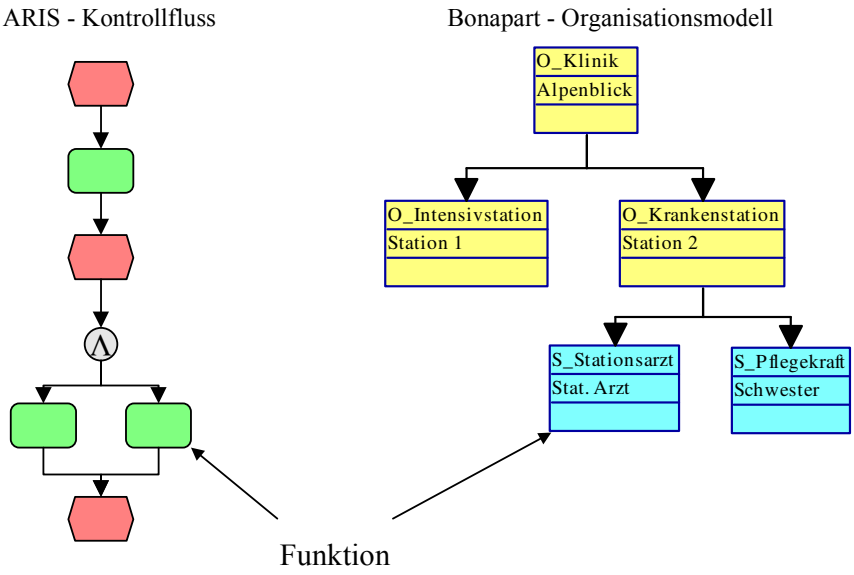


Abbildung 4.7 Homonyme bei Modellierungskonstrukten

4.3 Unvollständigkeit von Daten

In den vorangegangenen beiden Abschnitten wurden nur Systeme betrachtet, die ihre Modelldaten bereitstellen, also Klasse-3-Systeme. Das ist allerdings nur der Optimalfall. Kommen auch Klasse-1- und Klasse-2-Systeme dazu, dann hat man, zusätzlich zu den bisherigen Problemen, das Problem, dass Daten fehlen, die für die Transformation benötigt werden.

Abbildung 4.8 zeigt den bereits bekannten ChangeManagement-Prozess, diesmal allerdings mit allen 3 Systemklassen. Somit fehlen bei einem System die Modelldaten und bei dem anderen sowohl Modelldaten als auch die Log-Daten.

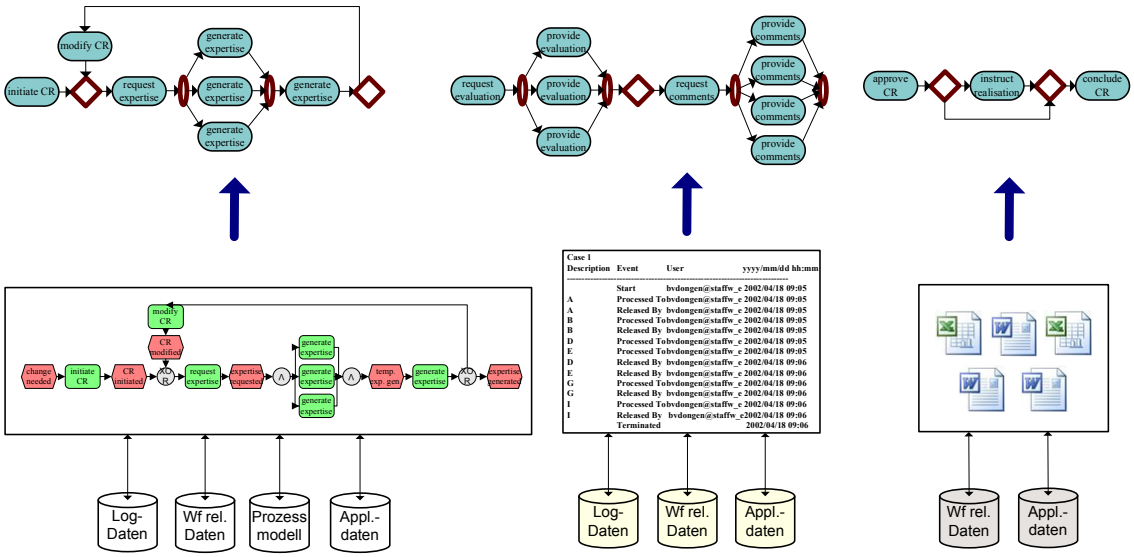


Abbildung 4.8 Transformation von Prozessfragmenten unterschiedlicher Systemklassen

Um eine Transformation zu ermöglichen, müssen bei den Systemklassen, die zu wenige Daten bereitstellen, diese irgendwie beschafft werden. Das ist gleichbedeutend mit der Aufwertung aller Systemklassen auf Klasse 3.

Je niedriger die Klasse, umso mehr Vorarbeit ist zu leisten, bevor die Transformation vorgenommen werden kann.

Bei Klasse-2-Systemen muss lediglich das Prozessmodell erstellt werden, um das System auf Klasse-3 aufzuwerten. Bei Klasse-1-Systemen müssen zuerst die Log-Daten gesammelt und dann das Prozessmodell erstellt werden.

Die Aufwertung der Systeme und die dafür nötigen Komponenten finden sich in Abschnitt 7.2.1.2.

4.4 Beispiel - Modellierungskonstrukte

Die Modellierungskonstrukte sind Bestandteile einer Prozessdefinition und beschreiben u.a. den Kontrollfluss. Es ist offensichtlich, dass unterschiedliche Modellierungskonstrukte unterschiedliche Prozessdefinitionen zur Folge haben. Die Unterschiede bei den Modellierungskonstrukten des Kontrollflusses, durch die es zu Problemen bei der Transformation kommen kann, werden im Folgenden am Vergleich der Systeme Staffware und MQSeries deutlich gemacht.

Für einen Vergleich der beiden Systeme anhand der Kontrollflusskonstrukte bieten sich die Workflow Patterns von W.M.P. van der Aalst [AHKB03] an, da diese sowohl von einer bestimmten Implementierung als auch von bestimmten Einsatzgebieten unabhängig sind. Laut Definition ist ein Pattern „die Abstraktion von etwas Konkretem, das sich in einem bestimmten Kontext wiederholt“ [AHKB03].

Sie stellen die Anforderungen an die Ausdrucksmächtigkeit von WfMSen ohne dabei bestimmte Beschreibungssprachen zu benutzen. Somit kann man mit ihrer Hilfe WfMSe objektiv miteinander vergleichen.

Insgesamt gibt es 20 Patterns, die sechs Gruppen zugeordnet sind. Einige davon sind grundlegende Kontrollkonstrukte, die für alle WfMSe essentiell sind, andere wiederum sind sehr speziell und werden nur ganz selten umgesetzt.

Eine genaue Beschreibung aller 20 Patterns findet sich im Anhang B.

4.4.1 Systemvergleich anhand von Workflow Patterns

In diesem Abschnitt werden Heterogenitäten bei den Modellierungskonstrukten am Beispiel der beiden Systeme Staffware und MQSeries Workflow aufgezeigt. Syntaktische Heterogenitäten, wie z.B. unterschiedliche Benennung, werden hier nicht betrachtet. Der Vergleich der beiden Systeme wird anhand der vorgestellten Workflow Patterns vorgenommen.

Tabelle 4.1 zeigt welche Patterns von den beiden Systemen unterstützt werden. Ein „+“ bedeutet eine direkte Unterstützung durch das jeweilige System, „+/-“ bedeutet, dass das Pattern durch andere Konstrukte abgebildet werden kann. „-“ bedeutet keine Unterstützung.

Pattern	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
MQSeries Workflow	+	+	+	+	+	+	+	-	-	-	+	-	+	-	-	-	-	-	-	-
Staffware	+	+	+	+	+	+/-	-	-	-	+	+	-	+	-	-	+/-	-	-	+	-

Tabelle 4.1 Unterstützung der Workflow Patterns durch Staffware und MQSeries [AHKB03]

Im Folgenden werden die Modellierungskonstrukte, die von beiden Systemen oder von einem der beiden unterstützt werden, betrachtet. Auf Patterns, die von keinem der Systeme unterstützt werden, wird nicht näher eingegangen, da es bei diesen zu keinen Heterogenitäten kommen kann.

Pattern 1 (Sequence): Wird von beiden Systemen direkt unterstützt.

Pattern 2 (Parallel Split): Wird von beiden Systemen, durch mehrere ausgehende Kanten einer Aktivität, unterstützt.

Pattern 3 (Synchronisation): Wird von MQSeries direkt unterstützt. In Staffware gibt es einen „Wait Step“, der auf die Beendigung aller eingehenden Kanten wartet.

Pattern 4 (Exclusive Choice): Wird von MQSeries, durch sich gegenseitig ausschließende Bedingungen auf den Transitionen, unterstützt. In Staffware gibt es einen „Condition Step“, in dem eine Bedingung zu *true* oder *false* evaluiert wird.

Pattern 5 (Simple Merge): Wird von MQSeries direkt unterstützt. Bei Staffware können die normalen Schritte mehrere eingehende Kanten haben.

Pattern 6 (Multi Choice): Wird von MQSeries, durch sich nicht ausschließende Bedingungen auf den Transitionen unterstützt. Keine direkte Unterstützung durch Staffware. Kann aber durch Hintereinanderschalten mehrerer Bedingungen simuliert werden. Dadurch ergeben sich allerdings komplexe und unübersichtliche Strukturen.

Pattern 7 (Synchronising Merge): Wird von MQSeries direkt unterstützt. Keine Unterstützung durch Staffware.

Pattern 10 (Arbitrary Cycles): Keine Unterstützung durch MQSeries. Wird von Staffware mit einigen Beschränkungen unterstützt.

Pattern 11 (Implicit Termination): Wird von beiden Systemen direkt unterstützt.

Pattern 13 (MI with a priori known design time knowledge): Wird von beiden Systemen durch eine Kombination von Splits und Joins unterstützt.

Pattern 16 (Deferred Choice): Keine Unterstützung durch MQSeries. Auch keine direkte Unterstützung durch Staffware, kann aber durch einen Parallel Split und den Abbruch der nicht-gewählten Aktivität simuliert werden.

Pattern 19 (Cancel Activity): Keine Unterstützung durch MQSeries. Direkte Unterstützung durch Staffware.

Wie man beim Vergleich der Patterns sehen konnte, gibt es unterschiedliche Heterogenitäten zwischen den Systemen. Zum einen gibt es Konstrukte, die von einem System direkt unterstützt werden, vom anderen wiederum gar nicht (Patterns 7, 10, 19) oder nur indirekt durch Kombinationen anderer Konstrukte (Patterns 6, 16). Zum anderen gibt es

auch Patterns, die zwar von beiden Systemen direkt unterstützt werden, allerdings in unterschiedlicher Form (Patterns 3, 4, 5). Möglichkeiten zur einheitlichen Darstellung dieser heterogenen Konstrukte werden im folgenden Kapitel diskutiert.

4.4.2 Mapping der Modellierungskonstrukte

In Kapitel 2.2 wurde bereits über die grundlegenden Möglichkeiten des Mappings diskutiert. In diesem Kapitel wird nun konkret betrachtet, welche der Möglichkeiten für die identifizierten Heterogenitäten bei den Modellierungskonstrukten am besten ist. Die Betrachtung erfolgt zunächst anhand der drei Kategorien, die am Ende von Abschnitt 4.4.1 identifiziert wurden: unterschiedliche Umsetzung der Konstrukte, direkte Unterstützung durch ein System vs. indirekte Unterstützung durch das andere und direkte Unterstützung durch ein System vs. keine Unterstützung durch das andere. Zum Schluss wird noch ein weiteres Problem aufgegriffen, das bei dem Systemvergleich nicht ersichtlich wurde.

4.4.2.1 Unterschiedliche Umsetzung der Konstrukte

Werden Konstrukte durch alle Systeme unterstützt, allerdings in unterschiedlicher Form, so hat ein Mapping auf ein maximales oder minimales Konstruktmodell große Nachteile. Beim maximalen Modell müssen alle Konstrukte der verschiedenen Systeme bei der Visualisierung angezeigt werden. Dadurch wird eine Vielzahl von Konstrukten visualisiert, die eigentlich denselben Sachverhalt ausdrücken. Beim minimalen Modell können diese Konstrukte allerdings nicht visualisiert werden, da sie trotz gleicher Bedeutung unterschiedliche Umsetzungen haben. Das schränkt die Ausdrucksmächtigkeit der Visualisierung enorm ein.

Die optimale Variante für das Mapping in diesem Fall ist die Abbildung auf ein kanonisches Modell. So können die unterschiedlichen Konstrukte, die alle die gleiche Bedeutung haben, auf ein Konstrukt des kanonischen Modells abgebildet werden. So bleibt die Ausdrucksmächtigkeit erhalten und eine übersichtliche einheitliche Visualisierung wird ebenfalls gewährleistet.

4.4.2.2 Direkte vs. indirekte Unterstützung der Konstrukte

Werden Konstrukte durch manche Systeme direkt und durch andere nur indirekt unterstützt, dann treten beim Mapping auf ein maximales oder minimales Modell genau die gleichen Probleme auf wie in Abschnitt 4.4.2.1.

Auch hier können die Probleme durch ein Mapping auf ein kanonisches Modell gelöst werden. Der Unterschied zu Abschnitt 4.4.2.1 ist allerdings, dass man hier nicht nur ein Konstrukt des Quellmodells auf das kanonische Modell abbilden muss. Hier hat meistens eine Kombination von Konstrukten des Quellmodells die gleiche Bedeutung wie

ein einziges Konstrukt des Zielmodells. Somit muss diese ganze Kombination auf ein einziges Konstrukt abgebildet werden.

4.4.2.3 Direkte vs. keine Unterstützung der Konstrukte

Werden Konstrukte durch manche Systeme direkt und durch andere gar nicht unterstützt, dann macht vor allem ein Mapping auf ein minimales Modell keinen Sinn. Der Grund ist der gleiche, wie bei den zwei vorangegangenen Abschnitten.

Einen Unterschied gibt es hier zum Mapping auf ein maximales Modell. In diesem Fall gibt es nicht so viele unterschiedliche Konstrukte mit gleicher Bedeutung wie in den vorherigen beiden Abschnitten, da bei einigen Systemen diese Konstrukte gar nicht unterstützt werden. Somit fällt dieser Nachteil hier nicht so schwer ins Gewicht. Trotzdem könnten unnötig viele unterschiedliche Konstrukte desselben Sachverhaltes visualisiert werden, was das Verständnis des Prozesses erschwert. Ein Vorteil wäre allerdings das einfachere Mapping.

Beim Mapping auf ein kanonisches Modell hat man wieder den Vorteil, dass man genügend Konstrukte haben kann, um die Ausdruckmächtigkeit nicht einzuschränken, allerdings auch nicht zu viele unterschiedliche, sondern nur die einheitlichen des kanonischen Modells. So hat man die Möglichkeit die Konstrukte abzubilden, die im Quellmodell vorhanden sind, wobei gleichzeitig die nicht vorhandenen Konstrukte die Ausdrucksmächtigkeit nicht beeinträchtigen.

4.4.2.4 Keine Unterstützung der Konstrukte durch das kanonische Modell

Da bei dem vorangegangenen Systemvergleich die beiden Systeme anhand der Patterns von W.M.P. van der Aalst miteinander verglichen wurden, die zusammen ein kanonisches Modell ergeben, waren immer alle Konstrukte der Quellmodelle im Zielmodell vorhanden. Dadurch ist ein Problem nicht aufgefallen. Es kann auch Konstrukte geben, die im Quellmodell vorhanden sind, aber im Zielmodell, also im kanonischen Modell, nicht. Denn selbst van der Aalst erhebt keinen Anspruch auf Vollständigkeit seiner Patterns. Und wenn ein anderes kanonisches Modell benutzt wird, das weniger Ausdrucksmächtigkeit besitzt, dann ist dieses Problem durchaus relevant.

Die optimale Lösung für das Problem wäre ein erweiterbares kanonisches Modell, bei dem man bei Bedarf neue Konstrukte anlegen kann. So kann sowohl die Ausdrucksmächtigkeit, als auch eine übersichtliche einheitliche Visualisierung erhalten bleiben.

Ist das kanonische Modell nicht erweiterbar, so muss man auf andere, weniger zufrieden stellende Lösungen zurückgreifen. Eine Möglichkeit wäre, das jeweilige Konstrukt, wenn möglich, in mehrere einfachere Konstrukte zu zerlegen, die aber zusammengenommen genau die gleiche Wirkung und Bedeutung haben. Dadurch würde man allerdings das einheitliche Prozessbild unnötig überladen und unübersichtlich machen. Die

Bestrebungen der Systemhersteller, die Modellierung und Visualisierung durch kompaktere Prozesse einfacher zu machen, wären umsonst.

Kann man das jeweilige Konstrukt durch einfachere Konstrukte nicht darstellen, so ist die eben erwähnte Lösung auch nicht möglich. Es bleibt dann nur noch die Möglichkeit, das jeweilige Konstrukt manuell nachzumodellieren oder im globalen Prozess mit einem Platzhalter anzudeuten, dass an der Stelle ein Modellierungskonstrukt nicht visualisiert werden konnte. Die zweite Lösung ist allerdings nicht akzeptabel. Deshalb soll an dieser Stelle die Wichtigkeit eines erweiterbaren kanonischen Modells nochmals betont werden.

4.5 Zusammenfassung

In diesem Kapitel wurden, ausgehend von einer einheitlichen Datenbasis, Probleme identifiziert, die bei der Transformation von Prozessfragmenten auf ein anderes Metamodell auftreten können. Dabei wurden Modellkonflikte sowohl im Kontroll- und Datenfluss, als auch im Organisationsmodell identifiziert. Zudem kann es, unabhängig von den bereits in Kapitel 3 identifizierten Namenskonflikten, Synonyme und Homonyme speziell auch auf der Prozessebene geben. Werden nicht nur Klasse-3-Systeme abgebildet, dann kommt zu den vorherigen Problemen hinzu, dass für die Transformation benötigte Daten fehlen, die zunächst beschafft werden müssen. Zum Schluss des Kapitels wurden, anhand von Modellierungskonstrukten, die Modellkonflikte im Kontrollfluss zweier bekannter WfMSe aufgezeigt.

5 Prozessintegration

Bei der Prozessintegration werden verschiedene Prozessfragmente zu einem Prozess zusammengefügt. Die Integration setzt auf den Prozessfragmenten auf, die bei der Prozesstransformation erzeugt werden. Diese sind sowohl syntaktisch als auch semantisch in einer einheitlichen Form. Dennoch muss noch Vorarbeit geleistet werden, bevor diese gemeinsam in den Gesamtprozess integriert werden können. Zum einen müssen die Beziehungen zwischen den Prozessfragmenten identifiziert werden (s. Abschnitt 5.1), zum anderen müssen Inkonsistenzen, die durch unabhängige Modellierung der Prozessfragmente entstehen, beseitigt werden (s. Abschnitt 5.2). Davon ausgenommen sind Teile des Prozesses, die außerhalb der Systemgrenzen nur bis zu einem gewissen Grad dargestellt werden dürfen (s. Abschnitt 5.3). Unabhängig von den letzten beiden Punkten, muss der Prozess vollständig abgebildet werden. Dazu müssen u.U. Aktivitäten ermittelt werden, die zwar im Laufe der Prozesse durchgeführt werden, in den Systemen allerdings nicht aufgeführt sind (s. Abschnitt 5.4). Als Gegensatz zu diesen Aktivitäten gibt es Aktivitäten, die mehr als einmal durchgeführt werden, deren Darstellung allerdings nur einmal erwünscht ist (s. Abschnitt 5.5). Zudem müssen Probleme geklärt werden, die durch mehrstufige hierarchische Prozesse entstehen können (s. Abschnitt 5.6). Erst nach Klärung dieser Fragen ist eine konsistente, integrierte Abbildung systemübergreifender Prozesse möglich.

5.1 Identifikation von Beziehungen zwischen Prozessfragmenten

Bevor Prozessfragmente zu einem Prozess zusammengefügt werden können, müssen zunächst Beziehungen zwischen diesen identifiziert werden. Die Beziehungen bestehen aus Aufrufen fremder Systeme und können sich in ihrer Gesamtheit, je nachdem wie sich die Systeme gegenseitig aufrufen, sehr unterschiedlich gestalten. Abbildung 5.1 zeigt einen Überblick der Beziehungstypen.

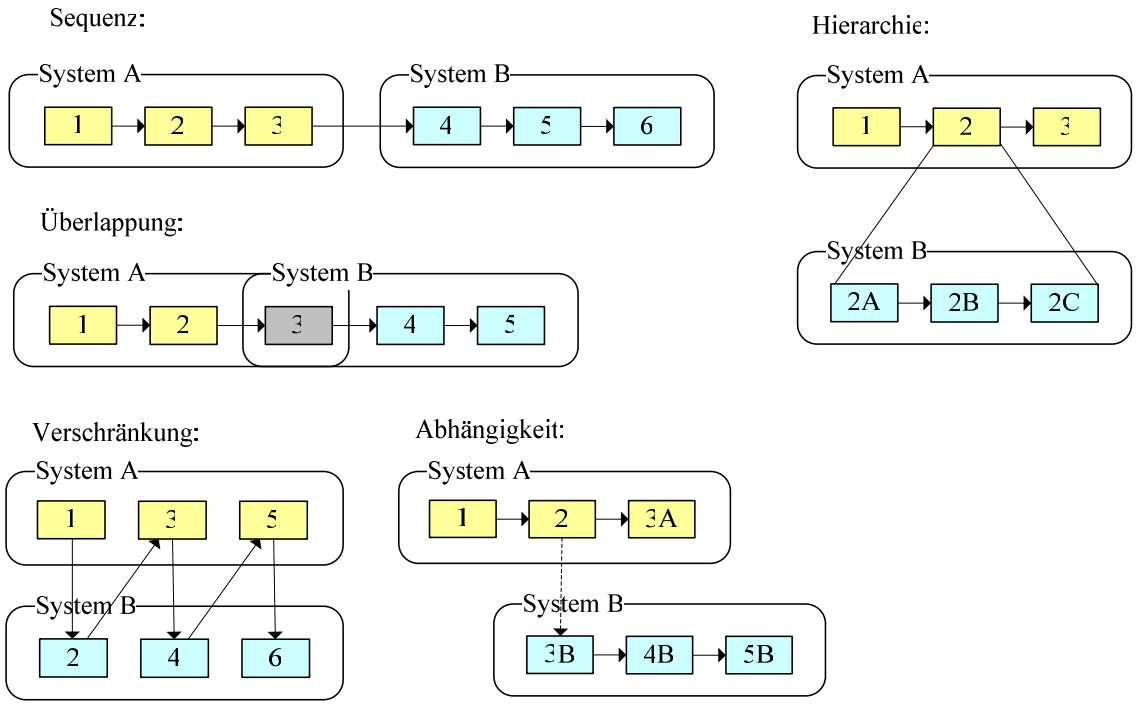


Abbildung 5.1 Beziehungstypen

Die einfachste Beziehung ist die Sequenz, bei der sich die Systeme nacheinander aufrufen. Dabei darf ein System, das von einem anderen System aufgerufen wurde, dieses nicht mehr aufrufen. Ansonsten ist es keine Sequenz mehr, sondern eine Verschränkung.

Bearbeiten mehrere Systeme die gleiche Aufgabe, so nennt man das Überlappung.

Hierarchische Beziehungen gibt es bei Subprozessen. Dabei wird eine grob modellierte Aufgabe eines Systems in einem anderen System detailliert ausgeführt.

Bei der Abhängigkeit wird kein anderes System aufgerufen, sondern es wird geprüft, ob in einem anderen System ein bestimmter Zustand vorliegt, wovon die Ausführung einer Aufgabe abhängt. In Abbildung 5.1 ist die Ausführung von Aktivität 3B vom Zustand der Aktivität 2 abhängig.

5.2 Inkonsistenzen durch unabhängige Modellierungen

Bei großen, system- oder auch unternehmensübergreifenden Prozessen wird die Modellierung der einzelnen Prozessfragmente normalerweise nicht von einem Modellierer allein vorgenommen. So gibt es sowohl in den Unternehmen als auch in den Abteilungen Spezialisten für bestimmte Systeme und Prozesstypen. Die Folge ist, dass die einzelnen Prozessfragmente, die in den Gesamtprozess integriert werden sollen, unterschiedlich detailliert sind und unterschiedliche Konstrukte für denselben Sachverhalt

verwenden. Dadurch erhält man eine inkonsistente Darstellung des Gesamtprozesses. In den folgenden beiden Abschnitten werden Beispiele für diese Probleme gegeben.

5.2.1 Feinheit der Modellierung

Bei der Modellierung von Prozessen kann man keine genaue Detaillierung für die Aktivitäten vorgeben. Es gibt zwar in den Unternehmen oft Richtlinien für die Modellierung, doch diese können nicht alle möglichen Aktivitätenarten oder Arbeitsschritte abdecken.

Auch durch die Definition der WfMC können unterschiedlich fein modellierte Prozesse nicht ausgeschlossen werden. Diese definiert eine Aktivität, als eine Arbeitseinheit, die einen logischen Schritt im Prozess darstellt. Eine Aktivität ist zwar die kleinste Arbeitseinheit im Prozess, die von einem WfMS den Bearbeitern zugewiesen wird, allerdings kann eine Aktivität aus mehreren Arbeitsschritten bestehen [WfMC99a].

Dadurch bleibt es letztendlich dem Modellierer überlassen aus wie vielen Arbeitsschritten eine Aktivität besteht. So kann es vorkommen, dass einige Abläufe bis ins letzte Detail und andere nur grob abgebildet werden.

In Abbildung 5.2 ist der gleiche Prozess links grob und rechts detailliert modelliert. Rechts werden alle Teilaktivitäten von B modelliert, links dagegen nur die gesamte Aktivität.

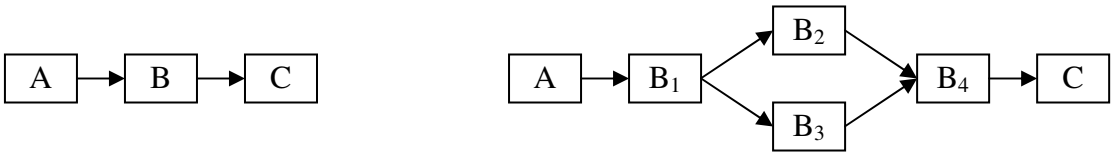


Abbildung 5.2 unterschiedlich fein modellierte Prozesse

5.2.2 Unterschiedliche verwendete Konstrukte

So wie die Feinheit der Modellierung je nach Modellierer anders sein kann, so können auch die verwendeten Konstrukte variieren, vor allem wenn ein Konstrukt vom System nicht direkt unterstützt wird. In so einem Fall muss man, wenn möglich, dieses mit Hilfe anderer darstellen. Dabei gibt es oft mehrere Möglichkeiten. Ein Beispiel dafür wird in Abbildung 5.3 gezeigt. Der erste Teil der Abbildung zeigt das *Multi Choice*–Pattern (s. Pattern 6 Anhang B) in seiner ursprünglichen Form. In diesem Beispiel soll nach Ausführung der Aktivität A Aktivität B ausgeführt werden, falls x<5, Aktivität C falls y>7. Die beiden Bedingungen schließen sich gegenseitig nicht aus, d.h. es können beide, eine von beiden oder keine der Aktivitäten ausgeführt werden.

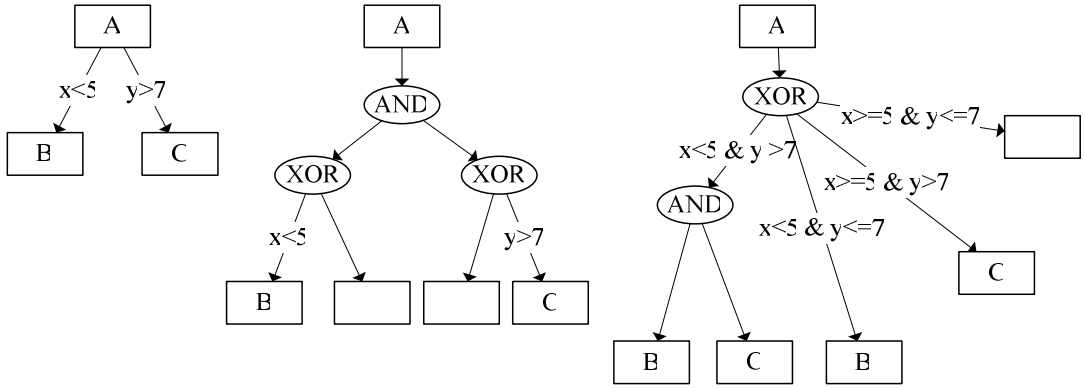


Abbildung 5.3 Abbildungsmöglichkeiten des *Multi Choice* [AHKB03]

Falls dieses Konstrukt von dem jeweiligen System nicht unterstützt wird, dann kann man es, wie in der Abbildung gezeigt, mit Hilfe anderer Konstrukte abbilden. Welche Variante gewählt wird, kann sich von Modellierer zu Modellierer unterscheiden. Bei der Integration der Prozessfragmente sollten solche Konstrukte vereinheitlicht werden, damit es nicht zu Inkonsistenzen kommt.

5.3 Forderungen durch Unterschiedliche Kooperationsgrade

Der Kooperationsgrad von Organisationseinheiten, beispielsweise mehrerer Unternehmen oder Abteilungen, beschreibt wie eng diese zusammenarbeiten. Ein hoher Kooperationsgrad bedeutet eine enge Zusammenarbeit, bei der Prozesse und vielleicht sogar manche Betriebsgeheimnisse offen gelegt werden. Bei niedrigen Kooperationsgraden liegt eher eine Auftraggeber-Auftragnehmer-Beziehung vor, bei der die internen Abläufe geheim bleiben.

Somit muss ein Prozess je nach Kooperationsgrad unterschiedlich abgebildet werden. So darf beispielsweise bei geringer Kooperation ein Teilprozess nur als Subprozess abgebildet werden, ohne dessen detaillierte Aktivitäten. Bei engerer Kooperation können ganze Teile oder der gesamte Teilprozess betrachtet werden.

Abbildung 5.4 zeigt ein Beispiel, bei dem die Details von Aktivität B nicht offen gelegt werden. B wird nur als Subprozess angezeigt.

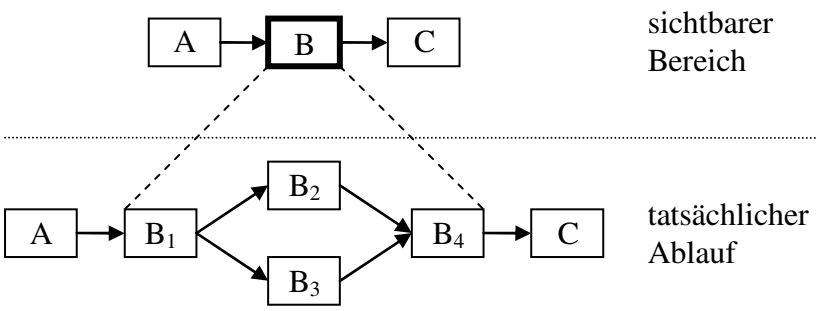


Abbildung 5.4 vom Ablauf abweichende Anzeige

Für dieses Problem gibt es zwei Lösungsansätze. Der erste ist, der Visualisierungskomponente nur diejenigen Daten bereitzustellen, die auch wirklich angezeigt werden dürfen. Somit muss diese Lösung von der Integrationskomponente umgesetzt werden.

Die zweite Lösung ist, alle Daten bereitzustellen und über vordefinierte Sichten (*Views*) die Anzeige einzuschränken. Diese Lösung muss dann komplett von der Visualisierungskomponente umgesetzt werden. Da diese auch noch andere *View*-bezogene Aufgaben durchführen muss, ist diese Lösung somit sinnvoller. Dann können alle *Views* von einer Komponente verwaltet werden. Ausführliches über *Views* (Arten, Vorgehensweisen usw.) findet sich in [Klo04].

5.4 Versteckte Aktivitäten

Versteckte Aktivitäten (*hidden activities*) sind Aktivitäten, die im Modell nicht aufgeführt sind, aber trotzdem durchgeführt werden.

Betrachtet man systemseitig unterstützte Prozesse in der Praxis, so stellt man fest, dass oft Aktivitäten ausgeführt werden, die im System nicht modelliert wurden. Grund dafür ist, dass ein Teil der Informationen und Aufgaben in den Köpfen der Mitarbeiter steckt und diese neben den modellierten Aufgaben automatisch ausgeführt werden. Um bei der Integration der Prozessfragmente ein vollständiges Prozessmodell abbilden zu können, müssen die versteckten Aktivitäten in irgendeiner Form realisiert bzw. gefunden werden. Dadurch, dass sie im System nicht protokolliert sind, ist das allerdings eine problematische Aufgabe. Entscheidend dabei ist vor allem, ob die Aktivitäten teilweise am System oder komplett manuell ausgeführt werden.

5.4.1 Manuell ausgeführte Aktivitäten

Bei Aktivitäten, die beispielsweise auf dem Papier oder am Telefon ausgeführt werden, entstehen und verändern sich Daten dynamisch in den Köpfen der Bearbeiter oder auf dem Papier. Ihre technische Erfassung ist äußerst schwierig oder fast unmöglich, da auf dem System keine Ereignisse stattfinden, die auf irgendwelche Aktivitäten hinweisen

könnten. Ihre Erfassung ist also eher ein organisatorisches als ein technisches Problem. Es wären höchstens technisch unterstützte Lösungen denkbar:

- Beim Start jeder Aktivität im System könnte eine Abfrage kommen, in der man die letzte ausgeführte Aktivität angeben muss. Weicht diese von dem, vom System erfassten Vorgänger ab, dann ist das ein Hinweis auf eine versteckte Aktivität und das Prozessmodell muss überprüft werden.
- Ähnlich könnte man auch auf dem Papier alle ausgeführten Aktivitäten protokollieren, was für die Bearbeiter vermutlich einen höheren Aufwand als mit systemseitiger Unterstützung bedeuten würde.
- Checkliste: alle im System aufgeführten Aktivitäten befinden sich auf einer Checkliste. Nach der Ausführung einer Aktivität wird diese auf der Checkliste abgehakt. Ist die eben ausgeführte Aktivität nicht auf der Liste, so fehlt sie auch im System.
- Beobachtung der Mitarbeiter: Die Mitarbeiter werden bei der Ausführung der Aktivitäten beobachtet und ihre Arbeit wird ständig mit dem Soll-Prozess verglichen.

Problem vor allem beim letzten Punkt ist, dass so eine Beobachtung sehr teuer ist, weil ein Mitarbeiter oder Experte in dieser Zeit nichts anderes macht. Dadurch ist eine Beobachtung nur für kurze Zeit möglich, was dazu führen kann, dass eventuelle Ausnahmen, trotz investiertem Aufwand, doch nicht erfasst werden. Vor allem bei langläufigen Prozessen (z.B. Leasing- oder Kreditvertrag) ist diese Variante nicht durchführbar. Etwas günstiger sind die anderen Möglichkeiten, die allerdings für die Bearbeiter zusätzlichen Aufwand nach sich ziehen und bei diesen somit auf Ablehnung stoßen dürften. In solchen Fällen muss man die Bearbeiter von der Wichtigkeit der Aufgabe überzeugen, damit sie die Ermittlung der versteckten Aktivitäten gründlich durchführen.

Da die verursachten Kosten unabhängig von der gewählten Variante sehr hoch sind, sollten diese nur so lange eingesetzt werden, bis eine unbedingt zu vermeidende Fehlerquote ausgeschlossen werden kann.

5.4.2 Am System ausgeführte Aktivitäten

Für versteckte Aktivitäten, die zumindest teilweise am System ausgeführt werden, gelten auch alle oben aufgeführten Möglichkeiten. Bei diesen gibt es u.U. zusätzlich automatisierte Möglichkeiten. Man könnte Wrapper (s. Abschnitt 2.2.1.3) einsetzen, die ein System überwachen und alle seine Ereignisse (z.B. Veränderungen der Daten) aufzeichnen. Anschließend kann man diese Ereignisse mit den Log-Daten vergleichen. Zusätzliche, durch den Wrapper aufgezeichnete Ereignisse sind ein Hinweis auf versteckte Ak-

tivitäten. Dann muss manuell festgestellt werden, ob wirklich zusätzliche Aktivitäten stattgefunden haben.

Da die einzelnen Systeme, aus unterschiedlichen Gründen, oft nur einen Teil der Ereignisse protokollieren, kann es durchaus vorkommen, dass der Wrapper Ereignisse von Aktivitäten findet, die nicht versteckt sind. Um die mehrfache Abbildung gleicher Aktivitäten zu vermeiden, gibt es zwei Möglichkeiten. Die erste ist, für die gefundenen zusätzlichen Ereignisse zu überprüfen, ob die Aktivität, zu der sie gehören bereits existiert. Die andere ist, die Ereignisse, die durch den Wrapper aufgezeichnet werden, von vorn herein zu filtern, so dass nur nötige Ereignisse aufgezeichnet werden. Damit wird vermieden, dass nicht relevante Ereignisse unnötigen Mehraufwand produzieren.

5.5 Überlappende Aktivitäten

Im Gegensatz zu den versteckten Aktivitäten sind überlappende Aktivitäten sogar öfters vorhanden, obwohl sie den gleichen Kontext haben. In manchen Fällen ist dann eine mehrfache Anzeige nicht erwünscht. Das bedeutet also, dass eine oder mehrere dieser Aktivitäten ausgeblendet werden müssen. Dabei ergeben sich zwei Fragen.

Zum einen muss entschieden werden, welche von den Aktivitäten angezeigt und welche ausgeblendet werden sollen. Dabei sind mehrere Vorgehensweisen denkbar:

- Die einfachste basiert auf der Tatsache, dass die Aktivitäten alle im gleichen Kontext sind. Dadurch spielt es keine Rolle, welche von ihnen angezeigt wird. Somit kann man die erste anzeigen und alle nachfolgenden weglassen.
- Soll allerdings die mehrfache Abbildung aus Gründen der Übersichtlichkeit vermieden werden, ist es sinnvoller, die Struktur des Prozesses zu untersuchen und die Aktivitäten aus den komplexeren Bereichen wegzulassen.
- Eine weitere Möglichkeit, die ebenfalls eine vorherige Untersuchung des Prozesses benötigt, ist, diejenigen Aktivitäten wegzulassen, deren Fehlen die geringsten Auswirkungen auf die Laufzeitdaten, beispielsweise auf die Zustände, hat. Denn lässt man eine bereits laufende oder beendete Aktivität weg, so kann das Auswirkungen auf den Zustand der nachfolgenden Aktivität haben. Dies wird in Abbildung 5.5 an einem Beispiel gezeigt. Im ersten Fall wird eine bereits beendete Aktivität weggelassen. Auch nach dem Weglassen der Aktivität bleibt der Zustand von Aktivität C korrekt. Im zweiten Fall dagegen wird eine laufende Aktivität weggelassen. Dadurch ist die nachfolgende Aktivität, also Aktivität C weder aktiviert noch laufend. Durch das Weglassen von B müsste aber C aktiviert sein, da der Vorgänger bereits erledigt ist. Somit ist der entstandene Zustand inkorrekt, was entweder angepasst oder angemerkt werden muss.

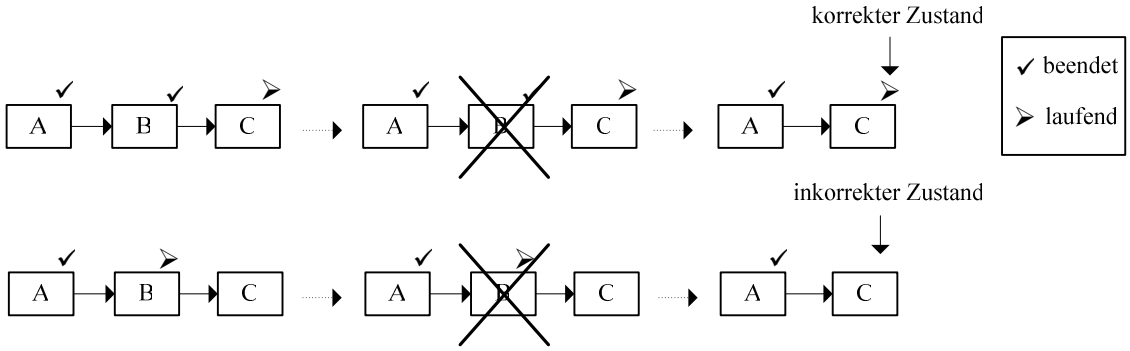


Abbildung 5.5 Inkonsistenzen durch Weglassen überlappender Aktivitäten

Zum anderen stellt sich wieder die Frage, wie das Problem gelöst werden soll. Auch hier bieten sich die beiden Möglichkeiten wie in Abschnitt 5.3, entweder die Datenmenge zu bearbeiten und somit bereits auf Datenebene die nicht benötigten Aktivitäten wegzulassen oder aber über Sichten erst bei der Anzeige zu entscheiden welche Aktivitäten angezeigt werden sollen, an.

5.6 Mehrstufige hierarchische Prozesse

Hierarchische Prozesse sind Prozesse, deren Bestandteile oder zumindest ein Bestandteil ebenfalls ein Prozess ist. Die Prozesse, die die Bestandteile darstellen, werden Subprozesse genannt. Die Hierarchie kann sich über mehrere Stufen erstrecken. Dann haben die Subprozesse ebenfalls Subprozesse. Dies wird in Abbildung 5.6 verdeutlicht.

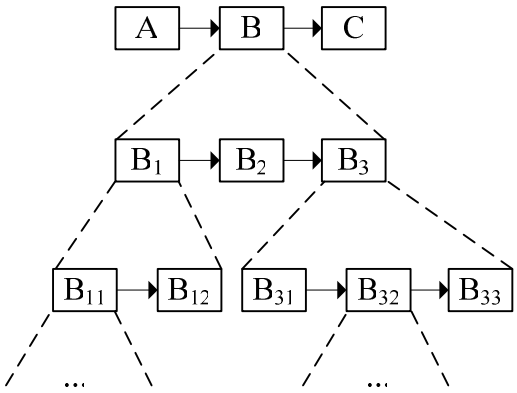


Abbildung 5.6 mehrstufiger hierarchischer Prozess

Bei solchen Prozessen muss man zunächst entscheiden, ob die Hierarchie abgebildet werden soll oder nicht. Falls die Abbildung erwünscht ist, so stellt sich zusätzlich die Frage, wie viele Hierarchiestufen abgebildet werden sollen. Dabei muss betrachtet werden, welcher Detaillierungsgrad für die Visualisierung gewählt wurde und die Anzahl

der abzubildenden Stufen so gewählt werden, dass eine konsistente Abbildung gewährleistet bleibt.

Auch hier können *Views* zur konsistenten Visualisierung eingesetzt werden.

Falls die einzelnen Stufen von unterschiedlichen Systemen ausgeführt werden, ist die Beschaffung der Daten nicht ganz trivial. Denn die einzelnen Systeme stellen immer nur ihre eigenen Daten bereit. Dadurch fehlen die detaillierten Daten zu den Subprozessen. Um diese zu beschaffen muss rekursiv vorgegangen werden, indem bei der Anforderung der Daten jedes System auf einer bestimmten Hierarchiestufe die Daten der Systeme auf der nächst niedrigeren Hierarchiestufe anfordert.

5.7 Zusammenfassung

In diesem Kapitel wurden Aufgaben und Probleme aufgezeigt, die bei der Integration von Prozessfragmenten zu einem einheitlichen Prozess bewältigt werden müssen und es wurden Konzepte zu ihrer Lösung vorgestellt. Die erste Aufgabe ist die Identifikation von Beziehungen zwischen den Prozessfragmenten. Dabei gibt es verschiedene Beziehungsarten, von einfachen sequentiellen Beziehungen bis hin zu hierarchischen Beziehungen. Bei hierarchischen Prozessen kann es vorkommen, dass diese aus mehreren Hierarchiestufen bestehen. Dann müssen die Daten aller hierarchisch untergeordneten Systeme beschafft werden und es muss auf einen einheitlichen Detaillierungsgrad bei der Abbildung geachtet werden. Auf eine einheitliche Abbildung muss auch wegen unabhängiger Modellierung durch unterschiedliche Modellierer geachtet werden, wodurch gleiche Sachverhalte unterschiedlich fein oder mit unterschiedlichen Konstrukten modelliert sein können. Zudem muss je nach der Intensität der Zusammenarbeit zwischen den beteiligten Organisationseinheiten entschieden werden, welche Details eines Prozesses offen gelegt werden. Ein ähnliches Problem, bei dem ebenfalls gewisse Aktivitäten ausgeblendet werden sollen, ergibt sich durch überlappende Aktivitäten, die im Prozess öfter vorkommen, aber nur einmal angezeigt werden sollen. Schließlich gibt es noch, im Gegensatz zu jenen Aktivitäten, versteckte Aktivitäten, die zwar ausgeführt werden, aber im System nicht vermerkt sind und somit zunächst identifiziert werden müssen.

6 Instanzdatenintegration

Hat man ein einheitliches Prozessmodell erzeugt, so muss dieses noch mit Laufzeitdaten gefüllt werden. Dazu müssen die Applikations-, Log- und Workflow relevanten Daten in das Prozessmodell integriert werden. Auch hier ist das erste Problem, dass die Daten normalerweise in unterschiedlichen Datenquellen und Formaten vorliegen. Aber auch hier kann man von diesen Problemen abstrahieren (s. Kapitel 3 für Lösungen). Bei der eigentlichen Instanzdatenintegration treten die ersten Probleme bei der Zuordnung zusammengehöriger Instanzen zueinander auf, auch Korrelation genannt (s. Abschnitt 6.1). Nach der Korrelation gibt es, wie bei den Modelldaten, jede Menge Konflikte zu beseitigen. Zum einen treten Konflikte auf, die durch Änderungen der Prozessmodelle während Transformation und Integration entstanden sind (s. Abschnitt 6.2). Zum anderen gibt es auch bei den Laufzeitdaten ähnliche Heterogenitäten wie bei den Modelldaten, die in Abschnitt 6.3 behandelt werden. Am Ende des Kapitels, in Abschnitt 6.4, werden die Probleme anhand der Integration von Zuständen konkret aufgezeigt.

6.1 Korrelation instanzspezifischer Daten

Bei der Korrelation instanzspezifischer Daten geht es um das Problem, die Instanzen der einzelnen Teilprozesse und somit auch ihre Daten einander richtig zuordnen zu können.

Werden in einem verteilten Prozess Prozessfragmente auf unterschiedlichen Systemen gleichzeitig ausgeführt, dann reicht ein üblicher Identifikator, wie z.B. der Name oder eine ID, für diese und die dazugehörigen Daten nicht aus. Denn ein lokal eindeutiger Name (z.B. „Dokument prüfen“) oder eine lokale ID sind global meist nicht eindeutig.

Ähnliche Probleme treten in BPEL4WS (kurz: BPEL) auf. BPEL4WS bedeutet *Business Process Execution Language for Web Services* und erlaubt auf Basis von WSDL (*Web Service Definition Language*) die Verknüpfung von Web Services zu Geschäftsprozessen. Da BPEL-Prozesse meist verteilt ablaufen und von mehreren Aufrufern genutzt werden, müssen sich diese eindeutig identifizieren lassen, um Nachrichten den korrekten Sitzungen zuordnen zu können. Für die Identifikation werden so genannte Korrelationsmengen benutzt, die einmal initialisiert werden und danach für alle weiteren Kommunikationen mit einem Dienst die eindeutige und korrekte Zuweisung der Nachricht zu einer Sitzung ermöglichen. Die Korrelationsmengen sind Mengen von Eigenschaften, die die konstanten Identifikationsmerkmale von Nachrichten bilden. Solche Eigenschaften können beispielsweise eine Kundennummer, eine Bestellnummer oder auch ein Name sein [BIM+03, BAE04].

Zusammengefasst besteht eine Korrelationsmenge aus einer Menge von instanzspezifischen Daten, die einen Prozess eindeutig identifizieren.

Ein ähnliches Konzept könnte man auch beim Mapping systemübergreifender Prozesse verfolgen und die Aktivitätsinstanzen mit Hilfe einer Menge von Applikationsdaten identifizieren.

Abbildung 6.1 verdeutlicht noch einmal das Problem. Betrachtet man die Instanzen der beiden Prozesse, die zu dem Gesamtprozess zusammengefügt werden, so kann man ohne Applikationsdaten nicht eindeutig sagen welche Instanzen zusammengehören. Nimmt man die Teilenummer dazu, die von beiden Systemen verwendet wird, so ist die Zuordnung eindeutig. In manchen Fällen kann es sein, dass eine Nummer allein nicht ausreicht, weil sie vielleicht in vielen anderen Instanzen auch benutzt wird. In solchen Fällen muss man noch weitere Daten, wie z.B. Bestellnummer, Kundennummer usw., zur Korrelationsmenge hinzufügen, bis diese eindeutig wird.

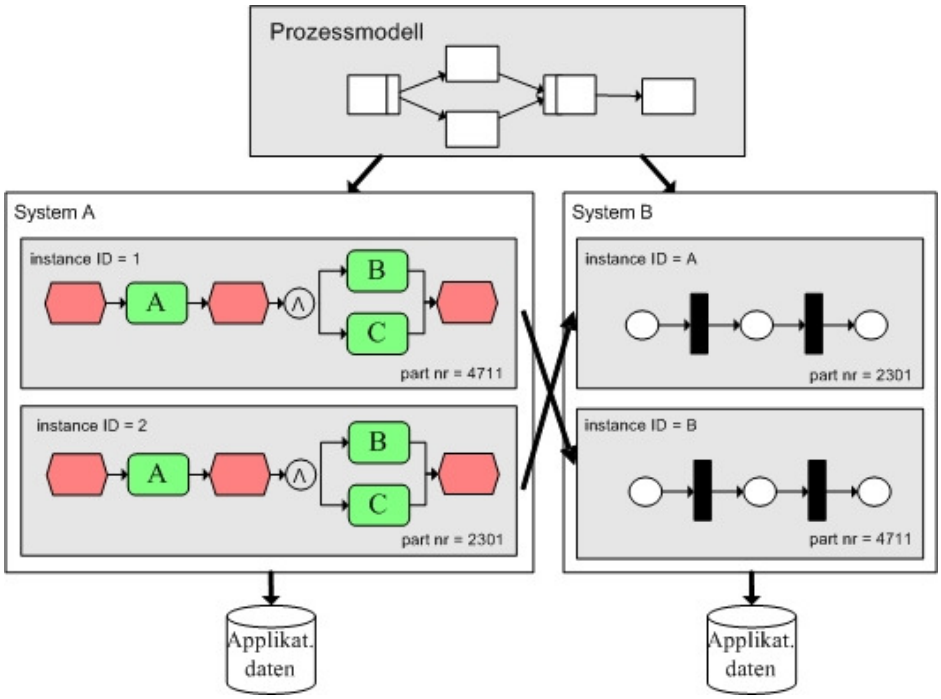


Abbildung 6.1 Korrelation mit Hilfe von Applikationsdaten

6.2 Konflikte durch Änderungen bei der Prozesstransformation und -integration

Wie wir in den Kapiteln 4 und 5 gesehen haben, kann sich nach der Transformation und Integration der Prozess enorm ändern. Das wirkt sich auch auf die Integration der Instanzdaten aus, die zu dem ursprünglichen Prozess gepasst haben. Nach den Änderungen des Prozessmodells können diese nämlich namentlich, strukturell und semantisch

von den zugehörigen Modellelementen abweichen. Diese Probleme werden im Folgenden genauer betrachtet.

6.2.1 Probleme durch Namensänderungen

Durch die Beseitigung von Synonymen und Homonymen (s. Abschnitt 4.2), sowie durch Änderungen von Schlüsseln (s. Korrelation), werden Applikations-, Log- und Workflow relevante Daten u.U. falschen Instanzen oder Elementen zugeordnet.

Dieses Problem kann gelöst werden, indem man bei den Änderungen der Namen der Modelldaten eine Schlüssel- bzw. Namensumsetzungstabelle benutzt. Das Prinzip des Verfahrens wird in Tabelle 6.1 verdeutlicht. Ein Eintrag ist immer ein 3-Tupel, der aus dem alten und dem neuen Bezeichner und dem System des alten Bezeichners besteht. Dadurch können sowohl der alte Name im jeweiligen Quell-System oder der neue Name im integrierenden System ermittelt werden, je nachdem welcher benötigt wird. Durch die Benutzung eines 3-Tupels können in einer einzigen Tabelle die Änderungen der Prozesse aller Systeme aufgenommen werden.

System	alter Name	neuer Name
System A	Funktion X	Aktivität X
System B	leitende Funktion	leitende Position
System C	Kunde	Klient
…	…	…

Tabelle 6.1 Namensumsetzungstabelle

Eine andere Möglichkeit ist, eine Tabelle pro Quell-System zu benutzen. Dann ist durch die Zuordnung der Tabelle zu einem bestimmten System ein 2-Tupel als Eintrag ausreichend.

Welche Realisationsvariante besser ist, hängt vom jeweiligen Szenario ab. Bei vielen Quellsystemen und wenigen Änderungen ist eine globale Umsetzungstabelle besser, als viele kleine Tabellen mit wenigen Einträgen. So wird weniger Overhead erzeugt. Bei wenigen Quellsystemen mit vielen Änderungen sind lokale Tabellen für jedes einzelne System geschickter, da der Zugriff auf diese schneller erfolgen kann als auf eine einzige Tabelle mit sehr vielen Einträgen.

Welche Variante wirklich besser ist, muss aber im jeweiligen Szenario, abhängig von den Zielen und gegebenen Systemen, entschieden werden.

6.2.2 Konflikte durch Strukturänderungen

Gliedert man die strukturellen Änderungen am Prozessmodell nach ihrer Relevanz für die Instanzdaten, so kommt man auf drei Arten von Änderungen. Es gibt andere Konstrukte, mehr Konstrukte oder weniger Konstrukte, als vor der Änderung. Die Ursachen und ihre Auswirkungen auf die Instanzdaten werden in den folgenden Abschnitten behandelt.

6.2.2.1 Andere Konstrukte

Dass nach der Transformation und Integration eines Prozessmodells andere Konstrukte als zuvor benutzt werden, liegt zunächst hauptsächlich an den Heterogenitäten zwischen den Metamodellen und den, durch die Systeme umgesetzten, Konstrukten. So werden beispielsweise Schleifen anders umgesetzt oder bisher existierende Verzweigungen müssen anders dargestellt werden, wie an dem Beispiel vom OR-Split in Staffware gezeigt (s. Abschnitt 4.1.1). Das wirkt sich natürlich auf die Log- und Workflow relevanten Daten aus. Je nachdem welches System das Quellsystem ist, ergeben sich unterschiedliche Problemstellungen. Dies soll im Folgenden, aufbauend auf dem angesprochenen Beispiel (s. Abbildung 6.2), verdeutlicht werden.

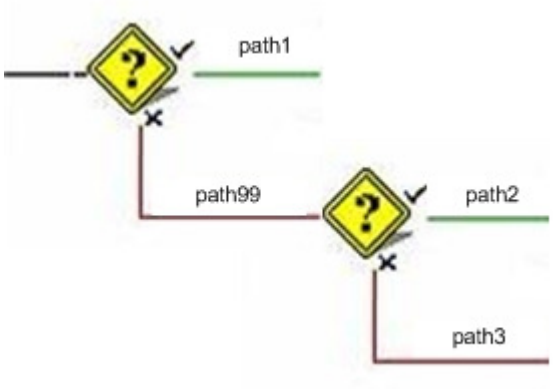


Abbildung 6.2 Darstellung eines OR-Splits in Staffware

Angenommen, das Zielsystem sei Staffware und das Quellsystem unterstützt das OR-Split-Konstrukt direkt. Dann gibt es an dem Entscheidungspunkt im Quellsystem beispielsweise folgende Bedingung:

```
„if (x<5) choose path1
   else if (x=5) choose path2
      else chose path3;”
```

In Staffware müsste es dafür zwei Entscheidungspunkte geben, an denen folgende Bedingungen gelten würden:

- am ersten Entscheidungspunkt:

„if (x<5) choose path1
else choose path99“

- am zweiten Entscheidungspunkt an „path99“:

„if (x=5) choose path2
else choose path3“

Es ist offensichtlich, dass in so einem Fall die Log-Daten geändert werden müssen. Denn im ersten Fall gibt es nur für eine Verzweigung Log-Daten, bei der beispielsweise der Wert der Entscheidungsvariablen und der eingeschlagene Pfad gespeichert werden:

„decision_point1: x=10, path3 taken;“

Im zweiten Fall müssten die entsprechenden Log-Daten folgendermaßen aussehen:

„decision_point1: x=10, path99 taken;
decision_point2: x=10, path3 taken;”

Somit müssen auch die Log-Daten transformiert werden, um bei der Abbildung richtig zugeordnet werden zu können. In diesem Fall besteht die Transformation aus Erweiterungen um einen Hilfspfad („path99“) und einen zusätzlichen Eintrag für den zweiten Entscheidungspunkt.

Vertauscht man Quell- und Zielsystem, dann ist die Grundproblematik die Gleiche, nur dass die Transformation dann in die andere Richtung geht. Es müssen also Einträge zusammengefasst werden.

Zu ähnlichen Problemen kommt es auch, wenn man bei der Integration der Modelldaten Inkonsistenzen beseitigt, die durch unterschiedliche Modellierer entstanden sind (s. Abschnitt 5.2), wie beispielsweise unterschiedliche Konstrukte für denselben Sachverhalt.

6.2.2.2 Mehr Konstrukte

Mehr Konstrukte im Zielmodell als im Quellmodell können entstehen, wenn man Subprozesse des Quellmodells im Zielmodell in einzelne Aktivitäten auflöst, den Detaillierungsgrad der Modellierung erhöht, versteckte Aktivitäten aufdeckt oder ein nicht vorhandenes Prozessmodell aus den Applikations- und/oder Log-Daten herstellt.

Wenn man diese vier Fälle betrachtet, dann fällt auf, dass man bei den ersten beiden vorhandene Konstrukte lediglich erweitert. Bei den letzten beiden kommen neue Konstrukte hinzu.

Findet eine Erweiterung statt, dann muss man entsprechend auch die Log- und die Workflow relevanten Daten erweitern. Bei der Auflösung von Subprozessen in ihre Aktivitäten, muss man beispielsweise den Log-Eintrag oder den Zustand dieses Subprozesses in ebenso viele Teile aufteilen. Dabei ergeben sich Probleme, wenn der betroffene

Subprozess gerade läuft, da in diesem Fall gewisse Aktivitäten bereits abgeschlossen sind und manche noch gar nicht gestartet wurden.

Kommt bei der Änderung ein neues Konstrukt hinzu, dann muss man wieder unterscheiden, ob es eine versteckte Aktivität ist, die manuell hinzugefügt wurde oder ob es ein automatisch generiertes Prozessmodell ist.

Denn bei einer manuell eingefügten Aktivität gibt es normalerweise keinerlei oder nicht ausreichende Instanzdaten, ansonsten hätte man mit Hilfe von Process Mining das Modell automatisch generieren können. In so einem Fall müssen auch die Laufzeitdaten manuell eingefügt werden. In manchen Fällen können bestimmte Daten aus dem Zustand des Prozesses abgeleitet werden (s. Abschnitt 6.4.1.2).

Bei einem automatisch generierten Modell, sind normalerweise genügend Log-Daten vorhanden, um mit Hilfe dieser und eventuell der Applikationsdaten, fehlende Daten, wie z.B. Zustände, zu erzeugen.

6.2.2.3 Weniger Konstrukte

Weniger Konstrukte im Zielmodell als im Quellmodell können entstehen, wenn man Aktivitäten des Quellmodells im Zielmodell in einen Subprozess zusammenfasst, den Detaillierungsgrad der Modellierung vergröbert oder überlappende Aktivitäten weglässt.

Dadurch, dass im Prozessmodell weniger abgebildet wird, sind diese Änderungen einfacher zu handhaben, als die aus den vorherigen Abschnitten.

Bei den ersten beiden Fällen muss man lediglich eine Menge von Daten zusammenfassen.

Und im letzten Fall, wenn man überlappende Aktivitäten weglässt, genügt es die entsprechenden Laufzeitdaten wegzulassen. An dieser Stelle muss man, wenn möglich, Zustandsanpassungen vornehmen, falls die weggelassene Aktivität in einem Zustand war, dessen Fehlen die Zustände der nachfolgenden Aktivitäten inkonsistent erscheinen lässt.

6.2.2.4 Zusammenfassung

Zusammengefasst kann man sagen, dass es durch die Vielzahl der Probleme und ihrer Variationen, keine standardisierte Lösung geben kann. Je nach zu integrierenden Metamodellen, müssen spezielle Transformations- und Integrations-Regeln festgelegt werden. Die Anwendung dieser Regeln muss von den tatsächlich durchgeführten Änderungen bei der Transformation und Integration abhängig gemacht werden.

6.2.3 Konflikte durch semantische Änderungen

Durch die Transformation der Modelldaten können sich bei diesen auch semantische Änderungen ergeben. Diese Änderungen beeinträchtigen allerdings die Integration der Instanzdaten zunächst nicht. Die rein strukturelle Integration wäre nach der Beseitigung der Probleme aus den Abschnitten 6.2.1 und 6.2.2, durch die Anpassung der Instanzdaten möglich. Allerdings würden sie semantisch nicht zu den Modelldaten passen, zu denen sie gehören.

Da aber die Instanzdaten, genauso wie die Modelldaten, jede Menge semantische Heterogenitäten aufweisen können, müssen die Instanzdaten der verschiedenen Systeme ebenfalls auf ein gemeinsames Metamodell abgebildet werden. Führt man diese Transformationen analog zu denen der Modelldaten durch, so sind die semantischen Konflikte zwischen Modelldaten und dazugehörigen Instanzdaten automatisch beseitigt.

Semantische und weitere Heterogenitäten bei Instanzdaten werden im folgenden Abschnitt aufgezeigt.

6.3 Heterogenitäten bei Instanzdaten

Heterogenitäten können bei allen Instanzdaten auftreten, weil je nach System unterschiedliche Daten gespeichert werden, die auch noch unterschiedliche Bedeutung haben können. In den folgenden Abschnitten werden die Heterogenitäten zusammen mit den Konflikten, die sie verursachen können, aufgezeigt.

6.3.1 Applikationsdaten

Die Unterschiede bei den Applikationsdaten sind offensichtlich. Je nach Applikationssystem kann man unterschiedliche Daten haben (z.B. CAD vs. Word-Dokument). Die semantischen Heterogenitäten bei den Applikationsdaten kann und braucht man nicht zu beseitigen, da die Applikationsdaten unterschiedlichen Zwecken dienen und auch unterschiedliche Bedeutungen haben müssen. Somit sind diese Heterogenitäten normal. Die Applikationsdaten müssen lediglich soweit integriert werden, dass zum einen auf diese in einheitlicher Form zugegriffen werden kann, und dass sie zum anderen in der Visualisierungskomponente in irgendeiner Form angezeigt werden können.

Der einheitliche Zugriff ist mit den Technologien zur Datenintegration (aus Kapitel 3) möglich und ist nötig, wenn Applikationsdaten beispielsweise für die Korrelation instanzspezifischer Daten benötigt werden (s. Abschnitt 6.1).

Bei der Visualisierung kann man, je nach Bedarf, lediglich das Vorhandensein der Daten oder aber die kompletten Daten mit Hilfe des jeweiligen Applikationssystems anzeigen. Der Aspekt, in welcher Form Daten visualisiert werden, ist allerdings nicht Thema dieser Arbeit.

6.3.2 Log-Daten

Je nachdem, was die Entwickler eines Systems für wichtig erachten, werden in den einzelnen Systemen unterschiedliche Ereignisse protokolliert. Die aufgezeichneten Ereignisse können sich sowohl in ihrer Anzahl und Bedeutung als auch in ihrer Granularität unterscheiden. Dies wird im Folgenden anhand der Log-Daten zweier WfMSe (Staffware und MQSeries Workflow) verdeutlicht.

Die quantitativen Unterschiede bei den gespeicherten Log-Daten werden offensichtlich, wenn man die folgenden zwei Tabellen miteinander vergleicht. Tabelle 6.2 zeigt einen Ausschnitt der MQSeries Workflow Log-Daten. (Die komplette Tabelle ist im Anhang C zu finden.) Tabelle 6.3 zeigt alle von Staffware gespeicherten Log-Daten.

Schon der Ausschnitt der MQSeries Log-Daten zeigt mehr Attribute pro Log-Eintrag, als es in Staffware insgesamt gibt. Wenn man die komplette Tabelle von MQSeries im Anhang betrachtet, sieht man, dass den 6 angezeigten Staffware-Attributen 25 von MQSeries gegenüberstehen.

Field Name	**Column name of database table**	**Type**	**Explanation**
Timestamp (DB2)	CREATED	TIMESTAMP Mandatory	Date and time the audit trail record is written.
Timestamp (Oracle)	CREATED	TIMESTAMP_ WF Mandatory	Date and time the audit trail record is written.
Event	EVENT	INTEGER Mandatory	Type of event as indicated in Table 5 on page 76.
Process Name	PROCESS_NAME	VARCHAR (63) Mandatory	Name of the process instance.
Process Identifier	PROCESS_ID	IDENTIFIER Mandatory	Object identifier of the process instance.
Top-level Name	TOP_LVL_PROC_NAME	VARCHAR (63) Mandatory	Name of the top-level process instance if the process instance is executing as subprocess, or the same as in process name if the process instance is a top-level process instance.
Top-level Identifier	TOP_LVL_PROC_ID	IDENTIFIER Mandatory	Object identifier of the top-level process instance if the process is executing as subprocess, or the same as in process identifier if the process instance is a top-level process instance.
Parent Process Name	PARENT_PROC_NAME	VARCHAR (63) Optional	Name of the parent process instance if the process instance is executing as a subprocess.
Parent Process Identifier	PARENT_PROC_ID	IDENTIFIER Optional	Object identifier of the parent process instance if the process instance is executing as a subprocess.
Process Model Name	PROC_TEMPL_NAME	VARCHAR (32) Mandatory	Name of the process model.

Tabelle 6.2 Ausschnitt der MQSeries Logdaten [IBM04]

Case Number	eindeutige Instanznr., die von Staffware vergeben wird, sobald ein Instanz gestartet wird
Case Description	kurze Beschreibung der Instanz. Wird normalerweise von der Person vergeben, die die Instanz startet
Case Reference	wird von Staffware vergeben. Besteht aus zwei Zahlen: die erste gibt die Prozedur an, die zweite die Instanz
Started by	beschreibt den Starter der Instanz. Besteht aus zwei Teilen: aus einem Benutzernamen und den Namen des Staffware-Servers, von dem aus die Instanz gestartet wurde
Date/Time	Datum und Zeit der Ausführung
Action	Die Aktion, die stattgefunden hat. Der erste Eintrag ist immer "Case started by…". Die folgenden Einträge beinhalten den Namen des jeweiligen Schrittes, gefolgt von der Aktion und dem Namen der Benutzers, dem der Schritt zugewiesen wurde oder der den Schritt veranlaßt hat. (der Knoten des Staffware Servers, an dem der Benutzer registriert ist, ist auch angegeben) "Case terminated normally" oder "Case terminated prematurely by..." ist der letzte Eintrag. (ja nachdem ob die Instanz normal beendet wurde oder vom Administrator abgebrochen wurde)

Tabelle 6.3 Staffware Log-Daten [Sta00]

Durch diese beiden Tabellen werden auch semantische Unterschiede sofort offensichtlich. Denn MQSeries Workflow speichert alle Daten, die Staffware auch speichert und weitere, die in Staffware nicht bekannt sind. So gibt es beispielsweise in MQSeries einen Eintrag „Parent Process ID“, der die ID des übergeordneten Prozesses enthält, falls der Prozess als Subprozess ausgeführt wird, in Staffware dagegen nicht.

An dieser Stelle stellt sich allerdings die Frage, welche Daten überhaupt benötigt werden. Für Process Mining beispielsweise werden weniger Daten benötigt als für die Visualisierung.

Will man die Daten für Process Mining benutzen, dann reichen die Daten von Staffware durchaus aus.

Case 1

Diractive	Description	Event	User	yyyy/mm/dd hh:mm
		Start	bvdongen@staffw_e	2002/04/18 09:05
A		Processed To	bvdongen@staffw_e	2002/04/18 09:05
A		Released By	bvdongen@staffw_e	2002/04/18 09:05
B		Processed To	bvdongen@staffw_e	2002/04/18 09:05
B		Released By	bvdongen@staffw_e	2002/04/18 09:05
D		Processed To	bvdongen@staffw_e	2002/04/18 09:05
E		Processed To	bvdongen@staffw_e	2002/04/18 09:05
D		Released By	bvdongen@staffw_e	2002/04/18 09:06
E		Released By	bvdongen@staffw_e	2002/04/18 09:06
G		Processed To	bvdongen@staffw_e	2002/04/18 09:06
G		Released By	bvdongen@staffw_e	2002/04/18 09:06
I		Processed To	bvdongen@staffw_e	2002/04/18 09:06
I		Released By	bvdongen@staffw_e	2002/04/18 09:06
		Terminated		2002/04/18 09:06

Abbildung 6.3 Staffware Log (Beispiel) [EMiT]

Denn aus Staffware-Logs, wie in Abbildung 6.3, kann, falls diese von ausreichend vielen Instanzen eines Prozesses vorliegen, der in Abbildung 6.4 gezeigte Prozess erzeugt werden. Die benötigten Daten hierfür sind in Staffware vorhanden (s. Abschnitt 2.2.1.3). Diese müssen nur in die passende Form gebracht werden, denn die Process-Mining-Tools, wie EMiT oder LittleThumb, brauchen als Input XML-Dateien mit der passenden DTD. Die Transformation in die passende DTD ist nichts anderes als ein Mapping auf ein kanonisches Modell.

Somit ist es egal, ob die Systeme viele oder gerade mal für das Process Mining ausreichende Daten speichern.

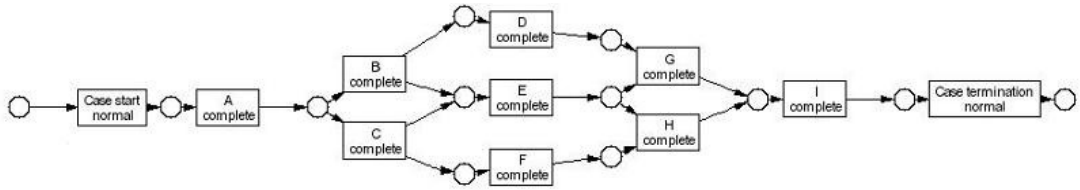


Abbildung 6.4 aus Log-Daten erzeugtes Prozessmodell [EMiT]

Will man die Daten allerdings für die Visualisierung des bisherigen Verlaufs benutzen, dann muss zunächst entschieden werden, welche Daten überhaupt visualisiert werden sollen. Die nötigen Daten müssen dann auf das entsprechende Zwischenmodell, analog den Modelldaten, abgebildet werden.

Die letzte, der anfangs angesprochenen drei verschiedenen Heterogenitäten bei Daten, die Granularität soll anhand der Zustände, gezeigt werden, die bei den Log-Daten mitgespeichert werden. Dazu zeigen die folgenden beiden Tabellen die möglichen Zustände der beiden Systeme. Dabei fällt auf, dass MQSeries Workflow 16 mögliche Zustände hat, Staffware dagegen nur fünf. Hinzu kommt, dass in Staffware die Zustände ganz anders sind. Denn in Staffware gibt es nicht die üblichen Zustände, stattdessen werden diese durch Symbole repräsentiert, deren Bedeutungen kaum denen anderer WfMSe entsprechen. Wie die Symbole intern gespeichert werden, kann aus den Staffware Manuals nicht entnommen werden, da von der Tibco Ltd. (dem aktuellen Staffware-Eigentümer) keinerlei interne Daten offen gelegt werden.

Betrachtet man beispielsweise den Fall, dass eine Aktivität nicht mehr ausgeführt werden kann, so stellt man einen enormen Unterschied fest. In Staffware ist dieser Zustand implizit gegeben, indem eine beendete Aktivität nicht mehr in der Arbeitsliste angezeigt wird. Der Grund dafür ist nicht bekannt. In MQSeries dagegen gibt es dafür mehrere Zustände: „finished“, „force-finished“, „terminated“, „executed“, „deleted“ und „expired“. Somit ist diese Information viel feingranularer als in Staffware.

Code	Activity State
21200	Ready
21201	Running
21202	Finished
21203	CheckedOut
21204	Force-Finished
21205	Terminated
21206	Suspended
21207	InError
21208	Executed
21209	Skipped
21210	Deleted
21211	Suspending
21212	Terminating

Tabelle 6.4 MQSeries Zustände [IBM04]

gelbe Seite	Das Workitem wurde seitdem es in der Arbeitsliste angezeigt wird noch nicht geöffnet. Zusätzlich wird der zugehörige Text in der Workitem-Liste blau angezeigt.
weiße Seite	Das Workitem wurde bereits geöffnet, aber wieder zurückgestellt.
Briefumschlag	Dieses Workitem kann direkt aus der Arbeitsliste freigegeben werden.Eine Bearbeitung durch den Benutzer ist nicht nötig.
Schloss	Das Workitem ist gesperrt und der Benutzer kann nicht mehr darauf zugreifen. Dies ist normalerweiser der Fall, wenn das Workitem bereits von einer anderen Person göffnet wurde. Zusätzlich wird der zugehörige Text in der Workitem-Liste grau angezeigt.
weiße Seite mit rotem Kreuz	Das Workitem ist nicht mehr verfügbar. Dies ist der Fall wenn das Workitem bereits von einem anderen Benutzer freigegeben wurde.

Tabelle 6.5 Staffware „Zustände“ [Sta00]

6.3.3 Workflow relevante Daten

Bei den Workflow relevanten Daten sind die Heterogenitäten bedingt durch andere Daten. Zum einen sind sie ein Teil der Applikations- und der Log-Daten und können somit die in den zwei vorangegangenen Abschnitten diskutierten Heterogenitäten aufweisen.

Zum anderen sind sie sowohl vom benutzten Prozess- als auch vom Metamodell, auf dem sie basieren, abhängig. Heterogenitäten bei den Workflow relevanten Daten, die durch das Prozessmodell bedingt sind, können beispielsweise durch unterschiedliche Verzweigungskonstrukte entstehen, wie in Abschnitt 6.2.2.1 vorgestellt. Weitere Workflow relevante Daten, wie z.B. die Zustände, können, bedingt durch das Metamodell, viele weitere Heterogenitäten aufweisen. Diese werden im folgenden Abschnitt an einem ausführlichen Beispiel aufgezeigt.

6.4 Beispiel - Zustände

In diesem Abschnitt werden am Beispiel der Zustände weitere Probleme aufgezeigt, die bei der Integration von Instanzdaten auftreten können. Die Probleme beginnen schon bei der Speicherung der Zustände, die explizit oder implizit sein kann. Nachdem die Zustände vorliegen, muss auch hier überlegt werden, welches Zwischenmodell (s. Abschnitt 2.2.2) beim Mapping der Zustandsmodelle benutzt wird. Die Vor- und Nachteile der möglichen Modelle werden in diesem Abschnitt abgewogen.

6.4.1 Speicherung der Zustände

Der Zustand einer Instanz des Gesamtprozesses hängt von den Zuständen der jeweiligen Instanzen der Prozessfragmente ab. Diese werden je nach System explizit gespeichert oder müssen mit Hilfe anderer Daten ermittelt werden (implizite Zustände). Abhängig davon unterscheiden sich die Daten, die man für die Ermittlung und Darstellung des Zustandes des Gesamtprozesses benötigt.

6.4.1.1 Explizite Zustände

Bei explizit gespeicherten Zuständen können die Zustände aus der Instanz gelesen und abgebildet werden. Hierbei müssen u.U. analoge Transformationen wie beim Prozessmodell oder bei Log-Daten vorgenommen werden.

6.4.1.2 Implizite Zustände

Speichert ein System keine Zustände (z.B. die Systemklassen 1 und 2), so müssen diese aus den vorliegenden Log- und Modelldaten ermittelt werden. Hierzu sind folgende Daten nötig:

- Start- und Endereignisse von Aktivitäten
- Ein- und Ausgangssemantik der Aktivitäten (z.B. *all-in* beim AND-Join oder *all-out* beim AND-Split)
- Vorgänger und Nachfolger aller Aktivitäten

Aus diesen Daten können dann ausgehend vom Startknoten die Zustände der einzelnen Aktivitäten ermittelt werden.

So können beispielsweise bei der Sequenz aus Abbildung 6.5, abhängig von den Log-Daten, die Zustände aus

Tabelle 6.6 abgeleitet werden. Bei einer Sequenz reicht es, das letzte Ereignis zu betrachten. So ist beispielsweise bei dem Ereignis „B started“ klar, dass Aktivität A beendet sein muss (Zustand „COMPLETED“), dass B im Zustand „RUNNING“ ist und dass C noch nicht aktiviert wurde („NOT_ACTIVATED“).

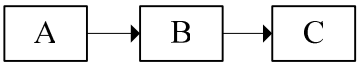


Abbildung 6.5 Beispielsequenz

Ereignisse	Aktivität A	Aktivität B	Aktivtät C
case started	ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED
A started	RUNNING	NOT_ACTIVATED	NOT_ACTIVATED
A completed	COMPLETED	ACTIVATED	NOT_ACTIVATED
B started	COMPLETED	RUNNING	NOT_ACTIVATED
B completed	COMPLETED	COMPLETED	ACTIVATED
C started	COMPLETED	COMPLETED	RUNNING
C completed	COMPLETED	COMPLETED	COMPLETED
case completed	COMPLETED	COMPLETED	COMPLETED

Tabelle 6.6 Zustände für Abbildung 6.5

Weitere Beispiele, mit anderen Grundkonstrukten, für die Herleitung von Zuständen finden sich im Anhang D.

Der Zustand der gesamten Prozessinstanz, kann abhängig von den Zuständen der einzelnen Aktivitäten, folgendermaßen ermittelt werden:

- Die Prozessinstanz ist aktiviert („ACTIVATED“), sobald eine Aktivität aktiviert ist.
- Die Prozessinstanz ist laufend („RUNNING“), sobald eine Aktivität gestartet wurde.
- Ob eine Prozessinstanz beendet („COMPLETED“) ist, hängt von der Implementierung des jeweiligen Systems ab. Dafür gibt es drei verschiedene Möglichkeiten:
  - Bei einer Endaktivität im System, ist die Prozessinstanz beendet, sobald die einzige Endaktivität beendet/erreicht ist.
  - Bei mehreren Endaktivitäten ist sie beendet, sobald eine der Endaktivitäten beendet/erreicht ist.
  - Bei impliziter Terminierung, ist sie beendet, sobald alle Aktivitäten des Prozesses beendet sind.

6.4.2 Zwischenmodelle für das Mapping von Zuständen

Auch beim Mapping von Zuständen gibt es ein minimales, maximales und ein kanonisches Zwischenmodell. Diese werden im Folgenden als Zustandsmodelle bezeichnet und die Vor- und Nachteile der jeweiligen Varianten werden aufgezeigt.

6.4.2.1 Maximales Zustandsmodell

Beim maximalen Zustandsmodell werden, für die Integration, alle Zustände aller Systeme in Betracht gezogen. Das bedeutet, dass die Menge der betrachteten Zustände eine Vereinigung der Zustände aller Systeme ist.

Definition: Sei Z(i) die Zustandsmenge von System i, wobei $0<i\leq n$ und n=Anzahl der Systeme. Dann ist die Menge der betrachteten Zustände $Z(b)= \bigcup Z(i), \forall i$.

Abbildung 6.6 zeigt die Abbildung zweier Zustandmodelle [IBM04, Rei00] auf das maximale Zustandsmodell. Die Abbildung ist einfach, weil es zu jedem Zustand des Quellmodells genau den gleichen Zustand im Zielmodell gibt. Zudem bleibt die Semantik der Zustände erhalten, da die die Mächtigkeit des Quellmodells nicht eingeschränkt wird. Somit hat das Mapping eine hohe Qualität (s. Abschnitt 2.2.3).

Nachteil ist allerdings, dass die Übersicht durch die Zustände aller Systeme verloren geht.

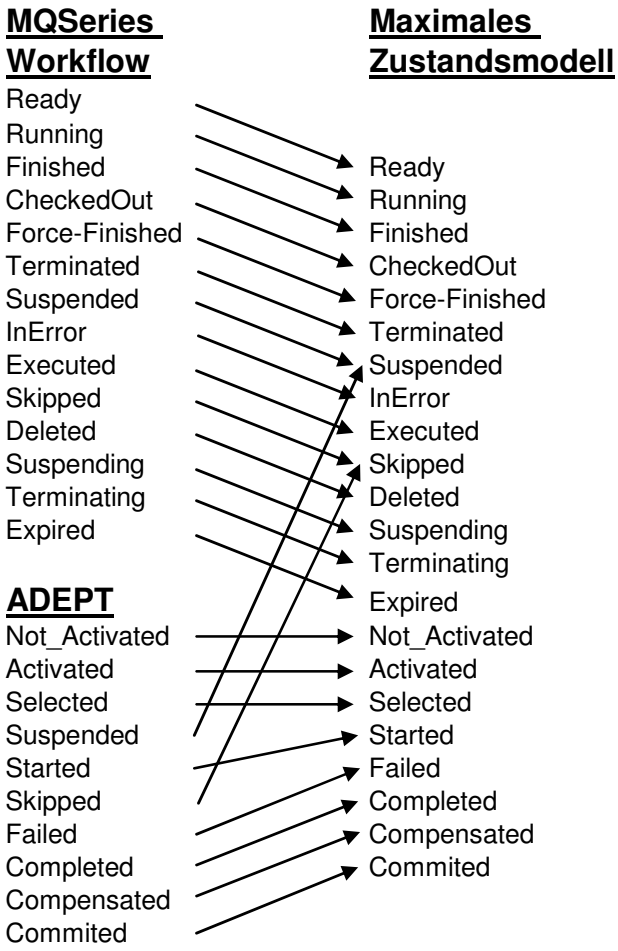


Abbildung 6.6 Abbildung zweier Zustandsmodelle auf das maximale Modell

6.4.2.2 Minimales Zustandsmodell

Das minimale Zustandsmodell besteht nur aus denjenigen Zuständen, die in allen Systemen angeboten werden. Die Menge der Zustände für die Visualisierung ist somit die Schnittmenge der Zustände aller Systeme.

Definition: Sei Z(i) die Zustandsmenge von System i, wobei $0<i\leq n$ und n=Anzahl der Systeme. Dann ist die Menge der betrachteten Zustände $Z(b)= \bigcap Z(i), \ \forall i$.

Abbildung 6.7 zeigt die Abbildung zweier Zustandsmodelle auf das minimale Zustandsmodell. Durch die Unterschiede der beiden Quellmodelle wird die Mächtigkeit der Zustandsmodelle auf zwei Zustände beschränkt, wodurch keine vernünftige Abbildung stattfinden kann. Die Zustände der Quellmodelle können somit gar nicht oder nur mit einer sehr niedrigen Qualität abgebildet werden. Vorteil ist eine sehr übersichtliche Darstellung der Zustände.

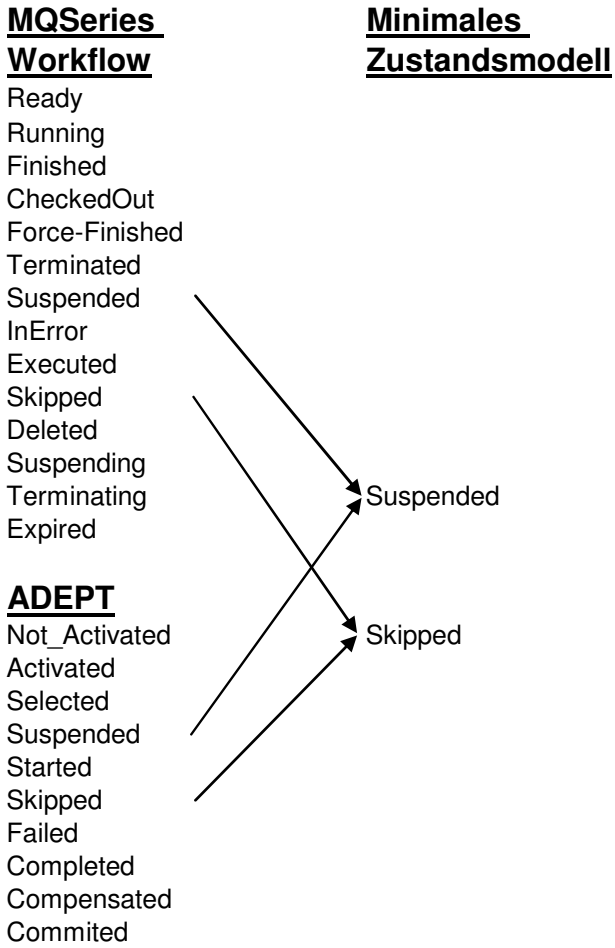


Abbildung 6.7 Abbildung zweier Zustandsmodelle auf das minimale Modell

6.4.2.3 Kanonisches Zustandmodell

Beim kanonischen Zustandsmodell wird die Menge der Zustände vorgegeben. Das Zustandsmodell ist somit unabhängig von den Zustandsmengen, die abgebildet werden sollen.

Definition: Z(g)=fest, unabhängig von Z(i), $\forall$ i.

So eine vorgegebene Zustandsmenge könnten beispielsweise die Zustände sein, die von der WfMC vorgegeben werden, oder ein anderes kanonisches Modell, das entweder auf bestimmte Anwendungsfelder angepasst oder allgemein und gut erweiterbar ist.

Die WfMC schlägt 4 Zustände für Aktivitäten vor [WfMC99a]. Diese werden in Tabelle 6.7 angezeigt.

Zustand	Beschreibung
inactive	Die Aktivitätsinstanz wurde erstellt, aber u.U.noch nicht aktiviert; es existieren noch keine Arbeitspositionen für diese Aktivität.
active	Eine oder mehrere Arbeitspositionen wurden erstellt und zur Ausführung zugewiesen.
suspended	Die Aktivitätsinstanz steht still; es werden keine weiteren Arbeitspositionen gestartet, bevor sie fortgesetzt wird.
completed	Die Aktivitätsinstanz wurde beendet.

Tabelle 6.7 Aktivitätszustände der WfMC [WfMC99a]

Nimmt man das WfMC-Modell als kanonisches Modell und bildet die beiden bisherigen Quellmodelle auf dieses ab, so ergibt sich ein Mapping wie in Abbildung 6.8.

Durch das kanonische Modell kann man nun mehr Zustände als bei dem minimalen Zustandsmodell abbilden, behält aber dennoch die Übersicht, weil nicht so viele Zustandsmodelle gemischt werden wie beim maximalen Zustandsmodell. Die Mächtigkeit des Modells wird nicht eingeschränkt, sondern bleibt so, wie sie definiert wurde.

Allerdings hat das kanonische Zustandsmodell auch Nachteile.

Erstens ist ein Mapping mit optimaler Qualität nur für wenige Zustände möglich. Will man die anderen Zustände aus Abbildung 6.8 ebenfalls abbilden, so wird dabei deren Semantik geändert. Eine andere Möglichkeit ist, ein erweiterbares Zustandsmodell zu wählen, bei dem bei Bedarf weitere Zustände zur bestehenden Menge hinzugefügt werden können.

Zweitens ist die Abbildung der Zustände schwieriger als bei den anderen Modellen, da die Zustände der Systeme normalerweise in anderer Form als im kanonischen Modell vorliegen. Somit müssen diese meistens transformiert werden, wobei wieder Konflikte auftreten können.

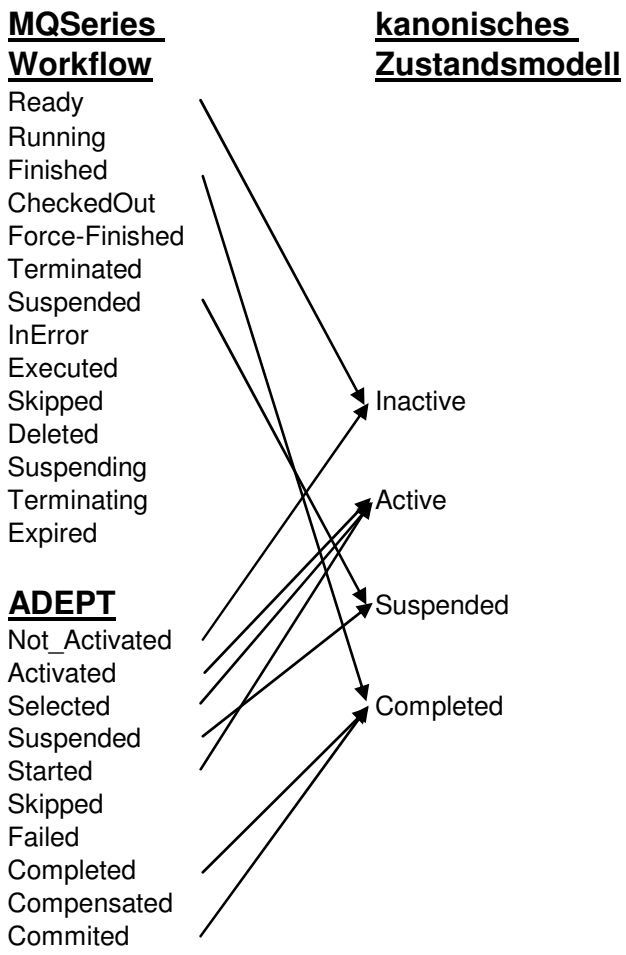


Abbildung 6.8 Abbildung von zwei Zustandsmodellen auf ein kanonisches Modell

Dennoch ist ein kanonisches Modell nach Abwägen der Vor- und Nachteile das Beste der vorgestellten Zustandsmodelle. Um die angesprochenen Konflikte zu beseitigen, werden diese im Folgenden genauer betrachtet und es wird eine mögliche Lösung aufgezeigt.

6.4.2.3.1 Konflikte beim Mapping von Zuständen auf ein kanonisches Modell

Beim Mapping von Zuständen von einem Quell- auf ein Zielmodell treten Konflikte auf, wenn Quell- und Zielzustände sich in irgendeiner Form unterscheiden, wenn also ein Zustand des Quellmodells im Zielmodell nicht existiert oder andersrum, ein Zustand des Zielmodells im Quellmodell nicht existiert. Von den beiden Fällen ist nur der erste interessant, da das Mapping vom Quell- zum Zielmodell hin erfolgt. (Wenn es mehr Zustände gibt, auf die abgebildet werden könnte, als die, die man abbilden muss, so ist das kein wirkliches Problem.) Durch diese Konflikte können Verfälschungen und semantische Abwertungen im integrierten Prozess entstehen.

Die Konflikte zwischen Quell- und Zielmodell haben ihre Hauptursachen in der Bedeutung und in der Granularität der Zustandsmodelle:

- Bedeutung: Werden zwei völlig unterschiedliche Zustände, deren Bedeutung nicht vollkommen angepasst werden kann, aufeinander abgebildet, so ist klar, dass der Zielzustand nicht genau das wiedergibt, was mit dem Quellzustand gemeint war. Das wäre beispielsweise der Fall, wenn man aus Abbildung 6.8 den Zustand „InError“ von MQSeries Workflow auf einen der Zustände des kanonischen Zustandsmodells abbilden würde. In diesem gibt es nämlich keinen Zustand, der eine zumindest annähernd ähnliche Bedeutung wie „InError“ hat.
- Granularität: Bei Zustandsmodellen mit unterschiedlicher Granularität der darstellbaren Zustände kommt es zu ähnlichen Problemen. Die Bedeutung der Zustände, die nicht optimal aufeinander abgebildet werden können, kann zwar ähnlich sein, diese liegen aber in unterschiedlichen Details vor. Betrachtet man wieder die Abbildung 6.8, so kann man sehen, dass beispielsweise der Zustand „active“ des kanonischen Modells in ADEPT viel detaillierter vorliegt. Dort gibt es die Zustände „ACTIVATED“, „SELECTED“ und „STARTED“, die einen ähnlichen Zustand beschreiben, allerdings detaillierter. Man kann sagen, dass die drei ADEPT-Zustände bei ihrer Abbildung auf „active“ verallgemeinert werden. Somit werden sie semantisch abgewertet.

  Wenn das kanonische Modell einen höheren Detaillierungsgrad hat, dann sind auch semantische Aufwertungen möglich. Dazu sind noch zusätzliche Daten aus dem Prozess- und dem Metamodell nötig, mit deren Hilfe man Annahmen treffen und so u.U. die groben Zustände des Quellmodells auf speziellere Zustände des Zielmodells abbilden kann.

6.4.2.3.2 Lösungen für Konflikte beim Mapping von Zuständen auf ein kanonisches Modell

Um die Konflikte zu lösen, die beim Mapping von Zuständen durch unterschiedliche Bedeutungen und Granularitäten auftreten, hat man mehrere Möglichkeiten.

Eine Möglichkeit ist, die Abbildung trotz semantischer Verfälschung oder Abwertung zuzulassen und sie zu vermerken, beispielsweise durch einen Kommentar oder durch eine Markierung. Diese Lösung ist zwar einfach, das Ergebnis ist allerdings verbesserungswürdig.

Eine bessere Lösung bietet sich, wenn ein erweiterbares kanonisches Modell benutzt wird. In so einem Fall kann man bei Bedarf benutzerdefinierte Zustände definieren. Allerdings muss man bei der Definition aufpassen, da bei zu vielen neu definierten Zuständen das kanonische Zustandmodell schnell zu einem maximalen Zustandsmodell mutieren kann.

Um so eine Mutierung aber auch ein falsches Mapping zu vermeiden, sollte eine andere Vorgehensweise benutzt werden. Man muss den vorherigen Zustand und das Ereignis,

das zur Zustandsänderung geführt hat, betrachten und diese auf das kanonische Modell abbilden. Nach der Abbildung kann der alte Zustand mit dem Ereignis - nun beide im kanonischen Modell - in einen neuen Zustand des kanonischen Zustandsmodells übergeführt werden.

Kann der vorherige Zustand auch nicht ohne Probleme auf das kanonische Modell abgebildet werden, dann müssen u.U. alle Zustände von Anfang an neu bewertet werden. Diese Variante ist aber nur dann sinnvoll und möglich, wenn sich die Ereignisse der beiden Modelle weniger unterscheiden, als ihre Zustände.

6.5 Zusammenfassung

In diesem Kapitel wurden verschiedene Probleme bei der Integration von Instanzdaten und Lösungsansätze für diese aufgezeigt. Eines der größten Probleme dabei ist die Zuordnung zusammengehöriger Instanzdaten, ohne globale Schlüssel, zueinander. Dieses Problem wurde mit Hilfe von Korrelationsmengen gelöst.

Ein weiteres Problem sind Änderungen des Prozessmodells, die bei der Prozesstransformation und -integration entstehen, durch die die Instanzdaten nicht mehr zu den Modelldaten passen. Die dabei entstehenden Konflikte durch Namensänderungen können relativ einfach mit Hilfe von Namensumsetzungstabellen aufgelöst werden. Auch die durch semantische Änderungen entstandenen Konflikte können durch Transformationen, analog zu denen der Prozesstransformationen, aufgelöst werden. Strukturelle Änderungen dagegen verursachen mehr Probleme. Diese sind sowohl von Prozesstransformation als auch -integration abhängig. Zudem spielen die beteiligten Metamodelle eine große Rolle, so dass für diese Probleme keine standardisierte Lösung angegeben werden kann.

Außerdem wurden zwischen Instanzdaten ähnliche Heterogenitäten identifiziert, wie bei den Modelldaten. Bei den Applikationsdaten sind diese bedingt durch die unterschiedlichen Zwecke der Applikationsdaten. Bei den Log- und den Workflow relevanten Daten hängen sie von dem jeweiligen System und Metamodell ab, zu denen sie gehören.

Zum Schluss wurden am Beispiel der Zustände einige spezielle Probleme aufgezeigt. So gibt es beispielsweise Zustandsmodelle mit wenigen und andere mit vielen Zuständen. Um eine optimale Abbildung zwischen diesen zu gewährleisten, müssen u.U. mit Hilfe bereits auf das Zielmodell abgebildeter Ereignisse, die Zustände neu bewertet werden.

7 Architektur

In diesem Kapitel wird eine Architektur zur Lösung der in dieser Arbeit beschriebenen Problematik vorgestellt. Dafür wird in Abschnitt 7.1 zunächst die gesamte Architektur für die systemübergreifende Visualisierung von Arbeitsabläufen grob dargestellt. Dieser Teil soll die spätere Einordnung dieser Arbeit und auch der daraus resultierenden Architektur in den Gesamtkontext der systemübergreifenden Visualisierung erleichtern. Nach der Einordnung der Architektur wird sie in Abschnitt 7.2 detailliert vorgestellt. Am Ende des Kapitels werden noch Möglichkeiten zur Aktualisierung der Daten diskutiert (s. Abschnitt 7.3).

7.1 Gesamtarchitektur der Visualisierungskomponente

Abbildung 7.1 zeigt die gesamte Architektur der Visualisierungskomponente.

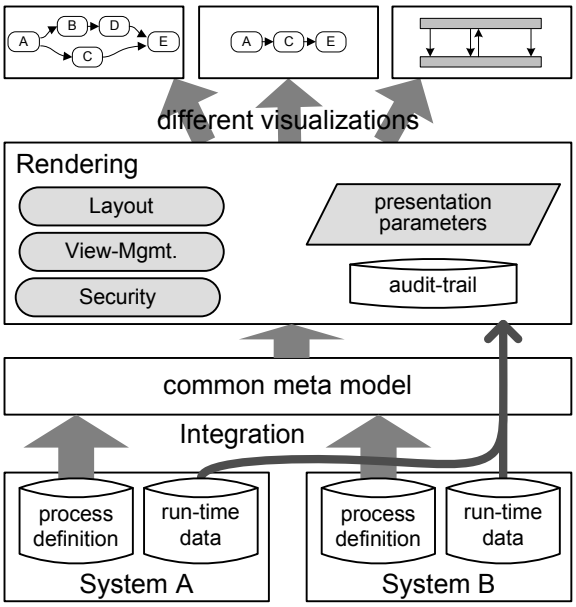


Abbildung 7.1 Architektur der Visualisierungskomponente [BRB05]

Wie bereits erwähnt ist der erste Schritt zu einer systemübergreifenden, einheitlichen Visualisierung die Integration der Modelldaten. Die Daten der verschiedenen Systeme werden auf ein gemeinsames Metamodell (*common meta model*) abgebildet, das als Grundlage für die weitere Visualisierung dient. Ähnlich müssen auch die Laufzeitdaten, d.h. Applikations-, Log- und Workflow relevante Daten, integriert werden.

Nachdem die Daten der Systeme auf das gemeinsame Metamodell abgebildet wurden, dienen sie als Grundlage für die *Rendering*-Komponente, die daraus Prozessmodelle erzeugt. Diese besteht aus mehreren Subkomponenten.

Das *View-Management* ist zuständig für die Restrukturierung der Prozessdaten, damit, abhängig vom Benutzer, die gewünschte bzw. erlaubte Sicht auf den Prozess dargestellt werden kann.

Die Berechnung der geometrischen Anordnung der graphischen Elemente ist Aufgabe der *Layout*-Komponente. Das Layout kann automatisch oder halb-automatisch ermittelt werden. Eine reine manuelle Erzeugung der Prozessmodelle soll vermieden werden. Aufgrund der zahlreichen Visualisierungsmöglichkeiten wäre dies fast unmöglich und mit einem enormen Aufwand verbunden.

Sicherheitsaspekte werden von der *Security*-Komponente behandelt.

Die *Presentation Parameters* dienen der Verwaltung der Konfigurationen der verschiedenen Darstellungen. So soll eine änderungsfähige und Benutzer-spezifische Visualisierung ermöglicht werden [BRB05].

7.2 Architektur der Mapping Komponente

In diesem Abschnitt wird eine Architektur für eine Mapping Komponente vorgestellt. Mapping umfasst dabei Datentransformation und -migration, Prozesstransformation und -integration, sowie die Instanzdatenintegration. Das sind also alle Probleme, die in den Kapiteln drei bis sechs aufgezeigt wurden.

Zur Lösung dieser Probleme besteht die Mapping-Komponente aus zwei Komponenten, aus einer Buildtime-Komponente, die ein integriertes Prozessmodell erstellt und aus einer Runtime-Komponente, die die nötigen Laufzeitdaten zum integrierten Prozessmodell liefert. Bevor aber diese Komponenten in den Abschnitten 7.2.1 und 7.2.2 detailliert beschrieben werden, wird zunächst mit Hilfe von Abbildung 7.2 die Mapping-Komponente in die Gesamtarchitektur der Visualisierungskomponente eingeordnet. Die Abbildung zeigt den unteren Teil der Gesamtarchitektur, die in Abschnitt 7.1 kurz vorgestellt wurde. Der Pfeil markiert die Stelle, an der die Mapping-Operationen stattfinden.

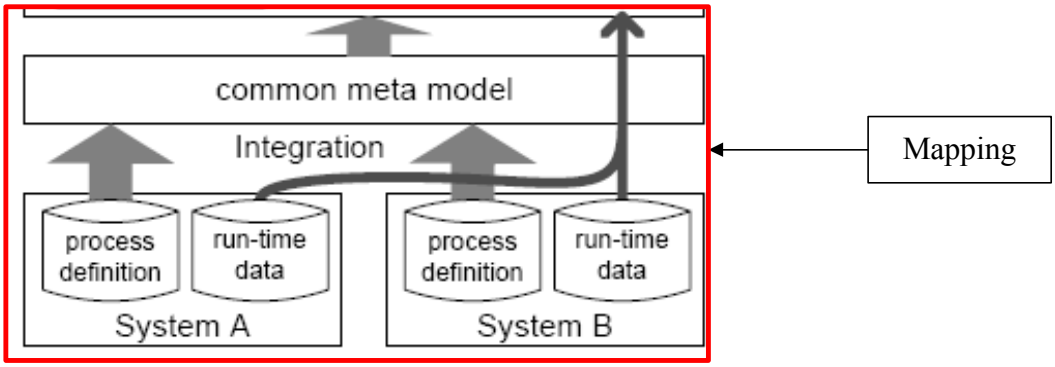


Abbildung 7.2 Teil der Architektur der Visualisierungskomponente [BRB05]

7.2.1 Architektur der Buildtime-Komponente

Bevor die Einzelteile der Architektur der Buildtime-Komponente beschrieben werden, soll die ganze Problematik noch einmal kurz zusammengefasst werden. Dabei werden die Aufgaben aufgezeigt, die für die Erstellung eines Prozessmodells durchgeführt werden müssen (s. Abschnitt 7.2.1.1). Abschnitt 7.2.1.2 betrachtet von diesen Aufgaben einen speziellen Teil genauer, der für die Beschaffung fehlender Daten nötig ist. Nach diesen Grundlagen werden in Abschnitt 7.2.1.3 die Komponenten vorgestellt, die diese Aufgaben durchführen.

7.2.1.1 Ablauf der Prozesstransformation und -integration

In der Abbildung 7.3 sind verschiedene Systemklassen und der Change-Management-Prozess in bereits bekannter Form symbolisiert. Die Bedeutung der anderen verwendeten Symbole ist folgender Beschreibung zu entnehmen.

Informationen, die als Input für Aufgaben benötigt werden oder bei der Bearbeitung von Aufgaben entstanden sind. Dabei ist unerheblich, in welchem Format die Informationen vorliegen. Ob als XML-Datei, Datenbankeintrag oder ob lediglich die Möglichkeit vorhanden ist über eine API (*Application Programming Interface*) auf diese zuzugreifen. Wichtig ist nur, dass die Komponente, die die jeweiligen Informationen benutzt, auf sie unabhängig vom Quellsystem der Informationen, einheitlich zugreifen kann.

Stellt eine Regelbasis dar. Diese werden für Prozesstransformation und -integration benötigt.

Aufgabe Bildet eine Aufgabe ab, die im Zuge der Erstellung eines Prozessmodells durchgeführt werden muss.

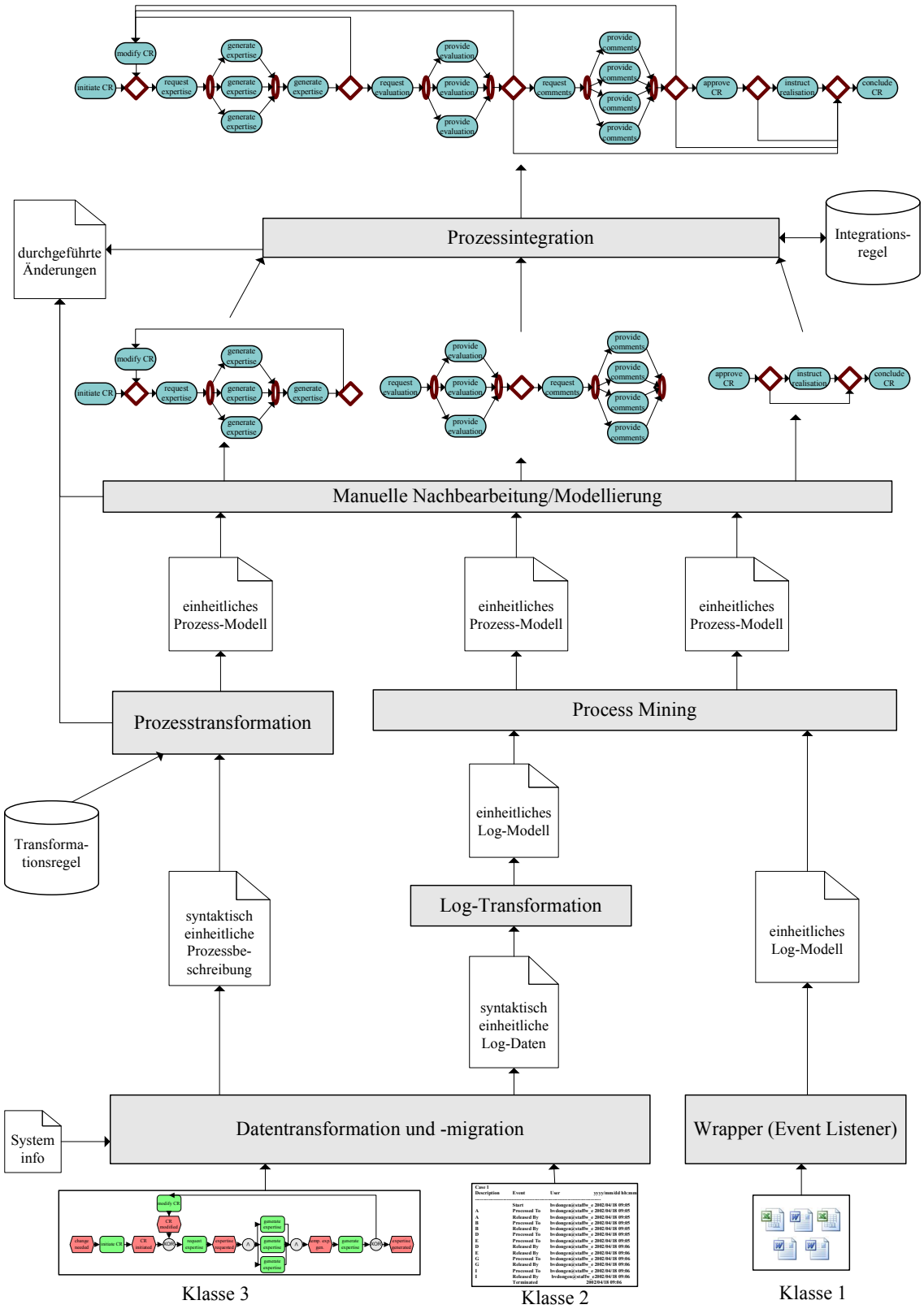


Abbildung 7.3 Ablauf der Prozessmodellerstellung

Ausgangspunkt für die Prozesstransformation und -integration sind die Daten, die von Systemen unterschiedlicher Klassen und Hersteller bereitgestellt werden. Das ist in der Abbildung durch jeweils ein System der Klasse 3, Klasse 2 und Klasse 1 angedeutet. Das Klasse-3-System stellt für die *Buildtime*-Komponente ein Prozessmodell bereit, wodurch die Integration des Prozesses dieses Systems erleichtert wird. Bei den beiden anderen Systemen dagegen fehlen diese Daten und müssen somit zunächst beschafft werden. Die Beschaffung dieser Daten bedeutet eine Aufwertung der Systeme. Eine genaue Beschreibung hierzu findet sich in Abschnitt 7.2.1.2.

Alle Komponenten, außer dem Wrapper (s. Abschnitt 7.2.1.2), benötigen ein bestimmtes Datenformat als Input. Da diese Voraussetzung durch die unterschiedlichen Quellsysteme nicht gegeben ist, ist die erste zu erledigende Aufgabe, diese Daten in ein einheitliches Format und in eine einheitliche Syntax zu bringen. Das wird durch die Datentransformation und -migration erreicht.

Die dabei entstandenen Daten dienen als Grundlage für weitere Transformationen, wie Prozesstransformation oder Log-Daten-Transformation. Die Log-Daten müssen transformiert werden, da für das Process Mining ebenfalls eine einheitliche Schnittstelle vorliegen muss. In der gleichen Form müssen für das Process Mining die vom Wrapper generierten Log-Daten vorliegen. Denn unabhängig davon, ob sie vom Quellsystem oder vom Wrapper aufgezeichnet werden, dienen die Log-Daten an dieser Stelle der automatischen Erstellung von Prozessmodellen mit Hilfe von Process Mining.

Die durch Process Mining und Prozesstransformation entstandenen Prozessmodelle liegen in einheitlicher Form vor, d.h. sie sind im selben Datenformat und basieren alle auf dem gleichen Metamodell. Dadurch wird Modellierungskomponenten eine einheitliche Schnittstelle geboten. Modellierungskomponenten sind an dieser Stelle nötig um die transformierten oder generierten Prozessmodelle vor der Integration überprüfen zu können. In Fällen, in denen automatisch kein Prozessmodell erstellt werden konnte, kann an dieser Stelle ein Prozessmodell vollständig manuell erstellt werden.

Der letzte Schritt, der auf einheitlichen Prozessfragmenten aufsetzt, ist die Prozessintegration. Hier werden die Prozessfragmente in ein gemeinsames Modell zusammengefügt und dabei u.U. letzte Änderungen vorgenommen.

Das Ergebnis ist ein einheitliches Gesamtmodell, das der Visualisierungskomponente als Grundlage dient.

7.2.1.2 Aufwertung der Systeme

Wie bereits früher in dieser Arbeit gezeigt, werden durch die jeweiligen Systemklassen unterschiedliche Daten bereitgestellt (s. Abschnitt 2.1.2.2). Nur Klasse-3-Systeme stellen die Modelldaten bereit, die für eine Prozessintegration nötig sind. Bei den Systemen der Klasse 1 und 2 müssen diese Daten zunächst beschafft werden. Das ist gleichbedeu-

tend mit der Aufwertung der Systeme auf Klasse 3. Bei Klasse-2-Systemen kann dies direkt mit Hilfe von Process Mining erfolgen. Klasse 1-Systeme dagegen müssen zunächst auf Klasse 2 aufgewertet werden. Wir werden also zunächst diesen Schritt vornehmen.

- Aufwertung von Klasse 1 auf Klasse 2:

  Wie bereits bekannt, stellen Klasse-1-Systeme nur Applikationsdaten und, auf diesen basierenden, Workflow relevante Daten bereit. Diese Applikationsdaten haben zunächst keine Verbindung zu einem Prozess oder zu einer Aktivität in einem Prozess. Mit Hilfe von Log-Daten kann dieser Bezug hergestellt werden.

  Für die Beschaffung von Log-Daten kann man, wie bereits in Abschnitt 2.2.1.3 beschrieben, einen Wrapper benutzen. Der Wrapper ist eine Art *Event-Listener*, der das jeweilige System kapselt, was heißt, dass er alle Ereignisse dieses Systems beobachtet. Ereignisse, wie beispielsweise Zugriffe auf die Applikationsdaten, können so protokolliert werden.

  Durch Hinzunahme der Log-Daten sind die bisherigen Klasse-1-Systeme nun auf Klasse 2 aufgewertet.

  Um weitere Transformationen zu vermeiden, wird vorausgesetzt, dass die generierten Log-Daten für die Aufwertung auf Klasse 3, sowohl syntaktisch, als auch semantisch bereits im richtigen Format vorliegen. Das stellt aber kein Problem, da ein Wrapper für jedes Applikationssystem speziell programmiert werden kann. Somit kann der Wrapper von Anfang an auf die jeweiligen Vorgaben eingestellt werden.

- Aufwertung von Klasse 2 auf Klasse 3:

  Bei Klasse-2-Systemen hat man zunächst auch nicht genügend Daten, um den Prozess problemlos visualisieren zu können. Hier muss auch Vorarbeit geleistet werden. Dabei sollen Modelldaten dazukommen.

  Die Applikationsdaten sind dabei wenig hilfreich, da sie ohne weitere Informationen keinen Bezug zu einem Prozess haben. Auch die Workflow relevanten Daten sind für diesen Zweck nicht zu gebrauchen. Also muss man die nötigen Informationen für ein Prozessmodell aus den Log-Daten gewinnen. Mit Hilfe von Process Mining und den richtigen Log-Daten lässt sich ein Klasse-2-System auf Klasse 3 aufwerten. Process Mining und die dafür notwendigen Log-Daten wurden in Abschnitt 2.2.1.3 beschrieben.

  Um auch hier unnötige Transformationen am generierten Modell zu vermeiden, soll das Modell, das beim Process Mining generiert wird, syntaktisch und semantisch im richtigen Format für die nächste Komponente, die diese Daten braucht, vorliegen.

7.2.1.3 Aufbau der Buildtime-Komponente

Die Buildtime-Komponente ist aus Teilen aufgebaut, die die Aufgaben aus Abbildung 7.3 erledigen. Diese Komponenten sind in Abbildung 7.4 zu sehen.

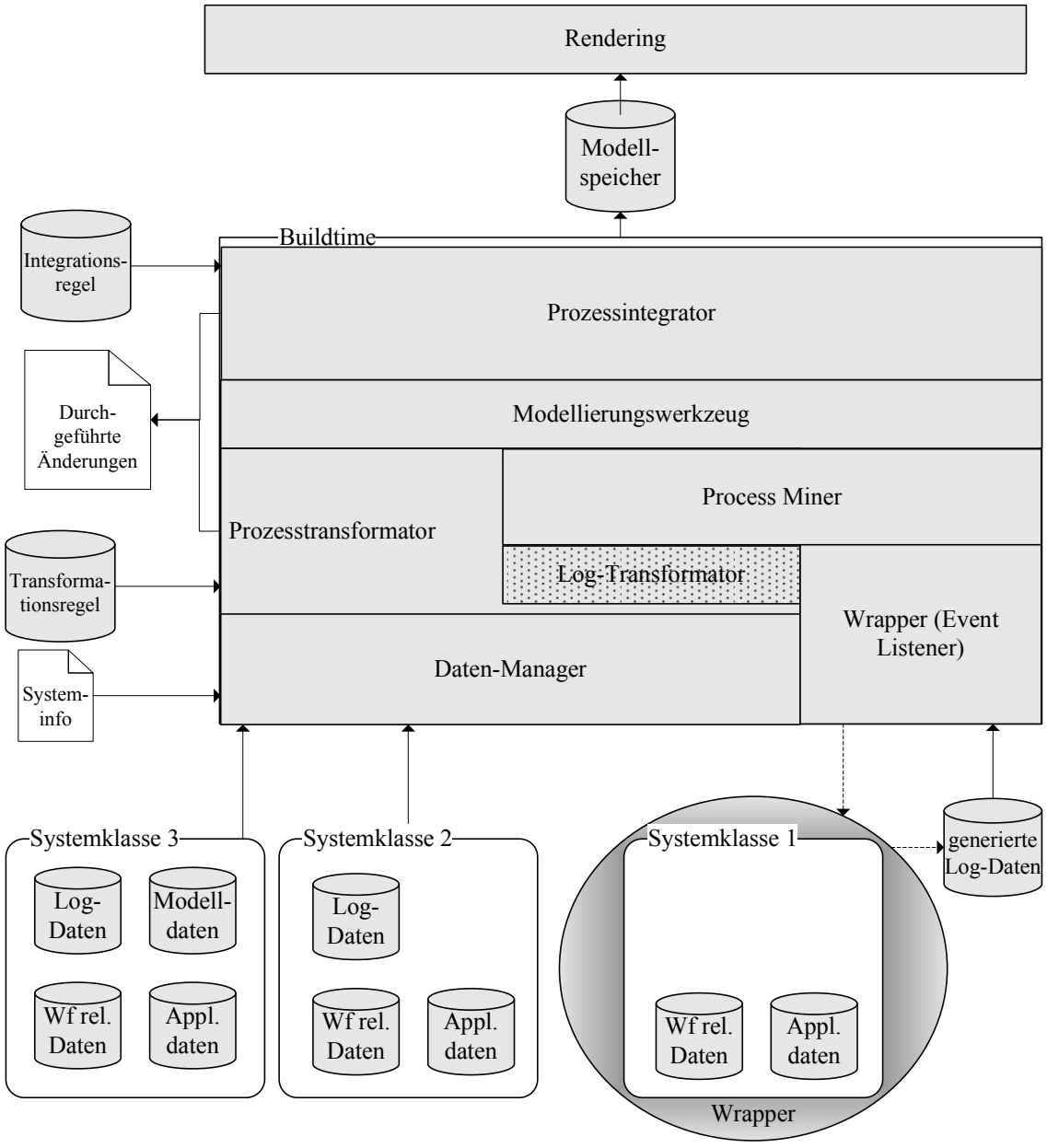


Abbildung 7.4 Aufbau der Buildtime-Komponente

Durch die unterschiedlichen Systemklassen und dadurch unterschiedlichen Ausgangssituationen zwischen diesen, ist keine klare Schichtenarchitekur möglich. Dennoch ist es sinnvoll, den Aufbau so weit wie möglich in Schichten oder zumindest Teilschichten zu gliedern, da so deutlicher wird, wo die gemeinsamen Schnittstellen zwischen den Komponenten sind. Durch die Bedingungen, dass Prozeduraufrufe nur von oben nach unten, der Datenfluss jedoch nur von unten nach oben erlaubt sind und die Vorgabe dass keine

Schichten übersprungen werden dürfen, wird die Abhängigkeit der Komponenten voneinander deutlich gemacht. Eine Beschreibung der Komponenten und ihrer Abhängigkeit voneinander findet im Folgenden statt.

- Daten-Manager: Der Daten-Manager befindet sich auf der untersten Ebene der Architektur. Dieser greift auf die Datenquellen von Klasse-2- und -3-Systemen zu. Dabei lädt er die Modell- und Log-Daten der Systeme und beseitigt syntaktische Heterogenitäten. Dafür braucht der Daten-Manager Informationen, die Metadaten über die Quellsysteme bzw. über ihre Daten erhalten. Die Umsetzung kann mit den aus Kapitel 3 bekannten Technologien erfolgen. Als Output liefert der Daten-Manager syntaktisch einheitliche Daten, die vom Prozesstransformator weiterverarbeitet werden.
- Prozesstransformator: Transformiert Modelldaten und Log-Daten in ein vorgegebenes Zwischenmodell und beseitigt dabei die Heterogenitäten zwischen den Modellen, die aus Kapitel 4 bekannt sind. Log-Daten sind zwar Laufzeitdaten, können aber, wie bereits erklärt, auch für Process Mining benutzt werden und werden somit auch von der Buildtime-Komponente benötigt. Die Transformation der Log-Daten ist analog zu der der Modelldaten und somit ist der Log-Transformator ein Teil des Prozesstransformators. Die transformierten Log-Daten sind allerdings über eine andere Schnittstelle zu erreichen, da diese dem *Process Miner* (s.u.) als Input dienen, anders als die transformierten Modelldaten, die vom Modellierungswerkzeug benötigt werden (s.u.).

  Die Transformationen beider Arten von Daten werden nach Transformationsregeln durchgeführt, die in einer eigenen Datenquelle abgelegt sind (s. Datenspeicher - Transformationsregel). Nach Durchführung jeder Transformation wird die durchgeführte Änderung am Quellmodell in einer Tabelle gespeichert, die später von der Runtime-Komponente benutzt wird (s. Tabelle - durchgeführte Änderungen).
- Wrapper: Wie in Abbildung 7.4 zu sehen ist, ist der Daten-Manager nicht die einzige Komponente, die auf die Quellsysteme zugreift. Der Wrapper ist für Klasse-1-Systeme zuständig. Wie bereits beschrieben, kapselt er die Systeme, indem er alle Ereignisse beobachtet, die mit dem jeweiligen System in Verbindung stehen. Ereignisse, die für die Erzeugung von Prozessmodellen benötigt werden, werden von ihm syntaktisch und semantisch in einer Form aufgezeichnet, die vom *Process Miner* benötigt wird.
- Process Miner: Diese Komponente führt das Process Mining durch (s. Abschnitt 2.2.1.3). Die dafür benötigten Daten erhält er entweder vom Prozesstransformator (genauer Log-Transformator) oder vom Wrapper. Unabhängig davon, ob diese Daten von Klasse-2-Systemen kommen und transformiert wurden oder ob sie

basierend auf den Applikationsdaten von Klasse-1-Systemen generiert wurden, sind sie syntaktisch und semantisch immer in der gleichen Form. Dadurch reicht die Implementierung eines einzigen Process Miners.

Wichtig beim Process Miner ist, dass er Modelle auf Basis des Metamodells erzeugt, das vom Modellierungswerkzeug als Input benutzt werden kann. Ansonsten müssten die erzeugten Prozessmodelle zunächst unnötigerweise durch den Prozesstransformator transformiert werden.

- Modellierungswerkzeug: Mit Hilfe eines Modellierungswerkzeugs kann man die durch den Prozesstransformator transformierten oder durch den Process Miner generierten Prozessmodelle überprüfen und gegebenenfalls nachbearbeiten. Wie der Process Miner bekommt auch das Modellierungswerkzeug von den beiden darunter liegenden Komponenten die benötigten Daten syntaktisch und semantisch in der gleichen Form. Diese bieten also die gleiche Schnittstelle an.

  Können von den beiden Komponenten gar keine Modelldaten geliefert werden, dann kann an dieser Stelle der Prozess manuell modelliert werden.

  Der Output des Modellierungswerkzeuges dient als Input für den Prozessintegrator.

- Prozessintegrator: Nachdem die Modelldaten alle auf Basis eines vorher festgelegten Metamodells vorliegen, werden die Prozessfragmente, die von den einzelnen Systemen stammen, vom Prozessintegrator zu einem Gesamtprozess zusammengefügt. Neben der Identifikation und Herstellung von Beziehungen zwischen den Prozessfragmenten, werden an dieser Stelle alle aus Kapitel 5 bekannten Aufgaben durchgeführt. Analog zum Prozesstransformator hat auch diese Komponente eine Regelbasis (s. Datenspeicher - Integrationsregel), anhand derer die Integrationen vorgenommen werden. Die für die Integration durchgeführten Änderungen werden zusammen mit den Änderungen des Prozesstransformators in einer Tabelle gespeichert (s. Tabelle - durchgeführte Änderungen).

  Das Ergebnis des Prozessintegrators ist das globale Prozessmodell, das der Rendering-Komponente als Grundlage für die Visualisierung dient (s. Abschnitt 7.1). Das Modell wird allerdings zuvor im Modellspeicher zwischengespeichert, damit es nicht jedes Mal erzeugt werden muss, wenn es benötigt wird.

- Datenspeicher: Für Daten, die in größeren Mengen vorkommen, werden eigene Datenspeicher vorgesehen. Diese gibt es für drei verschiedene Zwecke:

  - Transformationsregel: Die Transformationsregeln werden vom Prozesstransformator für das Mapping von einem Quell- auf ein Zielmodell benötigt. Eine Transformationsregel beschreibt dabei für jedes Konstrukt des Quellmodells auf welches Konstrukt des Zielmodells dieses abgebildet werden kann und

welche Vorbedingungen dafür gegeben sein müssen. Für jedes Paar von Quell- und Zielmodellen gibt es eigene Regeln, da diese von den Metamodellen der beiden Modelle abhängen.

Durch Erweiterungen der Regelbasis können neue Transformationen eingeführt werden.

- Integrationsregel: Die Integrationsregeln werden vom Prozessintegrator bei der Integration von Prozessfragmenten benutzt. Sie beschreiben welche Vorbedingungen für Aktionen bei der Integration erfüllt sein müssen und welche Aktionen möglich sind.

  Durch Erweiterung der Regelbasis können neue Aktionen auf den Prozessfragmenten durchgeführt werden.

- Modellspeicher: Diese Datenbank speichert die integrierten Modelldaten zusammen mit Informationen zu den Quellsystemen, von denen sie stammen. (Die Speicherung der Quellsysteme wird von der Laufzeitkomponente (s.u.) benötigt.) Eine Zwischenspeicherung der Modelldaten an dieser Stelle soll verhindern, dass nicht bei jeder Visualisierung eines Prozesses die ganze Prozedur der Modellerzeugung durchlaufen werden muss. Stattdessen kann dieser einfach aus dem Modellspeicher geladen werden.

- Generierte Log-Daten: Die generierten Log-Daten werden vom Wrapper aufgezeichnet. Sie werden vom Process Miner der Buildtime-Komponente und vom Instanzdaten-Integrator der Runtime-Komponente (s.u.) benötigt.

- Weitere externe Daten:

  - Dokumente mit Systeminformationen werden vom Data-Manager benötigt, da die zu verarbeitenden Informationen systemspezifisch sind. Mit Hilfe der Systeminformationen, u.a. dem Metamodell, auf dem die Daten des Systems basieren, können die Daten der einzelnen Systeme in ein syntaktisch einheitliches Format gebracht werden.

  - Eine Tabelle mit den durchgeführten Änderungen des Prozesstransformators und -integrators wird benötigt, damit die Runtime-Komponente die entsprechenden Anpassungen für die Integration der Instanzdaten vornehmen kann. Die protokollierten Änderungen sollen die Lösung der Probleme aus Abschnitt 6.2.2 ermöglichen.

7.2.2 Architektur der Runtime-Komponente

7.2.2.1 Beschaffung der Laufzeitdaten

Nachdem das Prozessmodell durch die Buildtime-Komponente bereitgestellt worden ist, müssen noch die Laufzeitdaten, also Applikations-, Log- und Workflow relevante Daten dafür beschafft werden. Der Ablauf dieser Datenbeschaffung ist in Abbildung 7.5 zu sehen.

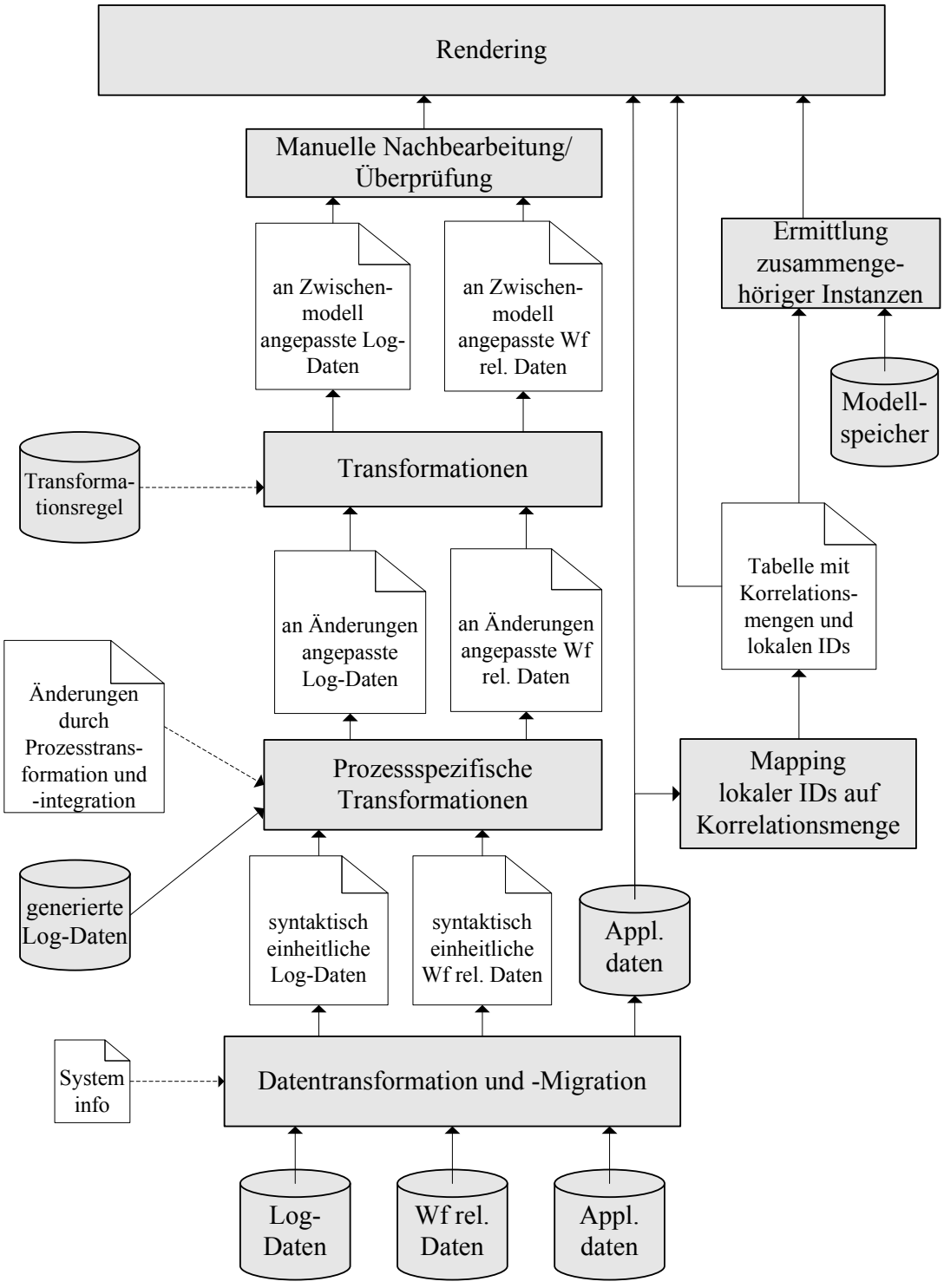


Abbildung 7.5 Beschaffung der Laufzeitdaten

Wie bereits bei der Buildtime-Komponente, müssen auch hier die Daten der verschiedenen Quellsysteme erst einmal in eine syntaktisch einheitliche Form gebracht werden. Somit ist auch hier der erste Schritt die Datentransformation und -migration. Danach können die Applikationsdaten direkt von der Rendering-Komponente gelesen und soweit benötigt angezeigt werden. Ihre semantischen Heterogenitäten müssen nicht wie bei den anderen Laufzeitdaten beseitigt werden, da Applikationsdaten ganz unterschiedliche Zwecke dienen und somit von unterschiedlicher Bedeutung sein können. Somit sind diese Heterogenitäten normal (s. auch Abschnitt 6.3.1). Neben der Visualisierung werden Applikationsdaten auch benutzt, um aus Ihnen Korrelationsmengen zu bilden. Die Korrelationsmengen dienen zur eindeutigen Identifikation von Instanzen und können somit benutzt werden, um Instanzen unterschiedlicher Systeme einander zuordnen zu können. Dazu dient eine Tabelle, die die erzeugten Korrelationsmengen zusammen mit den lokalen IDs der Systeme speichert.

Für die Beschaffung der Log- und Workflow relevanten Daten müssen analoge Aufgaben durchgeführt werden. Nachdem sie in einer syntaktisch einheitlichen Form bereitliegen, können sie auf die prozessspezifischen Änderungen angepasst werden, die durch Prozesstransformator und -integrator entstanden sind. Dazu wird die bereits beschriebene Tabelle mit diesen Änderungen benutzt. Bei Klasse-1-Systemen müssen die vom Wrapper generierten Log-Daten aus dem Datenspeicher gelesen werden. Für diese ist keine Datentransformation notwendig, da sie bereits im richtigen Format aufgezeichnet wurden.

Nach diesen Änderungen liegen die Daten syntaktisch einheitlich und strukturell auf den durch die Buildtime-Komponente geänderten Prozess angepasst vor. Somit kann nun eine analoge Transformation wie beim Prozessmodell vorgenommen werden, bei der die Daten auf ein Zwischenmodell abgebildet werden. Für die Transformation werden auch hier Transformationsregeln benutzt. Eine Integration muss an dieser Stelle nicht durchgeführt werden, da die Daten bereits an die Änderungen des Prozessintegrators angepasst worden und damit auch schon integriert sind.

Nachdem die Daten transformiert sind, werden sie noch manuell überprüft und eventuell nachbearbeitet, bevor sie von der Rendering-Komponente für die Visualisierung benutzt werden.

Die Komponenten, die die beschriebenen Aufgaben erledigen, werden im Folgenden Abschnitt vorgestellt.

7.2.2.2 Aufbau der Runtime-Komponente

Die Teile der Runtime-Komponente, die Aufgaben erledigen und in Abschnitt 7.2.2.1 beschrieben wurden, sind in Abbildung 7.6 abgebildet. Die Grafik ist analog zur Buildtime-Komponente aus Abschnitt 7.2.1.3 aufgebaut. Auch hier ist keine richtige Schich-

tenarchitektur möglich, da auf Applikationsdaten zugegriffen werden muss, ohne dass diese transformiert werden. Für die Applikationsdaten sind nur Datentransformation und -migration nötig.

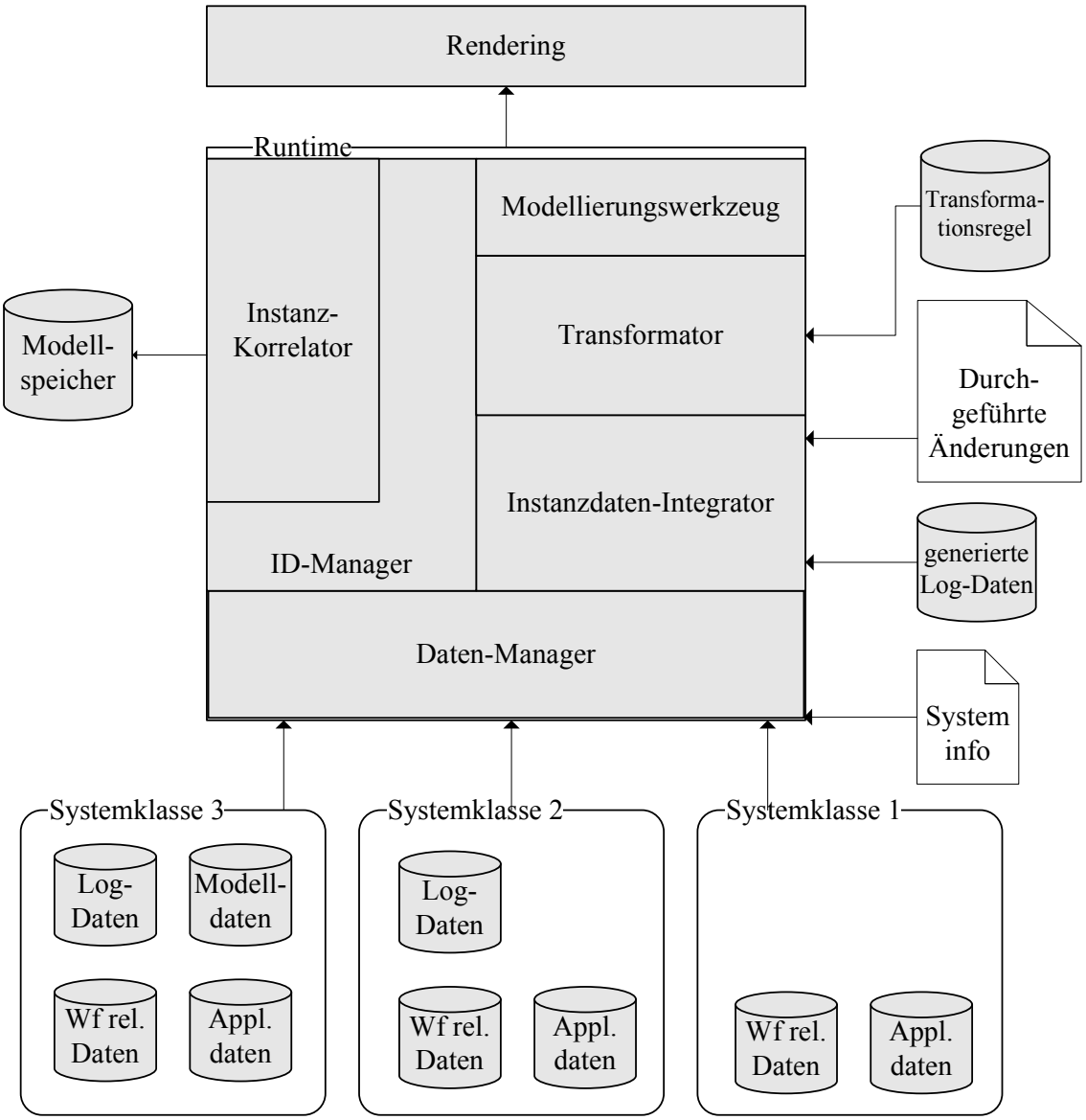


Abbildung 7.6 Aufbau der Runtime-Komponente

- Daten-Manager: Diese Komponente ist identisch zum Daten-Manager der Build-time-Komponente, bis auf den Unterschied, dass diese auf Applikations-, Log- und Workflow relevante Daten zugreift. Die syntaktisch einheitlichen Daten, die der Daten-Manager liefert, dienen als Input für den Instanzdaten-Integrator und für den ID-Manager.
- ID-Manager: Diese Komponente ordnet den lokalen Instanz-IDs von Systemen global eindeutiger IDs zu. Die globalen IDs sind Korrelationsmengen, die aus den Applikationsdaten aller lokalen Instanzen zusammengestellt werden. Der

ID-Manager speichert dabei zu den lokalen IDs auch ihre Quellsysteme, damit bei Anfragen, beispielsweise der Rendering-Komponente, mit einer Korrelationsmenge möglichst schnell die richtigen Systeme aufgefunden werden können. Die Ergebnisse des ID-Managers werden entweder direkt von der Rendering-Komponente oder vom Instanz-Korrelator benötigt.

- Instanz-Korrelator: Der Instanz-Korrelator ermittelt alle Instanzen die zu einem gegebenen Prozessmodell gehören. Dazu benutzt er die vom ID-Manager erzeugten Daten und Informationen aus dem Modellspeicher. Diese Informationen beinhalten alle Quellsysteme zu dem angegebenen globalen Prozessmodell.
- Instanzdaten-Integrator: Diese Komponente passt die Instanzdaten strukturell auf die durch Prozesstransformator und -integrator geänderten Prozesse an. Die hierfür nötigen Informationen werden aus dem Dokument mit den durchgeführten Änderungen bezogen. Die angepassten Instanzdaten sind der Input für den Transformator.
- Transformator: Analog zum Transformator der Buildtime-Komponente werden hier die Laufzeitdaten transformiert und auf ein vorgegebenes Zwischenmodell abgebildet. Die Anpassungen des Instanzdaten-Integrators sind zuvor notwendig, damit keine unnötigen Transformationen an den Instanzdaten eines Prozesses vorgenommen werden, der in dieser Form nicht mehr existiert.
- Modellierungswerkzeug: Diese Komponente dient der manuellen Überprüfung und gegebenenfalls Nachbearbeitung der Instanzdaten (analog Buildtime).

7.2.3 Gesamtarchitektur der Mapping-Komponente

Nachdem die Build- und die Runtime-Komponenten einzeln vorgestellt wurden, wird in Abbildung 7.7 ein Gesamtbild der Architektur gezeigt. Auf eine Erklärung der Komponenten wird an dieser Stelle verzichtet, da diese bereits in den vorhergehenden Abschnitten beschrieben wurden.

Es soll lediglich das Gesamtbild vermittelt werden, in dem die bereits aufgezeigten Abhängigkeiten und der Datenfluss zwischen den Komponenten deutlich werden.

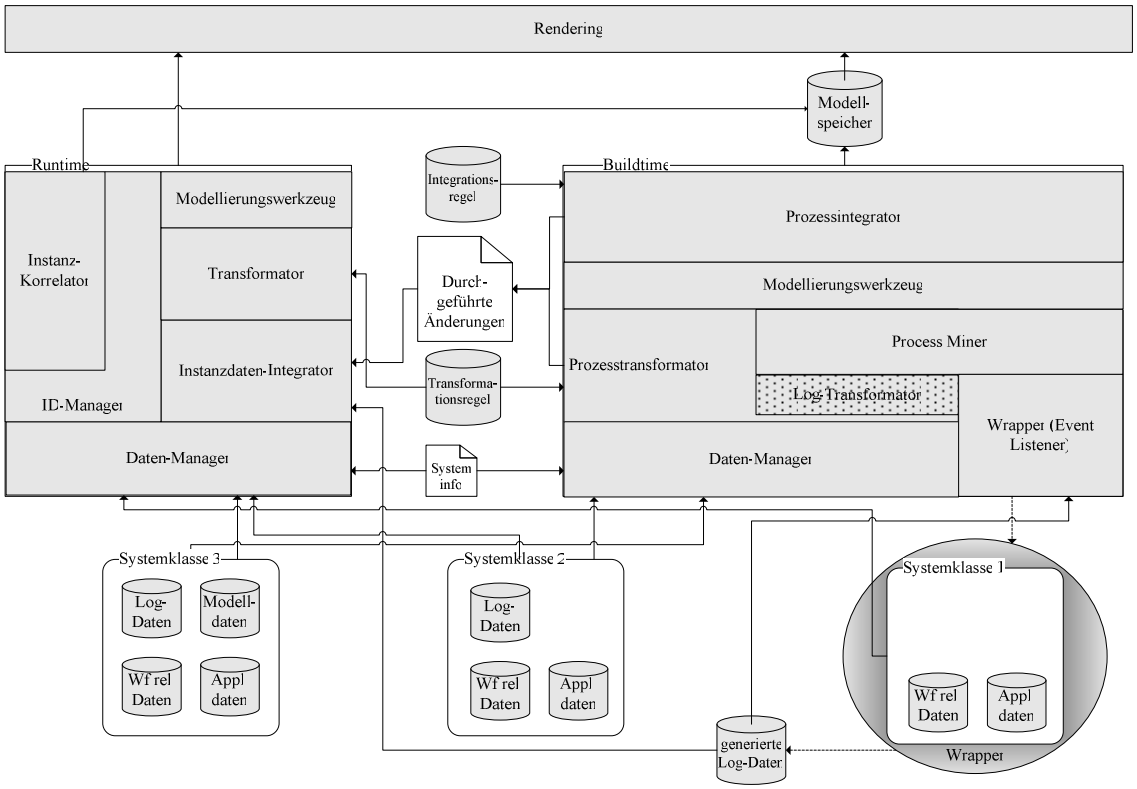


Abbildung 7.7 Gesamtarchitektur der Mapping-Komponente

7.2.4 Sequenzdiagramme

Nachdem die Architektur der beiden Komponenten und auch grob der Datenfluss zwischen ihren Subkomponenten vorgestellt wurden, wird in diesem Abschnitt für beide Komponenten jeweils ein typisches Szenario anhand eines Sequenzdiagrammes erklärt. Dabei werden auch Schnittstellen der Sub-Komponenten betrachtet.

Sequenzdiagramme für weitere Szenarien finden sich im Anhang E.

7.2.4.1 Buildtime

Das Beispiel-Szenario für die Buildtime-Komponente ist die Erzeugung eines Prozessmodells.

Die Erzeugung des Prozessmodells beginnt mit dem Versuch der Rendering-Komponente, das gewünschte Prozessmodell (im Diagramm engl. *Process Model* (PM)) aus dem Modellspeicher zu laden.

Wenn dieses nicht vorhanden ist, dann wird die *getIntegratedProcessModel*-Methode des Prozessintegrators aufgerufen. Als Parameter wird eine Menge von Quellsystemen angegeben, die an dem gewünschten Prozess beteiligt sind:

```
if (!PG.exists)
Prozessintegrator.getIntegratedProcessModel(sourceSystems)
```

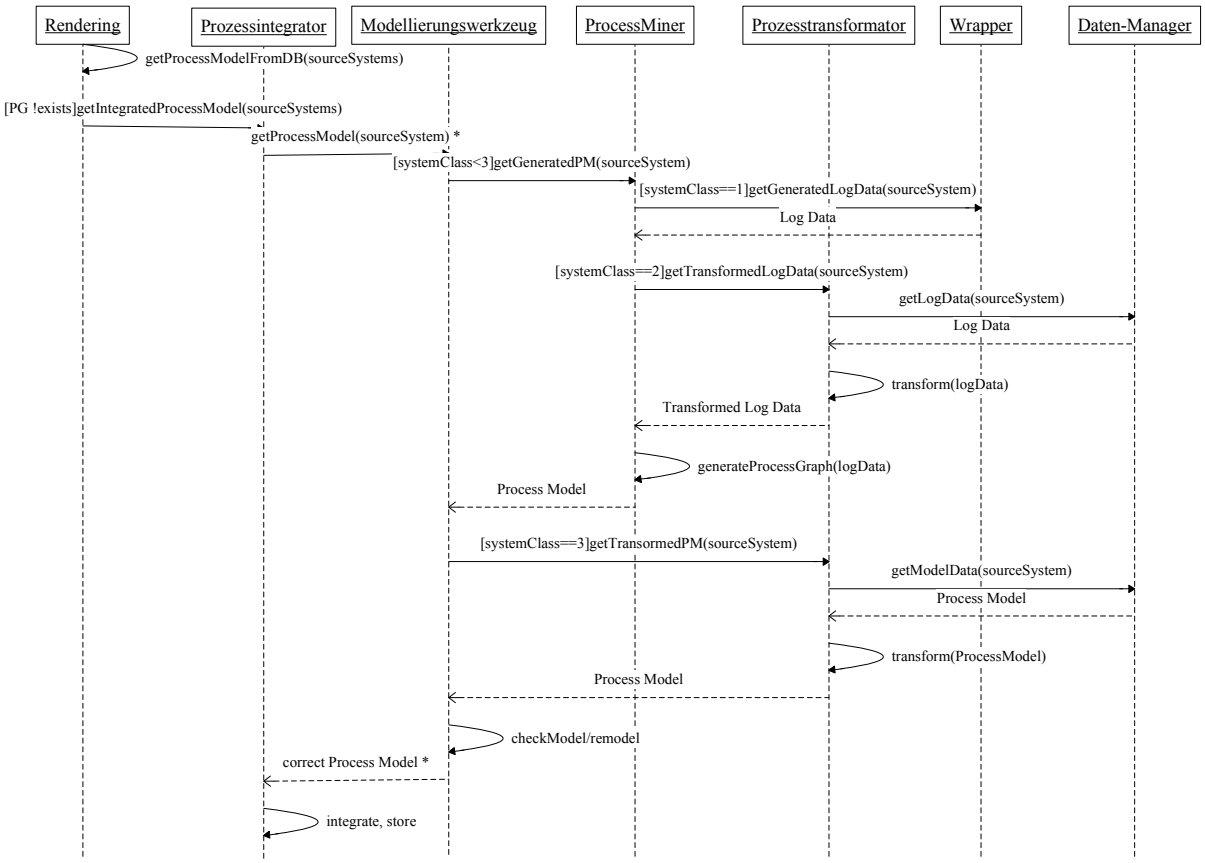


Abbildung 7.8 Sequenzdiagramm - Erzeugung Prozessmodell

Der Prozessintegrator seinerseits ruft wiederholt für alle Systeme die *getProcessModel*-Methode des Modellierungswerkzeugs auf. Bei jedem Aufruf wird ein anderes Quellsystem als Parameter übergeben:

```
for all sourceSystems repeat
{
   Modellierungswerkzeug.getProcessModel(sourceSystem);
}
```

Das Modellierungswerkzeug ruft zunächst abhängig von der Systemklasse entweder die *getGeneratedPM*-Methode des ProcessMiners auf oder die *getTransformedPM-Methode* des Transformators:

```
if (sourceSystem.systemClass==3)
Transformator.getTransformedPM(sourceSystem);
else ProcessMiner.getGeneratedPM(sourceSystem);
```

Für beide Aufrufe ist der Parameter das Quellsystem von dem der Prozess benötigt wird. Und von beiden wird, wenn möglich, ein Prozessmodell im gleichen Format zu-

rückgegeben. Falls kein Prozessmodell zurückgegeben wird, muss anstelle der Überprüfung das Modell komplett manuell modelliert werden.

Der ProcessMiner prüft ebenfalls die Klasse des Systems und abhängig davon holt er sich die benötigten Log-Daten vom Wrapper mit der *getGeneratedLogData*-Methode oder vom Prozesstransformator mit *getTransformedLogData*. Parameter dafür ist jeweils das Quellsystem.

```
if (sourceSystem.systemClass==2)
Transformator.getTransformedLogData(sourceSystem);

else Wrapper.getGeneratedLogData(sourceSystem);
```

In beiden Fällen werden Log-Daten im selben Format zurückgegeben. Der Wrapper kann diese aus den gesammelten Daten direkt zurückgeben, da diese bereits im richtigen Format vorliegen.

Der Prozesstransformator muss sowohl Log-Daten für den ProcessMiner als auch Modelldaten für das Modellierungswerkzeug vom Daten-Manager holen, da er nicht direkt auf die Daten der einzelnen Systeme zugreifen kann.

```
Daten-Manager.getLogData(sourceSystem);

Daten-Manager.getModelData(sourceSystem);
```

Nachdem die Daten vorliegen, werden diese transformiert, bevor sie dem ProcessMiner bzw. dem Modellierungswerkzeug zurückgegeben werden.

Im Modellierungswerkzeug werden die Modelle erstmal geprüft, unabhängig davon, ob sie vom ProcessMiner oder Transformator kommen. Falls kein Modell zurückgegeben wird, muss es manuell modelliert werden. Danach werden die Modelle an den Integrator geschickt, der diese in ein Gesamtmodell integriert, zwischenspeichert und danach an die Rendering-Komponente weitergibt.

7.2.4.2 Runtime

Als Beispiel-Szenario für die Runtime-Komponente wird die Beschaffung aller Instanzen, genauer deren Identifikatoren, zu einem gegebenen Prozessmodell erklärt.

Ausgangspunkt für dieses Szenario ist die Rendering-Komponente, die alle Instanz-IDs eines Prozessmodells braucht, um diese beispielsweise zur Auswahl anzubieten, damit eine davon ausgewählt und visualisiert werden kann. Dazu muss sie zunächst den Instanz-Korrelator aufrufen, damit dieser zu dem gewünschten Prozessmodell alle beteiligten Quellsysteme findet.

```
Instanz-Korrelator.getAllInstanceIDs(processModel);

getSourceSystems(processModel);
```

Nachdem die Quell-Systeme bekannt sind, kann der ID-Manager aufgerufen werden, der für die Instanzen der Quell-Systeme Korrelationsmengen erzeugen soll.

```
ID-Manager.getCorrelationSets(sourceSystems);
```

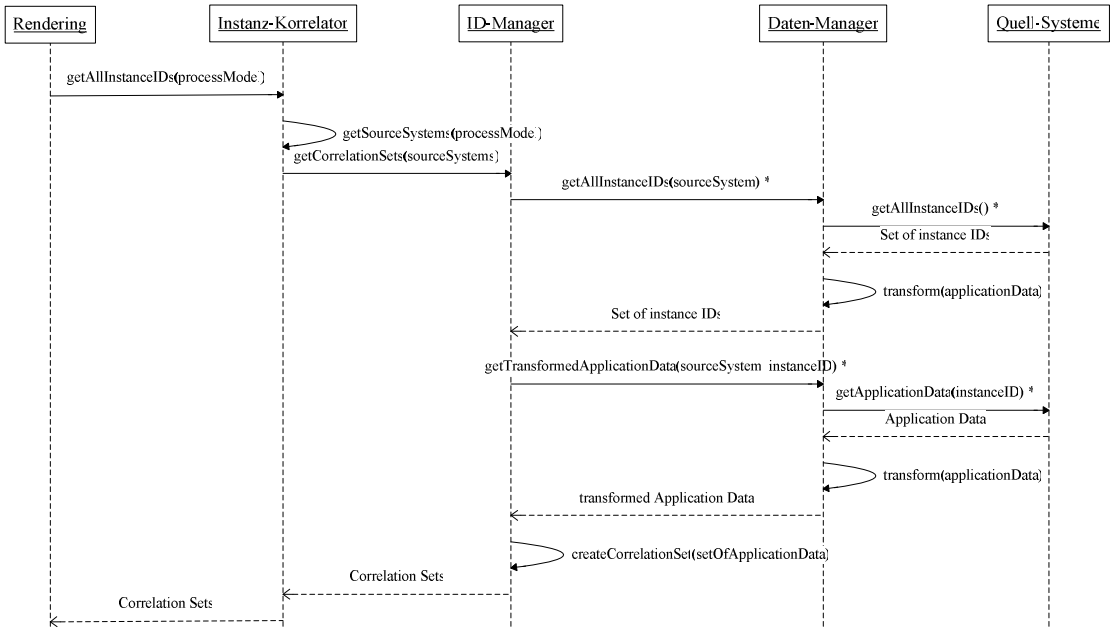


Abbildung 7.9 Sequenzdiagramm – Beschaffung aller Instanzen eines Prozessmodells

Das geschieht, indem dieser zu allen angegebenen Systemen zunächst die Instanz-IDs und danach alle Applikationsdaten zu den einzelnen Instanzen vom Daten-Manager anfordert.

```
for all sourceSystems repeat
{
   Daten-Manager.getAllInstanceIDs(sourceSystem);
}

for all sourceSystems repeat
   for all instanceIDs repeat
   {
      DatenManager.getTransformedApplicationData(sourceSystem,
      instanceID);
   }
```

Der Daten-Manager lädt die Daten von den jeweiligen Quell-Systemen, transformiert sie ins richtige Format und gibt sie an den ID-Manager zurück. Dieser erzeugt aus einer

Menge von Applikationsdaten Korrelationsmengen, die bis zur Rendering-Komponente zurückgegeben werden und diesem als Instanz-IDs der globalen Prozesse dienen.

7.3 Aktualisierung der Daten für die Visualisierung

Bei der Aktualisierung der Daten für die Visualisierung gibt es zwei grundsätzliche Varianten. Entweder der Client fordert die Aktualisierung an, das nennt sich Poll oder der Server aktualisiert bei jeder Änderung des Datenbestandes, das nennt sich Push. In unserem Fall ist der Client die Rendering-Komponente und der Server die Mapping-Komponente.

In den folgenden beiden Abschnitten werden zunächst die Vor- und Nachteile der beiden Varianten betrachtet, bevor entschieden wird, welche Variante in unserem Fall die Bessere ist.

7.3.1 Poll

Bei der Poll-Variante ist der Server passiv, das bedeutet, dass der Client für die Aktualisierung zuständig ist.

Vorteil dieser Variante ist, dass der Client selber entscheiden kann, wann aktualisiert werden soll. Dadurch kann eine sehr niedrige Kommunikationslast erreicht werden.

Allerdings können sich bei falscher Konfiguration, d.h. zu häufige oder zu seltene Aktualisierungsanfragen, auch Probleme ergeben.

Bei zu häufigem Nachfragen wird u.U. eine unnötig hohe Kommunikationslast erzeugt, wenn es beispielsweise keine Änderungen gab. Bei zu seltenem Nachfragen sind dagegen die Daten u.U. veraltet.

Eine mögliche Lösung für diese Probleme ist eine manuell und nicht zeitgesteuerte Anforderung der Daten. So kann der Benutzer die Daten aktualisieren lassen, wenn er diese wirklich aktuell braucht. Die Gefahr, dass der Benutzer die Aktualisierung vergisst und somit mit u.U. veralteten Daten arbeitet, besteht hier allerdings auch.

Diese Gefahr kann ein wenig reduziert werden, indem zusätzlich zur manuellen Aktualisierung eine zeitgesteuerte Aktualisierung mit einem hohen Aktualisierungsintervall benutzt wird. Dadurch entsteht keine unnötig hohe Kommunikationslast und dennoch können die Daten bei Bedarf auf dem aktuellsten Stand sein.

7.3.2 Push

Bei der Push-Variante ist der Server aktiv und informiert den Client bei jeder Veränderung des Datenbestandes.

Dadurch sind die Daten stets aktuell und es wird nur dann kommuniziert, wenn es wirklich etwas zu aktualisieren gibt. Allerdings kann dadurch bei vielen Änderungen auch sehr viel kommuniziert werden, obwohl u.U. keine Visualisierung benötigt wird. Somit wird eventuell doch unnötig kommuniziert.

Eine Möglichkeit dieses Problem zu lösen, ist das zeitgesteuerte Push. Diese Variante wird im ADEPT-System benutzt [ReDa02]. Dabei wird für das Push ein Aktualisierungs-Intervall vorgegeben. Somit wird nur nach der vorgegebenen Zeit und nur wenn es wirklich Änderungen gab kommuniziert. Um dadurch keine wichtigen Nachrichten zu verpassen, werden Prioritäts-Nachrichten unabhängig vom vorgegebenen Zeitintervall sofort weitergeleitet.

Betrachtet man die Vor- und Nachteile der beiden Varianten und unseren konkreten Anwendungsbereich, so wird schnell klar, dass nur eine Poll-Lösung wirklich in Frage kommt. Denn in unserem Fall gibt es viele Systeme, die u.U. ständig laufen und sehr viele Änderungen haben können. Der Client dagegen läuft nur, wenn eine Visualisierung eines Prozesses benötigt wird. Eine Push-Realisierung würde also sehr häufig zu unnützer Kommunikationslast führen.

Somit ist eine zeitgesteuerte Poll-Realisierung mit zusätzlicher Möglichkeit der manuellen Aktualisierung die beste Variante.

Dafür ist in den Schnittstellen der Komponenten bereits ein Parameter „time“ vorgesehen, der die letzte Aktualisierung angibt. Dadurch müssen nur noch die neuen Daten geladen werden, wodurch die Kommunikationslast zusätzlich reduziert wird.

7.4 Zusammenfassung

In diesem Abschnitt wurde eine Architektur für eine Mapping-Komponente entwickelt, die die Daten der verschiedenen Systeme auf ein einheitliches Metamodell abbildet und dabei die Probleme löst, die im ersten Teil der Arbeit identifiziert wurden. Die Mapping-Komponente ist Teil der Visualisierungs-Komponente, die u.a. noch eine Rendering-Komponente enthält. Diese baut auf den Ergebnissen der Mapping-Komponente auf und erzeugt aus diesen Prozessmodelle.

Die Mapping-Komponente besteht aus zwei Teilen, aus der Buildtime- und der Runtime-Komponente. Beide Komponenten müssen zunächst heterogene Daten integrieren, bevor sie weitere Aufgaben lösen können. Die Buildtime-Komponente ist für Prozesstransformation und Integration zuständig. Ein Teil dieser Aufgaben kann die Beschaffung fehlender Daten sein, wofür die Buildtime-Komponente Subkomponenten, wie einen Wrapper, Process Miner und Log-Transformator enthält.

Die Runtime-Komponente ist für die Integration der Instanzdaten zuständig. Dabei sind die durchzuführenden Aufgaben teilweise abhängig von durchgeführten Transformationen und Integrationen der Buildtime-Komponente.

Zur Aktualisierung der Daten wurde eine zeitgesteuerte Poll-Variante mit zusätzlicher Möglichkeit der manuellen Aktualisierung gewählt. Damit wird überflüssige Kommunikation minimiert, wobei die Aktualität der Daten, je nach Bedarf, flexibel gestaltet werden kann.

8 Beispiel einer Prozesstransformation

In Kapitel 4 wurden die Probleme vorgestellt, die bei der Transformation von Prozessmodellen auftreten können. Durch die Vielzahl der Metamodelle und durch die speziellen Probleme, die bei jedem Paar von Quell- und Zielmodell unterschiedlich sind, ist eine allgemeine Lösung für die Transformation nicht möglich. Deswegen wird in diesem Kapitel die Prozesstransformation anhand eines Beispiels demonstriert. Dazu werden eine Wertschöpfungskette und eine erweiterte ereignisgesteuerte Prozesskette (eEPK) des ARIS Toolsets auf ein kanonisches Modell abgebildet.

In Abschnitt 8.1 werden zunächst die wichtigsten Grundlagen für die benutzten Modelle eingeführt, bevor in Abschnitt 8.2 schrittweise die einzelnen Elemente, dann die Konstrukte und zum Schluss der ganze Prozess transformiert und die Grenzen der automatischen Transformation aufgezeigt werden.

8.1 Grundlagen

In den folgenden beiden Abschnitten werden Grundlagen eingeführt, die für das in diesem Kapitel behandelte Transformationsbeispiel benötigt werden. Zu diesen Grundlagen gehören das ARIS Toolset, dessen Prozessmodell das Quellmodell darstellen wird, und ein kanonisches Modell, auf das die Abbildung erfolgt.

8.1.1 ARIS Toolset

Das ARIS-Toolset ist ein Geschäftsprozessmodellierungswerkzeug, das neben der Erstellung von Geschäftsprozessen auch ihre Simulation und Analyse ermöglicht. Dafür bietet ARIS (Architektur integrierter Informationssysteme) eine Vielzahl an Methoden und Modellierungsmöglichkeiten. Für die Modellierung von Unternehmensabläufen eignen sich Wertschöpfungsketten und erweiterte ereignisgesteuerte Prozessketten (eEPKs) am besten. Mit Hilfe dieser beiden Diagrammtypen lassen sich Geschäftsprozesse hierarchisch auf mehreren Ebenen und somit in unterschiedlichen Detailstufen abbilden.

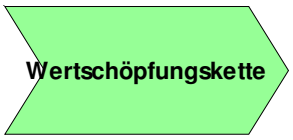
Auf die Modelldaten kann über eine Datenbank, eine exportierte XML-Datei oder *Reports* zugegriffen werden. Laufzeitdaten gibt es in ARIS nicht, und diese werden auch nicht benötigt, da für eine reine Prozesstransformation alle nötigen Daten bereitgestellt werden. Die Transformation und Migration dieser Daten führt der Datenmanager mit Hilfe der in Kapitel 3 vorgestellten Technologien durch. Somit kann von einer Datenbasis ausgegangen werden, die in einem vom Transformator lesbaren Format vorliegt.

In den folgenden beiden Abschnitten werden Wertschöpfungskettendiagramme und eEPKs mit ihren wichtigsten Elementen vorgestellt. Dabei werden wir uns bei den eEPKs auf die allgemeinen Elemente beschränken.

8.1.1.1 Wertschöpfungskettendiagramm

Auf dem Weg vom Lager zum Verbraucher durchlaufen Rohstoffe in einem Unternehmen eine Reihe von Veränderungen. Bei jeder dieser Veränderungen gibt es eine Wertsteigerung (auch Wertschöpfung genannt) und somit stellen diese Veränderungen einzelne Wertschöpfungsstufen dar. Durch ihre Aneinanderreihung bilden die Wertschöpfungsstufen eine Wertschöpfungskette, die in ARIS in Wertschöpfungskettendiagrammen abgebildet werden kann.

In ARIS gibt es keine Unterscheidung zwischen Wertschöpfungsstufe und Wertschöpfungskette. Einziges Element in diesem Diagrammtyp ist die Wertschöpfungskette. Die Unterscheidung ist nicht notwendig, da die Wertschöpfungsketten hierarchisch aufgebaut sein können und somit jede Wertschöpfungsstufe aus einer Wertschöpfungskette bestehen kann. Dabei beschreibt eine niedrigere Hierarchiestufe eine Wertschöpfungskette der übergeordneten Stufe detailliert.



Wertschöpfungsketten spezifizieren Funktionen, die direkt an der Wertschöpfung eines Unternehmens beteiligt sind.

Die Funktionen können miteinander verbunden sein. Je nach Anordnung kann die Verbindung die zeitliche Abfolge der Funktionen deutlich machen oder ihre hierarchische Anordnung [IDS04].

Abbildung 8.1 zeigt ein Beispiel einer Wertschöpfungskette auf drei Hierarchiestufen. Näheres zu Wertschöpfungsketten findet man in [IDS04].

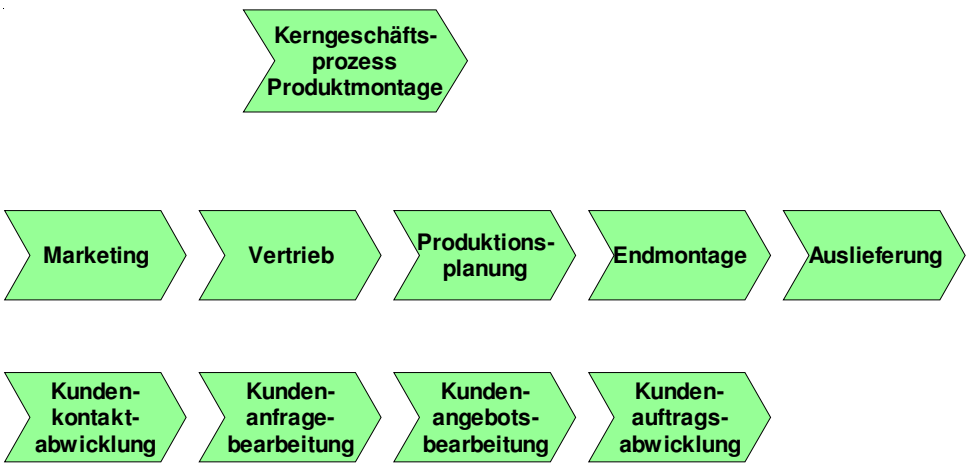


Abbildung 8.1 Beispiel einer Wertschöpfungskette

8.1.1.2 Erweiterte ereignisgesteuerte Prozessketten

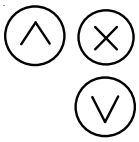
Erweiterte ereignisgesteuerte Prozessketten (eEPKs) beschreiben den detaillierten Ablauf von Geschäftsprozessen. Für jede Funktion der eEPK können Start- und Endereignisse angegeben werden. Ereignisse sind somit Auslöser und Ergebnisse von Funktionen [IDS04]. Im Folgenden werden zunächst die wichtigsten Elemente einer eEPK beschrieben, bevor ihre Anwendung in Abbildung 8.2 demonstriert wird.

Kontrollflusselemente:

Ein Ereignis beschreibt das Eingetretensein eines Zustandes, der eine Folge von Funktionen auslösen kann.

Funktionen beschreiben Arbeitsschritte, die für die Ausführung eines Geschäftsprozesses benötigt werden. Die Funktionen können elementare Arbeitsschritte beschreiben, die betriebswirtschaftlich nicht mehr zerlegbar sind, oder sie können durch eine Reihe weiterer Funktionen verfeinert werden. Dann stellen sie einen Subprozess dar. Durch die Verfeinerung wird eine hierarchische Abbildung der Funktionen erreicht, bei der die Funktion auf der höchsten Stufe grob und auf niedrigeren Stufen detailliert abgebildet ist.

Die Prozessschnittstelle verweist auf einen vorhergehenden oder nachfolgenden Prozess. Sie kann somit zur Partitionierung von Prozessen benutzt werden.



Regeln beschreiben unterschiedliche Formen von Verzweigungen und Zusammenführungen. Je nach Symbol beschreiben sie einen UND-, OR- oder XOR-Split/Join.

Kanten gibt es sowohl für den Kontroll- als auch für den Datenfluss. Für den Kontrollfluss beschreiben sie die zeitliche Abfolge der Funktionen, für den Datenfluss zeigen sie welche Daten Input oder Output von Funktionen sind.

Organigrammelemente:

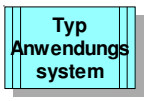
Beschreibt eine organisatorische Einheit im Unternehmen.

Eine Stelle ist die kleinste zu identifizierende Organisationseinheit in einem Unternehmen.

Eine Gruppe stellt die Gruppierung von Personen dar, die z.B. zur Lösung einer bestimmten Aufgabe für einen bestimmten Zeitraum zusammenarbeiten, z.B. in einem Projekt.

Personentypen umfassen Personen nach ihrer Position oder ihren Aufgaben im Unternehmen, wie z.B. Abteilungsleiter, Gruppenleiter, usw.

Daten- und Ressourcenelemente:



Ein Anwendungssystemtyp ist ein Software-Typ, der für bestimmte Funktionen verwendet wird, wie z.B. Lagerverwaltung, Vertrieb, usw.

Dokumente entstehen bei der Ausführung von Funktionen und dienen ebenfalls Funktionen als Input.

Fachbegriffe stellen die in einem Unternehmen zur Beschreibung der betrachteten Informationsobjekte existierenden Begrifflichkeiten dar.

Abbildung 8.2 zeigt die eEPK-Elemente, die für den ersten Teil des ChangeManagement-Prozesses benötigt werden.

Der Prozess wurde, im Vergleich zu den bisherigen Darstellungen, um Stellen, Dokumente und Anwendungssystemtypen ergänzt.

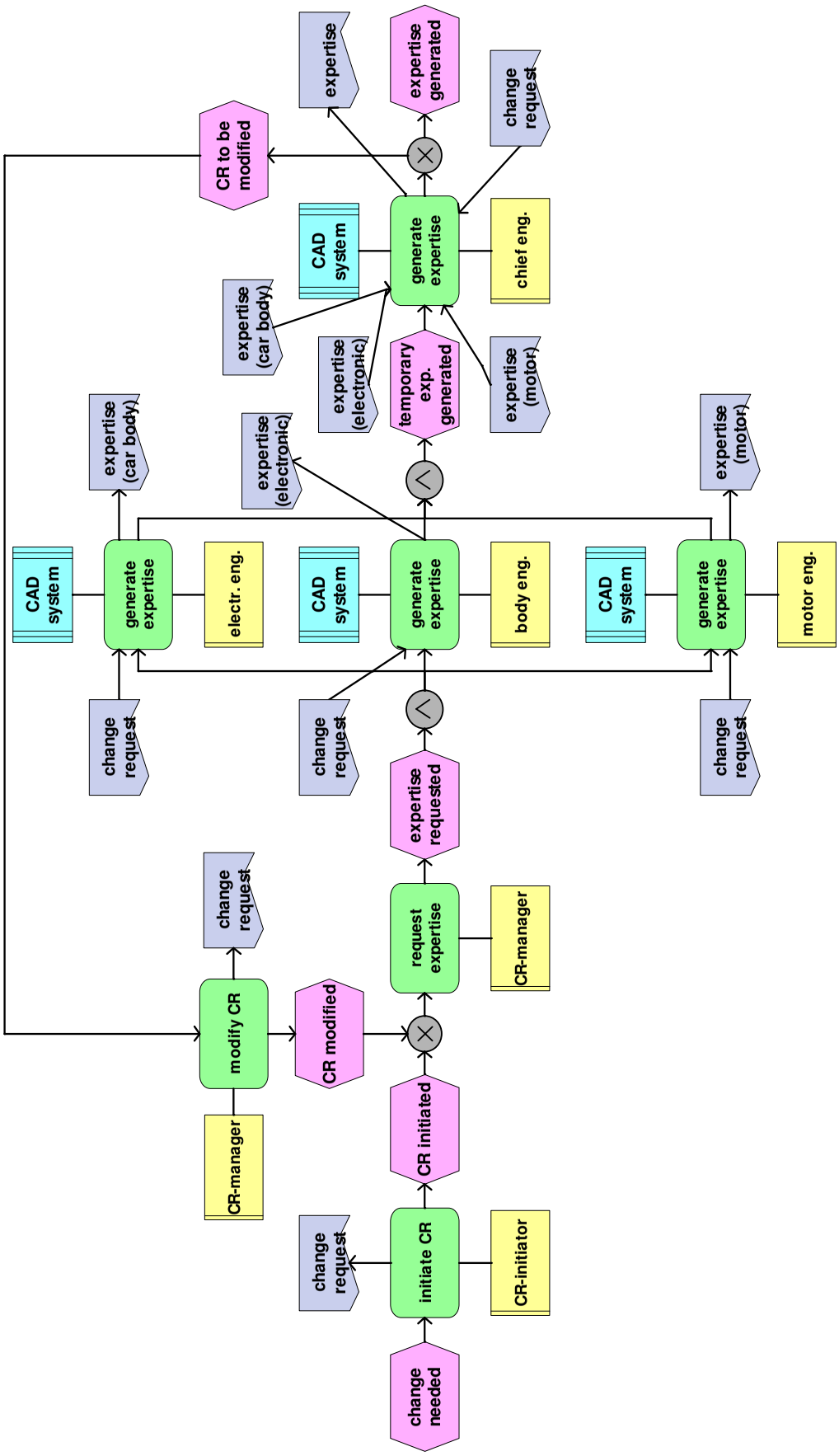


Abbildung 8.2 Change Management (erster Teil) als ARIS-eEPK

8.1.2 Kanonisches Metamodell

Nachdem die Elemente und Konstrukte des Quellmodells beschrieben wurden, wird im Folgenden ein Zielmodell vorgestellt, auf das diese abgebildet werden. Wie bereits in Abschnitt 2.2.2 diskutiert, wird dabei ein kanonisches Modell verwendet, da sowohl ein minimales, als auch ein maximales Modell oder ihre Variationen erhebliche Nachteile aufweisen.

Aber auch bei der Wahl des kanonischen Modells gibt es mehrere Möglichkeiten. Bevor man sich für ein Modell entscheidet, muss man sich überlegen, welche Anforderungen an so ein Metamodell gestellt werden. Anhand dieser Anforderungen können Metamodelle auf ihre Tauglichkeit als kanonisches Metamodell geprüft werden. Die wichtigsten Anforderungen sind [Men04, Bob05]:

- Allgemeingültigkeit: es sollen Prozesse aus allen möglichen Bereichen und all ihren Szenarien visualisiert werden können, egal in welcher Beschreibungssprache sie beschrieben wurden und in welchem Prozess-Management-System diese ablaufen.
- Vollständigkeit: alle in der Praxis vorkommenden Vorgänge und ihre Elemente, inklusive ihrer Attribute sollen abgebildet werden können.
- Verständlichkeit: das Modell soll einfach zu verstehen sein, um in der Praxis Akzeptanz zu erlangen.
- Eindeutigkeit: die visualisierten Elemente sollten den Zweck und die Bedeutung der symbolisierten Elemente deutlich machen. Semantisch unterschiedliche Elemente sollten auch unterschiedlich dargestellt werden.
- Erweiterbarkeit: eine der wichtigsten Anforderungen, soll sicherstellen, dass zusätzliche, zunächst nicht vorgesehene Informationen, eingebracht werden können.

Betrachtet man die normalen Metamodelle, die bei der Geschäftsprozessmodellierung verwendet werden, wie z.B. EPKs, Petrinetze oder Aktivitätennetze, so wird schnell klar, dass diese mindestens zwei der wichtigsten Anforderungen nicht erfüllen. Zum einen sind sie nicht vollständig. Für diese Metamodelle lassen sich unter den Workflow Patterns von W.M.P. van der Aalst [AHKB03] leicht Kontrollkonstrukte finden, die nicht umgesetzt werden können. Zum anderen sind die erwähnten Metamodelle alle nicht erweiterbar. Dadurch können die fehlenden Elemente oder Konstrukte nicht hinzugefügt werden.

Ein Metamodell, dass all die erwähnten Kontrollkonstrukte unterstützt ist YAWL (Yet Another Workflow Language) [AaHo02]. YAWL wurde an der Queensland University of Technology speziell zur Unterstützung aller Workflow Patterns entwickelt. Mittlerweile unterstützt YAWL auch alle *Data* und *Resource Patterns* [RHEA04a, RHEA04b].

Dadurch ist das Modell allerdings durch Elemente überladen, die bei den meisten Prozessen nicht benötigt werden, und die Anforderung der Verständlichkeit wird nicht mehr erfüllt.

Ein Metamodell, das alle oben genannten Anforderungen erfüllt, ist ein Metamodell von Ralph Bobrik [Bob05]. Dieses wurde im Rahmen des Projektes SPI (*Seamless Partner Integration*) bei DaimlerChrysler entwickelt, speziell im Hinblick auf die oben aufgeführten Anforderungen.

Das Modell ist übersichtlich gehalten, indem Attribute allgemein für Knoten, Kanten und für das Prozessmodell definiert werden. Diese Knoten und Kanten werden danach Schritt für Schritt spezialisiert. Somit ist das Metamodell sehr verständlich.

Außerdem ist das Modell allgemein und eindeutig beschrieben, so dass semantisch unterschiedliche Elemente auch unterschiedlich abgebildet werden können, unabhängig davon aus welchen Bereichen die beschriebenen Prozesse stammen.

Durch die Erweiterbarkeit des Modells, lassen sich alle in der Praxis vorkommenden Elemente inklusive ihrer Attribute abbilden. Sollten diese fehlen, können einfach benutzerdefinierte Elemente und Attribute angelegt werden.

Das Modell wird an dieser Stelle nicht näher beschrieben. Denn die für die Abbildung des ARIS-Modells benötigten Elemente und Attribute werden vor ihrer Verwendung in Abschnitt 8.2 erklärt. Die komplette Dokumentation des Metamodells findet sich in [Bob05].

8.2 Prozesstransformation

Nach den Grundlagen im vorangegangenen Abschnitt sollen nun Wertschöpfungsketten und eEPKs aus ARIS auf das vorgestellte kanonische Modell abgebildet werden. Dazu werden zunächst einzelne ARIS-Elemente und Konstrukte auf das kanonische Metamodell abgebildet bevor ein kompletter Prozess abgebildet wird. Manche Abbildungen können dabei direkt durchgeführt werden, manche nur mit Hilfe von Transformationen und bei einigen stößt man an die Grenzen der automatischen Transformation. Bei der Transformation werden, wenn möglich, allgemeine Elemente aus ARIS, mit Hilfe struktureller Informationen aus dem Modell, aufgewertet.

8.2.1 Abbildung der ARIS-Elemente auf das kanonische Modell

8.2.1.1 Wertschöpfungskettendiagramm

Das einzige Element des Wertschöpfungskettendiagramms, die Wertschöpfungskette, wird in ARIS als Funktion repräsentiert, die auf eine Aktivität vom Typ *Subprozess* im kanonischen Modell abgebildet wird. Das Attribut *Subprozess* der Aktivität ist vom Typ

Link und verweist auf den Subprozess, der die Funktion bzw. nach der Abbildung die Aktivität verfeinert. (s. Abbildung 8.3)

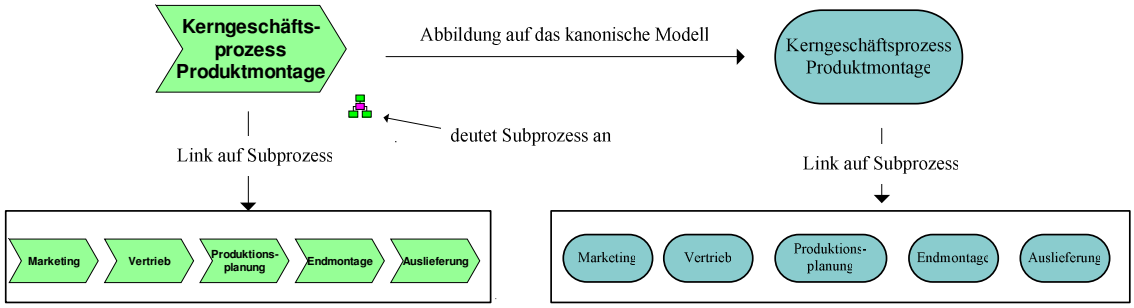


Abbildung 8.3 Abbildung einer Wertschöpfungskette auf das kanonische Modell

Der Grund für diese Abbildung ist, dass die Funktionen der Wertschöpfungskette, wie in Abschnitt 8.1.1.1 beschrieben, hierarchisch angeordnet sind. Funktionen werden durch Funktionen einer niedrigeren Ebene verfeinert bzw. auf der untersten Ebene der Wertschöpfungskette durch eEPKs. Somit sind diese Funktionen nichts anderes als Subprozesse.

8.2.1.2 eEPK

Bei eEPKs ist die Abbildung der Elemente nicht so einfach, da es anders als bei der Wertschöpfungskette nicht zu allen Elementen genau das passende Element des kanonischen Modells gibt. An solchen Stellen müssen Transformationen vorgenommen werden. Im Folgenden werden die vorgestellten eEPK-Elemente auf Elemente des kanonischen Modells abgebildet.

8.2.1.2.1 Ereignis

Die meisten Ereignisse in ARIS, bis auf zwei Ausnahmen, drücken das Ende einer Funktion aus, z.B. „CR requested“. Das ist in den meisten Metamodellen, wie auch beim verwendeten kanonischen Modell, implizit gegeben. Das bedeutet, dass die nachfolgende Aktivität erst dann ausgeführt werden kann, wenn die Vorhergehende beendet ist, ohne dafür explizit ein Ereignis angeben zu müssen. Somit können, wenn erwünscht, diese Ereignisse weggelassen werden. Das kann man entweder während der Transformation machen, indem man die Ereignisse auf das kanonische Metamodell nicht abbildet oder bei der Integration, indem man die im kanonischen Modell vorliegenden Ereignisse löscht. Die erste Variante ist effizienter, bei der zweiten dagegen ist das Vorgehen übersichtlicher und flexibler. Deswegen wird bei kommenden Algorithmen die zweite Variante betrachtet. In beiden Fällen müssen die Kanten der weggelassenen Ereignisse angepasst werden. Was dabei zu beachten ist, wird in Abschnitt 8.2.1.2.5 behandelt.

Die erwähnten Ausnahmen sind zum einen das Startereignis und zum anderen externe Ereignisse. Diese können zwar vom Prozess nicht beeinflusst werden, aber auf ihr Eintreten muss gewartet werden (z.B. auf das Eintreffen eines Briefes). Beide Arten haben gemeinsam, dass sie im Prozess keinen Vorgänger haben. Somit können sie beispielsweise mit Hilfe von Grapherreichbarkeitsalgorithmen leicht identifiziert werden.

Der Unterschied zwischen diesen beiden Ereignisarten ist, dass Startereignisse einen Prozess instanziieren, externe Ereignisse dagegen nicht. Diese können zwar auch einen Bearbeitungszweig starten, allerdings nur einen, der in einem bereits instanziiertem Prozess liegt. Da es keine weiteren Unterschiede gibt und in ARIS keine Attribute dafür vorgesehen sind, ist eine algorithmische Unterscheidung zwischen Startereignissen und externen Ereignissen nicht möglich.

Es bleibt nur die Möglichkeit, beide Ereignisarten auf ein Ereignis vom Typ *Startereignis* im kanonischen Modell abzubilden und bei der manuellen Nachbearbeitung die Unterscheidung zu treffen.

Eine semantische Aufwertung ist bei dem Ereignis oder den Ereignissen möglich, die das Ende des Prozesses darstellen. Durch Überprüfung des Graphen auf Ereignisse ohne Nachfolger kann man diese auch leicht identifizieren und auf Ereignisse vom Typ *Endereignis* abbilden.

8.2.1.2.2 Funktion

Funktionen von ARIS werden auf Aktivitäten des kanonischen Modells abgebildet. Um einen genaueren Typ im kanonischen Modell angeben zu können und die Elemente somit semantisch aufzuwerten, gibt es zwei Möglichkeiten weitere Informationen aus dem ARIS-Modell zu gewinnen. Zum einen aus den Attributen, die zu den Elementen gespeichert werden und zum anderen aus der Art der verbundenen Elemente.

- Attribute: Unter den Attributen der Funktion in ARIS kann unter dem Punkt *Bearbeitungsart* die Funktion genauer spezifiziert werden. Im Folgenden werden die Bearbeitungsarten und ihre Abbildung auf das kanonische Modell vorgestellt:
  - Manuell: Eine manuelle Funktion wird auf eine manuelle Aktivität im kanonischen Modell abgebildet.
  - Automatisch: Eine automatische Funktion wird von einem System ausgeführt und kann auf eine automatische Aktivität abgebildet werden.
  - Batch: Eine Batch-Funktion wird EDV-unterstützt und ohne Benutzereingriff ausgeführt und kann somit ebenfalls auf eine automatische Aktivität abgebildet werden.

- Online: Eine Online-Funktion wird EDV-unterstützt von Benutzern ausgeführt und wird somit auf eine interaktive Aktivität des kanonischen Modells abgebildet werden.

- Verbundene Elemente: Falls in den Attributen keine genaue Bearbeitungsart spezifiziert ist, dann müssen aus den verbundenen Elementen Rückschlüsse gezogen werden. Die wichtigsten Elemente dabei sind Anwendungssysteme und Elemente des Organisationsmodells wie z.B. Personen.
  - Ist eine Funktion nur mit einem Anwendungssystem verbunden, von dem sie bearbeitet wird, so wird diese auf eine automatische Aktivität im kanonischen Modell abgebildet. (s. Abbildung 8.4 a.)
  - Wird eine Funktion nur von einem Element des Organisationsmodells bearbeitet, so wird sie auf eine manuelle Aktivität abgebildet. (s. Abbildung 8.4 b.)
  - Ist eine Funktion gleichzeitig mit einem Anwendungssystem und mit einem Element des Organisationsmodells verbunden, so wird sie auf eine interaktive Aktivität abgebildet. (s. Abbildung 8.4 c.)

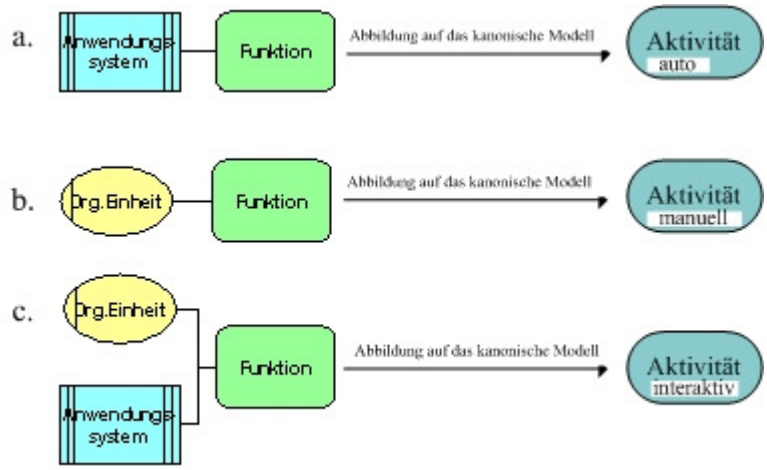


Abbildung 8.4 Semantische Aufwertungen bei der Transformation

8.2.1.2.3 Prozessschnittstelle

Prozessschnittstellen aus ARIS werden auf Ereignisse vom Typ *Prozessschnittstelle* abgebildet. Allerdings werden die Prozessschnittstellen von ARIS zusammen mit einem Ereignis auf das Ereignis des kanonischen Modells abgebildet. Bei der Erklärung soll Abbildung 8.5 helfen. Wie man dort sieht, sind die beiden Ereignisse vor bzw. im zweiten Prozess nach der Schnittstelle identisch. Das ist eine Voraussetzung für die Prozessschnittstellen. Fasst man eine Prozessschnittstelle und das Ereignis „Y beendet“ zu-

sammen, dann sind diese beiden Elemente zusammen das gleiche wie im kanonischen Modell ein Ereignis vom Typ Prozessschnittstelle. Dieses Ereignis hat ein Attribut *Partnerereignis*, das einen Verweis auf das identische Ereignis im anderen Prozess darstellt.

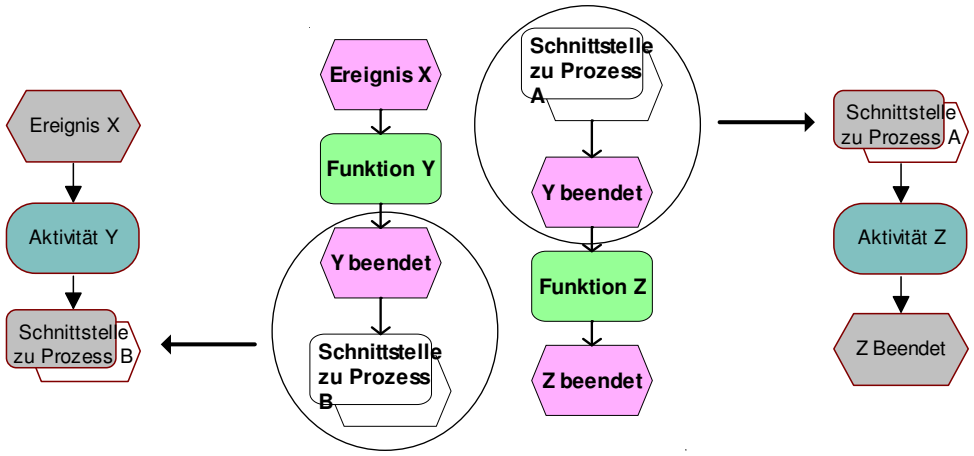


Abbildung 8.5 Abbildung einer Prozessschnittstelle

8.2.1.2.4 Regel

Die ARIS-Regeln werden im kanonischen Modell auf Strukturknoten abgebildet. Das sind leere Aktivitätsknoten vom Typ *empty*, für die keine Aktivität hinterlegt ist, die aber durch ihre Eingangs- und Ausgangssemantik für die Weglenkung eingesetzt werden können. Dafür wird ihr Strukturtyp auf *Split/Join* gesetzt. Zudem müssen ihre Eingangs- und Ausgangssemantiken auf die entsprechenden ARIS-Regeln angepasst werden. Dazu werden die Attribute *In-Typ* bzw. *Out-Typ*, je nach Regel, auf *XOR*, *OR* oder *AND* gesetzt.

8.2.1.2.5 Kanten

Kanten werden zu unterschiedlichen Zwecken eingesetzt und haben somit auch unterschiedliche Semantiken. Deswegen werden diese im Folgenden getrennt betrachtet.

- Kontrollflusskanten:

  Kontrollflusskanten in ARIS sind gerichtet und verbinden Funktionen, Ereignisse und Regeln miteinander. Je nachdem welche zwei Elemente sie verbinden, sind sie von unterschiedlichem Typ:

  - Wird ein Ereignis mit einer Funktion verbunden, so ist die Kante vom Typ *aktiviert*.
  - Verbindet sie eine Funktion mit einem Ereignis, so ist sie vom Typ *erzeugt*.

- Werden zwei Funktionen miteinander verbunden, so ist die Kante vom Typ *ist_Vorgänger_von.*
- Wird eine Regel mit einem der beiden anderen Elemente verbunden, so ist die Kante vom Typ *führt_zu*.

Bei der Abbildung der Kontrollflusskanten müssen zwei der bisherigen Abbildungen berücksichtigt werden. Zum einen gibt es keine Regel-Knoten mehr, sondern stattdessen leere Aktivitätsknoten (s. Abschnitt 8.2.1.2.4). Dadurch werden keine *führt_zu*-Kanten mehr benötigt. Zum anderen wurden u.U., wie in Abschnitt 8.2.1.2.1 erwähnt, die Ereignisse weggelassen, wodurch auch keine *aktiviert*- und *erzeugt*-Kanten mehr nötig wären.

Die Abbildung der Kanten vom Typ *ist_Vorgänger_von* ist dabei ganz einfach und kann direkt auf eine Kontrollflusskante des kanonischen Modells mit denselben Quell- und Zielknoten abgebildet werden.

Bei Kanten vom Typ *führt_zu* muss darauf geachtet werden, dass die Regeln auf leere Aktivitäten abgebildet wurden. Entsprechend müssen auch die Attribute, die Quell- und Zielknoten angeben, auf diese angepasst werden. Die Kante wird danach auf eine Kontrollflusskante des kanonischen Modells abgebildet.

Kanten von den Typen *erzeugt* und *aktiviert* werden auf einfache Kontrollflusskanten abgebildet. Diese Abbildung muss nur dann durchgeführt werden, falls die Ereignisse beibehalten werden.

Ansonsten, wenn Ereignisse aus Optimierungsgründen weggelassen werden, gibt es diese Kantentypen nicht mehr. Denn beim Weglassen eines Ereignisses wird das Ziel seiner eingehenden Kante das Ziel der bisherigen ausgehenden Kante. Danach wird die ausgehende Kante, die vom Typ *aktiviert* ist, gelöscht und der Typ der eingehenden Kante, vom Typ *erzeugt*, wird auf *ist_Vorgänger_von* geändert. Die beiden Kanten wurden somit zu einer zusammengefasst und das Ereignis kann gelöscht werden. Das wird in Abbildung 8.6 und im darauf folgenden Algorithmus (*deleteEvent(E)*) verdeutlicht. Dieser Algorithmus kann benutzt werden, wenn die optionale Eliminierung der Ereignisse bei der Prozessintegration durchgeführt wird.

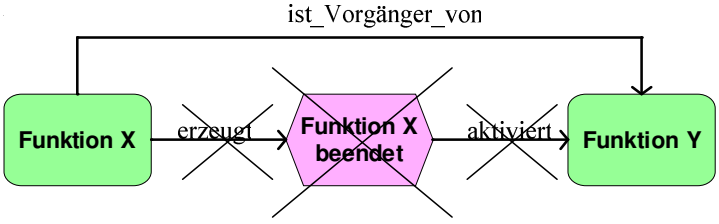


Abbildung 8.6 Beseitigung von Ereignissen des ARIS-Modells

deleteEvent(E)
//Löschen des angegebenen Ereignisses. Impliziert die Anpassung der
//eingehenden („in“) und ausgehenden („out“) Kante

input:
E: Ereignis, das gelöscht werden soll

begin:

//„destination“ ist das Ziel der Kante, somit ist „in.destination“ das Ziel der

//eingehenden Kante und „out.destination“ das Ziel der Ausgehenden.
in.destination := out.destination;
in.type:=“ist_Vorgänger_von“;
delete out;
delete E;

end

- Datenflusskanten:

  Datenflusskanten in ARIS sind gerichtet und verbinden Datenelemente mit einer Vielzahl von anderen Elementen. Je nachdem mit welchem Element die Datenelemente verbunden werden, sind sie von unterschiedlichem Typ. Die wichtigen Typen sind diejenigen, die Funktionen mit Datenelementen verbinden.

  - Geht die Kante von der Funktion zum Datenelement, so ist sie vom Typ *liest* oder *erzeugt_Output_auf*. Die *liest*-Kante wird auf eine Datenflusskante vom Typ *lesend* im kanonischen Modell abgebildet.

    Bei der *erzeugt_Output_auf*-Kante kann semantisch aufgewertet werden, indem untersucht wird, ob das Datenelement bereits existiert oder noch nicht. Existiert es noch nicht, dann wird die Kante auf eine Datenflusskante vom Typ *erzeugend* abgebildet, ansonsten auf den Typ *überschreibend* oder *anfügend*. Um diese Unterscheidung treffen zu können, muss entweder die jeweilige Funktion oder das Datenelement untersucht werden.

  - Geht die Kante vom Datenelement zur Funktion, so ist sie vom Typ *liefert_Input_für* und kann einfach auf eine Datenflusskante vom Typ *lesend* abgebildet werden.

- Bearbeitungskanten:

  Bearbeitungskanten in ARIS sind ungerichtet und ordnen Funktionen Bearbeiter zu. In welcher Form der Bearbeiter an der Funktion beteiligt ist, hängt vom Typ der Kante ab. Von den wichtigsten Typen wird im Folgenden die Abbildung auf das kanonische Modell beschrieben:

  - Kanten vom Typ *führt_aus* werden im kanonischen Modell auf eine Bearbeitungskante abgebildet, bei der die Funktion der Ausführungseinheit (s. Abschnitt 8.2.1.2.6) auf *Bearbeiter* gesetzt wird.
  - Kanten vom Typ *ist_DV-verantwortlich_für* oder *ist_fachlich_verantwortlich_für* werden auf Bearbeitungskanten abgebildet, bei der die Funktion der Ausführungseinheit (s. Abschnitt 8.2.1.2.6) auf *Verantwortlicher* gesetzt wird.

- Ressourcenkante

  Ressourcenkanten verbinden Ressourcen, wie Anwendungssysteme, mit Funktionen oder Datenelementen. Auch hier haben sie, je nach Element, unterschiedliche Typen.

  - Kanten, die Funktionen mit Ressourcen verbinden sind ungerichtet und vom Typ *unterstützt*. Sie werden auf ungerichtete Ressourcen-Kanten vom Typ *unterstützt* abgebildet.
  - Kanten, die Datenelemente mit Ressourcen verbinden sind gerichtet und je nach Richtung entweder vom Typ *liefert_Input_für* oder *erzeugt_Output_auf*. Sie werden genauso abgebildet wie die entsprechenden Datenflusskanten.

8.2.1.2.6 Ausführungseinheiten

Ausführungseinheiten erlauben eine flexible Bearbeiterzuordnung und somit eine bessere Lastverteilung. Die wichtigsten Ausführungseinheiten des ARIS-Toolsets und ihre Abbildung auf das kanonische Modell werden im Folgenden vorgestellt:

- Organisationseinheit

  Organisationseinheiten aus dem ARIS-Modell werden auf Ausführungseinheiten vom Typ *Organisationseinheit* abgebildet. Dieser Typ muss im Metamodell noch erstellt werden. Das ist nötig, da eine Spezialisierung der Organisationseinheit auf eines der bestehenden Typen (Person, Abteilung, Rolle, Funktion), eine erhebliche Einschränkung der Flexibilität bei der Zuweisung von Bearbeitern bedeuten würde.

- Stelle

  Eine Stelle wird auf eine Ausführungseinheit vom Typ *Rolle* abgebildet. Der Unterschied zwischen einer Stelle und einer Rolle ist lediglich, dass eine Stelle nur einem Benutzer und eine Rolle mehreren Benutzern zugeordnet werden kann. Durch die Richtung der Abbildung, von Stelle auf Rolle, stellt dies also keine Einschränkung dar.

- Gruppe

Eine Gruppe wird auf eine Ausführungseinheit vom Typ *Organisationseinheit* abgebildet. Zusätzlich wird in der Beschreibung der Ausführungseinheit die Gruppe erwähnt. Einen neuen Typ der Ausführungseinheit anzulegen wäre an dieser Stelle sinnlos, da die Gruppe auch so eindeutig abgebildet werden kann und ansonsten nur die Übersichtlichkeit darunter leiden würde.

- Personentyp

  Ein Personentyp wird auf eine Ausführungseinheit vom Typ *Rolle* abgebildet.

8.2.1.2.7 Anwendungssystemtyp

Ein Anwendungssystemtyp wird auf das Element Ressource vom Typ *System* abgebildet.

8.2.1.2.8 Datenelemente

- Dokument

  Ein Dokument wird auf ein Datenelement vom Typ *Dokument* im kanonischen Modell abgebildet.

- Fachbegriff

  Ein Fachbegriff wird auf ein Datenelement vom Typ *Begriff* abgebildet. Dieser Typ des Datenelements muss im kanonischen Modell erzeugt werden. Die Erzeugung eines solchen Typs im kanonischen Modell ist wichtig, da der Fachbegriff auf keines der klassischen Datenelemente (z.B. *Dokument* oder *einfacher Datentyp*) abgebildet werden kann und es sonst eine Verfälschung seiner Bedeutung zur Folge hätte.

8.2.2 Transformation von Konstrukten

8.2.2.1 Verzweigungen

Verzweigungen werden in ARIS immer wie links in Abbildung 8.7 modelliert.

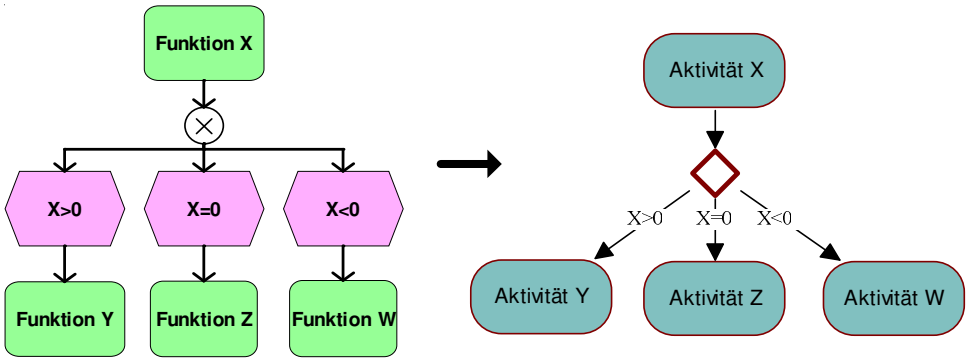


Abbildung 8.7 Verzweigung in ARIS und alternative Darstellung

Eine übersichtlichere und somit wünschenswertere Darstellung ist rechts davon zu finden. Bei dieser Darstellung sind die Ereignisse weggelassen und ihre Bedingungen stehen auf den Kanten.

Für so eine Darstellung muss man bei der Abbildung auf das kanonische Modell zunächst einmal die Ereignisse weglassen (s. Abschnitt 8.2.1.2.1) und entsprechend die Kanten anpassen (s. Abschnitt 8.2.1.2.5). Danach müssen die Namen der Ereignisse, die die Verzweigungsbedingung darstellen, auf die Beschreibung der Kanten abgebildet werden.

8.2.2.2 Schleifen

Schleifen sind in ARIS nicht explizit als solche gekennzeichnet. Da aber Schleifen ein durchaus wichtiges Konstrukt in einem Prozess sind, ist ihre explizite Abbildung durchaus sinnvoll. Dazu werden im Folgenden die ARIS-Konstrukte semantisch aufgewertet und im kanonischen Modell explizit abgebildet.

Im bereits bekannten ChangeManagement-Prozess ist eine solche implizite Schleife zwischen den Ereignissen „CR initiated“ und „expertise generated“ zu sehen (s. Abbildung 8.2).

Um das Vorgehen bei der semantischen Aufwertung zu erklären, wird von den sonstigen Konstrukten und Namen abstrahiert und ein einfacher abstrakter Prozess aus Abbildung 8.8 benutzt. Wie man dort sieht, sind Schleifenanfang und -ende jeweils eine *XOR*-Regel.

Um dieses Konstrukt explizit als Schleife abzubilden, müssen als erstes die beiden Regeln auf Kontrollkonstrukte vom Typ *Schleifenanfang* bzw. *Schleifenende* abgebildet werden.

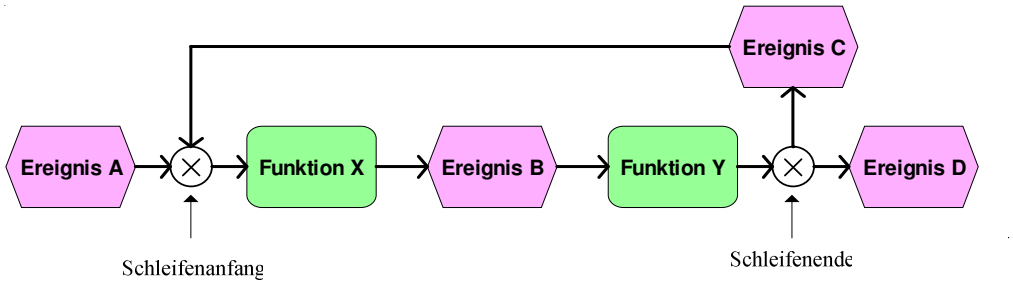


Abbildung 8.8 Beispiel einer Schleife in ARIS

Automatisch kann die Erkennung von Schleifenanfang und -ende durchgeführt werden, indem der Transformator beispielsweise alle Vorgänger und Nachfolger einer Regel ermittelt. Sind zwischen dieser und einer anderen Regel die ermittelten Vorgänger und Nachfolger gleich, so handelt es sich um eine Schleife und die beiden Regeln sind Schleifenanfang bzw. -ende und können auf die entsprechenden Konstrukte des kanonischen Modells abgebildet werden.

Nach dieser Abbildung müssen noch die bisherigen Kontrollflusskanten zwischen diesen Regeln auf Kontrollflusskanten vom Typ *Schleifenkante* des kanonischen Modells abgebildet werden.

Optional kann man auch, wie in Abschnitt 8.2.1.2.1 beschrieben, die Ereignisse weglassen. Dann müssen die Schleifenbedingungen in das Schleifenende als *Ausgangsbedingung* des Strukturknotens aufgenommen werden.

Wie man automatisch alle strukturierten Schleifen in einem ARIS-Prozess findet, wird im folgenden Algorithmus genau beschrieben. Strukturierte Schleifen sind Schleifen, die nur einen Eintritts- und Austrittspunkt haben und keine Überlappung mit anderen Schleifen zulassen [AHKB03]. (Symbole wie ❶ weisen auf Erklärungen hin, die am Ende des Algorithmus zu finden sind.)

```
getAllLoops(model)
//findet alle Schleifen im gegebenen Prozessmodell
 input:
   model: Prozessmodell, von dem alle Schleifen gesucht werden sollen
 output:
   Alle Schleifen des gegebenen Modells. Eine Schleife beinhaltet dabei Schleifen-
   anfangs- und -endknoten und alle Schleifenkanten.
 begin:
   rules := allXORRulesTopologicalSorted(model); ❶
   loops := ∅; ❷

   forall r ∈ rules do
       loopStart := r; //potentieller Schleifenanfang

       //alle XOR-Regeln holen, die Vorgänger oder Nachfolger sind, „loopStart“
       //ausgeschlossen („loopStart“ könnte sein eigener Nachfolger oder
       //Vorgänger werden)
       preds := allPrecedingXORRules(loopStart);
       succs := allSucceedingXORRules(loopStart);

       loopEnd := preds ∩ succs; //potentielles Schleifenende ❸
       edges := ∅; //enthält später alle zu einer Schleife gehörenden Kanten

       if (loopEnd ≠ ∅) then
           if ((loopEnd, loopStart) ∉ loops ) then ❹
               //alle Kanten zwischen den angegebenen Knoten
               edges := allEdgesBetween (loopStart, loopEnd);
               edges := edges ∪ allEdgesBetween (loopEnd, loopStart);
               loops := loops ∪ (loopStart, loopEnd, edges);
           endif
       endif
   done

   return loops;
 end
```

Erklärungen:

❶ Gibt alle XOR-Regeln topologisch sortiert zurück. Die XOR-Regeln sind die potentielle Schleifenanfangs- und -endknoten. Durch die topologische Sortierung ist es möglich Schleifenanfang und -ende voneinander zu unterscheiden. (siehe ❸) Eine Möglichkeit zur topologischen Sortierung findet sich in [JMSW04].

❷ Diese Menge hält, topologisch sortiert, alle Schleifenanfangs- und -endknoten, sowie die Kanten, die sie verbinden. (Ist anfangs leer.)

❸ Die Schnittmenge der beiden Mengen enthält alle XOR-Regeln, die sowohl Vorgänger, als auch Nachfolger des betrachteten Knotens sind. In einer strukturierten Schleife kann das nur ein einziger Knoten sein. Und durch die topologische Sortierung der XOR-Regel kann es nur der Endknoten sein, da die Suche zunächst vom Startknoten ausgeht. (weitere Überprüfung unter ❹)

❹ Überprüfung ob Schleifenanfang und -ende schon in der Menge der Schleifenelemente in umgekehrter Reihenfolge enthalten sind. Überprüfung muss gemacht werden, da sowohl Anfang, als auch Ende in einer Schleife Vorgänger und Nachfolger voneinander sein können. Nur wenn sie nicht enthalten sind, werden sie und die Kanten die sie verbinden, zur Menge hinzugefügt. Durch die topologische Sortierung in der strukturierter Schleife ist gewährleistet, dass der erste betrachtete Knoten einer Schleife der Startknoten ist.

8.2.2.3 Sprünge

Sprünge können in ARIS ebenfalls nur implizit dargestellt werden und werden somit in den normalen Ablauf eingebettet. Das wird in Abbildung 8.9 dargestellt.

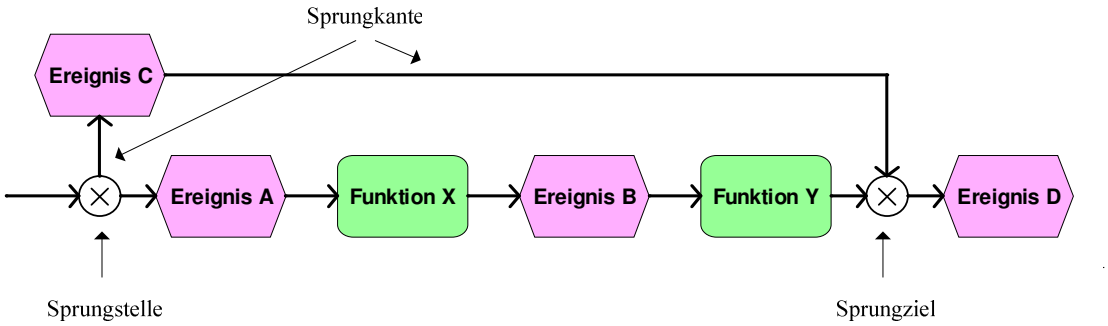


Abbildung 8.9 Beispiel für einen Sprung in ARIS

Wie bei Schleifen, ist hier ebenfalls eine explizite Abbildung im kanonischen Metamodell möglich. Dazu muss die erste Regel, von der der Sprung ausgeht, auf einen Strukturknoten vom Typ *Sprungstelle* und die Kanten zwischen dieser und der nächsten Regel auf Kontrollkanten vom Typ *Sprungkante* abgebildet werden.

Auch hier können optional die Ereignisse weggelassen werden (s. Abschnitt 8.2.1.2.1).

Die Sprungstellen findet man, indem von allen XOR-Regeln (XOR-Splits) jeweils alle ausgehenden Kontrollpfade bis zu der dazugehörigen XOR-Regel (XOR-Join) verfolgt werden. Ist auf einem Pfad keine Funktion zu finden, so müssen die Kanten zwischen den beiden XOR-Regeln auf dem betroffenen Pfad Sprungkanten sein und die erste XOR-Regel eine Sprungstelle. Wie man zu einem Split- den zugehörigen Join-Knoten findet, kann in [JMSW04] nachgelesen werden. Die restliche Vorgehensweise, wie man

die Sprünge in einem Prozessmodell findet, wird im folgenden Algorithmus verdeutlicht.

```
getAllJumps(model)
//Gibt alle Sprünge des gegebenen Prozessmodells zurück. Ein Sprung besteht aus
//Sprungstelle, Sprungziel und Sprungkanten
 input:
   model: Prozessmodell, dessen Sprünge gesucht werden sollen
 output:
   jumps: alle Sprünge des gegebenen Prozessmodells
 begin:
   splits := getAllXORSplits (model); ❶
   jumps := ∅;

   forall s ∈ splits do
      jumpStart := s; //potentielle Sprungstelle

      //Betrachtung aller von der potentiellen Sprungstelle ausgehenden Kanten
      forall b ∈ outgoingBranches(jumpStart) do
         //alle Kanten und Knoten des betrachteten Zweiges
         elements := allElementsOnBranch(b);
         edges := ∅;
         if (elements contains exactly (2 edges and 1 event)) then ❷
            jumpEnd := correspondingJoin(jumpStart); //potent. Sprungziel
            if ((jumpEnd, jumpStart) ∉ getAllLoops(model)) then ❸
               forall e ∈ elements with e_type=edge do ❹
                  edges := edges ∪ e;
                  jumps := jumps ∪ (jumpStart, jumpEnd, edges);
               done
            endif
         endif
      done
   done
   return jumps;
end
```

Erklärungen:

❶ Gibt alle XOR-Splits des Prozessmodells zurück. Diese sind die potentiellen Sprungstellen.

❷ Nur wenn ein Zweig genau zwei Kanten und ein Ereignis enthält, kann er potentiell einen Sprung darstellen. Das liegt an den eEPKs, ansonsten wäre es nur eine Kante und kein Ereignis.

❸ Sprungstelle und Sprungziel dürfen nicht in umgekehrter Reihenfolge unter den Schleifen vorkommen. Das könnte passieren, da ein Rücksprung in einer Schleife auch als Sprung identifiziert wird.

❹ Die Kanten, der bereits identifizierten Elemente auf dem betrachteten Zweig, sind die Sprungkanten.

8.2.3 Transformation eines kompletten Prozesses

Die Transformation eines kompletten eEPK-Prozesses ist nicht einfach die Summe der Abbildungen aller Elemente und Konstrukte, da Konstrukte aus Elementen bestehen und es bei diesen somit zu Überschneidungen der Abbildungen kommen würde.

Die Konstrukte dürfen zunächst lediglich identifiziert werden. Danach wird bei der Abbildung der einzelnen Elemente überprüft, ob diese zu einem bestimmten Konstrukt (z.B. Verzweigung, Sprung oder Schleife) gehören und somit eine spezielle Abbildung benötigen. Dabei muss schon bei der Identifikation der Konstrukte darauf geachtet werden, dass Elemente nicht in mehreren Konstrukten vorkommen. So einen Fall gab es im Algorithmus *getAllJumps(model)* in Abschnitt 8.2.2.3, wo geprüft werden musste, ob eine gefundene Sprungkante die Rücksprungkante einer bereits gefundenen Schleife ist. Da die Elemente immer zu den speziellsten Konstrukten gehören sollten, ist es am einfachsten, wenn als erstes die speziellen und dann die allgemeineren Konstrukte identifiziert werden.

Eine Ausnahme bei den Konstrukten stellen die Verzweigungen dar. Diese dürfen bzw. müssen sogar auch bei spezielleren Konstrukten wie z.B. Schleifen vorkommen, da sie Bestandteil dieser sind. Zudem soll bei der Abbildung der Verzweigungen lediglich eine übersichtlichere Darstellung erreicht werden, wodurch die Abbildung von Elementen speziellerer Konstrukte nicht geändert wird.

Der folgende Algorithmus verdeutlicht das Vorgehen:

```
mapProcessModel(model)
 input:
   model: Prozessmodell, das auf das kanonische Modell abgebildet werden soll.
 output:
   mappedModel: Prozessmodell im kanonischen Modell.
 begin:
   elements := allElements(model); //alle Knoten und Kanten des Prozessmodells
   mappedElements := ∅; ❶

   //Identifikation aller Konstrukte
   loops := getAllLoops(model);
   jumps := getAllJumps(model);
   splits := getAllSplits(model);
   … //hier weitere Konstrukte identifizieren

   forall e ∈ elements do
      //Abbildung spezieller Elemente
      if (e ∈ loops) then newElement := mapLoopElement(e); ❷
         else if (e ∈ jumps) then newElement := mapJumpElement(e); ❸
         … //hier können weitere Abbildungen eingefügt werden

         mappedElements := mappedElements ∪ {(e, newElement)}; ❹
      endif

      //Abbildung von Verzweigungen
      if (e ∈ splits) then ❺
         if (e ∈ mappedElements) then

            //Element wird nicht gemappt, sondern lediglich transformiert,
            //da es sich um einen bereits gemappten Split handelt
            newElement := transformMappedSplitElement(newElement);
         else newElement := mapSplitElement(e);
         endif
         mappedElements := mappedElements ∪ {(e, newElement)};
      … //hier können weitere Abbildungen eingefügt werden (analog zu
        //Verzweigungen)

      else ❻
         newElement := mapElement(e);
         mappedElements := mappedElements ∪ {(e, newElement)};
      endif
   done

   mappedModel := buildModel(mappedElements); ❼

   return mappedModel;
 end
```

Erklärungen:

❶ Alle abgebildeten ARIS-Elemente kommen zunächst, zusammen mit den dazugehörigen Elementen des kanonischen Modells, in diese Menge. Aus dieser wird bei ❼ das neue Modell erzeugt.

❷ Falls das betrachtete Element zu einer Schleife gehört, dann wird es als Schleifenelement abgebildet und bei ❹ zur Menge der abgebildeten Elemente hinzugefügt. Die Überprüfung, ob das Element Schleifenanfang, -ende oder die Schleifenkante ist, wird in der *mapLoopElement(element)* Methode durchgeführt und danach wird das passende Element des kanonischen Modells zurückgegeben.

❸ Analog zu ❷.

❹ Das neue Element wird zusammen mit dem abzubildenden Element zur Menge der abgebildeten Elemente hinzugefügt.

❺ Falls das betrachtete Element ein Splitknoten ist, muss überprüft werden, ob dieser Knoten zu einem der bereits abgebildeten Elemente gehört. Falls dies der Fall ist, so wird das bereits auf das kanonische Modell abgebildete Element transformiert. Bei der Transformation muss lediglich eine Anpassung des bereits vorhandenen Strukturknotens und seiner Kanten vorgenommen werden. Ist das Element noch nicht abgebildet, so kann eine normale Abbildung einer ARIS-Verzweigung auf eine Verzweigung im kanonischen Modell stattfinden.

❻ Falls das betrachtete Element kein spezielles Element ist, dann wird es anhand festgelegter Regeln auf das passende Element im kanonischen Modell abgebildet.

❼ Aus den Elementen, die auf das kanonische Modell abgebildet sind, wird das neue Prozessmodell erzeugt.

Geht man nach diesem Algorithmus vor und bildet die Prozessmodelle aus Abbildung 8.1 und Abbildung 8.2 ab, so kommt man zu den Modellen in Abbildung 8.10 und Abbildung 8.11. Wie man bei der Wertschöpfungskette sehen kann, gibt es hier kaum Abweichungen vom ARIS-Modell Statt verfeinerter Funktionen werden hier lediglich Aktivitäten vom Typ Subprozess verwendet, ansonsten ist alles gleich.

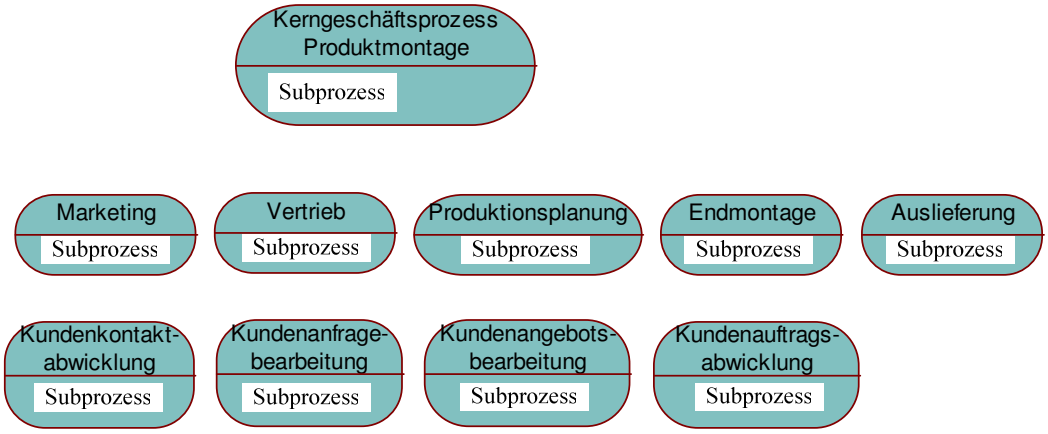


Abbildung 8.10 Wertschöpfungskette im kanonischen Modell

Die Abbildung der eEPK dagegen weicht stark vom Original ab. Der größte Unterschied ist, dass die Ereignisse fehlen. Die Schleifenbedingung, die durch Ereignisse dargestellt wurde, steht nun auf den Kanten. Zudem sind Schleifenanfangs- und -endknoten explizit gekennzeichnet. Aber auch die Schleifenkanten, sowie die anderen Kanten sind typspezifisch abgebildet, damit sie schon graphisch eindeutig erkennbar sind.

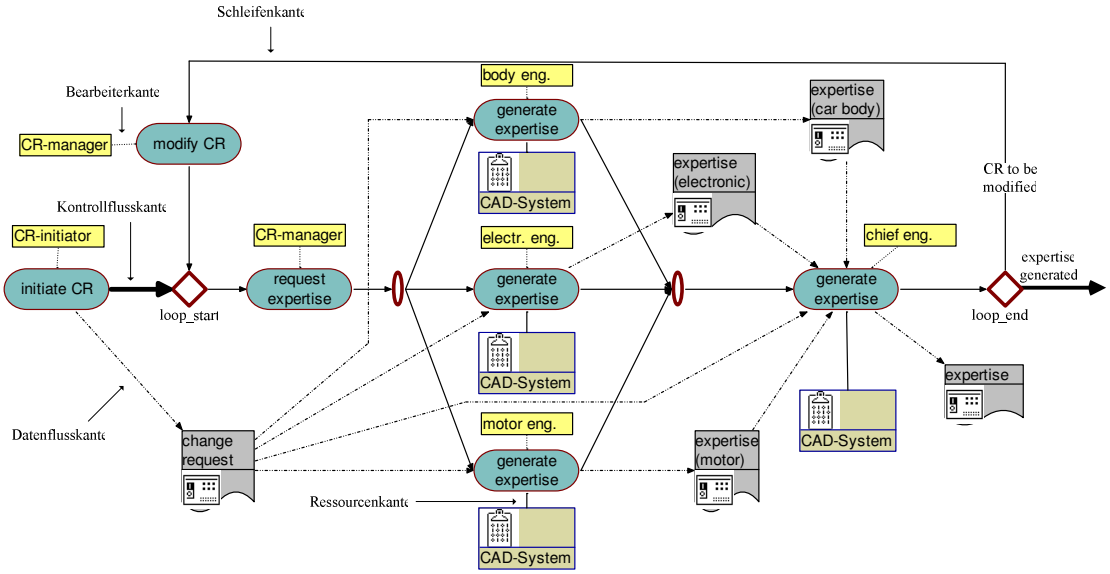


Abbildung 8.11 Erster Teil des ChangeManagements im kanonischen Modell

8.2.4 Grenzen automatischer Abbildungen

Die Abbildungen, die in den vorherigen Abschnitten vorgestellt wurden, können alle automatisch durch den Transformator vorgenommen werden. Doch es gibt auch viele Konstrukte, die nicht automatisch auf das kanonische Modell abgebildet werden kön-

nen. Diese Abbildungen muss man manuell mit dem Modellierungswerkzeug durchführen. Im den folgenden Abschnitten werden dafür ein paar Beispiele aufgezeigt.

8.2.4.1 Eingebundene Objekte

ARIS bietet die Möglichkeit unterschiedlichste Objekte in den modellierten Geschäftsprozess einzubinden. Das können Word-Dokumente, PDF-Dateien, aber auch Video-Sequenzen sein.

Die Objekte werden als OLE-Objekte (Objekt Linking and Embedding) eingebunden. Sie können als Verweis oder komplett als Objekt eingebunden werden. Im ersten Fall gibt es bei der Abbildung auf das kanonische Modell kein Problem.

Im zweiten Fall ist das gespeicherte Objekt nur noch ein BLOB (Binary Large Object) und somit kein strukturiertes Objekt. Solche Objekte können nicht automatisch auf das kanonische Metamodell abgebildet werden. Stattdessen muss der Benutzer das eingebundene Objekt identifizieren und dieses auf ein Datenelement abbilden. Dem *Zugriffslink*-Attribut des neuen Datenelements wird die Adresse des Originalobjekts als Wert zugewiesen.

8.2.4.2 Durch Annotationen definierte Konstrukte

Trotz einer Vielzahl von Elementen und Konstrukten gibt es auch Sachverhalte, die in ARIS nicht modelliert werden können. So lassen sich beispielsweise keine Funktionen modellieren, von denen es zur Laufzeit mehrere Instanzen geben soll.

In solchen Fällen gibt es projektspezifische Konventionen für den Umgang mit dem Problem. Eine Möglichkeit ist, in Annotationen zu beschrieben, was nicht modelliert werden konnte bzw. was optimalerweise hätte modelliert werden sollen. Diese Annotationen können aber nicht auf das passende Konstrukt des kanonischen Modells abgebildet werden, da sie normalerweise in einer maschinell nicht auswertbaren Form vorliegen. Somit müssen die Annotation und das Konstrukt, das von ihr ergänzt wird, manuell auf das richtige Konstrukt im kanonischen Modell abgebildet werden.

8.3 Zusammenfassung

In diesem Kapitel wurde die Prozesstransformation am Beispiel von ARIS-Modellen vorgestellt. Dazu wurde die Abbildung von Wertschöpfungskettendiagrammen und eEPKs auf ein kanonisches Metamodell demonstriert. Es hat sich gezeigt, dass Wertschöpfungskettendiagramme einfach abzubilden sind, da sie nur aus einem Elementtyp bestehen, während sich die Abbildung der eEPKs viel aufwändiger gestaltet, selbst bei Beschränkung auf die wesentlichen Elemente und Konstrukte. Entscheidend für die möglichen Abbildungen ist auch das kanonische Modell. Hier wurde ein semantisch reiches Modell gewählt, so dass neben direkten Abbildungen auch semantische Aufwer-

tungen von Elementen möglich waren, wodurch die neuen Elemente explizite Informationen über ihren Kontext im Modell besitzen.

Neben den durchgeführten Transformationen wurde auch gezeigt, dass die automatische Transformation bei manchen Konstrukten an ihre Grenzen stößt. So mussten die Schleifen auf strukturierte Schleifen eingeschränkt werden. Als BLOB eingebundene Objekte und Annotationen können höchstens manuell transformiert werden.

Da die Betrachtungen nur für die wichtigen Elemente und Konstrukte gemacht wurden, wurde der Algorithmus für die Transformation des gesamten Prozessmodells so gewählt, dass die Transformation ohne Probleme um weitere Elemente und Konstrukte erweitert werden kann. Dazu sind lediglich an den entsprechenden Stellen neue Methoden einzufügen, analog zu den bisherigen Methoden.

Um zu sehen, welche Ansätze in anderen Arbeiten gewählt wurden, werden im folgenden Kapitel verwandte Arbeiten vorgestellt. Neben der Transformation werden diese auch auf die anderen Aspekte dieser Arbeit untersucht und bewertet.

9 Related Work

In diesem Kapitel werden Arbeiten mit ähnlichen Zielen und Problemen vorgestellt. Dabei werden die bei der Datenintegration auftretenden Probleme nicht berücksichtigt, da hierfür bereits in Kapitel 3 Systeme zur Lösung vorgestellt wurden.

9.1 ArchiMate

Das ArchiMate-Projekt [LDL04] hat das Ziel, den Entwurfsprozess einer Unternehmensarchitektur möglichst gut zu unterstützen. Mit Unterstützung ist nicht nur die Visualisierung gemeint, sondern auch Änderungen in der integrierten Arbeitsumgebung. Die Unternehmensarchitektur umfasst dabei neben der statischen Infrastruktur auch die Geschäftsprozesse und Organisationsstruktur des Unternehmens. Somit ist es klar, dass dabei unterschiedliche Metamodelle und Systeme integriert werden sollen, die zunächst voneinander unabhängig sein können und für unterschiedliche Zwecke eingesetzt werden.

Bei der Integration wird zwischen zwei Arten unterschieden: der technischen und der konzeptuellen Integration.

Zur technischen Integration gehören die Daten-, Kontroll- und die visuelle Integration. Es sollen also neben der gemeinsamen Nutzung von Daten, Diagrammen und Modellen, sowohl die Kommunikation und Steuerung zwischen einer Menge von Werkzeugen, als auch die Benutzerinteraktion mit diesen unterstützt werden. Dafür werden, wie in der vorliegenden Arbeit, bei der Beschaffung der Log-Daten Wrapper eingesetzt, die die Werkzeuge kapseln und die benötigten Daten und Methoden in vordefinierter Art und Weise über eine gemeinsame Schnittstelle offen legen. Für Datenaustausch und -speicherung wird ein XML-Format (XMI) benutzt, wodurch gewisse Metadaten automatisch mitgespeichert werden und die Daten somit einfacher zu interpretieren sind. Details zur Lösung werden allerdings nicht offen gelegt.

Die konzeptuelle Integration soll unterschiedliche Metamodelle und Beschreibungssprachen integrieren. Dafür werden hier lediglich die direkte und indirekte Abbildung als Möglichkeit genannt, wobei auch hier die Indirekte umgesetzt wird. Das Zwischenmodell, das benutzt wird, heißt *Archimate Modelling Language* und wurde eigens zu diesem Zweck entwickelt. Bei der Abbildung der Modelle auf das ArchiMate-Modell werden die unterschiedlichen Konstrukte verallgemeinert. Das bedeutet, dass in der integrierten Arbeitsumgebung die Konstrukte sehr allgemein angezeigt werden, während sie in Wirklichkeit viel spezieller sein können. Falls man die Details sehen will, kann über die integrierte Arbeitsumgebung das Werkzeug des jeweiligen Konstrukts zur Vi-

sualisierung aufgerufen werden. Falls Konstrukte nicht abgebildet werden können, werden sie einfach weggelassen.

Die Architektur besteht aus drei Schichten: *Workbench*-, *Integration*- und *Tool*-Schicht. Die *Integrations*-Schicht befindet sich in der Mitte und verbindet die beiden anderen Schichten, indem sie der oben liegenden *Workbench*-Schicht, eine Schemabeschreibung der *Tool*-Schicht bereitstellt. Die Beschreibung gibt an, wie Archimate-Konstrukte nach unten hin spezialisiert oder nach oben hin verallgemeinert werden können. Die Kontrolle hat dabei immer die *Workbench*-Schicht, in der sich die integrierte Arbeitsumgebung befindet. Diese bestimmt, wann ein Modell transformiert oder im eigenen Werkzeug aus der *Tool*-Schicht geöffnet werden soll.

Ein weiterer Fokus des Projektes sind *Viewpoints*. *Viewpoints* sind Vorlagen für die Erstellung von Views und beschreiben welche Modellierungskonstrukte erlaubt sind, wie sie dargestellt werden und welche Verbindungen sie haben dürfen. Mit Hilfe der Viewpoints kann die Arbeitsumgebung auf bestimmte Benutzer eingestellt werden [LDL04]. Dieses Konzept könnte bei der Prozessintegration oder in der Visualisierungskomponente benutzt werden.

Die ArchiMate-Lösungen bieten einige gute Ansätze, allerdings auch einige, die im Kontext der vorliegenden Arbeit nicht akzeptabel sind. So werden Konstrukte in der integrierten Arbeitsumgebung verallgemeinert, statt semantisch aufgewertet. Oder sie werden einfach weggelassen, falls sie nicht ohne Probleme abgebildet werden können, anstatt, dass das Zielmodell erweitert wird. Trotz dieser Mängel wird bei ArchiMate die Transformation als Schlüssel zum Erfolg gesehen und es wird zugegeben, dass in diesem Bereich weiter geforscht werden muss.

9.2 IMPROVE

IMPROVE [Jae03] ist ein Projekt, das das Management von unternehmensübergreifenden Entwicklungsprozessen unterstützen soll. Im Mittelpunkt der Betrachtung steht dabei die Delegation von Aufgaben an Geschäftspartner, die auf die jeweiligen Aufgaben spezialisiert sind. Durch die Delegation über Systemgrenzen hinweg entstehen ähnliche Probleme wie die, die in dieser Arbeit diskutiert wurden.

Das Prozessmanagement-System AHEAD dient als Basis des ganzen Projektes und somit auch aller Betrachtungen, die auf einer sehr hohen konzeptuellen Ebene stattfinden.

Das Grundprinzip ist, dass der globale Prozess immer in AHEAD abläuft. Die delegierten Teilprozesse laufen verteilt auf potentiell unterschiedlichen Systemen ab. Dennoch ist die Steuerung im Gegensatz zum ArchiMate-Projekt dezentral, indem lokale Kopien von bestimmten Elementen gemacht werden. Somit können lokale Ereignisse auch im

globalen Prozess Aktivitäten auslösen und der Status eines Teilprozesses ist zudem auch bei ausgeschaltetem Partnersystem jederzeit bekannt.

Zur Demonstration einer Prozessabbildung wurde als zweites Prozessmanagement-System COSA gewählt. COSA erhält die von AHEAD in eine XML-Datei exportierten und danach transformierten Daten ebenfalls in einer XML-Datei. Welche Probleme bei heterogenen Datenquellen und Daten auftreten und wie diese behandelt werden können, wird nicht erwähnt.

Die Prozesstransformation wird mit XSLT (*Extensible Stylesheet Language Transformations*) [W3C99] durchgeführt. Dazu werden Konsistenzbedingungen für die Semantik-erhaltende Transformation und die komplette Ausführungslogik in PROGRES, einer dafür geeigneten Sprache, beschrieben. Im behandelten Fall sind das Transformationen von Aktivitätennetzen auf Petrinetze (Bedingung-Ereignis-Netze).

Durch dieses Vorgehen werden die in Kapitel 4 identifizierten Probleme formal erfasst, allerdings nicht in vollem Umfang, sondern nur am Beispiel der Abbildung von A-HEAD auf COSA. Laut Verfasser ist eine Transformation in umgekehrte Richtung oder zwischen zu unterschiedlichen Quell- und Zielmodellen nicht möglich. Zudem ist zu beachten, dass die Beschreibung der erwähnten Regel in PROGRES so aufwendig ist, dass sie selbst in dem Buch, das im Rahmen des Projektes und der Dissertation entstand, keinen Platz gefunden hat.

9.3 Process Mining

Auch im Bereich des Process Minings gibt es ähnliche Probleme wie in der vorliegenden Arbeit. Wie in den Grundlagen in Abschnitt 2.2.1.3 gezeigt, wird Process Mining eingesetzt, um aus Log-Daten von Systemen Prozessmodelle zu erstellen. Wendet man Process Mining auf unterschiedliche Systeme an, so müssen die u.U. heterogenen Log-Daten auf ein standardisiertes Format abgebildet werden, damit sie von einem Process-Mining-Werkzeug gelesen werden können. So ein Werkzeug ist beispielsweise *ProM*. *ProM* benötigt die Log-Daten der Systeme im MXML-Format, einem speziellem XML-Format.

Um erweiterbar für neue Systeme und Modelle zu bleiben, werden in *ProM* Plug-Ins benutzt. Die Import-Plug-Ins lesen die Log-Daten und transformieren sie in das MXML-Metamodell. Wie die Daten gelesen werden, falls sie nicht in einer XML-Datei vorliegen, wird nicht behandelt.

Für die Transformation werden ähnlich wie bei dem gezeigten Beispiel aus Kapitel 8 Daten-Elementen des Quellmodells Elemente des MXML-Metamodells zugeordnet. Die Zuordnung wird dabei als Instanziierung gesehen. So wird das MXML-Metamodell vom Datenmodell des Quellsystems instanziiert, das wiederum von den Log-Daten in-

stanziiert wird. Somit instanziieren die Log-Daten, über einen Zwischenschritt, auch das MXML-Metamodell und werden so auf das instanziierte Element abgebildet.

Nach der Transformation wird das Process Mining mit weiteren Plug-Ins durchgeführt, auf dessen Ergebnissen Analysen oder Konvertierungen auf andere grafische Modelle vorgenommen werden [DoAa05].

9.4 Weitere Arbeiten

Andere Arbeiten auf diesem Gebiet sind von den Zielen der vorliegenden Arbeit weiter entfernt als die bisher vorgestellten. Einige von diesen, wie beispielsweise [Gru05], sind auf einer hohen semantischen Ebene, auf der sie die Daten vollständig vernachlässigen und nur mit Hilfe von Ontologien die Abbildbarkeit von Metamodellen unterschiedlicher Ausdrucksmächtigkeit aufeinander beweisen. Ob eine praktische Umsetzung möglich ist und welcher Aufwand dazu nötig wäre, wird nicht betrachtet.

Zudem gibt es noch Arbeiten, die entweder zu spezifisch sind oder Probleme übergehen, anstatt sie zu lösen. Zu den ersteren gehören [MeZi05, Sta04] bei dem es um die Abbildung von BPEL4WS-Prozessen auf EPKs geht. Dabei liegt der Fokus so stark auf BPEL4WS, dass sich die Konzepte nicht vernünftig auf andere Beschreibungssprachen oder Metamodelle übertragen lassen. Im zweiten Fall, in [ZiMe05], werden EPK-Modelle auf BPEL4WS abgebildet. Das müsste bedeuten, dass EPKs den Ausgangspunkt der Transformation bilden. Stattdessen geht man von BPEL4WS-Elementen aus, die zusammen ein EPK-Element bilden. Das Ergebnis ist, dass das jeweilige EPK-Element auf BPEL4WS abgebildet werden kann, was bei einem derartigen Vorgehen bereits vorher klar ist. Dabei wird nämlich das eigentliche Problem übersehen, und zwar was man mit Elementen des Quellmodells macht, die im Zielmodell nicht vorkommen und eventuell auch aus anderen Elementen nicht zusammengesetzt werden können.

Bei CrossFlow [LuHo99] und [KGV00] werden Workflow-Management-Systeme mit Hilfe von Kontrakten (Verträgen), ähnlich wie bei Web Services, miteinander verbunden. Diese Art der Integration erlaubt einen hohen Grad an Entkoppelung. Dabei muss vor jeder Kommunikation ein Kontrakt für die Kommunikation und für die Ausführung des übertragenen Prozesses ausgehandelt werden. Da in unserem Fall eine Entkoppelung der Systeme nicht nötig ist, hätte so eine Vorgehensweise lediglich unnötige Kommunikationslast zur Folge. Zudem beschränken sich die Arbeiten auf WfMSe, was in unserem Fall auch nicht gegeben ist.

Organisationen, wie beispielsweise die WfMC (Workflow Management Coalition) schlagen standardisierte Schnittstellen (Schnittstelle 4) [WfMC99b] für die Kommunikation zwischen den Systemen vor, was die Kommunikation enorm erleichtern würde. Solche Standards können allerdings in der vorliegenden Arbeit nicht vorausgesetzt wer-

den. Selbst wenn die neueren Systeme die Standards umsetzen würden, eine Kommunikation mit den Legacy-Systemen wäre nicht möglich.

9.5 Zusammenfassung

Zusammenfassend kann man sagen, dass verwandte Arbeiten nur bedingt brauchbare Ergebnisse liefern. Das ArchiMate-Projekt hat zwar einige gute Ansätze in der Architektur und Vorgehensweise, löst aber das Problem der Transformation nicht zufriedenstellend. Zudem werden die Modelle bei der Visualisierung verallgemeinert und für Details muss auf das Original-Modell zugegriffen werden. Bei IMPROVE wird zwar die Transformation mit Hilfe formaler Beschreibungen von Ablauflogik und Konsistenzbedingungen gelöst, allerdings nur auf einer hohen konzeptionellen Ebene. Dadurch werden viele Probleme, vor allem auf Datenebene, übergangen. Für die semantische Integration ist es dennoch ein guter Ansatz, der allerdings noch viel zu aufwendig ist und noch weiter erforscht werden muss. Weiter erforscht werden muss auch der Ansatz vom Process Mining, in [DoAa05], der Log-Daten über Instanziierungsbeziehungen integriert. Allerdings wird auch hier das Problem der Transformation stark vereinfacht, indem als Beispiel zwei sehr ähnliche Metamodelle benutzt werden. Außerdem werden nur wenige Log-Daten benötigt, was das Problem zusätzlich vereinfacht.

Weitere Arbeiten auf diesem Gebiet bieten nur wenig verwertbare Informationen.

10 Zusammenfassung und Ausblick

Durch veränderte Strukturen in und zwischen Unternehmen, sowie durch die Komplexität der unternehmerischen Prozesse, werden mittlerweile viele Prozesse unternehmens-, abteilungs-, aber vor allem systemübergreifend ausgeführt. Die verteilte Ausführung auf zumeist heterogenen Systemen macht eine einheitliche Visualisierung unentbehrlich, um den Überblick nicht zu verlieren, Optimierungen und bei Zwischenfällen Änderungen vornehmen zu können. Dazu müssen die Prozessfragmente der einzelnen Systeme auf ein einheitliches Metamodell abgebildet und zu einem Gesamtprozess integriert werden. Da die Informationen zu den Prozessfragmenten je nach System in unterschiedlicher Form oder in unterschiedlichem Umfang vorliegen, ergeben sich bei der Abbildung jede Menge Probleme.

Ziel dieser Arbeit ist es, die auftretenden Probleme zu identifizieren, zu kategorisieren und Ansätze zu deren Lösung aufzuzeigen. Darauf basierend wird eine Architektur für eine Komponente entwickelt, die die Lösungen umsetzt und an einem konkreten Beispiel die Abbildung eines Prozessmodells auf ein vorgegebenes Metamodell demonstriert.

Die Kategorisierung der Probleme leitet sich von den für das Mapping notwendigen Schritten ab. Das sind Datenintegration, Prozesstransformation, Prozessintegration und Instanzdatenintegration.

Der erste Schritt bei der Abbildung der Prozessfragmente ist die Integration der Daten der Quellsysteme, wobei Prozessinformationen noch keine Rolle spielen. Die Daten werden lediglich auf ein einheitliches Format abgebildet. Dabei kann es u.a. aufgrund unterschiedlicher Namensgebung, Struktur und Semantik der Daten zu Konflikten kommen. Zudem kann es Ambiguitätsprobleme bei Instanzen und Probleme wegen unterschiedlichen Datenmodellen bei Datenbanken geben. Diese und ähnliche Probleme sind in der Informatik bereits bekannt und können, je nach Quellsystemen, mit Hilfe von föderierten Datenbanken, Data-Warehouse- oder EAI-Systemen gelöst werden.

Nachdem die Daten in einem einheitlichen Format vorliegen, werden bei der Prozesstransformation die einzelnen Prozessfragmente auf ein einheitliches Metamodell abgebildet. Für das einheitliche Metamodell gibt es verschiedene Möglichkeiten, von denen sich ein erweiterbares kanonisches Modell als die Beste herausgestellt hat. Für die Abbildung werden Modelldaten der Prozesse benötigt, die nicht von allen Systemen bereitgestellt werden. In solchen Fällen müssen die benötigten Daten von Wrappern oder von Process Mining-Werkzeugen beschafft werden. Optimalerweise sind die beschafften Daten bereits in der richtigen Form, basierend auf dem kanonischen Metamodell, wodurch Transformationen wie bei den Modelldaten der Quellsysteme vermieden wer-

den können. Dabei kann es nämlich zu Namens- und Modellkonflikten, sowohl im Kontroll- und Datenfluss, als auch im Organisationsmodell kommen. Da die Konflikte von den zwei Metamodellen abhängig sind, die aufeinander abgebildet werden sollen, ist eine allgemeine Lösung nicht möglich. Stattdessen muss für jedes Paar von Quell- und Zielmodell ein umfassendes Regelwerk aufgestellt werden, anhand dessen die Transformationen vorgenommen und die Quellmodelle auf das kanonische Metamodell abgebildet werden.

Sind die Prozessfragmente alle auf das kanonische Metamodell abgebildet, dann können sie im nächsten Schritt, nämlich bei der Prozessintegration, zu einem Gesamtprozess zusammengefügt werden. Dabei müssen zunächst Beziehungen zwischen diesen identifiziert und Inkonsistenzen, die durch unabhängige Modellierung der Subprozesse entstanden sind, beseitigt werden. Zudem müssen unterschiedliche Grade der Zusammenarbeit zwischen den Unternehmen berücksichtigt werden, wodurch u.U. Teile des Prozesses ausgeblendet werden müssen. Das kann entweder mit Hilfe von Sichten gemacht werden oder indem man die Daten von Prozessteilen, die nicht angezeigt werden sollen, gar nicht integriert. Ähnliche Fragen tauchen auch bei überlappenden Aktivitäten und mehrstufigen hierarchischen Prozessen auf, die sich auf die gleiche Art lösen lassen. Ein gegensätzliches Problem gibt es bei versteckten Aktivitäten. Diese müssen nicht ausgeblendet, sondern zunächst aufgefunden werden, weil sie nur in den Köpfen der Bearbeiter, nicht aber in den Systemen existieren. Dafür gibt es mehrere Lösungen, die allerdings sehr zeitaufwendig sind und von Systemen kaum unterstützt werden können. Nach Lösung all dieser Probleme und Konflikte, ist das statische Prozessmodell fertig.

Nun fehlt noch die Instanzdatenintegration, bei der das neu erzeugte Prozessmodell mit dynamischen Daten gefüllt wird. Auch dabei müssen zunächst die Heterogenitäten auf der Datenebene beseitigt werden, bevor zu Prozesstransformationen analoge Transformationen vorgenommen werden. Zusätzlich zu diesen Transformationen müssen weitere Anpassungen vorgenommen werden, da die Instanzdaten nun zu einem geänderten Prozessmodell passen müssen. Somit müssen die Änderungen von der Prozesstransformation und -integration auf die Laufzeitdaten übertragen und Konflikte, die durch Namens-, Struktur- und semantische Änderungen entstanden sind, gleichzeitig beseitigt werden. Diese lassen sich mit Hilfe von Namensumsetzungstabellen, Zusammenfassung bzw. Aufteilung von Daten oder im Falle der semantischen Änderungen einfach durch analoge Transformationen wie bei Modelldaten lösen. Ein weiteres Problem ist die Zuordnung der Instanzen der einzelnen Systeme zu den zugehörigen Instanzen anderer Systeme. Da die lokalen Schlüssel global nicht eindeutig sind, werden Applikationsdaten zur Korrelation der Instanzdaten benutzt.

Die Architektur zur Lösung dieser Probleme besteht aus einer Komponente, die das statische Prozessmodell erzeugt (Buildtime-Komponente) und aus einer, die für die Integration der Laufzeitdaten zuständig ist (Runtime-Komponente). Beide benutzen exter-

ne Regeln für die Änderungen, die an den Daten vorzunehmen sind. Die Buildtime-Komponente speichert zudem alle durchgeführten Änderungen, damit die Runtime-Komponente auf diese zugreifen und die Instanzdaten auf den geänderten Prozess anpassen kann.

Das Beispiel am Ende der Arbeit zeigt, dass sich beim Mapping von Prozessmodellen, je nach Quell- und Zielmodell, ganz spezielle Aspekte und Fragen ergeben können und dass somit der Aufwand enorm variieren kann. So können Wertschöpfungskettendiagramme fast direkt auf das benutzte kanonische Modell abgebildet werden, während bei eEPKs oft Anpassungen notwendig sind oder spezielle Konstrukte gar nicht abgebildet werden können. Bei einigen der Anpassungen handelt es sich um Optimierungen, wodurch das Prozessmodell im kanonischen Modell semantisch angereichert wird oder überflüssige Elemente weggelassen werden.

Sowohl am Mapping-Beispiel als auch durch die Probleme bei der Datentransformation wird deutlich, dass es bei der Transformation keine allgemeingültige Lösung gibt. Es muss für jedes Paar von Quell- und Zielmodellen ein umfassendes Regelwerk aufgestellt werden, das die Transformationen vorgibt. Das zeigen auch verwandte Arbeiten, die sich mit dem Problem der Transformation näher beschäftigen. Auch wenn diese noch sehr theoretisch und aufwendig sind, so bieten sie teilweise doch gute Ansätze, die auf Regelwerken und Ontologien aufbauen. Diese sollten weiter erforscht und auf ihre praktische Realisierbarkeit geprüft werden.

Auch für die Prozessintegration sind Ansätze denkbar, die auf Regelwerken basieren. Da aber die Aufstellung eines kompletten Regelwerkes enorm aufwendig ist, sollte anfänglich eine halb-automatische Lösung in Betracht gezogen werden, indem ein Teil der Regeln von Anfang an aufgestellt wird und die restlichen automatisch aus Benutzerinteraktionen hergeleitet werden. So wird das Regelwerk nach und nach erweitert und es werden immer weniger Benutzereingriffe benötigt, bis irgendwann eine beinahe automatische Transformation und Integration von Prozessmodellen erreicht ist. Einen vollautomatischen Ablauf wird es nicht geben können, weil Prozesse durch fehlende Standards immer Informationen enthalten können, die maschinell nicht verwertbar sind.

Literaturverzeichnis

[AaHo02] van der Aalst, W.M.P.; ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language, QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.

[AaWe04] van der Aalst, W.M.P.; Weijters, A.J.M.M.: Process Mining: A Research Agenda, Computers in Industry, 53(3):231-244, 2004, http://is.tm.tue.nl/research/processmining/papers/procmin-cii.pdf, zuletzt besucht am: 26.07.2005.

[AHKB03] van der Aalst, W.M.P.; ter Hofstede, A.H.M.; Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.

[Bae04] Bäuerle, T.: Exception Handling in the BPEL4WS Language, http://www.informatik.uni-ulm.de/dbis/01/lehre/ss04/hs/s_wfm/ausarbeitungen/Baeuerle_ExcpHandBPEL.zip, zuletzt besucht am 27.08.2005.

[BaGu01] Bauer, A.; Guenzel, H.: Data Warehouse Systeme – Architektur, Entwicklung, Anwendung, dpunkt Verlag, Heidelberg 2001.

[BIM+03] BEA Systems, IBM, Microsoft, SAP AG and Siebel, Systems Business Process Execution Language for Web Services version 1.1, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, zuletzt besucht am: 27.08.2005.

[Bob05] Bobrik, R.: Metamodell für die Prozessvisualisierung

[BRB05] R. Bobrik, M. Reichert, T. Bauer: Requirements for the Visualization of System-Spanning Business Processes. Proc. 16th Int'l Workshop on Database and Expert Systems Applications (DEXA 2005), pp. 948-954 Copenhagen, Denmark, August 2005

[Con97] Conrad, S.: Föderierte Datenbanksysteme – Konzepte der Datenintegration, Springer-Verlag, Heidelberg 1997.

[Dad96] Dadam, P.: Verteilte Datenbanken und Client/Server-Systeme - Grundlagen, Konzepte und Realisierungsformen, Springer-Verlag, Berlin 1996.

[DaSp02] Dadam, P; Specht, G.: Vorlesung – Datenbanksysteme, Eigenschaften und Architektur von DB-Systemen, Wintersemester 2002/03.

[DoAa05] van Dongen, B.F.; van der Aalst, W.M.P.: A Meta Model for Process Mining Data, Proceedings of the CAiSE'05 WORKSHOPS, volume 2, 2005.

[EMiT] Enhanced Mining Tool, http://is.tm.tue.nl/research/processmining/tools.htm#emit, zuletzt besucht am: 10.06.2005.

[Gru05] Grüninger, M.: Model-theoretic Approaches to Semantic Integration (Extended Abstract). Semantic Interoperability and Integration, 2005, http://drops.dagstuhl.de/opus/volltexte/2005/51/pdf/04391.SWM5.ExtAbstract.51.pdf, zuletzt besucht am 28.10.2005.

[HLGG00] Hoffner, Y.; Ludwig, H.; Gülcü, C.; Grefen, P.: Architecture for Cross-Organisational Business Processes. In Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000). pp. 2-11, Milpitas, CA, June 8 and 9, 2000.

[IBM04] IBM corp.: IBM Websphere MQ Workflow - Admistration Guide, Websphere MQWorkflow, V3.5, http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=SH12-6289-07, zuletzt besucht am: 26.07.2005.

[IDS04] IDS Scheer AG: ARIS Methode, ARIS 6 Collaborative Suite.

[Jae03] Jäger, D.: Unterstützung übergreifender Kooperation in komplexen Entwicklungsprozessen, Aachener Beiträge zur Informatik, Verlag Mainz, Aachen 2003.

[JMSW04] Jurisch, M.; Mihalca, T.; Sauter, P.; Waimer, M.: Praktikum ADEPT2 - Verwaltung von Prozessinstanzen in Workflow Systemen, Ulm, 2004.

[KGV00] Koetsier, M.J.; Grefen, P.W.P.J.; Vonk, J.: Contracts for Cross-Organizational Workflow Management, http://purl.org/utwente/fid/1290, zuletzt besucht am: 25.07.2005

[Klo04] Klotz, A.: View-Unterstützung in Prozess-Management-Systemen, Diplomarbeit, Ulm, 2004.

[LDL04] van Leeuwen, D.; ter Doest, H.; Lankhorst, M.M.: A Tool Integration Workbench for Enterprise Architecture - Integrating Heterogeneous Models and Tools, ICEIS (3) 2004: 470-478, 2004.

[LuHo99] Ludwig, H.; Hoffner, Y.: Contract-based Cross-Organisational Workflows - The CrossFlow Project. Cross-Organisational Workflow Management and Co-ordination, 1999, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-17/, zuletzt besucht am 27.10.2005.

[Men04] Mendling, J.: A Survey on Design Criteria for Interchange Formats, 2004, http://wi.wu-wien.ac.at/~mendling/publications/TR04-Interchange.pdf, zuletzt besucht am: 25.07.2005

[MeZi05] Mendling, J.; Ziemann, J.: EPK-Visualisierung von BPEL4WS Prozessdefinitionen. 7th Workshop Software-Reengineering (WSR 2005), Bad Honnef, Germany, 2005.

[Mue03] zur Muehlen, M.: Interface 5 - Issues and Directions, http://www.wfmc.org/standards/docs/Interface_5_Issues_and_Directions.pdf, zuletzt besucht am: 25.07.2005

[MWA04] de Medeiros, A.K.A.; Weijters, A.J.M.M.; van der Aalst, W.M.P.: Using Genetic Algorithms to Mine Process Models: Representation, Operators and Results, BETA Working Paper Series, WP 124, Eindhoven University of Technology, Eindhoven, 2004.

[Pry05] Pryss, R.: Enterprise Application Integration – Anforderungen, Ansätze und Technologien, Diplomarbeit, Ulm, 2005.

[Rei00] Reichert, M.: Dynamische Änderungen in Workflow-Management-Systemen, Dissertation, Universität Ulm, Ulm, 2000

[ReDa02] Reichert, M.; Dadam, P.: Vorlesung - Workflow-Management-Systeme, Implementierungs-, Architektur- und Verteilungsaspekte von WfMS, Wintersemester 2002/03.

[RHEA04a] Russell, N.; ter Hofstede, A.H.M.; Edmond, D.; van der Aalst, W.M.P.: Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[RHEA04b] Russell, N.; ter Hofstede, A.H.M.; Edmond, D.; van der Aalst, W.M.P.: Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

[Sta00] Staffware plc: Staffware Web Client User's Guide, Staffware 2000, 2000.

[Sta04] Stahl, C.: Transformation von BPEL4WS in Petrinetze, Diplomarbeit, Berlin, 2004, http://www.informatik.hu-

berlin.de/top/download/publications/stahl04.pdf, zuletzt besucht am: 29.10.2005.

[W3C99] XSL Transfomations, http://www.w3.org/TR/1999/REC-xslt-19991116, zuletzt besucht am: 29.10.2005.

[WfMC98] WfMC: Audit Data Specification (WfMC-TC-1015), http://www.wfmc.org/standards/docs/TC-1015_v11_1998.pdf, zuletzt besucht am 23.09.2005.

[WfMC99a] Workflow Management Coalition: Terminology & Glossary (WfMC-TC-1011), http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, zuletzt besucht am: 25.07.2005.

[WfMC99b] Workflow Management Coalition: Workflow-Standard – Interoperability Abstract Specification, http://www.wfmc.org/standards/docs/TC-1012_Nov_99.pdf, zuletzt besucht am: 08.11.2005.

[Wik05] http://de.wikipedia.org/wiki/Enterprise_Application_Integration, zuletzt besucht am: 09.11.2005.

[ZiMe05] Ziemann, J.; Mendling, J.: EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In Proceedings of the 7th International Conference "Modern Information Technology in the Innovation Processes of the Industrial Enterprises" (MITIP 2005).

Abkürzungsverzeichnis

API	Application Programming Interface
ARIS	Architektur integrierter Informationssysteme
BPEL	s. BPEL4WS
BPEL4WS	Business Process Execution Language for Web Services
BLOB	Binary Large Object
EPK	Ereignisgesteuerte Prozesskette
ETL	Extract, Transform, Load
DTD	Document Type Definition
ID	Identifikator
OLE	Object Linking and Embedding
u.a.	unter anderem
u.U.	unter Umständen
vDBMS	verteilte Datenbank Management Systeme
vs.	versus
WfMS(e)	Workflow Management System(e)
WSDL	Web Service Definition Language
XSLT	Extensible Stylesheet Language Transformation

Glossar

Aktivität: kleinste Arbeitseinheit im Prozess, die von einem WfMS Bearbeitern zugewiesen werden kann

API: (Application Programming Interface)
Schnittstelle für Anwendungsprogrammierung.

Applikation: Anwendung

Applikationsdaten: Anwendungsdaten

Arbeitsschritt: Bestandteile einer Aktivität

BLOB: (Binary Large Object)
binäre und unstrukturierte Objekte

BPEL4WS: (Business Process Execution Language for Web Services)
XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren einzelne Aktivitäten durch Webservices beschrieben sind

Buildtime: Modellierungszeit

Delegation: Übertragung von Aufgaben an unterstellte Organisationseinheiten

Change Management: Bearbeitung eines Änderungsantrages (Beispiel aus der Automobilindustrie gewählt)

DTD: (Document Type Definiton)
formale Definition für die Strukturierung von XML-Dokumenten (allg. SGML-Dokumenten)

EAI: (Enterprise Application Integration)
Integration von verschiedenen Applikationen auf unterschiedlichen Plattformen

Element: elementarer Bestandteil eines Prozessmodells (Knoten oder Kante)

Entity: s. Entität

Entität unterscheidbare, in der realen Welt eindeutig identifizierbare einzelne Objekte

Entscheidungspunkt: exklusive Verzweigung in Staffware

Gesamtprozess: Prozess, der aus allen Prozessen aller am systemübergreifenden Prozess beteiligten Systeme besteht.

Granularität: gibt an, wie fein ein Objekt beschrieben werden kann

heterogen: andersartig, ungleich

Input: Eingabe

Instanzdaten: s. Laufzeitdaten

Join: Stelle im Kontrollfluss mit mehreren eingehenden Kanten.

kanonisches Modell: ein Modell, dessen Elemente vorgegeben sind

Komponente: Bestandteil einer Architektur

Konnektor: Verbindung zwischen zwei oder mehreren Elementen

Konstrukt: s. Modellierungskonstrukt

Kontrakt: Vereinbarung, Vertrag

Laufzeitdaten: Daten, die während der Ausführung eines Prozesses entstehen

Legacy System: sind historisch gewachsene Applikationen im Bereich der Unternehmenssoftware, mit zumeist mangelnder Dokumentation und fehlenden oder proprietären Schnittstellen

Mapping: Abbildung eines Prozessmodells auf ein anderes. Der Ablauf beinhaltet Datenintegration, Prozesstransformation und -integration. Falls ein laufender Prozess „gemappt“ wird dann gehört auch die Instanzdatenintegration dazu.

Message Broker: ermöglicht durch den Transport von Mitteilungen die Kommunikation zwischen heterogenen Programmen

Middleware: Software mit Schnittstellencharakter; dient als Zwischenschicht zwischen Server- und Client-Komponenten eines verteilten Systems

Modellierungs-konstrukt: Bestandteil eines Prozessmodells; semantisch höherwertig als ein Element; besteht aus mehreren Elementen oder ist ein „spezielles Element“, das zur Steuerung des Prozesses eingesetzt wird

OLE: (Object Linking and Embedding)

von Microsoft entwickelter Standard, der einen einfachen Datenaustausch zwischen verschiedenen OLE-fähigen Applikationen ermöglicht

Output: Ausgabe

Overhead: Mehraufwand, der durch Hilfs- oder Verwaltungsdaten entsteht

Plug-In: Ergänzungs- oder Zusatzmodul für ein Softwareprogramm, mit dessen Hilfe die Einbindung in ein anderes Softwareprodukt erleichtert wird

Poll: Aktualisierungsvariante der Client-Server-Kommunikation, bei der der Client für die Aktualisierung der Daten zuständig ist

prorietär: nicht allgemein anerkannten Standards entsprechend

Prozessfragment: s. Teilprozess

Prozessmanagement-System: s. Workflow Management System

Push: Aktualisierungsvariante der Client-Server-Kommunikation, bei der der Server für die Aktualisierung der Daten zuständig ist

Quellmodell: Modell, das auf ein anderes Modell abgebildet wird

Reports: Berichte in ARIS

Runtime: Laufzeit

Soll-Prozess: Prozess, der optimalerweise ausgeführt werden sollte; meistens weicht der tatsächliche Ablauf vom gewünschten Ablauf ab

Split: Verzweigung

Sub-Komponente: stellt die Komponente einer Komponente dar

Subprozess: logisch abgeschlossener Teil eines Prozesses, der von einem übergeordneten Prozess aufgerufen wird

Supply Chain Management: beschäftigt sich mit der Verbesserung von Effektivität und Effizienz industrieller Wertschöpfungsketten

Teilprozess: auf einem System ausgeführter Teil des Gesamtprozesses

View: Sicht

Workflow Management System: dient der aktiven Steuerung arbeitsteiliger Prozesse; mit Hilfe von Workflow-Management-Systemen lassen sich Steuerungs- und Anwendungslogik voneinander trennen

Workflow-Engine: Kern eines Workflow Management Systems

XSLT: (Extensible Stylesheet Language Transformation)

	Programmiersprache zur Transformation von XML-Dokumenten
Zielmodell:	Modell, auf das ein anderes Modell abgebildet wird
Zwischenmodell:	Modell, auf das ein Quellmodell beim indirekten Mapping abgebildet wird. Das Zwischenmodell wiederum wird auf das Zielmodell abgebildet.

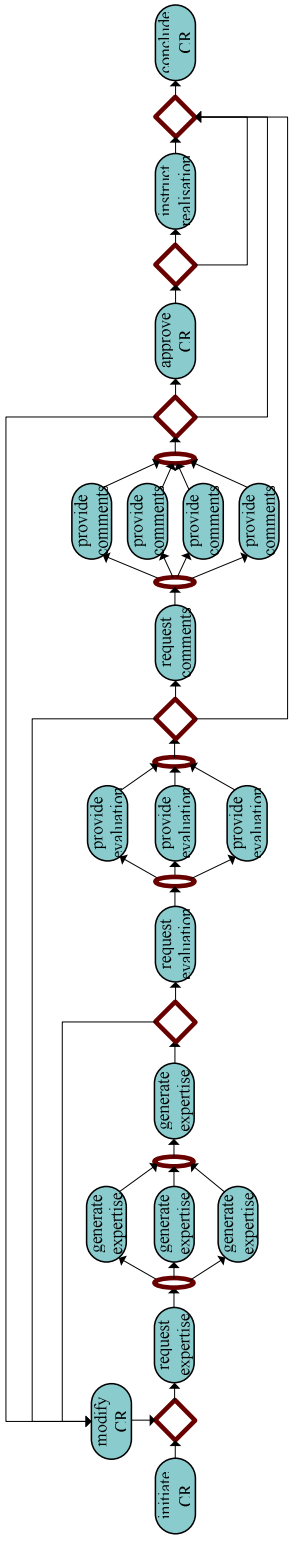
Die Quellen dieses Glossars sind die Quellen aus dem Literaturverzeichnis und die Online-Enyklopädie Wikipedia (www.wikipedia.de).

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

Anhang A (ChangeManagement-Prozess)



Anhang B (Workflow Patterns)

Basic Control Flow Patterns

Basic Control Flow Patterns sind elementare Kontrollkonstrukte, die für jeden Prozess essentiell sind und somit auch von jedem WfMS unterstützt werden. Im Folgenden findet man jeweils eine kurze Definition dieser Patterns. Abbildung 0.1 zeigt eine Zusammenstellung in einem Prozess [AHKB03].

Pattern 1 (Sequence) Eine Aktivität im Workflow wird aktiviert, nachdem eine andere Aktivität desselben Prozesses beendet wurde.

Pattern 2 (Parallel Split, AND-Split) Stelle im Workflow, an der ein einzelner Kontrollpfad in mehrere Kontrollpfade aufgespalten wird, die parallel ausgeführt werden können.

Pattern 3 (Synchronization, AND-Join) Stelle im Workflow, an der mehrere parallele Kontrollpfade zu einem Kontrollpfad zusammengeführt und somit synchronisiert werden. (Es wird vorausgesetzt, dass jeder der einmündenden Pfade nur einmal durchlaufen wird. Ansonsten siehe Patterns 13-15.)

Pattern 4 (Exclusive Choice, XOR-Split) Stelle im Workflow, an der aufgrund einer Entscheidung oder von Kontrolldaten, eine von mehreren Ausführungszweigen gewählt wird.

Pattern 5 (Simple Merge, XOR-Join) Stelle im Workflow, an der zwei oder mehrere alternative Ausführungszweige ohne Synchronisation zusammenlaufen. (Es wird vorausgesetzt, dass keine Zweige parallel ausgeführt werden. Ansonsten siehe Pattern 8 oder 9.)

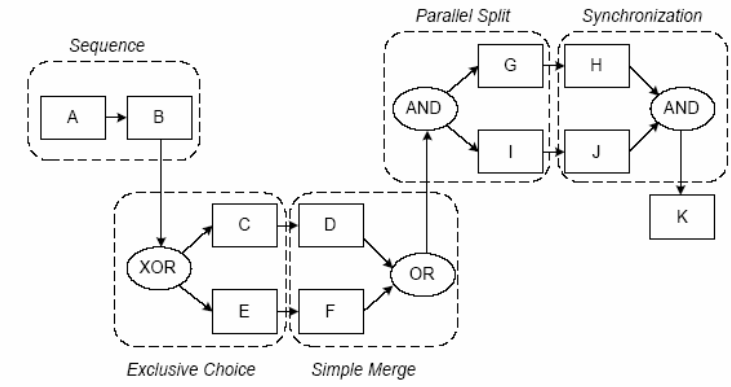


Abbildung 0.1 Basic Control Flow Patterns [AHKB03]

Advanced Branching and Synchronisation Patterns

Die folgenden Patterns ermöglichen komplexere Verzweigungs- und Synchronisationsstrukturen. Obwohl sie in realen Geschäftsprozessen häufig benötigt werden, werden sie durch einige WfMSe gar nicht oder nur indirekt unterstützt [AHKB03].

Pattern 6 (Multi Choice, OR-Split) Stelle im Workflow, an der aufgrund einer Entscheidung oder von Kontrolldaten, keiner, einer oder mehrere Ausführungszweige gewählt werden.

Dieses Pattern ist also eine Verallgemeinerung vom Parallel Split (Pattern 2) und vom Exclusive Choice (Pattern 4).

Pattern 7 (Synchronizing Merge) Stelle im Workflow, an der mehrere Bearbeitungszweige in einen Zweig zusammenlaufen. Wurden mehrere Zweige aktiviert, so ist eine Synchronisation der aktiven Ausführungszweige nötig. Bei nur einem aktiven Ausführungszweig entfällt die Synchronisation. Eine Voraussetzung des Synchronising Merge ist, dass ein bereits aktivierter Zweig erst nach Beendigung aller anderen Zweige wieder aktiviert werden darf. Dies ist für die Benutzung in Schleifen unabdingbar.

Dieses Pattern wird für die Zusammenführung nach einem Multi Choice benötigt.

Da zunächst nicht klar ist wie viele Zweige aktiviert wurden, ist es schwierig zu entscheiden, ob lediglich zusammengeführt oder auch synchronisiert werden muss. Entsprechend wird dieses Pattern von den wenigsten WfMS unterstützt. Ähnlich wie bei Pattern 6 kann auch dieses Pattern durch Konstrukte aus Parallel Split und Exclusive Choice zusammengestellt werden.

Pattern 8 (Multi-Merge) Stelle im Workflow, an der zwei oder mehrere Ausführungszweige ohne Synchronisation zusammenlaufen. Wird mehr als ein Zweig aktiviert, so wird die Aktivität nach dem Multi-Merge für jeden aktivierten Zweig einmal gestartet. Dabei spielt es keine Rolle ob die aktivierten Zweige nacheinander oder parallel durchlaufen werden.

Dieses Pattern wird verwendet, wenn zwei oder mehrere Aktivitäten die gleiche Folgeaktivität haben. Anstatt die Folgeaktivität mehrmals zu modellieren, kann einfach der Multi-Merge benutzt werden.

Einige WfMS bieten ein Merge-Symbol für die direkte Implementierung dieses Patterns an. Bei anderen muss man die sich wiederholende Aktivität mehrfach modellieren. (s. Abbildung 0.2) Dies ist allerdings nur möglich, wenn sich der Multi-Merge nicht in einer Schleife befindet. In diesem Fall ist die Anzahl der Instanzen, die von der jeweiligen Aktivität benötigt werden, zur Entwurfszeit noch nicht bekannt. Um das Problem zu lösen, müssen Pattern 14 oder Pattern 15 verwendet werden.

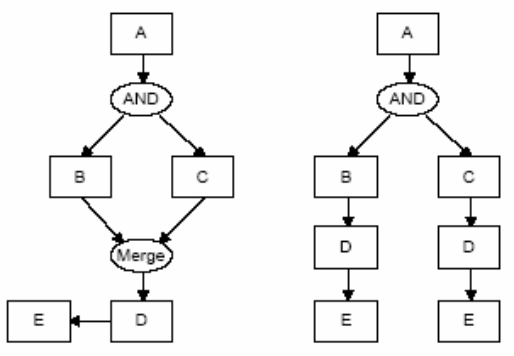


Abbildung 0.2 Implementierung des Multi-Merge [AHKB03]

Pattern 9 (Discriminator) Der Discriminator aktiviert die nachfolgende Aktivität, sobald einer der einmündenden Zweige beendet ist. Danach wird gewartet bis alle restlichen aktivierten Zweige beendet sind, diese lösen allerdings, anders als beim Multi-Merge, keine weitere Aktivierung der Folgeaktivität aus. Das Warten auf die Beendigung aller aktivierten Zweige ist dennoch wichtig, damit der Discriminator wieder in den Initialzustand zurückgesetzt wird und somit in Schleifen benutzt werden kann. Ohne Zurücksetzen könnte es passieren, dass zu einem bestimmten Zeitpunkt, mehrere Zweige aktiv sind, die in unterschiedlichen Schleifendurchläufen aktiviert worden sind.

Structural Patterns

Dieses Kapitel stellt zwei Patterns vor, bei denen in den WfMS typischerweise oft Beschränkungen auftreten. Da sich die Beschränkungen auf die Struktur des Workflows beziehen, gehören diese Patterns zu den Structural Patterns [AHKB03].

Pattern 10 (Arbitrary Cycles) Stelle im Workflow, an der eine oder mehrere Aktivitäten wiederholt ausgeführt werden können.

Arbitrary Cycles unterscheiden sich von den üblicherweise implementierten Schleifen dadurch, dass sie mehrere Eintritts- und Austrittspunkte haben können und dass man mehrere Schleifen überlappend ausführen kann. Schleifen, die nur einen Eintritts- und Austrittspunkt haben und keine Überlappung mit anderen Schleifen zulassen, werden Structured Cycles genannt.

Pattern 11 (Implicit Termination) Ein Prozess endet, wenn nichts mehr zu tun ist. Mit anderen Worten, es gibt weder aktive Aktivitäten im Workflow, noch können irgendwelche Aktivitäten aktiviert werden, wobei der Workflow in keinem Deadlock sein darf.

In den meisten WfMS ist ein Prozess beendet wenn ein spezieller Endknoten erreicht ist. Es kann auch mehrere Endknoten geben, wobei dann der Workflow beendet ist, sobald einer dieser Endknoten erreicht wurde. Dabei werden alle noch aktiven oder laufenden Aktivitäten abgebrochen.

Patterns involving Multiple Instances (MI)

Die Patterns dieses Kapitels besitzen Konstrukte, die von einzelnen Aktivitäten mehrere Instanzen zur gleichen Zeit erlauben. Diese Konstrukte werden allerdings von den meisten Workflow Engines nicht unterstützt, da es entweder Einschränkungen bei der Modellierung gibt oder weil einfach nicht berücksichtigt wurde, dass mehrere Instanzen einer Aktivität durchaus sinnvoll sein können.

Für Patterns mit mehreren Instanzen gibt es zwei Forderungen. Die erste ist die Möglichkeit von einer Aktivität mehrere Instanzen starten zu können. Diese Forderung gilt für alle Patterns dieses Kapitels. Die zweite Forderung ist die Möglichkeit der Synchronisation der laufenden Instanzen. Diese Forderung gilt allerdings nicht für Patterns, die laut Definition keine Synchronisation der Instanzen benötigen. In diesen Fällen wird eine implizite Terminierung der Instanzen gefordert.

Wird eine Synchronisation der Instanzen benötigt, so ist die Zahl der gestarteten Instanzen wichtig für die Workflow-Steuerung. Je nach Anwendungsfall kann diese Zahl schon beim Entwurf oder erst zur Laufzeit bekannt sein. Wird keine Synchronisation benötigt, so ist die Zahl der benötigten Instanzen irrelevant. Im Folgenden werden Patterns für alle möglichen Fälle vorgestellt [AHKB03].

Pattern 12 (Multiple Instances without synchronization) Innerhalb derselben Workflow-Instanz werden mehrere Instanzen einer Aktivität erzeugt. Die einzelnen Instanzen sind voneinander unabhängig, wodurch keine Synchronisation nötig ist.

Pattern 13 (MI with a priori known design time knowledge) Innerhalb derselben Workflow-Instanz werden mehrere Instanzen einer Aktivität erzeugt, wobei die Zahl der Instanzen zur Entwurfszeit bekannt ist. Sobald alle Instanzen beendet sind, kann die nachfolgende Aktivität gestartet werden.

Die Implementierung dieses Patterns ist einfach, da die Anzahl „N“ der benötigten Instanzen schon zur Entwurfszeit bekannt ist. In einigen Fällen kennt man zur Entwurfszeit die Anzahl der benötigten Instanzen allerdings nicht. Dafür gibt es die folgenden zwei Patterns.

Pattern 14 (MI with a priori known runtime knowledge) Innerhalb derselben Workflow-Instanz werden mehrere Instanzen einer Aktivität erzeugt, wobei die Zahl der Instanzen erst zur Laufzeit bekannt wird, aber noch bevor die Instanzen der jeweiligen Aktivität erzeugt werden müssen. Sind alle Instanzen beendet, so kann die nächste Aktivität gestartet werden. Dieses Pattern wird benötigt, wenn die Zahl der Instanzen von gewissen Prozessparametern oder von der Verfügbarkeit von Ressourcen abhängig ist.

Pattern 15 (MI with no a priori runtime knowledge) Innerhalb derselben Workflow-Instanz werden mehrere Instanzen einer Aktivität erzeugt, wobei die Zahl der benötigten Instanzen erst bekannt wird, wenn diese erzeugt werden müssen. Sind alle Instanzen

beendet, so kann die nächste Aktivität gestartet werden. Der Unterschied zwischen diesem und dem vorherigen Pattern ist, dass bei diesem Pattern Instanzen erzeugt werden können, wenn andere bereits ausgeführt werden oder sogar schon beendet sind.

State-based Patterns

Die bisher betrachteten Patterns waren unabhängig von Zuständen. Die meisten realen Workflow-Instanzen befinden sich allerdings meistens in einem bestimmten Zustand, in dem sie auf die Ausführung warten. Im Folgenden werden Patterns vorgestellt, die die auch diesen Aspekt berücksichtigen [AHKB03].

Da die beiden später zu betrachtenden Systeme keines dieser Patterns unterstützen, werden sie hier nur der Vollständigkeit halber kurz aufgeführt.

Pattern 16 (Deferred Choice) Stelle im Workflow, wo einer von mehreren Bearbeitungszweigen gewählt wird. Im Gegensatz zum XOR-Split ist die Wahl nicht explizit. Und im Gegensatz zum AND-Split werden zwar zunächst alle Zweige zur Ausführung angeboten, aber sobald ein Zweig gewählt wird können die anderen nicht mehr ausgeführt werden.

Zu beachten ist, dass die Wahl zum spätestmöglichen Zeitpunkt fällt, und zwar erst dann, wenn die erste Aktivität von einem der Bearbeitungszweige gestartet wird.

Pattern 17 (Interleaved Parallel Routing) Eine Menge von Aktivitäten wird in beliebiger Reihenfolge ausgeführt. Es wird jede Aktivität ausgeführt, die Reihenfolge wird allerdings erst zur Laufzeit entschieden und es werden keine zwei Aktivitäten zur gleichen Zeit ausgeführt. Dieses Pattern wird z.B. verwendet wenn mehrere parallele Aktivitäten auf gleiche Ressourcen oder Daten zugreifen, aber ein konkurrierender Zugriff zu Fehlern führen kann.

Pattern 18 (Milestone) Die Aktivierung einer Aktivität hängt vom Zustand der Workflow-Instanz ab.

Vor der Aktivierung der jeweiligen Aktivität, muss also der Zustand des Workflows geprüft werden.

Cancellation Patterns

Die folgenden Patterns sind für die Deaktivierung von Aktivitäten oder für den Abbruch von ganzen Workflow-Instanzen [AHKB03]. Da diese Patterns bei der Visualisierung nicht relevant sind, werden sie ebenfalls nur der Vollständigkeit halber kurz aufgeführt.

Pattern 19 (Cancel Activity) Eine aktivierte Aktivität wird deaktiviert.

Gibt es von einer Aktivität mehrere Instanzen, so ist mit diesem Pattern immer nur die Deaktivierung einer einzigen Instanz gemeint.

Pattern 20 (Cancel Case) Eine komplette Workflow-Instanz wird abgebrochen. Existieren von Aktivitäten mehrere Instanzen, so werden hiermit auch all diese Instanzen abgebrochen.

Anhang C (Vollständige Tabelle zu Tabelle 6.2)

Field Name	Column name of database table	Type	Explanation
Timestamp (DB2)	CREATED	TIMESTAMP Mandatory	Date and time the audit trail record is written.
Timestamp (Oracle)	CREATED	TIMESTAMP_ WF Mandatory	Date and time the audit trail record is written.
Event	EVENT	INTEGER Mandatory	Type of event as indicated in Table 5 on page 76.
Process Name	PROCESS_NAME	VARCHAR (63) Mandatory	Name of the process instance.
Process Identifier	PROCESS_ID	IDENTIFIER Mandatory	Object identifier of the process instance.
Top-level Name	TOP_LVL_PROC_NAME	VARCHAR (63) Mandatory	Name of the top-level process instance if the process instance is executing as subprocess, or the same as in process name if the process instance is a top-level process instance.
Top-level Identifier	TOP_LVL_PROC_ID	IDENTIFIER Mandatory	Object identifier of the top-level process instance if the process is executing as subprocess, or the same as in process identifier if the process instance is a top-level process instance.
Parent Process Name	PARENT_PROC_NAME	VARCHAR (63) Optional	Name of the parent process instance if the process instance is executing as a subprocess.
Parent Process Identifier	PARENT_PROC_ID	IDENTIFIER Optional	Object identifier of the parent process instance if the process instance is executing as a subprocess.
Process Model Name	PROC_TEMPL_NAME	VARCHAR (32) Mandatory	Name of the process model.
Process Model Valid from Date	TEMPL_VALID_FROM	TIMESTAMP Optional	The date the associated process model becomes valid.
Block Names	BLOCK_NAMES	VARCHAR (254) Optional	The concatenated names of all blocks in which the activity is contaided in. The various names are separated by a dot.
User ID	USER_NAME	VARCHAR (32) Optional	ID of the user associated with the event that caused the audit trail to be written. If the audit trail record is written by the MQ Workflow system, this field is not filled.
Second user ID	SECOND_USER_NAME	VARCHAR (32) Optional	ID of the second user associated with the event that caused the audit trail to be written.
Activity Name	ACTIVITY_NAME	VARCHAR (32) Optional	If the audit trail entry is associated with an activity, the field contains the name of the activity. If the audit trail entry is associated with a control connector, the field contains the name of the activity that is the source of the control connector.
Activity Type	ACTIVITY_TYPE	INTEGER Optional	If the audit trail record is written for an activity, the field contains the type of the activity as defined in Table 6 on page 79.
Activity Status	ACTIVITY_STATE	INTEGER Optional	If the audit trail record is written for an event associated with an activity, the field contains the status of the activity encoded as shown in Table 7 on page 80.
Second Activity Name	SECOND_ACT_NAME	VARCHAR (32) Optional	If the audit trail is written for an event associated with a control connector, the field contains the name of the target activity.
Command Parameters	COMMAND_PARAMETERS	VARCHAR (1024) Optional	If the event is the start of a program activity, the field contains the actual parameters passed when invoking the program.
Asociated Objekt	ASSOCIATED_OBJECT	IDENTIFIER Optional	Contains the identifier of the object associated with the event. Can be used to locate the object in the MQ Workflow database.
Object Description	OBJECT_DESCRIPTION	VARCHAR (254) Optional	Contains the description of the object associated with the event.
Program Name	PROGRAM_NAME	VARCHAR (32) Optional	If the event is the start of a program activity, the field contains the name of the program.
Activity Return	ACTIVITY_RC	LONG Optional	Return code of the activity.
Global Container	CONTAINER_CONTENT	BLOB (4MB) Optional	Contains the content of the global container as XML stream.
External Process	EXTERNAL_CONTEXT	VARCHAR (254) Optional	Contains the extemal context passed during the start call of the process instance.

Anhang D (weitere Beispiele zu Abschnitt 6.4.1.2)

Bedingte Verzweigung:

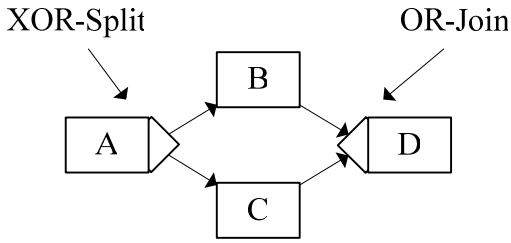


Abbildung 0.3 Beispiel – Bedingte Verzweigung

Ereignisse / Aktivität	A	B	C	D
case started	ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED
A started	RUNNING	NOT_ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED
A completed	COMPLETED	ACTIVATED	SKIPPED	NOT_ACTIVATED
B started	COMPLETED	RUNNING	SKIPPED	NOT_ACTIVATED
B completed	COMPLETED	COMPLETED	SKIPPED	ACTIVATED
D started	COMPLETED	COMPLETED	SKIPPED	RUNNING
D completed	COMPLETED	COMPLETED	SKIPPED	COMPLETED
case completed	COMPLETED	COMPLETED	SKIPPED	COMPLETED

Tabelle 0.1 Zustände für Abbildung 0.3 (abhängig von den geloggten Ereignissen)

Falls statt Aktivität B Aktivität C aktiviert wird, dann ist einfach Spalte B mit Spalte C zu vertauschen.

Parallele Verzweigung

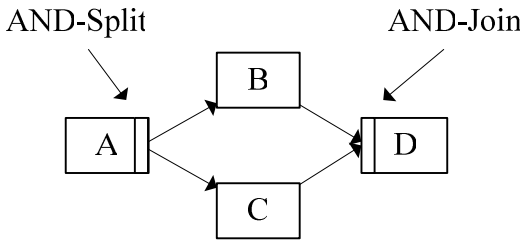


Abbildung 0.4 Beispiel - Parallele Verzweigung

Ereignisse \ Aktivität	A	B	C	D
case started	ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED
A started	RUNNING	NOT_ACTIVATED	NOT_ACTIVATED	NOT_ACTIVATED
A completed	COMPLETED	ACTIVATED	ACTIVATED	NOT_ACTIVATED
B started	COMPLETED	RUNNING	ACTIVATED	NOT_ACTIVATED
C started	COMPLETED	RUNNING	RUNNING	NOT_ACTIVATED
B completed	COMPLETED	COMPLETED	RUNNING	NOT_ACTIVATED
C completed	COMPLETED	COMPLETED	COMPLETED	ACTIVATED
D started	COMPLETED	COMPLETED	COMPLETED	RUNNING
D completed	COMPLETED	COMPLETED	COMPLETED	COMPLETED
case completed	COMPLETED	COMPLETED	COMPLETED	COMPLETED

Tabelle 0.2 Zustände für Abbildung 0.4 (abhängig von den geloggten Ereignissen)

Dabei ist die Reihenfolge der Start- und Endereignisse von B und C unbedeutend. Entscheidend ist nur, dass sowohl B als auch C beendet sein müssen bevor D gestartet werden kann.

Anhang E (Sequenzdiagramme für die Runtime-Komponente)

Alle Applikationsdaten zu einem bestimmten CorrelationSet und ab einem bestimmten Zeitpunkt holen: *getApplicationData(correlationSet, time)*

„time“ dient bei der Aktualisierung dazu, dass die Daten nicht immer komplett geladen werden, sondern nur ab der letzten Aktualisierung. (anfangs ist „time“=0)

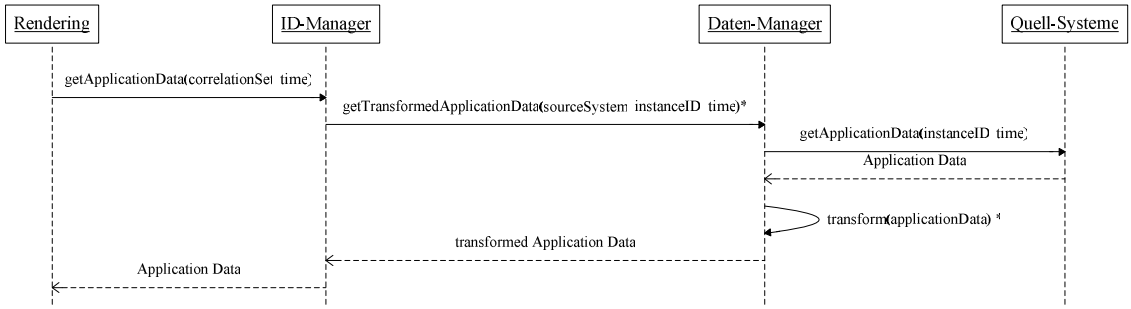


Abbildung 0.5 Sequenzdiagramm - Beschaffung aller Applikationsdaten

Alle Log-Daten zu einem bestimmten CorrelationSet holen (getLogData)

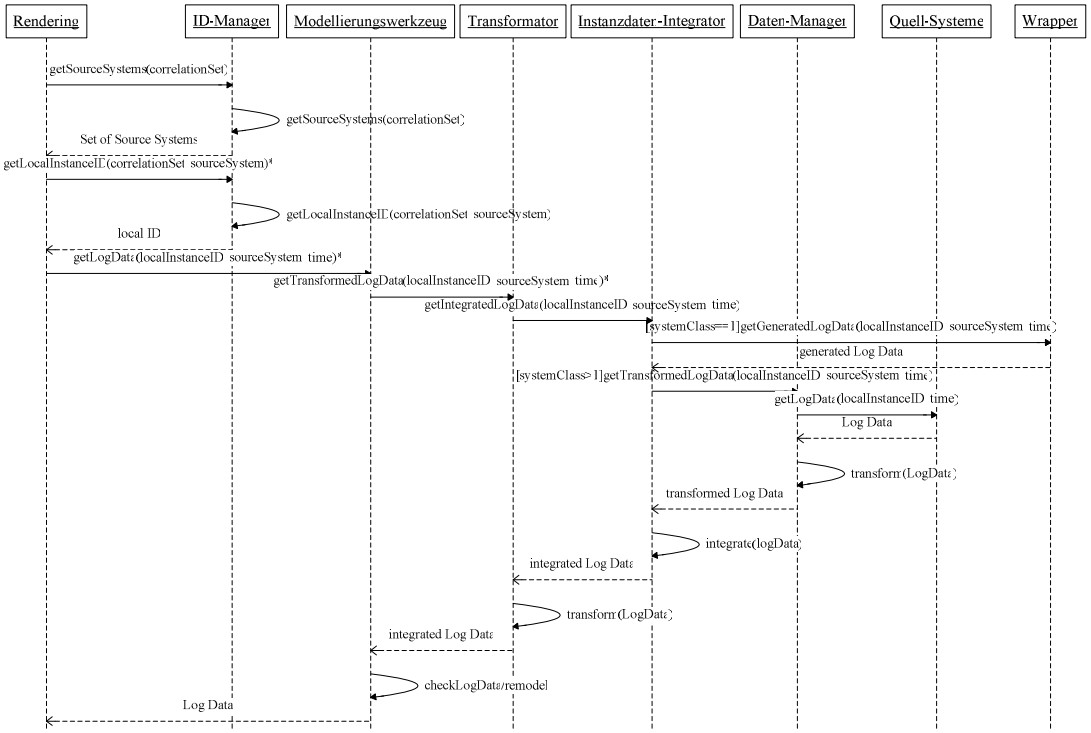


Abbildung 0.6 Sequenzdiagramm – Beschaffung aller Log-Daten

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 14.11.2005 _____

Tiberius Mihalca