Flexible, datengetriebene Workflows für den Publikationsprozess in digitalen Bibliotheken

Dissertation
zur
Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)
der Fakultät für Informatik und Elektrotechnik
der Universität Rostock

vorgelegt von
Dipl.-Inf. Sebastian Schick, geb. am 11.07.1980 in Rostock
aus Rostock

Rostock, 13.04.2012

urn:nbn:de:gbv:28-diss2012-0076-9

Gutachter:
Prof. Dr. Andreas Heuer, Universität Rostock
Prof. Dr. Manfred Reichert, Universität Ulm
Prof. Dr. Norbert Ritter, Universität Hamburg

Datum der Einreichung: 13.04.2012
Datum der Verteidigung: 19.07.2012

Kurzfassung

Die Verwaltung digitaler, wissenschaftlicher Publikationen, z.B. Dissertationen, hat sich in den letzten Jahren zu einem wichtigen Aufgabengebiet von Universitätsbibliotheken entwickelt. Um solche komplexen Szenarien zu strukturieren, zu kontrollieren und den Publikationsprozess zu unterstützen, werden Workflow-Management-Systeme eingesetzt. Für die Beschreibung des Publikationsprozesses werden dabei Prozessmodelle verwendet, die für alle zulässigen Dokumentvarianten die korrekte Abarbeitung garantieren müssen.

Im Publikationsprozess werden multimediale Dokumente bearbeitet, deren Struktur und Inhalt sich oft über den gesamten Prozess hinweg verändert. Häufig werden diese Dokumente in externen Datenquellen verwaltet, die nicht unter der Kontrolle des Workflow-Management-Systems stehen. Vorhandene Prozessmodelle bieten bisher keine ausreichende Unterstützung bei der Integration externer Dokumente. Es fehlen Mechanismen, um Inkonsistenzen zwischen den redundant verwalteten Dokumenten zu vermeiden.

Ziel der Arbeit war die Entwicklung eines Workflow-Management-Systems, das Dokumente aus externen Datenquellen einbinden kann und die flexible Adaption von Prozessinstanzen ermöglicht. Das Workflow-Management-System soll den einheitlichen, transaktionalen Zugriff auf die externen Datenquellen umsetzen und die flexible Adaption von Prozessinstanzen in Abhängigkeit vom Zustand externer Dokumente unterstützen.

Mit dem *tx+YAWL*-Ansatz wird ein Lösungskonzept vorgestellt, das den transaktionalen Zugriff auf externe Datenquellen ermöglicht. Dafür wird ein Konzept für die einheitliche Integration unterschiedlicher Datenquellen vorgestellt, das den direkten Zugriff auf externe Dokumente innerhalb der Kontrollflussperspektive ermöglicht. Mit Transaktionen stellen wir sicher, dass Eigenschaften wie Konsistenz, Rücksetzbarkeit und Dauerhaftigkeit für den Datenzugriff sichergestellt sind. Für den konsistenten Zugriff auf externe Datenquellen wird ein Mehrebenen-Transaktionsmodell verwendet. Durch die Einführung eines lokalen Arbeitsbereiches und transaktionaler Konzepte werden Eigenschaften wie Isolation und Atomarität für Prozessbereiche sichergestellt.

Mit dem *FlexY*-Ansatz wird ein Konzept für die flexible, datengetriebene Anpassung von Prozessinstanzen vorgestellt. Damit kann der Zustand von Dokumenten und deren Inhalt an beliebigen Stellen in einer Prozessinstanz überwacht werden. Entsprechend können Prozessbausteine aktiviert werden, die dokumentabhängige Arbeitsschritte zusammenfassen. Diese Prozessbausteine werden zur Laufzeit für die flexible Konstruktion und Ausführung von Prozessbereichen verwendet.

Die vorgestellten Ansätze werden durch einen Prototyp umgesetzt und in zwei unterschiedlichen Anwendungsgebieten, einem digitalen Bibliothekssystem und einem klinischen Assistenzsystem, eingesetzt.

Abstract

The management of digital, scientific publications, e.g dissertations, has become an important area of responsibility of university libraries. A common solution to control the complex publication process scenarios is the integration of a workflow management system within the digital library systems. For the description of the publication processes, process models are used. These models have to guarantee correct execution for all admissible document variations.

In the publication process multimedia documents are processed, which structure and content is often changing over the entire process. The documents are managed in external data sources, which are not under the control of the workflow management systems. Existing process models have insufficient support for the integration of external documents. It lacks mechanisms to avoid inconsistencies between redundantly managed documents. The goal is to develop a workflow management system, that can integrate documents of external data sources and allows for the flexible adaptation of process instances. The workflow management system has to implement a uniform and transactional access to external data sources. It has also to support the flexible adaptation of process instances, based on the state of the external documents. With transactions we associate properties such as consistency, durability and resettability.

The *tx+YAWL* approach presents a solution, which allows transactional access to external data sources. Therefore, a unified approach for the integration of different data sources is presented. It allows direct access to external documents within the control flow perspective. For consistent access to external data sources, a multi-level transaction model is used. By the introduction of a local work area and transactional concepts, properties such as isolation and atomicity are ensured for process parts.

The *FlexY* approach presents a solution for flexible, data-driven adaptation of process instances. Thus, the state of documents and the documents content can be monitored at any point in a process. Accordingly, process modules are activated, which assemble the document-oriented operations. These process blocks are used for the dynamic construction of process areas, which are executed at run time.

The approaches are implemented by a prototype, which is used within two different fields of application. A digital library system and a clinical assistant system are the example scenarios.

Danksagung

An dieser Stelle bedanke ich mich bei Herrn Prof. Dr. Andreas Heuer für die Überlassung des Themas und die Betreuung der Arbeit. Weiterhin bedanke ich mich bei Herrn Dr. Holger Meyer. Seine Ideen haben wesentlich zum Gelingen der Arbeit beigetragen.

Für interessante Diskussionen möchte ich mich bei Frau PD Dr. Meike Klettke und Herrn Markus Bandt bedanken. Weiterhin bedanke ich mich bei Herrn Robert Stephan. Insbesondere für seine Unterstützung und für wichtige Hinweise bei der Arbeit mit der digitalen Bibliothek *RosDok*. Mein Dank gilt auch Herrn Marcel Paleit für seine tatkräftige Unterstützung bei der Entwicklung des Prototypen.

Ein abschließender Dank gilt allen Studenten und Mitarbeitern, die an der Entwicklung beteiligt waren und mich bei der Erstellung der Arbeit unterstützt haben.

Inhaltsverzeichnis

Teil I

Motivation und Anforderungen

Kapitel 1

Einleitung

1.1 Problemstellung und Einordnung der Arbeit

Informationssysteme halten in immer mehr Bereichen der Informationsverarbeitung Einzug. Dabei ist es das Ziel, einen Großteil der für den Benutzer anfallenden Aufgaben automatisch abzuarbeiten. Je nach Anwendungsgebiet haben sich durch die verschiedenen Anforderungen unterschiedliche Systeme entwickelt.

Ein Einsatzgebiet von Informationssystemen liegt in der Publikation von digitalen, wissenschaftlichen Dokumenten. Die zu verwaltenden Dokumente variieren dabei in Struktur, Inhalt, eingebetteten Medien und ihrer physikalischen Repräsentation. Vielfach werden wissenschaftliche Publikationen (als PDF-Dokumente), aber auch spezielle Sammlungen (z. B. Notenhandschriften oder digitale Jahrbücher) digital bereitgestellt. Für die Veröffentlichung der digitalen Dokumente gehen Universitäten und Universitätsbibliotheken immer häufiger dazu über, eigene Dokumentenservern bzw. Repositorien anzubieten [GSU08]. Die Dokumentenserver bieten zudem die Möglichkeit der Volltextrecherche, den Zugriff über entsprechende Klassifikationen und die Zugangskontrolle für einzelne Dokumente an.

Wissenschaftliche Publikationen sind zunehmend mit komplexen Informationen angereichert. Hierzu zählen z. B. Experimentaldatensätze, Audio- und Video-Material, zusammengesetzte Objekte sowie beschreibende Daten (z. B. Metadaten). An dieser Stelle müssen sich die Bibliotheken neuen Herausforderungen stellen, um die Funktionalitäten an die veränderten Anforderungen anzupassen. Um komplexe Multimediadokumente geeignet zu verwalten und bereitstellen zu können, muss der gesamte Lebenszyklus digitaler Objekte unterstützt werden. Dies bezieht sich auf das gemeinschaftliche Erzeugen der Objekte, bis hin zur gemeinschaftlichen Exploration, Manipulation und Austausch [MMC+10, CCL+06]. Hierfür müssen die gängigen Operationen wie z. B. inhaltsbasierte Suche, welche an die Medientypen gebunden sind, erweitert werden. Neben diesen Anforderungen sollten digitale Bibliothekssysteme

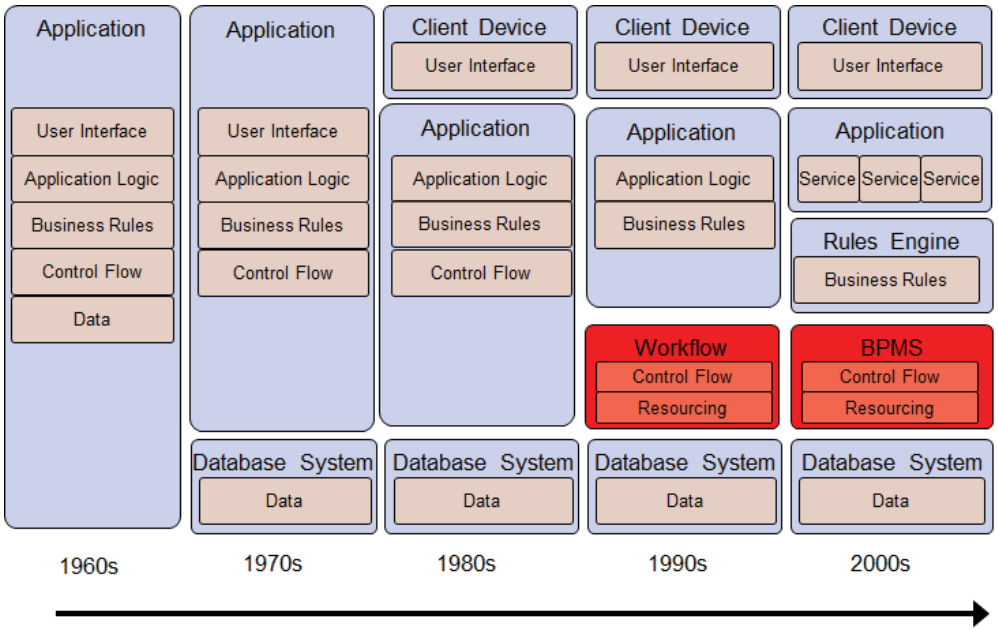


Abbildung 1.1: Entwicklung der Business-Management-System Technologie [HAAR10]

außerdem eine langfristige Archivierung und Verfügbarkeit der multimedialen Dokumente sicherstellen. Daneben werden auch verschiedene Services, die es dem Nutzer ermöglichen, Inhalte mit Hilfe von Web 2.0 Technologien zu erweitern, immer wichtiger.

Um solche komplexen Szenarien zu strukturieren und zu kontrollieren, ist der Einsatz von Workflow-Management-Systemen allgemein akzeptiert [BJVW10]. Sie bieten die Möglichkeit, dass Informationssystem (IS) schnell und flexibel an neue Situationen anzupassen. Abbildung 1.1 zeigt, wie sich die Architektur der Systeme über die Zeit hin verändert hat.

Prozessmodelle beschreiben explizit den Arbeitsablauf, der mit Hilfe des Workflow-Management-Systems (WFMSs) kontrolliert abgearbeitet wird [Wes07]. Aktivitäten sind dabei die atomaren Bausteine eines Prozessmodells, deren geordnete Ausführung eine bestimmte Aufgabe erfüllt. Eine Aktivität wird entweder automatisch oder durch den Benutzer ausgeführt.

Der *Publikationsprozess* beschreibt das Erstellen komplex strukturierter Dokumente in einer digitalen Bibliothek. Die aktuellen Systeme setzen oft voraus, dass die Dokumente bereits im Vorfeld erstellt wurden. Da die Prozesse in digitalen Bibliotheken aber von den verwendeten Dokumenttypen abhängen (z. B. ist die Indizierung für Medientypen wie Text, Bild oder Video unterschiedlich), sollte das Erzeugen nicht unabhängig von ihrer Benutzung erfolgen. Oftmals werden die Autoren jedoch nicht ausreichend bei der Erstellung solcher Dokumente unterstützt (z. B. bei der Zusammenstellung neuer logischer Dokumente). Der Aufwand, diese Prozessmodelle zu verwalten, steigt mit jedem Dokumenttyp, mit dem das Dokumentenmodell[1] erweitert wird. Deshalb sollten Dokumente und die Prozesse nicht unabhängig

[1] siehe Abschnitt 2.1.2

voneinander betrachtet werden. Ziel ist es zum einen, dass Prozesse auf Änderungen in den Dokumenten flexibel reagieren können. Zum anderen muss die Modellierung besser unterstützt werden. In dieser Arbeit wird die Erweiterung der Kontrollfluss-Perspektive eines Prozessmodells vorgeschlagen, indem Dokumente, deren Strukturen und Inhalte gemeinsam modelliert werden können.

1.1.1 Prozessvariablen und externe Datenquellen

Ein wichtiger Teil der Prozessmodellierung ist die Abbildung der Anwendungsanforderungen auf den Kontrollfluss des Prozessmodells. Der Kontrollfluss modelliert anhand von Regeln die Ausführungsreihenfolge der Aktivitäten. Regeln stellen dabei Bedingungen an Prozessvariablen dar, die entweder durch Nutzer aber auch durch die integrierten Services manipuliert werden. Damit die Regeln auf komplexe Dokumente, welche z. B. in verschiedenen externen Systemen verwaltet werden, angewendet werden können, müssen diese Daten auf Prozessvariablen abgebildet werden. In aktuellen Systemen ist die Umsetzung häufig nur durch den expliziten Aufruf einer Aktivität und dem damit verbundenen Aufruf einer Applikation möglich, welche die Daten kopiert [KR09]. Dieses Konzept der Datenintegration führt jedoch zu einer Reihe von Problemen, wie z. B. zu komplexen Prozessmodellen. Die Güte eines Prozessmodells wird häufig daran gemessen, wie genau es den eigentlichen Arbeitsablauf beschreibt [HMR08, SGGMR10]. Wird z. B. die Integration von Daten durch den Aufruf von Services gekapselt, gehen die Beziehungen zwischen Prozess und Daten verloren. Sie wird im Quelltext verborgen und ist damit schwer kontrollierbar und wartbar. Ein weiterer Nachteil besteht darin, dass nur eingeschränkte Funktionalitäten zum Verwalten der Daten im WFMS bereitstehen. Funktionen zum persistenten Speichern von Prozessvariablen stehen im WFMS üblicherweise nicht bereit. Die Daten müssen gegebenenfalls mit den Anwendungsdaten synchron gehalten werden, da sie nach Prozessende im WFMS gelöscht werden. Dies zieht wiederum komplexe Prozessmodelle nach sich.

Die Integration externer Datenquellen wird in [REHA04, RHEA05] näher betrachtet. Die Autoren stellen eine Reihe von Mustern vor, die die Verwendung von Daten in einem Prozessmodell beschreiben. Die Muster 14 und 15, aus [REHA04, RHEA05], beschreiben den Datenaustausch zwischen einer Prozess-Aktivität und externen Datenquellen. Die Muster 18 und 19, aus [REHA04, RHEA05], beschreiben den Datenaustausch von einem Prozess (engl. *case*) mit einer externen Datenquelle. Diese Muster werden aktuell aber von einigen Systemen nur teilweise umgesetzt. Da die Daten in den externen Datenquellen auch von anderen Systemen verändert werden können, die nicht unter der Kontrolle des WFMSs stehen, muss das WFMS außerdem geeignet auf solche Änderungen reagieren können.

Aspekte wie Integritätsbedingungen oder Isolationseigenschaften auf externen Datenquellen müssen ebenfalls betrachtet werden. Außerdem muss die Möglichkeit bestehen, Daten

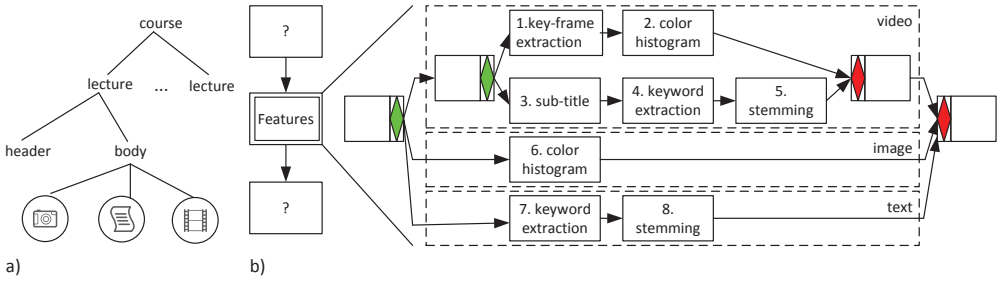


Abbildung 1.2: Ausschnitt aus einem Prozess

konsistent in die Datenquellen zurückzuschreiben. Um Inkonsistenzen zwischen externen Datenquellen und den redundant verwalteten Prozessvariablen zu vermeiden, müssen außerdem Transaktionskonzepte in der Kontrollflussperspektive und der Datenperspektive der Prozessmodelle eingeführt werden.

1.1.2 Flexible Prozesse

Neben der Integration von externen Datenquellen für den Kontrollfluss spielt der Zusammenhang zwischen Datenmodellen und Prozessmodellen eine sehr wichtige Rolle. Prozessmodelle beschreiben die Verarbeitung von Dokumenten, deren Dokumentmodelle alle möglichen Dokumentausprägungen beschreiben. Innerhalb eines Prozessmodells können Bereiche identifiziert werden, die die Bearbeitung eines Dokumentfragments[2] beschreiben. Abbildung 1.2 soll diesen Zusammenhang verdeutlichen, indem ein Ausschnitt aus dem Publikationsprozess komplexer Multimediadokumente gezeigt wird. Die Extraktion von medienspezifischen Merkmalen (Feature Extraction) ist ein wichtiger Bestandteil moderner Informationssysteme. Dabei muss jeder Dokumenttyp (Text, Bild und Video) durch entsprechende Prozessschritte bearbeitet werden.

Die Anforderungen an Informationssysteme steigen mit der Komplexität der Dokumentmodelle. Entsprechend wird die Dokument-Verarbeitung immer aufwendiger und muss durch geeignete Anwendungen unterstützt werden. Abbildung 1.2 verdeutlicht ein zweites Problem: Die Komplexität der Prozessmodelle steigt im gleichen Maß wie die der Dokumentmodelle. Verschiedene Frameworks versuchen durch die Kombination von serviceorientierten Architekturen und WFMS die Systeme flexibel zu halten [MMC+10, BOWN06, MK05]. Änderungen an den Dokumenten und damit auch eine Anpassung der Prozesse und benötigten Services sind so nur bedingt möglich.

Flexible Prozessmodelle bieten mehr Möglichkeiten, den Arbeitsablauf an die Gegebenheiten anzupassen [RRMD09]. Dabei wird grundsätzlich unterschieden, ob die Flexibilität auf

[2] siehe Abschnitt 2.1.2

Prozessmodell-Ebene oder auf Prozessinstanz-Ebene[3] umgesetzt wird. Daneben existieren ebenfalls hybride Verfahren. Eine ausführliche Klassifikation aktueller Verfahren wird in [WSR09, SMR+08] beschrieben.

Ansätze, die verschiedene Prozessvarianten in Abhängigkeit von Kontextinformationen verwalten [GAJVR08, HBR08], ermöglichen Flexibilität zur Modellierungszeit. Im Vorfeld wird anhand verschiedener Informationen eine Prozessvariante ausgewählt, die dann für die Bearbeitung verwendet wird. Dies führt zu wesentlich übersichtlicheren Prozessmodellen. Zusätzlich kann die Komplexität der Prozessinstanzen optimiert werden. Ändert sich während der Laufzeit eines Prozesses der Kontext (z. B. wird ein Dokumenttyp hinzugefügt), sind die Anpassungen der Prozessinstanz nur bedingt möglich. Zum einen werden Modellierungskonzepte wie Auswahl, mehrfache Ausführung von Aktivitäten (multiple tasks) oder Abbruch von Aktivitäten verwendet [SMR+08]. Zum anderen können in Abhängigkeit vom verwendeten Prozessmodell auch Änderungen an der Prozessinstanz vorgenommen werden, auf die weiter unten eingegangen wird.

Flexiblere Möglichkeiten bieten Ansätze, die Prozessmodelle bereitstellen, welche zur Laufzeit Änderungen an der Prozessinstanz ermöglichen. In [SMR+08] werden vier grundlegende Konzepte unterschieden: *Flexibilität durch Modellierung*, *Flexibilität durch Abweichung*, *Flexibilität durch Unterspezifikation* und *Flexibilität durch Änderungen*.

Viele Prozessmodelle unterstützten *Flexibilität durch Modellierung*. Während der Ausführung von Prozessinstanzen kann durch unterschiedliche Ausführungspfade flexibel auf verschiedene Situationen reagiert werden. Eine andere Möglichkeit Flexibilität zu unterstützen, bieten Konzepte die *Flexibilität durch Abweichung* umsetzen. Eine Abweichung ist z. B. das Überspringen von Aktivitäten.

Das Konzept der Unterspezifikation definiert innerhalb eines Prozessmodells Bereiche (*Platzhalter*), welche erst zur Laufzeit genau spezifiziert werden müssen. Dabei wird nochmals in *Late Modeling* und *Late Binding* unterschieden. Ansätze, die das Prinzip des *Late Bindings* verfolgen (z. B. [AHEA06a]), bieten die Möglichkeit, aus einer vorher definierten Menge von Prozessfragmenten (die wiederum einzelne Aktivitäten oder Subnetze darstellen können) auszuwählen. Nachteilig gestaltet sich hier, dass die zu integrierenden Prozessmodelle nicht zur Laufzeit anpassbar sind. Dieses Problem löst das Prinzip des *Late Modelings* und bietet die Möglichkeit, die Prozessfragmente, welche einen Platzhalter ersetzen, erst zur Laufzeit zu modellieren [SOS05]. Die Modellierung kann entweder (semi-) automatisch durch Bedingungen oder durch einen Nutzer gesteuert werden. Die Konfiguration mit Hilfe von Bedingungen ist von deren Definition abhängig. Eine direkte Abbildung von Dokumentoperationen auf Bedingungen, die die Anpassungen an einem Prozess beeinflussen, ist aber nicht möglich.

[3] siehe Abschnitt 2.2

Eine andere Möglichkeit bieten Ansätze, welche Flexibilität durch Änderungen ermöglichen. Änderungsoperationen auf Prozessinstanzen gestatten hier die flexible Anpassung der Prozessinstanzen [WRRM08]. Dabei werden verschiedene Operationen eingeführt, mit deren Hilfe Prozessbestandteile hinzugefügt, entfernt oder verschoben werden können. Ausgangspunkt für diese Operationen ist aber immer das Prozessmodell, an das die jeweiligen Instanzen angepasst werden. Dieses Vorgehen führt zu einem besonders hohen Aufwand, um die Korrektheit der Prozessinstanzen zu gewährleisten [RRD04a]. Sie werden hauptsächlich eingesetzt, um auf unvorhergesehene Ausnahmen während der Abarbeitung eines Prozessmodells reagieren zu können. Häufige Änderungen, die bei der Bearbeitung multimedialer Dokumente auftreten, können deshalb sehr aufwendig bei der Umsetzung sein.

1.2 Lösungsansatz

Ziel der Arbeit ist es, den Publikationsprozess in digitalen Bibliothekssystemen durch den Einsatz eines flexiblen, datengetriebenen Workflow-Management-Systems zu unterstützen. Der Publikationsprozess beschreibt dabei die verschiedenen Phasen (Schreiben, Produzieren, Verteilen, Finden, Zusammenstellen, Archivieren, Lesen / Nutzen) innerhalb des Lebenszyklus von Dokumenten.

In einem ersten Schritt wird der Publikationsprozess in Form einer Prozessbeschreibung modelliert, die mit einem Workflow-Management-System ausführbar ist. Eine Plug-in-Architektur ermöglicht die einheitliche Integration externer Datenquellen als XML-Sichten im WFMS. Jeder Datenquelle wird genau ein Plug-in zugeordnet, welches eine Schnittstelle für Anfrage- und Änderungsoperationen auf die Datenquelle bereitstellt. Durch entsprechende Modellierungskonstrukte können Prozessvariablen an eine Datenquelle via Plug-ins gebunden werden. Zugriff und Verwaltung von Plug-ins werden durch ein Framework bereitgestellt.

Die Konsistenz der Daten wird mit Hilfe von Bedingungen (Constraints), Ausnahmebehandlungen und Transaktionskonzepten ermöglicht. Bedingungen müssen überprüft werden und auf deren Verletzung muss geeignet reagiert werden. Mit Hilfe von Transaktionskonzepten muss die Konsistenz sichergestellt werden, indem der Zeitpunkt des Datenzugriffs nutzerunabhängig gesteuert wird.

Neben Nutzereingaben und Prozessvariablen muss die Abarbeitung des Publikationsprozesses maßgeblich durch den Inhalt und den Zustand der Dokumente aus externen Datenquellen gesteuert werden. Um dynamisch auf Änderungen dieser Dokumente reagieren zu können, wird ein flexibles Prozessmodell benötigt, welches es ermöglicht, die Prozessinstanzen in Abhängigkeit von Struktur und Inhalt der Dokumente zu verändern. Dazu muss der Zustand der Dokumente und deren Struktur zu bestimmten Zeitpunkten im Prozessverlauf abgefragt

werden. Operationen wie z. B. das Hinzufügen, Ändern oder Löschen von Dokumentteilen sollten direkt an die Änderung der Prozessinstanz geknüpft werden. Außerdem müssen Operationen bereitgestellt werden, die das Hinzufügen oder Löschen von Prozessfragmenten ermöglichen. Um eine einheitliche Modellierungsumgebung zu bieten, ist ein Prozessmodell zu erstellen, das alle geforderten Aspekte in einem Modell vereint.

1.3 Verwandte Arbeiten

In den Bereichen digitaler Bibliothekssysteme und Repositorien existieren eine Reihe verschiedener Lösungsansätze. Hier muss zwischen digitalen Bibliothekssystemen und institutionellen Repositorien unterschieden werden. Letztere sind vom Funktionsumfang auf die Bereitstellung von Dokumenten beschränkt. Außerdem existieren verwandte Arbeiten im Bereich des Workflow-Managements. Mehrere Ansätze betrachten Möglichkeiten für die Integration von externen Datenquellen. Der Teilaspekt der Flexibilisierung von Prozessmodellen wird ebenfalls betrachtet.

1.3.1 Digitale Bibliothekssysteme und Repositorien

Häufig wird allgemein von digitalen Bibliotheken gesprochen. Wir nehmen hier eine Einteilung in digitale Bibliothekssysteme und Repositorien vor, um auf deren spezifische Eigenschaften besser eingehen zu können. Digitale Bibliothekssysteme sind häufig Ergebnis von Forschungsprojekten. Sie unterstützen eine Reihe von Techniken, die den Publikationsprozess unterstützen. Dazu gehören z. B. flexible Architekturen, die eine einfache Anpassung der Systeme ermöglichen. Institutionelle Repositorien dienen oft ausschließlich der Verbreitung von Inhalten. Funktionalitäten für die Erstellung oder das Bearbeiten von Inhalten sind auf ein Minimum reduziert.

1.3.1.1 Digitale Bibliothekssysteme

Greenstone. In [BBDW05, BOWN06] wird das Framework Greenstone vorgestellt, welches die Möglichkeit bietet, digitale Bibliotheken zu generieren. Hauptaugenmerk wurde auf eine flexible und erweiterbare Architektur gelegt, die es erlaubt, auf einfache Weise neue Dokumentkollektionen zu erstellen. Für die interne Verwaltung von Dokumenten wird der "Metadata Encoding and Transmission Standard"(METS) [Lib11] verwendet.

Um den Nutzern das Erstellen von Kollektionen, aber auch das Einstellen von neuen Dokumenten, so einfach wie möglich zu gestalten, stellt Greenstone eine serviceorientierte Architektur (SOA) bereit. Die Software ist hierbei in verschiedene Module unterteilt, welche verschiedenen Phasen bei der Bereitstellung von Dokumenten (Expansion, Erkennen,

Codierung, Extraktion, Klassifikation, Indizierung und Validierung) widerspiegeln. Jedes Modul wird durch einen *Manager* repräsentiert, welcher verschiedene Plug-ins aufrufen kann. Anhand einer vorher festgelegten Konfiguration können so verschiedene Plug-ins für einen Manager aktiviert werden. Inwieweit ein Modul bzw. Manager für eine bestimmte Kollektion aktiviert wird oder nicht, wird durch eine weitere XML-Konfigurationsdatei beschrieben. Der Arbeitsablauf wird mit Hilfe verschiedener Techniken umgesetzt. *Makros* definieren den Ablauf durch eine einfache Sprache, bei der ein Name in Verbindung mit einem Inhalt in einer Textdatei angegeben wird. Makros können zusätzlich in *Pakete* zusammengefasst werden. *Runtime actions* erlauben die Integration einzelner Module. Diese Vorgehensweise ermöglicht die flexible Generierung unterschiedlicher digitaler Bibliotheken.

Fedora. Ein Repository-System ist das *Fedora Repository Project* [LPSW06]. *Fedora* steht dabei für die Abkürzung "Flexible Extensible Digital Object Repository Architecture". Die Open-Source Software besteht aus einem Kern, der leicht durch verschiedene Services erweitert werden kann. Basis für die Verwaltung der Daten sind Objekte, die aus verschiedenen Komponenten – die auch als *Datastreams* bezeichnet werden – zusammengesetzt werden. Eine Komponente kann entweder ein konkretes Dokument sein, aber auch ein Verweis auf eine externe Quelle, die durch einen Webservice gekapselt ist. Eine externe Quelle kann z. B. dafür genutzt werden, Inhalte erst zur Laufzeit zu generieren, was aber völlig transparent für den Nutzer abläuft. Außerdem können unterschiedliche Beziehungen zwischen Objekten definiert werden. In Verbindung mit einer einfachen Nutzeroberfläche können so sehr flexible Dokumentstrukturen erzeugt werden, die völlig unabhängig von den verwendeten Datentypen sind.

Verschiedene Aufgaben, wie z. B. die Indizierung von Objekten, werden mit Hilfe von Services umgesetzt. Hierfür stellt der Fedora-Kern einen Nachrichten-Provider bereit, der auf dem Java Messaging Service (JMS) basiert. Der Provider setzt ein asynchrones Publish-und-Subscribe-Verfahren ein. Services können sich am Provider anmelden, um über Änderungen an Objekten informiert zu werden.

Neben den Standard-Services für die Indizierung und Suche gibt es eine Reihe von Projekten, welche auf Fedora aufsetzen. Das *eSciDoc* Projekt [RSWH09] setzt auf Fedora als Repository-System auf und erweitert es um High-Level Services, die es dem Nutzer vereinfachen sollen, komplexe Anwendungen für die Publikationen von Daten zu erstellen. Neben neuen Objekttypen wird der Objekt-Lebenszyklus mit neuen Zuständen erweitert. Außerdem kann der Publikationsprozess mit Hilfe von Versionierungs- und Authentifizierungsmechanismen besser beschrieben werden.

Für Fedora existiert eine Erweiterung, die die Integration des Scientific-Workflow-Systems Kepler [LAB+06] ermöglicht. Kepler bietet die Möglichkeit, wissenschaftliche Primärdaten mit Hilfe selbst definierter Module zu verarbeiten. Ziel des *"Fedora-Kepler Integration*

demonstration projects"[4] ist es, Daten aus einem bestehenden Fedora Repository für Kepler Prozesse bereitzustellen und sie anschließend in das Repository zurück zu speichern.

DelosDLMS. Im Rahmen des EU-Projektes *"DELOS Network of Excellence"* wurde das digitale Bibliothekssystem *DelosDLMS* entwickelt [IMSS08]. Ziel der Architektur ist die Generierung modularer und erweiterbarer Bibliothekssysteme basierend auf einer serviceorientierten Architektur. Als Grundlage für das System werden die beiden Systeme OSIRIS und ISIS [BMR+07] verwendet. ISIS steht für *Interactive Similarity Search* und bietet verschiedene Funktionalitäten für Multimedia-Suche. OSIRIS steht für *Open Service Infrastructure for Reliable and Integrated process Support*. Es ist eine Middleware für die Ausführung verteilter P2P Prozesse.

Um eine Bibliotheksanwendung zu generieren, müssen alle benötigten Funktionalitäten in Services ausgelagert werden. Mit Hilfe der OSIRIS-Middleware werden diese Services sinnvoll kombiniert. Mit Hilfe des Modellierungstools *O'Grape*, kann die Abarbeitung verschiedener Services beschrieben werden. OSIRIS stellt hier ein transaktionales Modell zur Verfügung, welches Strategien für die Fehlerbehandlung bereitstellt. So kann eine korrekte parallele Abarbeitung der verteilten Prozesse ermöglicht werden. Zur Laufzeit wird anhand der Prozessdefinition ein passender Service ausgewählt. Die Auswahl kann zusätzlich durch verschiedene Kriterien wie Auslastung, Verfügbarkeit oder Funktionalität der angemeldeten Services beeinflusst werden. Standardmäßig stellt *DelosDLMS* eine breite Palette an Services bereit, die z. B. für die Feature-Extraktion, Visualisierung oder medienspezifische Indizierung eingesetzt werden können.

Fazit

Ein Trend im Bereich digitaler Bibliothekssysteme ist die Einführung serviceorientierter Architekturen. *Greenstone*, *Fedora* und *DelosDLMS* basieren darauf, dass die benötigten Aufgaben via Services, gegebenenfalls verteilt auf unterschiedliche Standorte, bereitgestellt werden. Da alle Systeme auf den Einsatz vollwertiger Workflow-Management-Systeme verzichten, ist auch die Beschreibung komplexer Publikationsprozesse schwierig umzusetzen. Bedingungen, Ressourcen und Datenabhängigkeiten in einem Modell zu modellieren und dann zu überwachen, wie es ein WFMS ermöglicht, wird nicht geboten.

Greenstone verwendet eine Plug-in-Architektur und gestattet so das System durch neue Services zu erweitern. Nachteilig ist dabei, dass die einzelnen Phasen für den Veröffentlichungsprozess nicht veränderbar sind. Lediglich die zu integrierenden Plug-ins einer Phase können variabel mit Hilfe von Konfigurationsdateien definiert werden. So kann nur eine einfache Beziehung zwischen Dokumentstruktur und -inhalt sowie Abarbeitung beschrieben werden.

[4] `https://wiki.duraspace.org/display/FEDORACREATE/Fedora-Kepler+Integration`

Fedora gestattet die Verwendung von sehr flexibel definierbaren Dokumentmodellen. Es ermöglicht, komplexe Dokumente, die in den unterschiedlichsten Gebieten Anwendung finden, zu verwalten. Mit Hilfe einer serviceorientierten Architektur kann der Kern leicht durch Funktionen erweitert werden. Der Ablauf des Veröffentlichungsprozesses ist ebenfalls stark durch den serviceorientierten Charakter vorgegeben. Durch den nachrichtenbasierten Ansatz ist eine sehr flexible Architektur entstanden, die es aber nur schwer ermöglicht, Änderungen im Ablauf durchzuführen. Außerdem kann die Abarbeitungsreihenfolge nur schwer kontrolliert werden.

eScidoc kontrolliert den Arbeitsablauf, indem jedes Dokument mit einem Zustand belegt wird. Einfache Abläufe können so dargestellt werden, wobei komplexe Publikationsprozesse nur schwer umzusetzen sind. Mit dem System *Kepler* steht zwar ein Workflow-Management-System zur Verfügung, das auf Daten in einem Fedora-Repository zurückgreifen kann, eine Steuerung des Publikationsprozesses ist aber auch hier nicht angedacht.

DelosDLMS basiert ebenfalls auf einer serviceorientierten Architektur, welche auf der Middleware *OSIRIS* aufbaut. *OSIRIS* ermöglicht eine einfache Umsetzung von Prozessen, die entsprechend der Anforderungen P2P-Services einbindet.

1.3.1.2 Institutionelle Repositorien

Neben den vorgestellten digitalen Bibliothekssystemen existieren eine Reihe von Open-Source Projekten, die für den Einsatz als einfache Dokumentenserver konzipiert sind. Anforderungen wie die Verwaltung möglichst unterschiedlicher Dokumenttypen mit einem flexiblen Dokumentmodell oder komplexe Arbeitsabläufe stehen hier nicht im Vordergrund der Entwicklung. Im Anschluss werden *MyCoRe* und *EPrints* als Vertreter vorgestellt.

MyCoRe. In [Lüt02] wird *MyCoRe* als gemeinschaftliche Entwicklung mehrerer deutscher Universitäten vorgestellt. Es basiert auf einem modularen System, das auf einem Kern und verschiedenen Beispielanwendungen aufbaut. Der *MyCoRe*-Kern fasst Funktionalitäten, die für den Betrieb eines Dokument-Management-Systems notwendig sind, zusammen. Auf Basis des *MyCoRe*-Kerns können nutzerdefinierte Anwendungen entwickelt werden. Der *MyCoRe*-Kern stellt ein flexibel definierbares Dokumentmodell bereit. Zum Kern zählen außerdem noch ein System zur Verwaltung von Klassifikationen, eine Nutzerverwaltung mit Zugriffsrechten auf Objektebene und eine anpassbare Nutzeroberfläche. Ein Event-Handler bietet die Möglichkeit, einfache Arbeitsabläufe innerhalb der Anwendung zu definieren. Konfigurationsdateien definieren dafür *Pipelines*, die z. B. die Arbeitsschritte für eine Volltextindizierung beschreiben. Außerdem wird eine rudimentäre Schnittstelle für die Integration von Workflow-Management-Systemen bereitgestellt, die als „Simple Workflow“ bezeichnet wird. Der „Simple Workflow“ basiert derzeit auf einer Menge von Java-Servlets, die die Bearbeitungsprozesse initiieren.

EPrints. Von der University of Southampton, England wird das Content-Repository EPrints entwickelt [Uni12]. Das System ist hauptsächlich für die Verwaltung von Publikationen ausgelegt, die durch Autoren selbst verwaltet werden. Dafür bietet *EPrints* eine Vielzahl von Möglichkeiten an, die den Publikationsprozess unterstützen. Zum Beispiel können Metadaten automatisch aus den Dokumenten extrahiert und aufbereitet werden. Die Arbeitsabläufe im Publikationsprozess können durch einen *Workflow* beschrieben werden. Ein Workflow besteht aus einer Konfigurationsdatei, die den Ablauf im Publikationsprozess in Form von Pfaden beschreibt. In einem Pfad beschreiben Stufen die jeweiligen Aufgaben im Arbeitsablauf. Jede Stufe beschreibt Komponenten für eine Eingabemaske auf dem Nutzerinterface. Dazu gehören z. B. Felder wie „Titel“ oder „Abstract“.

EPrints bietet auch klassische Funktionalitäten eines digitalen Bibliothekssystems wie z. B. Benachrichtigungsdienste (RSS) oder die Verwaltung von URNs. Erweitert werden kann das System z. B. mit Plug-ins. Sie bieten die Möglichkeit, Systemfunktionalitäten zu erweitern. Dieser Mechanismus wird hauptsächlich für den Im- und Export unterschiedlicher Dokumentformate verwendet.

Fazit
Die Systeme *MyCoRe* und *EPrints* bieten beide die Möglichkeit, institutionelle Repositorien umzusetzen. *MyCoRe* zeichnet sich durch den flexibel anpassbaren Softwarekern und das frei definierbare Datenmodell aus. *EPrints* bietet für Autoren eine einfache Umsetzung, eigene Publikationen zu verwalten. Dokumente können leicht bereitgestellt und teilweise automatisch mit Metadaten versehen werden. Beide Systeme stellen Komponenten zur Verfügung, die einfache Arbeitsabläufe mit Hilfe von Konfigurationsdateien kontrollieren. Komplexe Prozessmodelle können so jedoch nicht umgesetzt werden.

1.3.2 Workflow-Management

1.3.2.1 Ansätze basierend auf der Integration von Daten

Data Patterns. Russel et al. beschreiben in [REHA04, RHEA05] unterschiedliche Muster (Pattern), welche die Verwendung von Daten in einem Workflow-Management-System klassifizieren. Die Autoren haben dafür den Datenzugriff in die vier Kategorien *Sichtbarkeit von Daten*, *Daten-Interaktion*, *Datenübertragung* und *Daten-basiertes Routing* unterteilt.

Das Konzept *Sichtbarkeit von Daten* beschreibt, wie Daten definiert sind und wie sie von anderen Komponenten verwendet werden können. Muster aus der Kategorie *Daten-Interaktion* beschreiben den Datenaustausch zwischen den verschiedenen Komponenten eines WFMSs. Als Erweiterung dieser Pattern sind die Pattern in der Kategorie *Datenübertragung* zu sehen. Sie beschreiben, wie genau der Datenaustausch zwischen Workflow-Komponenten ausgeführt wird. Die Kategorie *datenbasiertes Routing* zeigt in welcher Weise Daten mit

anderen Perspektiven eines WFMSs interagieren können. Der Datenzugriff auf externe Datenquellen wird durch die Muster 15, 16 und 19, 20, in [REHA04, RHEA05], beschrieben. Muster 15 und 19 beschreiben jeweils wie Daten vom Case oder aus einer Aktivität in eine externe Quelle geschrieben werden. Umgekehrt beschreiben Pattern 16 und 20 den lesenden Zugriff auf einer externen Datenquelle für einen Case bzw. eine Aktivität.

Unabhängig vom System, kann mit Hilfe dieser Pattern die Funktionalität des Systems untersucht werden.

Data Guards. Eder und Lehmann stellen in [EL04] einen Ansatz für die Integration externer Daten in die Kontrollflussperspektive vor.

Die Integration der Datenquellen wird mit Hilfe von Plug-ins umgesetzt, welche XML-Sichten auf die Datenquellen erzeugen. Neben der Architektur für die Verwaltung von Plug-ins wurden in [EL05b] verschiedene Regeln definiert, um die Synchronisation der Daten zu beschreiben. Die Regeln (*Refresh* und *Push*) beschreiben, wann das WFMS Daten lesen oder schreiben soll. Dabei kann unterschieden werden, wann eine Datenkopie aktualisiert wird (vor jedem Lesezugriff, nur auf Anforderung, oder nie) und wann sie zurückgeschrieben werden muss (nach jeder Datenänderung, auf Anforderung, oder nie). Die Script-Sprache *WDL-X* wird eingeführt, um die verschiedenen möglichen Kombinationen der Regeln und Datenoperationen zu beschreiben.

Ein weiterer Aspekt der in [EL05a] vorgestellt wird, beschäftigt sich mit der Beziehung zwischen Kontrollfluss und externen Daten. Entscheidungen im Kontrollfluss werden häufig aufgrund von externen Daten getroffen. Ändern sich die Daten in der externen Datenquelle, kann dies dazu führen, dass Entscheidungen im Kontrollfluss rückgängig gemacht werden müssen. Das Konzept der *Data Guards* versucht das Problem zu lösen, indem bestimmte Bereiche im Prozess markiert werden, in denen Bedingungen an die Daten (aus externen Datenquellen) gestellt werden. Sobald eine Bedingung verletzt wird, wird eine Ausnahmebehandlung angestoßen.

PHILharmonic Flows. Im Projekt *PHILharmonic Flows* wird untersucht, wie eine bessere Integration von Daten in die Kontrollflussperspektive eines WFMSs umgesetzt werden kann [KR09, KR11]. Ziele sind dabei zum einen eine integrierte Sicht zwischen Daten und Prozessen zu entwickeln. Zum anderen sollen datengetriebene Prozesse ermöglicht werden.

Das Framework unterscheidet hierfür sogenannte *Micro-* und *Macro*-Prozesse. Ein *Micro*-Prozess beschreibt das Verhalten eines bestimmten Objekttyps. Hierfür stehen verschiedene Objektzustände zur Verfügung, die mit entsprechenden Transitionen verbunden sind. Zustandsübergänge sind an Attribute der Objekte geknüpft. Eine Menge von *Micro*-Prozessen wird innerhalb eines *Macro*-Prozesses zusammengefasst, der die Abarbeitung der *Mikro*-Prozesse koordiniert. Neben diesen Konzepten wird außerdem noch eine entsprechende

Rechteverwaltung eingeführt, die kontrolliert, welche Nutzer welche Daten verändern dürfen.

Proclets. Unabhängig von konkreten Datenstrukturen wird in [AMR09, MRA+10] ein Verfahren vorgestellt, welches Prozessstrukturen in Anlehnung an Datenstrukturen modelliert. Hierfür wird das Konzept der *Proclets* eingeführt. Ein Proclet ist ein Datenelement, welches mit einem Lebenszyklus erweitert wird. [ABEW00, ABEW01, AMR09]. Ein *Proclet* ist ein abgeschlossenes Prozessfragment, welches mit anderen Prozessfragmenten über sogenannte *channels* und *ports* kommunizieren kann. Jedem *port* kann zusätzlich eine Kardinalität zugeordnet werden, welche die Abhängigkeiten zwischen den zu bearbeitenden Daten ausdrücken soll. Das Konzept ermöglicht es, große Prozessstrukturen in kleine modulare Prozessbestandteile aufzuteilen, welche miteinander interagieren.

A Framework for Document-Driven Workflow Systems. In [WK05] stellen Wang und Kumar einen Framework für ein dokumentgetriebenes Workflow-System vor. Das Framework unterscheidet zwischen Dokumenten, Aktivitäten und Ressourcen. Eine Aktivität kann nur ausgeführt werden, wenn alle benötigten Dokumente und Ressourcen zur Verfügung stehen. Erst dann kann eine Instanz der Aktivität erzeugt werden. Weil kein Kontrollfluss zwischen den Aktivitäten modelliert wird, muss die Aktivierung über Ereignisse gesteuert werden. Ereignisse werden z. B. durch Dokumente erzeugt, wenn diese ihren Zustand ändern. Jeder Aktivität können Ereignisse mit Hilfe von sogenannten *Event Adaptern* zugeordnet werden. Sind alle notwendigen Ereignisse für eine Aktivität eingetroffen, wird eine Instanz erzeugt. Regeln bieten zusätzlich die Möglichkeit Bedingungen an Ereignisse zu knüpfen. Aktivitäten ändern die Eingabedokumente und erzeugen Ausgabedokumente. Eine Besonderheit dieses Ansatzes ist die vollständige Umsetzung in einem relationalen Datenbank-Management-System mit Hilfe von Triggern.

Kepler. In [LAB+06] wird das Scientific-Workflow-System *Kepler* vorgestellt. *Kepler* setzt das Akteur-basierte Modellierungskonzept (engl.*actor-oriented modeling paradigm*) um, indem Akteure als unabhängige Prozess-Komponenten modelliert sind. Akteure sind durch Ein- und Ausgabeports beschrieben, die über *channels* miteinander verbunden werden. Zusätzlich besteht die Möglichkeit, Akteure über Parameter zu konfigurieren. Die Interaktion der Akteure kann mit Hilfe unterschiedlicher Berechnungsmodelle beeinflusst werden, die *director* genannt werden. Weiterhin erlaubt Kepler die geschachtelte Modellierung von Modellen, um die Wiederverwendbarkeit der Prozessmodelle zu erhöhen.

Scientific-Workflow-Systeme werden insbesondere für die Datentransformation und Datenanalyse verwendet. Ein mögliches Anwendungsszenario besteht z. B. in der wiederholten Ausführung eines Prozessmodells mit unterschiedlichen Parametern. Ein grundlegendes Ziel, das bei Scientific-Workflow-Systemen verfolgt wird, ist die Optimierung komplexer Berechnungen, um beispielsweise deren mehrfache Ausführung zu vermeiden. Die verwalteten

Prozessinstanzen sind dabei selten völlig unabhängig voneinander, was bei Geschäftsprozessen oft der Fall ist [LWMB09].

Mit Product-Based Workflow Support (PBWS) stellen Vanderfeesten et al. in [VRA08, VRA11] einen Ansatz zur flexiblen Ausführung von Prozessen vor. Ausgangspunkt ist ein Produkt-Datenmodell (*Product Data Model* (PDM)), das durch eine baumartige Struktur beschrieben ist, dessen Knoten Datenelemente repräsentieren, die mit Kanten verbunden sind. Die Kanten stellen Operationen dar, die mehrere Datenelemente als Eingabeparameter haben können, aber nur genau ein Datenelement als Ausgabe erzeugen. Das PDM kann als Ausgangspunkt für die Generierung von Prozessstrukturen genutzt werden. In [VRA11] wird die Kombination von ProM und Declare[5] vorgeschlagen, die eine flexible Koordination der Prozesse ermöglicht, indem dem Anwender auf der Grundlage von Kostenberechnungen Vorschläge für mögliche Arbeitsschritte unterbreitet werden. Ein allgemeines Prozessmodell wird in diesem Ansatz deshalb überflüssig[VRA11].

Fazit
Der Ansatz *Data Guards* ermöglicht es, externe Datenquellen in die Kontrollfluss-Perspektive eines Workflow-Management-Systems zu integrieren. Dabei ist es nur bedingt möglich, Bedingungen an diese Daten zu überprüfen und auf Fehlersituationen zu reagieren. Mit *PHILharmonic Flows* wird ein anderer Weg beschritten, in dem die Abhängigkeit zwischen Daten und Prozessen auf Modellebene integriert wird. Daten aus externen Systemen werden dabei nicht betrachtet, können aber mit Hilfe sog. *Black-Box*-Aktivitäten prinzipiell eingebunden werden. Auch mit *Proclets* werden Datenabhängigkeiten über die Prozessstruktur abgebildet. Im Gegensatz zum *PHILharmonic Flows* Ansatz werden die Prozessmodelle aber nicht zwangsläufig hierarchisch aufgebaut. Dabei ermöglicht die Kommunikation zwischen den *Proclets* flachere Strukturen als bei *PHILharmonic Flows*. Der *PBWS*-Ansatz nutzt Abhängigkeiten zwischen Datenstrukturen, um den Nutzer Vorschläge für mögliche Prozessoperationen zu unterbreiten. Vorschläge werden dabei unter Beachtung von Leistungskriterien berechnet. Der Ansatz in [WK05] behandelt nur Daten, welche in einem relationalen Datenbank-Management-System abgelegt sind. Datenabhängigkeiten werden hier mit Hilfe von Ereignissen modelliert, die als Datenbank-Trigger umgesetzt sind. *Kepler* legt den Schwerpunkt auf die Verarbeitung von Daten, indem der Datenfluss zwischen Komponenten modelliert wird. Die Umsetzung komplexer Kontrollfluss-Muster, wie sie für Publikationsprozesse benötigt werden, ist mit *Kepler* deshalb nur bedingt umsetzbar.

1.3.2.2 Management von Prozessvarianten

Prozessvarianten ermöglichen die Modellierung verschiedener Aspekte eines Prozesses in einem Modell. Durch Konfiguration kann das Modell, abhängig von bestimmten Rahmenbe-

[5] siehe auch Abschnitt 1.3.2.2

dingungen, angepasst werden. Durch die Konfiguration wird eine Prozessvariante erzeugt. Die Komplexität und die Wartbarkeit von Prozessmodellen soll mit diesem Konzept erheblich vereinfacht werden.

C-YAWL. Gottschalk et al. [GAJVR08, GWJV+09] stellen einen generischen Ansatz vor, der Prozessmodelle mit konfigurierbaren Elementen erweitert. Die eingeführten Konzepte werden beispielhaft mit der existierenden Prozesssprache *YAWL* umgesetzt. Dafür wird die Erweiterung C-YAWL eingeführt.

Die Kernidee von C-YAWL ist die Definition von konfigurierbaren Elementen. Eine Aktivität wird hierfür mit Ports erweitert, wobei für jede eingehende Kante ein Eingangs-Port (engl. *inflow port*) und für jede ausgehende Kante ein Ausgangs-Port (engl. *outflow port*) definiert wird. Kanten stehen dabei für die Abbildung des Kontrollflusses.

Ein Port kann *aktiviert*, *geblockt* oder *versteckt* werden. Ist ein Eingangs-Port aktiviert, kann die entsprechende Aktivität durch diesen Pfad angestoßen werden. Ein geblockter Eingangs-Port verhindert zum einen die Ausführung der Aktivität, zum anderen wird der Kontrollfluss an dieser Stelle unterbrochen. Der folgende Ausführungspfad ist damit nicht mehr erreichbar (sog. *Blocking*). Im Gegensatz hierzu, ermöglicht ein versteckter Eingangs-Port die Ausführung einer Aktivität zu überspringen, aber den Kontrollfluss nicht zu unterbrechen (sog. *Hiding*). Ausgangs-Ports können nur aktiviert oder geblockt werden. Es gilt aber, dass es immer mindestens einen aktivierten Ausgangs-Port geben muss, da sonst die korrekte Abarbeitung des Prozesses nicht gewährleistet werden kann.

Die Konfiguration eines Prozessmodells wird durch ein Fragebogen-Modell ermöglicht, das verschiedene domänenspezifische Variablen mit einem Wahrheitswert belegt. Eine Variable oder auch Gruppen von Variablen werden durch eine Menge von Fragen konfiguriert [RLS+07]. In einem nächsten Schritt wird dann die Konfiguration eines Ports an eine Menge von solchen Variablen gebunden.

PROVOP. In [HBR08, Hal10] stellen Hallerbach et al. einen Ansatz vor, in dem mittels Änderungsoperationen ein Basisprozessmodell an den jeweiligen Anwendungskontext angepasst wird. Hierfür werden vordefinierte Änderungsoperationen sequentiell auf das Basisprozessmodell angewendet, um im Ergebnis ein sogenanntes Variantenmodell zu erzeugen, welches dann ausgeführt werden kann.

Um eine korrekte Konfiguration vornehmen zu können, werden die folgenden Konzepte eingeführt. Eine Änderungsoperation modifiziert ein Prozessmodell durch *Einfügen*, *Löschen*, *Verschieben* oder *Ändern* von Prozessfragmenten. Die Operation kann aber nur an explizit definierten Aufsetzpunkten im Prozessmodell angewendet werden. Ein Aufsetzpunkt beschreibt eine Position im Prozessmodell, über die Prozessfragmente bzw. Bereiche im Prozessmodell adressiert werden können. Die Auswahl der Änderungsoperation wird

entweder manuell oder automatisch anhand des Kontextes ermöglicht. *Provop* stellt sicher, dass die Reihenfolge und Kombination von Änderungsoperationen korrekt ist. Außerdem zeichnet sich der Ansatz durch eine flexible Ausführung der Prozessinstanzen aus. Es kann zur Ausführungszeit zwischen verschiedenen Prozessvarianten gewechselt werden.

1.3.2.3 Ansätze zur dynamischen Anpassung von Prozessmodellen

Nach Schonenberg et al. [SMR+08] lassen sich Ansätze, die Änderungen während der Ausführung von Prozessen ermöglichen, in zwei Kategorien unterteilen i) *Flexibilität durch Unterspezifikation* mit *nachträglichem Modellieren* (engl. *late modeling*) und *nachträglichem Einfügen* (engl. *late binding*) von Prozessfragmenten ii) *Flexibilität durch Modifikation*. Zu den Vertretern der Kategorie i) zählen *Worklets* [AHEA06a], *Declare* [PSA07] und *Pockets of Flexibility* [SOS05]. Ein Vertreter, der in die Kategorie *Flexibilität durch Modifikation* (ii) eingeordnet werden kann, ist ADEPT [RD98, RRKD05].

Worklets. In [AHEA06a] stellen Adams et al. ein Konzept für flexible Workflows auf der Basis einer serviceorientierten Implementierung vor, das das WFMS *YAWL* erweitert. Hierbei wird das Prinzip des *Late Bindings* umgesetzt. Ausgangspunkt ist ein Prozessmodell, das bestimmte Bereiche nicht vollständig ausmodelliert. Stellvertretend werden Platzhalter verwendet, die den Worklet-Service einbinden, welcher zur Laufzeit bereits modellierte Prozessfragmente (Worklets) einbindet. Ein Worklet stellt dabei ein *YAWL*-Prozess dar, der mit Hilfe des Worklet-Services aus einer Menge von Worklets ausgewählt wird. Die Auswahl kann über einen Regelbaum gesteuert werden, der sogenannte *Ripple Down Rules* verwendet. Die Regeln bilden dabei Kontextbedingungen hierarchisch ab.

Pockets of Flexibility. Das Konzept *Late Modeling* wird von Sadiq et al. [SOS05] mit *Pockets of Flexibility* umgesetzt. Dabei wird ähnlich wie bei [AHEA06a] das Prozessmodell nicht vollständig ausmodelliert. Das Konzept wird *Offene Instanz* (engl. *open instance*) genannt. Im Basisprozess werden sogenannte *Pockets* definiert, die ebenfalls einen speziellen Service aufrufen, der zur Laufzeit ein Prozessmodell zur Verfügung stellt. Im Gegensatz zu den *Worklets* wird an dieser Stelle aber ein Prozessmodell aus Aktivitäten und Regeln erzeugt. Die Auswahl der Aktivitäten wird durch den Nutzer gesteuert. Die Regeln steuern die Konstruktion der ausgewählten Aktivitäten. Zur Auswahl stehen *Verzweigung, Sequenz, Auswahl* und *Aufzählung*. Entsprechend der ausgewählten Aktivitäten und Regeln wird zur Laufzeit ein Prozessmodell generiert, welches dann ausgeführt wird.

DECLARE. Pesic et al. stellen in [PSA07, APS09] das System *Declare* vor, welches einem deklarativen Ansatz verfolgt. Es zählt zu den Vertretern des *late modeling*. *DECLARE* erlaubt es, Abhängigkeiten zwischen Aktivitäten in einem Prozess vollständig mit Hilfe von Bedingungen zu beschreiben. Zur Laufzeit wird überprüft, inwieweit die Abarbeitung der

Aktivitäten den Bedingungen genügt. *DECLARE* ermöglicht außerdem die Integration imperativer Prozessmodelle. Hierdurch können ähnliche Szenarien wie mit *Pockets of flexibility* umgesetzt werden. Beispielsweise kann *YAWL* in ein *DECLARE*-Model eingebunden werden. Auf der anderen Seite kann ein *YAWL*-Model ebenfalls *DECLARE* Modelle einbinden [AAH+09].

ADEPT. In [RD98, RRKD05] wird das WFMS ADEPT vorgestellt. ADEPT bietet die Möglichkeit, Änderungsoperationen während der Laufzeit einer Prozessinstanz auf dem Prozessmodell auszuführen. Grundlegende Operationen sind *Einfügen, Löschen* und *Überspringen* von Aktivitäten oder ganzen Prozessfragmenten. Spezielle Operationen (z. B. *Move*) ermöglichen es weiterhin, parallel ausgeführte Aktivitäten in eine sequentielle Ausführungsreihenfolge zu überführen und umgekehrt. Weiterhin wird ein Rollback-Mechanismus angeboten, der bestimmte Bereiche im Prozessmodell zurücksetzen kann. Um die Korrektheit und Konsistenz der Prozessmodelle nach der Anwendung der Operationen sicherzustellen, werden verschiedene Eigenschaften eingeführt. Ansätze, wie der von *ADEPT*, nehmen die Änderungen grundsätzlich am Prozessmodell vor. In einem nächsten Schritt wird überprüft, inwieweit die aktuelle Prozessinstanz auf das neue Prozessmodell überführt werden kann. Auch hierfür führt *ADEPT* eine Reihe von Regeln ein, welche sicherstellen, dass Änderungen wirksam werden [RRD04b].

Fazit

Ansätze wie *Provop* oder auch C-YAWL vereinfachen die Modellierung und Verwaltung komplexer Prozessmodelle. Dabei werden Basismodelle als Ausgangspunkt gewählt, die an die jeweiligen Anforderungen angepasst werden. Im Gegensatz zu C-YAWL kann der Provop-Ansatz nicht nur vollständige Modelle reduzieren, sondern auch via Änderungsoperationen neue Elemente hinzufügen. C-YAWL fehlt außerdem die Möglichkeit, zur Laufzeit einer Prozessinstanz diese entsprechend anzupassen. *Provop* ist abhängig vom verwendeten Prozess-Management-System und dessen Fähigkeiten. Da es im Publikationsprozess nicht immer möglich ist, alle Abhängigkeiten im Vorfeld zu definieren, kann eine Konfiguration des Prozessmodells nur bedingt angewendet werden. Viele Entscheidungen müssten daher weiter im Prozessmodell ausmodelliert werden.

Andere Wege werden in Ansätzen gegangen, in welchen via *late binding* oder *late modeling* Flexibilität zur Laufzeit ermöglicht wird. *Worklets* bieten nur die Möglichkeit, in Abhängigkeit von Regeln (die in Form eines Regelbaums definiert werden) ein konkretes Prozessmodell in einem vorher festgelegten Bereich im Prozess auszuwählen und auszuführen. Einen Schritt weiter geht der Ansatz von Sadiq et al. (*Pockets of Flexibility*), der es dem Nutzer ermöglicht, den unterspezifizierten Bereich im Prozessmodell selbst zu modellieren. Hierfür werden zahlreiche Aktivitäten bereitgestellt, deren Ausführungsreihenfolge durch Regeln kontrolliert wird. Die Flexibilität beider Ansätze ist insoweit eingeschränkt, als dass ein Worklet trotzdem zur Modellierungszeit definiert werden muss und somit alle möglichen

Prozessvarianten dennoch abgebildet werden müssen. Zum anderen können bei dem Ansatz der *Pockets of Flexibility* keine Kontextbedingungen beschrieben werden.

Als Vertreter für *Ad-hoc*-Prozessmodelle können mit *ADEPT* Änderungen zur Laufzeit vorgenommen werden. Hierfür wird ein Prozessmodell verändert und die dazugehörige Prozessinstanz in das neue Prozessmodell überführt. Diese Vorgehensweise ist sehr gut geeignet, um auf unvorhergesehene Ereignisse flexibel reagieren zu können. Häufige Änderungen, wie im Publikationsprozess notwendig, sind so aber nur mit hohem Aufwand umsetzbar.

1.4 Zielstellung

Die Herausforderung bei der Entwicklung einer prozessorientierten Architektur, welche den Publikationsprozess in digitalen Bibliotheken unterstützt, ist es, die flexible, datengetriebene Abarbeitung von Prozessen zu ermöglichen. Flexible, datengetriebene Abarbeitung bedeutet, dass zum einen die Dokumente aus externen Systemen geeignet in ein Workflow-Management-System eingebunden werden und zum anderen, dass die Prozesse zur Laufzeit, in Abhängigkeit von diesen Daten, angepasst werden können.

Die Forderung nach einer einheitlichen Integration von Dokumenten aus externen Datenquellen in ein WFMS setzt die Erfüllung mehrerer Bedingungen voraus. Die Daten müssen in einer einheitlichen Sprache beschrieben sein. Hier hat sich die *Extensible Markup Language* (XML) weitgehend durchgesetzt. Um eine effektive Verwaltung mehrerer unterschiedlicher Datenquellen zu ermöglichen, muss eine Architektur verwendet werden, welche die Datenquellen geeignet integriert. Die Datenquellen müssen hierfür entsprechende Interfaces für Lese- und Schreibzugriffe bereitstellen.

Neben einfachen Lese- und Schreibzugriffen muss das WFMS, unabhängig von der integrierten Datenquelle, auf Änderungen der Dokumente reagieren können. Weiterhin muss es möglich sein, geänderte Dokumente ebenfalls konsistent zurückzuschreiben. Um Inkonsistenzen zu vermeiden können Transaktionskonzepte, die im Bereich der Datenbank-Management-Systeme bereits seit vielen Jahren erfolgreich eingesetzt werden, als Grundlage für entsprechende Umsetzungen dienen. Sie bieten die Möglichkeit, mit Hilfe von Integritätsbedingungen komplexe Bedingungen an Daten zu stellen. Mit Triggern können Strategien definiert werden, um auf eventuelle Konflikte zu reagieren. Neben der Integration in das WFMS muss zusätzlich ein geeignetes Modell für die Modellierung von Prozessen bereitgestellt werden, welches ähnliche Konzepte unterstützt.

Prozessmodelle, die den Publikationsprozess abbilden, sind sehr stark abhängig von der Art der Dokumente. Sie sollen zur Laufzeit flexibel änderbar sein. So können komplexe Prozessmodelle vermieden werden, weil Abhängigkeiten nicht ausmodelliert sein müssen. Die Abhängigkeiten zwischen Dokumentstruktur und -inhalt und dem Prozessmodell müssen

daher geeignet dargestellt sein. Nicht der gesamte Prozess ist von allen Dokumentteilen gleichermaßen abhängig. Einzelne Prozessfragmente können Datenstrukturen und deren Inhalt zugeordnet werden, weil sie die Daten entweder verändern oder als Reaktion auf die Existenz von Datenstrukturen benötigt werden.

Um Prozesse zur Laufzeit ändern zu können, sind geeignete Änderungsoperationen zu definieren. Die Änderungsoperationen müssen in Abhängigkeit von Dokumentstruktur und Inhalt den Prozess verändern. Auf der einen Seite müssen Änderungen am Dokument erkannt werden. Dokumentteile können hinzugefügt, gelöscht oder geändert worden sein. Auf der anderen Seite müssen geeignete Operationen definiert werden, welche die Prozessinstanz verändern. Es sollen Operationen zur Verfügung stehen, welche es ermöglichen Aktivitäten in das Prozessmodell einzufügen oder Aktivitäten zu löschen. Das Verschieben von Aktivitäten kann durch die Kombination der genannten Operationen umgesetzt werden. Die Verbindung zwischen Dokumentänderungsoperationen und Prozessänderungsoperationen muss entsprechend beschrieben werden. Hier sind verschiedene Konstellationen möglich. Eine geänderte Dokumentstruktur kann das Hinzufügen einer oder mehrere Aktivitäten nach sich ziehen. Umgekehrt kann das aber auch zum Löschen von Aktivitäten führen. Gleiches gilt, wenn Dokumentstrukturen gelöscht werden.

1.5 Gliederung der Arbeit

Abgeleitet aus der Zielstellung ergibt sich folgende Gliederung der Arbeit.

In **Kapitel 2** werden aktuelle Techniken und Begriffe aus den Gebieten digitale Bibliotheken, Workflow-Management und Transaktionen eingeführt. Es wird ein Publikationsprozess betrachtet, für dessen Umsetzung vorhandene Systeme und Techniken aus dem Bereich Workflow-Management benutzt werden. Um die Integrität von Dokumenten aus externen Datenquellen sicherzustellen, werden transaktionale Konzepte verwendet.

In **Kapitel 3** werden anhand der Beispielanwendung „Dissertation Online“ Anforderungen an ein Workflow-Management-System untersucht, das den Autor im Publikationsprozess unterstützt.

In **Kapitel 4** wird das Lösungskonzept vorgestellt. Zunächst werden verschiedene Lösungsstrategien für die Integration von Datenquellen und für die flexible Umsetzung von Publikationsprozessen betrachtet. Anschließend wird ein Überblick über die Architektur für flexible, datengetriebene Prozessmodelle gegeben.

In **Kapitel 5** wird ein Framework für die einheitliche Integration von externen Datenquellen vorgestellt. Der Ansatz beschreibt, wie Datenquellen mit Hilfe einer Plug-in-Architektur einheitlich integriert und an externe Variablen gebunden werden.

In **Kapitel 6** wird untersucht, wie eine konsistente Sicht auf externe Datenquellen innerhalb von Prozessinstanzen sichergestellt werden kann. Zunächst werden verschiedene Ansätze geschachtelter Transaktionsmodelle betrachtet. Es wird gezeigt, mit welchen transaktionalen Konzepten die Kontrollflussperspektive von *YAWL* erweitert werden muss und welche Erweiterungen am Framework für die Datenintegration umgesetzt werden müssen. Die Konzepte aus den Kapiteln 5 und 6 basieren grundlegend auf dem von uns in [SMH11a] vorgestellten Ansatz.

In **Kapitel 7** wird ein Konzept für die flexible, datenzentrierte Ausführung von Publikationsprozessen vorgestellt. Dafür wird zunächst untersucht, wie anwendungsabhängige und dokumentabhängige Prozessbestandteile geeignet beschrieben werden können. Diese Trennung wird durch Prozessbausteine erreicht, die für die flexible Komposition von Prozessbereichen genutzt werden. Außerdem wird gezeigt, welchen Einfluss Zustand und Inhalt von Dokumenten auf die Komposition haben. Der vorgestellte Ansatz basiert grundlegend auf den von uns in [SMH11b] und [SMBH11] vorgestellten Konzepten.

In **Kapitel 8** wird ein Prototyp vorgestellt, in dem die Konzepte beispielhaft implementiert sind[6][7]. In **Kapitel 9** werden praktische Anwendungen vorgestellt. Abschnitt 9.2 basiert teilweise auf unseren in [BKSM11] und [SMBH11] vorgestellten Ergebnissen.

In **Kapitel 10** wird die vorgestellte Lösung zusammengefasst und mögliche Erweiterungen diskutiert.

[6] Marcel Paleit hat Teile der *FlexY*-Architektur implementiert

[7] Markus Bandt hat ein Plug-in für den Zugriff auf eine XML-Datenbank bereitgestellt.

Kapitel 2

Aktuelle Techniken aus der Forschung

Ein wesentliches Ziel dieser Arbeit ist die Unterstützung des Publikationsprozesses in digitalen Bibliotheken. Dazu wird ein detaillierter Überblick über Techniken und Begriffe aus den Gebieten *digitale Bibliotheken*, *Workflow-Management* und *Transaktionen* gegeben. Im ersten Abschnitt werden verschiedene Definitionen für digitale Bibliotheken vorgestellt. Nachfolgend wird ein Publikationsprozess für digitale Bibliotheken eingeführt, der den vollständigen Lebenszyklus von Dokumenten abdeckt.

Für die Umsetzung des vorgestellten Publikationsprozesses mit einer digitalen Bibliotheksanwendung wird die Integration eines Workflow-Management-Systems vorgeschlagen. Um die vorhandenen Systeme und Techniken aus dem Bereich des Workflow-Managements besser einordnen zu können, werden im zweiten Teil des Kapitels verschiedene Begriffe und Konzepte aus diesem Bereich vorgestellt. Zum einen werden verschiedene Konzepte für die Umsetzung flexibler Workflow-Prozesse vorgestellt. Zum anderen werden auch verschiedene Sichten eines Prozessmodells (Kontrollfluss- und Datenperspektive) genauer betrachtet.

Publikationsprozesse in digitalen Bibliotheksanwendungen sind dokumentzentriert. Die Dokumente werden z. B. durch Aktivitäten verändert und für Kontrollflussentscheidungen benötigt. Da die Dokumente in verschiedenen Datenquellen gespeichert werden, müssen konkurrierende Zugriffe durch das WFMS unterstützt werden. Mit Hilfe von Transaktionen kann die Integrität der Daten sichergestellt werden. In dritten Teil des Kapitels stellen wir deshalb das Konzept der Transaktionen vor.

2.1 Digitale Bibliothekssysteme und der Publikationsprozess

2.1.1 Dienste und Komponenten digitaler Bibliothekssysteme

Digitale Bibliothek. Das Konzept der digitalen Bibliothek wurde Anfang der 1990er Jahre eingeführt. Bis heute hat sich aber keine einheitliche Definition für den Begriff *digitale Bibliothek* durchgesetzt. In [FHM+01] wird gezeigt, dass der Begriff stark durch die Sichtweise des Betrachters geprägt ist. Beispielsweise sind Anwender stark auf den Inhalt fokussiert, welcher mit der digitalen Bibliothek verwaltet wird. Ein Bibliothekar setzt den Schwerpunkt dagegen auf die angebotene Dienstleistung und inwieweit damit neue Angebote für den Nutzer ermöglicht werden können. Heuer und Dittrich unterscheiden daher zwischen der *digitalen Bibliothek* als eine Sammlung von Dokumenten und der *digitalen Bibliothek* als Softwaresystem zur Verwaltung digitaler Dokumente [HD02]. Im Folgenden sollen verschiedene Definitionen vorgestellt werden, welche die unterschiedlichen Sichtweisen widerspiegeln.

Die IFLA (International Federation of Library Associations and Institutions) gibt im Rahmen des *IFLA Manifesto for Digital Libraries* [IFL10] die folgende Definition:

> "A digital library is an online collection of digital objects, of assured quality, that are created or collected and managed according to internationally accepted principles for collection development and made accessible in a coherent and sustainable manner, supported by services necessary to allow users to retrieve and exploit the resources."

Witten et al. geben in [WBN10] eine eher anwenderzentrierte Definition:

> A digital library is "[...] a focused collection of digital objects, including text, video, and audio, along with methods for access and retrieval, and for selection, and maintenance of the collection."

Fuhr et al. geben in [FTA+07] eine Definition für digitale Bibliotheken an, in der versucht wird die unterschiedlichen Betrachtungsweisen aufzugreifen:

> "A DL [digital library] is a particular kind of information system and consists of a set of components, typically a collection (or collections), a computer system offering diverse services on the collection (a technical infrastructure), people, and the environment (or usage), for which the system is built."

Heuer und Dittrich definieren in [HD02] den Begriff "Digitale Bibliothek" als:

> "eine Sammlung von Dokumenten mit bleibendem Wert, [...] deren Dokumente mit Metadaten beschrieben (erschlossen) werden, [...] die langfristig zitierbar bleiben müssen, [...] von denen es verschiedene Versionen geben kann,

> [...] wobei eine spezielle Version aber nach der Publikation nicht mehr geändert werden kann, [...] die verkauft und gekauft werden können (Dokumente kann man also "besitzen")."

Neben konkreten Definitionen erscheint es sinnvoller zu sein, einheitliche Kriterien für eine Klassifikation der unterschiedlichen Systeme und deren Funktionalitäten zu etablieren. Hierzu existieren verschiedene Modelle, die ihren Fokus aber auf verschiedene Kriterien legen.

5S Framework. Das *5S* Framework ist ein Modell, um digitale Bibliotheksanwendungen zu klassifizieren. Es wird in [GFWK04] vorgestellt. Es werden Formalismen für *Ströme* (engl. *streams*), *Strukturen* (engl. *structures*), *Räume* (engl. *spaces*), *Szenarien* (engl. *scenarios*) und *Gesellschaften* (engl. *societies*) definiert. Der Zusammenhang zwischen diesen Konzepten ist in Abbildung 2.1 dargestellt.

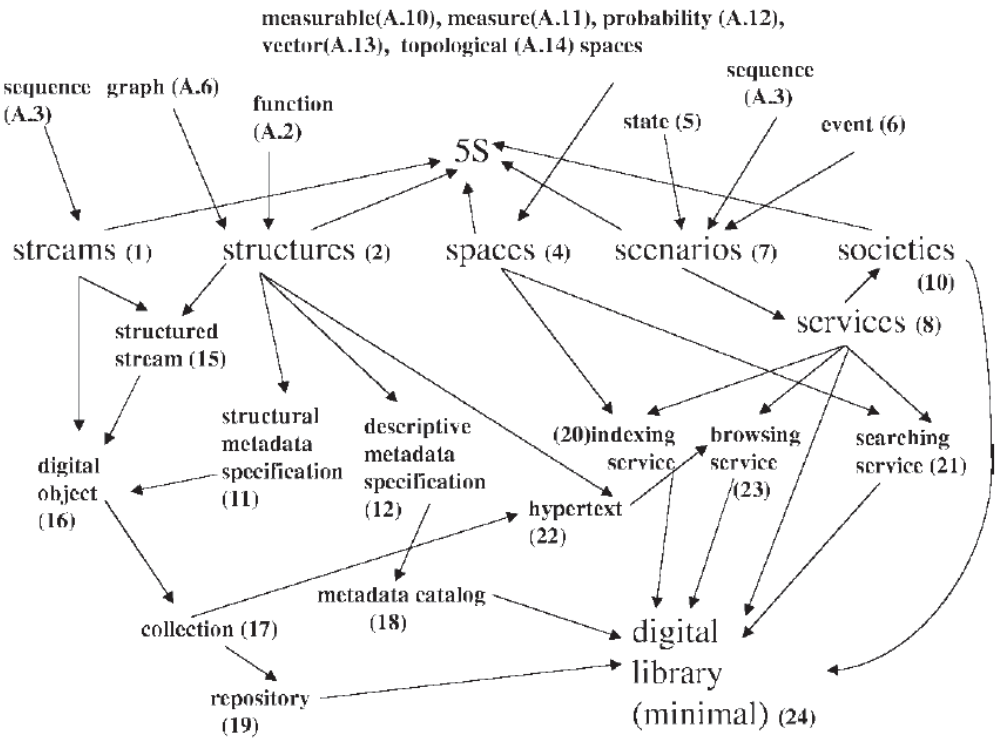


Abbildung 2.1: Konzepte des 5S Modells aus [GFWK04]

Ein (Daten-) *Strom* (engl. *stream*) stellt eine Sequenz von Elementen dar, wobei die Typen der Elemente beliebig sind. Außerdem findet noch eine Unterteilung in statische (ohne zeitlichen Aspekt) und dynamische (mit zeitlichem Aspekt) (Daten-) *Ströme* statt. *Strukturen* beschreiben, wie ein Element gegliedert ist. So definiert eine Struktur z. B. den logischen Aufbau eines Buches in Kapitel und Absätze. *Räume* (engl. *spaces*) stellen eine Kombination

von Elementen und Operationen auf diesen Elementen dar. Hierdurch unterscheiden sich Räume gegenüber *Strömen* und *Strukturen*. Ein *Szenario* beschreibt die Funktionalitäten des Systems aus der Sichtweise eines Anwenders. *Gesellschaften* beschreiben Entitäten wie Nutzer, aber auch Services oder Hardware, die im System verwendet werden. Zusätzlich werden die Beziehungen zwischen den Entitäten angegeben.

2.1.2 Multimediale Dokumente im Publikationsprozess

Digitale Bibliotheken sind einem Wandel unterzogen. Digitale Dokumente können durch immer komplexer werdende Strukturen und Datentypen beschrieben werden. Insbesondere Universitäten müssen auf diese gestiegenen Anforderungen reagieren. Für den weiteren Verlauf der Arbeit wird deshalb der Begriff des *multimedialen Dokumentes* definiert. Dafür wird der Begriff des *digitalen Dokumentes* verwendet, wie er von Endres und Fellner in [EF00] eingeführt wurde:

> "Ein digitales Dokument ist eine in sich abgeschlossene Informationseinheit, deren Inhalt digital codiert und auf einem elektronischen Datenträger gespeichert ist, so dass er mittels eines Rechners benutzt werden kann."

Die Definition von Endres und Fellner ist sehr allgemein gehalten. Schirmbacher charakterisiert ein Dokument nach seinen Bestandteilen [Sch07]. Der Inhalt, wird näher durch Text, Grafik, Bild, Bewegtbild, Audiosequenzen u. ä. gekennzeichnet. Weiterhin ist die Struktur des Dokumentes durch seine Bestandteile beschrieben. Neben der Struktur wird das Layout durch Formatinformationen angegeben. Die inhaltlichen, strukturellen und technischen Daten des Dokumentes sind mit Hilfe von Metadaten definiert.

Für die Beschreibung von digitalen Dokumenten existieren eine Reihe von verschiedenen Formaten, die die spezifischen Eigenschaften digitaler Bibliotheken unterstützen. Dazu zählen z. B. das vorrangig für den Datenaustausch verwendete *METS/MODS* Format [Lib11], welches von der Library of Congress betreut wird. Das *DoMDL* Format [CCPS05] wird in dem digitalen Bibliothekssystem *OpenDLib*[1] eingesetzt. Ein weiteres Modell ist das *Fedora Object Model* [LPSW06], welches dem gleichnamigen System *Fedora* als Basis dient. Speziell für die Beschreibung von Dissertationen im deutschsprachigen Raum wurde das Format *DiML* [Sch99] entwickelt. *DiML* ist eine Dokumenttypdefinition, welche Inhalt, Struktur und Layout für digitale Dissertationen beschreibt [Hum12]. Ohne auf die Details der einzelnen Modelle direkt einzugehen, ist an dieser Stelle anzumerken, dass diese Modelle grundsätzlich für die Verwaltung von Dokumenten in einem digitalem Bibliothekssystem gedacht sind. Komplexe Beziehungen zwischen Dokumenten, beispielsweise zeitliche und räumliche Aspekte, können jedoch nicht modelliert werden.

[1] `http://opendlib.iei.pi.cnr.it/`

Der Begriff des multimedialen Dokumentes ist im Rahmen der Arbeit nicht ausschließlich auf Modelle anwendbar, die für die Präsentation von Multimediadokumenten zum Einsatz kommen, wie z. B. *ZyX* [BK01], *STAMP* [BGVO05] oder auch *SMIL* [W3C11]. Hauptaugenmerk dieser Modelle liegt auf der Präsentation der Dokumente. Boll und Klas [BK01] führen hierfür verschiedene Perspektiven ein, welche ein solches Modell unterstützen muss. Dazu zählen zeitliche, räumliche und interaktive Modellierung. Außerdem führen die Autoren noch spezielle Anforderungen ein, welche die Wiederverwendbarkeit, Anpassbarkeit und darstellungsneutrale Beschreibung ermöglichen.

2.1.3 Der Publikationsprozess in einem digitalen Bibliothekssystem

Nach Schirmbacher [Sch07] verlagern sich die Aufgaben einer digitalen (wissenschaftlichen) Bibliothek immer deutlicher hin zu den Aufgaben Sammeln, Aufbewahrung, Erschließung und Verfügbarmachung von digitalen Dokumenten. Daneben sieht Schirmbacher die publizierende Institution (z. B. Forschungsinstitut oder Universität) in der Verantwortung, die wissenschaftliche Publikation zu veröffentlichen und zu verbreiten. Weiterhin soll sichergestellt werden, dass die Authentizität sowie die Integrität des Dokumentes gewahrt bleibt. Als letzten Punkt führt Schirmbacher die Langzeitarchivierung der Dokumente an, die durch die Institution sichergestellt sein muss. In [Sch10] definiert er den Publikationsprozess bzw. das elektronische Publizieren als “den Vorgang der Erstellung einer elektronischen Publikation“. Eine weitere Definition soll zusätzlich eine breitere Auslegung ermöglichen:

> “Unter dem elektronischen Publizieren [...] wird der gesamte Prozess der Erstellung, Verarbeitung, Speicherung und öffentlichen Bereitstellung einer elektronischen Publikation verstanden.“[Sch10]

Das elektronische Publizieren kann unter Berücksichtigung obiger Definition aber nicht nur als Folge von Schritten angesehen werden, sondern vielmehr als Kreislauf [Sch07]. Der Kreislauf beginnt beim Autor mit der Erstellung eines digitalen Dokumentes. Danach erfolgt die Publikation durch einen Verlag bzw. eine wissenschaftliche Institution. Im nächsten Schritt wird das Dokument durch eine Bibliothek bereitgestellt und somit dem Nutzer langfristig zur Verfügung gestellt. Hier schließt sich der Kreislauf, wenn der Autor als Nutzer der Bibliothek agiert. Dokumente gehen ebenfalls wieder in den Kreislauf ein, wenn sie als Teil anderer Dokumente verwendet werden oder neue Versionen von ihnen angelegt werden.

Soll ein digitales Bibliothekssystem den gesamten Publikationsprozess unterstützen, müssen neben den klassischen Funktionalitäten wie Aufbewahren, Erschließen und Verfügbarmachung besonders das Sammeln von Inhalten umfangreicher unterstützt werden. Hierfür können bekannte Techniken aus dem Bereich des Content Management (CM) eingesetzt werden. Heuer stellt daher in [HD02] fest, dass die Aufgabengebiete klassischer Content-

Management-Systeme (CMSe) und digitaler Bibliothekssysteme sich ergänzen und teilweiseüberschneiden.

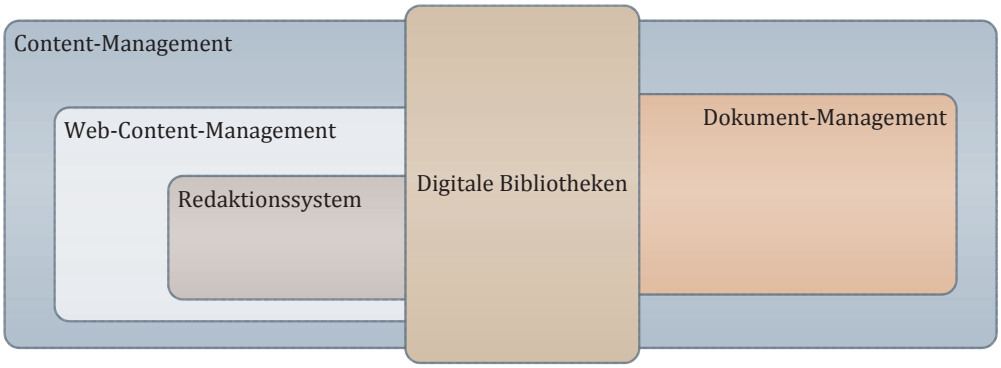


Abbildung 2.2: Einordnung digitales Bibliothekssystem nach [HD02] und [Heu09]

Die Aufgaben eines CMSs eher in der Verwaltung von kurzlebigen Inhalten wie Webseiten liegen. Dem gegenüber stehen die digitalen Bibliothekssysteme, deren Aufgabe in der langfristigen Verwaltung von Metadaten und Dokumenten liegt. Das klassische CM kann in die Bereiche *Web-Content-Management* und *Dokument-Management* unterteilt werden. Web-Content-Management-Systeme (WCMS) werden für die Verwaltung von Inhalten für Webseiten und andere internet-basierte Dienste verwendet [Rig09]. Ein WCMS wird als *Redaktionssystem* bezeichnet, wenn keine Möglichkeit angeboten wird, Informationen aus heterogenen Datenquellen zusammenzuführen und Multimediadokumente zu verwalten. *Dokument-Management-Systeme* (DMS) stellen hauptsächlich Funktionen für die Verwaltung und Vorgangsbeschreibung von Dokumenten bereit [GSSZ02].

Digitale Bibliotheken decken Funktionalitäten aus den Bereichen des *Web-Content-Management* und *Document-Mangagement* ab. Aufgrund der oben genannten Eigenschaften digitaler Bibliotheken, werden diese entsprechend Abbildung 2.2 eingeordnet[2].

Dokument-Lebenszyklus. Um das Publizieren von Multimediadokumenten in einem digitalen Bibliothekssystem vollständig zu unterstützen, haben wir in [Sch05, SMH07] einen Lebenszyklus für die Dokumente digitaler Bibliothekssysteme vorgestellt, der in Abbildung 2.3 dargestellt ist. Der beschriebene Prozess beschreibt allgemein den Lebenszyklus von multimedialen Dokumenten, wie sie oben definiert wurden. Ziel ist nicht, die jeweiligen Phasen im Zyklus detailliert zu untersetzen, da diese je nach Anwendungsgebiet und Einrichtung stark variieren können. Im Gegensatz zu Publikationsprozessen für wissenschaftliche Arbeiten, wie er beispielsweise in [Bjo05] vorgestellt wird, soll der Lebenszyklus gerade die Aufgaben einer digitalen Bibliothek mit betrachten und einschließen.

[2] In [HD02] werden die Zusammenhänge erläutert, die in [Heu09] als Grafik dargestellt werden.

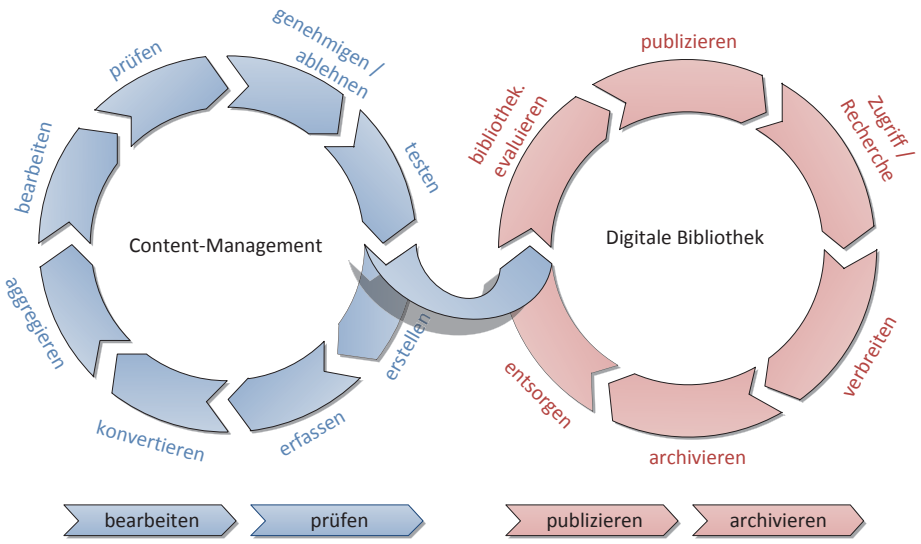


Abbildung 2.3: Lebenzyklus von Dokumenten in einer digitalen Bibliothek

Im linken Teil von Abbildung 2.3 sind die klassischen Schritte des CM beschrieben. Im rechten Teil werden die Aufgaben einer digitalen Bibliothek abgebildet.

Bearbeiten und Prüfen. Der Lebenszyklus beginnt mit dem *Erstellen* und *Erfassen* von Inhalten. Im Allgemeinen wird hier davon ausgegangen, dass Inhalte wie z. B. Bilder, Videos oder Texte mit speziellen Autorenwerkzeugen erstellt und via Schnittstelle hochgeladen werden (z. B. FTP-, HTTP-Schnittstelle oder Webformular). Dennoch besteht die Möglichkeit, Autorenwerkzeuge so zu erweitern, dass sie den erzeugten Inhalt in einer digitalen Bibliothek bereitstellen. Danach muss das Format gegebenenfalls *konvertiert* werden. Es können z. B. Richtlinien für Grafiken gelten, die zwar das Hochladen von *TIFF*-Dateien ermöglichen, aber eine dauerhafte Speicherung nur im *JPEG*-Format zulassen. In einem nächsten Schritt wird der Inhalt *aggregiert*, wobei dieser Schritt in [Sch05] in eine redaktionelle Verarbeitung, eine Segmentierung und die Anreicherung mit Metadaten unterteilt wird. Um eine bessere Wiederverwendbarkeit von Teildokumenten zu ermöglichen, wird der Inhalt segmentiert und vom Autor mit Metadaten angereichert. Im Schritt *bearbeiten* wird die Struktur für ein Multimediadokument definiert, indem Teildokumente und deren Beziehungen beschrieben werden. Je nach Anwendungsgebiet sind hier komplexe Strukturen möglich (siehe Abschnitt 2.1.2). In den Schritten *prüfen, genehmigen/ablehnen* und *testen* wird nachfolgend zum einen eine abschließende Kontrolle des Dokumentes durch den Autor vorgenommen. Zum anderen kann eine Qualitätssicherung durch ein Organ außerhalb der Bibliothek stattfinden, indem z. B. ein Gutachter- oder Lektor-Konzept, wie in [EF00] vorgestellt, angesetzt wird.

Publizieren und Archivieren. Soll die digitale Bibliothek den gesamten Lebenszyklus eines Dokumentes unterstützen, schließt sich an dieser Stelle ein weiterer Abschnitt an, der im rechten Teil von Abbildung 2.3 abgebildet ist.

Im Schritt *bibliothekarisch evaluieren* wird beispielsweise anhand von Klassifikationen entschieden, inwieweit ein Dokument mit erweiterten Metadaten versehen werden muss (durch einen Bibliotheksmitarbeiter) oder ob es in eine Archivierungsklasse eingeordnet wird. Je nach Anwendungsgebiet sind hier weitere bibliothekarische Aufgaben auszuführen. Im Schritt *publizieren* wird anschließend das Dokument für die Benutzer freigegeben. Hier sollten die Dokumente bereits indiziert sein, um einen geeigneten *Zugriff* bzw. *Recherche* zu ermöglichen. Außerdem ist es Aufgabe der Bibliothek, die Dokumente geeignet zu *verbreiten*. Nach einem vorher festgelegten Zeitraum (Archivierungsklasse des Dokumentes) wird ein Dokument *archiviert* und gegebenenfalls *entsorgt*.

2.2 Workflow-Management

In dieser Arbeit werden vorhandene Strategien für die Integration externer Datenquellen in Workflow-Management-Systeme erweitert, um die Daten innerhalb der Kontrollflussperspektive bereitzustellen. Weiterhin werden Strategien vorgestellt, die eine flexible Anpassung von Prozessinstanzen ermöglichen. Im ersten Abschnitt werden hierfür grundlegende Konzepte aus dem Bereich der Prozessmodellierung und der Prozesskomposition sowie die Prozesssprache *Yet Another Workflow Language* (*YAWL*) vorgestellt. Anschließend erfolgt die Vorstellung unterschiedlicher Muster für die Datenintegration in WFMSen. Ausgehend von der Integration verschiedenster Datenquellen soll eine flexible Ausführung von Prozessmodellen ermöglicht werden. Notwendige Arbeitsschritte und deren Abarbeitungsreihenfolge in einem Prozess können erst zur Laufzeit, in Abhängigkeit von Datenstrukturen oder Inhalten, bestimmt werden. Die unterschiedlichen Ansätze zur Modellierung flexibler Prozesse werden im dritten Abschnitt vorgestellt.

2.2.1 Prozess-Metamodell

Die Arbeitsabläufe in digitalen Bibliotheken sind durch die Publikationsprozesse vorgegeben. Diese sind je nach Anwendungsgebiet unterschiedlich. Beispielsweise haben Struktur und Inhalt der zu verwaltenden Dokumente direkten Einfluss auf die Arbeitsabläufe. Betrachtet man alle Arbeitsabläufe im Publikationsprozess als eine Einheit, kann man von einem Geschäftsprozess sprechen. Ein Geschäftsprozess (engl. *Business Process*) beschreibt ein Unternehmensziel, indem einzelne Bereiche des Geschäftsprozesses durch Prozessmodelle untersetzt werden. Ein solches Prozessmodell beschreibt die koordinierte Abarbeitung bestimmter Teilaufgaben, die durch Aktivitäten dargestellt werden. Innerhalb des Geschäftsprozess-Managements (engl. *Business Process Management* (BPM)), werden konkrete Teilaufgaben des Geschäftsprozesses automatisiert durch das Workflow-Management beschrieben und

überwacht. Häufig wird hierfür auch der Begriff Geschäftsprozess-Automatisierung (engl. *Business Process Automation* (BPA)) verwendet.

Das Workflow-Management wird eingesetzt, um Teilaufgaben eines Geschäftsprozesses IT-basiert umzusetzen und diese kontrolliert auszuführen. Workflow-Prozesse beschreiben zu diesem Zweck Arbeitsschritte und deren Abarbeitungsreihenfolge. In einem Prozessmodell werden Arbeitsschritte durch *Aktivitäten* beschrieben. Eine eindeutige Abarbeitungsreihenfolge der modellierten Aktivitäten wird durch den *Kontrollfluss* bestimmt, der *Sequenzen*, bedingte oder parallele Pfade und *Schleifen* von Aktivitäten beschreibt. Ein Prozessmodell aus dem Bereich digitale Bibliothek wird in Abbildung 3.2, Seite 57, gezeigt.

Für die Darstellung von Prozessmodellen hat sich bis jetzt kein einheitliches Prozess-Metamodell etabliert. Es existieren eine Reihe von Ansätzen, welche meist auf einem graphbasierten Konzept beruhen, in denen Aktivitäten durch Knoten und der Kontrollfluss durch Kanten zwischen Aktivitäten dargestellt wird. An dieser Stelle sollen nur einige Prozessmodelle beispielhaft aufgezählt werden: $ADEPT_{flex}$ [RD98], *Business Process Modeling Notation (BPMN)* [BPM11] und *Yet Another Workflow Language (*YAWL*)* vorgestellt in [AH05].

Zu einem *Prozessmodell* können konkrete Ausprägungen existieren – *Prozessinstanzen* – die von einem *Workflow-Management-System* (WFMS) erzeugt und verwaltet werden. Das WFMS koordiniert hierfür die Abarbeitung innerhalb einer Prozessinstanz, aber auch die gleichzeitige Verarbeitung beliebig vieler Prozessinstanzen.

Yet Another Workflow Language

Für die Präsentation von Prozessmodellen wird im Rahmen dieser Arbeit das Prozess-Metamodell *YAWL* (Yet Another Workflow Language) eingesetzt. *YAWL* ist eine Petri-Netz[3] basierte Sprache, die verschiedene Knoten- und Kantentypen für die Modellierung von Prozessmodellen bereitstellt. *YAWL* wird für die Arbeit verwendet, da es aufgrund einer komplexen Ressourcen-Verwaltung [OWK+11], einer umfangreichen Ausnahmebehandlung [AHAE07] und des formalen Modells in verschiedenen Projekten bereits erfolgreich zum Einsatz gekommen ist.

Neben dem Prozess-Metamodell *YAWL* wird im Rahmen dieser Arbeit auch das WFMS *YAWL* verwendet. Es kann Prozessmodelle der gleichnamigen Sprache ausführen. Erweiterungen, die am Prozess-Metamodell von *YAWL* vorgenommen werden, übertragen wir entsprechend auch auf das WFMS *YAWL*. Deshalb wird in den folgenden Abschnitten ein Überblick über die wesentlichen Konzepte von *YAWL* gegeben. Die formale Definition der *YAWL*-Spezifikation, die Definition eines *YAWL*-Netzes und die des Datenflussmodells von *YAWL* wird in Anhang A.2 gezeigt.

[3] Eine Einführung in Petri-Netze wird in [Rei10] gegeben.

2.2.1.1 Kontrollfluss-Perspektive

Zunächst werden die Elemente vorgestellt, die für die Modellierung innerhalb der Kontrollfluss-Perspektive gängiger graphbasierter Prozessmodelle verwendet werden können. Im Anschluss werden die vorgestellten Symbole mit einer formalen Definition untersetzt. Die Elemente werden am Beispiel des Prozess-Metamodells *YAWL* beschrieben.

Spezifikation und Netze
Eine *YAWL*-Spezifikation besteht aus einer Menge von Netzen (*YAWL*-Bezeichnung: *net*), die hierarchisch angeordnet werden können. Sie bilden zusammen einen Wurzelgraphen, dessen Wurzel ein Wurzelnetz ist. Das Wurzelnetz (*YAWL*-Bezeichnung: *top level net*) bezeichnet den Einstiegspunkt einer Spezifikation, von dem es genau einen gibt.

Ein Netz wird in *YAWL* durch eine Menge von Aktivitäten und Zuständen (*YAWL*-Bezeichnung: *condition*) beschrieben, deren Abarbeitung durch Kontrollflusskanten definiert ist. Die Elemente eines Netzes bilden so einen *gerichteten Graphen*. Da *YAWL* auf Konzepten von Petri-Netzen basiert, wird für die Abarbeitung und die Beschreibung von Zuständen einer Prozessinstanz ebenfalls das Konzept der Marken verwendet. Eine Marke beschreibt den Zustand einer Prozessinstanz und wird entsprechend der Regeln, die von Petri-Netzen her bekannt sind, durch ein Netz geschoben, um entsprechend die Aktivitäten zu aktivieren. Um komplexe Prozessmodelle mit *YAWL* modellieren zu können, führen wir eine Reihe spezieller Aktivitätstypen und Zustandstypen ein, die von *YAWL* unterstützt werden.

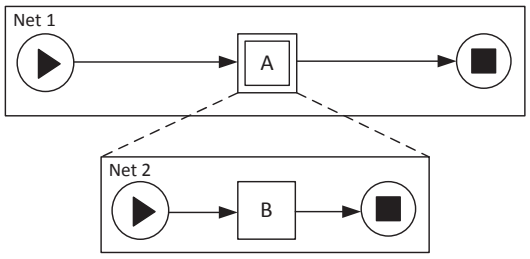


Abbildung 2.4: *YAWL*-Wurzelnetz mit Subnetz

Beispiel 2.1 (*YAWL*-Wurzelnetz)
In Abb. 2.4 ist eine Spezifikation dargestellt, die zwei Netze enthält. Das Netz mit der Bezeichnung *Net 1* ist das Wurzelnetz. Die Aktivität *A* wird außerdem in das Teilnetz *Net 2* erweitert, welches wiederum eine Aktivität *B* enthält.

Start- und Endzustand ([RH10])
In einem *YAWL*-Netz können verschiedene Zustände definiert werden, die in Abbildung 2.5 dargestellt sind. Ein einfacher Zustand, der zwischen zwei Aktivitäten eingefügt werden kann, ist in Abb. 2.5a abgebildet. Er entspricht einer *Stelle* in einem Petri-Netz [RH10].

(a) Condition (b) Input Condition (c) Output Condition

Abbildung 2.5: *YAWL* Symbole für Zustände

Zwei wichtige Zustände in jedem graphbasierten Prozessmodell sind der Start- und Endknoten. In einem *YAWL*-Netz muss genau ein definierter Anfangs- und ein definierter Endzustand festgelegt sein.

Abbildung 2.5b zeigt das *YAWL*-Symbol für den Startzustand (*YAWL*-Bezeichnung: *Input Condition*). Der Startzustand ist dadurch definiert, dass er keine eingehenden Kontrollfluss-Kanten besitzt. Demnach gibt es keine Aktivität in einem Netz, welche als Vorgänger des Startzustandes existiert.

In Abb. 2.5c wird das *YAWL*-Symbol für den Endzustand (*YAWL*-Bezeichnung: *Output Condition*) gezeigt. Der Endzustand zeichnet sich dadurch aus, dass er keine nachfolgenden Aktivitäten besitzt und demnach auch keine ausgehenden Kontrollfluss-Kanten enthält.

Um eine Instanz von einem *YAWL*-Netz zu erzeugen, muss eine Markierung in dessen Startzustand erzeugt werden. Die Netzinstanz wird beendet, wenn eine Markierung in dessen Endzustand eingefügt wird. Um eine Prozessinstanz (bzw. einen *case*) zu erzeugen, muss eine Marke in den Startzustand des Wurzelnetzes gelegt werden. Analog zu der Vorgehensweise bei Netzinstanzen wird eine Prozessinstanz beendet, wenn eine Markierung im Endzustand des Wurzelnetzes gesetzt wird.

Aktivitätstypen ([RH10])
In *YAWL* werden unterschiedliche Typen von Aktivitäten für die Modellierung von Prozessmodellen bereitgestellt. Dazu zählen atomare Aktivitäten, Aktivitäten mit mehreren Instanzen sowie zusammengesetzte Aktivitäten, die ebenfalls einen Typ mit mehreren Instanzen unterstützen (siehe Abb. 2.6).

(a) Atomic task (b) Composite task (c) Multiple instance task (d) Composite multiple instance task

Abbildung 2.6: *YAWL*-Symbole für Aktivitäten

Atomare Aktivität. Eine atomare Aktivität (*YAWL*-Bezeichnung: *atomic task*) wird durch eine Implementierung untersetzt, die im Rahmen dieser Aktivität ausgeführt wird (Abb. 2.6a).

Eine Implementierung kann z. B. die Einbindung eines Webservices oder einer Arbeitsliste (die als spezieller Service vom WFMS bereitgestellt wird) darstellen.

Neben einem Namen und einem eindeutigen Identifikator können Aktivitäten mit verschiedenen Perspektiven der Prozessmodellierung verknüpft werden. Dazu zählen z. B. die Angabe von Ressourcen (z. B. Personen oder auch Programme), die eine Aktivität ausführen sollen. Weiterhin können Daten wie Timer oder Eingabemasken für Daten, die einer Aktivität zugeordnet sind, definiert werden. In der Kontrollflussperspektive können Aktivitäten genau eine eingehende und eine ausgehende Kante zugeordnet werden. Die genannten Eigenschaften gelten für alle Aktivitätstypen, unabhängig von möglichen Erweiterungen.

Zusammengesetzte Aktivität. Im Gegensatz zur atomaren Aktivität wird einer zusammengesetzten Aktivität (*YAWL*-Bezeichnung: *composite task*) ein Netz zugeordnet (Abb. 2.6b), welches die Implementierung der Aktivität beschreibt. Das zugeordnete Netz bildet dabei eine neue Hierarchieebene unterhalb des Netzes ab, welchem die zusammengesetzte Aktivität zugeordnet ist. Aus diesem Grund darf ein Wurzelnetz nicht für die Dekomposition einer zusammengesetzten Aktivität verwendet werden. Ein Beispiel für eine zusammengesetzte Aktivität ist in Abb. 2.4 gegeben. [RH10]

Mehrfache Ausführung von Aktivitäten. Zusätzlich ermöglicht *YAWL* die mehrfache Ausführung von Aktivitäten, indem mehrere Instanzen erzeugt werden können (*YAWL*-Bezeichnung: *multiple instance task (MI)*). Für die Ausführung einer MI-Aktivität kann die Anzahl erzeugter Instanzen durch Ober- und Untergrenzen beschränkt werden. Falls die Anzahl zu erzeugender Instanzen nicht eingeschränkt werden kann, besteht die Möglichkeit, in Abhängigkeit eines Schwellwertes (der die Anzahl beendeter Instanzen angibt) die MI-Aktivität zu beenden. Abbildung 2.6c zeigt das Symbol für eine MI-Aktivität.

Neben MI-Aktivitäten können auch von einer zusammengesetzten Aktivität mehrere Instanzen erzeugt werden (*YAWL*-Bezeichnung: *Composite multiple instance task*). *YAWL* stellt hierfür ebenfalls ein entsprechendes Symbol bereit, das in Abb. 2.6d gezeigt wird.

Sequenz-Muster. Eine einfache Sequenz zwischen Aktivitäten wird in *YAWL* mit Hilfe von Kanten zwischen den Aktivitäten beschrieben. Abbildung 2.7 zeigt zwei mögliche Umsetzungen, wie das Sequenz-Muster mit *YAWL* modelliert werden kann. Die erste Möglichkeit wird in Abbildung 2.7a gezeigt. Die sequentielle Abarbeitung der Aktivitäten *A* und *B*, wird durch eine Kante mit Pfeil zwischen beiden Aktivitäten dargestellt. Eine äquivalente Modellierung wird in Abb. 2.7b dargestellt. Die sonst in einer Aktivität implizit enthaltende *condition* ist hier explizit modelliert.

Verzweigungs- und Synchronisierungsmuster ([RH10])
Um komplexe Kontrollfluss-Strukturen zu definieren, werden verschiedene Modellierungskonzepte verwendet.

(a) einfache Sequenz (b) Sequenz mit Zustand

Abbildung 2.7: Sequenz-Muster

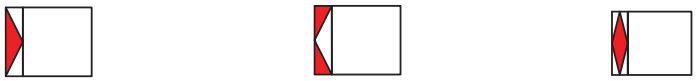
Verzweigungsmuster. In verschiedenen Situationen ist es notwendig, dass der Kontrollfluss in einem Prozessmodell in mehrere parallele Pfade aufgeteilt werden muss. Verschiedene Verzweigungsmuster beschreiben dabei unterschiedliche Effekte (siehe Abb. 2.8).

(a) AND-split (b) OR-split (c) XOR-split

Abbildung 2.8: *YAWL* Symbole für Verzweigungsmuster

Soll der Kontrollfluss in mehrere parallel laufende Pfade aufgeteilt werden, aktiviert der *AND-split* (siehe Abb. 2.8a) alle ausgehenden Kanten. Im Unterschied dazu kann der *XOR-split* (siehe Abb. 2.8c) in Abhängigkeit einer oder mehrerer Bedingungen nur einen Pfad aktivieren. Der *OR-split* (siehe Abb. 2.8b) erweitert die Funktionalität des *XOR-splits*, indem mehrere Pfade aktiviert werden können. [RH10]

Synchronisierungsmuster. Wurde der Kontrollfluss z. B. durch die Anwendung von Verzweigungsmustern in mehrere parallel laufende Pfade aufgeteilt, können Aktivitätsinstanzen mit Hilfe von Synchronisierungsmustern den Kontrollfluss wieder zusammenführen (siehe Abb. 2.9).

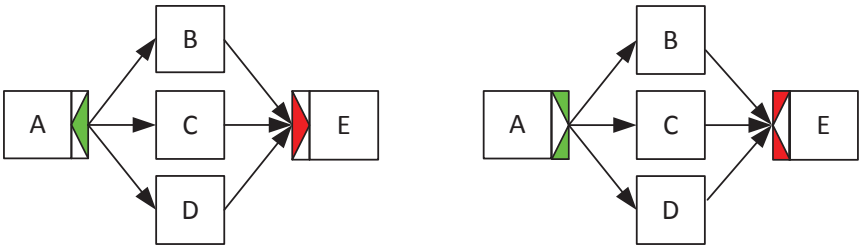


(a) AND-join (b) XOR-join (c) OR-join

Abbildung 2.9: *YAWL* Symbole für Join-Aktivitäten

Der *AND-join* (siehe Abb. 2.9a) synchronisiert alle eingehenden Kanten, wobei alle Kanten aktiviert sein müssen damit die Aktivität ausgeführt wird. Im Gegensatz dazu wird eine Aktivität mit *XOR-join* (siehe Abb. 2.9b) ausgeführt, wenn bereits eine eingehende Kante aktiviert ist.

Beispiel 2.2 (*YAWL AND-Join* und *AND-Split*)
In Abb. 2.10a wird ein Beispiel für die Anwendung der Muster des *AND-Join* und des *AND-Split* gezeigt. Aktivität *A* aktiviert alle ausgehenden Kanten und die damit verbundenen



(a) AND-Join/Split (b) XOR-Join/Split

Abbildung 2.10: AND-Join/Split und XOR-Join/Split Muster

Aktivitäten *B, C* und *D*. Die Aktivität *E* wird erst ausgeführt, wenn die Aktivitäten *B, C* und *D* abgearbeitet wurden.

Beispiel 2.3 (*YAWL XOR-Join* und *XOR-Split*)
Das in Abb. 2.10b gezeigte Muster für den *XOR-Join* und den *XOR-Split* lässt im Gegensatz zum AND-Join/Split nur die Abarbeitung einer der drei Aktivitäten *B, C* und *D* zu, nachdem *A* ausgeführt wurde. Entsprechend wird der *XOR-Join* aktiviert, wenn genau eine der Aktivitäten beendet wurde.

Der *OR-join* (siehe Abb. 2.9c) synchronisiert alle aktivierten eingehende Pfade. Deshalb muss sichergestellt sein, dass kein nicht aktivierter, eingehender Pfad aktivierbar ist. Eine detaillierte Betrachtung über die Umsetzung eines OR-Joins in *YAWL* wird in Wynn et al. [WEAH05] gegeben.

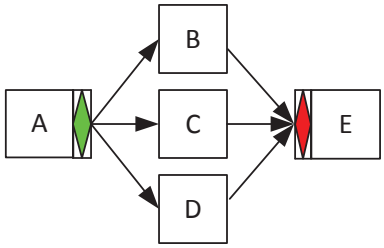


Abbildung 2.11: *YAWL* Muster für den OR-Join und den OR-Split

Beispiel 2.4 (*YAWL OR-Join*)
Der *OR-Join* wird zusammen mit dem *OR-Split* Muster in Abb. 2.11 gezeigt. Grundsätzlich ist die Anwendung ähnlich wie bei den bereits vorgestellten Mustern. Werden beispielsweise die Aktivitäten *B* und *C* durch den OR-Split in *A* aktiviert, kann *E* erst aktiviert werden, wenn *B* und *C* beendet wurden.

Abbruch-Muster ([RH10]):
Ein weiteres Konstrukt bei der Modellierung von Prozessmodellen mit *YAWL* ist die *cancelation region* (siehe Abb. 2.12).

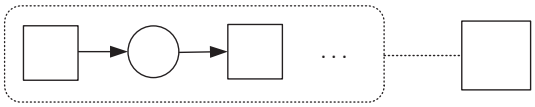


Abbildung 2.12: *YAWL* Symbol für eine *cancelation region*

Mit Hilfe der *cancelation region* können definierte Bereiche (eine Menge von Aktivitäten in einem Netz) abgebrochen werden. Zu diesem Zweck ist die cancelation region an eine andere Aktivität gebunden, deren Ausführung die Aktivierung der cancelation region anstößt.

Kontrollflussmuster:
Neben den oben vorgestellten Mustern stellen Russell et al. in [RHAM06] eine umfangreiche Sammlung von komplexen Kontrollflussmustern vor. Tabelle A.1, in Anhang A.1.1, zeigt, welche von diesen Mustern von *YAWL* unterstützt werden. Die vorgestellten Muster werden in den Kapiteln 5, 6 und 7 für die Modellierung von Prozessen verwendet.

2.2.1.2 Datenperspektive

In diesem Abschnitt werden die Elemente vorgestellt, welche *YAWL* für die Modellierung der Datenperspektive zur Verfügung stellt. Hierfür stehen die drei Datenelemente *Netzvariable, Aktivitätsvariable* und *MI-Variable* bereit. In Abbildung 2.13 wird die Verwendung der einzelnen Variablentypen an einem einfachen Beispiel veranschaulicht.

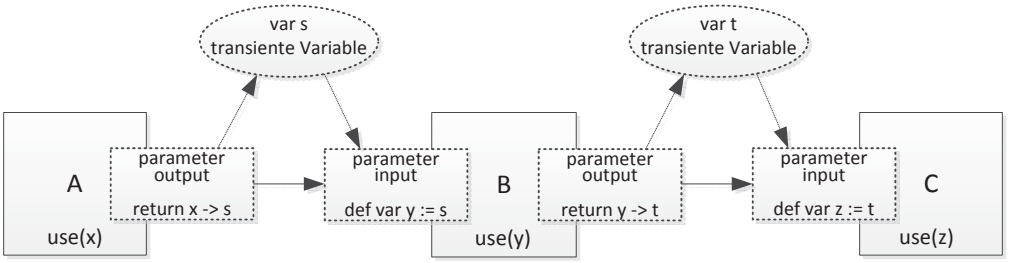


Abbildung 2.13: Datenübergabe mit Ein- und Ausgabe-Variablen nach [RH10]

Netzvariable. Eine Netzvariable ist genau einem Netz zugeordnet. Für jede Netzinstanz wird eine neue Instanz der Variable erzeugt. Die Netzvariable steht allen Aktivitäten innerhalb eines Netzes zur Verfügung. Ist das Netz in weitere Subnetze unterteilt (aufgrund einer zusammengesetzten Aktivität), ist die Sichtbarkeit der Netzvariable eingeschränkt und kann nicht direkt im Subnetz verwendet werden.

Aktivitätsvariable. Eine Aktivitätsvariable ist genau einer Aktivität zugeordnet, wobei jeder Aktivitätsinstanz eine eigene Instanz der Variablen zugeordnet wird. Sollen Daten an eine Aktivität übergeben werden, muss für die Aktivität eine Aktivitätsvariable als Eingabe-Variable definiert sein. Als Wert kann einer Variablen der Inhalt aus einer Netzvariablen oder eine

Konstante zugewiesen werden. Der Inhalt einer Netzvariablen wird mit Hilfe eines Parameters (*YAWL*-Bezeichnung: *input parameter*) übergeben, der einen XQuery-Ausdruck enthält. Diese Umsetzung entspricht der Vorgehensweise die in Muster 27 in [RHEA05] beschrieben ist (siehe Abb. 2.13). Ergebnisse einer Aktivität, die in einer Ausgabe-Variablen abgelegt ist, werden entsprechend einer Netzvariablen zugewiesen. Der Inhalt einer Aktivitätsvariablen wird mit Hilfe eines Parameters (*YAWL*-Bezeichnung: *output parameter*) an die Netzvariable übergeben, der einen XQuery-Ausdruck enthält. Diese Vorgehensweise entspricht dem Muster 28 in [RHEA05]. In *YAWL* ist es außerdem möglich, eine Aktivitätsvariable als Eingabe-Ausgabe-Variable zu definieren.

Beispiel 2.5 (Datenübergabe)
Das Prozessmodell in Abbildung 2.13 beschreibt drei Aktivitäten *A, B,* C, die sequentiell abgearbeitet werden. Für Aktivität *A* ist die Aktivitätsvariable *x*, für Aktivität *B* ist die Aktivitätsvariable *y* und für Aktivität *C* ist die Aktivitätsvariable *z* definiert. Die Variablen *s* und *t* sind als Netzvariablen festgelegt.

Für die Aktivität *A* ist die Aktivitätsvariable *x* als Ausgabe-Variable definiert, welche auf die Netzvariable *s* kopiert wird. Die Aktivitätsvariable von Aktivität *B* ist als Ein- und Ausgabe-Variable definiert. Der Wert von *s* wird entsprechend an die Eingabe-Variable *y* übergeben. Nachdem die Aktivität *B* erfolgreich abgearbeitet wurde, wird der Wert von *y* in die Netzvariable *t* geschrieben. Um die Aktivität *C* auszuführen, wird der Wert aus der Netzvariablen *t* in die Eingabe-Variable *z* geschrieben.

MI-Variable. Ist eine Aktivität als MI-Aktivität definiert, muss der Datenaustausch mit Netzvariablen aufwendiger beschrieben werden. Abbildung 2.13 verdeutlicht die Vorgehensweise, die in zwei Schritte unterteilt ist. Im ersten Schritt wird ein Parameter für den

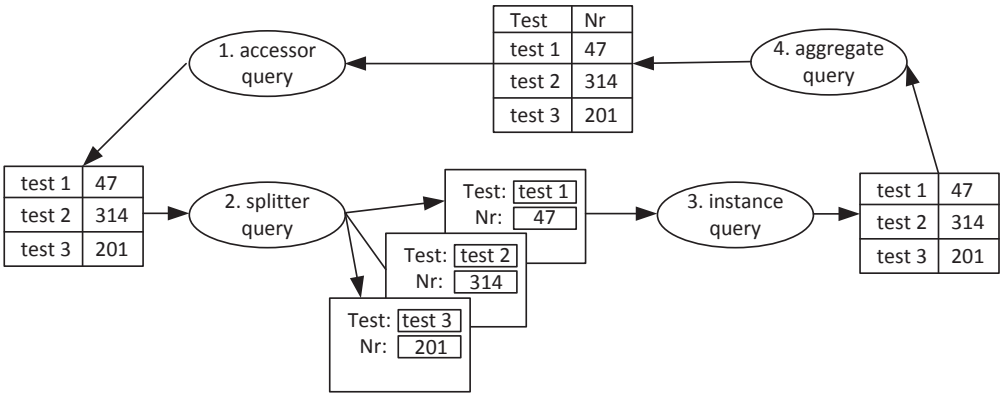


Abbildung 2.14: Datenübergabe bei MI-Aktivitäten nach [RH10]

Zugriff auf eine Netzvariable definiert, die mehrere Datensätze enthält (*YAWL*-Bezeichnung: *accessor query*). Die einzelnen Datensätze werden in einem zweiten Schritt so aufgeteilt, dass je ein Datensatz an eine Instanz der MI-Aktivität übergeben wird (*YAWL*-Bezeichnung:

splitter query). Für die Ausgabe muss im dritten Schritt ein Parameter definiert sein (*YAWL*-Bezeichnung: *instance query*), welcher die einzelnen Datensätze wieder zu einem Datensatz zusammenführt. Im vierten Schritt wird dieser Datensatz wieder mit Hilfe eines Parameters einer Netzvariable zugewiesen (*YAWL*-Bezeichnung: *aggregate query*). Die MI-Variable ist genau einer MI-Aktivität zugeordnet, wobei auch jeder MI-Aktivitätsinstanz eine eigene Instanz der Variable zugeordnet wird.

Kontrollflussentscheidungen
Neben dem Datenaustausch können Netzvariablen auch für Kontrollflussentscheidungen verwendet werden. Aktivitäten, die als *XOR*- oder *OR-Split* definiert sind, bieten die Möglichkeit, Bedingungen an ausgehende Kanten zu definieren. Eine Bedingung wird als XPath-Ausdruck[4] beschrieben und aktiviert einen ausgehenden Pfad, wenn dieser Ausdruck zu *true* evaluiert wird. Wird der Ausdruck zu *false* evaluiert, darf die Kante nicht aktiviert werden. Die beiden Split-Typen unterscheiden sich dahingehend, dass der XOR-Split nur den ersten und keinen anderen Pfad auswählt, der zu *true* evaluiert wird. Im Gegensatz dazu aktiviert der OR-Split alle mit *true* evaluierten Pfade.

Von den Datenmustern, die in [RHEA05] vorgestellt werden, unterstützt *YAWL* 13 Muster direkt. Tabelle A.2, in Anhang A.1.2, listet diese Muster auf. In Kapitel 3 werden wir zeigen, dass die unterstützten Muster für die Modellierung von Publikationsprozessen nicht ausreichend sind.

2.2.2 Grundlagen und Diskussion der Datenperspektive

Ein wesentliches Merkmal von Geschäftsprozessen ist die Verarbeitung von Daten. Nach Weske ([Wes07]) versetzt die explizite Verwendung von Datentypen und die Beschreibung von Datenabhängigkeiten zwischen Aktivitäten eines Prozesses ein Workflow-Management-System erst in die Lage, die Abarbeitung von Daten zu kontrollieren.

Daten und ihre Sichtbarkeit
Eine der bekanntesten Klassifikationen für Daten im Kontext von WFM wurde 1999 von der Workflow Management Coalition (WfMC) vorgestellt [WfM99]. Ziel der WfMC ist es, Daten nach ihrer Verwendung durch das WFMS einzuordnen (siehe Abb. 2.15). Das WFMS muss für die Abarbeitung der Prozessinstanzen die notwendigen Daten – *Workflow-Kontrolldaten* – verwalten. Hierzu zählen z. B. Informationen über Statusinformationen von Aktivitätsinstanzen, aber auch welche Ressourcen gerade welche Prozessinstanzen bearbeiten.

Daneben gibt es Daten – *Anwendungsdaten* – auf die das WFMS keinen Zugriff hat. Diese Daten werden vollständig durch die Anwendung verwaltet. Dies können z. B. komplex strukturierte Dokumente sein.

[4] Eine Einführung in XML, XPath und XQuery wird in [KM03] gegeben.

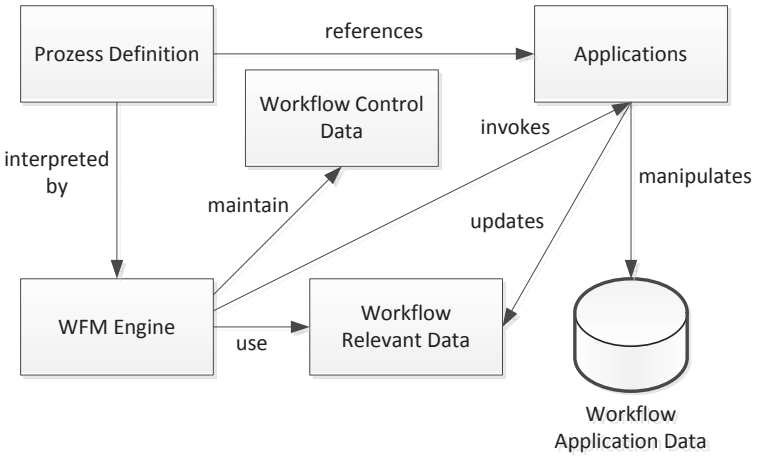


Abbildung 2.15: Einteilung von Daten in einem WFMS nach der WfMC (nach [WfM99] geändert)

In bestimmten Situationen werden Anwendungsdaten für die Abarbeitung einer Prozessinstanz benötigt. Diese Daten – *Workflow-relevante Daten* – werden dann durch beide Systeme (WFMS und Anwendung) benutzt und verändert. Das WFMS kann Daten von einer Workflow-Anwendung anfordern und an eine andere Workflow-Anwendung weiter geben. Workflow-Anwendungen (Anwendungen, die durch eine Aktivität aufgerufen werden) können so außerdem ihre Daten dem WFMS z. B. für den Kontrollfluss bereitstellen. Hierfür müssen die Daten im internen Speicher des WFMSs abgelegt werden.

Die Unterteilung der WfMC sollte nicht absolut betrachtet werden. Ausgehend von der Annahme, dass Daten auch außerhalb des Kontrollbereichs des WFMS geändert werden können, muss hier eine andere Herangehensweise entwickelt werden.

In [SOSF04] werden daher erweiterte Anforderungen an ein Datenmodell definiert. Die workflow-relevanten Daten werden dazu in 4 Untertypen unterteilt:

- Referenzierte Daten: zur Identifikation der WF-Instanzen
- Operationale Daten: werden für eine Aktivität benötigt
- Entscheidungsdaten: werden für den Kontrollfluss benötigt und sind eine Teilmenge von den operationalen Daten
- Kontextdaten: komplex strukturierte Ein- oder Ausgabe-Daten von Aktivitäten, die für die weitere Abarbeitung wichtig sind

Neben dieser detaillierteren Klassifizierung der Daten wird außerdem eine Unterteilung der Quelle einer Variablen vorgenommen. Da nicht alle Daten vom WFMS direkt bereitgestellt werden können, müssen diese aus anderen Quellen integriert werden. Daher ist es sinnvoll, die Variablen einer Aktivität in *intern* und *extern* zu unterteilen [SOSF04]. Der Begriff der workflow-relevanten Daten wird also nochmals in weitere Unterbegriffe aufgeteilt.

Datenquellen

Interne Daten werden vom WFMS verwaltet und im internen Speicher gehalten. Je nach verwendetem Datenflussmodell können die Daten von Aktivitäten gelesen und geschrieben werden. Da die internen Daten auch initialisiert werden müssen, werden sie mindestens zu einem Zeitpunkt mit Daten aus einer externen Quelle belegt.[SOSF04]

Externe Daten, die in externen Datenquellen verwaltet werden, sind zwar nicht intern, aber sie sind für eine Aktivität verfügbar. Das bedeutet, eine Aktivität kann auf Daten in externen Datenquellen zugreifen bzw. dort auch Daten erzeugen, diese werden aber nicht zwangsläufig dem WFMS bereitgestellt. Es besteht aber auch die Möglichkeit, externe Daten an das WFMS zu übergeben, die damit zu internen Daten werden. [SOSF04]

Daten-Muster

Eine detaillierte Betrachtung, wie Daten in einem WFMS verwaltet werden und wie sie gegebenenfalls aus externen Quellen integriert werden, wird in [REHA04, RHEA05] vorgestellt. Russell et al. beschreiben mit Hilfe von Mustern (*Data Patterns*), wie die verschiedenen Perspektiven umgesetzt werden können. Neben Mustern, die den Datenfluss beschreiben (Data Visibility, Data Transfer Patterns, Internal Data Interaction und Data-based Routing), zeigen die Muster 14 bis 25 in [REHA04] Konzepte für den Datenaustausch mit externen Quellen. An dieser Stelle sollen nur die Muster 14 bis 17 in [REHA04] näher betrachtet werden (siehe Abb. 2.16).

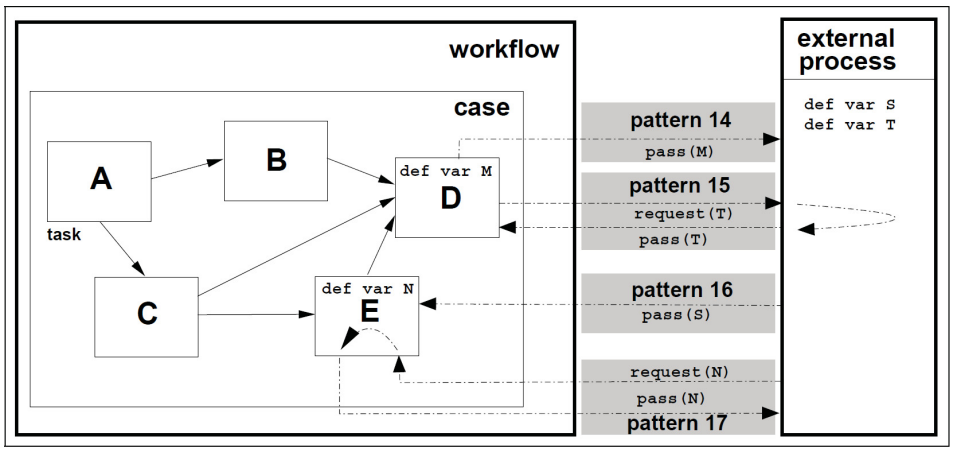


Abbildung 2.16: Data Pattern 14 und 17 ([REHA04])

Muster 14 beschreibt, wie der Wert der Variable *M* der Aktivität *D* an eine externe Quelle übergeben wird. Die entgegengesetzte Richtung beschreibt Muster 15 (das Lesen von einem Wert aus einer externen Quelle). Der Wert der externen Variable *T* wird gelesen.

Die Muster 16 und 17 beschreiben den gleichen Effekt mit dem Unterschied, dass die Aktion von der externen Datenquelle initiiert wird. Muster 16 beschreibt, wie der Wert von der

externen Variable *S* an die Aktivität *E* übergeben wird, und Muster 17, wie die externe Datenquelle den Wert der Variable *N* von Aktivität *E* anfordert.

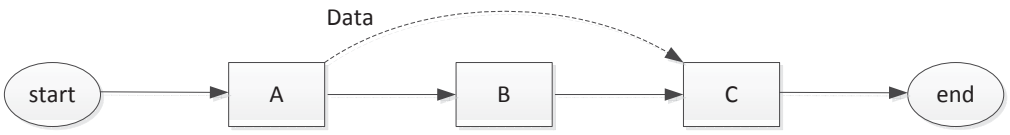
Datenintegration
Von besonderem Interesse bei datenzentrierten Prozessen ist die realisierte Methode, wie die externen Daten integriert werden. Auf der einen Seite müssen Informationen über die Datenquelle verwaltet werden. Auf der anderen Seite entsteht ein klassisches Integrationsproblem [Wes07]. Heterogene Datenquellen erfordern beispielsweise ein Mapping zwischen den unterschiedlichen Datenquellen.

Ein einfacher Ansatz, einen Datensatz aus einer externen Quelle während der Abarbeitung einer Aktivität zu verwenden, besteht darin, eine Referenz auf den Datensatz im internen Speicher zu halten. Diese Referenz wird an die Aktivität übergeben, die wiederum ein Programm ausführen kann, welches die Daten selbst aufruft. Diese Methode hat den Nachteil, dass die eigentlichen Daten nicht unter der Kontrolle des WFMS stehen und deshalb auch nicht persistent im internen Speicher verwaltet werden können. Ein weiterer großer Nachteil besteht darin, dass die Daten nicht in Kontrollflussbedingungen verwendet werden können, weil sie nicht im internen Speicher verwaltet werden.

Um diesen Nachteil ausgleichen zu können, bieten die meisten aktuellen WFMSe kleine Services bzw. ToolAgents an, die von einer automatischen Aktivität aufgerufen werden können. Diese Services können Daten aus externen Quellen lesen und in den internen Speicher überführen. Umgekehrt können Daten aus dem internen Speicher so auch an eine externe Datenquelle übergeben werden. Ein Nachteil dieser Lösung ergibt sich bei der Ausführung der Services. Eine Aktivität kann immer nur eine atomare Aufgabe ausführen. Sollen Daten also über diesen Mechanismus integriert werden, muss der Kontrollfluss mit einer Vielzahl von Aktivitäten erweitert werden, die dann nur für den Datenzugriff verwendet werden. Die Komplexität des modellierten Prozessmodells steigt stark an, worunter das Verständnis des Prozessmodells leidet und die Wartbarkeit schwieriger wird. Weiterhin besteht ein programmiertechnischer Aufwand, um den Datenaustausch mit Hilfe dieser Services umzusetzen. Systeme wie YAWL oder jBPM umgehen das Problem, indem die Skripte vor und nach dem Ausführen von Aktivitäten aufgerufen werden. Dadurch können die externen Datenquellen zumindest unabhängig von der eigentlichen Aktivität synchronisiert werden. Die Integration erfordert aber einen hohen programmiertechnischen Aufwand, da keine standardisierten Schnittstellen existieren. Jede Datenquelle muss einzeln eingebunden werden.

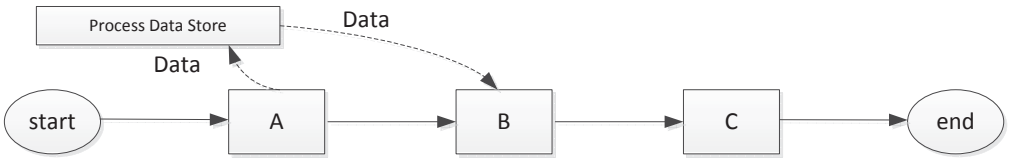
Datenfluss
Der Datenfluss beschreibt, wie Daten zwischen den Elementen eines Prozessmodells ausgetauscht werden. Im Allgemeinen hat eine Aktivität Eingabe- und Ausgabe-Variablen. So können Daten an die Implementierung der Aktivität übergeben werden bzw. die Implemen-

tierung kann Daten für den Prozess bereitstellen. Die integrierte Applikation kann z. B. ein Subnetz sein oder ein Service sein. Strategien für die Verwaltung interner Daten unterteilt [SOSF04] in drei Modelle:



(a) Explizite Integration von Daten

(b) Implizite Integration von Daten



(c) Implizite Integration von Daten über einen globalen Prozessspeicher

Abbildung 2.17: Strategien für die Verwaltung interner Daten

- die *explizite Integration* ermöglicht die Modellierung von Datenflusskanten zwischen Aktivitäten (siehe Abb. 2.17a).
- die *implizite Integration über den Kontrollfluss* übergibt Daten von einer Aktivität zur nachfolgenden Aktivität unter Ausnutzung von Kontrollflusskanten (siehe Abb. 2.17b).
- die *implizite Integration über einen globalen Prozessspeicher* ermöglicht es Aktivitäten, ihre Daten dort abzulegen. Gleichzeitig können Eingabe-Variablen mit Werten aus diesem Speicher belegt werden (siehe Abb. 2.17c). Diese Variante wird z. B. von *YAWL* unterstützt.

Probleme

Eine häufig gemachte Annahme ist, dass Daten welche in einem WFMS verwendet werden immer exklusiv von einer WF-Instanz verwendet werden. Dadurch können Seiteneffekte, die durch Manipulation interner und externer Daten entstehen, unberücksichtigt bleiben [EL05a].

Wenn Änderungen an operationalen Daten beachtet werden sollen, hat dies je nach Verwendung der Daten Auswirkungen auf WF-Instanzen. Die geänderten Daten beeinflussen die Instanzen dann auf verschiedene Weise. Wird z. B. in einem Autorendatensatz der Nachname geändert, sind folgende Umsetzungen möglich:

- Laufende Instanzen können die neuen Werte verwenden (Bsp.: Autorensatz wird nur

für die Kommunikation mit dem Autor verwendet.)
- Laufende Instanzen benötigen die alten Werte (Bsp.: Der Nachname des Autors muss bei Namensänderung für vorhandene Veröffentlichung gleich bleiben.)
- Die Änderungen haben keinen Effekt (Bsp.: Die Daten werden nicht mehr benötigt.)

Werden die Daten zusätzlich für den Kontrollfluss verwendet (Entscheidungsdaten), kann dies weitreichende Folgen für die Abarbeitung der WF-Instanzen haben ([EL05a]). Änderungen führen dazu, dass bestimmte Teile der Prozessinstanz nicht mehr gültig sind und verursachen Ausnahmen, die entsprechend behandelt werden müssen. Eine Möglichkeit, dem entgegenzuwirken, ist die Anwendung von Integritätsbedingungen auf externen Daten. Diese lassen sich aber nur schwer überprüfen und erfordern zumindest die Überführung der externen Daten in den internen Speicher. Dann können beispielsweise Vor- oder Nachbedingungen an die Daten gestellt werden, wenn diese von einer Aktivität verwendet werden.

Fazit
Die vorgestellten Konzepte für die Integration von Daten bieten verschiedene Möglichkeiten, die je nach Anwendungsfall Vor- und Nachteile mit sich bringen. Im Bereich digitale Bibliotheken werden Konzepte für die Integration externer Datenquellen für Publikationsprozesse benötigt. Dokumente sind in unterschiedlichen Datenquellen gespeichert, die konsistent in die Daten- und Kontrollflussperspektive der Prozessmodelle integriert werden müssen. Wir greifen deshalb in Abschnitt 4.1.1 die hier vorgestellten Konzepte noch einmal auf und diskutieren, ob sie für die Umsetzung von Publikationsprozessen genutzt werden können.

2.2.3 Grundlagen und Diskussion flexibler Prozessmodelle

Prozesse können in unterschiedlichen Situationen ausgeführt werden, die verschiedene Reaktionen erfordern. In Abhängigkeit von workflow-relevanten Daten muss beispielsweise ein Prozesspfad ausgewählt werden. Immer wichtiger wird die flexible Behandlung von Ausnahmen während der Ausführung einer Prozessinstanz. Ist es darüber hinaus nicht möglich, eine Ausnahme im Vorfeld zu erkennen, muss die Prozessinstanz flexibel änderbar sein. Gerade bei der Arbeit mit dokumentzentrierten Prozessen ist eine flexible Anpassung von Prozessinstanzen erforderlich. Dokumentstruktur und -inhalt können sich häufig ändern, was eine Anpassung der datenabhängigen Prozessbereiche nach sich ziehen kann. Deshalb sollen in diesem Abschnitt aktuelle Techniken aus dem Bereich flexibler Prozessmodelle vorgestellt werden.

In der Literatur gibt es eine Reihe von Arbeiten, welche die Flexibilität von Prozessmodellen in Abhängigkeit der unterstützten Techniken einordnen. In [WRRM08] stellen Weber et al. Muster vor, die Änderungen an Prozessen beschreiben. Dabei wurden 14 Muster gefunden, welche ein Prozessmodell direkt ändern und 4 Pattern, die vordefinierte Bereiche zur Laufzeit verändern. Die formale Semantik dieser Änderungsmuster beschreiben Rinderle-Ma et al. in

[RMRW08]. Auf dieser Arbeit aufbauend, haben Weber et al. in [WSR09] eine umfassende Klassifizierung von Ansätzen im Bereich des Workflow-Management vorgenommen.

In [SMR+08] haben Schonenberg et al. eine weitere Klassifizierung von Techniken aus dem Bereich flexibler Prozessmodelle vorgestellt, die wir im Weiteren auch verwenden. Flexibilität wird dabei in folgende Kategorien unterteilt:

- Flexibilität durch Modellierung
- Flexibilität durch Abweichung
- Flexibilität durch Unterspezifikation
- Flexibilität durch Modifikation

2.2.3.1 Flexibilität durch Modellierung

Eine einfache Methode, mit einem Prozessmodell flexibel auf unterschiedliche Situationen während der Abarbeitung zu reagieren, besteht darin, unterschiedliche Ausführungspfade einzuführen. In Abhängigkeit von workflow-relevanten Daten wird dabei zur Laufzeit der Pfad ausgewählt, welcher für die aktuelle Situation am besten eignet ist.

Definition 2.1 (Flexibilität durch Modellierung)
"Flexibility by Design is the ability to incorporate alternative execution paths within a process model at design time allowing the selection of the most appropriate execution path to be made at runtime for each process instance."[SMR+08]

Für die Umsetzung der Flexibilität durch Modellierung stellen die meisten Prozess-Metamodelle unterschiedliche Konstrukte bereit. Dazu zählen beispielsweise *parallele Ausführung*, die *Auswahl*, *wiederholte Ausführung* aber auch die Möglichkeit, *Aktivitäten abbrechen* zu können (eine vollständige Aufzählung wird in [SMR+08] gegeben).

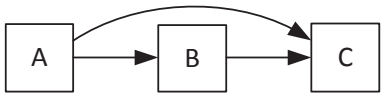


Abbildung 2.18: Flexibilität durch Modellierung: Auswahl-Operator [SMR+08]

Beispiel 2.6 (Flexibilität durch Modellierung)
Ein Beispiel wird in Abb. 2.18 gezeigt. Das abgebildete Prozessmodell enthält 3 Aktivitäten, wobei in *A* eine *Auswahl* zwischen *B* und *C* erfolgt. So kann die Abarbeitung von *B* ausgelassen werden.

2.2.3.2 Flexibilität durch Abweichung

In manchen Situationen kann eine Prozessinstanz nicht genau so abgearbeitet werden, wie es im Prozessmodell vorgegeben ist. Das ist beispielsweise der Fall, wenn in einer Ausnahmesituation die Ausführung einer Aktivität ausgelassen werden muss. Dafür führen manche Systeme Operationen ein, die es ermöglichen, vom aktuellen Prozessmodell abzuweichen, indem eine entsprechende Operation ausgeführt wird. Die Änderungen sollen hier aber grundsätzlich keine Änderung am zugrundeliegenden Prozessmodell hervorrufen. Genauer gesagt, sollen alle anderen und neuen Instanzen, die dem Prozessmodell zugeordnet sind, weiterhin unverändert ablaufen.

Definition 2.2 (Flexibilität durch Abweichung)
"Flexibility by Deviation is the ability for a process instance to deviate at runtime from the execution path prescribed by the original process without altering its process model. The deviation can only encompass changes to the execution sequence of tasks in the process for a specific process instance, it does not allow for changes in the process model or the tasks that it comprises."[SMR+08]

Für die Umsetzung dieser Art von Flexibilität werden in [SMR+08] fünf Konzepte vorgestellt, die hier nur kurz beschrieben werden sollen. Die *Undo task A* soll die Möglichkeit bieten, den Prozess in den Zustand genau vor der Ausführung von *A* zu versetzen. *Redo task A* soll das erneute Ausführen von A ermöglichen, falls diese gerade beendet wurde. *Skip task A* ermöglicht das Überspringen von A. Daneben bieten manche Modelle auch die Möglichkeit mehrere Instanzen einer Aktivität zu erzeugen (siehe MI-Aktivität). Als letztes Konzept wird die *invoke task A* eingeführt. Damit können einzelne Aktivitäten unabhängig vom Kontrollfluss ausgeführt werden, um dann mit der normalen Abarbeitung fortzufahren.

(a) vor der Skip-Operation (b) nach der Skip-Operation

Abbildung 2.19: Flexibilität durch Abweichung: Skip-Operator [SMR+08]

Beispiel 2.7 (Flexibilität durch Abweichung)
Ein Beispiel in dem der Skip-Operator angewendet wird ist in Abb. 2.19 gezeigt. Die Linke Seite (Abb. 2.19a) stellt den Zustand einer Prozessinstanz dar, nachdem *B* aktiviert wurde. Auf der rechten Seite (Abb. 2.19b) wird der Zustand nach der Anwendung des Skip-Operators gezeigt. *B* wurde in diesem Fall nicht ausgeführt.

2.2.3.3 Flexibilität durch Unterspezifikation

Für bestimmte Bereiche in einem Prozessmodell kann bereits während der Modellierung entschieden werden, ob in bestimmten Bereichen eine Auswahl zwischen verschiedenen Prozesspfaden erfolgen muss. Eine Verschärfung dieses Problems tritt auf, wenn erst während der Laufzeit einer Prozessinstanz die Auswahl und gegebenenfalls auch erst die Definition eines Prozesspfades erfolgen kann. Eine mögliche Lösungsstrategie ist die Unterspezifikation des Prozessmodells.

Definition 2.3 (Flexibilität durch Unterspezifikation)
"Flexibility by Underspecification is the ability to execute an incomplete process model at run-time, i.e., one which does not contain sufficient information to allow it to be executed to completion. Note that this type of flexibility does not require the model to be changed at run-time, instead the model needs to be completed by providing a concrete realisation for the undefined parts."[SMR+08]

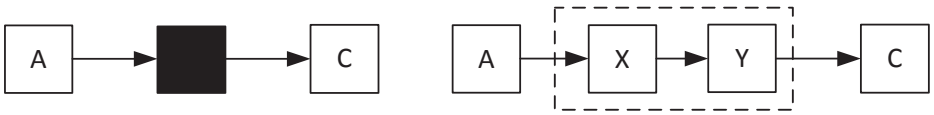
Flexibilität durch Unterspezifikation wird in einem Modell durch die Angabe von Platzhaltern ermöglicht. Ein Platzhalter stellt den nicht weiter spezifizierten Bereich in einem Modell dar. Trotzdem muss das Modell wohlgeformt und valide sein. Für die Umsetzung wird in der Literatur hauptsächlich zwischen zwei Verfahren unterschieden: *late binding* und *late modeling*.

Late binding. Das Konzept des *late bindings* ist von Programmiersprachen her bekannt. Zur Laufzeit wird ein Prozessfragment aus einer vorher festgelegten Menge ausgewählt und ausgeführt. Charakteristisch für dieses Verfahren ist die Einschränkung auf vorhandene Prozessfragmente, die nicht erst zur Laufzeit erzeugt werden dürfen. Die Prozessfragmente müssen außerdem den strukturellen und semantischen Anforderungen des Platzhalters genügen [WSR09]. Ein- und Ausgabedaten müssen z. B. korrekt verarbeitet werden können.

Late modeling. Als Erweiterung zum *late binding* kann beim *late modeling* zur Laufzeit ein Prozessfragment erzeugt werden, um es anstelle des Platzhalters auszuführen. Die Konstruktion der Prozessfragmente wird dabei häufig durch Bedingungen eingeschränkt.

Für das *late binding* und das *late modeling* können zwei unterschiedliche Zeitpunkte definiert werden, wann die Umsetzung des Platzhalters erfolgt [SMR+08]. Entweder wird der Platzhalter umgesetzt, wenn die Prozessinstanz erzeugt wird, vor der Aktivierung des Platzhalters oder während der Ausführung des Platzhalters.

Neben der Fragestellung wann, und wie ein Prozessfragment für Platzhalter eingebunden wird ist es außerdem wichtig zu entscheiden, wie oft dies getan wird. In [SMR+08] wird daher noch zwischen einer *statischen Umsetzung*, die nur einmal beim ersten Aufruf durchgeführt wird, und einer *dynamischen Umsetzung*, die bei jedem Aufruf des Platzhalters durchgeführt wird, unterschieden.



(a) vor der Umsetzung (b) nach der Umsetzung

Abbildung 2.20: Flexibilität durch Unterspezifikation: Platzhalter (nach [SMR+08])

Beispiel 2.8 (Flexibilität durch Unterspezifikation)
Die Anwendung eines Platzhalters wird in Abb. 2.20 gezeigt. Abbildung 2.20a zeigt den Zustand bevor der Platzhalter umgesetzt worden ist. In Abb. 2.20b wurde ein Fragment mit den Aktivitäten *X* und *Y* anstelle des Platzhalters eingefügt.

2.2.3.4 Flexibilität durch Modifikation

In bestimmten Situationen können Ausnahmebehandlungen nicht mit kurzzeitigen Abweichungen vom Prozessmodell gelöst werden. Außerdem ist es nicht immer möglich, die Ausnahmen schon im Vorfeld zu bestimmen. In solchen Situationen ist es häufig sinnvoll, ein Prozessmodell permanent zu verändern, indem beispielsweise Aktivitäten gelöscht oder hinzugefügt werden. Abhängig von den Gegebenheiten können sich diese Änderungen auf einen Teil der Prozessinstanzen oder auf alle Prozessinstanzen eines Prozessmodells beziehen. Das Konzept der Modifikation wird z. B. von den Systemen ADEPT2 [RRKD05] oder WASA2 [VW99, Wes00] angewendet.

Definition 2.4 (Flexibilität durch Modifikation)
"Flexibility by Change is the ability to modify a process model at run-time such that one or all of the currently executing process instances are migrated to a new process model. [...] [The] model constructed at design time is modified and one or more instances need to be transferred from the old to the new model." [SMR+08]

Auswirkung der Änderung
Es wird unterschieden, ob eine Änderung nur auf existierende Instanzen Auswirkung hat (*vorübergehende Änderung*) oder, ob die Änderung permanent am Prozessmodell vorgenommen wird (*evolutionäre Änderung*). Im letzten Fall hat die Änderung Einfluss auf alle neuen Prozessinstanzen eines Prozessmodells.

Änderungszeitpunkt
Änderungen können weiterhin zu unterschiedlichen Zeitpunkten durchgeführt werden. In [SMR+08] wird daher eine Unterscheidung in *Eintrittszeit* (engl. *entry time*) und *während der Abarbeitung* (engl. *on-the-fly*) vorgenommen.

Eintrittszeit. Änderungen, die genau in dem Moment durchgeführt werden, wenn eine Instanz erzeugt wird. Alle *vorübergehenden Änderungen* an einer Instanz haben dann nur Auswirkungen auf diese Instanz. *Evolutionäre Änderungen* hingegen werden auf alle neuen, aber auf keine existierenden Instanzen angewendet.

On-the-fly. Änderungen können zu jedem Zeitpunkt vorgenommen werden. Alle *vorübergehenden Änderungen* an Instanzen haben ebenfalls nur Auswirkungen auf diese Instanz. *Evolutionäre Änderungen* werden hingegen auf allen existierenden und neuen Instanzen angewendet. Ein Ansatz, wie diese Änderungen unterstützt werde können, wird z. B. in [RRD03] vorgestellt.

Migrationsstrategien

Wenn Änderungen an Instanzen vorgenommen werden, gibt es unterschiedliche Strategien, wie diese umgesetzt werden können. Beim *forward recovery* werden alle betroffenen Instanzen abgebrochen. Das *backward recovery* bricht ebenfalls alle Instanzen ab, führt gegebenenfalls eine kompensierende Aktion aus, und startet danach die Instanzen neu. Das Übergehen bzw. Ignorieren von Änderungen wird durch die Strategie *proceed* beschrieben. Als noch komplexere Methode kann die *transfer* Strategie betrachtet werden. Hier wird jede Instanz in einen Zustand des neuen Modells überführt.

(a) vor der Umsetzung (b) nach der Umsetzung

Abbildung 2.21: Flexibilität durch Modifikation: Löschen [SMR+08]

Beispiel 2.9 (Flexibilität durch Modifikation)
In Abb. 2.21 wird die Anwendung einer Löschoperation auf eine Prozessinstanz gezeigt. Abbildung 2.21a zeigt den Zustand vor dem der Modifikation durch den Operator. In Abbildung 2.21b ist das Ergebnis nach dem Löschen zu sehen. Aktivität *B* wurde aus dem Kontrollfluss entfernt.

Fazit

Die vorgestellten Flexibilisierungskonzepte bieten unterschiedliche Möglichkeiten, die je nach Anwendungsfall Vor- und Nachteile mit sich bringen. Im Bereich digitaler Bibliotheken werden Konzepte für die Flexibilisierung von Prozessmodellen für den Publikationsprozess benötigt. Prozessmodelle sind dort stark datenzentriert und müssen zur Laufzeit flexibel auf Zustandsänderungen dieser Daten angepasst werden. Wir greifen deshalb in Abschnitt 4.1.2 die hier vorgestellten Konzepte noch einmal auf und diskutieren, ob sie für die Umsetzung von Publikationsprozessen genutzt werden können.

2.3 Transaktionen

Die Verwaltung konkurrierender Datenzugriffe ist eine Kernkompetenz von Datenbanksystemen. Datenbanksysteme führen hierfür das Prinzip der Transaktion ein. Transaktionen sichern die semantische sowie operationale Integrität der Daten in einem Datenbanksystem zu. Eine Transaktion stellt dabei eine Folge von Operationen dar, die unter Einhaltung des ACID-Prinzips die Daten von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt. [Gra81, HR83]

- **A**tomicity (Atomarität) beschreibt die Eigenschaft einer Transaktion, dass sie entweder ganz oder gar nicht ausgeführt wird. Muss eine Transaktion abgebrochen werden, so ist besonders wichtig, dass alle Zwischenergebnisse der Transaktion zurückgesetzt werden.
- **C**onsistency (Konsistenz) legt fest, dass die Daten vor und nach dem Ausführen einer Transaktion die Integritätsbedingungen erfüllen müssen. Dies gilt ebenfalls, wenn eine Transaktion abgebrochen wurde.
- **I**solation (Isolation) ermöglicht das Ausführen von nebenläufigen Transaktionen, ohne dass diese sich gegenseitig beeinflussen. Der Nutzer eines Systems soll insbesondere den Eindruck gewinnen, dass er alleine auf den Daten arbeitet.
- **D**urability (Dauerhaftigkeit) beschreibt die Anforderung, dass Daten nach Abschluss einer Transaktion dauerhaft gespeichert sein müssen.

Für die weiteren Betrachtungen in diesem Kapitel sollen im Folgenden Konzepte aus dem Bereich Transaktionen vorgestellt werden. Dazu wird der Begriff der Transaktionen weiter eingegrenzt und definiert. Außerdem wird ein kurzer Überblick über das Konzept der Serialisierbarkeit gegeben. Die folgende Definition ist aus [WV01] entnommen und orientiert sich an dem Page-Modell. Grundlegend ist eine Transaktion eine Folge von Aktionen der Form $r(x)$ oder $w(x)$.

$$t = p_1 \dots p_n \tag{2.1}$$

wobei $n < \infty$, $p_i \in \{r(x), w(x)\}$ mit $1 \leq i \leq n$. Das Element x ist dabei in der Menge $D = \{x, y, z, \dots\}$ enthalten, die eine Menge von Datenelementen beschreibt. Jedem dieser Datenelemente können unteilbare Lese- und Schreiboperationen zugeordnet werden. Mit r wird eine Leseoperation und mit w eine Schreiboperation angegeben. Um eine einzelne Transaktion zu beschreiben, wird folgende Definition verwendet:

$$\begin{aligned} p_j &= r(x) : \text{Schritt } j \text{ liest das Datenelement } x \\ p_j &= w(x) : \text{Schritt } j \text{ schreibt das Datenelement } x \end{aligned} \tag{2.2}$$

Die Interpretation einzelner Transaktionsschritte (siehe Gleichung 2.2) ist nicht möglich, weil keine weiteren Informationen darüber bekannt sind. Deshalb wird versucht, die syntaktische Interpretation so generell und einfach wie möglich zu gestalten.

- Ist der j-te Schritt $p_j = r(x)$ einer Transaktion eine Leseoperation, dann wird der aktuelle Wert von x einer lokalen Variable v_j zugewiesen:

$$v_j := x \tag{2.3}$$

- Wenn der j-te Schritt $p_j = w(x)$ einer Transaktion eine Schreiboperation ist, dann wird ein eventuell neu berechneter Wert in x geschrieben. Grundsätzlich ist ein Wert, welcher in einer Transaktion geschrieben wird, nicht unabhängig von allen anderen Daten, die innerhalb der gleichen Transaktion vorher gelesen wurden. Der Wert von x ist abhängig vom Rückgabewert einer Funktion f_j, die aber unbekannt ist. Dies wird formal folgendermaßen ausgedrückt:

$$x := f_j(v_{j_1}, \ldots, v_{j_k}) \tag{2.4}$$

  mit:

$$\{j_1, \ldots, j_k\} = \{j_r | p_{j_r} \text{ ist eine Leseoperation } \wedge j_r < j\} \tag{2.5}$$

  Als Parameter für die Funktion f_j werden alle Variablenwerte $v_{j_r} \leq r \leq k$ verwendet, die vor dem j-ten Transaktionsschritt ausgeführt wurden.

Beispiel 2.10 (Transaktion)
Als Beispiel wird folgende Transaktion diskutiert:

$$t = r(x)r(y)r(z)w(u)w(x)$$

Die Transaktion besteht aus den 5 Schritten $p_1 = r(x)$, $p_2 = r(y)$, $p_3 = r(z)$, $p_4 = w(u)$ und $p_5 = w(x)$. Die ersten drei Schritte weisen entsprechend die Werte von x, y und z den Variablen v_1, v_2 und v_3 zu. Die Werte für u und x ergeben sich aus $u = f_4(v_1, v_2, v_3)$ und aus $x = f_5(v_1, v_2, v_3)$

Bisher wurde angenommen, dass eine Transaktion aus einer sequentiellen Anordnung von Transaktionsschritten besteht. Dieses Kriterium soll nun abgeschwächt werden, um beispielsweise eine parallele Bearbeitung unterschiedlicher Transaktionen zu erlauben.

Definition 2.5 (Transaktion nach [WV01])
Eine Transaktion t ist eine partielle Ordnung von Aktionen, der Form $r(x)$ oder $w(x)$, mit $x \in D$, in der mehrfache Lese- und Schreiboperationen auf demselben Datenelement geordnet sind. Formal gilt:

$$t = (op, <)$$

mit op ist eine endliche Menge von Operationen der Form $r(x)$ oder $w(x)$, mit $x \in D$. Eine partielle Ordnung auf der Menge op wird durch $< \subseteq op \times op$ beschrieben, für die gilt: Wenn $\{p,q\} \subseteq op$ und beide Operationen auf dem gleichen Datenelement arbeiten und mindestens eine der beiden Operationen eine Schreiboperation ist, dann muss $p < q \vee q < p$ gelten.

Entsprechend Definition 2.5 wird also eine Ordnung auf Transaktionsschritten gefordert, wenn a) eine Lese- und Schreiboperation auf demselben Datenelement ausgeführt wird und b) wenn zwei Schreiboperationen auf demselben Datenelement arbeiten.

Weil davon auszugehen ist, dass eine lokale Variable in einem Programm nach dem erstmaligen Lesen immer zur Verfügung steht, können Redundanzen innerhalb einer Transaktion vermieden werden, indem

- innerhalb einer Transaktion dasselbe Datenelement nur einmal gelesen oder geschrieben wird,
- und kein Datenelement (nochmal) gelesen wird, nachdem es geschrieben wurde.

Serialisierbarkeit

In modernen Anwendungen müssen Transaktionen parallel voneinander ausgeführt werden. Führt man eine Menge von Transaktionen $T = \{t_1, \ldots, t_n\}$ ohne weitere Vorkehrungen parallel aus, kann dies zu Inkonsistenzen zwischen Lese- und Schreiboperationen unterschiedlicher Transaktionen auf den gleichen Datenelementen führen. Bekannte Beispiele hierfür sind die Phänomene *lost-update*, *inconsistent-read* und *dirty-read*.

- *lost-update:* Eine zweite Transaktion t_2 überschreibt durch die Operation $w_2(x)$ den Wert $w_1(x)$ einer parallel laufenden Transaktion t_1, welche diesen vorher bereits geschrieben hat.
- *inconsistent-read:* Eine Transaktion t_1 berechnet beispielsweise die Summe über einer Menge von Datenelementen, während eine Transaktion t_2 eine Teilmenge der Datenelemente ändert.
- *dirty-read:* Eine Transaktion t_1 schreibt einen Wert $w_1(x)$, der von Transaktion t_2 mit $r_2(x)$ gelesen wird. Danach bricht Transaktion t_1 unerwartet ab, wobei der gelesene Wert in t_2 ungültig wird.

Um den Begriff der Serialisierbarkeit genauer zu definieren, werden im Folgenden verschiedene Begriffe und Notationen eingeführt. Die *History* ist die Vereinigung aller Transaktionsschritte der beteiligten Transaktionen. Als *Schedule* wird ein Ablaufplan aus einer Folge von Transaktionsschritten bezeichnet. Ein *Schedule* ist ein Präfix einer *History*[5]. In einem Datenbanksystem ist der *Scheduler* für die korrekte verschränkte Ausführung von

[5] History wird teilweise auch als *vollständiger Schedule* bezeichnet.

unterschiedlichen Transaktionen zuständig. Der *Scheduler* mischt die Transaktionsschritte derart, dass die lokale Reihenfolge nicht verändert wird. Um eine verschränkte Ausführung von Transaktionsschritten zu erzeugen, wird die Funktion $shuffle(T)$ eingeführt. Sie enthält alle verschränkten Ausführungen aller Einzelschritte aus allen in der Menge T enthaltenden Transaktionen T_i. Ein *Schedule* s ist also definiert als $s' \in shuffle(T)$ [WV01]. Um einen vollständigen *Schedule* zu erhalten, müssen die Operationen c_i oder a_i hinzugefügt werden.

Definition 2.6 (Schedules und Histories nach [WV01])
$T = \{t_1, \ldots, t_n\}$ mit $t_i \in T, t_i = (op_i, <_i)$ ist eine Menge von Transaktionen. op_i ist die Menge von Operationen aus t_i mit deren Ordnung $<_i$ $(1 \leq i \leq n)$.

1. Eine *History* für T ist das Paar $s = (op(s), <_s)$ mit:
   - $op(s) \subseteq \bigcup_{i=1}^{n} op_i \cup \bigcup_{i=1}^{n} \{a_i, c_i\}$ und $\bigcup_{i=1}^{n} op_i \subseteq op(s)$
     Die History s besteht aus der Vereinigung aller Operationen der beteiligten Transaktionen, zusammen mit einer Terminierungsoperation c_i (commit) oder a_i (abort) für jede Transaktion $t_i \in T$.
   - $(\forall i, 1 \leq i \leq n)\ c_i \in op(s) \iff a_i \notin op(s)$
     Für jede Transaktion gibt es entweder eine *commit* oder ein *abort*, aber nicht beides.
   - $\bigcup_{i=1}^{n} <_i \subseteq <_s$
     Alle Ordnungen von Transaktionen sind in der partiellen Ordnung enthalten, die durch s gegeben ist.
   - $\forall i, 1 \leq i \leq n)(\forall\ p \in op_i)\ p <_s a_i$ oder $p <_s c_i$
     Der letzte Schritt einer Transaktion ist immer ein *commit* oder ein *abort*.
   - Jedes Paar von Operationen $p, q \in op(s)$ aus zwei unterschiedlichen Transaktionen, die auf das gleiche Datenelement zugreifen und mindestens eine Schreiboperation ausführen, sind in s wie folgt geordnet: $p <_s q$ oder $q <_s p$
2. Ein *Schedule* ist ein Präfix einer *History*.

Um entscheiden zu können, wann eine Folge von Transaktionsschritten unterschiedlicher Transaktionen korrekt ausgeführt werden kann, wird das Konzept der Serialisierbarkeit eingeführt. Zwei *Schedules* s und s' sind Äquivalent ($s \approx s'$), wenn sie die gleichen Operationen ($op(s) = op(s')$) enthalten und wenn der Effekt von s gleich dem Effekt von s' ist. Um den Begriff der Äquivalenz genauer zu definieren, werden in der Literatur verschiedene Verfahren vorgeschlagen. Häufig beschriebene Verfahren sind beispielsweise *Sichtenserialisierbarkeit* (siehe [WV01, S. 82]) oder die *Konfliktserialisierbarkeit* (siehe [WV01, S. 92]).

Fazit

In Abschnitt 2.2.2 wurden grundlegende Verfahren vorgestellt, wie Daten in einem WFMS verwaltet werden. Externe Datenquellen werden häufig in Form von workflow-relevanten Daten verwaltet. Die redundante Datenhaltung kann zu inkonsistenten Datenzuständen

führen, wenn z. B. eine Anwendung die Daten in der externen Datenquelle ändert. Deshalb führen wird in Kapitel 6 ein Transaktionsmodell ein, das einen konsistenten Datenzugriff auf externe Datenquellen ermöglicht. Das Transaktionsmodell ist eine Erweiterung, die in das WFMS *YAWL* integriert wird.

Transaktionen und Workflows wurden in verschiedenen Systemen zusammen eingesetzt. In Abhängigkeit vom Anwendungsfall unterscheiden sich die Systeme darin, wie die Kopplung beider Ansätze vorgenommen wurde. Grefen stellt in [Gre02] eine Taxonomie vor, die die Ansätze in sechs Klassen einordnet. Dabei werden die zwei Hauptklassen *separate Modelle* und *integriertes Modell* unterschieden. *Separate Modelle* werden unterteilt in:

- Workflows über Transaktionen (*WF/TR*): Die Transaktionsmodelle erweitern Workflowmodelle semantisch.
- Transaktionen über Workflows (*TR/WF*): Die Workflowmodelle erweitern Transaktionsmodelle mit Prozessstrukturen.
- Transaktionen und Workflows gleichrangig (*TR+WF*): Die beiden Modelle können als Submodelle von einem Prozessmodell betrachtet werden.

Integriertes Modelle werden unterteilt in:

- Hybride transaktionale Workflowmodelle (*TRWF*): Ein einziges hybrides Modell wird verwendet, das sowohl Transaktions- als auch Workflow-Konzepte enthält.
- Transaktionen in Workflows (*WF*): Ein einzelnes Workflowmodell, in dem transaktionale Aspekte auf Workflow-Primitive abgebildet werden.
- Workflows in Transaktionen (*TR*): Ein Transaktionsmodell, in dem Workflow-Aspekte in Transaktion-Primitive abgebildet werden.

Das Transaktionsmodell, welches in Kapitel 6 vorgestellt wird, kann in die Klasse *TR+WF* eingeordnet werden.

Kapitel 3

Anforderungsanalyse

Publikationsprozesse stellen besondere Anforderungen an die Prozessunterstützung in digitalen Bibliotheksanwendungen. Im ersten Abschnitt führen wir deshalb den Publikationsprozess für digitale Bibliotheksanwendungen am Beispiel „Dissertation Online“ ein. Daraus ergeben sich unterschiedliche Anforderungen an die Datenintegration, die in Abschnitt 3.2 diskutiert werden. Neben der Datenintegration werden die Anforderungen an die Flexibilität von Prozessmodellen in Abschnitt 3.3 betrachtet.

3.1 Prozessunterstützung für den Publikationsprozess von Dissertationen

Digitale Bibliotheksanwendungen müssen zunehmend den gesamten Lebenszyklus digitaler Multimediadokumente abbilden. Wir betrachten deshalb die Prozessunterstützung in digitalen Bibliotheksanwendungen in Abschnitt 3.1.1. Danach wird der Publikationsprozess „Dissertation Online“ in Abschnitt 3.1.2 vorgestellt.

3.1.1 Prozessunterstützung in digitalen Bibliotheksanwendungen

Der Lebenszyklus von Dokumenten in digitalen Bibliotheken beschreibt das Erstellen komplex strukturierter Multimediadokumente (siehe Abschnitt 2.1.2). Die Autorenunterstützung für das Erstellen dieser Dokumente ist allerdings in aktuellen Systemen nicht ausreichend. Es fehlt an geeigneten Methoden, Prozessmodelle zu beschreiben, welche den komplexen Anforderungen des Publikationsprozesses gerecht werden.

Eine wichtige Rolle spielt die Integration von Daten bei der Prozessmodellierung. Aktuelle Prozess-Metamodelle bieten zwar einen besonders großen Funktionsumfang bei der

Modellierung des Kontrollflusses, die Datenflussperspektive wird hingegen nur sehr wenig unterstützt. Eine Modellierung von Datenzugriffen auf externe Datenquellen ist gegenwärtig nicht möglich. Hier müssen meist eigene Erweiterungen für den Datenzugriff bereitgestellt werden. Insbesondere bei der Verwendung von unterschiedlichen multimedialen Dokumenten (Audio, 2D/3D, Video, Still-Image, Volltext, ...), die in verschiedenen Medien-Servern gespeichert werden, ist aber eine Modellierung von besonderem Interesse.

In einem digitalen Bibliothekssystem erfordert die Verwaltung komplexer Multimediadokumente die Integration diverser inhaltsabhängiger Operationen. Es muss z. B. für jeden unterstützten Datentyp eine entsprechende Methode für die inhaltsbasierte Suche bereitgestellt werden. Unterschiedliche Datentypen benötigen spezielle Techniken für die Indizierung. Daraus lässt sich eine direkte Beziehung zwischen Dokumenten bzw. Dokumentteilen und Prozess-Aktivitäten ableiten. Um der dynamischen Struktur der Multimediadokumente gerecht zu werden, müssen die Prozessmodelle daher flexibel auf deren Inhalt und Struktur anpassbar sein.

3.1.2 Der Publikationsprozess am Beispiel *Dissertation Online*

In diesem Abschnitt stellen wir ein Dokumentmodell und ein Prozessmodell für die elektronische Publikation von Dissertationen vor. Aufbauend auf diesem Beispiel, werden in den folgenden Abschnitten die verschiedenen Anforderungen erläutert.

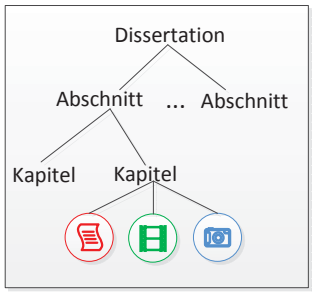


Abbildung 3.1: Ein vereinfachtes Dokumentmodell für Dissertationen

Dokumentmodell *DiML*

Für die interne Verwaltung einer Dissertation wird ein komplexes, logisches Dokumentmodell benötigt. *DiML* ist ein mögliches Format zur Beschreibung von Dissertationen. In Anhang B.2 wird ein *DiML* Dokument für eine Dissertation gezeigt. Das Dokument verwendet nicht alle möglichen Elemente, beschreibt aber wesentliche Elemente des Formats. Es dient als Grundlage für die folgenden Beispiele. Ein *DiML*-Dokument ist immer in die drei Bereiche *front, body* und *back* unterteilt. Im Abschnitt *front* sind Metadaten wie Angaben über den Autor, Gutachter und Kurzfassung angegeben. Der Abschnitt *body* beschreibt das Dokument mit seiner Kapitelstruktur, seinen Abschnitten, Paragraphen und Medienelementen. Der letzte

Abschnitt in einem *DiML*-Dokument (*back*) enthält z. B. Verzeichnisse (Literaturverzeichnis, Abkürzungsverzeichnis, etc.), Selbständigkeitserklärung und Anhänge.

Abbildung 3.1 zeigt eine stark vereinfachte, graphische Darstellung der Dokumentstruktur, die mit dem *DiML*-Format beschrieben wird. Die Abbildung soll im Folgenden verwendet werden, um Zusammenhänge zwischen Prozessmodell und Dokument zu verdeutlichen. Zur Vereinfachung wird davon ausgegangen, dass eine Dissertation mehrere Kapitel enthält. Von einer Unterteilung der Kapitel in Abschnitte, Paragraphen etc. wird hier abstrahiert, um die Komplexität der Beispiele möglichst gering zu halten. Wichtig für die Anwendung der vorgestellten Konzepte ist, ob ein Kapitel bzw. ein Dokument bestimmte Dokumenttypen enthält. Die verschiedenen Typen (hier Text, Bild und Video) werden durch die unterschiedlichen Piktogramme repräsentiert.

Prozessmodell

Abbildung 3.2 zeigt den Geschäftsprozess für den Publikationsprozess *Dissertation Online* [Deu11]. Das Prozessmodell ist in Anlehnung an die Vorgaben durch *Dissertation Online* modelliert, ohne auf implementationsabhängige Fragestellungen, wie beispielsweise Konzepte für die Integration externer Datenquellen, genauer einzugehen. Ein Lösungsvorschlag aus [Deu11] für einen Geschäftsprozess, wird in Anhang B.1 vorgestellt.

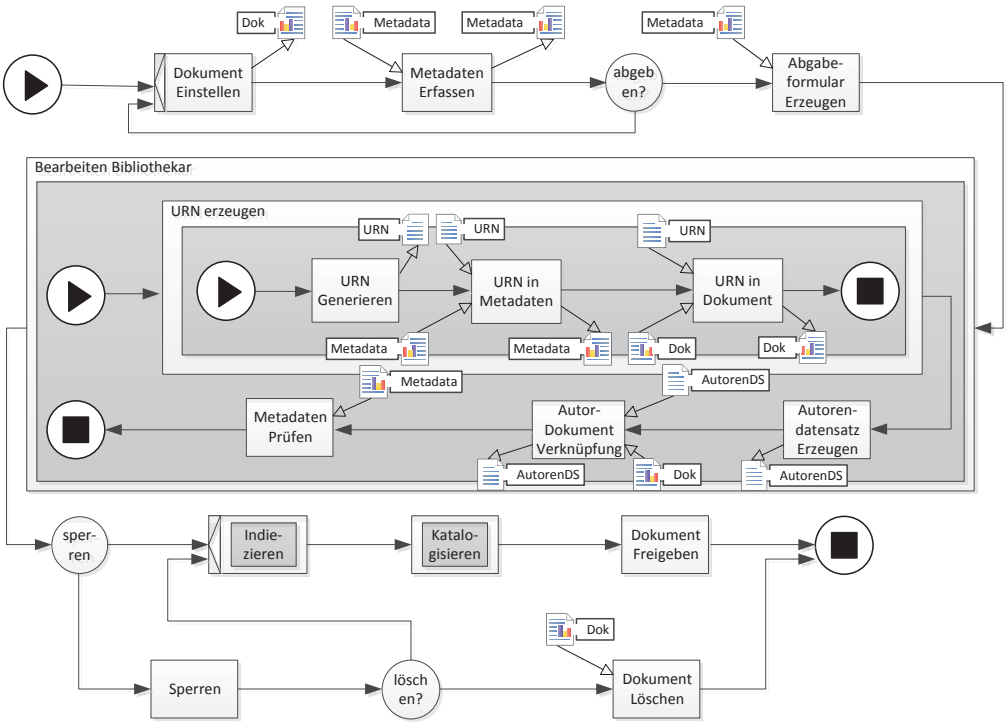


Abbildung 3.2: Publikationsprozess *Dissertation Online*

Das Prozessmodell in Abbildung 3.2 startet mit der Aktivität *Dokument einstellen*, die es dem Autor ermöglicht eine Dissertation mit allen Dokumenten bereitzustellen. Die Aktivität er-

zeugt als Ausgabe einen Dokumentendatensatz *Dok* und speichert zusätzlich alle Dokumente im Repository. Dokumente können beispielsweise PDF-Dokumente, Primärdaten, Bilder und Videos sein. Das Bibliothekssystem kann weiterhin komplex strukturierte, multimediale Dokumenttypen unterstützen.

Im nächsten Schritt werden alle Metadaten erfasst (Aktivität *Metadaten Erfassen*), wobei ein Metadatensatz *Metadata* erzeugt wird. Entscheidet der Promovend, die Arbeit endgültig abzugeben, wird ein Abgabeformular erzeugt. Die Aktivität *Abgabeformular* verwendet hierfür die Daten aus dem Metadatensatz.

Im Anschluss werden in der komplexen Aktivität *Bearbeiten Bibliothekar* eine URN erzeugt, ein Autorendatensatz (*AutorenDS*) angelegt, das Dokument mit dem Autorendatensatz verknüpft und die eingegebenen Metadaten durch einen Bibliothekar überprüft. Eine URN wird wiederum im Subprozess *URN erzeugen* generiert. Dabei ruft die Aktivität *URN generieren* einen Service auf, der eine URN liefert. Diese URN muss im System gespeichert werden. Danach schreibt die Aktivität *URN in Metadaten* die URN in die Metadaten. Hierfür muss die URN und der Metadatensatz eingelesen werden. Dann wird die URN in das Dokument (Dissertation als PDF) geschrieben, wofür als Eingabe die URN und das Dokument benötigt werden.

Die Aktivität *AutorDokumentVerknüpfung* benötigt als Eingabe den Autorendatensatz und das Dokument, um eine entsprechende Verbindung anzulegen. Im Anschluss darauf überprüft der Bibliothekar nochmal die Metadaten (Aktivität *Metadaten Prüfen*). Hat die Dissertation einen Sperrvermerk, muss sie für den Zugriff gesperrt werden (Aktivität *Sperren*). Nach einer Frist wird entschieden, ob die Dissertation dann gelöscht wird (Aktivität *Dokument löschen*), oder weiter verarbeitet werden kann. Die komplexe Aktivität *Indizieren* bereitet die Dokumente für die Indizierung vor und erstellt notwendige Indizes. Der Subprozess wird an späterer Stelle in diesem Kapitel vorgestellt. In der nachfolgenden Aktivität *Katalogisieren* werden entsprechende Katalogdaten aus dem Katalog übernommen, welche im Vorfeld durch einen Bibliothekar dort extern eingepflegt wurden. Abschließend wird das Dokument durch die Aktivität *Dokument freigeben* für den Nutzer der digitalen Bibliothek freigegeben.

Neben den Aktivitäten ist außerdem festgelegt, dass die Variablen *Dok, Metadata und AutorenDS* mit externen Datenquellen synchronisiert werden sollen. Die Variable *URN* kann intern verwaltet werden, da sie nicht unabhängig vom Dokument gespeichert werden muss. Alle Aktivitäten, die in der komplexen Aktivität *Bearbeiten Bibliothekar* verwendet werden, dürfen ihre erzeugten Daten erst mit den externen Datenquellen synchronisieren, wenn alle Aktivitäten erfolgreich abgearbeitet wurden.

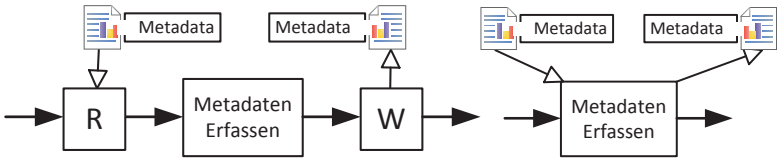
Im Folgendem sollen Anforderungen an die Integration externer Datenquellen und die Möglichkeiten zur Flexibilisierung von Prozessmodellen zur Unterstützung von Publikationsprozessen durch Workflow-Managementsysteme diskutiert werden.

3.2 Datenintegration

In diesem Abschnitt werden Anforderungen an die Datenintegration in WFMS analysiert. Dazu werden zunächst zwei unterschiedliche Varianten für die Integration von externen Datenquellen vorgestellt. Danach werden Anforderungen an die Datenintegration formuliert.

3.2.1 Problemstellung

Anhand des *YAWL*-Prozessmodells in Abbildung 3.2, sollen unterschiedliche Möglichkeiten diskutiert werden, wie die Integration von Datenquellen mit dem *YAWL*-System umgesetzt werden kann und welche Probleme dabei auftreten.



(a) Explizit modellierter Datenaustausch (b) Implizit modellierter Datenaustausch

Abbildung 3.3: Datenaustausch mit *YAWL*

Integration von Datenquellen
In Abschnitt 2.2.2 wurden bereits zwei unterschiedliche Varianten für die Integration externer Datenquellen vorgestellt. Mit *YAWL* können diese Techniken umgesetzt werden. Abbildung 3.3 zeigt zwei mögliche Umsetzungen, wie die Variable *Metadaten* mit einer externen Datenquelle synchronisiert werden kann. Die Aktivität *Metadaten Erfassen* muss die Variable *Metadaten* lesen und schreibt eine neue Version der Variablen in die Datenquelle zurück. Im Folgenden werden die beiden Varianten vorgestellt.

Variante 1 (Zugriff mit einem Service): In Abbildung 3.3a wird die Umsetzung der ersten Variante gezeigt. Wie in Abschnitt 2.2.2 aufgezeigt, kann ein (Web-)Service für die Synchronisation mit einer externen Datenquelle verwendet werden. Wie bei den meisten WFMSen besteht bei *YAWL* ebenfalls nur die Möglichkeit, genau einen Service pro Aktivität einzubinden: Entweder den Datenservice oder einen Service der einen externe Funktion aufruft. Dies führt dazu, dass der Datenzugriff explizit modelliert werden muss. In Abbildung 3.3a werden hierfür die Aktivitäten *R* für lesenden Zugriff und *W* für schreibenden Zugriff eingeführt, welche entsprechende Services für den Datenaustausch integrieren. Damit die Aktivität *Metadaten Erfassen* Zugriff auf die Variable erhält, muss diese auf eine globale Netzvariable abgebildet werden.

Variante 2 (Zugriff mit dem Data Gateway): Eine zweite mögliche Umsetzung wird in Abbildung 3.3b dargestellt. Weil *YAWL* keine Modellierungskonzepte für die Verwendung von Daten in der Kontrollflussperspektive bereitstellt, ist diese Notation gewählt worden. Mit Hilfe eines Data Gateways können Variablen mit externen Datenquellen synchronisiert werden. Hierfür ist ein Java-Programm erforderlich, dass die jeweilige Datenquelle korrekt einbinden kann. Dies wird von *YAWL* nicht bereitgestellt.

Strategien für die Datenintegration

Das Prozessmodell in Abbildung 3.2 beschreibt den Sichtbarkeitsbereich der Variablen und deren Anbindung an externe Datenquellen nicht vollständig. Zunächst muss deshalb der Sichtbarkeitsbereich für Variablen und der Zeitpunkt für die Synchronisation mit den Datenquellen festgelegt werden. Entsprechend den Möglichkeiten von *YAWL* (siehe Abschnitt 2.2.1.2) kann eine Variable entweder als Netzvariable oder Aktivitätsvariable modelliert werden. In Abschnitt 2.2.2 wurde weiterhin die Unterscheidung zwischen *internen* und *externen* Variablen eingeführt. *Interne Variablen* müssen mit einem Wert aus einer externen Datenquelle initialisiert werden. Während der Laufzeit der Prozessinstanz kann der Wert durch Aktivitäten geändert werden. *Externe Variablen* (siehe Abschnitt 2.2.2) können je nach modellierter Zugriffsart beim Starten oder Beenden der Aktivität mit der externen Datenquelle synchronisiert werden. Es besteht zusätzlich noch die Möglichkeit, den Wert der Variablen in eine interne Variable zu kopieren. In Abschnitt 3.1 wurde bereits festgelegt, nur die Variablen *Dok, Metadata und AutorenDS* als externe Variablen zu modellieren. Weil *YAWL* keine Implementierungen für die Integration externer Datenquellen mit Hilfe eines Data Gateways anbietet, sollen für die Synchronisation der externen Variablen Services verwendet werden, die eine Verbindung zur Datenquelle bereitstellen. Deshalb wird für jede externe Variable eine Netzvariable in *YAWL* angelegt. Aktivitäten, die einen Wert aus einer Datenquelle lesen oder einen Wert schreiben, müssen diesen mit der Netzvariable synchronisieren. Nur so ist ein Datenaustausch zwischen den Aktivitäten möglich (siehe Abschnitt 2.2.1.2). Für die Synchronisation mit der externen Datenquelle wird der in Abschnitt 3.2.1 eingeführte explizite Datenaustausch verwendet.

Am Beispiel der Variable *Metadata* werden unterschiedliche Lösungsstrategien für die Umsetzung des Prozessmodells diskutiert. Bezogen auf das Beispiel in Abbildung 3.2 ergeben sich folgende Lösungsstrategien:

Strategie 1: Die Variable wird immer mit der externen Datenquelle synchronisiert, wenn sie von einer Aktivität verwendet wird.

Strategie 2: Es ist ausreichend, wenn die Variable nur an bestimmten Punkten im Kontrollfluss synchronisiert wird.

Strategie 3: Die Variable wird nur beim Erzeugen und Beenden einer Prozessinstanz mit der Datenquelle synchronisiert.

Die gewählte Strategie hat somit Einfluss auf den Typ und die Werte der Variablen.

Um **Strategie 1** umzusetzen, muss die Variable *Metadata* mit Hilfe von Lese- und Schreib-Aktivitäten integriert werden. Abbildung 3.4 zeigt, wie das Prozessmodell aus dem Beispiel angepasst werden muss. Diese Art der Umsetzung ist sehr aufwendig, weil jeder Datenzugriff durch eine eigene Aktivität modelliert werden muss. Gleichzeitig kann nicht sichergestellt werden, dass die Aktivitäten *Metadaten Erfassen* und *Abgabeformular* mit denselben Werten der Variablen *Metadata* arbeiten. Eine andere Anwendung kann die Daten in der externen Datenquelle, zwischen dem Beenden der einen Aktivität und dem Starten der anderen Aktivität, ändern.

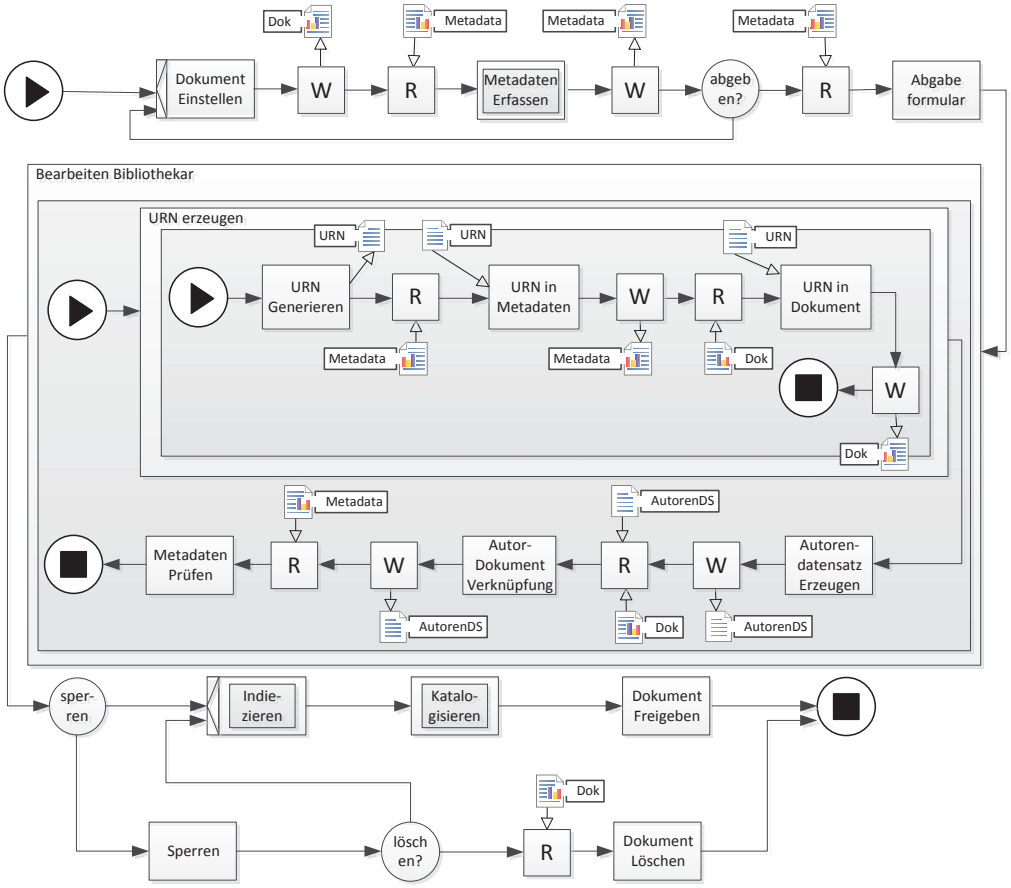


Abbildung 3.4: Publikationsprozess *Dissertation Online* mit explizitem Datenzugriff

Eine Umsetzung von **Strategie 2** wird in Abbildung 3.5 gezeigt. Hier kann die Variable *Metadata* beispielsweise einmal gelesen werden und erst von der Aktivität *Metadaten Prüfen* in die externe Datenquelle geschrieben werden. Es muss eine Netzvariable angelegt werden, welche den Wert allen Aktivitäten bereitstellt. An dieser Lösung ist nachteilig, dass externe Anwendungen zwischenzeitlich keinen Zugriff auf die Ergebnisse des Prozesses haben. Weiterhin muss der Prozessmodellierer genau wissen, wann eine Synchronisation sinnvoll ist und welche Auswirkungen dies auf den weiteren Prozessverlauf hat.

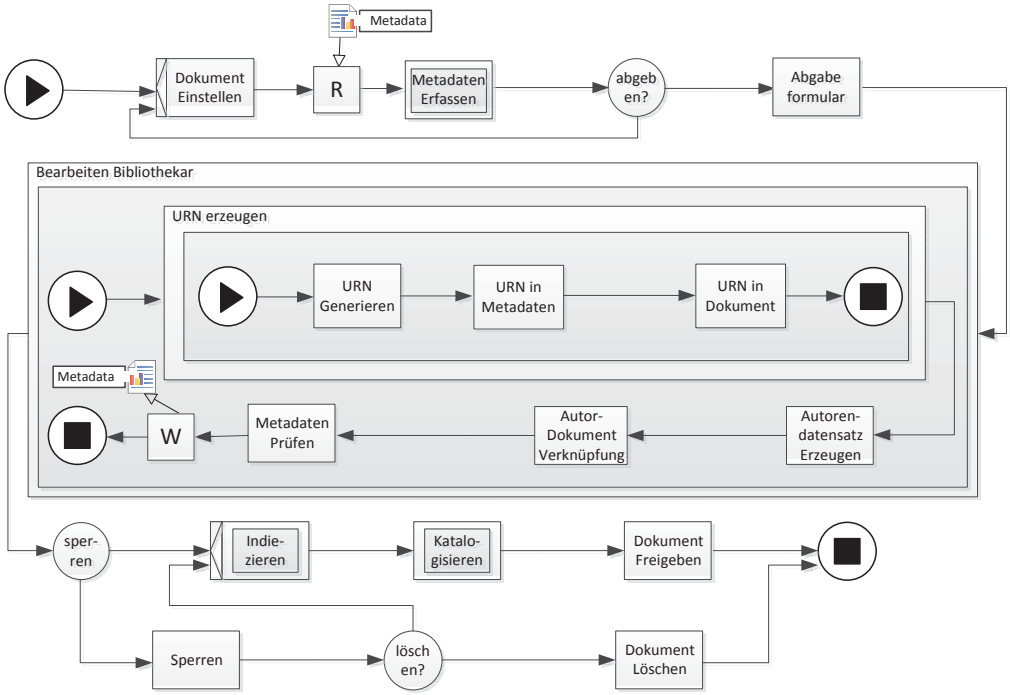


Abbildung 3.5: Publikationsprozess *Dissertation Online* mit Datenzugriff nach Strategie 2

Die Umsetzung von **Strategie 3** ist ein Spezialfall von Strategie 2. Die Synchronisation wird auf das Erzeugen und Beenden einer Prozessinstanz gelegt. Der Nachteil dieser Umsetzung besteht darin, dass externe Anwendungen keinen Zugriff auf die Daten des Prozesses haben.

Die vorgestellten Beispiele verdeutlichen zwei wesentliche Probleme.

1. Die Synchronisierung mit externen Datenquellen kann nicht direkt mit den Konzepten des WFMS kontrolliert werden. Der Prozessmodellierer muss die Synchronisation durch die verwendeten Modellierungskonzepte explizit beeinflussen.
2. Der Datenzugriff muss explizit durch zusätzliche Aktivitäten modelliert werden. Dies führt zu aufgeblähten Prozessmodellen, die zudem zu einer höheren Fehleranfälligkeit führen. Der Nutzer muss zusätzlich entscheiden, wann eine Synchronisation erfolgen muss und wann diese eventuell Konflikte erzeugt.

3.2.2 Anforderungen an die Datenintegration

Aus den genannten Problemen ergeben sich eine Reihe von Anforderungen, die für die Integration von externen Datenquellen gelten müssen [SMH11a].

1. Datenmodellierung
   (a) Das WFMS muss externe Daten mit Hilfe eines eigenen Variablentyps für externe Variablen integrieren.

   (b) Der Datenzugriff auf externe Datenquellen muss für den Nutzer transparent sein.
2. Datenkontrolle
   (a) Aktivitäten sollen auf die externen Daten nur über externe Variablen zugreifen. Services sollen hierfür nicht mehr verwendet werden.
   (b) Das WFMS muss den Zeitpunkt für den Zugriff auf externe Datenquellen kontrollieren können.
3. Transaktionskontrolle
   (a) Das WFMS soll Transaktionskonzepte für externe Variablen unterstützen.
   (b) Für externe Variablen sollen Integritätsbedingungen, Rücksetzbarkeit und Korrektheit sichergestellt werden.
   (c) Das WFMS soll geeignete Ausnahmebehandlungen bei Verletzung der oben genannten Anforderungen bieten.

Die vorgestellten Anforderungen an die Datenmodellierung, Datenkontrolle und Transaktionskontrolle werden im Folgenden näher erläutert.

Datenmodellierung

Anforderung 3.1 (Externe Variablen): Für den Datenzugriff auf externe Datenquellen sollen externe Variablen durch ein explizites Modellierungskonzept unterstützt werden. Hierfür muss das Prozess-Metamodell mit einem geeigneten Konstrukt, z. B. einer externen Variable, erweitert werden.

Anforderung 3.2 (Transparenter Datenzugriff): Als Grundlage für die Integration externer Datenquellen mit Hilfe von externen Variablen muss der Datenzugriff transparent für den Nutzer sein. Der Nutzer muss eine Datenquelle für die Synchronisation angeben, die Lese- und Schreibstrategie auswählen und gegebenenfalls eine Abbildung auf eine interne Variable definieren.

Datenkontrolle

Anforderung 3.3 (Zugriff auf externe Datenquellen): Der Zugriff auf externe Datenquellen mit Hilfe von externen Variablen muss durch das WFMS ermöglicht und kontrolliert werden. Hierfür muss das WFMS eine geeignete Architektur bereitstellen, womit externe Datenquellen einheitlich integriert und verwaltet werden können. Dazu zählen eine einheitliche Schnittstellendefinition und die Möglichkeit, konkurrierende Datenzugriffe auf Datenquellen zu verwalten.

Transaktionskontrolle

Anforderung 3.4 (Transaktionskonzepte): Der Datenzugriff soll unabhängig vom Kontrollfluss bzw. von Nutzerentscheidungen erfolgen. In Anlehnung an Datenbank-Manage-

ment-Systeme sollen externe Variablen hierfür unter Transaktionskontrolle stehen und Konzepte wie Konsistenz, Isolation, Dauerhaftigkeit oder Korrektheit unterstützen. Der Nutzer soll transaktionale Bereiche definieren, die analog zu Datenbank-Transaktionen den Datenzugriff auf externe Datenquellen steuern. Eine elementare Forderung besteht darin, dass innerhalb eines transaktionalen Bereiches das WFMS entscheidet, wann ein Datenzugriff stattfindet.

Anforderung 3.5 (Transaktionale Bereiche): Konzepte, wie Integritätsbedingungen, Korrektheit und Rücksetzbarkeit sollen innerhalb transaktionaler Bereiche sichergestellt werden. Hierfür muss das Prozess-Metamodell geeignete Modellierungskonzepte bereitstellen, die eine Modellierung der Bereiche ermöglichen. Die Modellierungskonzepte müssen aber auch vom WFMS geeignet unterstützt werden.

Anforderung 3.6 (Ausnahmebehandlungen): Für den Fall, dass Fehler innerhalb transaktionaler Bereiche auftreten, müssen geeignete Ausnahmebehandlungen definiert werden. Das WFMS muss deren Modellierung und Umsetzung unterstützen. Mögliche Ausnahmebehandlungen bei verletzten Integritätsbedingungen können beispielsweise den Abbruch eines transaktionalen Bereiches oder einer ganzen Prozessinstanz nach sich ziehen. Für mögliche Abbrüche müssen Strategien für die Rücksetzbarkeit angeboten werden.

3.3 Prozesskomposition

Die Prozessmodelle, die den Publikationsprozess abbilden, sind sehr stark datenzentriert. In diesem Abschnitt sollen die sich daraus ergebenden Abhängigkeiten und Anforderungen analysiert werden.

3.3.1 Problemstellung

Mit zunehmender Verbreitung digitaler Dokumente verändern sich auch die Einsatzmöglichkeiten von digitalen Bibliothekssystemen und Anforderungen an die Dokumentmodelle. Durch die immer komplexer werdenden Dokumentmodelle, mit multimedialen Bestandteilen, steigen die Anforderungen an digitale Bibliothekssysteme. Außerdem müssen die Bibliothekssysteme häufiger Aufgaben aus den Bereichen des klassischen Content-Managements übernehmen.

Publikationsprozess
In einem Bibliothekssystem muss der Publikationsprozess den gesamten Lebenszyklus der Dokumente unterstützen. Dazu müssen die einzelnen Phasen vom *Erstellen der Dokumente* bis zum *Publizieren und Archivieren* unterstützt werden (siehe Abschnitt 2.1.3). Um die

komplexen Arbeitsabläufe unabhängig von der Anwendungslogik zu beschreiben, werden WFMSe eingesetzt. Die Arbeitsabläufe sind dann durch Prozessmodelle beschrieben und so unabhängig von der restlichen Applikationslogik.

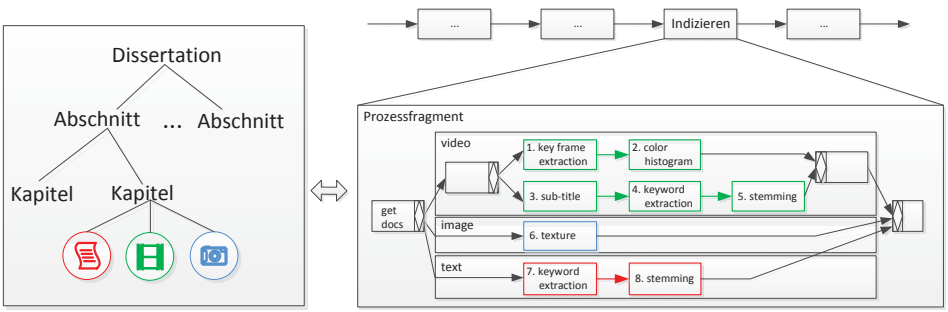
Komplexe Dokumentmodelle beschreiben die Struktur von Multimediadokumenten, die Bilder, Texte, Videos, numerische Daten und andere Datentypen enthalten. Prozessmodelle müssen für diese Multimediadokumente den gesamten Lebenszyklus beschreiben. Inhalt und Struktur der Dokumente werden während der Ausführung einer Prozessinstanz beliebig oft geändert. Dies steht im direkten Kontrast zu einfachen Repositorien, die abgeschlossene Dokumente verwalten (z. B. PDF-Dateien). Ein Dokument kann daher unterschiedliche Ausprägungen haben. Prozessmodelle müssen Arbeitsabläufe so beschreiben, dass die unterschiedlichen Ausprägungen der Dokumente zu keinen Fehlern bei der Ausführung von Prozessinstanzen führen.

Prozessmodelle

Publikationsprozesse zeichnen sich durch zwei Eigenschaften aus, die die Anforderungen an ein Prozess-Metamodell beeinflussen.

Der Publikationsprozess hat eine fest *vorgegebene Grundstruktur* (siehe Abschnitt 2.1.3), die sich sehr selten ändert. Die anwendungsspezifischen Prozessbereiche orientieren sich an Bibliotheksprozessen, die sich seit langem bewährt haben. Diese Prozesse sind abhängig von der konkreten Anwendung, müssen aber nicht flexibel änderbar sein. Änderungen erfordern oft eine Anpassung von Anwendungen und sollten deshalb durch neue Versionen eines Prozessmodells eingeführt werden.

Daneben sind *dokumentabhängige Prozessbereiche* sehr stark datenzentriert. Innerhalb der oben genannten Grundstruktur existieren Prozessbereiche, deren Abarbeitung direkt von den Daten abhängig ist. Am Beispiel des Teilprozesses für die Indizierung von unterschiedlichen Dokumenttypen im Publikationsprozess *Dissertation Online* sollen diese Beziehungen betrachtet werden.



(a) Dokumentstruktur (b) Prozess

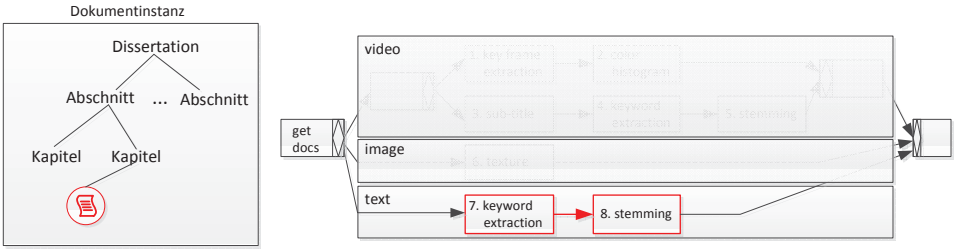
Abbildung 3.6: Subprozess für die Indizierung einer Dissertation

In Abbildung 3.6 wird der Teilprozess der Indizierung eines Dokumentes gezeigt. Der Teilprozess ist in Abbildung 3.2 als komplexe Aktivität *Indizieren* modelliert. Anhand der Dokumentstruktur (siehe Abbildung 3.6a) muss entschieden werden, welche Pfade im Prozess aktiviert bzw. durchlaufen werden müssen. Dazu wertet die Aktivität *get docs* das zu verarbeitende Dokument aus, um die verwendeten Dokumenttypen zu ermitteln. Enthält ein Dokument ein Video, dann werden Key-Frames extrahiert, die im Anschluss für die Extraktion von Bild-Eigenschaften genutzt werden. Im Beispiel wird ein Histogramm für jeden Key-Frame erstellt. Wenn das Video einen Untertitel enthält, kann dieser parallel zu den genannten Schritten extrahiert und anschließend mit Hilfe von Volltextindizierung indiziert werden. Enthält das Dokument ein Bild, werden Texturen extrahiert. Texte werden ebenfalls in einem eigenen Prozesspfad bearbeitet. Enthält ein Dokument einen Text, werden Stichworte extrahiert, die mit Hilfe von Stammwortreduktionen verarbeitet werden.

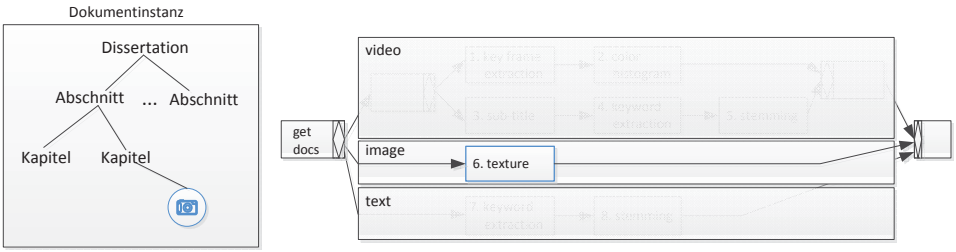
Das Beispiel zeigt, dass bestimmte Aktivitäten wie beispielsweise Key-Frame-Extraktion und die Verarbeitung eines Untertitels parallel ausgeführt werden können. Andere Aktivitäten müssen hingegen in einer bestimmten Reihenfolge als Sequenz ausgeführt werden. Der Untertitel muss z. B. vor der Volltextindizierung extrahiert worden sein. Weiterhin ist die Ausführung mancher Aktivitäten von der Existenz bestimmter Dokumentteile abhängig. Der Aufwand für Prozessänderungen wächst somit auch mit jedem neuen Dokumenttyp, der eingeführt wird.

Neben den Abhängigkeiten zwischen einzelnen Aktivitäten sind auch ganze Prozesspfade vom Zustand und Inhalt des Dokumentes abhängig. Enthält die Dissertation z. B. nur Text, wird der Pfad für die Textindizierung (siehe Abbildung 3.7a) durchlaufen. Die Prozesspfade für die Indizierung von Bildern und Videos werden nicht durchlaufen. Sind in einem Teildokument nur Bildinformationen enthalten (siehe Abbildung 3.7b), wird nur der Prozesspfad für die Merkmalsextraktion von Bildern aktiviert. Wohingegen die Prozesspfade für die Volltext- und Videoindizierung deaktiviert bleiben. Es sind aber auch Kombinationen von Prozesspfaden möglich, wie Abbildung 3.7c zeigt. Dieser Fall tritt ein, wenn das Dokument Text und Videos enthält, aber kein Bild. Dementsprechend wird nur der Prozesspfad für die Merkmalsextraktion von Bildern nicht durchlaufen. Die Beispiele zeigen, dass Prozesspfade immer in der Prozessinstanz enthalten sind. Das gilt auch, wenn sie nicht benötigt werden.

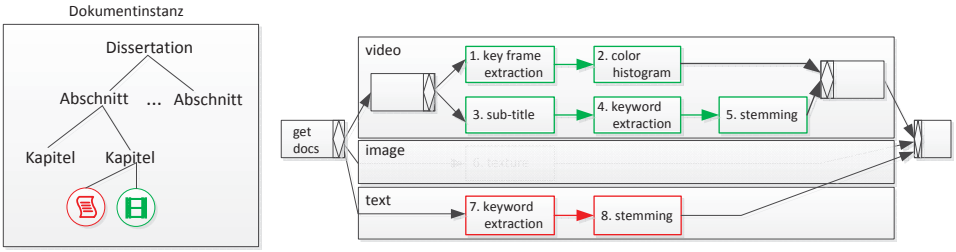
Die Abhängigkeiten zwischen Dokumenten und Aktivitäten sind nicht nur lokal für bestimmte Prozessbereiche festzustellen, sie können sich über den gesamten Prozess erstrecken. Wird beispielsweise ein Video im Dokument verwendet, werden entsprechende Aktionen in unterschiedlichen Bereichen des Prozesses aufgerufen. Zum Beispiel müssen in den komplexen Aktivitäten *Metadaten Erfassen, Metadaten Prüfen, Indizieren* und *Katalogisieren* inhaltsabhängige Operationen aufgerufen werden. Es müssen also die richtigen Services eingebunden (Indizierung etc.) aber auch fachkundige Bibliothekare für die Bearbeitung ausgewählt werden (Metadaten überprüfen, Katalogisierung).



(a) Subprozess für Dokument mit Text



(b) Subprozess für Dokument mit Bild



(c) Subprozess für Dokument mit Text und Video

Abbildung 3.7: Varianten für den Subprozess Indizierung einer Dissertation

3.3.2 Anforderungen an die Prozesskomposition

Ein Prozessmodell muss alle Abhängigkeiten und Bedingungen abbilden, die durch verschiedene Dokumenttypen entstehen. Die Anzahl der unterschiedlichen Dokumenttypen in einem Dokumentmodell hat entsprechend Einfluss auf die Komplexität der Prozessmodelle.

Deshalb soll die Wartbarkeit und Erweiterbarkeit der Prozessmodelle verbessert werden. Dies soll durch die Bereitstellung eines flexiblen, datengetriebenen Prozessmodells erreicht werden.

Dokumentabhängige Operationen

Weil die Arbeitsabläufe sehr stark von den zu bearbeitenden Dokumenten abhängig sind, können sie nicht unabhängig von den Dokumenten modelliert und ausgeführt werden. Ein Prozessmodell muss deshalb insbesondere dokumentabhängige Operationen wie die Angabe von Metadaten oder die Indizierung der Dokumente flexibel integrieren können. In unserem Ansatz stellen wir deshalb folgende Anforderungen, die Prozessmodelle unterstützen müssen.

1. Modellierung
   (a) Vermeidung komplexer Prozessmodelle
   (b) Wiederverwendbarkeit von dokumentabhängigen Prozessfragmenten
   (c) Entkopplung anwendungsspezifischer Prozessfragmente von dokumentspezifischen Prozessfragmenten
   (d) Definition von Operationen zur Anpassung der Prozessinstanzen
2. Ausführung
   (a) Einfügen oder Löschen dokumentabhängiger Prozessfragmente in den Prozess in Abhängigkeit von Kontextänderungen
   (b) Einfügen oder Löschen dokumentabhängiger Prozessfragmente nur in festgelegten Prozessbereichen
   (c) Operationen auf Dokumenten
   (d) Konstruktionsregeln für die Komposition dokumentabhängiger Prozessfragmente

Die vorgestellten Anforderungen an die Modellierung und Ausführung von Prozessmodellen werden im Folgenden näher erläutert.

Modellierung

Anforderung 3.7 (Komplexe Prozessmodelle vermeiden): Umso komplexer die zu verarbeitenden Dokumente werden, desto komplexer werden auch die Prozessmodelle. Ziel soll es sein, die Komplexität der dokumentspezifischen Bestandteile eines Prozessmodells möglichst gering zu halten. So kann z. B. die Einführung neuer Dokumenttypen erleichtert werden.

Anforderung 3.8 (Wiederverwendbarkeit von Prozessfragmenten): Die dokumentspezifischen Prozessbestandteile sollen in Form von Prozessfragmenten wiederverwendbar sein. Prozessfragmente ermöglichen eine einfachere Wartbarkeit von Prozessen. Zum Beispiel kann ein Prozessfragment *Text Indizieren* für unterschiedliche Dokumenttypen verwendet werden (siehe Beispiel).

Anforderung 3.9 (Anwendungsspezifische Prozessfragmente von dokumentspezifischen Prozessfragmenten entkoppeln): Das Prozessmodell soll eine klare Trennung zwischen anwendungs- und dokumentspezifischen Bestandteilen ermöglichen. Hiermit soll der dynamischen Struktur der dokumentspezifischen Bestandteile Rechnung getragen werden. Im bibliothekarischen Grundprozess sollen keine Änderungen vorgenommen werden. Ausgenommen sind in diesem Fall Ausnahmebehandlungen, welche hier aber nicht im Zusammenhang mit den Publikationsprozessen betrachtet werden.

Ausführung

Anforderung 3.10 (Einfügen oder Löschen dokumentabhängiger Prozessfragmente in den Prozess in Abhängigkeit von Kontextänderungen): Prozessinstanzen sollen in Abhängigkeit vom aktuellen Kontext (hier Zustand und Inhalt der Dokumente), zur Laufzeit angepasst werden. Ein Basisprozessmodell kann den anwendungsspezifischen Prozess abbilden. Prozessinstanzen werden zur Laufzeit, in Abhängigkeit von Kontextänderungen, angepasst. Damit sollen zum einen weniger komplexe Prozessstrukturen und zum anderen die Erweiterbarkeit der Prozessmodelle ermöglicht werden.

Anforderung 3.11 (Einfügen oder Löschen dokumentabhängiger Prozessfragmente nur in festgelegten Prozessbereichen): Innerhalb eines Prozesses soll in definierten Bereichen flexibel auf aktuelle Dokumentausprägungen reagiert werden können. Dadurch soll verhindert werden, dass anwendungsspezifische Prozessfragmente verändert werden. Dies ist notwendig, weil der bibliothekarische Grundprozess streng vorgegeben ist.

Anforderung 3.12 (Operationen auf Dokumenten): Für die Ausführung flexibler Prozesse ist nicht nur die aktuelle Dokumentausprägung wichtig (ob ein Dokumenttyp enthalten ist oder nicht). Eine wichtige Rolle spielen auch die Operationen, die auf den Dokumenten ausgeführt wurden. Es ist z. B. ein Unterschied, ob ein Dokumenttyp Video nicht enthalten ist oder ob er während der Bearbeitung des Prozesses, gelöscht wird. Das Löschen eines Dokumenttyps kann hier Kompensationsoperationen erfordern. Deshalb muss eine Menge von Grundoperationen auf Dokumenten festgelegt werden, und es müssen die Auswirkungen ihrer Ausführung auf die Prozessinstanz beachtet werden.

Anforderung 3.13 (Konstruktionsregeln für die Komposition dokumentabhängiger Prozessfragmente): Sind mehrere Prozessbausteine gleichzeitig aktiviert, muss für die Komposition der dokumentabhängigen Prozessfragmente eine Regelmenge (Konstruktionsregeln) definiert sein. Diese Regelmenge beschreibt Abhängigkeiten zwischen den einzelnen Prozessbausteinen, z. B. eine Reihenfolge oder die Möglichkeit Prozessbausteine parallel auszuführen.

3.4 Aktuelle WFM-Systeme

In diesem Abschnitt werden ausgewählte Workflow-Management-Systeme vorgestellt und deren Funktionsumfang auf die oben gestellten Anforderungen hin untersucht.

Together XPDL and BPMN Workflow Server. Der *Together XPDL and BPMN Workflow-Server* (TWS) [Tog11b] nutzt XPDL V2.1 als interne Repräsentation für Prozesse. XPDL

wird von der WfMC als Austauschformat und für die Serialisierung von BPMN Modellen entwickelt [Sha08]. Für die Modellierung von Prozessen bietet Together einen eigenen Editor an. Der *XPDL und BPMN Workflow-Editor* [Tog11a] setzt BPMN für die grafische Darstellung ein. Der Workflow-Server von Together kann externe Datenquellen über sogenannte *Tool Agents* einbinden. Das Konzept der Tool Agents entspricht den Vorgaben aus dem Referenzmodell [WfM98] der WfMC, *Interface 3*, für WFMS. Aktivitäten können einen Tool Agent aufrufen, Daten übergeben und Daten vom Tool Agent empfangen. Eine Kontrolle über den Datenzugriff und die explizite Modellierung von externen Datenquellen, sowie die Angabe von transaktionalen Eigenschaften sind nicht möglich. Prozessmodelle können mit Hilfe von Subprozessen hierarchisch modelliert werden. Flexibilität zur Laufzeit und Modularität wird nicht unterstützt.

jBPM. Das WFMS *jBPM*[1] bietet die Möglichkeit, Prozessmodelle umzusetzen, die mit *BPMN 2.0* modelliert sind. Zu den unterstützten Konzepten gehören eine Monitoring-Komponente, flexible Prozesse und Transaktionen basierend auf *JPA/JTA* (Java Persistence API / Java Transaction API). Externe Datenquellen können mit Hilfe unterschiedlicher Konzepte integriert werden. *ActionScripts* erlauben es beim Starten und Beenden unterschiedlicher Aktivitätstypen aufgerufen zu werden. Dabei besteht die Möglichkeit, Java-Programmcode auszuführen. Ein Action Script hat Zugriff auf alle globalen Variablen und die Prozessvariablen. Weiterhin können Services integriert werden, die via Service Tasks verwendbar sind. Hier werden entsprechend Prozessdaten als Eingabe- bzw. Ausgabeparameter der Services definiert. Ein Nachteil ist, dass Action Scripts auf Java-Klassen angewiesen sind, die implementiert werden müssen. Daneben ist der Datenzugriff nicht durch das WFMS gesteuert. Transaktionskonzepte zur Unterstützung des Datenaustausches mit externen Datenquellen werden nicht unterstützt. *jBPM* bietet eine Reihe unterschiedlicher Möglichkeiten für die flexible Modellierung und Ausführung von Prozessen an. Prozesse können mit Hilfe von *reusable sub-processes* oder *embedded sub-processes* hierarchisch unterteilt werden. Die beiden Konzepte unterschieden sich darin, dass ein *reusable sub-process* einen anderen Prozess integriert. Wohingegen ein *embedded sub-processes* nur eine hierarchische Aufteilung innerhalb desselben Prozessmodelles darstellt. Zur Laufzeit können mit Hilfe eines *ad-hoc sub-process* Elements, Prozessfragmente in beliebiger Reihenfolge aufgerufen werden. Dabei können innerhalb dieses Bereiches auch neue Aktivitäten hinzugefügt werden. Es können aber keine Regeln oder Beziehungen zwischen den Fragmenten dargestellt werden. Die Reihenfolge wird immer vom Nutzer festgelegt.

YAWL. Das WFMS *YAWL* wurde bereits in Abschnitt 2.2.1 beschrieben. *YAWL* bietet die Möglichkeit, externe Datenquellen mit Hilfe von Services oder Data Gateways zu integrieren. In beiden Fällen ist die Definition von Java-Programmcode erforderlich. Data Gateways bieten den Vorteil, dass sie vor bzw. nach Ausführung eines Netzes oder einer

[1] `http://www.jboss.org/`

Aktivität aufgerufen werden. Eine Aktivität kann Daten an einen Service übergeben und nach dessen Ausführung empfangen. Transaktionale Konzepte werden von *YAWL* nicht unterstützt. Dynamische Anpassungen von Prozessmodellen zur Laufzeit bietet *YAWL* durch die Integration von *Worklets* [AHEA06b] an. Hier kann zur Laufzeit anhand von Regeln ein vorher definiertes Prozessmodell ausgeführt werden. Der Ansatz ermöglicht eine flexible Auswahl von Prozessen, deren Anzahl mit den möglichen Operationen sehr stark wächst. Prozessvarianten müssen in immer neuen Prozessmodellen dargestellt werden.

ADEPT2. Im Rahmen des ADEPT-Projektes wurde von Reichert und Dadam das WFMS *ADEPT* [RD98, Rei00] und als Weiterentwicklung *ADEPT2* [RRKD05] entwickelt. *ADEPT2* bietet die Möglichkeit, strukturelle Veränderungen an Prozessinstanzen vorzunehmen. Hierfür werden eine Reihe von Änderungsoperationen bereitgestellt, welche auf ein gegebenes Prozessschema angewendet werden. Führt die Anwendung der Operationen zu einem neuen, korrekten und konsistenten Prozessmodell, können laufende Prozessinstanzen in Abhängigkeit des Zustandes der Prozessinstanz, an das neue Prozessmodell angepasst werden. *ADEPT2* bietet die Möglichkeit, komplexe Datenabhängigkeiten innerhalb des Prozessmodells zu definieren. Es stehen ein Datenflussschema und sogenannte *Sync*-Kanten zur Verfügung. Letztere ermöglichen das Synchronisieren von Datenzugriffen verschiedener Aktivitäten paralleler Prozesszweige. Der Schwerpunkt wird auf einen korrekten Datenfluss gelegt, der auch nach Änderungen des Kontrollflusses korrekt sein muss. Die Datenintegration aus externen Datenquellen wird über die eingebundenen Applikationen geregelt. Es können Daten an die Applikation übergeben werden und eine Applikation kann Daten an den Prozess übergeben, wenn sie beendet ist. Konzepte, die Transaktionen unterstützen, werden nicht angeboten. Bei der Modellierung flexibler Prozesse bietet *ADEPT2* eine große Vielfalt an Operationen an, die es ermöglichen Prozessinstanzen zur Laufzeit zu verändern. Weil der Fokus auf Änderungsoperationen gelegt wird, die vom Nutzer zur Laufzeit angewendet werden, wird eine automatische Konfiguration von Prozessinstanzen zur Laufzeit nicht unterstützt. Umfangreiche Änderungen an einem Prozessmodell führen außerdem zu komplexen Folgen von Änderungsoperationen, die durch die Sicherstellung der Konsistenz und Korrektheit sehr aufwendig werden können.

Tabelle 3.1: Anforderungen an Datenintegration und Prozesskomposition ausgesuchter WFMS

Anforderung	Datenintegration			Prozesskomposition	
System	Modellierung	Kontrolle	Transaktionen	Modellierung	Ausführung
jBPM	+/-	-	-	+/-	+/-
TWS	-	-	-	-	-
YAWL	+/-	-	-	+/-	+/-
ADEPT2	-	-	-	+/-	+/-

+: volle Unterstützung, +/-: teilweise Unterstützung, -: keine Unterstützung

Fazit

Aus Tabelle 3.1 geht hervor, dass die Anforderungen aus dem Bereich der Datenintegration von den Systemen *jBPM* und *YAWL* teilweise unterstützt werden. Es verbleibt aber nach wie vor ein großer programmiertechnischer Aufwand beim Nutzer. Die Anforderungen an die Prozesskomposition werden außer vom WFMS TWS von allen anderen Systemen zumindest teilweise unterstützt. Trotzdem ergeben sich erhebliche Unterschiede zwischen den Systemen. *ADEPT2* setzt sich durch den gebotenen Funktionsumfang deutlich von den anderen Systemen ab, kann aber trotzdem nicht alle Anforderungen unterstützen. *YAWL* und *jBPM* bieten zwar Möglichkeiten, Prozesse flexibel zur Laufzeit anzupassen, können die Anforderungen an die Datenintegration und die Prozesskomposition nur sehr eingeschränkt unterstützen. Abschließend kann gesagt werden, dass kein WFMS die vorgestellten Anforderungen vollständig erfüllt.

Teil II

Flexible Prozesskomposition am Beispiel von Publikationsprozessen in digitalen Bibliotheken

Kapitel 4

Flexible, datengetriebene Prozessmodelle

In Kapitel 3 wurden verschiedene Anforderungen formuliert, die für dokumentzentrierte Prozessmodelle erfüllt sein müssen. In diesem Kapitel werden Lösungsstrategien zur Umsetzung diskutiert. In Abschnitt 4.1 werden vorhandene Konzepte für die Integration von externen Datenquellen und für die Umsetzung von flexiblen Prozessen betrachtet. Es wird gezeigt, dass die Anforderungen nur teilweise umgesetzt werden können. Zur Lösung der Probleme schlagen wir in Abschnitt 4.2 einen eigenen Ansatz vor. Es wird ein Überblick über die entwickelten Konzepte gegeben, die in den Kapiteln 5, 6 und 7 im Detail beschrieben werden.

4.1 Konventionelle Lösungsstrategien

In diesem Abschnitt sollen unterschiedliche Lösungsstrategien für die Modellierung von Prozessmodellen betrachtet werden, die einen Publikationsprozess umsetzen. Im ersten Teil werden dafür verschiedene Strategien für die Modellierung und Integration von externen Datenquellen vorgestellt. Der zweite Teil stellt allgemeine Konzepte imperativer Prozessmodelle und deklarativer Prozessmodelle vor, um flexible Publikationsprozesse zu modellieren. Um eine Einschätzung der Einsetzbarkeit der unterschiedlichen Ansätze zu ermöglichen, werden sie anhand der in Kapitel 3 vorgestellten Anforderungen eingeordnet.

4.1.1 Lösungsstrategien für die Datenintegration

Die Integration von Daten wird in unterschiedlichen Ansätzen betrachtet, die im Folgenden vorgestellt werden.

Imperative Prozessmodelle

Prozessmodelle, die imperativ modelliert werden, verwalten Anwendungsdaten in einem internen Speicher, der vom WFMS verwaltet wird. Der Zugriff auf diese workflow-relevanten Daten wird durch ein Sichtbarkeitskonzept beschrieben (siehe Abschnitt 2.2.2). Grundsätzlich können diese Daten nicht außerhalb des WFMS geändert werden. Abbildung 3.4 (siehe Abschnitt 3.2) zeigt am Beispiel von *YAWL*, wie der Datenzugriff mit Hilfe von Prozessdaten umgesetzt werden kann. Im Prozessmodell müssen alle Variablen, die von verschiedenen Aktivitäten benutzt werden, global definiert sein. Der Sichtbarkeitsbereich wird damit auf die jeweilige Hierarchieebene begrenzt. Werden Daten in einem Subprozess benötigt, müssen sie gesondert bereitgestellt werden. Informationen dazu werden in Abschnitt 2.2.2 gegeben.

Für die Anforderungen aus Abschnitt 3.2.2 ergeben sich folgende Punkte:

Datenmodellierung. Der transparente Datenzugriff auf externe Datenquellen ist nur teilweise möglich. Die Integration externer Datenquellen wird mit Hilfe spezieller Services bzw. Skripte ermöglicht, die häufig genauso wie eine integrierte Applikation behandelt werden. Dabei besteht die Notwendigkeit, Programmcode für den Datenzugriff bereitzustellen.

Datenkontrolle. In Abschnitt 3.2 wird weiterhin gefordert, dass externe Daten nur über spezielle Konstrukte, wie externe Variablen, aufgerufen werden. Hierfür darf kein Service für die Datenintegration eingebunden werden, der die Benutzung einer Aktivität für ihren eigentlichen Verwendungszweck verhindert. Im Beispiel werden eine Reihe von Pseudo-Aktivitäten eingeführt, welche den Datenzugriff durch den Aufruf von Services ermöglichen (in Abbildung 3.3 mit *R* und *W* gekennzeichnet). Systeme wie jBPM bieten die Möglichkeit, Aufrufe externer Datenquellen unabhängig von Serviceaufrufen in einer Aktivität durchzuführen. Sie können aber auch keinen transparenten Zugriff bieten, weil immer noch Extra-Aufwand für die Integration betrieben werden muss (siehe Abschnitt 3.4). Da die aufgerufenen Services als Black-Box gegenüber dem WFMS betrachtet werden müssen, ist an dieser Stelle auch keine Kontrolle von Lese- und Schreiboperation durch das WFMS möglich.

Transaktionskontrolle. Weil der Datenzugriff mit Hilfe von Services, welche als Black-Box betrachtet werden müssen, realisiert wird, können auch nur bedingt Transaktionskonzepte umgesetzt werden. Integritätsbedingungen können teilweise lokal auf Prozessvariablen als Vor- und Nachbedingungen von Aktivitäten bestimmt werden. Über solche Bereiche in einem Prozessmodell können keine Bedingungen angegeben werden.

Object aware processes

In [KR11] wird der Begriff *object aware processes* eingeführt. Das vorgestellte Prozessmodell bietet eine bessere Integration von Daten in das Prozessmodell. Dabei wird das Hauptaugenmerk auf Prozessmodelle gelegt, die mit den benutzen Datenstrukturen konform sind. Ein Ziel liegt darin, Anwendungsdaten im Prozessmodell bereitzustellen und geeignet

mit den anderen Perspektiven wie Kontrollfluss und Ressourcen zu verknüpfen. Hierfür werden Datenmodelle zusammen mit Objektzuständen modelliert, die dann mit datengetriebenen Prozessmodellen kombiniert werden [KR11]). So kann eine integrierte Sicht auf den Prozess und die benutzten Daten bereitgestellt werden. Für die Anforderungen aus Abschnitt 3.2.2 ergeben sich folgende Punkte:

Datenmodellierung. Ein wesentlicher Vorteil bei der Modellierung, wie sie in [KR11] vorgeschlagen wird, liegt in der starken Verknüpfung zwischen Daten und den unterschiedlichen Prozessperspektiven. Der Datenzugriff auf externe Datenquellen wird aber nicht betrachtet. Die Daten werden vollständig im Prozessmodell integriert und zusammen mit Zustandsmodellen können so komplexe Abhängigkeiten zwischen Daten und Aktivitäten beschrieben werden. Der Datenaustausch mit externen Datenquellen wird aber nicht betrachtet.

Datenkontrolle. Das primäre Ziel besteht darin, alle benötigten Anwendungsdaten in das Prozessmodell zu integrieren. Es wird davon ausgegangen, dass Daten innerhalb einer Aktivität mit Anwendungen ausgetauscht werden können. Daten, die für die Ausführung einer Aktivität bzw. einer Anwendung benötigt werden, müssen daher durch eine andere Aktivität im Vorfeld bereitgestellt werden.

Transaktionskontrolle. Da die Daten vollständig durch das WFMS verwaltet werden, können komplexe Bedingungen daran gestellt werden. Diese Bedingungen werden unter anderem durch den Einsatz von Zustandsmodellen sichergestellt, die für komplexe Datenobjekte angelegt werden können. Transaktionskonzepte für den Datenaustausch mit externen Datenquellen können, genau wie bei imperativen Prozessmodellen, nicht umgesetzt werden.

4.1.2 Lösungsstrategien für die Umsetzung flexibler Publikationsprozesse

In Abschnitt 2.2.3 wurden verschiedene Techniken für die Umsetzung flexibler Prozessmodelle vorgestellt. Eine Möglichkeit, flexible Prozesse zu beschreiben, die von vielen imperativen Prozessmodellen bereitgestellt werden, ist die Flexibilität durch Modellierung (siehe Abschnitt 2.2.3.1). Imperative Prozessmodelle können vereinfacht werden, indem im Vorfeld eine Konfiguration der Prozessmodelle vorgenommen wird [GAJVR08]. Kontextabhängig können so angepasste Prozessmodelle erzeugt werden, in denen alle Prozessbestandteile entfernt wurden, welche zur Laufzeit nicht benötigt werden. Ansätze, die Flexibilität durch Unterspezifikation umsetzen, können zur Laufzeit Prozessfragmente einbinden. Hierfür müssen Platzhalter im Prozessmodell angegeben werden (siehe Abschnitt 2.2.3.3). Eine spezielle Form der Flexibilität durch Unterspezifikation sind deklarative Prozessmodelle. In imperativen Prozessmodellen werden alle Ausführungsvarianten explizit modelliert. Im Gegensatz dazu wird bei einem deklarativen Ansatz versucht, mit Hilfe von Bedingungen

implizit eine Ausführungsreihenfolge von Aktivitäten zu definieren. Dabei sind alle Reihenfolgen, welche den Bedingungen genügen, korrekt. Weitere Bedingungen schränken die Menge der möglichen Ausführungsreihenfolgen ein. [Pes08, Seite 80]

Das Konzept *Flexibilität durch Anpassung* wird durch Ansätze wie z. B. *ADEPT2* [RRKD05] umgesetzt. Änderungsoperationen (z. B. Einfügen oder Löschen von Aktivitäten) ermöglichen die Anpassung der Prozessmodelle. Wird ein korrektes Prozessmodell durch die Anwendung der Änderungsoperationen erzeugt, können die Änderungen zur Laufzeit an die entsprechenden Prozessinstanzen propagiert werden (siehe Abschnitt 2.2.3.4).

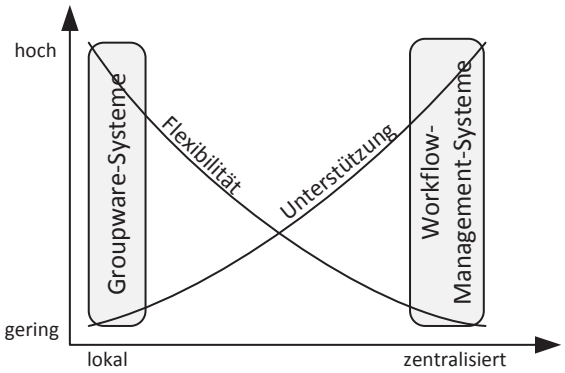


Abbildung 4.1: Kompromiss zwischen Flexibilität und Prozessunterstützung nach [Pes08, Seite 11]

Abbildung 4.1 zeigt, wie sich die Flexibilität von Modellen auf die Unterstützung von Nutzern auswirkt. Der grundlegende Unterschied zwischen den dargestellten Systemen (Groupware und WFMS) liegt im Zeitpunkt, wann Entscheidungen getroffen werden. WFMS bieten demnach ein hohes Maß an Unterstützung für den Nutzer, aber ein geringes Maß an Flexibilität. Bei Groupware-Systemen kehrt sich die Relation um. Sie bieten mehr Flexibilität aber wenig Unterstützung für den Nutzer. [Pes08, Seite 11]

Flexibilität durch Modellierung oder Konfiguration imperativer Prozessmodelle
Ein gewisses Maß an Flexibilität kann durch entsprechende Modellierungskonzepte erreicht werden. Die meisten Systeme unterstützen die parallele Ausführung von Aktivitäten. Eine Auswahl oder Iteration von Aktivitäten wird ebenfalls von den meisten Systemen unterstützt. Die verschränkte Ausführung von Aktivitäten oder Mehrfachinstanzen und das Abbrechen von Aktivitäten werden nicht von allen Systemen unterstützt [SMR+08]. Für die Anforderungen aus Abschnitt 3.3.2 ergeben sich folgende Punkte:

Prozessmodellierung. Grundsätzlich kann man Publikationsprozesse mit imperativen Prozessmodellen beschreiben. Konstrukte wie Auswahl oder auch die parallele Ausführung (gegebenenfalls mit Bedingungen) können verwendet werden, um beispielsweise Abhängigkeiten von Datenstrukturen zu beschreiben. Dabei können aber sehr komplexe Prozessmodelle entstehen, die sehr schwer wartbar sind. Die Entkopplung von anwendungsspezifischen

Prozessbestandteilen von dokumentspezifischen Bestandteilen kann durch entsprechende Strukturierung des Prozessmodells umgesetzt werden.

Prozessausführung. Ein wichtiger Aspekt bei der Modellierung von Publikationsprozessen ist die Datenabhängigkeit der Prozessmodelle. Bestimmte Entscheidungen können erst zur Laufzeit, in Abhängigkeit von den Daten, getroffen werden. Imperative Prozessmodelle bieten hierfür die Konstrukte wie Auswahl, Iteration oder Mehrfachausführung an, müssen aber alle Möglichkeiten ausmodellieren. Dies führt zu komplexen Prozessmodellen, die zur Laufzeit nicht flexibel anpassbar sind. Demnach ist es auch nicht möglich, Prozessbestandteile zur Laufzeit, in Abhängigkeit von Daten bzw. Dokumenten, zu verändern.

Konfigurierbare Prozessmodelle

Ein wesentlicher Nachteil imperativer Prozessmodelle ist die Komplexität der Prozessmodelle. Diese Komplexität wird unter anderem auch dadurch erhöht, dass kontextabhängige Entscheidungen immer zur Laufzeit behandelt werden. Viele Bereiche, wie die Auswahl von Aktivitäten oder parallele Pfade werden in Abhängigkeit von Kontextinformationen ausgeführt. Häufig stehen die Bedingungen für die Aktivierung dieser Bereiche aber schon vor der Instanziierung eines Prozessmodells fest. Konfigurierbare Prozessmodelle ermöglichen es, in Abhängigkeit dieser Bedingungen eine Anpassung im Vorfeld vorzunehmen (sog. Prozesskonfiguration). Im Prozessmodell werden nur noch die Bereiche modelliert (z. B. durch XOR-Split oder OR-Split Knoten), deren Auswahl erst zur Laufzeit entschieden werden kann.

Die Anwendung konfigurierbarer Prozessmodelle ist für die Beschreibung von Publikationsprozessen zwar sinnvoll, löst aber nicht die angesprochenen Probleme. Besonders der Publikationsprozess ist stark von Daten bzw. Dokumenten abhängig, welche bearbeitet werden. Durch die große Dynamik der Dokumentstruktur und der Dokumentinhalte können viele Entscheidungen erst zur Laufzeit getroffen werden. Das Beispiel des Teilprozesses für die Indizierung (siehe Abschnitt 3.3.1) demonstriert die Problematik anschaulich. Weil erst zur Laufzeit feststeht, welche Dokumenttypen im Dokument verwendet werden, würde die Prozessstruktur sich nicht verändern. Der Vorteil einer Konfiguration greift also nicht.

Flexibilität durch Unterspezifikation

Eine weitere Möglichkeit, Flexibilität in einem Prozessmodell zu ermöglichen, bietet die Unterspezifikation. Für die Anforderungen aus Abschnitt 3.3.2 ergeben sich folgende Punkte:

Prozessmodellierung. Im Prozessmodell sind Platzhalter definiert, die zur Laufzeit einer Prozessinstanz die Integration von Prozessfragmenten erlauben. In Abschnitt 2.2.3.3 wurde zusätzlich die Unterscheidung zwischen *late binding* und *late modeling* eingeführt. Für den Publikationsprozess führt das Konzept des *late bindings* zu keiner großen Verbesserung. Es müssen für jede mögliche Kombination von Dokumenttypen (Bild, Video, Text, etc.)

Prozessfragmente bereitstehen. Die Anzahl und Komplexität der Prozessfragmente hängt damit direkt von den Dokumenttypen ab.

Dieses Problem kann aber durch das Prinzip des *late modelings* behoben werden. Zur Laufzeit können die Prozessfragmente, entsprechend der aktuellen Dokumentstruktur und -inhalt, modelliert werden. Im Anschluss kann das Prozessfragment ausgeführt werden.

Prozessausführung. Die Platzhalter in einem Prozess werden durch dokumentspezifische Prozessfragmente ersetzt. Weil die Dokumente häufig geändert werden, müssen die Prozessfragmente, welche für einzelne Platzhalter zur Verfügung stehen, zur Laufzeit erzeugt werden. Eine Umsetzung sollte hier Platzhalter mit *late modeling* verwenden, welche in Abhängigkeit von den Dokumenten Prozessfragmente automatisch generieren. Soll das Konzept für die Modellierung von Publikationsprozessen genutzt werden, muss eine geeignete Möglichkeit gefunden werden, wie die Prozessfragmente automatisch erzeugt werden können. Autoren und andere am Prozess beteiligte Personen sollen dabei nicht mit der Modellierung geeigneter Prozessfragmente beauftragt werden. Sie sind keine Spezialisten im Bereich der Prozessmodellierung und haben oft auch kein Wissen über die Prozessstrukturen.

Flexibilität durch deklarative Beschreibungen

Eine spezielle Form der Unterspezifikation ist das deklarative Konzept. Der Vorteil deklarativer Prozessmodelle liegt darin, dass Prozessmodelle sehr flexibel umgesetzt werden können. Dies wird erreicht, indem die Ausführungsreihenfolge implizit durch Bedingungen festgelegt wird. Dadurch wird dem Nutzer ein hoher Freiheitsgrad bei der Auswahl von Aktivitäten ermöglicht. Für die Anforderungen aus Abschnitt 3.3.2 ergeben sich folgende Punkte:

Prozessmodellierung. Die Modellierung mit einem deklarativem Ansatz kann dann sehr aufwendig werden, wenn eine Abweichung bzw. flexible Ausführung nicht erwünscht oder möglich ist. Im Publikationsprozess ist die Abarbeitung anwendungsspezifischer Prozessbestandteile fest vorgegeben, sodass eine flexible Ausführung nicht erwünscht ist. Die Menge der zu definierenden Bedingungen steigt in diesem Fall stark an.

Prozessausführung. Durch den Einsatz deklarativer Prozessmodelle, wie beispielsweise *DECLARE* [PSA07], kann der Aufwand für die Modellierung in dokumentspezifischen Prozessbereichen erheblich verringert werden. Im Modell können alle Varianten beschrieben werden, die in Abhängigkeit von Daten ausgeführt werden. Eine Anpassung von Bedingungen und verwendeten Aktivitäten ist bei *DECLARE* zur Laufzeit ebenfalls möglich.

Flexibilität durch Modifikation

Eine weitere Möglichkeit, Flexibilität bei der Prozessausführung zu erreichen, bietet das Prinzip der Modifikation. Für die Anforderungen aus Abschnitt 3.3.2 ergeben sich folgende Punkte:

Prozessmodellierung. Je nach Umsetzung werden unterschiedliche Strategien vorgeschlagen, wie die Prozessinstanzen in das neue Prozessmodell überführt werden (siehe Abschnitt 2.2.3.4). Hierfür stehen eine Reihe von Änderungsoperationen zur Verfügung. Einzelne Aktivitäten oder ganze Prozessfragmente können aus einer Prozessinstanz entfernt oder hinzugefügt werden. Je nach Umsetzung wirken sich die Operationen nur lokal auf eine einzelne Instanz oder auf alle Prozessinstanzen eines Schemas aus. Teilweise kann im Modell die Menge und Art der nutzbaren Prozessfragmente eingeschränkt werden. Konformitätsregeln stellen sicher, dass alle Änderungen strukturell und semantisch korrekt sind.

Prozessausführung. Eigentlich soll das Konzept Änderungen an Prozessinstanzen ermöglichen, welche im Vorfeld nicht ersichtlich sind. Automatische, kontextabhängige Änderungen sind nicht vorgesehen, aber prinzipiell möglich (z. B. mit dem Ansatz AgentWork [MGR04]). Im Publikationsprozess beeinflussen Dokumentoperationen wie das Einfügen oder Löschen von Dokumentfragmenten die Prozessinstanzen. Weiterhin sind Änderungen immer lokal für eine Prozessinstanz gültig. Fügt ein Autor ein Bild ein, muss die entsprechende Prozessinstanz angepasst werden. Andere Prozessinstanzen dürfen in diesem Fall nicht verändert werden. Änderungen an Prozessinstanzen müssen lokal bleiben und dürfen nicht auf ein globales Prozessmodell übertragen werden.

Soll die Modifikation von Prozessmodellen eingesetzt werden, muss die Möglichkeit bestehen, Änderungsoperationen am Prozessmodell lokal einzuschränken. Außerdem dürfen nur vorher festgelegte Prozessfragmente für diese Operationen verwendet werden. Dafür müssen Konzepte entwickelt werden, um Operationen und Prozessfragmente an den Dokumentzustand zu binden. Autoren und andere am Prozess beteiligte Personen sollen auch nicht die Aufgaben übernehmen, die Prozessmodelle selbständig zu manipulieren.

4.1.3 Fazit

Imperative Prozessmodelle bieten die Möglichkeit, Publikationsprozesse im Bereich digitaler Bibliotheksanwendungen zu beschreiben. Ein wesentlicher Nachteil entsteht durch die Datenabhängigkeit der Prozessmodelle. Zum einen ist die Integration externer Datenquellen ein Problem. Auf der anderen Seite entstehen komplexe Prozessmodelle, weil kontextabhängige Entscheidungen erst zur Laufzeit getroffen werden können. Flexibilität kann z. B. durch Unterspezifikation oder Modifikation (z. B. Ad-hoc-Adaption) der Prozessinstanzen erreicht werden (siehe Abschnitt 2.2.3). Der Kontext wird hier maßgeblich durch die Struktur und den Inhalt der Dokumente beeinflusst.

Eine mögliche Lösung, Prozessmodelle so zu beschreiben, dass Arbeitsabläufe flexibler an den Anwendungskontext angepasst werden können, kann der Einsatz deklarativer Prozessmodelle bieten. In dokumentspezifischen Bereichen des Publikationsprozesses können so

weniger komplexe und flexiblere Prozessmodelle zur Anwendung kommen. Dieser Vorteil kann aber in den anwendungsspezifischen Prozessbereichen nicht umgesetzt werden, da dort Abweichungen vom Arbeitsablauf nicht erwünscht sind. Komplexe Bedingungen führen dann zu unübersichtlichen Prozessmodellen. Weiterhin kann der Einsatz deklarativer Prozessmodelle durchaus problematisch sein, weil der Aufwand und die Komplexität bei der Modellierung sehr hoch sind [PWZ+11].

Techniken, die Prozessinstanzen zur Laufzeit anpassen, erscheinen hier am sinnvollsten. Je nach Methode kann *late modeling* oder das Prinzip der Modifikation eingesetzt werden.

4.2 Architektur für flexible, datengetriebene Prozesse

Im Abschnitt 4.1 wurden verschiedene Möglichkeiten vorgestellt, wie der Publikationsprozess mit existierenden Ansätzen unterstützt werden kann. Die Ansätze wurden dahingehend untersucht, inwieweit eine vorher festgelegte Menge von Anforderungen (siehe Abschnitt 3) umgesetzt werden kann. Dabei hat sich gezeigt, dass keiner der vorgestellten Ansätze alle Anforderungen vollständig unterstützt.

Im Rahmen dieser Arbeit wird ein Konzept vorgestellt, welches die Modellierung und Ausführung von Prozessmodellen für den Publikationsprozess in digitalen Bibliotheksanwendungen unterstützt. In [SMH11b] haben wir einen Ansatz vorgestellt, der die Modellierung und Ausführung flexibler Publikationsprozesse ermöglicht. Durch die Kombination imperativer und deklarativer Konzepte in einem Prozessmodell können die Anforderungen aus dem Publikationsprozess an ein Prozessmodell unterstützt werden. Das Modell ermöglicht die Definition von strukturierten Prozessbestandteilen wie beispielsweise den Geschäftsprozess in einer digitalen Bibliothek. Daneben können dokumentspezifische Prozessbestandteile mit Hilfe deklarativer Techniken flexibel modelliert und ausgeführt werden. In [SMH11a] haben wir den Ansatz *tx+YAWL* vorgestellt, der es ermöglicht Prozessvariablen an externe Datenquellen zu binden. So kann der Datenaustausch mit einer externen Datenquelle direkt durch das WFMS kontrolliert werden. In Anlehnung an Techniken für Transaktionen, die aus der Datenbanktheorie bekannt sind, unterstützt *tx+YAWL* Eigenschaften wie Atomarität, Konsistenzerhaltung, Isolation und Dauerhaftigkeit (ACID) für den Datenaustausch. Verschiedene Transaktionstypen unterstützen die speziellen Anforderungen an langlaufende Arbeitsschritte in einem Prozess. Außerdem können Datenquellen über eine einheitliche Architektur eingebunden werden.

In Abschnitt 4.2.1 werden die Konzepte für Datenintegration vorgestellt. Das Modell für die Modellierung und Ausführung von Prozessmodellen für den Publikationsprozess wird in Abschnitt 4.2.2 vorgestellt.

4.2.1 Datenintegration: *tx+YAWL*

Die Integration externer Datenquellen kann auf unterschiedliche Weise umgesetzt werden. *tx+YAWL* soll einen einfachen Mechanismus bieten, um workflow-interne Daten mit Workflow-externen Daten zu synchronisieren. Bevor ein Überblick über die Konzepte von *tx+YAWL* gegeben wird, sollen verschiedenen Ziele für die Integration diskutiert werden.

Ziele

Ein wesentliches Ziel, welches bei der Integration externer Datenquellen verfolgt wird, ist die Synchronisation zwischen workflow-internen und Workflow-externen Daten. Workflow-interne Daten werden in Form von Prozessvariablen den jeweiligen Aktivitäten bereitgestellt. Die Variablen werden außerdem externen Services bereitgestellt und für Kontrollflussentscheidungen verwendet. Datenquellen müssen dementsprechend so integriert werden, dass sie mit einer internen Variable synchronisiert werden können. Eine solche Synchronisation sollte in die Phasen der Prozessmodellierung und der Prozessausführung integriert werden. Aktivitäten, die Zugriff auf externe Datenquellen benötigen, sollen diese integrieren, ohne den Zugriff selbst zu verwalten. Dabei ist besonders der Aufruf von Programmcode durch eine Aktivität zu vermeiden, der die Kommunikation mit der externen Datenquelle implementiert. Ein geeignete Methode um dies zu erreichen ist die Definition eines neuen Variablentyps. Datenquellen sollen dem WFMS über *externe Variablen* bereitgestellt werden. Eine externe Variable wird mit Hilfe eines (parameter-)Mappings an eine externe Datenquelle gebunden. Um einen einheitlichen Zugriff auf unterschiedliche Typen von Datenquellen zu ermöglichen, müssen alle Datenquellen eine gleiche Schnittstelle für den Datenzugriff bereitstellen. Hierfür schlagen wir eine Plug-in-Architektur vor, die jede Datenquelle durch ein Plug-in kapselt. Der Zugriff auf Plug-ins kann so vom WFMS gesteuert und verwaltet werden.

Die Integration von Datenquellen mit Hilfe von externen Variablen zieht, aufgrund der redundanten Datenhaltung, verschiedene Probleme nach sich. Weil die Daten an unterschiedlichen Orten gespeichert sind, müssen geeignete Konzepte für die Synchronisation zwischen den Datenquellen bereitgestellt werden. Das WFMS muss beispielsweise über Änderungen in der Datenquelle geeignet informiert werden. Dies ist notwendig, da die Daten nicht ausschließlich unter der Kontrolle des WFMS stehen und auch von anderen Applikationen geändert werden können. Weiterhin müssen Daten, welche durch das WFMS verändert wurden, konsistent in die Datenquelle zurückgeschrieben werden.

Ein geeignetes Mittel, den Datenaustausch mit Datenquellen zu kontrollieren, stellen Transaktionen dar. Eigenschaften wie Konsistenz, Isolation, Dauerhaftigkeit oder Korrektheit der Daten können so sichergestellt werden. Wir schlagen deshalb vor, dass Prozessmodell mit transaktionalen Eigenschaften zu erweitern. Transaktionen sollen dabei den Datenaustausch zwischen Datenquellen und WFMS kontrollieren.

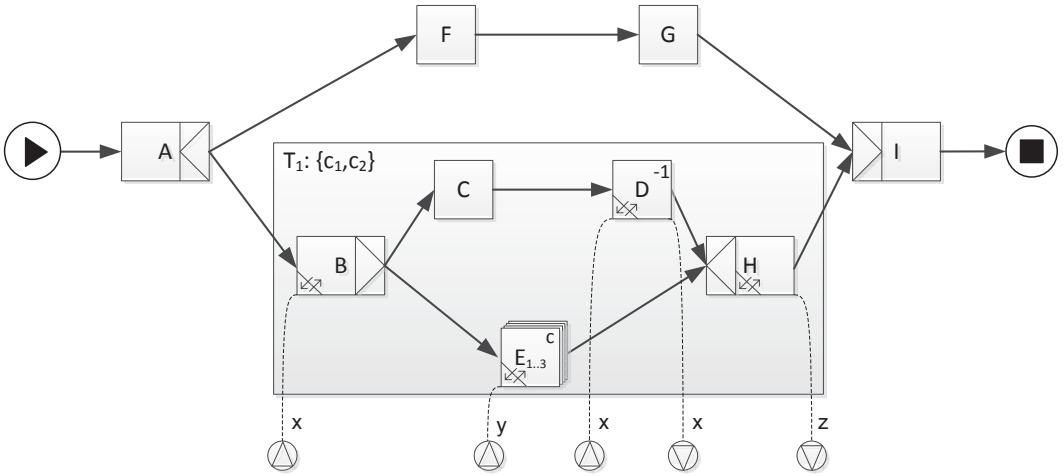


Abbildung 4.2: *tx+YAWL* Beispiel

Der Ansatz *tx+YAWL*

Der von uns vorgestellte Ansatz *tx+YAWL* ermöglicht einen konsistenten Zugriff auf externe Datenquellen. Hierfür haben wir das Prozessmodell *YAWL* so erweitert, dass ein Zugriff auf externe Datenquellen innerhalb der Kontrollflussperspektive modelliert werden kann. Um inkonsistente Datenhaltung innerhalb des WFMSs zu vermeiden, wurden Transaktionskonzepte innerhalb der Kontroll- und Datenflussperspektive eingeführt. Transaktionale Sichten ermöglichen es, Eigenschaften wie Konsistenz, Isolation, Dauerhaftigkeit und Korrektheit zu definieren. Daneben können außer atomaren Aktivitäten auch Prozessbereiche als atomare und isolierte Bausteine einer Transaktion definiert werden. So ist es auch möglich, Integritätsbedingungen auf Prozessbereichen zu definieren und zu überwachen.

Das Prozessmodell in Abbildung 4.2 zeigt die wesentlichen Konzepte von tx+YAWL.

Datenmodellierung. Externe Datenquellen können über *externe Variablen* in die Kontrollflussperspektive integriert werden. In Abbildung 4.2 stehen dazu die externen Variablen *x, y* und *z* zur Verfügung. Die Variable *x* wird von den Aktivitäten *B, E* und *D* gelesen und von der Aktivität *D* in die Datenquelle zurück geschrieben. Variable *y* wird nur von Aktivität *E* gelesen. Aktivität *H* schreibt einen Wert in die externe Variable *z*.

Aktivitäten müssen auf Werte aus externen Datenquellen über externe Variablen zugreifen.

Datenkontrolle. Die Integration der unterschiedlichen Datenquellen wird über spezielle Plug-ins ermöglicht. Ein Plug-in kapselt eine Datenquelle und kann mit Hilfe eines Mappings an eine externe Variable gebunden werden. Das Plug-in stellt einfache Methoden zum lesen ($T_i.r(x_j)$) und schreiben ($T_i.w(x_j)$) einer bestimmten Version einer Variablen bereit. Außerdem können an dieser Stelle die transaktionalen Grenzen durchgesetzt werden. Hierfür stehen die Commit-Operation ($T_i.c()$) und die Abort-Operation ($T_i.a()$) bereit.

Transaktionskontrolle. Um neben atomaren Aktivitäten auch Prozessbereiche transaktional zu unterstützen, werden transaktionale Sphären eingeführt. In Abbildung 4.2 ermöglicht es die Sphäre T_1, den Datenzugriff aller enthaltenden Aktivitäten (*B,C,D,E,H*) zu kontrollieren. Neben Eigenschaften wie Konsistenz, Isolation, Dauerhaftigkeit oder Korrektheit können auch Bedingungen (c_1, c_2) an die externen Variablen gestellt werden.

4.2.2 Flexible Publikationsprozesse

Sollen komplexe Publikationsprozesse durch den Einsatz eines WFMSs unterstützt werden, so sind die einzelnen Arbeitsschritte in ein Prozessmodell abzubilden. Ein Prozessmodell besteht dann aus Aktivitäten, die atomare Arbeitsschritte aus dem Publikationsprozess abbilden. Hier hat sich gezeigt, dass die Komplexität dieser Prozessmodelle direkt von den zu verwaltenden Dokumenttypen beeinflusst wird (siehe Abschnitt 3.3).

Unsere Beobachtungen haben gezeigt, dass die Anzahl der unterschiedlichen Dokumenttypen direkt Einfluss auf die Kontrollflussstrukturen des Prozessmodelles hat. Viele Arbeitsschritte sind abhängig vom Dokumenttyp und entsprechend im Prozessmodell abzubilden. Die verschiedenen Dokumenttypen sind dann häufig auch in unterschiedlichen Repository-Systemen bzw. Medienservern gespeichert, deren Integration ebenfalls Einfluss auf die Komplexität des Kontrollflusses hat (siehe Abschnitt 3.2).

Aus den genannten Gründen schlagen wir ein Modell vor, welches die Beziehung zwischen Dokumentstruktur und -inhalt und flexibel anpassbarer Prozessstrukturen berücksichtigt. Ein solches Modell sollte Dokumente und Prozessmodelle nicht unabhängig voneinander betrachten. Weil sich Dokumente im Laufe des Publikationsprozesses häufig ändern, sollte eine Prozessinstanz flexibel auf diese Änderungen reagieren können. Änderungen an einem Dokument, inhaltlich als auch strukturell, können auf einfache Operationen wie *einfügen*, *löschen* oder *aktualisieren* von Elementen abgebildet werden. In unserem Modell können diese Operationen direkt an Änderungsoperationen geknüpft werden, welche die Struktur der Prozessmodelle verändern. Entsprechend können Prozessbestandteile gelöscht, hinzugefügt oder aktualisiert werden.

Abbildung 4.3 zeigt, wie der Subprozess *Indizieren* aus Abschnitt 3.3 mit unserem Modell umgesetzt wird. Die Aktivität *get docs* überwacht den Zustand der Dokumente, der die Generierung eines Prozessfragments in der Aktivität *fearture extraction* beeinflusst. Für die Generierung stehen verschiedene Prozessbausteine zur Verfügung (siehe dazu Abbildung 4.3 links oben), die je nach Dokumentzustand ausgewählt werden.

Prozessmodellierung. Um die Beziehungen zwischen Dokumentstruktur und -inhalt und dem Prozessmodell zu beschreiben, schlagen wir eine Aufteilung des Prozessmodells in zwei Ebenen vor. Eine *high-level*-Prozessebene bildet den Geschäftsprozess der digitalen

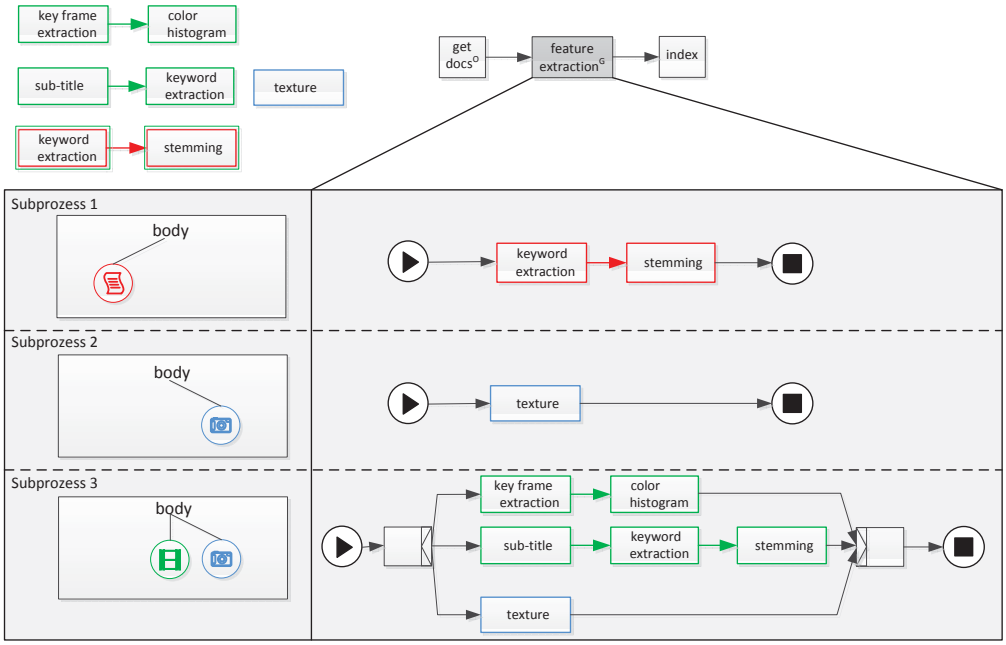


Abbildung 4.3: Ein flexibles, dokumentabhängiges Prozessmodell

Bibliotheksanwendung ab. Der Geschäftsprozess ist klar strukturiert und unterliegt nur selten Änderungen. Eine flexible Anpassung ist daher nicht notwendig. Die *low-level*-Prozessebene bildet die dokumentabhängigen Prozessbestandteile ab. Sie unterliegen häufigen Änderungen und müssen daher flexibel zur Laufzeit anpassbar sein.

Hierfür haben wir das Prozessmodell *YAWL* mit folgenden Konzepten erweitert [SMH11b]:

- Ein *Observer* liefert den aktuellen Zustand von Dokumenten.
- Ein *Generator* erzeugt einen Subprozess in Abhängigkeit vom Zustand verschiedener Dokumente.
- *Bricklets* sind wiederverwendbare Prozessbausteine, die zur Komposition von Subprozessen von einer Generator-Task verwendet werden.

Für die Modellierung des High-Level-Prozesses können alle in *YAWL* vorhandenen Modellierungskonzepte verwendet werden. Abbildung 4.3 zeigt einen Ausschnitt aus dem Publikationsprozess mit den Aktivitäten *get docs, feature extraction* und *index*. Um die dokumentabhängigen Prozessbestandteile zu modellieren, muss der neu eingeführte Aktivitätstyp *Generator* an geeigneten Bereichen in das Prozessmodell eingefügt werden. Im Beispiel soll die Merkmalsextraktion (feature extraction) flexibel an die aktuelle Dokumentstruktur angepasst werden. Die Aktivität *feature extraction* ist deshalb als Generator modelliert (siehe Index G). Der Generator verwendet mehrere *Bricklets*, um zur Laufzeit den Subprozess für die Indizierung von Dokumenten zu erzeugen. Im Beispiel stehen sechs unterschiedliche Bricklets zur Verfügung (siehe Abbildung 4.3 oben links).

Die Komposition der Bricklets wird durch eine Menge von vorher im Modell definierten Regeln gesteuert. *Konstruktionsregeln* definieren dabei eine Beziehung zwischen zwei Bricklets. Zum Beispiel soll das Bricklet *stemming* immer nach dem Bricklet *keyword extraction* ausgeführt werden. So können unterschiedliche Beziehungen zwischen einzelnen Bricklets definiert werden.

Ob ein Bricklet für die Komposition von Subnetzen in einem Generator verwendet wird, wird durch eine oder mehrere Observer-Aktivitäten gesteuert. Ein Observer ermittelt anhand vorher festgelegter Regeln, den aktuellen Zustand von Dokumenten. *Matching-Regeln* beschreiben dabei Inhalt oder Struktur der Dokumente. Je nachdem ob die Dokumente den Regeln entsprechen oder nicht, wird eine Menge von Bricklets aktiviert oder deaktiviert. In Abbildung 4.3 ist die Aktivität *get docs* als Observer definiert (siehe Index O). So kann ein Generator unterschiedliche Varianten von aktivierten Bricklets erzeugen. Beispielweise sollen die Bricklets *stemming* und *keyword extraction* aktiviert werden, wenn in einem Dokument Text enthalten ist (siehe Subprozess 1 in Abbildung 4.3). Subprozess 2 zeigt den Fall, wenn nur Bilder im Dokument enthalten sind. Subprozess 3 zeigte eine Kombination von den zwei unterschiedlichen Dokumenttypen Video und Bild. Das Beispiel in Abbildung 4.3 zeigt aber nur eine kleine Auswahl an möglichen Subprozessen.

Prozessausführung. Während der Abarbeitung einer Prozessinstanz, die mit Observern und Generatoren erweitert wurde, ermöglicht dieser Ansatz, flexibel auf Änderungen der Dokumentstruktur zu reagieren.

Mit dem Aktivitätstyp *Observer* kann der aktuelle Dokumentzustand bestimmt werden. In Abhängigkeit der vorher modellierten Matching-Regeln, können unterschiedliche Dokumentzustände erkannt werden. Dokumente aus unterschiedlichen Datenquellen können mit Hilfe der in Abschnitt 4.2.1 vorgestellten Methode zur Integration externer Datenquellen eingebunden werden.

Ist der aktuelle Zustand und Inhalt der Dokumente bestimmt, werden alle Generatoren konfiguriert. Anhand dieser Konfiguration ermöglicht unser Ansatz, zur Laufzeit einen geeigneten Subprozess zu generieren. Der Subprozess wird anhand von Bricklets und Konstruktionsregeln erzeugt. Deshalb enthält der Subprozess nur Aktivitäten, die für die Bearbeitung der aktuellen Dokumente notwendig sind. Unser Ansatz basiert also nicht auf vorher konfigurierten Prozessvarianten.

Zusammenfassung

In Kapitel 4 wurde ein Überblick über die eigenen Ansätze *tx+YAWL* und *FlexY* gegeben. In den folgenden Kapiteln werden diese Ansätze vorgestellt. In Kapitel 5 wird ein Konzept für die einheitliche Integration von Datenquellen gezeigt. Aufbauend auf diesem Konzept wird in Kapitel 6 das *tx+YAWL*-Modell vorgestellt, das eine transaktionale Integration von Datenquellen ermöglicht. In Kapitel 7 wird das *FlexY*-Modell vorgestellt.

Kapitel 5

Datenintegration

In diesem Kapitel wird das Konzept für die Integration von externen Datenquellen vorgestellt. Dabei wird der Schwerpunkt auf eine einheitliche Integration der Datenquellen gelegt. Ziel ist es, das WFMS *YAWL* so zu erweitern, dass eine einheitliche Schnittstelle für den Zugriff auf heterogene Datenquellen bereitgestellt werden kann. Mit Hilfe dieser Schnittstelle kann das WFMS den Zugriff auf die externen Datenquellen kontrollieren und überwachen. Eine Kontrolle ist aber nur möglich, wenn der Datenzugriff explizit modelliert wird. Dafür müssen Modellierungskonzepte eingeführt werden, mit denen der Datenzugriff auf externe Datenquellen beschrieben werden kann.

In Abschnitt 5.1 wird anhand eines Beispiels die Umsetzung für die Datenintegration vorgestellt. Ein Konzept für die einheitliche Integration von externen Datenquellen wird in Abschnitt 5.2 beschrieben. In Abschnitt 5.3 wird gezeigt, wie das Prozess-Metamodell erweitert werden muss, damit externe Datenquellen verwendet werden können. Wie externe Variablen in Prozessmodellen benutzt werden können, wird in Abschnitt 5.4 beschrieben. In Abschnitt 5.5 wird der vorgestellte Ansatz mit verwandten Arbeiten verglichen und es erfolgt eine Zusammenfassung des Kapitels in Abschnitt 5.6.

5.1 Externe Variablen und Datenquellen

Prozess-Metamodelle ermöglichen es, komplexe Arbeitsabläufe zu beschreiben. Es stehen eine Vielzahl unterschiedlicher Kontrollfluss-Konstrukte bereit, durch deren Anwendung eine korrekte Abarbeitungsreihenfolge unterschiedlicher Services und Ressourcen ermöglicht wird. Neben der korrekten Abarbeitungsreihenfolge nehmen Daten eine immer wichtigere Rolle im Prozessmodell ein. Die Daten werden zentralisiert in Datenbanken, Repository-Systemen oder anderen Datenquellen verwaltet und stehen so unterschiedlichen Anwendungen zur Verfügung. Sollen diese Daten innerhalb eines Prozesses verwendet werden, müssen geeignete Strategien gefunden werden, sie zu integrieren (siehe Abschnitt 3.2).

Ein grundlegendes Ziel dieser Arbeit besteht darin, ein Konzept für die Integration von externen Datenquellen in das Prozessmodell von *YAWL* umzusetzen. Wir erweitern *YAWL* derart, dass der Datenzugriff auf Datenobjekte einer externen Datenquelle direkt modelliert und durch das WFMS ausgeführt werden kann. Hierfür kann im Modell die Datenquelle und ein konkretes Datenobjekt modelliert werden. Wie und wann eine Synchronisation mit der Datenquelle stattfindet, muss vom WFMS während der Abarbeitung einer Prozessinstanz kontrolliert werden. In Kapitel 6 führen wir daher ein Transaktionsmodell ein, das einen konsistenten Datenaustausch zusichert.

Das Prozess-Metamodell wird mit einem neuen Variablentyp erweitert, der die Integration von externen Datenquellen im Prozessmodell ermöglicht. Der neue Variablentyp *externe Variable* erweitert das Prozess-Metamodell, um die Modellierung von Datenzugriffen auf externe Datenquellen innerhalb der Kontrollflussperspektive zu ermöglichen. Der Variablentyp ist so definiert, dass bereits im Modell eine Datenquelle eindeutig ausgewählt werden kann. Weiterhin kann im Prozessmodell definiert werden, ob eine Lese- oder Schreiboperation auf der externen Datenquelle ausgeführt werden soll. Eine einheitliche Integration der Datenquelle wird mit Hilfe einer Plug-in-Architektur ermöglicht. Jede Datenquelle wird durch ein eigenes Plug-in repräsentiert.

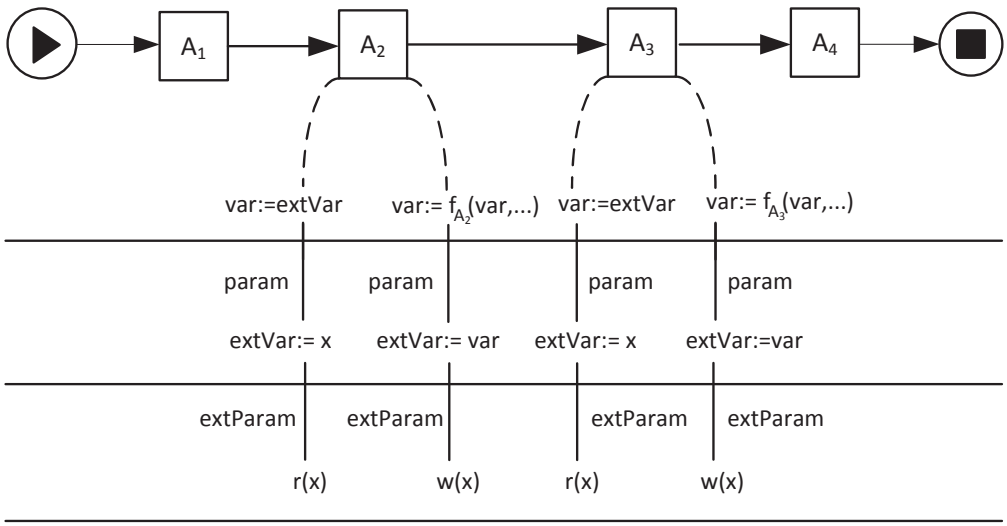


Abbildung 5.1: Datenzugriff mit externen Variablen

Das Konzept für die Integration von externen Datenquellen wird in Abbildung 5.1 gezeigt. Das Prozessmodell zeigt einen Prozess, in dem die vier Aktivitäten A_1, A_2, A_3 und A_4 sequentiell in der angegebenen Reihenfolge abgearbeitet werden. Die beiden Aktivitäten A_2 und A_3 haben jeweils eine Aktivitätsvariable mit dem Bezeichner *var*.

Die vorgestellte Erweiterung ermöglicht es, die im Beispiel definierte Variable *var* an die *externe Variable extVar* zu binden. Hierfür wird, wie in *YAWL* üblich, ein Parameter *param* verwendet. Die externe Variable *extVar* (vom Typ XML) wird ihrerseits mit Hilfe

eines Parameters *extParam* an ein Plug-in gebunden. Das Plug-in stellt eine XML-Sicht auf eine Datenquelle bereit und bietet zusätzlich eine Schnittstelle zum Ausführen von Leseoperationen $r(x)$ und Schreiboperationen $w(x)$ an. Durch die Erweiterung können Aktivitäten direkt Datenobjekte aus externen Datenquellen lesen oder nach erfolgreicher Ausführung Datenobjekte in eine Datenquelle schreiben.

Im Folgenden werden die verschiedenen Konzepte des Modells beschrieben. Um Datenquellen einheitlich in das WFMS zu integrieren, wird in Abschnitt 5.2 die Plug-in-Architektur vorgestellt. Die Datenquellen, die durch ein Plug-in gekapselt werden, können mit Hilfe von externen Variablen im Prozessmodell integriert werden. Eine Erweiterung des *YAWL*-Prozess-Metamodells mit dem Konzept der externen Variablen wird in Abschnitt 5.3 vorgestellt. Damit das *YAWL*-WFMS das Konzept unterstützt, wurde es durch ein Framework erweitert. Abschnitt 8.2 stellt die Architektur vom Data Access Framework (DAF) vor.

5.2 Einheitlicher Datenzugriff auf externe Datenquellen

5.2.1 Integration externer Datenquellen

Um innerhalb der Prozessmodelle eine einheitliche, transparente Integration von Datenquellen zu gewährleisten, muss der Datenzugriff vereinheitlicht werden. Hierfür wird eine Sprache benötigt, die alle Dokumente gleichermaßen beschreiben kann. Außerdem ist eine Anfragesprache notwendig, mit der alle Datenquellen angefragt werden können. Für den Datenaustausch zwischen WFMS und Datenquelle ist eine feste Menge an Schnittstellen notwendig.

Der Aufwand für die Integration von externen Datenquellen hängt stark von der Heterogenität der Datenquellen ab. Es kann zwischen *technischer Heterogenität* und *Interface-Heterogenität* unterschieden werden. Die *technische Heterogenität* beschreibt Unterschiede durch verschiedenen Zugriffsmethoden wie z. B. *SQL, CORBA, HTML* oder *OQL*. *Interface-Heterogenität* tritt auf, wenn Interfaces mit unterschiedlicher Mächtigkeit existieren. Manche Anfrage-Schnittstellen bieten beispielsweise keine Negation an oder beschränken Anfragen indem beispielsweise die maximale Anzahl der *Join-Operationen* vorgegeben wird. [BKLW99]

In den letzten Jahren hat sich XML immer mehr als Sprache zum Beschreiben von Datenformaten durchgesetzt. XML ermöglicht auf eine einfache Art und Weise, Dokumente mit unterschiedlichem Inhalt zu beschreiben. Dabei sind XML-Dokumente selbstbeschreibend, weil Inhalte und Struktur im gleichen Dokument dargestellt werden. Ein weiterer Vorteil von XML liegt darin, dass neben strukturierten Daten auch semistrukturierte oder dokumentzentrierte Daten dargestellt werden können. Als Anfragesprache für XML-Dokumente

steht *XQuery* bzw. *XPath* zur Verfügung. Die Dokumentstruktur von XML-Dokumenten wird durch *DTD*- oder *XML-Schema*-Beschreibungen definiert. Deshalb soll für den hier vorgestellten Ansatz XML für den Datenaustausch verwendet werden.

Damit das WFMS unter der Verwendung von XML und einer XML-Anfragesprache auf die Datenquellen zugreifen kann, müssen entweder alle Datenquellen entsprechend mit einer XML-Schnittstelle erweitert werden oder das WFMS muss so erweitert werden, dass alle Operationen auf die spezifischen Operationen der Datenquellen transformiert werden.

Weil die Erweiterung der Datenquelle aus verschiedenen Gründen nicht immer möglich ist, haben wir uns dazu entschieden, das WFMS entsprechend anzupassen. Für die Erweiterung sind zwei Ansätze möglich:

- **Ansatz 1 (Integration durch Konfiguration einer Komponente):** Eine mögliche Umsetzung ist die Implementierung einer Komponente, welche alle Anfragen entgegennimmt und sie an die entsprechende Datenquelle weiter leitet. Die Auswahl der Datenquelle wird durch eine Konfiguration vorgenommen, mit der jede Anfrage erweitert werden muss. Dieser Ansatz wird dadurch limitiert, dass die Möglichkeiten der Integration durch die Konfiguration bestimmt wird. Außerdem wird der dynamische Wechsel zwischen Datenquellen erschwert.
- **Ansatz 2 (Integration durch eine Plug-in-Architektur):** Eine sehr flexible Möglichkeit für die Integration von unterschiedlichen Datenquellen bietet eine Plug-in-Architektur. Durch den Einsatz eines Interfaces können alle Plug-ins die vom WFMS verwendeten einheitlichen Schnittstellen implementieren. Ein Plug-in implementiert die für eine Datenquelle spezifischen Operationen, um zum einen die Daten als XML bereitzustellen und zum anderen Operationen an der Datenquelle auszuführen. Ein wesentlicher Vorteil dieses Konzepts liegt darin, dass ein Plug-in die Struktur und Funktionalität genau einer Datenquelle kapselt und eine Erweiterung durch weitere Plug-ins dadurch problemlos möglich ist.

5.2.2 Einheitliche Integration von Datenquellen mit Hilfe von Plug-ins

Im Folgenden soll das allgemeine Konzept hinter der eingeführten Plug-in-Architektur beschrieben werden. Wir zeigen, wie mit Hilfe von einem Plug-in eine XML-Sicht auf eine externe Datenquelle bereitgestellt werden kann.

In Anlehnung an die Daten-Muster 14 und 15 in [REHA04] (siehe dazu auch Abschnitt 2.2.2), erweitern wir *YAWL* mit einer Plug-in-Architektur, um den Datenaustausch mit externen Datenquellen zu ermöglichen. Ein Plug-in kapselt die Datenquelle vollständig und stellt so eine neue Schnittstelle für das WFMS bereit. Abbildung 5.2 zeigt die mit unserer Architektur verfolgte Idee.

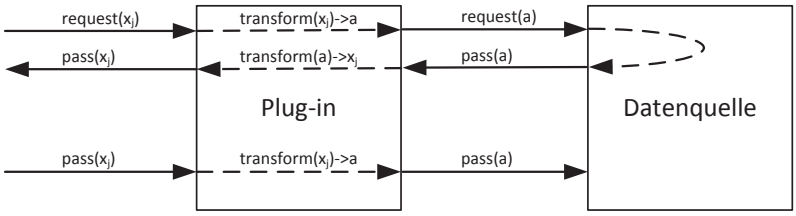


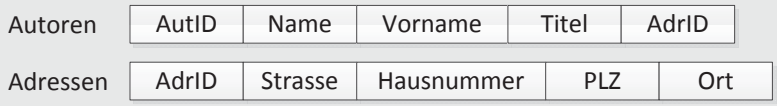
Abbildung 5.2: Konzept für ein Plug-in

Ein Plug-in kapselt die Struktur und die Funktionalität einer Datenquelle gegenüber dem WFMS. Für die korrekte und eindeutige Identifikation einer Datenquelle respektive eines Plug-ins, muss jedes Plug-in einen eindeutigen Identifikator *pID* besitzen. Um eine XML-Sicht auf eine Datenquelle zu erzeugen, muss entweder die Datenquelle selber oder das Plug-in diese bereitstellen. Dazu muss die XML-Sicht mit einem XML-Schema beschrieben werden. Das Schema ist notwendig, damit Anfragen und Ergebnisse korrekt validiert werden können. Diese Sicht auf die Daten einer Datenquelle wird mit einem XML-Schema XSD_{pID} beschrieben.

Für den Datenaustausch muss das Plug-in Lese- und Schreiboperationen bereitstellen, über die der Zugriff auf Datenobjekte möglich ist. Weil *YAWL* intern XML als Beschreibungssprache für Daten und XQuery als Anfragesprache verwendet, sollen auch Anfragen an die externen Datenquellen so gehandhabt werden. Deshalb müssen Anfragen auf die Sicht XSD_{pID} als XQuery Anfragen formuliert sein und vom Plug-in verarbeitet werden können. Das Plug-in muss eine Methode bereitstellen, um Daten aus der Datenquelle zu lesen ($request(x_j)$) und Daten in eine Quelle zu schreiben ($pass(x_j)$).

Bindet ein Plug-in eine Datenquelle ein, die keine XML-Sicht unterstützt, muss diese gegebenenfalls vom Plug-in selbst erzeugt werden. Dies setzt voraus, dass das Plug-in XQuery-Anfragen in die Anfragesprache der Datenquelle überführt. Ergebnisse, die von einer Datenquelle auf eine Anfrage hin geliefert werden, müssen entsprechend der definierten XML-Sicht (XSD_{pID}) vom Plug-in in XML transformiert werden ($transform(x_j)$). Gleiches gilt, wenn Datenobjekte aus einer Prozessinstanz heraus in eine Datenquelle geschrieben werden sollen. Hier muss das XML-Datenobjekt durch das Plug-in entsprechend in das Datenformat der Datenquelle transformiert werden.

Beispiel 5.1 (Plug-in Transformation)
Eine Anfrage, zur Ermittlung der Adresse eines Autors, soll an ein relationales Datenbanksystem gestellt werden. Das Datenbanksystem stellt die beiden Relationen *Autoren* und *Adressen* bereit. Eine *XQuery-Anfrage* muss dafür in die Ziel-Anfragesprache *SQL* überführt werden. Dazu muss das Plug-in eine entsprechende Methode ($transform(x_j)$) bereitstellen, welche die Anfrage *a* erzeugt.




```
XQuery-Anfrage:   x_j :=  //autor[id=1]/adresse
SQL-Anfrage:      a   :=  select * from autoren,adressen
                          where autoren.AutID=1 and
                                autoren.AdrID = adressen.AdrID;
```

Im nächsten Schritt muss das Ergebnis (z. B. ein *ResultSet*) in ein XML-Dokument überführt werden, welches dem XML-Schema XSD_{pID} entspricht (siehe Auflistung 5.1).

Um ein konkretes Datenobjekt identifizieren zu können, muss das Plug-in ein Datenobjekt anhand einer festgelegten *ID* bereitstellen können. Ein Plug-in stellt weiterhin Schnittstellen bereit, um eine Verbindung aufbauen zu können, für den Datenaustausch und um Serviceinformationen auszutauschen.

XML-Sichten auf Relationalen Daten
In der Literatur werden eine Reihe von Techniken diskutiert, wie Relationale Datenquellen durch XML-Sichten repräsentiert werden können. Das Framework *SilkRoute*, welches in [FKS+02] vorgestellt wird, ermöglicht die Bereitstellung von relationalen Datenquellen durch XML-Sichten. Das grundlegende Vorgehen besteht darin, dass XML-Anfragen in SQL-Anfragen transformiert werden.

Auf Probleme die bei Updates auf XML-Sichten auftreten können, die über relationale Datenquellen erzeugt wurden, geht [WRM06] ein. Es wird ein Algorithmus vorgestellt mit dem bestimmt wird, unter welchen Umständen ein Update möglich ist. In [SKS+01] wird besonderes Augenmerk auf komplexe XQuery-Anfragen gelegt, die geschachtelt sind.

5.2.3 Plug-in Schnittstellen

Damit Plug-ins reibungslos integriert werden können, müssen sie neben einem einheitlichen Datenformat auch eine einheitliche Schnittstelle bieten. Eine im Datenbankbereich weit verbreitete Lösung, unterschiedliche Datenquellen in Anwendungsprogramme zu integrieren, wird durch die *Java-Database-Connectivity-Architektur* (JDBC) umgesetzt. Wir streben eine Lösung an, die sich an den Konzepten der JDBC-API orientiert.

Wir schlagen deshalb eine Architektur vor, die es ermöglicht, Plug-ins unabhängig von der integrierten Datenquelle zu verwenden. Dazu muss jedes Plug-in drei Komponenten bereitstellen: einen *Plug-in Driver*, eine *Plug-in Connection* und ein *Plug-in Statement*.

Abbildung 5.3 zeigt die Plug-in-Architektur. Grundsätzlich muss jedes Plug-in einen Treiber (engl. *driver*) bereitstellen.

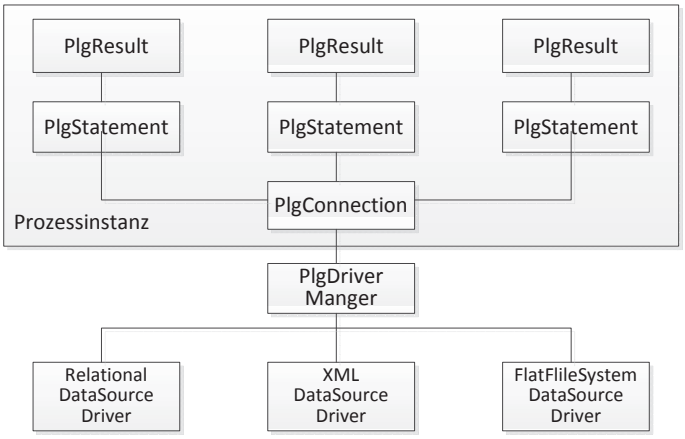


Abbildung 5.3: Plug-in-Architektur

Der Treiber stellt das Interface zur Datenquelle bereit. Er verwaltet die verschiedenen Verbindungsobjekte zu einer Datenquelle. Ein Plug-in-Treiber wird durch den Plug-in *Treiber-Manager* (*PlgDriverManager*) verwaltet.

Der Plug-in-Treiber-Manager verwaltet alle registrierten Plug-ins. Dafür stellt er ein Interface bereit, über das Plug-ins registriert werden können. Außerdem wird der Treiber-Manager dafür benutzt, Verbindungsobjekte für konkrete Plug-ins zu erzeugen und zu verteilen.

Ein Plug-in-*Verbindungsobjekt* (*PlgConnection*) stellt eine logische Verbindung zu einem Plug-in dar. Es kapselt alle Session-Informationen, welche für den Datenzugriff benötigt werden. Wenn das Plug-in Transaktionen unterstützt, können hier lokale Transaktionen gestartet und mit einem lokalen commit $T_i.c()$ oder abort $T_i.a()$ beendet werden (siehe dazu auch Kapitel 6).

Ein Plug-in-*Statement* führt die eigentlichen Lesezugriffe $r_i.(x)$ und Schreibzugriffe $w_i.(x)$ auf die Datenquelle aus und sendet sie an die Datenquelle. Hierfür verwendet das Statement den Plug-in-Treiber zusammen mit der Methode `executeQuery()`. Außerdem wird innerhalb eines Statements die Verarbeitung der XQuery-Anfrage vorgenommen, welche im Prozessmodell angegeben ist. Bei Anfragen wird der XQuery-Ausdruck in eine von der Quelle unterstützte Anfragesprache umgeformt. Stellt die Datenquelle Ergebnisse nicht als XML-Fragment bereit, wird es im Statement entsprechend transformiert. Je nach Datenquelle und Datenformat kann diese Umformung sehr komplex sein (siehe dazu auch Abschnitt 5.2.2).

Das Ergebnis (*PlgResult*) einer Anfrage wird durch ein Statement erzeugt. Es wird direkt an die entsprechende Aktivität ausgeliefert.

5.3 Externe Variablen und Workflow-Aktivitäten

In Abschnitt 5.2 wurde die Grundvoraussetzung für die Benutzung von Datenobjekten aus externen Datenquellen durch die Einführung einer Plug-in-Architektur geschaffen (siehe Abschnitt 5.2). Um diese Datenquellen in einen Prozess einbinden zu können, muss das Prozess-Metamodell so erweitert werden, dass Aktivitäten transparenten Zugriff auf die Datenobjekte erhalten (vgl. Anforderung 3.2).

Wir erweitern deshalb das Prozess-Metamodell von *YAWL* um einen neuen Variablentyp, welcher die Integration von externen Datenquellen ermöglicht. Eine *externe Variable* soll ein Datenobjekt aus einer externen Datenquelle im Prozessmodell bereitstellen. Dafür muss die externe Variable mittels externem Parameter an ein Plug-in gebunden werden. So besteht die Möglichkeit, Datenobjekte aus einer externen Datenquelle, die mit Hilfe eines Plug-ins dem WFMS bereitgestellt werden, in das Prozessmodell einzubinden. Im Folgenden sollen die Erweiterungen des Prozess-Metamodells erläutert werden.

5.3.1 Externe Variablen

Eine *externe Variable* ermöglicht den Datenzugriff auf ein Plug-in innerhalb einer Prozessbeschreibung zu modellieren. Jede *externe Variable* repräsentiert genau ein (komplex strukturiertes) Datenobjekt aus einer externen Datenquelle. Um die Variable mit einem Wert zu belegen, muss eine Datenquelle bzw. das entsprechende Plug-in ausgewählt werden, das die Daten zur Verfügung stellen kann. Außerdem muss eine Auswahlmöglichkeit für ein Datenobjekt bestehen, falls das Plug-in mehr als ein Datenobjekt bereitstellen kann. Eine externe Variable muss also an eine Datenquelle gebunden werden, und die Angabe eines identifizierenden Attributes für die Auswahl eines Datenobjekts muss möglich sein.

In Definition 5.1 haben wir das formale Modell von *YAWL* (siehe Anhang A.2) deshalb erweitert.

Definition 5.1 (Erweiterungen des *YAWL*-Modells)
— Erweiterung der *YAWL*-Spezifikation —

- $VarType: VarID \rightarrow \{Net, Task, MI\} \cup \{Ext\}$ beschreibt den Sichtbarkeitsbereich von Variablen. Der Variablentyp *Ext* stellt eine Erweiterung zum Standard-*YAWL*-Modell dar und beschreibt eine *externe Variable*.
- *PlugInID* ist die Menge aller Plug-ins, die in einem Modell verwendet werden.
- *extParamID* ist die Menge aller *externen Parameter* in einem Netz. Ein externer Parameter beschreibt die Abbildung zwischen einer externen Variable und einem Plug-in.

— Erweiterung des *YAWL*-Datenflussmodells —

- $extParamVar: extParamID \rightarrow VarID^{Ext} \times PlugInID$ ist eine Funktion, welche zu einer externen Variable das dazugehörige Plug-in bestimmt.
- $extPar: extParamID \rightarrow extMapping$ definiert das Mapping zwischen einer externen Variable und einem Plug-in.

In einem ersten Schritt führen wir den neuen Variablentyp *Ext* ein, sodass gilt $VarID^{Ext} = \{v \in VarID \mid VarType(v) = Ext\}$. *Ext* beschreibt alle Variablen, die an ein Plug-in gebunden sind. Eine Variable vom Typ *Ext* kann nicht direkt innerhalb eines Netzes oder einer Aktivität verwendet werden. Die Menge von Plug-ins in einem Netz wird durch (*PlugInID*) beschrieben. Jedes Plug-in hat eine eindeutige ID *pID* über die es adressiert werden kann. Die Abbildung zwischen einem Plug-in und einer externen Variable wird durch einen *externen Parameter* beschrieben. Die Menge von externen Parametern in einem Netz ist durch *extParamID* gegeben. Ein externer Parameter beschreibt, wie eine externe Variable auf ein Plug-in abgebildet wird.

5.3.2 Ein Mapping-Ansatz - *map*

In den meisten WFMS kann der Nutzer eigene komplexe XML-Schema-Typen XSD_{pm} definieren, die als Datentypen für Variablen im Prozessmodell verwendet werden können. Ein Plug-in beschreibt die Sicht auf die Datenquelle hingegen mit einem eigenen XML-Schema XSD_{plg}. Weil das XML-Schema XSD_{plg} eine komplexe Datenstruktur beschreiben kann, muss man die Frage stellen, ob dieses auch vollständig im WFMS bereitgestellt werden soll. Wir haben deshalb zwei Methoden diskutiert, welche „Sicht“ eine externe Variable auf ein Plug-in haben soll. Die zwei Methoden werden im Folgenden noch einmal kurz vorgestellt.

Methode 1 (ein globales Repository):
Eine mögliche Umsetzung besteht darin, dass der Datentyp XSD_{plg} im globalen Repository für XML-Schema-Typen registriert wird, welches vom WFMS verwendet wird. So ist es relativ einfach, jeder externen Variable den Datentyp des referenzierten Plug-ins zuzuweisen. Aufwendige Transformationen bei der Integration eines Plug-ins entfallen dabei. Nachteilig zeigt sich bei dieser Lösung die Einschränkung bei der Prozessmodellierung. Es können nur externe Variablen erzeugt werden, die genau dem bereitgestellten Datentyp eines Plug-ins umsetzen. Werden unterschiedliche Sichten auf eine Datenquelle benötigt, müssen auch unterschiedliche Plug-ins erstellt werden. Es besteht keine Möglichkeit, Datentypen einzuschränken oder gegebenenfalls zu erweitern.

Methode 2 (eine Sicht):
Eine andere Vorgehensweise ist die Definition einer *Sicht* auf ein Plug-in, welches durch den Schema-Typ XSD_{plg} repräsentiert wird. Sichten sind im Allgemeinen durch ein eigenes Schema und eine Berechnungsvorschrift definiert. Demnach wird jeder externen Variable ein beliebiger XML-Schema-Typ XSD_{extV} zugewiesen, der als Schema der Sicht aufgefasst werden kann. Der Schema-Typ XSD_{extV} ist im globalen Repository für XML-Schema-Typen des WFMS registriert. Die Berechnungsvorschrift für die Sicht ist durch einen *XQuery*-Ausdruck beschrieben, der eine Anfrage auf die XML-Sicht des Plug-ins darstellt. Sie liefert als Ergebnis ein XML-Fragment vom Typ XSD_{extV}, welches der externen Variable als Datenobjekt zugewiesen wird. Vorteile dieser Methode liegen darin, dass das Prozessmodell konzeptionell weniger abhängig von der Umsetzung eines Plug-ins ist. Änderungen an Datenquellen oder Plug-ins können mit weniger Aufwand umgesetzt werden, als bei Methode 1. Außerdem kann durch die Nutzung eines Sichtenansatzes die logische Datenunabhängigkeit im Prozessmodell erhöht werden.

Für diese Arbeit soll ein Ansatz gewählt werden, welcher der Vorgehensweise von Methode 2 entspricht. Im Folgenden soll die Umsetzung diskutiert werden.

Probleme mit Sichten
Werden Sichten auf Daten erzeugt, entstehen eine Reihe von bekannten Problemen, wenn Änderungsoperationen auf den Sichten ausgeführt werden sollen. Aus dem Bereich Datenbanken sind z. B. Probleme mit *Projektionssichten, Selektionssichten* und *Verbundsichten* bekannt. Ziel bei der Nutzung von Sichten ist es, unterschiedliche Kriterien bei Update-Operationen zu gewährleisten. Wichtig sind Effektkonformität und Konsistenzerhaltung. Das

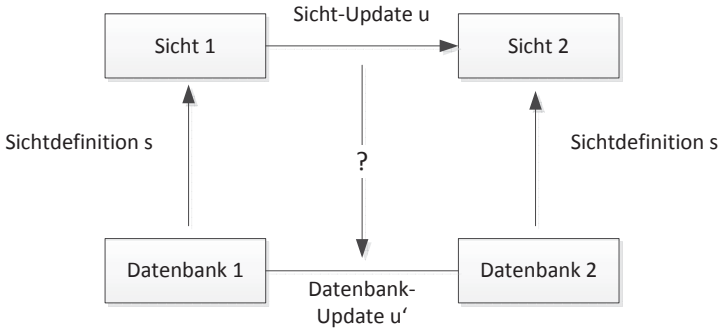


Abbildung 5.4: Sicht-Update-Transformation nach [SSH07]

generelle Vorgehen bei Update-Operationen auf Sichten, wird in Abbildung 5.4 dargestellt. Der Nutzer arbeitet auf der virtuellen Relation *Sicht 1*, die durch die Sichtdefinition *s* auf der *Datenbank 1* berechnet wird. Durch die Sicht-Update-Operation *u* nimmt der Nutzer eine Änderung an der *Sicht 1* vor, die zu der neuen *Sicht 2* führen soll. Die Sicht-Update-Operation *u* kann aber nicht direkt auf der *Sicht 1* ausgeführt werden. Die Operation muss in Datenbank-Update-Operationen auf der Datenbank 1 übersetzt werden. Das Datenbanksystem muss

die Sicht-Update-Operation in Datenbank-Update-Operationen transformieren, so dass eine *Datenbank 2* erzeugt wird. Wird nun die Sichtdefinition *s* auf die *Datenbank 2* angewendet, entsteht ebenfalls die *Sicht 2*. Eine Sicht-Update-Operation ist effektkonform, wenn gilt:

$$u(s(db_1)) = s(u'(db_1))$$

Eine Sicht-Update-Operation ist konsistenzerhaltend, wenn nach deren Anwendung die Datenquelle in einem konsistenten Zustand ist.

Wird durch den *XQuery*-Ausdruck ein XML-Fragment e mit $e \in XSD_{extV}$ erzeugt und es gilt $XSD_{extV} \subset XSD_{plg}$ kann das Hinzufügen von Elementen zu Problemen führen. Beispielsweise tritt ein Problem auf, wenn ein Element als zwingend definiert ist, aber nicht in der Sicht enthalten ist. Selektiert der *XQuery*-Ausdruck bestimmte XML-Fragmente aufgrund einer Bedingung in der *where*-Klausel aus, kann dies zu Problemen führen, wenn Änderungen an Attributen vorgenommen werden, die in der where-Klausel verwendet werden.

Ein weiteres Konzept von *XQuery* ist die Möglichkeit, durch die Anwendung der *return*-Klausel in einem *XQuery*-Ausdruck, Ergebnisse einer Anfrage in beliebige XML-Strukturen zu überführen. In vielen Anwendungsgebieten ist die *return*-Klausel deshalb ein nützliches Werkzeug, um komplexe Verarbeitungen auf Ergebnissen zu vermeiden. Unabhängig davon, macht eine solche Transformation das Ausführen von Updates auf den originalen Daten, die durch das Plug-in bereitgestellt werden, schwierig. Wir schränken deshalb die Nutzung der *return*-Klausel dahingehend ein, dass für das Ergebnis des *XQuery*-Ausdrucks gelten muss: $XSD_{extV} \subseteq XSD_{plg}$.

Ein anderes Instrument, welches aus dem Bereich der relationalen Datenbanken bekannt ist, ist die *Join*-Operation. *XQuery* bietet die Möglichkeit an, Daten aus unterschiedlichen Dokumenten miteinander zu verbinden. So können komplexe Sichten über mehrere Dokumente bzw. Datenquellen erzeugt werden. Erzeugt man Verbundsichten, können verschiedene Probleme beim Ändern der Sicht auftreten. Update-Operationen müssen in Teil-Operationen für die beteiligten Datenquellen übersetzt werden. Dies ist häufig nur mit hohem Aufwand möglich, teilweise können auch keine Teil-Operationen angegeben werden. Im Rahmen dieser Arbeit soll deshalb eine Kombination mehrerer Plug-ins in einem *XQuery*-Ausdruck nicht gestattet sein. Deshalb sind *Joins* ebenfalls nicht erlaubt.

In Datenbanksystemen werden die restriktiven Einschränkungen für Sicht-Updates im SQL:2003 Standard abgeschwächt. Unter bestimmten Voraussetzungen sind z. B. Updates auf *Join-Sichten* erlaubt. Auch das Hinzufügen von Elementen in eine *Join-Sicht* kann unter bestimmten Umständen erlaubt werden. Die erlaubten Update-Operationen sind aber nicht effektkonform [SSH07].

Das Ziel dieser Arbeit ist es nicht eine automatische Generierung von XML Sichten für Plug-ins bereitzustellen. Sichten und die darauf erlaubten Update-Operationen müssen durch den Nutzer definiert werden. Wir beschränken uns deshalb auf die Definition von Schnittstellen für die Plug-in-Architektur.

Aus den bisherigen Betrachtungen ergibt sich folgende Definition für eine Berechnungsvorschrift für eine Sicht auf ein Plug-in:

Definition 5.2 (Berechnungsvorschrift Sicht)
Ein Mapping *extMapping* ist definiert als $(pId, distKey, map, r_p, w_p)$, dabei ist

- *pId* eine Referenz auf ein Plug-in, welches eingebunden werden soll,
- *distKey* ein Schlüsselattribut, welches genau ein (komplex strukturiertes) XML-Fragment in der Datenquelle identifiziert,
- *map* ein *XQuery*-Ausdruck, der die Transformation zwischen der externen Variable und der Datenquelle, die durch ein Plug-in (mit der ID *pID*) gekapselt wird, beschreibt,
- r_p definiert als eine lokale Lese-Strategie für die externe Variable, und
- w_p definiert als eine lokale Schreib-Strategie für die externe Variable.

Mit Hilfe des Mappings *extMapping* wird eine Sicht auf genau ein Plug-in definiert. Hierfür muss eine Plug-in ID *pID* angegeben werden, welche ein Plug-in referenziert. Die eigentliche Berechnungsvorschrift wird durch die *Mapping-Vorschrift map* beschrieben. *map* ist eine *XQuery*-Anfrage für deren Ergebnis $XSD_{extV} \subseteq XSD_{plg}$ gelten muss. Das Attribut *distKey* kann verwendet werden, um ein bestimmtes Datenobjekt in einer Datenquelle zu identifizieren. Der Wert kann einerseits bereits zur Modellierungszeit festgelegt werden, andererseits kann er auch durch den Wert einer anderen Prozessvariable belegt werden. So kann dynamisch, zur Laufzeit einer Prozessinstanz, dass Mapping beeinflusst werden. Beispielsweise ist die ID für einen Autor erst zur Laufzeit einer Prozessinstanz bekannt. Sie kann erst durch das System festgestellt werden. Dabei legen Lese- und Schreibstrategien im Prozessmodell fest, wann eine externe Variable mit der Datenquelle abgeglichen werden soll. In den folgenden zwei Unterabschnitten werden die verwendeten Lese- und Schreibstrategien betrachtet.

5.3.3 Lese- und Schreibstrategien

Verschiedene Anwendungsszenarien erfordern, dass der Zeitpunkt für die Synchronisation zwischen externer Variable und Plug-in festgelegt werden kann. Um eine bessere Kontrolle über die Lese- und Schreiboperationen zu bekommen, führen wir Lese- und Schreibstrategien ein und unterscheiden zwischen normalem (*immediate*) und transaktionalem (*consistent*) Datenzugriff. Die unterschiedlichen Strategien sollen eine bessere Kontrolle über diese Operationen ermöglichen.

Eine Lesestrategie r_p definiert, wann eine Kopie (bzw. eine externe Variable) aktualisiert werden oder isoliert von externen Änderungen der Originaldaten bleiben soll. Wir unterscheiden zwei Lesestrategien:

- Die Strategie *immediate read* erfordert für jeden Lesezugriff ein update der externen Variable mit der Datenquelle. Die Aktualisierung soll dabei unabhängig von der transaktionalen Strategie stattfinden.
- Die Strategie *consistent read* erfordert eine transaktionale Sphäre, in welcher der Datenzugriff kontrolliert stattfinden kann. Innerhalb der transaktionalen Sphäre muss ein konsistenter Datenzugriff gewährleistet sein, der die korrekte Version des Datenobjekts für den Lesezugriff bereithält.

Eine Schreiboperation auf einer externen Variable schreibt Datenobjekte in eine Datenquelle zurück. Wann eine Schreiboperation durchgeführt wird, wird durch die Schreibstrategie w_p definiert. Die Schreibstrategie bestimmt, ob das WFMS Änderungen an einer externen Variable sofort an die Datenquelle weiterreicht oder ob die Operation verzögert wird. Hierfür führen wir zwei Schreibstrategien ein.

- Die Strategie *immediate write* gibt alle Änderungen an einer externen Variable sofort an die Datenquelle weiter.
- Die Strategie *consistent write* erfordert eine transaktionale Sphäre, in welcher die Operation kontrolliert stattfinden kann. Innerhalb der transaktionalen Sphäre wird entschieden, ob die Operation die Daten unverzüglich in die externe Datenquelle zurückschreibt, oder ob das WFMS die Operation verzögert.
- Mit der Strategie *no write* können schreibgeschützte (read only) Variablen simuliert werden. Sie können beispielsweise für Transaktionen verwendet werden, die als *read only* definiert sind.

5.3.4 Lokale Integritätsbedingungen

Werden externe Datenobjekte in einem WFMS benutzt, erfordert dies die Überprüfung von Integritätsbedingungen. Dies ist notwendig, weil die Daten nicht unter der Kontrolle des WFMSs stehen. Auf dieser Ebene werden deshalb lokale Integritätsbedingungen eingeführt. Eine lokale Integritätsbedingung kann für jede externe Variable angegeben werden. Sie wird für jede externe Variable verwaltet und überprüft.

Die lokale Integritätsbedingung kann mit Hilfe der *where*-Klausel einer *XQuery*-Anfrage (*map*) umgesetzt werden. Zugelassen sind Vergleiche zwischen Variablen und zwischen Variablen und Konstanten. *XQuery* bietet auch die Möglichkeit, orthogonal Anfragen in die *where*-Klausel einzubinden. Dieser Fall wird in der Arbeit nicht betrachtet, weil auf diese Weise z. B. auch Verbundoperationen umgesetzt werden können. Ziel der Arbeit ist

es, die Umsetzung von XML-Sichten mit Hilfe bekannter Konzepte umzusetzen, ohne auf Spezialfälle einzugehen. Es sollen keine Lösungsstrategien für die Umsetzung von XML-Sichten auf beliebige Datenquellen bereitgestellt werden.

Wenn eine Integritätsbedingung verletzt wird, findet eine geeignete Ausnahmebehandlung statt. In Abhängigkeit davon, in welchem transaktionalen Kontext die Ausnahme aufgetreten ist, werden unterschiedliche Ausnahmebehandlungen durchgeführt. Dies kann beispielsweise das Zurücksetzen einer Aktivität bedeuten oder das Ausführen einer kompensierenden Aktivität oder Transaktion (siehe Kapitel 6).

Dynamische Integritätsbedingungen müssen im Rahmen einer transaktionalen Sphäre überprüft werden. Dies wird in Kapitel 6 betrachtet.

5.4 Einbinden von externen Variablen

Abbildung 5.5 zeigt ein Prozessmodell, in dem die Aktivitäten B, E, D und H auf die externen Variablen x, y und z zugreifen. Eine externe Variable kann nicht direkt innerhalb eines Netzes

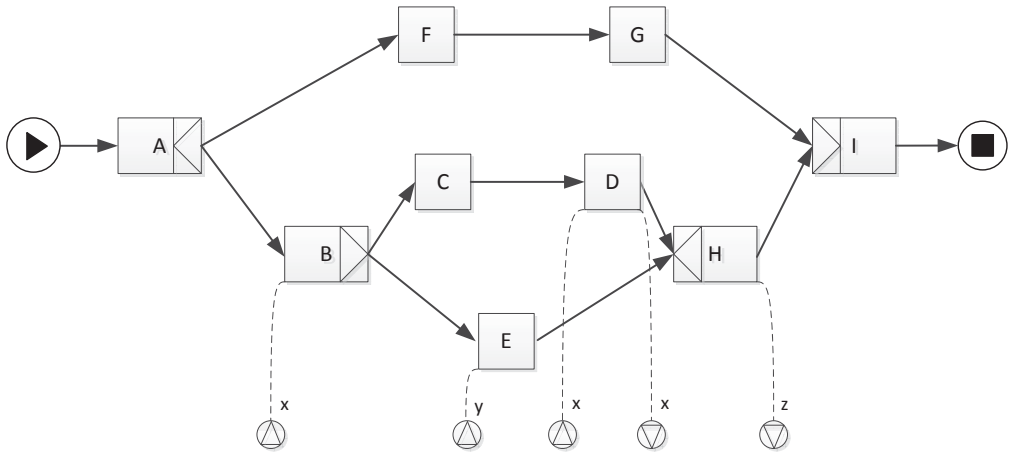


Abbildung 5.5: Prozessmodell mit einer Erweiterung für externe Variablen

oder einer Aktivität verwendet werden. Wir schlagen vor, das Konzept der Input- und Output-Parameter von *YAWL* zu verwenden. Soll eine Aktivität eine externe Datenquelle nutzen, so muss eine Aktivitätsvariable mit Hilfe eines Input- oder Output-Parameters an eine externe Variable gebunden werden. Die Notation im Prozessmodell (in Abbildung 5.5) ist eine vereinfachte Darstellung.

Ein Dreieck, bei dem die Spitze nach oben zeigt, beschreibt eine Leseoperation auf einer externen Variable. Aktivität B liest z. B. den Wert der externen Variable x. Ein Dreieck, bei dem die Spitze nach unten zeigt, beschreibt eine Schreiboperation auf einer externen Variable. Aktivität D schreibt z. B. den Wert der externen Variable x. In beiden Fällen zeigt

die vereinfachte Notation nicht, welcher Aktivitätsvariable die externe Variable zugewiesen wird.

```
1 <! ELEMENT autor (name, anschrift, ...) >
2 <! ELEMENT name (vorname, nachname, titel) >
3 <! ELEMENT anschrift (strasse, hausnummer, plz, ort) >
4 ...
```

Auflistung 5.1: Ausschnitt einer DTD für die Ausgabe eines Plug-ins

```
1  <autor>
2    <name>
3      <vorname>Hans</vorname>
4      <nachname>Mustermann</nachname>
5      <titel>Dr.</titel>
6    </name>
7    <anschrift>
8      <strasse>Albert-Einstein-Strasse</strasse>
9      <hausnummer>22</hausnummer>
10     <plz>18055</plz>
11     <ort>Rostock</ort>
12   </anschrift>
13   ...
14 </autor>
```

Auflistung 5.2: XML-Fragment als Ausgabe eines Plug-ins

```
1 for     $var in //autor
2 where   $var/name/nachname/text()=?
3         and $var/name/vorname/text()=?
4 return  $var/anschrift
```

Auflistung 5.3: Beispielanfrage für ein Mapping

Beispiel 5.2 (Plug-in Mapping)
Auflistung 5.2 zeigt ein einfache XML-Fragment, das durch ein Plug-in erzeugt wird. Es genügt der DTD in Auflistung 5.1, die einen Ausschnitt aus einer DTD darstellt. Das Fragment stellt einen Datensatz für einen Autor dar, der z. B. einen Namen und eine Anschrift als komplexe Elemente enthält. Auf dieses XML-Fragment kann das in Auflistung 5.3 gezeigte Mapping in Form einer *XQuery*-Anfrage angewendet werden. Es liefert entsprechend ein XML-Fragment für die Anschrift eines Autors. Der Nachname und der Vorname müssen zur Laufzeit aus den Prozessdaten entnommen werden.

Ein Mapping *m* müsste wie folgt definiert werden:

- *pID*: z. B. address_plug_in
- *map*: der XQuery-Ausdruck aus Auflistung 5.3
- *distKey*: aus dem komplexen Attribut *Name*, das durch das Plug-in entsprechend in das Mapping eingesetzt wird.

```
1  <name>
2    <vorname>Hans</vorname>
3    <nachname>Mustermann</nachname>
4  </name>
```

- r_p: *immediate*
- w_p: wird in diesem Fall nicht benötigt.

5.5 Diskussion

In der Literatur finden sich viele Ansätze, die Konzepte für die Integration von Daten in Prozessmodelle vorschlagen. Diese Ansätze können in zwei Kategorien unterteilt werden. In datenzentrierten Prozessmodellen wird versucht, die strikte Trennung zwischen Datenfluss und Kontrollfluss aufzuheben. Der Aufruf von Aktivitäten wird an den Zustand von Daten gebunden (siehe Abschnitt 1.3.2.1). Hier werden beispielhaft *PHILharmonicFlows* [KR09, KR11] und das *Case Handling* [AWG05] betrachtet. Die zweite Kategorie von Ansätzen soll eine bessere Integration externer Datenquellen im Kontrollfluss ermöglichen. Hier steht eine einheitliche Integration von unterschiedlichen Datenquellen im Vordergrund. Wir betrachten den Ansatz *Data Plug-ins*, der in [EL04] vorgestellt wird und den Ansatz *SIMPL* aus [RRS+11].

PHILharmonicFlows definiert komplexe Abhängigkeiten zwischen Datenobjekten und Aktivitäten [KR09, KR11] (siehe auch Abschnitt 1.3.2.1). Ziel ist insbesondere eine bessere Integration von Datenabhängigkeiten in das Prozessmodell. Sie werden zur Koordinierung der Prozessinstanzen eingesetzt. So ist eine datenabhängige Aktivierung von Prozessschritten möglich. Externe Datenquellen werden in *PHILharmonicFlows* nicht betrachtet. Die Eingabe von Daten wird entweder mit Formularen oder über Aktivitäten umgesetzt. Komplexe Multimediadokumente können mit diesem Ansatz nur durch den Aufruf von Aktivitäten verarbeitet werden. Eine einheitliche Integration der Datenquellen ist nicht möglich.

Case Handling beschreibt ein Konzept für datenzentrierte Prozessmodelle [AWG05]. Aktivitäten in einem *Case* werden in Beziehung zu atomaren Datenelementen beschrieben, die entweder *frei, optional* oder *zwingend* sind. Um einen Prozessschritt (Aktivität) zu beenden, müssen alle zwingenden Datenelemente mit einem Wert belegt sein. So können Arbeitsabläufe wesentlich flexibler beschrieben werden. Aktivitäten können beispielsweise parallel ausgeführt werden, wenn Daten bereits teilweise vorliegen. Der Fokus beim Case Handling liegt auf der flexiblen, datengetriebenen Aktivierung von Prozessschritten. Daten stehen in Form von Formularen bereit, deren Elemente atomar sind. Die Integration externer Datenquellen wird nicht explizit unterstützt. Der Datenaustausch mit externen Systemen

muss mit Hilfe von Aktivitäten umgesetzt werden. Der vorgestellte Ansatz erlaubt weiterhin nur, dass immer nur genau eine Person an einem Case arbeiten darf. Im Publikationsprozess ist eine gleichzeitige Bearbeitung durch mehrere Personen aber zwingend notwendig. Das System bietet keine Möglichkeit an, globale Integritätsbedingungen oder transaktionale Konzepte umzusetzen, um Inkonsistenzen zwischen den redundant gehaltenen Daten zu vermeiden.

Data Plug-ins ist ein Framework für den Datenzugriff auf externe Datenquellen, das Eder et al. in [EL04, EL05b, EL05a] vorstellen (siehe auch Abschnitt 1.3.2.1). Das Framework ist dem hier beschriebenen Ansatz sehr ähnlich. Die Integration der externen Datenquellen in die Kontrollflussperspektive basiert auf XML. Die Datenquellen werden ebenfalls mit Hilfe von Plug-ins integriert, welche durch einen Service eingebunden werden. Im Gegensatz zu *tx+YAWL* (siehe Kapitel 6) können Integritätsbedingungen nur auf einzelne Variablen gestellt werden. Der hier verfolgte Ansatz, einen konsistenten Datenzugriff zu ermöglichen, kann nicht umgesetzt werden. Lese- und Schreiboperationen sind unter Kontrolle des Nutzers und müssen bereits im Prozessmodell angegeben werden. Dafür kann der Nutzer verschiedene Richtlinien definieren, welche die Synchronisation mit den Datenquellen beschreiben. Entsprechend werden keine globalen Integritätsbedingungen oder transaktionalen Konzepte unterstützt, um Inkonsistenzen zwischen den redundant gehaltenen Daten zu vermeiden.

SIMPL ist ein erweiterbares Framework, das eine Abstraktion für Datenmanagement und -Vorsorge in Simulations-Workflows bereitstellt [RRS+11]. Ähnlich zu dem hier vorgestellten Ansatz werden externe Datenquellen eingeführt. Datenquellen führen *data management commands* aus, die z. B. aus SQL-Anfragen bestehen können. Sie können *Datencontainer* definieren, in denen Datenkollektionen verwaltet werden. Die *data set variable* beschreibt eine Prozessvariable, welcher Daten aus einem Datencontainer zugewiesen werden können. Das Konzept ist mit der externen Variablen vergleichbar. Für Simulations-Workflows müssen die Daten häufig direkt verarbeitet werden. Deshalb können in SIMPL externe Datenquellen nur innerhalb spezieller Aktivitäten verwendet werden. Das Prozessmodell wird bereits während der Phase der Modellierung mit diesen Aktivitäten erweitert. Dieses Vorgehen hat zur Folge, dass die Anforderungen nur teilweise umgesetzt werden können. Außerdem werden keine Konzepte für einen konsistenten Datenzugriff bereitgestellt (siehe Kapitel 6).

5.6 Zusammenfassung

In Kapitel 5 wurde die Grundlage für einen konsistenten Datenzugriff auf externe Datenquellen in Form eines Frameworks gelegt. Mit Hilfe des *Data Access Frameworks* können unterschiedliche externe Datenquellen innerhalb der Kontrollflussperspektive bereitgestellt werden. Dazu wurden das Konzept der *externen Variable* eingeführt. Eine externe Variable

kann innerhalb der Kontrollflussperspektive von Aktivitäten verwendet werden, um auf externe Daten zuzugreifen. Dazu wird die externe Variable an ein Plug-in gebunden. Plug-ins kapseln externe Datenquellen und beschreiben gleichzeitig eine einheitliche Schnittstelle für externe Datenquellen. Die Plug-ins werden durch das Framework verwaltet.

Durch das Framework ist der Zugriff auf externe Datenquellen möglich, ohne dass der Kontrollfluss durch zusätzliche Aktivitäten für den Datenzugriff erweitert werden muss. Weiterhin ist das Framework durch seine Architektur einfach zu erweitern. In Kapitel 6 wird das Framework mit einer transaktionalen Komponente erweitert. So können Inkonsistenzen zwischen den redundant gehaltenen Daten vermieden werden.

Kapitel 6

Mehrebenen-Transaktionsmodell

In Kapitel 5 wurde eine *YAWL*-Erweiterung zur Integration externer Datenquellen in ein Prozessmodell vorgestellt. Externe Datenquellen werden mit Hilfe einer Plug-in-Architektur einheitlich in das WFMS integriert. Durch den neuen Variablentyp *externe Variable* kann der Datenzugriff direkt im Prozessmodell modelliert werden.

In diesem Kapitel wird das Mehrebenen-Transaktionsmodell *tx+YAWL* vorgestellt, das einen konsistenten Datenaustausch mit externen Datenquellen für WFMSe erlaubt. Dazu werden in Abschnitt 6.1 zunächst die an ein Transaktionsmodell zu stellenden Anforderungen beschrieben. Diese ergeben sich aus den spezifischen Eigenschaften der Datenintegration, wie sie in Kapitel 5 vorgestellt wurden, und speziellen Eigenschaften von Prozessmodellen, wie beispielsweise deren geschachtelte Struktur. In Abschnitt 6.2 werden unterschiedliche Ansätze für geschachtelte Transaktionsmodelle auf ihre Anwendbarkeit hin untersucht. Neben den beiden Ansätzen der geschlossen und offen geschachtelten Transaktionen wird auch das Konzept der Mehrebenen-Transaktionsmodelle vorgestellt. Anschließend wird *tx+YAWL* als eigene Entwicklung vorgestellt, das auf dem Konzept der Mehrebenen-Transaktionsmodelle aufbaut. In Abschnitt 6.3 werden die verschiedenen Ebenen genauer betrachtet und transaktionale Sphären eingeführt. Sie bieten die Möglichkeit, in einem Prozessmodell Bereiche zu definieren, für die ein konsistenter Datenzugriff auf externe Variablen sicherzustellen ist. Um die parallele Abarbeitung von Transaktionen zu ermöglichen, wird in Abschnitt 6.4 ein Konzept für eine mögliche Umsetzung vorgeschlagen. Weil *YAWL* die vorgestellten Konzepte nicht direkt unterstützt, wird in Abschnitt 6.5 eine Transformation von *tx+YAWL* nach *YAWL* beschrieben. In Abschnitt 6.6 wird der vorgestellte Ansatz mit verwandten Arbeiten verglichen und es erfolgt eine Zusammenfassung des Kapitels in Abschnitt 6.7.

6.1 Anforderungen an einen konsistenten Datenzugriff

6.1.1 Problemstellung

Das Konzept der externen Variablen, das in Kapitel 5 vorgestellt wurde, betrachtet reine Lese- und Schreibzugriffe auf externe Datenquellen. Jeder Zugriff auf eine externe Variable führt zu einem Zugriff auf die entsprechende Datenquelle. Das Bereitstellen einer konsistenten Sicht auf die Daten muss durch die Datenquelle sichergestellt werden. Hat die Datenquelle dazu keine entsprechenden Möglichkeiten, kann der Datenzugriff via externer Variablen zu inkonsistenten Datenzuständen führen.

Am Beispiel des Prozessmodells aus Abbildung 5.1, Seite 90, soll das Problem erläutert werden. In diesem Prozessmodell wird die externe Variable *var* jeweils von der Aktivität A_2 und A_3 gelesen und geschrieben. Diese Form von Datenzugriffen kommt im Bereich digitaler Bibliotheken häufig vor. Die datenzentrierten Prozesse in einer digitalen Bibliothek bearbeiten ein konkretes Dokument als Folge von Aktivitäten, wie z. B. das schrittweise Anlegen eines Metadatensatzes. Im Beispiel kann der direkte Zugriff auf die externe Variable *var* zu inkonsistenten Datenzuständen führen, weil die Aktivitäten A_2 und A_3 während der Abarbeitung des Prozessmodells unterschiedliche Sichten auf die externen Datenquellen haben (inkonsistentes Lesen). Der inkonsistente Zustand entsteht, weil das Dokument *var* nicht unter der Kontrolle des WFMS steht. Das Dokument kann während der Abarbeitung der Prozessinstanz von einer anderen Anwendung in der Datenquelle geändert werden. Ein Problem tritt genau dann auf, wenn das Dokument geändert wird, nachdem die Aktivität A_2 gelesen hat und bevor die Aktivität A_3 gestartet wurde (und damit gelesen hat). Wenn also die Abarbeitungsreihenfolge $A_2 \prec A_3$ gilt, muss davon ausgegangen werden, dass $var_{A_2} \neq var_{A_3}$ gilt. Ein weiteres Problem kann auftreten, wenn Schreiboperationen auf externen Variablen von Teilergebnissen anderer Aktivitäten abhängen. Im Beispiel schreibt Aktivität A_3 den Wert *var*, wobei dieser von der korrekten Abarbeitung der Aktivität A_2 (mit $A_2 \prec A_3$) abhängig ist. Liest A_3 den Wert der externen Variable neu ein, kann dieser zwischenzeitlich verändert worden sein. Dies stellt eine klassische Mehrbenutzer-Anomalie dar, die beispielsweise durch den Einsatz eines Zwei-Phasen-Sperrprotokolls in der Datenquelle aufgelöst werden kann. Ein Konzept, um einen konkurrierenden Datenzugriff über mehrere Anwendungen hinweg sicherzustellen, bieten Transaktionen (siehe Abschnitt 2.3).

Um Aktivitäten einer Prozessinstanz einen konsistenten Datenzugriff auf externe Datenquellen zu ermöglichen, stellen wir ein Transaktionsmodell für WFMSe vor. Dem liegt die Annahme zugrunde, dass alle Lese- und Schreiboperationen auf Variablen, die an eine externe Variable gebunden sind, als Operationen einer Transaktion angesehen werden können. Das Transaktionsmodell muss dafür folgende Aufgaben umsetzen:

- Transaktionale Eigenschaften (ACID-Eigenschaften) sollen eine konsistente Sicht auf

externe Variablen sicherstellen.

- Die transaktionalen Eigenschaften müssen in Prozessmodellen geeignet modelliert werden können.
- Das Konzept muss in ein WFMS integrierbar sein.
- Für die Umsetzung sollen Semantik und Kontrollfluss von Prozessmodellen ausgenutzt werden.

Die nächsten beiden Unterabschnitte zeigen, warum sich ein flaches Transaktionsmodell (siehe Abschnitt 2.3) nicht eignet, den Datenaustausch mit externen Datenquellen im Publikationsprozess zu unterstützen. Ein Transaktionsmodell, das einen konsistenten Datenzugriff für externe Variablen sicherstellen soll, muss sowohl die sich durch die Integration von externen Datenquellen, als auch die durch die Struktur der Prozessmodelle ergebenden Anforderungen erfüllen.

6.1.2 Datenzugriff und resultierende Probleme für Transaktionen

Im klassischen Transaktionsmodell sollen Transaktionen innerhalb kurzer Zeit abgearbeitet sein [Gra81]. Je nach verwendetem Sperrprotokoll, führt das Sperren einzelner Datenquellen in einem flachen Transaktionsmodell dazu, dass andere Transaktionsschritte blockiert oder abgebrochen werden. Zeitliche Verzögerungen, welche durch blockierte oder abgebrochene Transaktionen auftreten, sind für Anwendungen aufgrund der kurzen Zeiträume, in der eine Transaktion ausgeführt wird, zu vernachlässigen. In einem Prozess kann die Bearbeitung einzelner Aktivitäten sehr lange dauern. Langlaufende Aktivitäten können Daten und Ressourcen über einen Zeitraum von Stunden, Tagen, Wochen oder sogar Monaten binden. Deshalb ist es nicht wünschenswert, alle Datenquellen, die innerhalb einer Prozessinstanz benötigt werden, bereits am Anfang zu sperren und erst am Ende der Prozessinstanz wieder freizugeben. Das Sperren von Datenquellen bzw. Datensätzen in einer Datenquelle würde bei langlaufenden Prozessen auch dazu führen, dass die Datenquellen für andere Anwendungen während der gesamten Ausführungsdauer dieser Prozessinstanz nicht mehr nutzbar sind.

Wenn Datenquellen nicht für die Dauer der Abarbeitung einer Prozessinstanz gesperrt werden, können die Datenobjekte von anderen Anwendungen unabhängig verändert werden. Um dennoch ACID-Eigenschaften für den Datenzugriff aus einer Prozessinstanz heraus zu gewährleisten, muss das WFMS den Datenzugriff geeignet kontrollieren. Eine solche Unterstützung soll möglich sein, wenn eine Datenquelle keine Transaktionsunterstützung anbietet.

Das WFMS muss kontrollieren, wann Datenobjekte mit externen Datenquellen unter Beachtung der Datenkonsistenz synchronisiert werden können. Dies kann grundsätzlich erreicht werden, wenn die Datenobjekte nicht bei jedem neuen Zugriff innerhalb einer Prozessinstanz mit der Datenquelle synchronisiert werden. Eine mögliche Lösung besteht darin, die

Datenobjekte im WFMS zwischenzuspeichern. Das Anlegen lokaler Versionen von Datenobjekten erfordert dann aber geeignete Synchronisationsstrategien, um zumindest auf der Seite des WFMS Eigenschaften wie Atomarität, Konsistenz, Isolation und Dauerhaftigkeit gewährleisten zu können.

Nicht jede Aktivität oder jeder Prozesspfad greift auf externe Datenquellen zu. Sie müssen deshalb auch nicht Teil einer Transaktion sein. Das Transaktionsmodell muss aus diesem Grund die Möglichkeit bieten, ACID-Eigenschaften nicht nur für den gesamten Prozess (also die gesamte Transaktion), sondern auch für Teilbereiche eines Prozesses zu definieren. Ein solcher Teilbereich wird im Folgenden *transaktionaler Bereich* genannt.

Ziel ist es, dass das WFMS mit Hilfe *transaktionaler Bereiche* die Synchronisation von externen Variablen und der entsprechenden externen Datenquelle kontrolliert durchführt. Zusammenfassend ergeben sich für den Datenzugriff folgende Anforderungen:

Anforderung 6.1: Lokal gespeicherte Versionen von externen Datenobjekten erfordern geeignete Synchronisationsverfahren, um Inkonsistenzen zu vermeiden.

Anforderung 6.2: Transaktionale Bereiche sollen Prozessbereiche kennzeichnen, für die ACID-Eigenschaften gelten sollen.

Anforderung 6.3: Transaktionale Bereiche sollen die Isolation von Prozessbereichen hinsichtlich der verwendeten Datenobjekte sicherstellen, ohne dabei die Parallelität bzw. Nebenläufigkeit der Prozessbereiche vollständig aufzuheben.

Anforderung 6.4: Prozesse bzw. Transaktionen können sehr lange laufen. Dieser Aspekt sollte durch ein Transaktionsmodell dadurch unterstützt werden, dass langlaufende Transaktionen vermieden werden. So sollen z. B. logische Sperren (im Sinne eines Zwei-Phasen-Sperrprotokolls) auf Datenquellen oder das häufige Zurücksetzen von Transaktionen vermieden werden.

6.1.3 Prozessstruktur und resultierende Probleme für Transaktionen

Prozesse weisen in der Regel eine geschachtelte Struktur auf, welche eine parallele Abarbeitung von Aktivitäten ermöglicht. Ein flaches Transaktionsmodell unterstützt diese parallele Ausführung von Transaktionsschritten nicht. Das zu entwickelnde Transaktionsmodell muss entsprechend geschachtelte Strukturen unterstützen, die ein paralleles Ausführen ermöglichen. Die parallele Ausführung von Arbeitsschritten ist aber im allgemeinen nicht mit der Forderung der Isolation vereinbar.

Da die Bearbeitung eines (Sub-)Prozesses oder einer Aktivität sehr lange dauern kann, müssen Strategien für die Rücksetzbarkeit gefunden werden. Ein Subprozess, der bereits erfolgreich abgearbeitet wurde, muss eventuell zurückgesetzt werden, wenn dessen Vaterprozess nicht erfolgreich ausgeführt werden konnte [WV01]. Daraus folgt, dass der Effekt einer Aktivität nur valide ist, wenn verschiedene andere Aktivitäten auch erfolgreich abgearbeitet wurden [Ley95]. Hier müssen Verfahren eingesetzt werden, die die Kompensation von Aktivitäten und Subprozessen ermöglichen. Besonders für Prozessmodelle muss ein Konzept angeboten werden, das eine semantische Kompensation ermöglicht. Dies ist notwendig, weil Effekte einer Transaktion nicht immer einfach rücksetzbar sind. Beispielsweise kann eine E-Mail, die von einer Aktivität versendet wurde, nicht einfach zurückgesetzt werden. Gegebenenfalls muss eine zweite E-Mail verschickt werden. Es gibt aber auch bestimmte Prozessbereiche, deren Abbruch nicht zwangsläufig zum Abbruch des ganzen Prozesses (bzw. der Vatertransaktion) führen muss. Hier muss das Transaktionsmodell entsprechend flexibel gestaltet werden können.

In verschiedenen Situationen besteht außerdem die Notwendigkeit, Bedingungen an Daten zu kontrollieren und diese zuzusichern. Der Geltungsbereich der Bedingungen kann sich über alle Aktivitäten in einem Prozessmodell erstrecken, aber auch auf einzelne Prozessbereiche eingegrenzt sein. Das WFMS muss deshalb in der Lage sein, die Einhaltung von Bedingungen über eine Menge von externen Variablen und transaktionalen Bereichen zu überprüfen. Zusammenfassend ergeben sich für Prozessstrukturen folgende Anforderungen:

Anforderung 6.5: Das Transaktionsmodell muss geschachtelte Strukturen unterstützen.

Anforderung 6.6: Für transaktionale Prozessbereiche müssen Strategien für die Rücksetzbarkeit von Transaktionen unterstützt werden.

Anforderung 6.7: Das Transaktionsmodell muss die Möglichkeit bieten, Integritätsbedingungen über transaktionalen Bereichen zu definieren.

Der Datenzugriff auf externe Datenquellen innerhalb von Prozessinstanzen unterliegt besonderen Problemen hinsichtlich der Datenkonsistenz und der Datenintegrität. Die Anforderungen 6.4 bis 6.7 beschreiben dabei notwendige Voraussetzungen, welche ein Transaktionsmodell erfüllen muss. Die Anforderungen ergeben sich aus den Bedingungen der redundanten Datenhaltung und aus der Struktur der Prozessmodelle.

6.1.4 Fazit

Für die Umsetzung der Anforderungen, die in den Abschnitten 6.1.2 und 6.1.3 vorgestellt wurden, müssen folgende Komponenten entwickelt werden:

- Ein Transaktionsmodell, mit dem die transaktionalen Eigenschaften in Prozessmodellen beschrieben werden können. Für die Umsetzung werden geeignete Modellierungskonzepte benötigt.
- Ein Transaktionsmanager, der den Datenzugriff auf externe Datenquellen kontrolliert. Das Transaktionsmodell muss den Anforderungen gerecht werden, die sich durch die speziellen Eigenschaften der Prozessmodelle und den Anforderungen für den Datenzugriff ergeben.
- Eine Recovery-Komponente, durch die eine Ausnahmebehandlung für den Fall von Transaktionsabbrüchen ermöglicht wird.

6.2 Ein 4-Ebenen-Transaktionsmodell - *tx+YAWL*

6.2.1 Ein Lösungsansatz für geschachtelte Transaktionen

Geschlossen geschachtelte Transaktionen. Ein Transaktionsmodell, welches die Anforderungen aus Abschnitt 6.1 grundsätzlich unterstützt, ist das geschachtelte Transaktionsmodell [Mos85].[1] Es ermöglicht eine parallele Ausführung von Transaktionen (siehe Anforderung 6.5). Das Modell von Moss [Mos85] erlaubt die Schachtelung von Transaktionen in beliebig viele Subtransaktionen, die zusammen einen Transaktionsbaum bilden. Die Wurzel des Baumes stellt dabei die *Top-Level-Transaktion* dar. Eine Top-Level-Transaktion sichert der Anwendung die ACID-Eigenschaften zu, muss also im Falle eines Fehlers korrekt rücksetzbar sein. Eine isolierte Ausführung einzelner Top-Level-Transaktionen muss demnach möglich sein. Auch kann eine Top-Level-Transaktion erst erfolgreich beendet werden, wenn alle Subtransaktionen beendet wurden. Im Gegensatz zu Top-Level-Transaktionen können Subtransaktionen aber unabhängig von der Vatertransaktion ein *Commit* oder *Abort* durchführen. Für den Fall, dass eine Subtransaktion nicht erfolgreich beendet werden kann, kann je nach Subtransaktionstyp ein geeignetes Recovery durch die Vatertransaktion durchgeführt werden. Eine Subtransaktion kann *nicht-vital* sein, dann kann ein Abbruch in der Vatertransaktion ignoriert werden. Ist eine Menge von alternativen Subtransaktionen angegeben (*contingent transaction*), dann kann die Vatertransaktion versuchen, diese anstelle der fehlerhaften Transaktion auszuführen. Außerdem kann die Vatertransaktion auch versuchen, die Subtransaktion erneut auszuführen (*retry*). Die höhere Flexibilität wird an dieser Stelle aber mit der teilweisen Aufgabe der Atomarität von Teiltransaktionen erkauft. Wird beispielsweise ein Kontingent an Teiltransaktionen ausgeführt, kann man nicht mehr vom Prinzip *ganz oder gar nicht* sprechen.

Ein wesentlicher Nachteil des Modells in [Mos85] liegt darin, dass Ergebnisse von Subtransaktionen erst sichtbar werden, wenn die *Top-Level-Transaktion* erfolgreich beendet wurde.

[1] [Mos85] baut auf der Dissertation [Mos81] auf.

Modelliert man einen Prozess als Top-Level-Transaktion und Subprozesse oder parallele Prozesspfade als Subtransaktionen, so können Teilergebnisse der Subprozesse nicht freigegeben werden. Die parallele Ausführung von Transaktionen und Anwendungen wird so weiterhin stark eingeschränkt. Dies ist aber eine wesentliche Eigenschaft (siehe Anforderung 6.3), die das Transaktionsmodell für den hier betrachteten Anwendungsfall erfüllen muss.

Offen geschachtelte Transaktionen. Das oben aufgezeigte Problem wird mit dem Konzept der offen geschachtelten Transaktionen behoben, indem die Forderung der Isolation für Transaktionen teilweise abgeschwächt wird. Ergebnisse von Subtransaktionen können bereits nach der Commit-Operation einer Subtransaktion für andere, parallel laufende Transaktionen bereitgestellt werden. Möglich wird dies durch den Einsatz von sogenannten kompensierenden Transaktionen [Gra81]. Eine kompensierende Transaktion ist notwendig, um im Fall des Abbruchs der Vatertransaktion den Effekt der Subtransaktion zurücknehmen zu können. Die kompensierende Transaktion beschreibt dabei die notwendigen Operationen, um den Effekt der zugehörigen Transaktion rückgängig zu machen. Das Konzept der offen geschachtelten Transaktionen erfüllt damit die Anforderungen 6.6 und 6.3. Die Idee wird beispielsweise in [GMS87] verwendet, um langlaufende Transaktionen mit Hilfe von *Sagas* [GMS87] umzusetzen.

Das Mehrschichten-Transaktionsmodell. Eine spezielle Ausprägung der offen geschachtelten Transaktionen ist das Mehrschichten-Transaktionsmodell [Wei88, WS91, Wei91]. Eine ähnliche Idee wird auch in [BSW88] vorgestellt. Die Idee des Mehrschichten-Transaktionsmodells besteht darin, die Anzahl der Ebenen zu begrenzen und damit alle Transaktionsbäume auf dieselbe Höhe zu beschränken. Ein Transaktionsbaum hat demnach $n+1$ Ebenen, wobei L_i die $i$te Ebene darstellt. Die Ebene L_{i+1} ist eine Sequenz von L_i-Operationen der Ebene L_i mit $i = 0,\ldots,n-1$. Die Blattebene des Transaktionsbaumes wird durch die Ebene L_0 definiert. Für jede Ebene L_i wird eine (dynamische) Menge von Objekten OBJ_i und eine (feste) Menge von Operatoren F_i angegeben. Eine L_i-Operation setzt sich aus dem Paar (f,w) zusammen, wobei $f \in F_i$ ein L_i-Operator ist, dessen Parameterliste w mit Objekten aus OBJ_i belegt wird. [Wei88]

Der Einsatz eines Mehrschichten-Transaktionsmodells bringt Vorteile für den hier betrachteten Anwendungsfall. Die Anwendungsschichten lassen sich gut auf die Schichten des Transaktionsmodells abbilden. Durch die Einführung des Schichtenkonzepts können die für eine Transaktionsverwaltung wichtigen Operationen auf unterschiedlichen Abstraktionsebenen betrachtet werden. Besonders Konflikte können so auf den jeweiligen Schichten gesondert betrachtet werden (was aber nicht bedeuten soll, dass diese unabhängig voneinander sind!). Höhere Schichten bieten wesentlich mehr semantische Informationen über die umgesetzten Operationen, die für eine parallele Abarbeitung genutzt werden können. Parallelität auf unteren Schichten ist schwieriger umzusetzen, da Konflikte und Pseudokonflikte wegen fehlender semantischer Informationen nur schwer zu unterscheiden sind [Wei88].

Definition 6.1 (n-Schichten-Transaktion T_j [Wei88])
T_j besteht aus:

- n nichtleeren Menge $A_i^{(j)}$ von L_i-Aktionen $(0 \leq i < n)$ der Form
  $A_{n-1}^{(j)} = \{a_{j1}, \ldots, a_{jv_{n-1}}\}$
  $A_i^{(j)} = \{a_{w1}, \ldots, a_{wv_i} \mid a_w \in A_{n+1}^{(j)}\}$ für $0 \leq i < n-1)$
- und n partiellen Ordnungen $\ll_i^{(j)} \subseteq A_i^{(j)} \times A_i^{(j)}$, die L_i-(Intratransaktions-) Präzedenzrelationen genannt werden $0 \leq i < n)$

Wie oben bereits erwähnt, setzt sich in einer Mehrschichten-Transaktion jede Ebene aus L_i-Operationen zusammen. Jede L_i Operationen tritt gegenüber der Ebene L_{i-1} (für $i > 0$) als Transaktion auf, und gegenüber Ebene L_{i+1} (für $i < n$)) als Aktion.[Wei88]

$$\begin{aligned} trans(a) &:= a_w \\ act(a) &:= \{a_{wv_iv_{i-1}} \in A_{i-1}^{(j)}\} \end{aligned} \tag{6.1}$$

Dabei gibt die in Gleichung 6.1 angegebene Beziehung $trans(a) = a_w$ die erzeugende L_{i+1}-Aktion zu einer Aktion a aus. Die L_i Aktion a_w wird dann auch Transaktion genannt, die a enthält. Die Beziehung $act(a)$ gibt die Menge aller L_{i-1}-Aktionen an, die von einer Aktion a gestartet werden.[Wei88]

Definition 6.2 (Historie [WV01])
Die Menge von globalen Transaktionen ist gegeben als $T = \{t_1, \ldots, t_n\}$, wobei jede globale Transaktion $t_i \in T$ ein Baum ist, der durch ein Paar $(op_i, <_i)$ mit der Menge op_i von Baumknoten und der partiellen Ordnung $<_i$ auf den Blattknoten definiert ist. Für eine Menge von Transaktionen T ist s eine Historie mit partiell geordnetem Wald $(op(s), <_s)$, mit der Knotenmenge $op(s)$ und der partiellen Ordnung $<_s$.

1. $op(s) \subseteq \bigcup_{i=1}^{n} op_i \cup \bigcup_{i=1}^{n} \{a_i, c_i\}$, und $\bigcup_{i=1}^{n} op_i \subseteq op(s)$. Dabei enthält s die Operationen der gegeben Transaktionsbäume und a_i (*Abort*) oder c_i (Commit) beschreiben die Terminierungsoperationen einer Transaktion $t_i \in T$.
2. $(\forall i, 1 \leq i \leq n)\ c_i \in op(s) \Leftrightarrow a_i \notin op(s)$.
3. a_i or c_i sind Blattknoten und der Vatertransaktion t_i.
4. $\bigcup_{i=1}^{n} <_i \subseteq <_s$, die Ordnung aller Transaktionen ist in der partiellen Ordnung von s enthalten.
5. $(\forall i, 1 \leq i \leq n)\ (\forall p \in op_i)\ p <_s a_i$ or $p <_s c_i$
6. Für Jedes Paar von Blattoperationen $p, q \in op(s)$, aus unterschiedlichen Transaktionen, die auf dasselbe Datenobjekt zugreifen und wenn mindestens eine Operation eine Schreiboperation ist, müssen die beiden Operationen in s geordnet sein. Entweder muss $p <_s q$ oder $q <_s p$ gelten.

Ein Präfix von einer Historie $s = (op(s), <_s)$ ist ein Wald $s' = (op(s'), <'_s)$. Für s' gilt weiterhin $op(s') \subseteq op(s)$. Außerdem gilt $<'_s \subseteq <_s$, so dass für jedes $p \in op(s')$ und alle Vorfahren von p und alle Knoten $q \in op(s)$ mit $q <_s p$ in $op(s')$ sein müssen. Wird auf $op(s')$ eingeschränkt, dann muss $<'_s$ gleich $<_s$ sein. [WV01]

In nächsten Abschnitt wird die Idee der Mehrschichten-Transaktionen aufgegriffen und auf die in Kapitel 5 vorgestellte Architektur für die Integration von externen Datenquellen angewendet.

6.2.2 tx+YAWL im Überblick

Hier wird die Idee der Mehrschichten-Transaktionsmodelle aufgegriffen und im Folgenden das Mehrschichten-Transaktionsmodell *tx+YAWL* vorgestellt. *tx+YAWL* ist eine transaktionale Erweiterung für *YAWL*. Durch den Einsatz von *tx+YAWL* soll ein konsistenter Datenzugriff auf externe Datenquellen für Aktivitäten einer Prozessinstanz ermöglicht werden.

Die Ebenen des Transaktionsmodells werden auf Grundlage der Anwendungsarchitektur aus Kapitel 5 definiert. *tx+YAWL* definiert auf Grundlage der vorgegebenen Architektur ein 4-Ebenen-Transaktionsmodell.

Ebene L_0 (Datenzugriff). Die unterste Ebene L_0 beschreibt den direkten Datenaustausch mit externen Datenquellen. Die auf dieser Ebene bereitgestellten Operationen sind einfache Lese- und Schreiboperationen auf externen Datenobjekten. Die Operationen werden durch Plug-ins bereitgestellt, die eine Datenquelle kapseln und die nicht zwangsläufig ein eigenes Transaktionsmanagement bereitstellen müssen.

Ebene L_1 (Workflow-Aktivitäten). Die Basiselemente der Ebene L_1 sind Workflow-Aktivitäten. Sie stellen die Bausteine der Transaktionen auf dieser Ebene dar. Operationen auf dieser Ebene sind Lese- oder Schreibzugriffe auf eine externe Variable. Ein Lese- oder Schreibzugriff auf eine externe Variable in Ebene L_1 fasst den Zugriff auf ein Plug-in in Ebene L_0 zusammen.

Ebene L_2 (Kontrollfluss und transaktionale Sphären). Die Elemente einer Transaktion in Ebene L_2 werden durch die Lese- und Schreiboperationen auf Aktivitätsvariablen modelliert. Lese- und Schreiboperationen auf Aktivitätsvariablen in Ebene L_2 sind Zugriffe, die wiederum auf externe Variablen der Ebene L_1 abgebildet werden. Die Modellierung von (Sub-)Transaktionen wird auf dieser Ebene mit Hilfe von transaktionalen Sphären (gestricheltes Rechteck in Abbildung 6.1) unterstützt. Eine (Sub-)Transaktion setzt sich aus einer Menge von Aktivitäten zusammen.

Ebene L_3 (Instanz und globale Transaktionen). Auf der obersten Ebene L_3 des Transaktionsmodells wird die Sicht der Prozessinstanz beschrieben. Die globalen Elemente (globale transaktionale Sphären) der Ebene L_2 werden hier zu einer Top-Level-Transaktion zusammengesetzt.

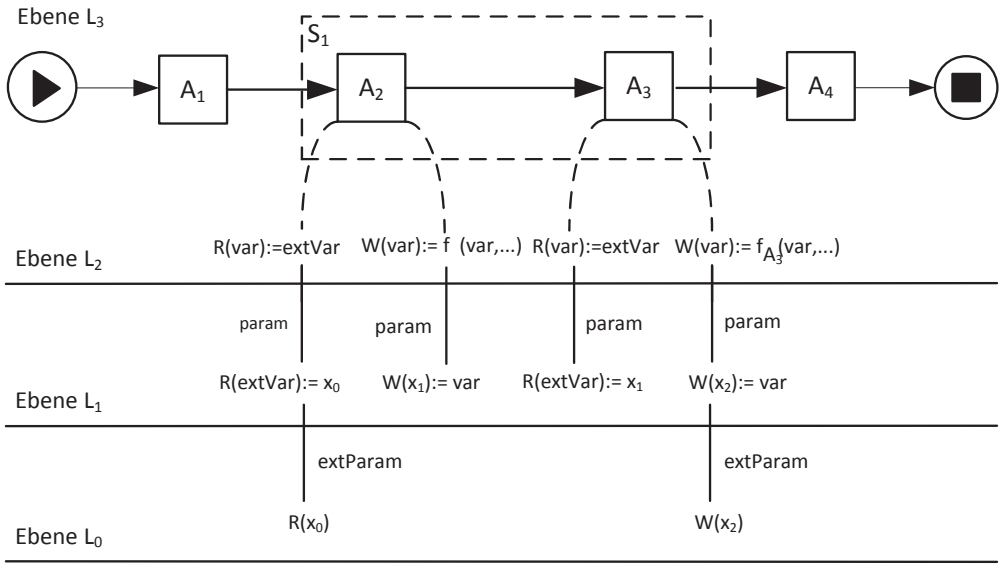


Abbildung 6.1: Transaktionsmodell mit 4 Ebenen

Am Beispiel des in Abbildung 6.1 abgebildeten Prozessmodells, soll das Mehrschichten-Transaktionsmodell *tx+YAWL* vorgestellt werden. Der Datenzugriff ist in diesem Beispiel mit der in Kapitel 5 gezeigten Erweiterung zur Integration externer Datenquellen umgesetzt.

Das Prozessmodell in Abbildung 6.1 modelliert die vier Aktivitäten A_1, A_2, A_3 und A_4, die sequentiell abgearbeitet werden. Die beiden Aktivitäten A_2 und A_3 greifen auf die externe Variable *extVar* lesend zu. Die Aktivität A_3 belegt die Variable *extVar* zusätzlich mit einem neuen Wert. Die Variable *extVar* ist über einen Parameter jeweils an eine Aktivitätsvariable *var* gebunden. Der externe Parameter *extParam* bindet die externe Variable *extVar* außerdem an ein Plug-in, welches die Lese- und Schreiboperationen bereitstellt.

Auf der obersten Ebene L_3 werden entsprechend der Ebenen-Architektur alle globalen Sphären der Ebene L_2 zusammengefasst. Im Beispiel ist in Ebene L_2 eine transaktionale Sphäre S_1 modelliert. Sie umfasst alle Aktivitäten, welche auf eine externe Variable zugreifen und beschreibt die Transaktionsgrenzen. Operationen auf der Ebene L_2 sind die Lese- und Schreibzugriffe auf der Aktivitätsvariable *var*, die für die beiden Aktivitäten definiert ist. Es ergeben sich also je zwei Lese- und Schreiboperationen auf dieser Ebene. Die Operationen werden auf Lese- und Schreiboperationen der Ebene L_1 abgebildet. In Ebene L_1 wird entschieden, inwieweit eine Operation an Ebene L_0 weiter geleitet wird oder eine lokale Version des Datenobjektes verwendet wird. Im Beispiel wird der erste Lesezugriff auf die externe

Variable *extVar* auf die Operation $r(x_0)$ der Ebene L_0 abgebildet. Die letzte Schreiboperation $w(x_2)$, die von Aktivität A_3 erzeugt wurde, wird ebenfalls an ein Plug-in weitergereicht (an Ebene L_0). Alle anderen Lese- und Schreiboperationen werden lokal verwaltet, um eine konsistente Sicht auf die Datenquellen zu gewährleisten. Hierdurch kann weiterhin die Anzahl der Operationen auf Ebene L_0 stark reduziert werden. Auf der Blattebene L_0 werden die unteilbaren Lese- und Schreiboperationen auf Plug-ins ausgeführt.

6.3 *tx+YAWL*- Ebenen

In diesem Abschnitt werden die 4 Ebenen von *tx+YAWL* vorgestellt. Ziel ist es, zu zeigen, wie die Operationen der einzelnen Ebenen aufeinander abgebildet werden.

6.3.1 Ebene L_0 - Datenzugriff

Die Blattebene im *tx+YAWL* Modell wird durch die Ebene L_0 repräsentiert. Betrachtet man die Architektur aus Abbildung 6.1, wird in Ebene L_0 der Zugriff auf externe Datenquellen umgesetzt. In Abschnitt 5.2 wurde bereits gezeigt, wie Datenquellen mit Hilfe einer Plug-in-Architektur in das WFMS integriert werden können.

Auf Ebene L_0 werden die elementaren Operationen für den transaktionalen Datenzugriff bereitgestellt. Die Menge der Operatoren F_0 ist durch

$$F_0 = \{R, W\} \tag{6.2}$$

beschrieben. Der Leseoperator $f_{0,1} = R$ korrespondiert mit der Plug-in-Schnittstelle zum Lesen eines Datenobjekts[2]. Entsprechend korrespondiert $f_{0,2} = W$ mit der Plug-in-Schnittstelle zum Schreiben eines Datenobjekts. Die Objekte OBJ_0 der Ebene L_0 sind durch die Menge der Datenobjekte gegeben, mit $OBJ_0 = \{d_{j1}, \ldots, d_{jv_{n-1}}\}$. Eine L_0-Operation ist entweder mit $r(x)$ oder $w(x)$ gegeben, wobei für den Parameter $x \in OBJ_0$ gilt.

6.3.2 Ebene L_1 - Workflow-Aktivitäten

Im Prozess-Metamodell werden in der Ebene L_1 externe Variablen definiert, über die auf externe Datenquellen innerhalb einer Prozessbeschreibung zugegriffen werden kann. Das Konzept ist in Abschnitt 5.3 beschrieben.

[2]Die erste Ziffer des Indexes beschreibt die Ebene auf welcher der Operator definiert ist. Der zweite Wert dient zur Nummerierung

Operationen der Ebene L_1
Die Menge der Operatoren F_1 auf Ebene L_1 ist gegeben durch

$$F_1 = \{R, W\} \tag{6.3}$$

Der Leseoperator $f_{1,1} = R$ beschreibt einen Lesezugriff auf eine externe Variable. Entsprechend beschreibt der Schreiboperator $f_{1,2} = W$ einen Schreibzugriff auf eine externe Variable.

Für eine L_1-Aktion $a := r(x)$ einer Transaktion T_j gilt

$$act(a) := \{r(x) \in A_0^{(j)}\} \tag{6.4}$$

Für eine L_1-Aktion $a := w(x)$ einer Transaktion T_j gilt

$$act(a) := \{w(x) \in A_0^{(j)}\} \tag{6.5}$$

Eine Leseoperation auf einer externen Variable in Ebene L_1 wird also auf genau eine Leseoperation auf Ebene L_0 abgebildet. Eine Schreiboperation auf einer externen Variable in Ebene L_1 wird ebenfalls auf genau eine Schreiboperation auf Ebene L_0 abgebildet. Für die Abbildung wird ein externer Parameter verwendet, wie er in Abschnitt 5.3.2 definiert ist.

In Abbildung 6.1 wird der externen Variable *extVar* der Wert x_0 zugewiesen, der durch ein Plug-in bereitgestellt wird.

Ein lokaler L_1-Arbeitsbereich
Um Isolation, Konsistenz und Rücksetzbarkeit für den Datenzugriff auf beliebige Datenquellen zu gewährleisten, können nicht alle L_1-Operationen direkt an L_0-Operationen weitergereicht werden. Würde jede Leseoperation auf Ebene L_1 auch zu einer Leseoperation auf Ebene L_0 umgesetzt, kann dies unter Umständen zu inkonsistenten Datenzuständen führen, wie in Abschnitt 6.1 bereits erwähnt wurde. Gleiches gilt für Schreiboperationen. Im Ergebnis würden anderen Anwendungen inkonsistente Zwischenergebnisse bereitgestellt. Um diesem Problem entgegenzuwirken und um die Umsetzung der Anforderungen 6.4, 6.1 und 6.3 zu unterstützen, werden lokale Versionen der externen Daten angelegt. Dafür wird auf Ebene L_1 ein lokaler Arbeitsbereich für externe Variablen eingeführt. Das Anlegen von Versionen innerhalb des WFMSs bedarf dann aber geeigneter Synchronisationsstrategien, damit keine inkonsistenten Daten aus der Datenquelle gelesen oder inkonsistente Daten in eine Datenquelle geschrieben werden.

Existiert bereits eine lokale Version der Variable, wird diese verwendet und kein externer Datenzugriff durchgeführt. Wird der Wert einer externen Variable geschrieben, so wird zuerst die lokale Version im L_1-Arbeitsbereich überschrieben. Existiert dort keine lokale Version

der Variable, wird sie neu angelegt. Eine lokale Version im L_1-Arbeitsbereich wird erst in die externe Datenquelle zurückgeschrieben, nachdem die Transaktion mit der Operation $c()$ erfolgreich beendet wurde. Aus dem lokalen Arbeitsbereich dürfen vor dem Commit keine geänderten Daten in die externe Datenquellen geschrieben werden, was häufig auch als *defered update* bezeichnet wird. Das entspricht der Ersetzungsstrategie $\neg$*steal*, die in [HR83] eingeführt wird. Wurde die Sphäre mit einem Commit beendet, dann werden geänderte Versionen direkt in die Datenquelle geschrieben. Das entspricht der Propagierungsstrategie *force* und der Einbringstrategie $\neg$*atomar* aus [HR83]. Enthält eine Transaktion Subtransaktionen, dann verwendet die Subtransaktion den gleichen L_1-Arbeitsbereich, mit denselben lokalen Versionen. Weil das Transaktionsmodell nach dem Prinzip der offen geschachtelten Transaktionen umgesetzt ist, werden lokale Versionen ebenfalls in die externe Datenquelle zurückgeschrieben, nachdem die Subtransaktion mit der Operation $c()$ erfolgreich beendet wurde. Die Version bleibt aber für alle parallel laufende Subtransaktionen und die Vatertransaktionen im lokalen Arbeitsbereich erhalten.

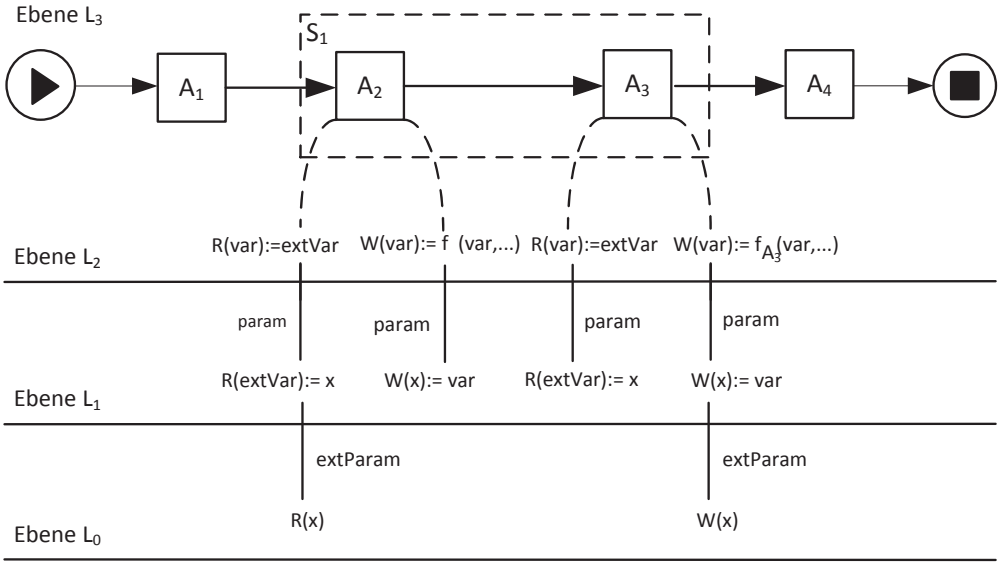


Abbildung 6.2: Transaktionsmodell mit lokalem L_1-Arbeitsbereich

Abbildung 6.2 zeigt das Transaktionsmodell mit allen vier Ebenen. Ebene L_1 ist mit dem oben beschriebenen L_1-Arbeitsbereich erweitert, sodass es keine Eins-zu-eins-Umsetzung von L_1-Operationen auf L_0-Operationen gibt.

Beispiel 6.1 (Lokaler L_1-Arbeitsbereich)
Am Prozess in Abbildung 6.2 soll die Umsetzung erläutert werden. Im Beispiel erzeugt der erste Lesezugriff (durch Aktivität A_2) auf die externe Variable *extVar* eine lokale Version der Variable im L_1-Arbeitsbereich. Sie wird durch eine Leseoperation auf Ebene L_0 erzeugt. Die Schreiboperation durch Aktivität A_2 überschreibt die lokale Version im L_1-Arbeitsbereich. Im nächsten Schritt liest die Aktivität A_3 die Variable *extVar*, wobei der Wert aus dem lokalen

L_1-Arbeitsbereich übernommen wird. Die Schreiboperation von Aktivität A_3 führt dazu, dass erst die Version der Variable *extVar* im lokalen L_1-Arbeitsbereich überschrieben wird. Danach wird der Wert der lokalen Version des L_1-Arbeitsbereiches in die externe Datenquelle geschrieben, weil die Transaktion beendet wurde. Im Beispiel wird dies durch die Grenzen der transaktionalen Sphäre bestimmt (siehe Abschnitt 6.3.3).

Der L_1-Arbeitsbereich ermöglicht es, Aktivitäten eine konsistente Sicht auf die Datenquellen zu geben. Probleme, die durch die parallele Ausführung von mehreren (Sub-)Transaktionen oder parallelen Prozesspfaden entstehen, die auf denselben Datenobjekten arbeiten, werden in Abschnitt 6.4 betrachtet.

6.3.3 Ebene L_2 - Kontrollfluss und transaktionale Sphären

Operationen der Ebene L_2
Im Ebenen-Konzept werden die Operationen von Ebene L_2 durch Sequenzen von Lese- und Schreiboperationen auf externen Variablen der Aktivitäten definiert. Eine L_2-Aktion ist wiederum als Menge von L_1-Aktionen umgesetzt. Die Menge der Operatoren F_2 auf Ebene L_2 ist gegeben durch

$$F_2 = \{R, W\} \tag{6.6}$$

Der Leseoperator $f_{2,1} = R$ beschreibt einen Lesezugriff auf eine Aktivitätsvariable, die an eine externe Variable gebunden ist. Entsprechend beschreibt der Schreiboperator $f_{2,2} = W$ einen Schreibzugriff auf eine Aktivitätsvariable, die an eine externe Variable gebunden ist.

Für eine L_2-Aktion $a := r(x)$ einer Transaktion T_j gilt

$$act(a) := \{r(x) \in A_1^{(j)}\} \tag{6.7}$$

Für eine L_2-Aktion $a := w(x)$ einer Transaktion T_j gilt

$$act(a) := \{w(x) \in A_1^{(j)}\} \tag{6.8}$$

Eine Leseoperation auf einer Aktivitätsvariable in Ebene L_2 wird also auf genau eine Leseoperation auf Ebene L_1 abgebildet. Eine Schreiboperation auf einer Aktivitätsvariable in Ebene L_2 wird ebenfalls auf genau eine Schreiboperation auf Ebene L_1 abgebildet. Für die Abbildung wird jeweils ein *YAWL*-Parameter verwendet.

In Abbildung 6.1 wird der Aktivitätsvariable *var* der Wert x zugewiesen, der durch die externe Variable *extVar* bereitgestellt wird.

Transaktionale Sphären

Ein wesentliches Konzept für die Umsetzung des Mehrschichten-Transaktionsmodelles *tx+YAWL* sind *transaktionale Sphären*. Wenn eine Aktivität Variablen an externe Variablen bindet (siehe Abschnitt 5.3), muss dieser Datenzugriff vom WFMS geeignet kontrolliert und überwacht werden. Transaktionale Sphären sollen deshalb die folgenden Möglichkeiten bieten:

- transaktionale Eigenschaften an konkrete Prozessbereiche zu binden.
- die Kontrollflussperspektive mit entsprechenden Modellierungskonzepten zu erweitern.
- Lese- und Schreiboperationen auf externe Variable zu Transaktionen zusammenzufassen.

Eine transaktionale Sphäre kann entweder aus einer einzelnen Aktivität, aus einer Menge von Aktivitäten oder aus einer zusammengesetzten Aktivität bestehen. Die Idee baut auf der Notation von Sphären auf, die Davies in [Dav78] vorgestellt hat. Eine Definition für transaktionale Sphären folgt in Abschnitt 6.3.3. Im Folgenden wird der Begriff der transaktionalen Sphäre gleichbedeutend mit *Sphäre* verwendet.

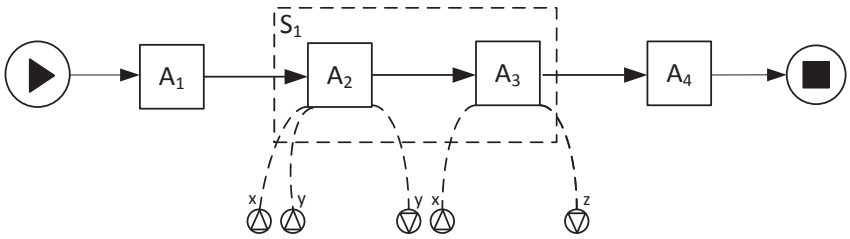


Abbildung 6.3: YAWL Prozess mit Transaktionaler Sphäre S_1

Zunächst soll die Idee der Sphäre anhand eines Beispiels erläutert werden.

Beispiel 6.2 (Idee Sphäre)
In Abbildung 6.3 wird ein einfaches *YAWL*-Prozessmodell mit den 4 Aktivitäten A_1, A_2, A_3 und A_4 gezeigt. Im Prozessmodell sind weiterhin die externen Variablen x, y, z definiert, die jeweils an unterschiedliche externe Datenquellen gebunden sind. Die Aktivitäten A_2 und A_3 lesen den Wert von der externen Variable x. Die Aktivität A_2 liest und schreibt den Wert von der externen Variable y. Zusätzlich belegt die Aktivität A_3 die Variable z mit einem neuen Wert. Um den Datenzugriff über die externe Variable als Transaktion zu definieren, sind die beiden Aktivitäten A_2 und A_3 von der Sphäre S_1 umgeben.

Ein Beispiel wie Sphären im Publikationsprozess verwendet werden können, wird in Anhang B.4 gezeigt.

Eigenschaften einer Sphäre

Transaktionale Sphären sollen nicht nur dafür genutzt werden, Bereiche in einem Prozessmodell zu kennzeichnen, die transaktional sein sollen. Eine Sphäre soll außerdem dafür

genutzt werden, atomare Aktionen, Integritätsbedingungen, Lese- und Schreibzeitpunkte von externen Variablen und Transaktionstypen zu definieren.

Atomare Aktionen. Aktivitäten innerhalb einer Sphäre sollen also entweder alle Änderungen auf externen Variablen, die innerhalb einer Sphäre gemacht wurden, vollständig in die Datenquellen schreiben oder bei einem Fehler zurücknehmen. Nachdem alle Aktivitäten einer Sphäre erfolgreich ausgeführt wurden, müssen alle Werte die in eine externe Variable geschrieben wurden, persistent in die entsprechenden Datenquellen geschrieben werden.

Integritätsbedingungen. Eine Anwendung kann es erfordern, dass innerhalb festgelegter Prozessbereiche für externe Variablen auch Integritätsbedingungen sichergestellt werden müssen. Für eine Sphäre können Integritätsbedingungen definiert werden, die auf die externen Variablen anzuwenden sind. Vor und nach dem Ausführen der Sphäre, müssen sich die Variablen in einem konsistenten Zustand bezüglich der Integritätsbedingungen befinden. Damit können Integritätsbedingungen nicht nur als Vor- und Nachbedingungen von Aktivitäten definiert werden, sondern auch für Prozessbereiche in einem Prozess.

Isolation. Sphären werden isoliert voneinander ausgeführt. Das bedeutet, dass Aktivitäten eine Sicht auf externe Datenquellen erhalten müssen, genau so also ob sie alleine auf den Daten arbeiten. Für zwei Aktivitäten $A_i \in S_1$ und $A_j \in S_2$, welche die gleiche externe Variable z verwenden, muss die Sicht auf die Variable z so sein, als ob sie alleine darauf arbeiten.

Lese- und Schreibzeitpunkte. Um eine konsistente Sicht auf eine Datenquelle, die an eine externe Variable gebunden ist, zu gewährleisten, darf sie innerhalb einer Sphäre nur jeweils einmal gelesen und geschrieben werden. Zwischenergebnisse müssen durch das Framework geeignet gespeichert und mit den Datenquellen synchronisiert werden (siehe Anforderung 6.1). Der Zugriff auf externe Variablen darf nur innerhalb einer transaktionalen Sphäre stattfinden. Nur so kann der konsistente Datenzugriff auf externe Quellen gewährleistet werden.

Transaktionstyp. Je nach Anforderungen, die durch verschiedene Anwendungen entstehen, können einer Sphäre unterschiedliche transaktionale Eigenschaften zugeordnet werden. Der Typ einer (Sub-)Sphäre kann aus der Menge {*basic, compensating, contingent, non-transactional, read-only, non-vital*} gewählt werden. Um die unterschiedlichen Typen einer Sphäre im Prozessmodell einfacher zu modellieren, haben wir das Prozess-Metamodell von *YAWL* mit speziellen Symbolen für die jeweiligen Sphären erweitert. In Abschnitt 6.3.3 werden die einzelnen Transaktionstypen, die eine Sphäre haben kann, vorgestellt.

Die folgende Definition berücksichtigt die bisher genannten Eigenschaften einer Sphäre.

Definition 6.3 (Transaktionale Sphäre)
Eine transaktionale Sphäre S_i ist einer Transaktion T_i und einer Menge $(A, type, readset(T_i), writeset(T_i), C, <)$ zugeordnet. Dabei ist:

- A_i die Menge aller Aktivitäten in einer Sphäre. Innerhalb einer Sphäre sind alle in *YAWL* erlaubten Typen von Aktivitäten erlaubt.
- $type(T)$ der Transaktionstyp mit {*non-transactional, basic, read-only, non-vital, compensating, contingent*}.
- C die Menge aller Constraints, die für eine Sphäre gelten müssen.
- $<_s$ ist die partielle Ordnung von Transaktionsschritten einer Transaktion $T_i = p_1, \ldots, p_n$. Transaktionsschritte bzw. Aktionen haben entweder die Form $p_j = r(x)$ oder $p_j = w(x)$, wobei gilt $x \in readset(T_i) \cup writeset(T_i)$. Die Ordnung von Lese- und Schreiboperationen für eine einzelne Aktivität ist die Reihenfolge in der die Operationen in der Parameterliste der Aktivität ausgewertet werden.

Für die Interpretation der einzelnen Operationen p_j einer Transaktion T_i gilt:

- wenn $p_j = r(x)$, dann wird der aktuelle Wert von x der lokalen Variable v_j zugewiesen.
- wenn $p_j = w(x)$, dann wird der Wert von x gegebenenfalls neu berechnet, mit $x := f_j(v_{j1}, ..., v_{jk})$. Mit der Funktion f_j wird der Effekt der Aktivität A_j beschrieben. Die Schritte $j_1..j_k$ beschreiben die bis zu diesem Zeitpunkt ausgeführten Leseoperationen von T_i.

Am Beispiel des in Abbildung 6.3 gezeigten Prozessmodells soll die Definition 6.3 erläutert werden. Für die Sphäre S_1 wird eine Transaktion $T_1 = r(x), r(y), w(y), w(z)$ erzeugt. Die Ordnung $<_s$ der Operationen ergibt sich aus der Reihenfolge der Aktivitäten in S_1, mit $r(x) <_s r(y) <_s w(y) <_s w(z)$. Der Typ der Transaktion ist *basic*, weil kein Typ explizit angegeben ist.

Transaktionstypen
Wie bereits erwähnt, beschreiben Sphären transaktionale Bereiche in einem Prozessmodell. Ein solcher Bereich kann entweder aus einer einzelnen Aktivität, einer Menge von Aktivitäten oder einer zusammengesetzten Aktivität bestehen. Sphären können nach dem Prinzip der offen geschachtelten Transaktionen zusammengesetzt werden. Parallele Sphären bilden also parallele (Sub-)Transaktionen. Enthält eine Sphäre S_1 eine zusammengesetzte Aktivität, welche wiederum eine Sphäre S_2 enthält, wird eine Transaktionshierarchie T_1 und T_{11} erzeugt, wobei T_{11} eine Subtransaktion von T_1 darstellt.

Im Folgenden werden die Symbole für die unterschiedlichen Transaktionstypen zusammen mit einer Beschreibung des Sphärentyps vorgestellt.

Basistransaktion (*basic*). Eine Sphäre, der nicht explizit ein anderer Transaktionstyp zugewiesen wird, ist vom Typ *basic*. Als Grundbausteine einer Sphäre werden Operationen *OP* verwendet. Jede Basistransaktion T_i sichert die ACID-Eigenschaften zu. Muss eine Basistransaktion abgebrochen werden, müssen die Effekte aller bereits ausgeführten Aktivitäten zurückgesetzt werden können. Insbesondere Schreiboperationen auf externen Variablen dürfen nicht zu inkonsistenten Zuständen in der Datenquelle führen.

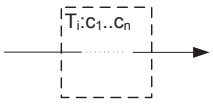


Abbildung 6.4: Transaktionale Sphäre vom Typ *basic*

Abbildung 6.4 zeigt eine Sphäre, die den Transaktionstyp *basic* hat, weil kein Typ explizit zugewiesen ist. Der Sphäre ist die Transaktion T_i zugeordnet, zusammen mit den Constraints c_1 bis c_n.

Kontingent-Transaktionen (*contingent*). Bricht eine Sphäre unerwartet ab, und ist vom Typ *contingent*, hat die Vatertransaktion $T^C_{i_{1,k}}$ (bzw. die umschließende Sphäre) die Möglichkeit, zwischen mehrere alternativen Kontingent-Transaktionen $T_{i,1}, \ldots, T_{i,k}$ auszuwählen, von denen nur eine erfolgreich ausgeführt werden muss. Gelingt dies nicht, muss die Vatertransaktion ebenfalls abgebrochen werden. Abbildung 6.5 zeigt die Notation für eine Transaktion

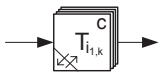


Abbildung 6.5: Sphäre mit dem Typ *contingent*

$T^C_{i_{1,k}}$ (bzw. für eine Sphäre) mit dem Typ *contingent*. Das *C* im oberen rechten Rand steht hier für *contingent*.

Kompensierte Transaktion (*compensating*). In einem offen geschachtelten Transaktionsmodell werden kompensierende Transaktionen benötigt, um eine große Anzahl von „Rückrufaktionen“ anderer Transaktionen zu verhindern, die Änderungen der abzubrechenden Transaktion gelesen haben und gegebenenfalls bereits ihr *EOT* (End of Transaction) erreicht haben [Wei88]. Im Fall eines unerwarteten Abbruchs einer Sphäre mit dem Typ *compensating* kann durch die Definition einer kompensierenden Aktion, der Effekt der Sphäre zurückgesetzt werden. Eine kompensierte Transaktion T_i^{-1} ist also ein Paar aus einer Transaktion T_i und einer **kompensierenden Transaktion** T_i^{-1}. Normalerweise kann die Kompensation mit Hilfe einer inversen Operation erreicht werden. Manche Anwendungsfälle lassen dies aber nicht zu, wie beispielsweise das verschicken einer E-Mail. Dann muss eine geeignete Kompensation durchgeführt werden, indem eine zweite E-Mail verschickt wird.

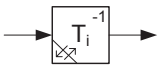


Abbildung 6.6: Sphäre mit dem Typ *compensating*

Abbildung 6.6 zeigt eine Transaktion T_i (bzw. eine Sphäre), für die eine kompensierende Transaktion angegeben ist. Dies wird durch die $^{-1}$ im oberen rechen Rand der Sphäre dargestellt.

Nicht-vitale Transaktion (*non-vital*). Eine nicht-vitale Transaktion T_i^{nv} kann fehlschlagen. Bricht eine nicht-vitale Transaktion (bzw. eine Sphäre) ab, muss dies nicht zum Abbruch der Vatertransaktion (bzw. der Vatersphäre) führen. Dennoch muss auch für eine nicht-vitale Transaktion die Atomarität gewährleistet werden. Das bedeutet, dass partielle Ergebnisse trotzdem zurückgenommen werden müssen!

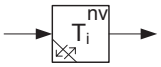


Abbildung 6.7: Aktivität mit dem Typ *non-vital*

Abbildung 6.7 zeigt eine Sphäre T_i vom Typ *non-vital*. Dies wird durch die Buchstaben *nv* im oberen rechten Rand der Aktivität dargestellt.

Lesetransaktionen (*read-only*). Eine Lesetransaktion T_i^{ro} lässt nur lesende Zugriffe auf externe Variablen zu. Das Ändern von Daten ist innerhalb der Transaktion nicht erlaubt. Diese Einschränkung ist in bestimmten Situationen für die Optimierung von Datenzugriffen wichtig. Verfahren, die nach dem Prinzip des Mehrversionen-Concurrency-Control umgesetzt sind, erlauben z. B. die Synchronisation zwischen Lesetransaktionen und gegenüber anderen Transaktionen aufzuheben. Dadurch kann eine Optimierung für den Durchsatz von Lesezugriffen erreicht werden, die mit einer Verringerung von Konflikten einhergeht.

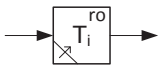


Abbildung 6.8: Aktivität mit dem Typ *read-only*

Abbildung 6.8 zeigt eine Sphäre T_i vom Typ *read-only*. Dies wird durch die Buchstaben *ro* im oberen rechten Rand der Aktivität dargestellt.

Nicht-transaktional (*non-transactional*). In bestimmten Fällen ist es erwünscht, dass der Datenzugriff auf externe Variablen, auch innerhalb einer bereits definierten Sphäre, unabhängig von einer transaktionalen Unterstützung stattfinden soll. Hierfür kann eine Transaktion mit dem Typ *non-transactional* versehen werden. Dann werden alle Lese- und Schreiboperationen sofort und ohne Bedingungen zu beachten durchgeführt. In diesem Fall muss die Anwendung die Integrität der Daten sicherstellen.

6.3.4 Ebene L_3 - Instanz und globale Transaktionen

Ein *YAWL*-Prozessmodell besteht immer aus einem Wurzelnetz, welches die oberste Ebene innerhalb der Prozesshierarchie darstellt. Ein solches Netz kann mehrere *globale* Sphären enthalten. Globale Sphären sind semantisch unabhängig voneinander, sie können sich aber trotzdem beeinflussen. Bricht eine globale Sphäre beispielsweise ab, kann dies zum Abbruch der Prozessinstanz führen.

Wir führen deshalb auf der Ebene L_3 die *Top-Level* Sphäre S_{top} ein. Eine Top-Level Sphäre S_{top} definiert eine transaktionale Sphäre, welche alle globalen Sphären zusammenfasst. Die Top-Level Sphäre repräsentiert also die Prozessinstanz und ermöglicht eine Fehlerbehandlung auf dieser Ebene. Beispielsweise kann hier die Prozessinstanz abgebrochen oder neu gestartet werden.

6.4 Concurrency-Control und Recovery

In den vorherigen Abschnitten wurden die Ebenen des Mehrschichten-Transaktionsmodells *tx+YAWL* vorgestellt. Das Modell wurde mit dem Ziel eingeführt, Aktivitäten eine konsistente Sicht auf externe Daten zu geben. In diesem Abschnitt wird ein Verfahren vorgestellt, das die parallele Abarbeitung von Transaktionen bzw. Sphären ermöglicht. Eine sequentielle Abarbeitung ist in diesem Fall nicht gewünscht, weil dies zu einer streng sequentiellen Abarbeitung von Prozessbereichen führen würde. Weil es nicht Ziel dieser Arbeit war, ein vollständiges Transaktionsmodell für den Zugriff auf externe Datenquellen zu entwickeln, stellt die folgende Lösung nur einen Vorschlag dar.

6.4.1 Mehrebenen-Serialisierbarkeit

Abbildung 6.9 zeigt ein Prozessmodell mit externen Variablen. Neben dem Prozessmodell sind entsprechend dem *tx+YAWL*-Modell, die 4 Ebenen angegeben, die den Datenzugriff durch das WFMS charakterisieren. Für eine parallele Abarbeitung von Transaktionen wird im Vorfeld das Mehrschichten-Modell betrachtet. Das *tx+YAWL*-Modell weist einige Besonderheiten bzw. Einschränkungen gegenüber dem allgemeinen Mehrschichten-Transaktionsmodell auf, die für eine vereinfachte Mehrschichten-Serialisierbarkeit ausgenutzt werden sollen.

In Abbildung 6.9 sieht man, dass nicht alle L_1-Operationen auch zu L_0-Operationen umgesetzt werden. Aus Sicht der Prozessinstanz bzw. der Transaktionen einer Prozessinstanz berücksichtigt dieses Verhalten, dass Datenobjekte innerhalb einer Transaktion nur einmal gelesen und nur einmal geschrieben werden dürfen (siehe Abschnitt 6.3.3).

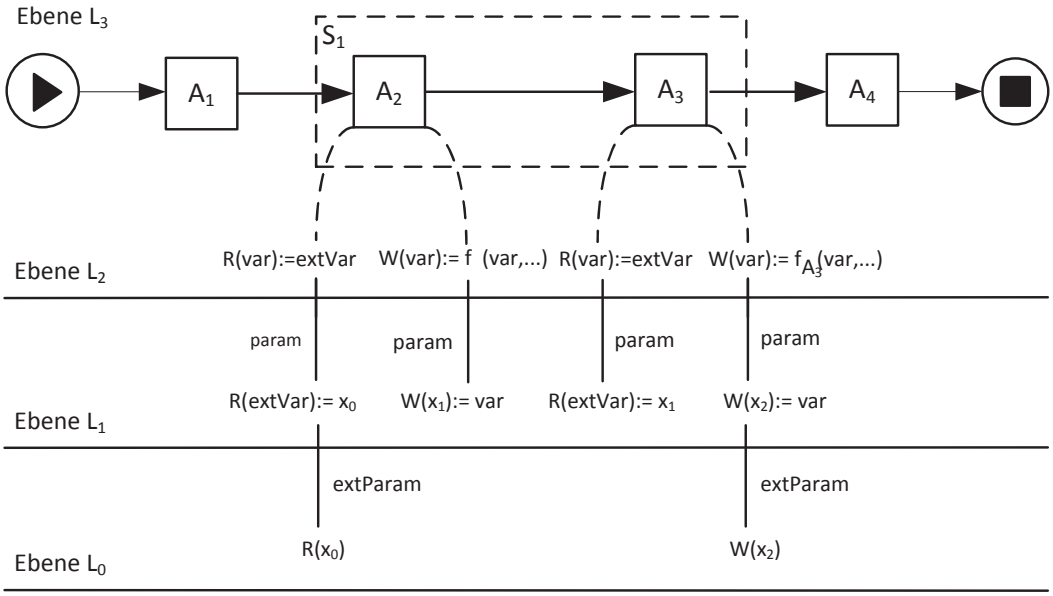


Abbildung 6.9: Mehrebenen-Modell für den Datenzugriff auf externe Datenquellen

Eine weitere Einschränkung besteht darin, dass jeder L_2-Operation genau eine L_1-Operation zugeordnet ist. Eine L_2-Operation ist also ein Spezialfall, für den gilt

$$act(a) = \{a'\}$$

Definition 6.4 (Isolierte Teilbäume [WV01])
Ein Knoten p und der dazugehörige Teilbaum in einem Schedule s sind *isoliert* voneinander, wenn:

- für alle Knoten q, die keine Vorfahren oder Nachfolger von p sind, die Eigenschaft gilt, dass für alle Blätter l von q entweder $l <_s p$ oder $p <_s l$
- für jede $i > 0$ die Nachfolger von p mit dem Abstand i von p total geordnet sind.

Definition 6.5 (Abgeschnittener Teilbaum)
Ein isolierter Teilbaum kann abgeschnitten werden, indem er durch seine Wurzel ersetzt wird [WV01].

Wendet man die Definition 6.4 und 6.5 auf L_2-Operationen an, können alle L_1-Operationen durch L_2-Operationen ersetzt werden. Dies ermöglicht es, Serialisierbarkeit nur noch auf Ebene der L_2-Schedules zu überprüfen und entsprechende Concurrency-Control-Strategien anzuwenden.

Auf Ebene L_2 können für ein Prozessmodell Sphären modelliert werden, wobei jeder Sphäre S_i eine Transaktion T_i zugeordnet ist. Die Operationen der Sphären ergeben einen Schedule s mit der Ordnung $<_s$. Die Ordnung $<_s$ ergibt sich aus der im Kontrollfluss modellierten Rei-

henfolge von Aktivitäten und der Reihenfolge, wie die Parameter einer Aktivität aufgerufen werden.

Definition 6.6 (Reihenfolge Sphären)
Zwei Sphären S_1 und S_2 sind:

- sequentiell: wenn $\forall a_i \in S_1, d_j \in S_2 : a_i < d_j$
- parallel: wenn $(b_2 <_s c_1) \vee (b_2 <_s a_1)$
  wobei b_i den Anfang einer Transaktion T_i bezeichnet.

Ein serialisierter Schedule s' kann anhand semantischer Information aus dem Kontrollfluss bereits im Vorfeld bestimmt werden.

Um die parallele Abarbeitung von Transaktionen auf Ebene L_1 zu verbessern, verwenden wir ein Mehrversionen-Concurrency-Protokoll mit Snapshot Isolation (siehe Definition 6.7). Hierdurch kann ein hoher Grad an Flexibilität gegenüber flachen Transaktionsmodellen erreicht werden.

6.4.2 Mehrversionen-Concurrency-Control

6.4.2.1 Mehrversionen-Schedule

Um die parallele Ausführung von Operationen auf höheren Ebenen zu verbessern, wird der L_1-Arbeitsbereich durch einen Mehrversionen-Cache umgesetzt. Das Prinzip des Mehrversionen-Concurrency-Control (MVCC) wird in [WV01] vorgestellt. Dem MVCC liegt die Idee zugrunde, dass jede Schreiboperation auf einem Datenobjekt eine neue Version dafür anlegt. Somit können Leseoperationen immer auf die jeweils letzte, gültige Version eines Datenobjekts zugreifen. Um sicherzustellen, dass alle Operationen die korrekte Version verwenden, wird hierfür eine Versionsfunktion h eingeführt.

$$h(r_i(x)) \quad = w_j(x) \text{ für ein } w_j(x) <_s r_i(x), \text{ und } r_i(x) \text{ liest } x_j \tag{6.9}$$

$$h(w_i(x)) \quad = w_i(x), \text{ and } w_i(x) \text{ schreibt } x_i \tag{6.10}$$

Die Funktion $h(r_i(x))$ bestimmt die Version von x, welche $r_i(x)$ lesen soll. Dabei wird die Schreibweise $r_i(x_j)$ verwendet. Sie drückt aus, dass eine Version x_j gelesen wird, welche von $w_j(x_j)$ durch die Versionsfunktion h erzeugt wurde. [WV01]

Wir erweitern also den Schedule s so, dass alle L_1-Operationen durch die Versionsfunktion h erweitert werden. So können auf dieser Ebene, durch die Einführung von Versionen, Konflikte

zwischen L_1-Operationen reduziert werden.

$$op_1(s) = h(\bigcup_{i=1}^{n} op_1(t_i)) \tag{6.11}$$

für alle $t \in T$, und für alle Operationen $p, q \in op_1(t)$ gilt: (6.12)

$$p <_t q \Rightarrow h(p) <_s h(q)$$

Jedes Datenobjekt, das durch eine Transaktion verwendet wird, wird durch die Transaktion T_0 mit $w_0(x_0)$ initialisiert.

6.4.2.2 Mehrversionen-Concurrency-Control mittels Snapshot Isolation

Eine Möglichkeit, das Mehrversionen-Concurrency-Control umzusetzen, besteht in der Anwendung des Konzepts der *Snapshot Isolation* (*SI*), welches in [BBG+95] erstmals vorgestellt wurde. Wir verfolgen hier ebenfalls die Idee, dass jede Sphäre bzw. Transaktion ihr eigene konsistente Sicht auf die verwendeten Datenquelle erhält. Die Umsetzung erfolgt in *tx+YAWL* mit einem Cache, der die lokalen Versionen verwaltet. Es wird also für jede Transaktion zum Zeitpunkt, an dem die Transaktion startet, eine Sicht auf die Datenquelle erzeugt (ein Snapshot).

Eine Transaktion T_i, die aus Sequenzen von Leseoperationen $r_i(x)$ und Schreiboperationen $w_i(x)$ besteht, startet mit der Operation b_i (Begin of Transaction). Die Transaktion wird entweder mit einem Commit c_i oder einem Abort a_i beendet. Jeder Transaktion wird weiterhin ein *Readset* RS_i und ein *Writeset* WS_i zugeordnet. Das Readset RS_i enthält alle Datenobjekte, die in T_i gelesen, aber nicht geschrieben werden. Das Writeset WS_i enthält alle Datenobjekte, die in T_i geschrieben werden.

Definition 6.7 (Snapshot Isolation [WV01])
Ein Mehrversionen-Schedule erfüllt das Kriterium der *Snapshot Isolation* wenn gilt:

- Die Versionsfunktion muss jede Leseoperation $r_i(x)$ auf die letzte gültige Version des Datenobjektes abbilden, die durch eine Schreiboperation $w_j(x_j)$ erzeugt wurde.
  $\forall r_i(x_j) : w_j(x_j) < c_j < b_i < r_i(x_j)$
  $\nexists w_h(x_h)$: $w_h(x_h) < b_i$ und $c_j < c_h < b_i$
- Die Writesets von zwei Transaktionen T_i und T_j müssen paarweise disjunkt sein. Es gilt:
  $\forall T_i, T_j$ mit $b_i < b_j < c_i$ oder $b_j < b_i < c_j$, dann dürfen T_i und T_j nicht gleiche Datenobjekte schreiben, d. h. $WS_i \cap WS_j = \varnothing$

Eine Vielzahl von Problemen, die bei der *SI* auftreten können, werden von Fekete et al. in [FLO+05] diskutiert. Entsprechend Definition 6.7 können z. B. bei der *SI* Inkonsistenzen

beim Schreiben auf gleichen Datenobjekten auftreten. Eine Möglichkeit, dem Problem des *Lost-Update* beim *SI* entgegenzuwirken, besteht darin, dass die erste Transaktion mit einem erfolgreichen Commit gewinnt. Die andere Transaktion muss dann zurückgesetzt werden. Im Kontext langlaufender Prozesse ist diese Lösung aber nicht immer sinnvoll. Deshalb stellen wir in Abschnitt 6.4.2.3 eine speziell auf Publikationsprozesse ausgerichtete Behandlung von Schreibkonflikten vor.

6.4.2.3 Behandlung von Schreibkonflikten

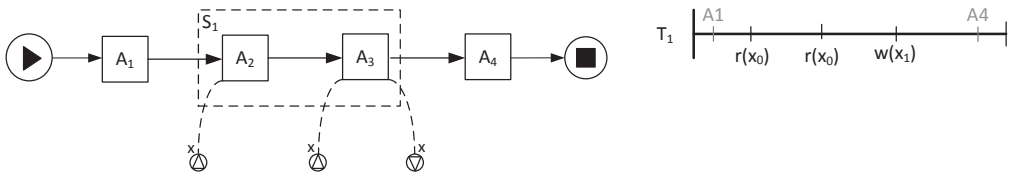
Werden zwei Sphären parallel ausgeführt und schreiben sie auf ein gleiches Datenobjekt, kommt es zu einem Schreibkonflikt. Es gilt also $WS_i \cap WS_j \neq \varnothing$. Ein solches Problem kann im Vorfeld durch Analyse des Prozessmodells erkannt werden. Der Nutzer hat nun die Möglichkeit, den Konflikt aufzulösen. Bleibt ein Schreibkonflikt bestehen, können beide Sphären den Wert schreiben. Die Schreiboperation der Sphäre mit dem letzten Commit überschreibt den Wert der anderen Sphäre. Ein Ausweg wäre die sequentielle Ausführung der beiden Sphären. Dies könnte vom Nutzer explizit angegeben werden.

Um das Prinzip der *SI* im *tx+YAWL*-Modell nutzen zu können, wird Definition 6.3 der Sphäre um ein Readset *RS* und ein Writeset *WS* erweitert. Die beiden Sets werden jeweils zu Beginn jeder Sphäre initialisiert. Die Datenobjekte für das Readset werden vor der ersten Verwendung aus den Datenquellen gelesen. Die Datenquellen im Writeset werden mit „*null*“ initialisiert, sofern gilt $x_i \in WS \ \wedge \ x_i \notin RS$.

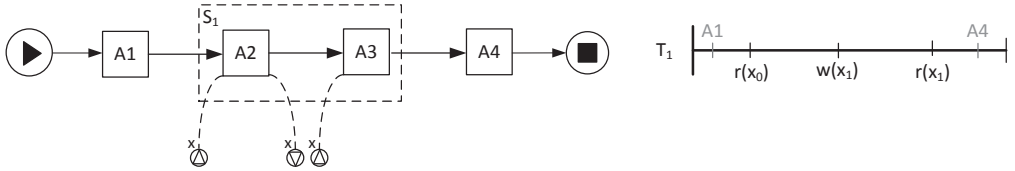
Sequentielle Prozesspfade
Im Folgenden wird anhand mehrerer Fallbeispiele die Nutzung von *SI* im *tx+YAWL*-Modell erläutert. Die ersten drei Fälle in Abbildung 6.10 basieren jeweils auf einem einfachen Prozessmodell mit 4 Aktivitäten A_1,A_2,A_3 und A_4, von denen die beiden Aktivitäten A_2 und A_3 innerhalb der Sphäre S_1 liegen. Je nach Fall greifen die beiden Aktivitäten auf unterschiedliche externe Variablen zu.

Fall 1: In Abbildung 6.10a greifen die Aktivitäten A_2 und A_3 auf die externe Variable x zu. Auf der rechten Seite der Abbildung ist der resultierende Schedule für *Prozess 1* zu sehen. Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$, wobei x_0 mit einem initialen Wert aus der entsprechenden Datenquelle initialisiert wird. Beide Aktivitäten lesen den initialen Wert x_0 aus dem RS_1 von S_1. Anschließend wird der Wert x_1 von A_3 erzeugt. Am Ende von S_1 wird der Wert x_1 in die Datenquelle geschrieben.

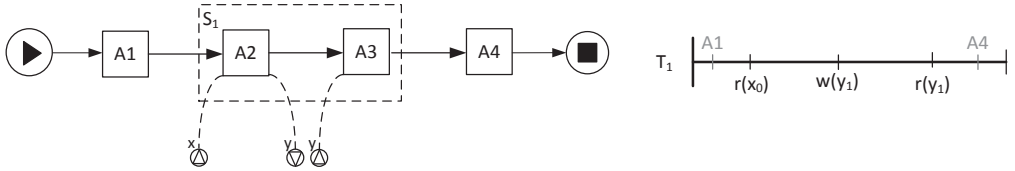
Fall 2: In Abbildung 6.10b greifen die Aktivitäten A_2 und A_3 auf die externe Variable x zu. Auf der rechten Seite der Abbildung ist der resultierende Schedule für *Prozess 2* zu sehen. Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$, wobei x_0 mit einem initialen Wert aus der entsprechenden Datenquelle initialisiert wird.



(a) Prozess 1



(b) Prozess 2



(c) Prozess 3

Abbildung 6.10: Sphären mit sequentiellem Datenzugriff

Im Gegensatz zum Prozessmodell in Abbildung 6.10a liest hier A_2 den Wert x_0 und erzeugt, nachdem die Aktivität abgearbeitet wurde, den neuen Wert x_1. Anschließend wird der Wert x_1 von A_3 gelesen. Am Ende von S_1 wird der Wert x_1 in die Datenquelle geschrieben.

Fall 3: In Abbildung 6.10c greifen die Aktivitäten A_2 und A_3 auf die externen Variablen x, y zu. Auf der rechten Seite der Abbildung ist der resultierende Schedule für *Prozess 3* zu sehen. Für S_1 gilt $RS_1 = \{x_0, y_0\}$ und $WS_1 = \{y\}$, wobei x_0, y_0 mit einem initialen Wert aus der entsprechenden Datenquelle initialisiert werden.

Aktivität A_2 liest den Wert x_0 und erzeugt, nachdem die Aktivität abgearbeitet wurde, den neuen Wert y_1. Anschließend wird der Wert y_1 von A_3 gelesen. Am Ende von S_1 wird der Wert y_1 in die Datenquelle geschrieben.

Probleme treten immer dann auf, wenn nach Definition 6.7 zwei Sphären die gleichen Datenobjekte manipulieren. Dieser Fall soll im nächsten Beispiel erläutert werden.

Fall 4: In Abbildung 6.11 greifen die Aktivitäten A_2 und A_3 auf die externen Variablen x, y zu. Aktivität A_2 ist Teil von Sphäre S_1. Aktivität A_3 ist Teil von Sphäre S_2. Nach Definition 6.6 sind S_1 und S_2 parallel. Auf der rechten Seite der Abbildung sind zwei mögliche Schedules für die Abarbeitung zu sehen. Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$, für S_2 ist $RS_2 = \varnothing$ und $WS_2 = \{x\}$. Das Datenobjekt $x_0 \in S_1$ wird mit einem initialen Wert aus der entsprechenden Datenquelle initialisiert. Aktivität A_2 liest den Wert x_0 und erzeugt, nachdem die Aktivität

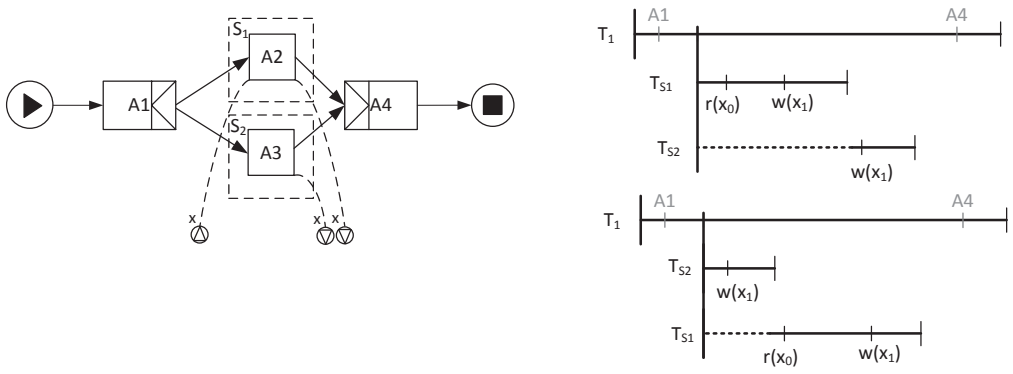


Abbildung 6.11: Prozess mit parallelen Sphären

abgearbeitet wurde, den neuen Wert x_1. Anschließend wird der Wert x_1 von A_3 geschrieben.

Weil im Beispiel die Eigenschaft $WS_1 \cap WS_2 = \{x\}$ gilt, entsteht ein Schreibkonflikt zwischen den beiden Sphären. In diesem Fall setzt sich die Sphäre durch, die als letztes ein Commit gemacht hat.

Die beiden Schedules auf der rechten Seite zeigen eine mögliche serielle Ausführung der beiden Sphären, indem eine von beiden verzögert ausgeführt wird. Sphäre S_1 schreibt entsprechend den Wert x_1 in die Datenquelle. Sphäre S_2 schreibt ebenfalls die lokale Version x_1 in die Datenquelle.

Parallele Prozesspfade

Innerhalb einer Sphäre bzw. einer Transaktion, die für eine Sphäre erzeugt wird, ergibt sich die Ordnung der Operationen direkt aus dem Kontrollfluss. Sind parallele Prozesspfade innerhalb einer Sphäre modelliert, und greifen so parallele Aktivitäten auf dieselben externen Variablen zu, kann es ebenfalls zu Konflikten zwischen Lese- und Schreiboperationen kommen. Parallele Prozesspfade, welche die gleichen Datenobjekte schreiben, stehen deshalb ebenfalls in Konflikt zueinander. Grundsätzlich ist es problematisch in einem Prozessmodell parallele Prozesspfade zu modellieren, welche auf die gleichen Datenobjekte zugreifen. Werden keine weiteren Vorkehrungen getroffen, ist eine konsistente Sicht auf die Datenobjekte nicht immer möglich. Probleme treten auf, wenn Datenobjekte mehrfach parallel geschrieben werden oder Aktivitäten unterschiedliche Sichten auf Datenobjekte haben. Weil es im Vorfeld schwierig ist, die Dauer für die Abarbeitung einer Aktivität zu bestimmen, kann für parallele Prozesspfade keine feste Reihenfolge für Aktivitäten bestimmt werden.

Anhand von Fallbeispielen soll gezeigt werden, wie parallele Prozesspfade im *tx+YAWL*-Modell behandelt werden. Die Beispielfälle bauen alle auf einem Prozessmodell mit den vier Aktivitäten A_1, A_2, A_3 und A_4 auf, wovon A_2 und A_3 parallel ausgeführt werden. Die Sphäre S_1 enthält alle vier Aktivitäten. Ziel ist, durch geeignete Strategien, eine Operationen-

Reihenfolge zu erzeugen, für die alle Aktivitäten innerhalb der Sphäre eine konsistente Sicht auf die Variablen haben.

Deshalb werden parallele Prozesspfade intern als Subtransaktionen umgesetzt. Im Gegensatz zu Sphären sind die internen Subtransaktionen als geschlossen geschachtelte Transaktionen realisiert. Wir schlagen an dieser Stelle vor, die Struktur des Prozessmodells auszunutzen um eine korrekte Reihenfolge der Operationen zu erzeugen. Die Prozessstruktur ist im Vorfeld bekannt und kann somit für die Realisierung einer einfachen Strategie ausgenutzt werden.

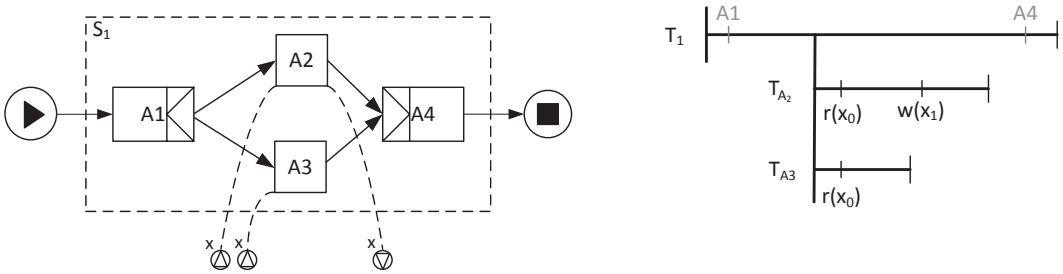


Abbildung 6.12: Prozessmodell mit parallelen Prozesspfaden in einer Sphäre ohne Konflikte

Fall 5: In Abbildung 6.12 greifen die Aktivitäten A_2, A_3 auf die externe Variable x zu. Auf der rechten Seite der Abbildung ist der resultierende Schedule zu sehen. Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$, wobei x_0 mit einem initialen Wert aus der entsprechenden Datenquelle initialisiert wird.

Aktivität A_2 liest den Wert x_0 und erzeugt, nachdem die Aktivität abgearbeitet wurde, den neuen Wert x_1. Aktivität A_3 liest ebenfalls den Wert x_0. Am Ende von S_1 wird der Wert x_1 in die Datenquelle geschrieben. In diesem Fall ist die Reihenfolge für die Abarbeitung der Aktivitäten A_2 und A_3 nicht von Bedeutung. Dieser Fall entspricht einer seriellen Ausführung $T_{A_2} < T_{A_3}$ oder $T_{A_3} < T_{A_2}$.

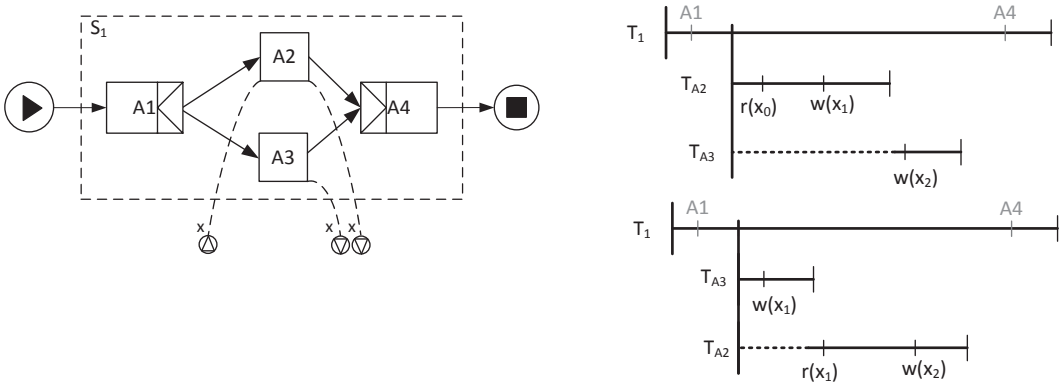


Abbildung 6.13: Prozessmodell mit parallelen Prozesspfaden in einer Sphäre mit Lese-Schreib-Konflikt

Fall 6: In Abbildung 6.13 greifen die Aktivitäten A_2, A_3 auf die externe Variable x zu.

Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$. In diesem Fall liest Aktivität A_2 den Wert von x und schreibt nach erfolgreicher Abarbeitung einen neuen Wert für x. Die parallele Aktivität A_3 schreibt nach erfolgreicher Abarbeitung einen neuen Wert für x.

Die Leseoperation von A_2 steht mit der Schreiboperation von A_3 in Konflikt. In diesem Fall muss die Abarbeitung des einen Pfades verzögert werden. Hat, wie in der ersten Situation, Aktivität A_2 bereits x gelesen, muss die Schreiboperation von A_3 solange verzögert werden, bis A_2 geschrieben hat. Dies entspricht dann der seriellen Ausführung $T_{A_2} < T_{A_3}$. Hat A_3 bereits geschrieben, müssen keine weiteren Vorkehrungen getroffen werden. Dies entspricht dann der seriellen Ausführung $T_{A_3} < T_{A_2}$. Allerdings geht in jedem Fall ein Zwischenergebnis für x verloren. Dies muss beim Modellieren eines solchen Prozessmodells klar sein.

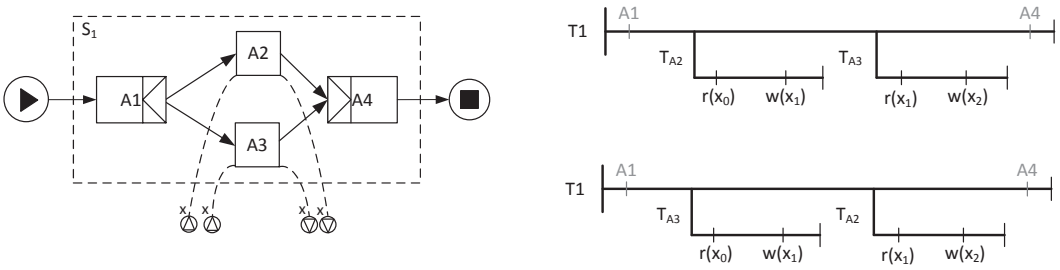


Abbildung 6.14: Prozessmodell mit parallelen Prozesspfaden in einer Sphäre mit Lese-Schreib-Lese-Konflikt

Fall 7: In Abbildung 6.14 greifen die Aktivitäten A_2, A_3 beide lesend und schreibend auf die externe Variable x zu.

Für S_1 gilt $RS_1 = \{x_0\}$ und $WS_1 = \{x\}$. In diesem Fall liest Aktivität A_2 den Wert von x und schreibt nach erfolgreicher Abarbeitung einen neuen Wert für x. Die parallele Aktivität A_3 liest ebenfalls den Wert x und schreibt nach erfolgreicher Abarbeitung einen neuen Wert für x.

Die Leseoperation von A_2 steht mit der Schreiboperationen von A_3 in Konflikt und umgekehrt. In diesem Fall bleibt nur die Möglichkeit, eine serielle Reihenfolge zu erzwingen, indem beide Aktivitäten sequentiell ausgeführt werden.

Entweder wird $T_{A_2} < T_{A_3}$ erzwungen, wie im oberen Beispiel von Abbildung 6.14, oder es wird $T_{A_3} < T_{A_2}$ realisiert, wie im unteren Beispiel von Abbildung 6.14.

Fazit

In diesem Abschnitt wurde der Zugriff auf externe Variablen in Prozessstrukturen hinsichtlich auftretender Schreibkonflikte untersucht. Dafür wurden unterschiedlich Kontrollflussstrukturen im Zusammenhang mit den Zugriffsoperationen betrachtet. Durch die Auswertung

der Prozessmodelle können auftretende Konflikte bereits im Vorfeld erkannt werden. In den Fällen 4, 5 und 6 kann durch eine verzögerte Ausführung von Aktivitäten die korrekte Abarbeitung sichergestellt werden. Der beschriebene Konflikt im Fall 7 kann nur durch eine sequentielle Ausführungsreihenfolge aufgelöst werden.

In Kapitel 7 wird der Ansatz *FlexY* vorgestellt, der eine flexible, datenabhängige Generierung von Prozessmodellen umsetzt. Auf der Basis von Prozessbausteinen werden dort, in Abhängigkeit von externen Datenobjekten, Prozessfragmente generiert. Die generierten Kontrollflussstrukturen setzen sich aus sequentiell und parallel abzuarbeitenden Prozessbausteinen zusammen. Deshalb sind die vorgestellten Strukturen besonders für die Flexibilisierung relevant. Dort können durch die Anwendung der vorgestellten Lösungen Schreibkonflikte innerhalb der generierten Kontrollflussstrukturen erkannt werden. Außerdem können Schreibkonflikte vermieden werden, indem diese bei der Generierung der Kontrollflussstrukturen beachtet werden.

6.4.3 Recovery

In einem WFMS können während der Abarbeitung von Prozessen verschiedene Fehler auftreten. In [EL96] erfolgt z. B. eine Unterteilung in *Workflow-Engine Fehler*, *Fehler von Aktivitäten* und *Kommunikationsfehler zwischen Scheduler und Aktivitäten*. Ziel des Recovery ist es, den letzten konsistenten Zustand nach einem Fehler zu erreichen. Wir betrachten im Folgenden die Fehler *Datasource Unavailable* und *ExternalConstraint Violation* und welchen Einfluss diese Fehler auf transaktionale Sphären haben. Wird eine Sphäre aufgrund eines solchen Fehlers abgebrochen, muss ein geeignetes Recovery gestartet werden. Es muss sichergestellt werden, dass keine verwendete Datenquelle in einen inkonsistenten Zustand überführt wird. Außerdem müssen alle lokalen Versionen geeignet zurückgesetzt werden. Weil es nicht Ziel dieser Arbeit war, ein vollständiges Transaktionsmodell für den Zugriff auf externe Datenquellen zu entwickeln, beschränken wir uns hier auf Fehler, die durch Aktivitäten hervorgerufen werden. Im Folgenden wird daher auch kein vollständiges Konzept vorgestellt.

6.4.3.1 Fehlerarten

Die bei der Ausführung von Aktivitäten auftretenden Fehler können in *Systemfehler* und *semantische Fehler* unterteilt werden [EL96]. Systemfehler treten auf, wenn z. B. das System abstürzt. Im Gegensatz dazu beschreiben semantische Fehler Ausnahmen während der Abarbeitung des Prozesses. Eine Ausnahme kann z. B. eine abgebrochene Merkmalsextraktion aufgrund eines falschen Datenformats sein, die zu einem Abbruch einer Aktivität führt.

In [AHAE07] stellen Adams et al. eine erweiterte Menge von insgesamt sieben unterschiedlichen Ausnahmetypen vor. Vier verschiedene *Constraint Types* beschreiben Regeln, die vor und nach der Ausführung von Aktivitäten oder Prozessinstanzen überprüft werden können. Die Ausnahme *Time Out* tritt auf, wenn eine Aktivität nach einer vorgegebenen Zeit nicht beendet wurde. Die Ausnahme *Item Abort* wird erzeugt, wenn die Aktivität während der Abarbeitung abgebrochen wird. Durch ein externes Ereignis kann die Ausnahme *Externally Triggered* erzeugt werden. Wenn eine Resource nicht erreichbar ist, wird die Ausnahme *Resource Unavailable* erzeugt. Werden Bedingungen an Daten verletzt, wird die Ausnahme *Constraint Violation* erzeugt. Die Fehlerarten aus [AHAE07] werden durch das *YAWL*-WF-MS unterstützt und werden deshalb auch für *tx+YAWL* verwendet. Zu den oben genannten Ausnahmen müssen zusätzlich Fehler betrachtet werden, die im Zusammenhang mit dem externen Datenzugriff in *tx+YAWL* stehen. In *tx+YAWL* wird ein Cache verwendet, der das Konzept des MVCC umsetzt (siehe Abschnitt 6.4.1). So können unterschiedliche Versionen der Datenobjekte verwaltet werden. An dieser Stelle müssen keine Fehler betrachtet werden, die aufgrund von Versionskonflikten oder blockierten Aktivitäten auftreten können. Diese Konflikte können durch verzögerte Ausführung von Aktivitäten aufgelöst werden.

Wir führen die folgenden zwei Fehlertypen ein:

- *Datasource Unavailable*: Die externe Datenquelle ist nicht erreichbar.
- *ExternalConstraint Violation*: Eine Integritätsbedingung, die an eine Sphäre gebunden ist, wurde verletzt.

6.4.3.2 Recovery nach einem Fehler

Im *tx+YAWL*-Modell bieten transaktionale Sphären die Möglichkeit, Aktivitäten zu einer (Teil-)Transaktion zusammenzusetzen. Jeder dieser Aktivitäten kann eine transaktionale Eigenschaft bzw. ein Transaktionstyp zugewiesen werden (siehe Abschnitt 6.3.3). Schlägt die Abarbeitung einer Aktivität fehl, hat das unterschiedliche Auswirkungen auf den gesamten Prozessverlauf.

Lokale Versionen externer Daten
Wird eine Sphäre abgebrochen, müssen alle lokalen Versionen der verwendeten externen Variablen gelöscht werden. Es ist nicht notwendig, kompensierende Operationen auszuführen, die Änderungen in den Datenquellen zurücknehmen, weil lokale Versionen erst nach einem Commit in die Datenquelle geschrieben werden (siehe Abschnitt 6.3.2).

Im *tx+YAWL*-Modell wird ein offen geschachteltes Transaktionsmodell verwendet. Ist eine Sphäre Teil einer anderen Sphäre (Subsphäre), hat ihr Transaktionstyp Einfluss auf die Vatertransaktion und deren Abbruchverhalten. In *tx+YAWL* wurden verschiedene Transaktionstypen eingeführt. Je nach Transaktionstyp, führt ein Fehler zu unterschiedlichen Recovery-Maßnahmen (siehe Abschnitt 6.3.3).

Eine Sphäre kann Subsphären enthalten, die vor dem Auftreten eines Fehlers bereits ein lokales Commit durchgeführt haben. Um kaskadierenden Abbrüchen entgegenzuwirken, wird in der Literatur häufig der Einsatz von kompensierenden Operationen vorgeschlagen (z. B. *spheres of joint compensation* [Ley95] oder *ConTracts* [WR92]. In Abschnitt 6.3.3 wurde deshalb der Typ *Kompensierte Transaktion* eingeführt. Damit können kompensierbare Operationen definiert werden, die im Fall eines Abbruchs ausgeführt werden.

Für die Umsetzung von kompensierenden Operationen nutzt *tx+YAWL* die Möglichkeiten des *Exlet*-Ansatzes. Wie oben bereits gesagt wurde, wird dieser von *YAWL* unterstützt.

Fazit
In den Abschnitten 6.3 und 6.4 wurde ein Mehrebenen-Transaktionsmodell unter Nutzung transaktionaler Sphären vorgestellt. Sphären haben folgende Eigenschaften:

- Einer Sphäre wird ein Transaktionstyp zugeordnet: {*non-transactional, basic, read-only, non-vital, compensating, contingent*}.
- Sphären werden nach dem Prinzip offen geschachtelter Transaktionen zusammengesetzt. In einem Prozessmodell bilden parallele Sphären parallele Transaktionen und geschachtelte Sphären werden auf eine entsprechende Transaktionshierarchie abgebildet.
- Parallele Prozesspfade innerhalb einer Sphäre werden als interne Subtransaktionen umgesetzt, die als geschlossen geschachtelte Transaktion realisiert sind.

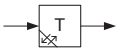
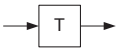
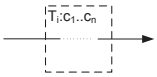
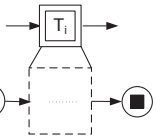
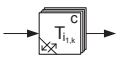
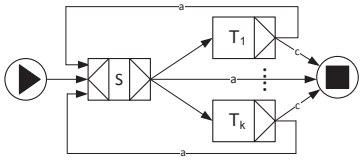
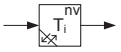
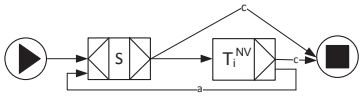
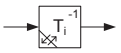
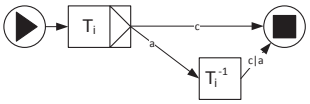
Gegenüber dem allgemeinen Mehrschichten-Transaktionsmodell werden folgende Eigenschaften von *tx+YAWL* für die Umsetzung einer vereinfachten Mehrschichten-Serialisierbarkeit ausgenutzt:

- Datenobjekte werden innerhalb einer Transaktion nur einmal gelesen und nur einmal geschrieben.
- Jeder L_2-Operation wird genau einer L_1-Operation zugeordnet.
- Serialisierbarkeit und entsprechende Concurrency-Control-Strategien müssen nur auf Ebene der L_2-Schedules überprüft und angewendet werden.

6.5 *tx+YAWL* zu *YAWL* Transformation

Das vorgestellte Konzept der vier Ebenen von *tx+YAWL* wird von *YAWL* nicht direkt unterstützt. Deshalb stellen wir eine Methode vor, die die erweiterten Prozessmodelle in ein Standard-*YAWL* Modell transformiert. Damit die Modellierungskonzepte, die in den Ebenen L_2 und L_3 von der *YAWL*-Engine ausgeführt werden können, müssen diese in ein *YAWL*-Modell transformiert werden. Tabelle 6.1 zeigt die grundlegenden Elemente des Modells.

Tabelle 6.1: *tx+YAWL* Konzepte und die abgeleitete *YAWL*-Semantik

tx+YAWL	**Beschreibung**	***YAWL*-Repräsentation**
	Transaktion T vom Typ *basic* (mit ACID-Eigenschaften).	
	Transaktionale Sphäre T_i mit Integritätsbedingungen c_i.	
	Kontingent-Transaktion $T^C_{i_{1,k}}$.	
	Nicht-vitale Transaktion T^{nv}.	
	Kompensierte Transaktion T mit Kompensation T^{-1}.	

In Abhängigkeit der modellierten Transaktionstypen werden unterschiedliche Abbildungen auf *YAWL*-Konstrukte verwendet.

Transaktionale Sphäre. Besteht der transaktionale Bereich (Sphäre) aus einer einzelnen Aktivität, wird diese in eine *YAWL*-Aktivität überführt. Ist eine transaktionale Sphäre über mehrere Aktivitäten und Prozesspfade definiert, wird diese in eine komplexe Aktivität T_i mit Subnetz SN_i umgewandelt. Das Subnetz enthält alle Aktivitäten, welche innerhalb der Sphäre liegen. Wurden für die Sphäre Integritätsbedingungen definiert, wird ein *Data Controller* (siehe Abschnitt 8.2) für das Subnetz SN_i erzeugt. Der DC beschreibt die Menge der externen Variablen, die innerhalb der Sphäre definiert wurden, zusammen mit den Integritätsbedingungen. Dies gilt auch für alle weiteren *tx+YAWL*-Sphären-Typen, für die Integritätsbedingungen definiert wurden.

Kontingent-Transaktion. Eine Kontingent-Transaktion wird ebenfalls in eine komplexe Aktivität T_i mit Subnetz SN_i umgewandelt. Das Subnetz SN_i enthält eine Menge von Subtransaktionen $\{T_1, \ldots, T_k\}$. Die Systemaktivität S ermöglicht nun die Ausführung einer Subtransaktion. Schlägt deren Ausführung mit $T_k.a()$ fehl, kann durch S ein neuer Versuch mit der nächsten Subtransaktion durchgeführt werden. Wenn eine der Subtransaktionen erfolgreich mit $T_k.c()$ ausgeführt wurde, dann wird die Kontingent-Transaktion mit einem Commit beendet. Führt keine der Subtransaktionen T_k zu einem erfolgreichen Commit, muss S mit einem *Abort* die gesamte Transaktion $T_{i_{1,k}}$ abbrechen.

Nicht-vitale Transaktion. Für eine Nicht-vitale Transaktion T_i^{nv} wird eine komplexe Aktivität T_i mit Subnetz SN_i erzeugt. In diesem Fall enthält das Subnetz SN_i eine Systemaktivität S, welche die Aktivität T^{nv} mehrmals hintereinander ausführen kann. Führt das zu keiner korrekten Ausführung der Aktivität T^{nv} kann S mit einem Commit die Transaktion beenden.

Transaktion mit Kompensation. Eine Transaktion mit Kompensation wird auch in eine komplexe Aktivität T_i mit einem Subnetz SN_i umgewandelt. Das Subnetz SN_i enthält die Transaktion T_i und die kompensierende Aktivität T_i^{-1}. Wird die Transaktion mit $T_i.a()$ abgebrochen, wird die kompensierende Transaktion T_i^{-1} ausgeführt.

Beispiel

Die vorgestellten Regeln für eine Abbildung von *tx+YAWL* auf *YAWL*-Konzepte soll nun anhand eines kurzen Beispiels gezeigt werden. Abbildung 6.15 stellt ein Prozessmodell

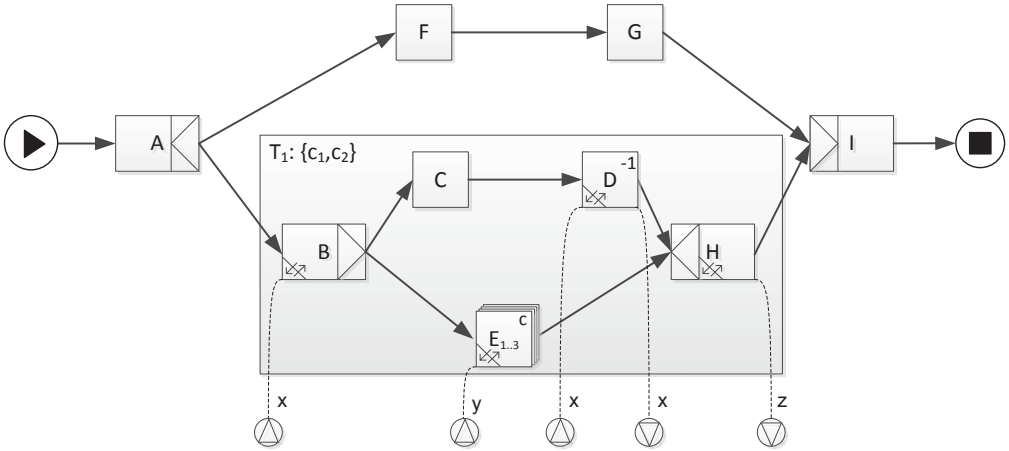


Abbildung 6.15: *tx+YAWL* Prozessmodell

dar, dass mit *tx+YAWL*-Konzepten modelliert ist. Eine Abbildung auf *YAWL*-Konzepte unter

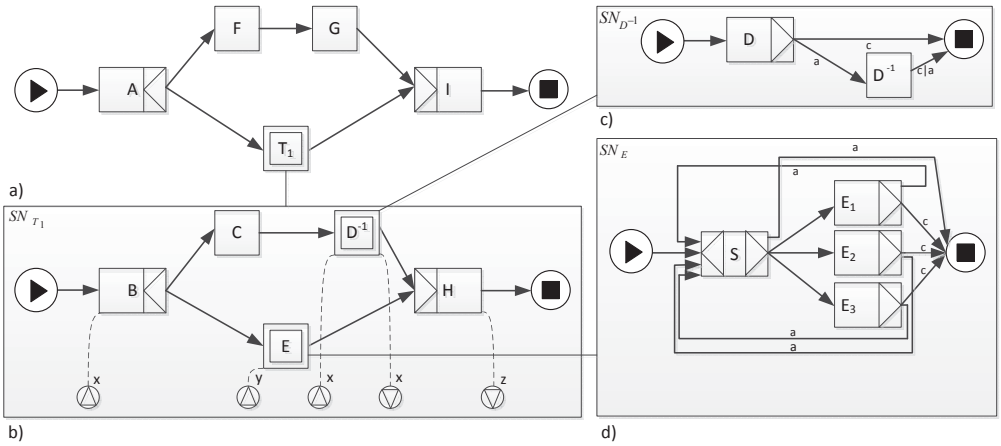


Abbildung 6.16: *YAWL* Prozessmodell, das aus *tx+YAWL*-Modell erzeugt wird

Anwendung der oben beschriebenen Regeln ist in Abbildung 6.16 dargestellt.

Beispiel 6.3 (Abbildung *tx+YAWL* auf *YAWL*-)
Die Top-Level-Transaktion aus Abbildung 6.15 wird durch ein *YAWL*-Netz in Abbildung 6.16 a) dargestellt. Die transaktionale Sphäre T_1 wird als Subnetz SN_{T_1} der komplexen Aktivität T_1 in Abbildung 6.16 b) umgesetzt. Das Subnetz SN_1 enthält die Aktivitäten B, C, D, E und H, welche Teil der Sphäre T_1 sind. Innerhalb des Subnetzes SN_1 wird die kompensierbare Transaktion D^{-1} als komplexe Aktivität D^{-1} und das Subnetz $SN_{D^{-1}}$ modelliert. Das Subnetz $SN_{D^{-1}}$ ist in Abbildung 6.16 c) dargestellt. Es enthält die Aktivität D und die dazugehörige Kompensationstransaktion D^{-1}, welche im Fall eines Fehlers ausgeführt wird. Das Subnetz SN_1 enthält außerdem die Kontingent-Transaktion E_3, die als komplexe Aktivität E mit dem Subnetz SN_E umgesetzt ist. Das Subnetz SN_E in Abbildung 6.16 d) enthält drei Kontingent-Transaktionen bzw. Aktivitäten E_1, E_2, E_3, die für die Ausführung zur Verfügung stehen. Die externen Variablen der transaktionalen Aktivitäten bzw. Sphären werden als Netzvariablen innerhalb der Subnetze $SN_{D^{-1}}$ und SN_E umgesetzt.

Für die Sphäre T_1 wird weiterhin ein DC erzeugt. Der DC wird an das Subnetz SN_{T_1} gebunden und entsprechend aktiviert, wenn das Subnetz gestartet, deaktiviert oder beendet wird. Die Menge der Integritätsbedingungen ist gegeben durch $C = \{c_1, c_2\}$. Entsprechend ist die Menge der zu überwachenden externen Variablen gegeben durch $E = \{x, y, z\}$.

6.6 Diskussion

In der Literatur werden verschiedene Ansätze betrachtet, die Transaktionen in Zusammenhang mit Workflow-Management bringen. Einige dieser Ansätze werden im Folgenden vorgestellt und *tx+YAWL* gegenüber diesen abgegrenzt.

Das Konzept der Transaktion wird im Zusammenhang mit Workflow-Management für unterschiedliche Anwendungsfälle genutzt. Eine Reihe von Ansätzen verwenden Transaktionen, um die Ausnahmebehandlung bei auftretenden Fehlern in einer Prozessinstanz zu verbessern. In diesen Ansätzen werden vordergründig Techniken betrachtet, um bekannte Fehlersituationen einer laufenden Prozessinstanz zu erkennen und um zu entscheiden wie auf diese Fehler geeignet reagiert werden kann. Um Atomarität und Rücksetzbarkeit von Prozessinstanzen zu gewährleisten, werden unterschiedliche Recovery-Maßnahmen eingeführt. Isolationseigenschaften werden aufgrund der Prozesseigenschaften häufig abgeschwächt. Konzepte für die Kompensation von Aktivitäten sind daher sehr wichtig.

In [Ley95] stellt Leymann das Konzept „*spheres of joint compensation*“ vor. Eine Sphäre beschreibt hier eine Menge von Aktivitäten, die im Fehlerfall mit Hilfe von Kompensationsaktionen zurückgesetzt werden. Jeder Aktivität und jeder Sphäre wird eine eigene

Kompensationsaktion zugeordnet. Sphären ermöglichen so die Definition transaktionaler Bereiche. Die atomaren Schritte einer Sphäre sind Aktivitäten (mit ACID-Eigenschaften). Für das sogenannte *Backout* einer Sphäre stehen die Verfahren *retry*, *undo* und *return* zur Verfügung. Im Gegensatz zu *tx+YAWL* können Sphären nicht nur geschachtelt werden, sondern sich auch überlappen. Muss eine Sphäre kompensiert werden, müssen alle Kompensationsaktionen in umgekehrter Reihenfolge der bereits ausgeführten Aktivitäten ausgeführt werden. Für eine korrekte Kompensation von Aktivitäten und Sphären werden Kompensations-Container definiert. Sie enthalten Daten, die für die Kompensation benötigt werden. Die Isolation wird aufgeweicht, indem Zwischenergebnisse für alle Aktivitäten einer Prozessinstanz zur Verfügung stehen. Aktivitäten können Zwischenergebnisse auch an externe Datenquellen weitergeben. Die Datenzugriffe sind aber beschränkt auf workflow-relevante Daten. Sie können nicht kontrolliert werden, weil die Aktivitäten als *Black-Boxes* behandelt werden. Um im Fehlerfall inkonsistente Zustände zu vermeiden, können Datenabhängigkeiten zwischen Sphären beachtet werden. Existieren solche Abhängigkeiten, können kaskadierende Abbrüche von Sphären angestoßen werden. Integritätsbedingungen über transaktionalen Bereichen, wie sie im *tx+YAWL*-Model umgesetzt sind, können hier nicht angegeben werden. Außerdem wird in diesem Ansatz keine Nebenläufigkeit betrachtet. Der gleichzeitige Zugriff auf Daten wird daher nicht berücksichtigt.

ConTracts ist ein Konzept für langlaufende Transaktionen, das in [WR92, RS95] vorgestellt wird. Ein *ConTract* beschreibt einen Prozess, der aus einer Menge von Schritten besteht. Die Abarbeitung dieser Schritte wird durch ein *Skript* beschrieben. Einem *ConTract* stehen lokale Variablen (Kontextelemente) zur Verfügung, die in einem transaktionalen Speicher abgelegt werden. Lokale Variablen sind dabei nur für einen *ConTract* sichtbar und können nur durch dessen Schritte verändert werden. Zwischenergebnisse können anderen Schritten zur Verfügung gestellt werden. Um den nebenläufigen Zugriff auf die Zwischenergebnisse (Objekt) zu ermöglichen, können Bedingungen bzw. Invarianten eingefügt werden. Eine Invariante beschreibt eine Bedingung an ein Objekt, die nach dem Ausführen eines Schrittes erfüllt sein muss. Ist eine Invariante verletzt, können Kompensationsschritte ausgeführt werden. So kann nur auf inkonsistente Datenzustände in Form von Nachbedingungen einzelner Schritte reagiert werden. In *tx+YAWL* besteht zusätzlich die Möglichkeit Bedingungen an Prozessbereiche zu stellen. Die atomaren Operationen eines *ConTracts* sind Schritte, die ACID-Eigenschaften haben. Eine Unterteilung in Lese- und Schreiboperationen wie im *tx+YAWL*-Modell ist nicht möglich. Atomarität wird abgeschwächt, indem *forward recovery* unterstützt wird.

Panta Rhei ist ein von Eder et al. vorgestelltes WFMS, welches transaktionale Konzepte umsetzt [EGL98]. Das WFMS basiert auf dem *Workflow Activity Model (WAMO)* [EL95, EL96]. Transaktionale Eigenschaften können im WAMO-Modell an Aktivitäten gebunden werden. In erster Linie wird unterschieden, ob eine Aktivität *vital* oder *nicht-vital* ist. Bricht

eine *nicht-vitale* Aktivität mit einem Fehler ab, kann der (Sub-)Prozess ungehindert weiter ausgeführt werden. Um Atomarität und Isolationseigenschaften sicherzustellen, wird das Konzept der Kompensation verwendet. Bricht eine *vitale* Aktivität ab, wird der (Sub-)Prozess kompensiert. Transaktionale Bereiche können durch eine geeignete Schachtelungsstruktur der Prozessmodelle erreicht werden. Dies entspricht dem Konzept der Sphären im *tx+YAWL*-Modell. Kontingent-Transaktionen können mit Hilfe von speziellen Kontrollflusskonstrukten (*free choice* und *ranked choice*) umgesetzt werden. So können Prozesspfade (Kontingente) modelliert werden, die im Fehlerfall ausgeführt werden. Für Recovery-Informationen können workflow-relevante Daten verwendet werden. Dafür besteht die Möglichkeit, diese Daten zu loggen und im Fehlerfall kompensierenden Aktivitäten zur Verfügung zu stellen. Weil der Ansatz Transaktionen für die Fehlerbehandlung einsetzt, werden keine Konzepte für die Datenkonsistenz angeboten. Nebenläufigkeit wird in Form von Zugriffsrechten auf die lokalen Datenelemente ermöglicht. Der Datenaustausch mit externen Datenquellen kann nicht kontrolliert werden.

In [AHST97] stellen Alonso et al. das WFMS *OPERA* vor. Transaktionale Konzepte werden in *OPERA* an Sphären gebunden. Es werden drei unterschiedliche Arten von Sphären [Alo97] unterschieden. Die *spheres of atomicity* beschreiben Bereiche, für die Atomarität gefordert wird (*ganz oder gar nicht*). Die *spheres of persistence* beschreiben Bereiche, in denen die Schritte aller enthaltenen Aktivitäten persistent gemacht werden müssen. So wird ein *forward recovery* ermöglicht. Die *spheres of isolation* ermöglichen eine konsistente Bearbeitung von Prozessdaten. Werden die *spheres of isolation* zusammen mit *spheres of atomicity* genutzt, entspricht das dem Konzept der geschachtelten Mehrebenen-Transaktionen. In [HA00] stellen Hagen und Alonso die Umsetzung der *spheres of atomicity* vor. Wie in *tx+YAWL* werden verschiedene Transaktionstypen eingeführt, die die Eigenschaften der Aktivitäten beschreiben. Der Ansatz geht über den *tx+YAWL*-Ansatz hinaus, weil mehr Transaktionstypen definiert werden. So kann beispielsweise zwischen kompensierbaren und rücksetzbaren Aktivitäten unterschieden werden. Außerdem besteht die Möglichkeit, *kritische Aktivitäten* zu beschreiben. Wenn eine Sphäre nicht kompensierbare Aktivitäten enthält, ist eine *kritische Aktivität* die erste nicht kompensierbare Aktivität. Integritätsbedingungen über transaktionalen Bereichen können nicht beschrieben werden.

In [SABS02] werden die transaktionalen Eigenschaften Atomarität und Isolation im Kontext von WFMS, untersucht. Um eine flexible Fehlerbehandlung zu ermöglichen, werden Atomarität und Isolation nicht unabhängig betrachtet. Atomarität wird auf der Ebene von Aktivitäten definiert. Datenzugriffe und Integritätsbedingungen werden in [SABS02] nicht beachtet. Die Ausnahmebehandlung geht über die Möglichkeiten von *tx+YAWL* hinaus. Es können Beziehungen zwischen kompensierenden Aktivitäten und den *normalen* Aktivitäten beschrieben werden. Die Terminierungseigenschaften von Aktivitäten werden in *kompensierbar*, *wiederholbar* und *pivot* unterschieden. Wenn eine Aktivität nicht kompensierbar ist,

dann ist sie vom Typ *pivot*. Für eine *wiederholbare* Aktivität muss nach *n* Wiederholungen eine korrekte Abarbeitung garantiert werden. Ein Prozessmanager stellt unter Beachtung von Terminierungseigenschaften der Aktivitäten, dem Status der Prozessinstanzen und den Abhängigkeiten zwischen Aktivitäten eine korrekte Abarbeitungsreihenfolge der Aktivitäten her. Sperren auf Ebene der Aktivitäten unterstützen die verschränkte Ausführung.

In [PML07] wird ein Prozessmodell für die koordinierte Ausführung von *BPEL scopes* vorgestellt. In BPEL können transaktionale Prozessbereiche in einem Prozessmodell mit Hilfe von *scopes* beschrieben werden. Ähnlich wie Sphären in *tx+YAWL*, können so Aktivitäten zu einer Transaktion zusammengefasst werden. Der Ansatz ermöglicht eine bessere Kompensation von *scopes*, wenn externe Partner an einer internen Transaktion eines BPEL-Prozesses teilnehmen. Für *scopes* gelten die transaktionalen Eigenschaften der Atomarität, Isolation und Konsistenz. Ein *scope* kann mit Aktionen erweitert werden, die im Fehlerfall ausgeführt werden. Ein *scope* mit einem *compensation handler*, kann z. B. mit einer kompensierbaren Sphäre verglichen werden. Isolation wird erreicht, indem jedem *scope* lokale Daten zugewiesen werden. Der Datenaustausch mit externen Datenquellen wird hier aber nicht betrachtet. Integritätsbedingungen können durch *fault handler* umgesetzt werden, die an einen *scope* gebunden sind. So können die von Services erzeugten Daten auf Konsistenzen hin überprüft werden.

6.7 Zusammenfassung

In diesem Kapitel wurde eine transaktionale Erweiterung des *YAWL*-Modells vorgestellt, die eine konsistente Integration externer Datenquellen erlaubt. Externe Datenquellen können auf unterschiedliche Weise in einen Prozess integriert werden. Aktivitäten können die Daten mit Hilfe von Serviceaufrufen oder Programmaufrufen im Prozess verwenden. Eine andere Möglichkeit bietet das Data Access Framework, welches in Kapitel 5 vorgestellt wurde. In beiden Fällen können Inkonsistenzen bei der Datenhaltung auftreten, weil die Zusicherung von transaktionalen Eigenschaften wie Atomarität und Isolation nicht gewährleistet werden.

Um inkonsistente Datenhaltung im WFMS zu vermeiden, haben wir deshalb die Kontrollflussperspektive von *YAWL* mit transaktionalen Konzepten erweitert. Das *tx+YAWL*-Modell erweitert dafür das Konzept der externen Variablen, welches in Kapitel 5 vorgestellt wurde. *tx+YAWL* bietet die Möglichkeit, Bereiche in einem Prozessmodell zu definieren, für die transaktionale Eigenschaften hinsichtlich der integrierten externen Variablen sichergestellt werden. Dafür wurde ein 4-Ebenen-Transaktionsmodell definiert, welches Atomarität, Isolation, Serialisierbarkeit und Rücksetzbarkeit für den *YAWL*-Prozess umsetzt. Um transaktionale Eigenschaften im Prozessmodell beschreiben zu können, wurde es mit transaktionalen Sphären erweitert (Anforderung 3.5). Eine transaktionale Sphäre enthält Aktivitäten, deren Lese-

und Schreiboperationen die atomaren Schritte einer Transaktion darstellen. Innerhalb einer Sphäre wird eine isolierte Sicht auf externe Variablen sichergestellt. Inkonsistenzen, die aufgrund redundanter Datenhaltung auftreten, können so verhindert werden (Anforderung 6.1). Externe Variablen werden gegenüber Änderungen in externen Datenquellen oder durch parallel laufende Prozesse isoliert. Außerdem können Integritätsbedingungen auf externen Variablen definiert werden, die innerhalb einer Sphäre gelten müssen (Anforderung 6.7). Durch die Sphäre wird außerdem kontrolliert, wann Änderungen an externen Variablen in die Datenquellen propagiert werden.

Das klassische Transaktionskonzept kann nicht auf Prozesse angewendet werden. Die strikte Isolation einzelner Transaktionen führt bei langlaufenden Transaktionen dazu, dass Zwischenergebnisse erst sehr spät anderen Anwendungen zur Verfügung stehen. Deshalb wurden Sphären im *tx+YAWL*-Modell im Sinne eines offen geschachtelten Transaktionsmodells umgesetzt (Anforderungen 6.4 und 6.5). Die Isolation wird aufgeweicht, indem Zwischenergebnisse von Teiltransaktionen frühzeitig anderen Prozessen und Anwendungen bereitgestellt werden. Dafür werden die Daten am Ende jeder (Sub-)Sphäre in die externe Datenquelle geschrieben. Weil die Daten nicht direkt mit der externen Datenquelle synchronisiert werden können, wurde ein lokaler Arbeitsbereich eingeführt. Um die Isolation paralleler Transaktionen bzw. Sphären zu ermöglichen, wurde das Konzept des Mehrversionen-Concurrency-Control umgesetzt (Anforderung 6.3).

Um Atomarität für Transaktionen zu erreichen, wurden im *tx+YAWL*-Modell verschiedene Transaktionstypen eingeführt. Kompensierende Transaktionen ermöglichen es, Effekte nicht rücksetzbarer Transaktionen mit einer Menge von Aktionen zu kompensieren (Anforderung 6.6). Kann eine Transaktion nicht korrekt abgearbeitet werden, kann zwischen mehreren Alternativen Transaktionen ausgewählt werden (Kontingent-Transaktionen). Fehler bei der Ausführung nicht-vitaler Transaktionen führen nicht zum Abbruch der Vatertransaktion.

Für die Vermeidung von Transaktionsabbrüchen kann in *tx+YAWL* anhand der Prozessstruktur eine automatische Serialisierung von Aktivitäten vorgenommen werden. Ein geeignetes Recovery wird mit Hilfe von Exception-Handling durchgeführt. *YAWL* bietet dafür bereits eine Reihe von Techniken an, die in *tx+YAWL* integriert wurden.

Kapitel 7

Flexible, datengetriebene Workflows

In den Kapiteln 5 und 6 wurde ein Konzept für die transaktionale Integration von Datenquellen vorgestellt. Welchen Einfluss diese Daten auf den Kontrollfluss von Prozessmodellen haben, wurde dort jedoch nicht betrachtet. Diese Zusammenhänge werden in diesem Kapitel untersucht. Ziel ist es, ein Konzept zu entwickeln, das die flexible, datengetriebene Anpassung von Prozessinstanzen ermöglicht. Die Prozessmodelle sollen deshalb nur den Basisprozess einer Anwendung beschreiben, der dynamisch an die Dokumentstrukturen und die Dokumentinhalte angepasst werden kann. Dafür müssen dynamische Anteile in den Dokumenten identifiziert und geeignet beschrieben werden. So soll die Wartbarkeit und Erweiterbarkeit der Prozessmodelle erhöht werden.

In Abschnitt 7.1 wird der vorgestellte Ansatz für flexible, datengetriebene Workflows anhand eines Beispiels beschrieben. In diesem Zusammenhang werden die grundlegenden Komponenten und Konzepte des Ansatzes eingeführt. In Abschnitt 7.2 werden anwendungsabhängige und dokumentabhängige Prozessbestandteile identifiziert. Prozessbausteine sollen eine solche Trennung im Prozessmodell ermöglichen. In Abschnitt 7.3 werden Aktivierungsoperationen auf Dokumenten vorgestellt, die die Anpassung der Prozessinstanzen beeinflussen. Es wird ein Konzept für die Umsetzung im Prozessmodell vorgeschlagen. In Abschnitt 7.4 wird ein Konzept für die flexible Komposition von dokumentabhängigen Prozessbereichen beschrieben. In Abschnitt 7.5 wird der vorgestellte Ansatz schließlich mit verwandten Arbeiten verglichen und es erfolgt eine Zusammenfassung des Kapitels in Abschnitt 7.6.

7.1 *FlexY*- Ein Modell für dynamische Publikationsprozesse

Im Publikationsprozess sind viele Operationen direkt von den zu bearbeitenden Dokumenten abhängig. Der Prozess in Abbildung 3.2, Seite 57, enthält z. B. dokumentabhängige Aktivitä-

ten wie *Metadaten Erfassen*. Auch Aktivitäten wie die *Indizierung* oder die *Katalogisierung* sind dokumentabhängig. Eine separate Verarbeitung einzelner Dokumenttypen ist besonders für Anwendungen wichtig, bei denen Multimediadokumente wiederverwendbar sein sollen. Ist ein Bild im Dokument enthalten, sollten Metadaten genauso wie Indexeinträge unabhängig vom Gesamtdokument angelegt werden. Dementsprechend werden für jeden Dokumenttyp, wie Bild, Text oder Video, eigene Operationen benötigt. Multimediale Dokumente ändern Struktur und Inhalt während des Publikationsprozesses dynamisch. Der Prozess muss so modelliert sein, dass die dynamischen Anteile in den Dokumenten korrekt verarbeitet werden[1]. Ist z. B. ein Bild enthalten, wohingegen Videos noch nicht eingefügt wurden, muss das Prozessmodell trotzdem alle Operationen korrekt ausführen. Das Prozessmodell muss dazu alle zulässigen Dokumentvarianten unterstützen.[2] Das führt häufig zu komplexen Prozessmodellen, da unterschiedliche Dokumentvarianten durch unterschiedliche Prozesspfade verarbeitet werden müssen. Außerdem kommt es so häufig auch zu redundant modellierten Prozessfragmenten. Dies erschwert die Wartung der Modelle zusätzlich. In Abbildung 3.6b, Seite 65, müssen die Aktivitäten *keyword extraction* und *stemming* mehrfach im Modell eingefügt werden. So können die Varianten mit Text aber ohne Video bzw. umgekehrt realisiert werden. Außerdem ist so eine parallele Abarbeitung möglich, falls beide Dokumenttypen im Dokument enthalten sind.

Probleme im Publikationsprozess treten unter anderem auf, weil

- multimediale Dokumente häufig Struktur und Inhalt verändern,
- unterschiedliche Dokumentvarianten zu komplexen Prozessmodellen führen und
- häufig Prozessfragmente redundant modelliert werden müssen.

FlexY

Um die Modellierung und Erweiterbarkeit von Prozessmodellen für den Publikationsprozess zu verbessern, können verschiedene Eigenschaften der Prozessmodelle ausgenutzt werden. Dokumentspezifische Bereiche in einem Prozessmodell sollen zur Laufzeit flexibel an die aktuelle Dokumentstruktur und den Inhalt der Dokumente angepasst werden. Dafür müssen die *dokumentspezifischen Prozessbestandteile* geeignet beschrieben werden. Ziel ist es, *Prozessbausteine* zu definieren, die wiederverwendbar sind und flexibel in eine Prozessinstanz integriert werden können. Es muss ein geeignetes Konzept definiert werden, welches die richtigen Prozessbausteine in Abhängigkeit der bearbeiteten Dokumente zum richtigen Zeitpunkt im Prozessmodell integriert. Weil die Integration der Prozessbausteine vom Dokumentzustand abhängig ist, ist es wichtig, den Dokumentzustand geeignet zu beschreiben. Weiterhin muss eine Beziehung zwischen dem Dokumentzustand und den Prozessbausteinen modellierbar sein.

[1] In Anhang B.2 und B.3 werden jeweils zwei Beispieldokumente gezeigt.
[2] Eine Dokumentvariante ist durch eine Kombination von erlaubten Dokumenttypen gegeben.

Im Folgenden wird ein flexibles, datengetriebenes Prozessmodell vorgestellt, das dynamische Publikationsprozesse in digitalen Bibliotheksanwendungen unterstützt.

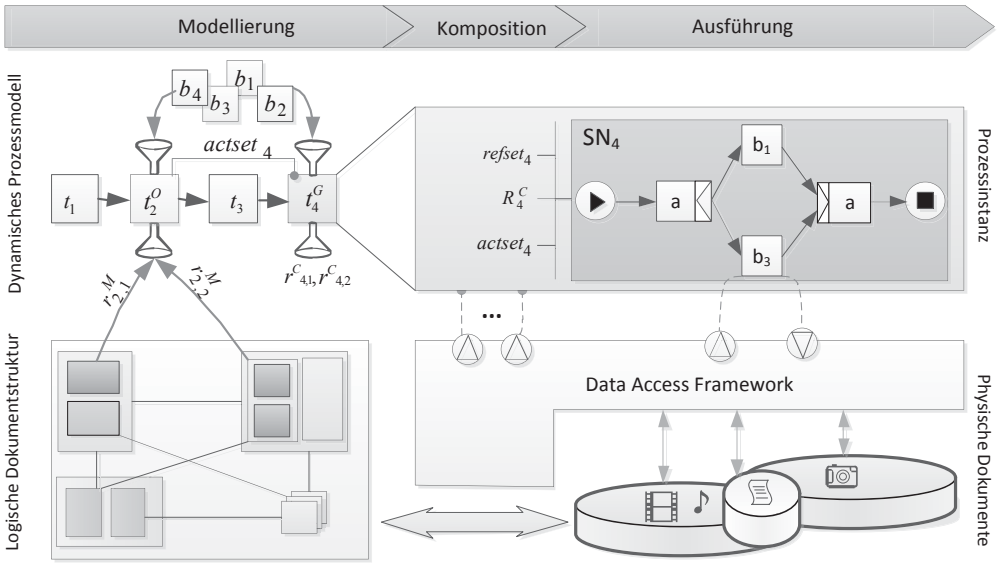


Abbildung 7.1: FlexY: Ein Modell für flexible Prozesse

Für die Umsetzung nutzen wir das *YAWL* Workflow-Management-System. *YAWL* unterstützt das Konzept der Unterspezifikation, kann aber nur bereits modellierte Prozessmodelle, nach dem *late-binding*-Prinzip, integrieren (siehe Ansatz *Worklets* in [AHEA06b]). Entsprechend müssen die von *YAWL* gebotenen Funktionalitäten hinsichtlich der Unterstützung von flexiblen Prozessen erweitert werden (siehe Abschnitt 3.4). Wir haben hierfür die *YAWL*-Erweiterung *FlexY* entwickelt. Sie steht für ***Flex**ible **Y**AWL*. Die Erweiterung bietet die Möglichkeit, Prozessmodelle flexibel zur Laufzeit, in Abhängigkeit von Dokumentzuständen anzupassen. An festgelegten Punkten in einem Prozessmodell können Prozessfragmente nach dem Prinzip des *late-modeling* generiert und ausgeführt werden. Abbildung 7.1 zeigt die Architektur von *FlexY*.

Basisprozess. Der Basisprozess einer digitalen Bibliotheksanwendung bildet den Geschäftsprozess der Anwendung ab. Die Schritte im Geschäftsprozess werden auf Aktivitäten und Netze abgebildet. Die Reihenfolge der Abarbeitung wird in der Kontrollflussperspektive von *YAWL* beschrieben. Für die Modellierung können alle *YAWL*-Modellierungskonzepte verwendet werden. In Abbildung 7.1 ist der Geschäftsprozess durch die Aktivitäten t_1 bis t_4 beschrieben, die nacheinander abzuarbeiten sind.

Dokumentabhängige, wiederverwendbare Prozessfragmente. Die dokumentabhängigen Prozessfragmente werden durch Prozessbausteine abgebildet. Im *FlexY*-Modell repräsentieren *Bricklets* diese dokumentabhängigen Prozessfragmente. Bricklets entkoppeln die dokumentspezifischen Prozessfragmente vom Basisprozess. Sie sind nicht im eigentlichen

Sinne Bestandteil des Prozessmodells des Basisprozesses. Ein Bricklet ist durch eine Menge von Aktivitäten beschrieben, die eine konkrete, dokumentspezifische Aufgabe wie das Indizieren von Text oder das Indizieren von Bildern modellieren. In Abbildung 7.1 sind vier Bricklets b_1 bis b_4 modelliert.

Flexible Prozessbereiche. Im Basisprozess sind neben Aktivitäten, die den Geschäftsprozess beschreiben, auch Platzhalter definiert, welche dokumentspezifische Aufgaben umsetzen. Die Platzhalter werden als *Generatoren* bezeichnet. Eine dokumentspezifische Aufgabe im Publikationsprozess ist z. B. das Indizieren von unterschiedlichen Dokumenttypen wie Bilder, Texte oder Videos. Generatoren ermöglichen es, Bricklets zur Laufzeit zu Subnetzen zusammen zu bauen und auszuführen. Gleichzeitig schränken sie die Bereiche im Prozessmodell ein, in denen Änderungen möglich sind. In Abbildung 7.1 ist die Aktivität t_4^G als Generator-Aktivität definiert (der Index G steht für Generator). Der Generator t_4^G erzeugt das Subnetz SN_4, das nach Aktivierung der Generator-Aktivität direkt generiert und ausgeführt wird.

Dokumentzustand. Die Generierung von Subnetzen in einem Generator ist notwendig, weil im Vorfeld nicht festgelegt werden kann, welchen Zustand die zu bearbeitenden Dokumente haben. Aus diesem Grund kann erst zur Laufzeit ermittelt werden, welche Bricklets verwendet werden müssen. Enthält ein Dokument nur Text und Bilder, werden nur Bricklets für die Verarbeitung dieser beiden Dokumenttypen benötigt. Der Dokumentzustand wird durch spezielle Aktivitäten, den sogenannten *Observern*, zur Laufzeit ermittelt. Hierfür werden im Basisprozess Oberserver-Aktivitäten an vorher festgelegten Punkten eingefügt. In Abbildung 7.1 ist die Aktivität t_2^O als Observer-Aktivität definiert (der Index O steht für Observer). Ein Observer kann für beliebig viele Generatoren die Menge der aktiven Bricklets verändern.

Regeln. Für die Auswahl von Bricklets durch einen Observer werden Regeln (Matching-Regeln) definiert. Eine *Matching-Regel* beschreibt die Beziehung zwischen einem Dokumentfragment, beliebig vielen Bricklets und einem Generator. Der Observer wertet dann die ihm zugeordnete Menge an Matching-Regeln zur Laufzeit aus. In Abbildung 7.1 sind dem Observer t_2^O die zwei Regeln $r_{2,1}^M$ und $r_{2,2}^M$ zugeordnet.

Eine zweite Regelmenge beschreibt, wie ein Observer die Menge der aktivierten Bricklets zusammen bauen muss. Die *Konstruktionsregeln* geben den Kontrollfluss im Subnetz vor und die Anzahl, wie oft ein Bricklet in das Subnetz eingeht. In Abbildung 7.1 sind dem Generator t_4^G die zwei Regeln $r_{4,1}^C$ und $r_{4,2}^C$ zugeordnet.

7.2 Basisprozess und Prozessbausteine

Ein Prozessmodell zur Modellierung dokumentzentrierter Publikationsprozesse muss flexibel an den Zustand von Dokumenten angepasst werden. Deshalb wird in diesem Abschnitt eine Unterteilung in anwendungs- und dokumentabhängige Prozessfragmente vorgenommen.

7.2.1 Basisprozess

Anwendungs- und dokumentabhängige Prozessfragmente
Prozessmodelle müssen an die gestiegenen Anforderungen digitaler Bibliotheksanwendungen angepasst werden. Insbesondere der Kontrollfluss moderner Bibliotheksanwendungen wird immer komplexer. Publikationsprozesse müssen immer mehr Aufgaben aus dem Bereich des Content-Management (siehe Abschnitt 2.1.3) umsetzen und werden dadurch immer aufwendiger und umfassender. Die Komplexität der Dokumentmodelle steigt zusätzlich mit der Anzahl unterschiedlicher multimedialer Dokumenttypen. Eine Anforderung an den Publikationsprozess ist deshalb die Reduzierung der Komplexität von Prozessmodellen (siehe Anforderung 3.7), wodurch die Wartbarkeit der Prozessmodelle verbessert werden soll.

Die Arbeitsabläufe in digitalen Bibliothekssystemen sind sehr stark dokumentzentriert. Deshalb lassen sich die Aktivitäten in zwei unterschiedliche Kategorien unterteilen. Die eine Menge der Aktivitäten ist den anwendungsspezifischen Prozessfragmenten zuzuordnen. Die andere Menge der Aktivitäten kann den dokumentspezifischen Prozessfragmenten zugeordnet werden. Diese Eigenschaft lässt sich ausnutzen, um eine Trennung der Prozessfragmente im Prozessmodell vorzunehmen.

Anwendungsspezifische Prozessfragmente. Der Geschäftsprozess einer digitalen Bibliotheksanwendung wird durch anwendungsspezifische Prozessfragmente beschrieben. Der Geschäftsprozess beschreibt die Prozessfragmente, welche für alle Prozessvarianten gleich sind. Änderungen am Prozessmodell gehen häufig mit Änderungen der Anwendung einher. Eine dynamische Anpassung des Geschäftsprozesses ist deshalb für diesen Anwendungsfall nicht notwendig. Die anwendungsspezifischen Prozessfragmente werden für die Modellierung eines Basisprozesses verwendet.

Dokumentabhängige Prozessbestandteile. Die kontextabhängige Aktivitäten einer digitalen Bibliotheksanwendung werden durch dokumentabhängige Prozessbestandteile beschrieben. Der Kontext wird durch den Inhalt und den Zustand der Dokumente beschrieben. Eine kontextabhängige Aktivität zeichnet sich dadurch aus, dass sie eine bestimmte, für einen Dokumenttyp spezifische, Operation beschreibt. Dies kann beispielsweise die Angabe von Metadaten für einen konkreten Dokumenttyp sein, aber auch die Indizierung von einem Dokumenttyp. Allen kontextabhängigen Aktivitäten gemein ist, dass sie nur in das Pro-

zessmodell eingefügt werden, wenn die Dokumente den entsprechenden Dokumenttyp als Dokumentfragment enthalten.

Lösungsansatz Basisprozess

Die Komplexität von Prozessmodellen kann reduziert werden, indem dokumentabhängige Prozessbestandteile extrahiert werden und nur bei Bedarf in das Prozessmodell eingehen. Der Geschäftsprozess beschreibt dabei alle Prozessbestandteile, die unabhängig vom Kontext für alle Prozessinstanzen gelten. Mit Hilfe geeigneter Techniken kann der Geschäftsprozess an den Kontext angepasst werden, indem dokumentabhängige Prozessbestandteile eingefügt werden. In Abschnitt 4.1.2 wurden dafür bereits unterschiedliche Lösungsstrategien vorgestellt. Demnach eignen sich Ansätze, die die Methode des *late modeling* oder das Prinzip der Modifikationen anwenden, besonders für die Umsetzung von flexiblen Publikationsprozessen.

Für beide Konzepte muss im Vorfeld ein Prozessmodell definiert werden, welches initial ausgeführt wird. Das initiale Prozessmodell wird auch als Basisprozess bezeichnet. Der Basisprozess kann zur Laufzeit an den Anwendungskontext angepasst werden. Hier beeinflussen Zustand und Inhalt der Dokumente die Adaption der Prozessinstanzen.

Für den Publikationsprozess müssen alle anwendungsabhängigen Prozessfragmente immer im Prozessmodell enthalten sein. Sie stellen sicher, dass der Geschäftsprozess korrekt abgearbeitet wird. Der Basisprozess sollte alle anwendungsabhängigen Prozessfragmente enthalten.

Dokumentabhängige Prozessfragmente können grundsätzlich auch in den Basisprozess eingefügt werden. Eine Umsetzung ist dann aber nur mit Prozessmodellen möglich, die *Flexibilität durch Modifikation* unterstützen. Nicht benötigte Prozessfragmente können dann kontextabhängig entfernt, aber auch wieder hinzugefügt werden. Dagegen bietet das Konzept der *Flexibilität durch Unterspezifikation* nicht die Möglichkeit, alle Prozessbereiche einer Prozessinstanz zu modifizieren. Modifikationen sind dabei nur in Platzhaltern möglich, die im Vorfeld festgelegt werden müssen. Entsprechend sollten hierfür keine dokumentabhängigen Prozessfragmente im Basisprozess enthalten sein.

Der Basisprozess für einen Publikationsprozess sollte nur den Geschäftsprozess abbilden. Er unterscheidet sich für die unterschiedlichen Dokumenttypen in seiner Ausprägung. Im Basisprozess müssen z. B. *Richtlinien* umgesetzt werden, die für einen Dokumenttyp während des Publikationsprozesses einzuhalten sind. Außerdem dienen die beschriebenen Arbeitsabläufe der *Qualitätssicherung* im Publikationsprozess. Die *Einhaltung von Ordnungen* muss genauso sichergestellt werden, wie die *Qualität* der Daten (siehe auch Abschnitt 2.1). Aus den genannten Gründen ist eine flexible Anpassung von Basisprozessen nicht erwünscht. Im Gegensatz dazu müssen dokumentabhängige Prozessfragmente flexibel in den Basisprozess eingebunden werden. Weil *YAWL* das Konzept der *Flexibilität durch Unterspezifikation*

unterstützt, wird dieser Ansatz im Folgenden weiter verwendet. Wir stellen deshalb einen Ansatz vor, der das Prinzip des *late modelings* mit deklarativen Techniken kombiniert. Ein Basisprozess beschreibt die anwendungsspezifischen Prozessfragmente und Platzhalter, in denen dokumentabhängige Operationen ausgeführt werden.

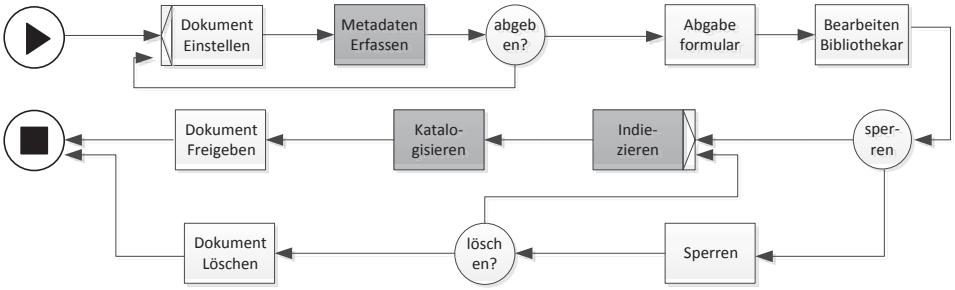


Abbildung 7.2: Basisprozess für Dissertation Online

Beispiel 7.1 (Basisprozess)
Der Publikationsprozess für die Anwendung *Dissertation Online* wird in Abbildung 3.2, Seite 57, gezeigt. Die komplexen Aktivitäten *Metadaten Erfassen*, *Indizieren* und *Katalogisieren* enthalten dokumentabhängige Prozessbereiche. Die restlichen Aktivitäten werden für dieses Beispiel als anwendungsabhängige Aktivitäten betrachtet. Abbildung 7.2 zeigt eine mögliche Umsetzung für den Basisprozess. Ein Beispiel für dokumentabhängige Prozessfragmente ist in Abbildung 3.6b, Seite 65, gegeben. Die Abbildung zeigt den Subprozess für die Aktivität *Indizieren*. Der Subprozess enthält die dokumentabhängigen Prozessfragmente für die Indizierung von Videos, Bildern und Texten.

Wie dokumentabhängige Prozessfragmente definiert und modelliert werden, wird im nächsten Abschnitt beschrieben. Ein Beispiel, wie die Ansätze *tx+YAWL* und *FlexY* zusammen eingesetzt werden, wird in Anhang B.4 vorgestellt.

7.2.2 Prozessbausteine

Um eine dynamische Anpassung dokumentabhängiger Prozessbestandteile zu erreichen, müssen diese geeignet beschrieben werden. Wir führen im Prozessmodell Prozessbausteine (*Bricklets*) ein, welche die dokumentabhängigen Prozessbestandteile beschreiben. Danach werden zwei Varianten diskutiert, wie Bricklets modelliert werden können.

Bricklets
Dokumentabhängige Prozessfragmente werden durch Bricklets beschrieben. Ein Bricklet beschreibt eine Operation, die in den Basisprozess eingefügt werden kann. Es ist ein Baustein, der durch ein korrektes *YAWL*-Netz beschrieben ist. Es muss mindestens einen Start- und einen Endzustand, sowie eine Aktivität enthalten. Das Netz eines Bricklets kann aber auch sehr

komplexe Arbeitsabläufe beschreiben, die wiederum Subnetze enthalten. Bricklets können so in unterschiedlichen Prozessmodellen wiederverwendet werden (siehe Anforderung 3.8).

Für jeden Dokumenttyp, der im Publikationsprozess bearbeitet werden soll, müssen im *FlexY*-Modell Bricklets definiert sein. Ein Bricklet beschreibt dokumentabhängige Operationen für einen Dokumenttyp. So entsteht für jedes Anwendungsgebiet ein Pool von Bricklets, der dokumentspezifische Bricklets bereitstellt. Ist ein Dokumenttyp im Dokument enthalten, müssen alle abhängigen Bricklets in den Basisprozess eingefügt werden.

Modellierung. Bricklets sind kein direkter Bestandteil einer Prozessspezifikation. Deshalb können sie unabhängig von den Prozessmodellen verwaltet werden. *FlexY*-Prozessmodelle enthalten Generatoren, die auch als Platzhalter fungieren (siehe Abschnitt 7.4). Einem Generator werden verschiedene Bricklets zugeordnet. Eine Einschränkung, wie häufig ein Bricklet in einem Prozessmodell eingesetzt werden kann, besteht nicht. Welchen Prozessmodellen und welchen Generatoren ein Bricklet zugeordnet wird, muss während der Modellierung der flexiblen Prozessmodelle vom Anwender entschieden werden. Eine zwingende Einschränkung besteht nur hinsichtlich der benötigten Prozessdaten der Bricklets. Das Prozessmodell bzw. der Platzhalter muss alle Prozessdaten bereitstellen, die für die Abarbeitung der Bricklets benötigt werden. Wir haben *YAWL* dahingehend erweitert, dass die Menge $B = \{b_1, b_2, \ldots, b_n\}$ (mit $i \in \mathbb{N}$) von Bricklets in einem eigenen Verzeichnis innerhalb von *YAWL* abgelegt ist. So ist der Zugriff durch Generatoren jederzeit über eine eindeutige ID möglich.

Die Modellierung von Bricklets unterliegt den gleichen Anforderungen wie die des Basisprozesses. Die modellierten dokumentabhängigen Operationen müssen den Anforderungen und Regeln der digitalen Bibliotheksanwendung entsprechen. Deshalb dürfen Bricklets nicht während der Ausführung von Prozessinstanzen modelliert bzw. erzeugt werden.

Ausführung. Die Ausführung von Bricklets ist an bestimmte Punkte in einem Prozessmodell gebunden. Sie dürfen nur durch einen Generator ausgeführt werden, dem sie im Prozessmodell zugeordnet wurden. Ein Bricklet beschreibt eine dokumentspezifische Aktion, die direkt an einen konkreten Dokumentzustand und/oder -inhalt geknüpft ist. Im Publikationsprozess spielen dabei die Existenz von Dokumentfragmenten und Dokumentstrukturen genauso eine Rolle, wie das Fehlen entsprechender Strukturen und Fragmente. Je nach Anwendungsfall beschreiben Bricklets also Aktionen, die nur dann ausgeführt werden, wenn ein bestimmter Dokumentzustand vorliegt oder nicht vorliegt. Wann und wie ein Bricklet innerhalb einer Prozessbeschreibung verwendet wird, wird durch einen Observer kontrolliert (siehe dazu Abschnitt 7.3).

Definition 7.1 (Bricklet)
Ein Bricklet b_i ist ein valides *YAWL*-Prozessmodell, welches mindestens eine Aktivität enthalten muss. Die Menge $B = \{b_1, b_2, \cdots, b_i\}$, $i \in \mathbb{N}$ ist die Menge aller *Bricklets*. Bricklets werden zu Subnetzen SN_i zusammengesetzt.

Modellierungsvarianten

Bricklets sollen die Entstehung von komplexen Prozessmodellen vermeiden (Anforderung 3.7) und wiederverwendbar sein (Anforderung 3.8). Je nach Anwendungsfall muss deshalb eine geeignete Granularität der Bricklets gefunden werden. Eine mögliche Strategie für die Modellierung von Bricklets besteht darin, atomare Operationen zu beschreiben (Variante 1). Ziel ist es, nicht mehr als die notwendigen Aktivitäten in einem Bricklet zusammen zu fassen. So kann eine hohe Flexibilität bei der Generierung von Subnetzen in Generatoren erreicht werden. Der Aufwand für die Komposition von Subnetzen in den Platzhaltern steigt entsprechend der Anzahl der Bricklets. Im Gegensatz dazu können auch möglichst viele Aktivitäten in einem Bricklet zusammengefasst werden (Variante 2). Dies führt zu geringerer Flexibilität bei der Generierung der Subnetze, verringert aber den Aufwand bei der Generierung.

Am Beispiel des Subprozesses „Indizierung“ (siehe auch Abbildung 3.6b) sollen die beiden Strategien diskutiert werden. Ausgangspunkt ist der Beispielprozess in Abbildung 3.2, Seite 57. Für den Prozess wurde bereits in Abschnitt 7.2.1 ein Basisprozess erstellt (siehe Abbildung 7.2). Hier werden jetzt die dokumentabhängigen Prozessbestandteile extrahiert.

Variante 1 (atomare Operationen):

Die erste Variante beschreibt eine Umsetzung, in der alle Aktivitäten durch ein Bricklet repräsentiert werden.

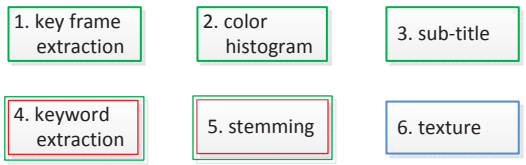


Abbildung 7.3: Bricklets für die Indizierung einer Dissertation nach Variante 1

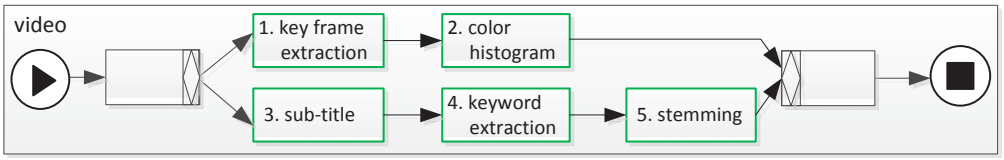
Abbildung 7.3 zeigt eine mögliche Umsetzung mit je einem Bricklet für jede Aktivität. Alle Bricklets, die für die Indizierung eines Videos benötigt werden, sind grün umrandet (Bricklets 1 bis 5). Die Bricklets, die für die Indizierung von Text benötigt werden, sind rot umrandet (Bricklets 4 und 5). Eine blaue Umrandung haben alle Bricklets, die für die Indizierung von Bildern benötigt werden (Bricklet 6).

Die Aktivitäten *keyword-extraction* und *stemming* werden in den Pfaden für *video* und *text* verwendet. In diesem Fall reicht es aus, wenn jeweils nur ein Bricklet erzeugt wird, weil ein Bricklet mehrfach verwendet werden kann. Die mehrfache Verwendung von Bricklets ist hier grafisch durch die doppelte Umrandung gekennzeichnet.

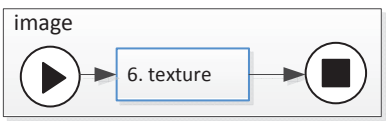
Da jede Aktivität durch ein eigenes Bricklet beschrieben ist, muss der Kontrollfluss bzw. die Abarbeitungsreihenfolge der Bricklets gesondert beschrieben werden. Hierfür verwenden

wir Konstruktionsregeln (siehe Abschnitt 7.4). Die große Anzahl der Bricklets erhöht hier jedoch die Anzahl solcher Regeln drastisch.

Variante 2 (Komplexe Operationen):
Aktivitäten, die inhaltlich zusammen gehören, können in einem Bricklet zusammengefasst werden. Eine mögliche Umsetzung besteht darin, dass die drei Indizierungsschritte, *video*, *text* und *bild*, jeweils durch ein Bricklet dargestellt werden.



(a) Bricklet *video*



(b) Bricklet *image*

(c) Bricklet *text*

Abbildung 7.4: Bricklets für die Indizierung einer Dissertation nach Variante 2

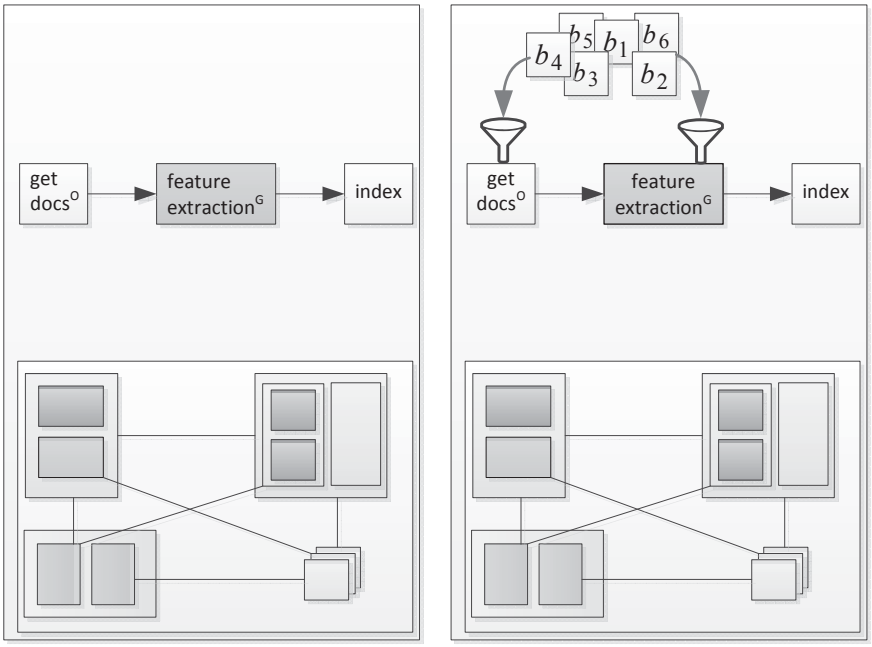
Abbildung 7.4 zeigt drei Bricklets, die eine mögliche Realisierung der jeweiligen Indizierung darstellen. Das Bricklet *video* enthält zwei Prozesspfade und ist in Abbildung 7.4a abgebildet. Der Kontrollfluss und die Benennung der Aktivitäten wird aus dem Subprozess übernommen. Im ersten Pfad wird entsprechend eine Sequenz der Aktivitäten *key frame extraction* und *color histogram* modelliert. Der zweite, parallele Pfad beschreibt eine Sequenz der Aktivitäten *sub-titel*, *keyword-extraction* und *stemming*. Das Bricklet *image* enthält nur die Aktivität *texture* (siehe Abbildung 7.4b). Im Bricklet *text* wird eine Sequenz der Aktivitäten *keyword-extraction* und *stemming* modelliert (siehe Abbildung 7.4b). Der Kontrollfluss und die Benennung der Aktivitäten wird ebenfalls aus dem Subprozess übernommen.

Fazit
Die Menge der zu verwaltenden Bricklets ist in Variante 2 im Vergleich zur Umsetzung in Variante 1 um die Hälfte gesunken. Auch die Anzahl von Aktivierungsregeln und Konstruktionsregeln wird in Variante 2 entsprechend geringer ausfallen. Der Vorteil, Aufgaben bzw. Aktivitäten nur einmal zu modellieren und mehrfach wiederzuverwenden, kommt in Variante 2 nicht zum Tragen. Im Beispiel aus Abbildung 7.4 wird die Aktivität `stemming` in die Bricklets *text* und *video* eingebunden. In diesem Fall führen beide Aktivitäten die gleiche Aufgabe aus. Eine Anpassung der Aktivität, z. B. das Hinzufügen eines weiteren Eingabeparameters, würde in Variante 2 dazu führen, dass beide Bricklets anzupassen sind. Welche der beiden Varianten angewendet wird, hängt maßgeblich vom Anwendungsfall ab. Eine Kombination beider Varianten ist sinnvoll, um die Anzahl der Regeln (Matching-Regeln und Konstruktionsregeln) auf ein sinnvolles und beherrschbares Maß zu beschränken.

Es muss aber Ziel sein, Aktivitäten die mehrfach in einem Prozessmodell verwendet werden, durch Bricklets zu modellieren, um die Wiederverwendbarkeit zu erhöhen und den Änderungsaufwand zu verringern. Der Anwender kann z. B. durch neue Entwurfsmethoden unterstützt werden, die die minimale Überdeckung von Prozesspfaden ermitteln. In [UDGBR11] und [EDGB+12] werden Ansätze für die Bestimmung von *Klonen* vorgestellt (*clone detection*). Klone stellen Prozessfragmente dar, die entweder vollständig (*exact clone*) oder teilweise (*approximate clone*) mit anderen Prozessfragmenten übereinstimmen. Um Klone aus den Prozessen extrahieren zu können, dürfen sie nur einen Eintrittsknoten und einen Austrittsknoten enthalten (*Single-Entry, Single-Exit*). Der Ansatz der *clone detection* kann für den Entwurf von Bricklets genutzt werden, um mehrfach verwendete Prozessfragmente zu identifizieren und für die Umsetzung als Bricklets vorzuschlagen. In dieser Arbeit liegt der Schwerpunkt jedoch nicht auf der Entwicklung solcher Entwurfsmethoden, die deshalb auch nicht weiter betrachtet werden.

7.2.3 Beispiel

Der Basisprozess in Abbildung 7.2 wird in diesem Beispiel mit Bricklets weiter untersetzt. Wir greifen dafür einen Ausschnitt aus dem Subprozess *Indizierung* heraus und wandeln ihn in eine Prozessstruktur um (siehe Abbildung 7.5). Das Ergebnis ist in Abbildung 7.5a



(a) Schritt 1: Prozessfragment das die Indizierung von Inhalten beschreibt

(b) Schritt 2: Bricklets für die Merkmalsextraktion definieren

Abbildung 7.5: Beispiel *FlexY*-Modell, Basisprozess mit Bricklets

dargestellt. Im Prozessmodell ist der Observer *get Docs*^{O} dafür verantwortlich den aktuellen Zustand der Dokumente zu erfassen. Im Generator *feature extraction*^{G} werden alle Bricklets, die für die Merkmalsextraktion benötigt werden, zur Laufzeit zu einem Subnetz zusammen gebaut und ausgeführt. Danach kann die eigentliche Indizierung in der Aktivität *index* angestoßen werden.

In Abbildung 7.5b wird dementsprechend die Menge der notwendigen Bricklets eingeführt. Die Bricklets werden durch die Rechtecke mit den Bezeichnungen $b_1, \ldots, b_6$ dargestellt. Der Index eines Bricklets entspricht der Nummerierung der Bricklets aus Abbildung 7.3. Das Bricklet b_1 entspricht also dem Bricklet *key frame extraction*. Entsprechendes gilt für die anderen Bricklets.

7.3 Überwachung von Dokumentänderungen im Basisprozess

7.3.1 Kontext: Änderungen an Dokumenten überwachen

In Abschnitt 7.2 wurde eine Trennung von anwendungsspezifischen und dokumentspezifischen Dokumentbestandteilen durch die Einführung von Basisprozessen und Bricklets erreicht. Im nächsten Schritt muss definiert werden, wann und wie Bricklets in den Basisprozess eingefügt werden sollen. Um einen Basisprozess automatisch und kontextabhängig zu verändern, muss definiert werden:

- wann der Basisprozess geändert wird,
- wo der Basisprozess geändert wird und
- wie der Basisprozess geändert wird.

Wann der Basisprozess geändert wird, hängt direkt vom Zustand der Dokumente ab. Je nach Dokumentzustand, muss der Basisprozess in unterschiedlichen Bereichen verändert werden. Aktivierungsoperationen im Basisprozess beschreiben, wann Bricklets *eingefügt* oder *gelöscht* werden.

Observer
Der Basisprozess muss in Abhängigkeit des Zustands und der Zustandsänderungen einzelner Dokumente geändert werden. Der Dokumentzustand ist durch den Inhalt und die Struktur eines Dokumentes gegeben. Bricklets sind im Publikationsprozess von verschiedenen Zuständen und Zustandsänderungen der Dokumente abhängig. Dazu zählen:

- Die Existenz von bestimmten Typen von Dokumentfragmenten. Sind z. B. im Dokument Bilder enthalten, müssen alle Operationen für Bilder in den Basisprozess eingefügt werden.

- Die Veränderung des Zustands des Dokumentes. Wenn z. B. alle Bilder aus dem Dokument gelöscht werden, müssen alle Bricklets für Bilder aus dem Basisprozess entfernt werden. In so einem Fall kann aber auch das Hinzufügen von kompensierenden Bricklets erforderlich sein.
- Der Zustand und der Inhalt von Dokumenten zu einem bestimmten Zeitpunkt. Ein Dokument muss z. B. vor der *Freigabe* eine *Einverständniserklärung* enthalten, die von einem Mitarbeiter kontrolliert werden muss.

Änderungen an Dokumenten können sehr häufig vorkommen. *Observer* bieten die Möglichkeit, an beliebigen Stellen in einem Prozessmodell den Zustand von einem oder mehreren Dokumenten zu ermitteln. In Abhängigkeit von diesem Dokumentzustand können Bricklets für unterschiedliche Prozessbereiche aktiviert oder deaktiviert werden.

Definition 7.2 (Observer-Aktivität)
T^O ist die Menge aller *Observer-Aktivitäten* $t_i^O = (R_i^M, \text{match})$ mit $i \in \mathbb{N}$. Dann ist:

- $R_i^M = \{r_{i,1}^M, r_{i,2}^M, \cdots, r_{i,j}^M\}$ mit $j \in \mathbb{N}$ die Menge der Matching-Regeln (matching rules)
- $r_{i,j}^M : \text{match}(pexpr_j) \mapsto (op, t_m^G, B_j)$ eine Matching-Regel
- $pexpr_j$ ein XPath-Ausdruck, der Dokumentfragmente adressiert, welche die Auswahl der Bricklets beeinflussen können
- $op = \{add, delete, merge, undo\}$ die Menge aller Aktivierungsoperationen
- $t_m^G \in T^G$ eine Generator-Aktivität
- $B_j \subseteq refset_m$ die Menge der zu aktivierenden Bricklets

Ein Observer wird immer nach Aktivitäten, die wesentliche Änderungen an Dokumenten hervorrufen, in ein Prozessmodell eingefügt. Zustandsänderungen haben, in Abhängigkeit vom Zustand der Prozessinstanz, unterschiedliche Auswirkungen auf die Prozessinstanz. Weil sich der Zustand eines Dokumentes während der Abarbeitung häufig ändern kann, wird so auch die Anzahl der Aktivierungsoperationen auf der Prozessinstanz verringert.

7.3.2 Matching-Regeln

Um die Verwendung von Bricklets zu beschreiben und zu steuern, muss eine Beziehung zwischen Dokumenten und den davon abhängigen Bricklets definiert werden. Diese Beziehung wird durch eine Matching-Regel beschrieben. Eine Matching-Regel $r_{i,j}^M$ enthält einen XPath 1.0 konformen Pfadausdruck $pexpr_j$, der genau ein Dokumentfragment adressiert. Der Pfadausdruck ermöglicht es, neben ganzen Dokumenten auch beliebige Dokumentfragmente zu adressieren. Die folgenden Beispiele für Pfadausdrücke beziehen sich auf das in Anhang B.2 vorgestellte Dokument im *DiML*-Format.

Für die Auswertung des Pfadausdruckes wird die Funktion $match(pexpr_j)$ verwendet. Die Funktion $match(pexpr_j)$ wertet den Pfadausdruck $pexpr_j$ aus und liefert eine Knotenmenge zurück. Soll auf das Nichtvorhandensein eines Dokumentfragments hin überprüft werden, muss der Pfadausdruck entsprechend angepasst werden. In diesem Fall kann z. B. eine Kombination aus den beiden XPath-Funktionen[3] *not* und *exists* verwendet werden. Im Beispiel liefert der Pfadausdruck `exists(//mm[@format='AVI']` genau dann *true* zurück, wenn die zurückgegebene Knotenmenge nicht leer ist. Das Dokument enthält in diesem Fall mindestens ein Element, das eine Videodatei einbindet. Soll eine kompensierende Aktion ausgeführt werden, kann mit der Funktion *not*() getestet werden, ob die Ergebnisknotenmenge leer ist. Ist kein Bild (hier mit dem Format *JPEG* oder *GIF*) im Dokument enthalten, liefert `not(//mm[@format='JPEG' or @format='GIF']`[4] den Wert *true* zurück. Eine Vorgabe kann hier nicht gemacht werden, da dies vom Anwendungsfall abhängt.

Die Auswertung durch die *match*-Funktion liefert die Information darüber, ob ein Dokumentfragment enthalten ist oder nicht. Die Funktion *match* nutzt für die Auswertung eines Pfadausdrucks intern das *Data Access Framework* (siehe Kapitel 5). Jedem Pfadausdruck wird genau eine externe Variable zugeordnet. Über die Definition der externen Variable wird gesteuert, wie der Zugriff auf die Datenquelle erfolgt. Die unterschiedlichen Zugriffsmethoden werden in Abschnitt 6.3 betrachtet.

Die Bricklets, die einem Dokumentfragment zugeordnet werden, sind in der Menge B_j beschrieben. Eine Matching-Regel wird von einem Observer verwendet, um das Aktivierungsset $actset_m$ für genau eine Generator-Aktivität t_m^G zu verändern. Das Aktivierungsset $actset_m$ enthält ein oder mehrere Bricklets, welche die Generator-Aktivität für die Konstruktion eines Subnetzes verwendet (siehe Abschnitt 7.4). Jedem Pfadausdruck $pexpr_j$ ist hierfür genau eine Operation *op* zugeordnet, welche das Aktivierungsset bearbeitet. Aus diesem Grund muss für die Menge $B_j \subseteq refset_m$ gelten. Die Menge $refset_m$ beschreibt hier die Menge aller Bricklets, die ein Generator t_m^G verwenden darf. Um den Status von Bricklets unterschiedlicher Generatoren zu verändern, können mehrere Matching-Regeln für einen Observer angelegt werden. Demnach ist eine *Observer-Aktivität* durch eine Menge von *Matching-Regeln* R_i^M und durch die Funktion *match* definiert.

7.3.3 Aktivierungsoperationen auf Konstruktionsregelmenge

Der Dokumentzustand hat während der Ausführung einer Prozessinstanz, unterschiedliche Auswirkungen auf den Publikationsprozess. Zu Beginn einer Prozessinstanz führt die Existenz eines Bildes z. B. dazu, dass für die Indizierung ein entsprechendes Bricklet aktiviert

[3] `http://www.w3schools.com/Xpath/xpath_functions.asp`

[4] Der Pfadausdruck wird durch Anwendung der Funktion *boolean*(*arg*) auf eine booleschen Wert reduziert. Die Funktion boolean(arg) liefert für eine leere Knotenmenge *false* und für eine nicht leere Knotenmenge *true*.

werden muss. Wird zu einem späteren Zeitpunkt das Bild wieder gelöscht, müssen die Bricklets für die Indizierung entfernt werden. Außerdem müssen gegebenenfalls die vorhandenen Indizes bereinigt werden, was die Aktivierung eines weiteren Bricklets hervorruft.

Über die Funktion $match(pexpr_j)$ kann die Existenz eines Dokumentfragments bestimmt werden. Über die Aktivierungsoperationen *op* kann festgelegt werden, wie das Konstruktionsset einer Generator-Aktivität manipuliert werden soll. Die Operation *op* wird aber nur ausgeführt, wenn $match(pexpr_j) == true$ gilt.

Definition 7.3 (Aktivierungsregeln)
Die Menge $op = \{add,\ delete,\ merge,\ undo\}$ beschreibt die möglichen Operationen, um das Konstruktionsset $actset_m$ einer Generator-Aktivität t_m^G zu manipulieren.

Die ***add*-Funktion** fügt die Menge B_j dem Konstruktionsset $actset_m$ der Generator-Aktivität t_m^G hinzu.

$$add(B_j,\ pexpr_j) \mapsto actlist_m \cup add(B_j,\ pexpr_j) \tag{7.1}$$

Für die *add*-Funktion gilt, dass Bricklets auch mehrfach in ein Konstruktionsset eingefügt werden können. In diesem Fall wird aber nicht die eigentliche Anzahl der Bricklets erhöht, weil das Konstruktionsset im Sinne einer Mathematischen Menge keine mehrfachen Elemente zulässt. Ist ein Bricklet bereits aktiviert, wird der Pfadausdruck $pexpr_j$ der Menge von Pfadausdrücken $f(b_k)$ des Bricklets zugeordnet (siehe Abschnitt 7.4). Soll eine mehrfache Ausführung von Bricklets verhindert werden, so muss die *merge*-Funktion verwendet werden.

Die ***merge*-Operation** unterscheidet sich von der *add*-Funktion dahingehend, dass sie nur angewendet wird, wenn $b_k \notin actset_m$ gilt. Demnach werden dem Bricklet keine weiteren Pfadausdrücke zugeordnet.

$$merge(B_j,\ pexpr_j) \mapsto actlist_m \cup add(B_j,\ pexpr_j) \tag{7.2}$$

Die ***delete*-Operation** entfernt den Pfadausdruck $pexpr_j$ für die Menge B_j von Bricklets, welche im Konstruktionsset $actset_m$ enthalten sind.

$$delete(B_j,\ pexpr_j) \mapsto actlist_m - delete(B_j,\ pexpr_j) \tag{7.3}$$

Die ***undo*-Operation** ermöglicht das Einfügen von kompensierenden Bricklets.

$$undo(B_j^{-1},\ pexpr_j) \mapsto actlist_m \cup add(B_j^{-1},\ pexpr_j) \tag{7.4}$$

Ein kompensierendes Bricklet kann in das Konstruktionsset eingefügt werden, nachdem ein Dokumentfragment aus einem Dokument entfernt wurde und das einfache Löschen von Bricklets nicht ausreicht. Es führt roll-back-Operationen für die Bricklets B_j durch.

7.3.4 Beispiel

Der Typ und die Parameter einer Regel sind von verschiedenen Faktoren abhängig, die bei der Modellierung durch den Anwender beachtet werden müssen:

- Der Typ einer Regel wird in Abhängigkeit vom Observer und dessen Position im Prozessmodell ausgewählt. Aus dem Prozessverlauf ergibt sich, ob Dokumentfragmente hinzugefügt oder entfernt werden konnten. Entsprechend müssen Bricklets hinzugefügt oder entfernt werden. Wurden bereits Operationen ausgeführt, die kompensiert werden müssen, ist das Hinzufügen kompensierender Bricklets sinnvoll.
- Die Anzahl der Bricklets, die an eine Regel übergeben werden, richtet sich hauptsächlich nach zwei Auswahlkriterien: Die Anzahl unterschiedlicher Dokumenttypen, die in Dokumenten erlaubt sind bzw. nach dem Verwendungszweck des ausgewählten Generators.

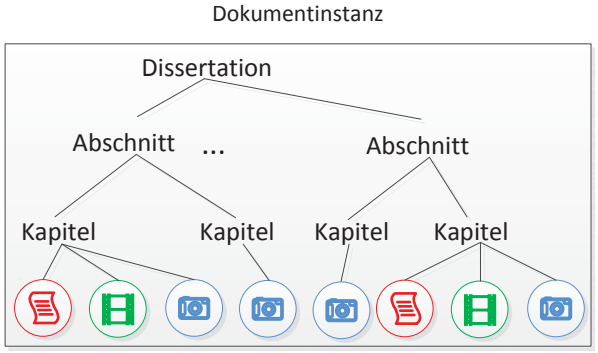


Abbildung 7.6: Dokument mit mehreren gleichen Fragmenttypen

Im Folgenden wird am Beispiel gezeigt, wie Regeln aufgestellt werden. Der Observer *get Docs*^{O} in Abbildung 7.7 wird im Prozessmodell für die Konfiguration der nachfolgenden Generator-Aktivität *feature exraction*^{G} genutzt. Die Regeln sollen daher Bricklets für die Indizierung unterschiedlicher Dokumenttypen aktivieren, weshalb *add*-Regeln verwendet werden.

Um die richtigen Bricklets auszuwählen, müssen die Dokumentmodelle der zu verarbeitenden Dokumente analysiert werden. In Abbildung 3.7, Seite 67, werden unterschiedliche Dokumentausprägungen gezeigt, die die Angabe der Regeln beeinflussen. In Abbildung 3.7a hat das Dokument nur ein Textelement, entsprechend werden hier nur Bricklets für die Indizierung von Text-Elementen benötigt. Das Dokument in Abbildung 3.7b enthält nur ein Bild. Analog werden hier nur Bricklets für die Indizierung von Bildern benötigt. Ein Bild und ein Video sind im Dokument aus Abbildung 3.7c enthalten. Hier werden dann alle oben genannten Bricklets benötigt. Im Beispiel aus Abbildung 7.6 müssen mehrere Bilder verarbeitet werden. Der Nutzer muss jedes Bild einzeln mit Metadaten anreichern. Eine Indizierung der Bilder muss ebenfalls einzeln stattfinden. Für die Bearbeitung des Textes

müssen hingegen je nach Typ Metadaten vergeben werden, wohingegen für die Indizierung des Textes eine Ausführung aller benötigten Aktivitäten ausreicht.

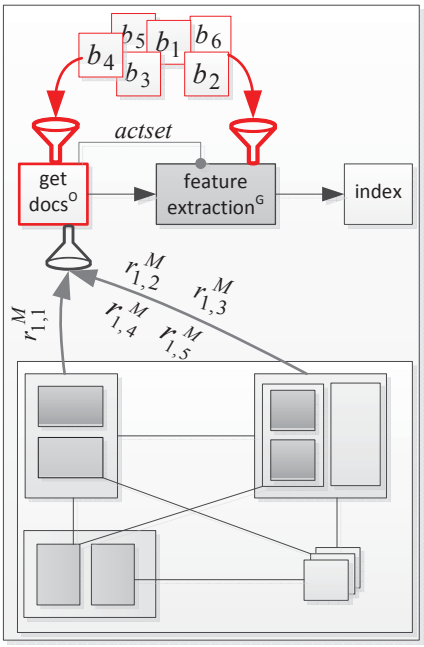


Abbildung 7.7: Beispiel *FlexY*-Prozessfragment, Schritt 3: Auswahl von Bricklets für die Merkmalsextraktion

Abbildung 7.7 stellt mehrere Matching-Regeln dar, die für den Observer *get docs*^{O} definiert sind.

Regel 1 (Bild hinzufügen): Im Prozess werden Dokumente verarbeitet, die Bilder vom Typ *JPEG* und *GIF* enthalten können. Deshalb wird in Regel 1 das Bricklet b_6 aktiviert, das für die Verarbeitung beider Dokumenttypen genutzt wird. Das Bricklet wird in diesem Fall für den Generator mit der ID *feat.extr.* aktiviert (*feat.extr.* entspricht dem Generator *feature extraction*^{G}).
$r^M_{1,1}$=`add(feat.extr.,`b_6`,"exists(//mm[@format='JPEG' or @format='GIF' ])")`

Regel 2 (Video hinzufügen): Dokumente können Videos mit den Typen *AVI* oder *WMV* enthalten. Deshalb werden in Regel 2 die Bricklets b_1, b_2, b_3, b_4, b_5 aktiviert, die die Indizierung von Videos beschreiben. Die Bricklets werden ebenfalls im Generator mit der ID *feat.extr.* aktiviert.
$r^M_{1,2}$=`add(feat.extr.,`$\{b_1,b_2,b_3,b_4,b_5\}$`,"exists(//mm[@format='AVI' or @format='WMV'])")`

Regel 3 (Text hinzufügen): Damit auch Dokumente verarbeitet werden können, die Texte enthalten, werden in Regel 3 die Bricklets b_4, b_5 aktiviert. Die Bricklets werden ebenfalls im

Generator mit der ID *feat.extr.* aktiviert.
$r^M_{1,3}$=`add(feat.extr.,{`b_4, b_5`},"exists(//p[not(child::mm)])")`

weitere Regeln:
In einem Observer können beliebig viele Matching-Regeln definiert sein, die unterschiedliche Generatoren bedienen. Weitere Regeln im Observer *get docs*^{O} sind:

Regel 4 (Bricklets entfernen): Enthält ein Dokument z. B. nur Bilder mit dem Typ *TIFF* sollten die Bricklets b_7, b_8 aus dem Generator mit der ID *katalog.* gelöscht werden. Diese können z. B. durch einen anderen Observer im Vorfeld aus einem anderen Grund aktiviert worden sein.
$r^M_{1,4}$ =`delete(katalog.,{`b_7, b_8`},"boolean(exists(//p[descendant::mm [@format='TIFF']]) and not(//p[descendant::mm[@format ='JPEG' or @format='GIF']]))")`

Regel 5 (Bricklets kompensieren): Für den Fall dass ein Dokument gelöscht wird, müssen die erzeugten Indexdaten aus dem Index entfernt werden. Dafür wird das Bricklet b_{10} im Generator mit der ID *deleteDoc.* aktiviert.
$r^M_{1,5}$=`undo(deleteDoc.,{`b_{10}`},"exists(//mm[ @format='JPEG' or @format='GIF'])")`

7.4 Dynamische Prozesskomposition

In diesem Abschnitt wird das Konzept für die flexible Komposition von dokumentabhängigen Prozessbereichen mit Hilfe von Bricklets eingeführt.

7.4.1 Konstruktion von Prozessbereichen

Die Entkopplung von anwendungs- und dokumentspezifischen Prozessbestandteilen in einem Prozessmodell wird durch die Einführung von Bricklets (siehe Abschnitt 7.2) erreicht. Matching-Regeln werden von Observern ausgewertet und ermöglichen so die dokumentabhängige Integration von Bricklets in ein Prozessmodell (siehe Abschnitt 7.3). Hier wird das Prozessmodell von *YAWL* so erweitert, dass Bricklets zur Laufzeit flexibel im Prozessmodell integriert und ausgeführt werden können (siehe Anforderung 3.10).

Das *FlexY*-Prozessmodell setzt Flexibilität nach dem Prinzip des *late-modeling* um. Wir haben das *YAWL*-Prozessmodell daher mit einem neuen Aktivitätstyp erweitert, der als Platzhalter in einem Prozessmodell eingesetzt wird. Ein solcher Platzhalter wird im *FlexY*-Modell als *Generator* bezeichnet. Dem Prinzip des *late-modelings* entsprechend, werden erst zur Laufzeit Prozessfragmente in die Prozessinstanz eingefügt. Im *FlexY*-Modell werden

Bricklets für die Generierung dieser Prozessfragmente verwendet. Generatoren erweitern den Basisprozess um Bereiche, die zur Laufzeit eine automatische Komposition und Ausführung von Subnetzen umsetzen.

Definition 7.4 (Generator-Aktivität)
T^G ist die Menge von *Generatoren* $t_i^G : (R_i^C,\ actset_i,\ refset_i) \mapsto SN_i$ mit $i \in \mathbb{N}$. Dann ist:

- $R_i^C = \{r_{i,1}^C,\ r_{i,2}^C,\ \cdots,\ r_{i,j}^C\}$ mit $j \in \mathbb{N}$ als eine Menge von Konstruktionsregeln definiert.
- $actset_i$ eine Menge von aktivierten Bricklets (b_k) und dazugehörigen Pfadausdrücken $f(b_k) = \{pexpr_{k,1},\ \ldots,\ pexpr_{k,j}\}$ mit $j = 1,\ \ldots,\ n$, die durch verschiedene Observer-Aktivitäten t_m^O ausgewählt wurden.
- $refset_i$ die Menge aller Bricklets t_i^G, die ein Generator verwenden darf.
- SN_i ein valides Subnetz, welches ausgeführt wird, wenn t_i^G im Kontrollfluss verarbeitet wird.

Modellierung. Im Publikationsprozess beschreibt der Basisprozess den Ablauf der anwendungsbezogenen Prozessaktivitäten. Generatoren beschreiben Platzhalter innerhalb eines solchen Prozessmodells, welche die Komposition und Ausführung dokumentspezifischer Prozessfragmente umsetzen. Sie können an beliebigen Punkten im Prozessmodell integriert werden.

Ein Generator erzeugt zur Laufzeit automatisch ein Subnetz (Prozessfragment), welches durch den selben Generator ausgeführt wird. Um eine automatische Komposition zu ermöglichen, muss für jeden Generator die Menge der erlaubten Bricklets festgelegt werden. Nicht jedes Bricklet (und damit nicht jedes dokumentspezifische Prozessfragment) darf an jeder beliebigen Stelle in einem Prozessmodell ausgeführt werden. Die Menge der für einen Generator t_i^G erlaubten Bricklets wird mit $refset_i$ bezeichnet. Zur Modellierungszeit können so einem Generator genau die Bricklets zugeordnet werden, die dort für die Generierung eines Subnetzes benötigt werden. Grundsätzlich kann die Menge, die ein einzelner Generator für die Generierung eines Subnetzes verwendet, beliebig groß sein.

Ausführung. Zur Laufzeit wird für einen Generator t_i^G aus den Bricklets $b_j \in refset_i$ ein Subnetz SN_i generiert. Weil die Ausführung von Bricklets dokumentabhängig ist, dürfen nicht alle erlaubten Bricklets für die Komposition verwendet werden. Die Aktivierung einzelner Bricklets wird durch verschiedene Observer durchgeführt (siehe Abschnitt 7.3). Ein Observer aktiviert oder deaktiviert Bricklets in Abhängigkeit von Dokumentzuständen. Die Menge der Bricklets, welche für einen Generator aktiviert sind, ist durch die Aktivierungsmenge $actset_i$ gegeben. Für die Aktivierungsmenge gilt zusätzlich die Beziehung $actset_i \subseteq refset_i$. Je nach Operation der Matching-Regel, fügt ein Observer je Bricklet auch den aktivierenden Pfadausdruck in die Aktivierungsmenge ein. Werden durch einen Pfadausdruck $pexpr_j$ mehrere Bricklets aktiviert, wird der Pfadausdruck zu jedem Bricklet hinzugefügt. Die

Pfadausdrücke beschreiben die Dokumentfragmente, welche durch das Bricklet zu bearbeiten sind. Nur so ist es möglich, dass zur Laufzeit der genaue Dokumentzustand bearbeitet werden kann. Für die Generierung des Subnetzes SN_i kann der Generator nur Bricklets $b_j \in actset_i$ verwenden.

7.4.2 Konstruktionsregeln

Um eine sinnvolle Konstruktion des Subnetzes SN_i für einen Generator t_i^G sicherzustellen, muss der Kontrollfluss des Subnetzes vorgegeben werden. Hierfür werden Konstruktionsregeln R_i^C verwendet, welche die Generierung der Subnetze kontrollieren.

Eine Konstruktionsregel $r_{i,j}$ definiert eine Beziehung zwischen Bricklets. Für den Publikationsprozess ist es notwendig, verschiedene Beziehungen zwischen Aktionen zu beschreiben. Für einzelne Bricklets spielt die Abarbeitungsreihenfolge eine wichtige Rolle. Gleichartige Elemente eines Dokumentfragments, müssen mehrmals sequentiell von einem Bricklet bearbeitet werden. In manchen Fällen können diese Elemente auch mehrmals parallel durch das selbe Bricklet bearbeitet werden. Im *FlexY*-Modell sind drei Beziehungstypen definiert, um Beziehungen zwischen Bricklets zu modellieren.

Reihenfolge der Bricklets. Für zwei Aktionen bzw. Bricklets ist sicherzustellen, dass ein Bricklet nach dem anderen ausgeführt wird. Hierfür muss während der Komposition des Subnetzes SN_i, der Kontrollfluss so generiert werden, dass eine Reihenfolge in der Abarbeitung eingehalten wird. Diese Beziehung wird durch die Konstruktionsregel $b_k \prec b_l$ beschrieben. Das Bricklet b_l ist irgendwann nach b_k auszuführen.

Sequentielle Ausführung. Im Publikationsprozess ist häufig die mehrfache Ausführung von Aktionen bzw. Bricklets notwendig. Ist im Dokument eine Menge von gleichartigen Dokumentfragmenten enthalten (z. B. mehrere Bilder), so können diese durch ein Bricklet mehrfach hintereinander bearbeitet werden. Diese Beziehung wird durch die Konstruktionsregel $b_k \overset{n}{\prec}$ beschrieben. Das Bricklet b_k wird genau n-mal in das Subnetz SN_i eingefügt. Dabei wird durch den Kontrollfluss sichergestellt, dass die n Bricklets sequentiell, hintereinander, ausgeführt werden.

Parallele Ausführung. Für bestimmte Aktionen bzw. Bricklets die ebenfalls n-mal ausgeführt werden sollen, gilt die Einschränkung der sequentiellen Ausführung nicht. Sie können parallel ausgeführt werden. Diese Beziehung wird durch die Konstruktionsregel $b_k \overset{n}{\|}$ beschrieben. Das Bricklet b_k wird genau n-mal in das Subnetz SN_i eingefügt. Dabei beschreibt der Kontrollfluss die parallele Ausführung der n Bricklets.

Die Regeln für die parallele und sequentielle Ausführung von Bricklets schließen sich gegenseitig aus. Für ein Bricklet b_k darf entweder eine Regel $b_k \overset{n}{\|}$ oder eine Regel $b_k \overset{n}{\prec}$

definiert werden. Eine gleichzeitige Definition beider Regeln in einem Generator ist nicht zugelassen.

Standardregel. Wenn für ein Bricklet $b_k \in actset_i$ keine Regel definiert ist oder keine Regel angewendet werden kann, wird es parallel zu den anderen aktivierten Bricklets in das Subnetz eingefügt. So kann eine möglichst unabhängige und parallele Ausführung im Subnetz SN_i erreicht werden.

Definition 7.5 (Konstruktionsregeln)
R_i^C ist die Menge von Konstruktionsregeln $r_{i,j}^C$. Eine Konstruktionsregel $r_{i,j}^C \in \{b_k \prec b_l,\ b_k \overset{n}{\prec},\ b_k \overset{n}{||}\}$ definiert, wie ein *Bricklet* b_k in das Subnetz SN_i eingefügt wird, wenn $b_k \in actset_i$ gilt.

- $b_k \prec b_l$: Wenn Bricklet $b_l \in actset_i$, ist b_l direkt nach b_k auszuführen.
- $b_k \overset{n}{\prec}$: Das Bricklet b_k ist n mal sequentiell einzufügen.
- $b_k \overset{n}{||}$: Das Bricklet b_k ist n mal parallel einzufügen.

Eigenschaften der Regelmenge
Die Regelmenge für die Konstruktion von Subnetzen muss terminieren und die Anwendung muss unabhängig von der Reihenfolge der Aktivierung sein.

Definition 7.6 (Terminierung Regelverarbeitung)
Ausgehend von der Annahme, dass zu einem bestimmten Zeitpunkt eine endliche Menge von Regeln durch verschiedene Observer-Aktivitäten aktiviert wurde und keine weiteren Regeln durch Observer-Aktivitäten aktiviert werden, soll gelten: die Verarbeitung der aktivierten Regeln **terminiert**, wenn nach endlicher Zeit alle Regeln verarbeitet wurden.

Terminierung. Die Regelmenge terminiert in Bezug auf Definition 7.6, weil keine rekursive Modellierung von Regeln erlaubt ist. Eine Regel kann keine weitere Regel aktivieren und so Zyklen erzeugen. Aus diesem Grund terminiert die Regelmenge immer.

Konfluenz. Der Begriff *Konfluenz* wird im Bereich aktiver Datenbanken verwendet und beschreibt eine Eigenschaft der Regelmenge bezüglich der Ausführungsreihenfolge von Regeln. Werden in einem aktivem Datenbanksystem eine Menge von nicht priorisierten Regeln gleichzeitig aktiviert ist diese Menge konfluent, wenn der resultierende Datenbankzustand nicht von der Abarbeitungsreihenfolge der Regeln abhängt [AWH92]. In Abschnitt 7.4.3 wird gezeigt, wie Konfluenz erreicht wird.

Eigenschaften der generierten Netze
Für die korrekte Ausführung von *YAWL*-Prozessmodellen dürfen die erzeugten Subnetze insbesondere keine Verklemmungen und Zyklen enthalten.

Verklemmungen. entstehen, wenn ein Prozesspfad mit einem OR-Split aufgeteilt wird und mit einem AND-Join wieder synchronisiert wird [TBB08]. Während der Generierung der Subnetze können Verklemmungen vermieden werden, weil die Subnetze in Abhängigkeit der Dokumente erzeugt werden. Deshalb wird keine Auswahl in Form eines *deferred choice* oder *OR-Splits* im Subnetz benötigt. Eine Auswahl zwischen mehreren Bricklets wird durch die Matching-Regeln bereits zur Laufzeit vorgenommen. Im Subnetz werden demnach nur Bricklets eingefügt, die auch ausgeführt werden müssen. Das ist auch der Grund dafür, dass keine Konstruktionsregel für diesen Fall eingeführt wurde. Unabhängig davon sind die Konstrukte *deferred choice* und *OR-Splits* in den Prozessmodellen der Bricklets uneingeschränkt nutzbar.

Zyklen. Für die Generierung der Prozessmodelle dürfen keine OR-Splits verwendet werden. Enthält ein Graph Zyklen, kann daraus kein korrektes *YAWL*-Prozessmodell erzeugt werden. Entweder hat mindestens ein Knoten 2 oder mehr Eingangskanten oder die letzte Aktivität ist mit der ersten Aktivität im Netz verbunden. Deshalb muss sichergestellt werden, dass durch die Anwendung der Konstruktionsregeln kein Zyklus im Prozessmodell erzeugt wird. Die Konstruktionsregeln müssen so angegeben werden, dass kein Zyklus bei der Generierung entstehen kann. Das wird sichergestellt, indem für jede neu eingefügte Konstruktionsregel $r_{i,j}$ ein Test auf Zyklenfreiheit der Regelmenge durchgeführt wird. In Abschnitt 7.4.3 wird gezeigt, wie auf Zyklenfreiheit getestet wird.

Weil Zyklen nicht erlaubt sind, wird keine Regel für eine wiederholte Ausführung von Bricklets definiert. Aktionen, die mehrmals auszuführen sind, müssen die Wiederholung im Prozessmodell des Bricklets umsetzen. Soll eine Wiederholung mehrerer Bricklets ermöglicht werden, kann dies z. B. durch den Kontrollfluss im Basisprozess und einen weiteren Generator gelöst werden.

7.4.3 Komposition von Prozessmodellen

Es wird beschrieben, wie Subprozesse von Generatoren generiert werden. Ein Generator soll automatisch und unabhängig vom Nutzer, Subnetze zur Laufzeit generieren und ausführen. Zur Generierung eines Subnetzes, werden die Aktivierungsmenge $actset_i$ und die Konstruktionsmenge R_i^C verwendet. Sie beschreiben, welche Bricklets eingebunden werden müssen und wie die Bricklets in das Subnetz einzubauen sind. Nachdem ein korrektes Subnetz erzeugt wurde, muss es direkt im Anschluss an die Konstruktion ausgeführt werden. Wir haben die Komposition bzw. Generierung der Subnetze in zwei Teilschritte unterteilt. Im ersten Schritt wird ein gerichteter azyklischer Graph erzeugt, dessen Knotenmenge sich aus den Bricklets zusammensetzt. Im zweiten Schritt wird der Graph in ein *YAWL*-Prozessmodell transformiert, welches ausgeführt wird.

Ein Algorithmus für die Konstruktion von Subprozessen

Als Zwischenschritt wird aus der Menge der Bricklets, die in der Aktivierungsmenge $actset_i$ eines Generators t_i^G enthalten sind, ein gerichteter azyklischer Graph erzeugt. Die Knotenmenge des Graphen ergibt sich aus der Menge der Bricklets, die in der Aktivierungsmenge enthalten sind. Aus der Anwendung der Konstruktionsregeln ergibt sich die Menge der gerichteten Kanten im Graphen. *YAWL* bietet unterschiedliche Möglichkeiten für die Umsetzung der Regeln $b_k \overset{n}{\prec}$ und $b_k \overset{n}{\|}$. Aus diesem Grund wird eine Expansion der Knoten erst während der Transformation des Graphen in das *YAWL*-Modell vorgenommen. Im Graphen werden die betroffenen Knoten entsprechend markiert.

Definition 7.7 (Graph)
Gegeben ist ein gerichteter azyklischer Graph $G_i^C = (V, E)$. V ist die Menge aller Knoten und E ist die Menge aller gerichteten Kanten. Der Graph G_i^C enthält genau einen Startknoten „*start*“ $\in V$ und genau einen Endknoten „*end*“ $\in V$.

Definition 7.8 (Grad Bricklet)
Der Eingangsgrad $deg^-(b_k)$ gibt die Anzahl der eingehenden Kanten für ein Bricklet b_k an. Der Ausgangsgrad $deg^+(b_k)$ gibt die Anzahl von ausgehenden Kanten für ein Bricklet b_k an.

```
1  initialize G with G.V = {start, end} ∪ actset_i and G.E = {}
2  foreach r^C_{i,j} ∈ R^C_i {
3          if r^C_{i,j} equals b_k ≺ b_l and {b_k, b_l} ⊆ actset_i {
4                  add directed edge (b_k, b_l)
5          }
6  }
7  foreach r^C_{i,j} ∈ R^C_i {
8          if r^C_{i,j} equals b_k ≺^n and b_k ∈ actset_i {
9                  n = count(actset_i.f(b_k));
10                 b_k.setRuleType(seq); b_k.setIteration(n);
11         }
12         if r^C_{i,j} equals b_k ||^n and b_k ∈ actset_i {
13                 n = count(actset_i.f(b_k));
14                 b_k.setRuleType(par); b_k.setIteration(n);
15         }
16 }
17 foreach b_k ∈ G.V {
18         if deg^-(b_k) = 0 {
19                 add directed edge (start, b_k)
20         }
21         if deg^+(b_k) = 0 {
22                 add directed edge (b_k, end)
23         }
24 }
```

Auflistung 7.1: Algorithmus zum Generieren eines Graphen, aus der Menge der Bricklets

Für die Konstruktion eines Graphen, verwenden wird den Algorithmus aus Auflistung 7.1, der in fünf Teilschritte unterteilt ist.

Schritt 1 (Initialisieren): Der Graph wird initialisiert, indem der Startknoten *start* und der Endknoten *end* eingefügt werden (siehe 7.1 Zeile 1). Zusätzlich wird für jedes Bricklet $b_k \in actset_i$ ein Knoten im Graph erzeugt ($G.V = \{start, end\} \cup actset_i$). Die Menge der Kanten $G.E = \{\}$ wird mit einer leeren Menge initialisiert. Nachdem der Graph initialisiert wurde, müssen die Konstruktionsregeln R_i^C auf die Knoten im Graphen angewendet werden. Der Algorithmus beschreibt die Reihenfolge für die Anwendung der Regeln entsprechend ihres Typs.

Schritt 2 (Sequenzen): Als erstes werden alle Regeln $R_{i,j}^C$ ausgewertet, die eine Reihenfolge zwischen zwei Bricklets beschreiben (Zeile 2-6). Jede dieser Regeln beschreibt eine Reihenfolge zwischen genau zwei Bricklets. Um die Regel $r_{i,j}^C = b_k \prec b_l$ anwenden zu können, muss für die Bricklets $\{b_k, b_l\} \subseteq actset_i$ gelten. Ansonsten wird die Regel nicht angewendet. Sind beide Bricklets aktiviert, wird eine gerichtete Kante (b_k, b_l) im Graphen hinzugefügt.

In Schritt 3 und 4 werden alle anderen Regeln bearbeitet (Zeile 7-16). Für die weitere Bearbeitung wurde hier die Entscheidung getroffen, keine Expansion der mehrfachen sequentiellen und parallelen Ausführung vorzunehmen. Dieser Schritt wird erst zum Zeitpunkt der Transformation des Graphen in ein *YAWL*-Netz vorgenommen. Der Arbeitsschritt ist in zwei Phasen unterteilt.

Schritt 3 (Sequenzielle Ausführung): In der ersten Phase werden alle Regeln $R_{i,j}^C$ ausgewertet, die eine mehrfache, sequentielle Ausführung von einem Bricklet b_k definieren. Um die Regel $r_{i,j}^C = b_k \overset{n}{\prec}$ anwenden zu können, muss ebenfalls $b_k \in actset_i$ gelten. Ist dies der Fall, wird die Anzahl der zu bearbeitenden Dokumentfragmente $n = count(actset_i.f(b_k))$ ermittelt (Zeile 9). Hierfür werden die Pfadausdrücke ausgewertet und die Elemente gezählt. Die Anzahl gibt an, wie oft das Bricklet b_k in den Graphen eingefügt werden muss. Dann wird der Typ der Regel für den Knoten mit $b_k.setRuleType(seq)$ gesetzt. Außerdem wird die Anzahl für die Wiederholung mit $b_k.setIteration(n)$ festgelegt.

Schritt 4 (Parallele Ausführung): In der zweiten Phase werden alle Regeln $R_{i,j}^C$ ausgewertet, die eine mehrfache, parallele Ausführung von einem Bricklet b_k definieren. Um die Regel $r_{i,j}^C = b_k \overset{n}{||}$ anwenden zu können, muss ebenfalls $b_k \in actset_i$ gelten. Ist dies der Fall, wird die Anzahl der zu bearbeitenden Dokumentfragmente $n = count(actset_i.f(b_k))$ ermittelt (Zeile 13). Analog zum sequentiellen Fall gibt die Anzahl an, wie oft ein Bricklet b_k in das Subnetz eingeht. Dann wird der Typ der Regel für den Knoten mit $b_k.setRuleType(par)$ gesetzt. Außerdem wird die Anzahl für die Wiederholung mit $b_k.setIteration(n)$ festgelegt.

Schritt 5 (Fehlende Kanten): Im letzten Schritt, muss für alle Knoten kontrolliert werden, ob sie erreichbar sind und ob eine Kante von ihnen weg führt (Zeile 27-34). Dazu werden zuerst alle Knoten, die keine eingehende Kante haben, mit dem Startknoten *start* verbunden (Zeile 29). In einem weiteren Schritt werden alle Knoten, die keine ausgehende Kante haben, mit dem Endknoten *end* verbunden (Zeile 32). Die beiden letzten Schritte stellen sicher, dass Knoten parallel ausgeführt werden. Dieser Fall tritt immer dann auf, wenn keine Regel für den Knoten aktiviert wurde oder der Knoten nur eine eingehende oder ausgehende Kante besitzt.

Zyklenfreiheit

Nachdem ein Graph aus der Menge der Bricklets generiert wurde, wird er auf Zyklenfreiheit überprüft. Gerichtete Graphen können mit Hilfe einer topologischen Sortierung der Knotenmenge auf Zyklenfreiheit getestet werden. Enthält der Graph einen Zyklus, muss die Konstruktion mit einem Fehler abgebrochen werden.

Während der automatischen Konstruktion der Graphen dürfen keine Zyklen entstehen. Eine Möglichkeit das sicherzustellen besteht darin, die Zyklenfreiheit der generierten Graphen bereits während der Modellierung der Konstruktionsregeln R_i^C sicherzustellen. Dazu wird die Menge der Konstruktionsregeln während der Modellierung immer durch die Konstruktion eins Graphen überprüft. Wird eine neue Konstruktionsregel der Menge R_i^C hinzugefügt, wird ein Graph unter Anwendung von R_i^C auf die Menge $refset_i$ erzeugt. Enthält der erzeugte Graph einen Zyklus, muss die Regelmenge angepasst werden. Der Nutzer kann so bereits während der Modellierung sicherstellen, dass kein Zyklus durch die Anwendung der Regelmenge entstehen kann.

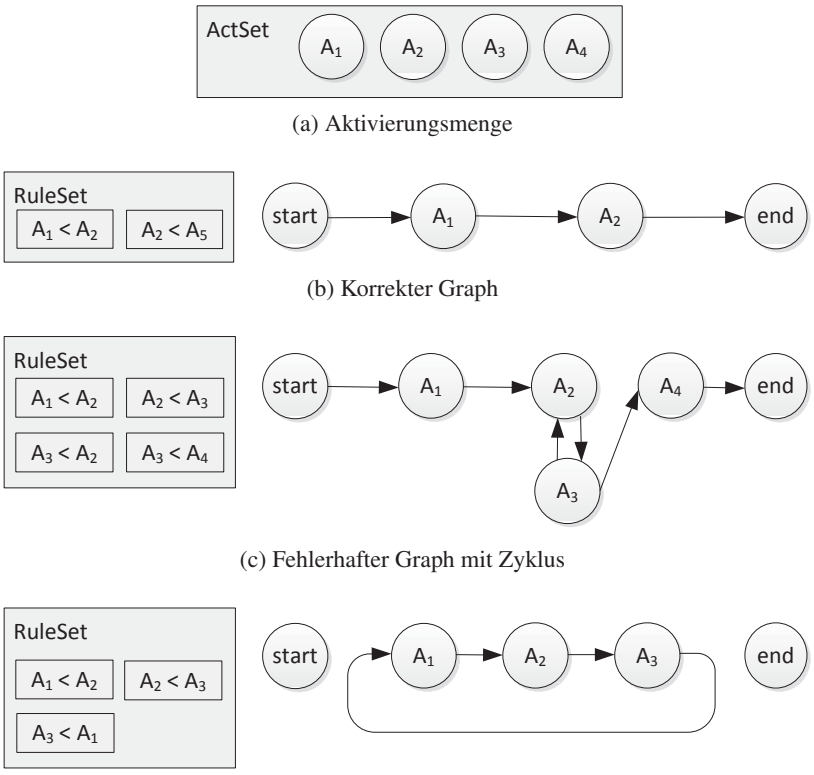
Konfluenz

Die Konfluenz der Regeln wird im *FlexY*-Ansatz erreicht, indem die Generierung von Subnetzen von der Aktivierung der Bricklets entkoppelt wird. Observer aktivieren eine Menge von Bricklets. Diese Menge bestimmt die Konstruktionsregeln, die für die Konstruktion von Subnetzen benötigt werden (siehe Abschnitt 7.4.3). Die Menge der Konstruktionsregeln ist stabil während ein Generator ausgeführt wird. Mit Hilfe des vorgestellten Algorithmus zur Konstruktion von Subnetzen kann weiterhin sichergestellt werden, dass die Liste der Konstruktionsregeln immer in gleicher Weise abgearbeitet wird. Der erzeugte Graph ist entsprechend unabhängig von der Reihenfolge, wie die Konstruktionsregeln aktiviert werden. Deshalb ist Konfluenz gegeben.

Beispiel 7.2 (Konstruktion Graph)

In Abbildung 7.8 werden Beispiele für die Konstruktion von Graphen gezeigt. Im Beispiel wird für die Konstruktion der Graphen die Aktivierungsmenge aus Abbildung 7.8a verwendet.

Die Anwendung der Regeln $A_1 \prec A_2$ und $A_2 \prec A_5$ führt zu einem korrektem Graphen, der in Abbildung 7.8b dargestellt ist. In diesem Fall hat die Regel $A_2 \prec A_5$ keinen Einfluss auf die



(a) Aktivierungsmenge

(b) Korrekter Graph

(c) Fehlerhafter Graph mit Zyklus

(d) Fehlerhafter Graph ohne Start- und Endknoten

Abbildung 7.8: Beispiele für die Generierung von Graphen

Konstruktion des Graphen, weil das Bricklet A_5 nicht aktiviert wurde.

In Abbildung 7.8c wird ein fehlerhafter Graph gezeigt, der einen Zyklus enthält. In diesem Fall kann der Zyklus aufgelöst werden, indem entweder die Regel $A_2 \prec A_3$ oder die Regel $A_3 \prec A_2$ aus der Regelmenge entfernt wird.

Abbildung 7.8d zeigt einen fehlerhaften Graphen. Der letzte Knoten im Graphen ist mit dem ersten Knoten verbunden. Dadurch entsteht ein Zyklus, der aufgelöst werden muss. Außerdem kann in diesem Fall der Graph nicht vollständig erzeugt werden. Der Startknoten wird nicht mit dem Knoten A_1 verbunden, weil dieser bereits eine eingehende Kante vom Knoten A_4 besitzt. Ein ähnliches Problem tritt beim Endknoten auf. Der Knoten A_4 wird nicht mit dem Endknoten verbunden, weil er bereits eine ausgehende Kante zum Knoten A_1 besitzt.

7.4.4 Transformation - Graph auf *YAWL*-Netz

Im letzten Abschnitt wurde beschrieben, wie aus der Menge der aktivierten Bricklets ein Graph erzeugt wird. Der Graph repräsentiert nur ein Zwischenergebnis bei der Generierung

des Subnetzes. Er ist nicht ausführbar und muss deshalb erst in ein ausführbares *YAWL*-Netz transformiert werden. Das Ergebnis der Transformation des Graphen muss ein valides *YAWL*-Prozessmodell sein.

Transformationsregeln

Die Transformation von einem Graphen in ein *YAWL*-Netz wird durch eine Menge von Transformationsregeln definiert.

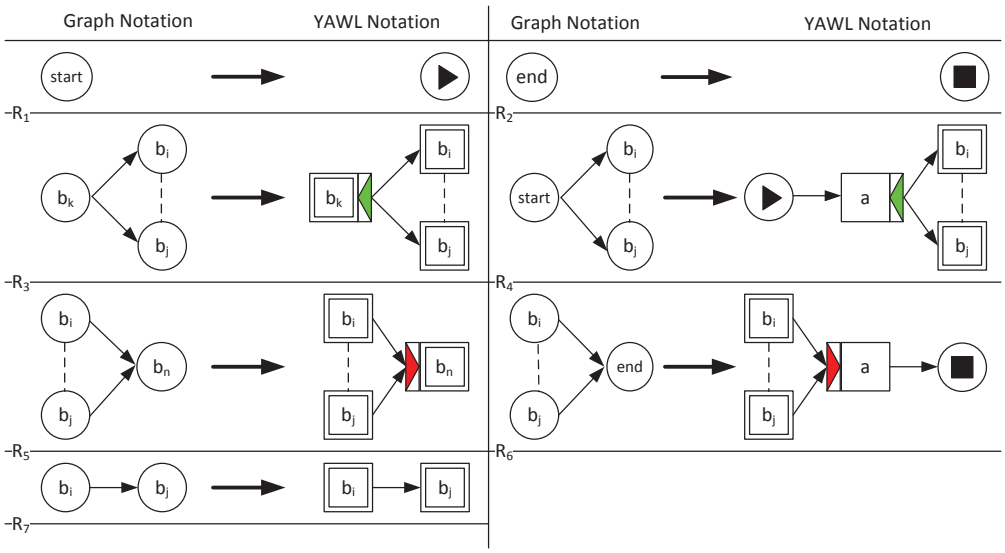


Abbildung 7.9: Regeln für die Transformation eines Graphen in ein *YAWL*-Netz

Die Transformationsregeln sind in Abbildung 7.9 dargestellt und werden im Folgenden einzeln erläutert. Zunächst wird ein leeres *YAWL*-Netz erzeugt, in das entsprechend der Transformationsregeln, die jeweiligen Elemente eingefügt werden. In das Netz werden für jeden Knoten bzw. für jedes Bricklet im Graphen genau eine *composite-task* (zusammengesetzte Aktivität) eingefügt. Diese Aktivität wird immer mit dem Netz, welches dem Bricklet zugeordnet ist, erweitert.

Regel R_1: Für den Startknoten im Graphen wird eine neue *Input Condition* in das *YAWL*-Netz eingefügt.

Regel R_2: Für den Endknoten im Graphen wird eine neue *Output Condition* in das *YAWL*-Netz eingefügt.

Regel R_3: Hat ein Knoten b_k mehr als eine ausgehende Kante, muss eine *YAWL*-Aktivität b_k mit einem *AND-split* in das *YAWL*-Netz eingefügt werden. Die Knoten b_i bis b_j werden auf einfache *YAWL*-Aktivitäten b_i bis b_j abgebildet. Außerdem wird zwischen der Aktivität b_k und allen Aktivitäten b_i bis b_j eine Kontrollflusskante eingefügt.

Regel R_4: Hat der Startknoten *start* mehr als eine ausgehende Kante, wird eine neue *Input Condition* in das *YAWL*-Netz eingefügt. Würden alle ausgehenden Kanten direkt mit den *YAWL*-Aktivitäten verbunden, ist eine Nutzerinteraktion für die Auswahl einer Aktivität notwendig. Eine automatische Abarbeitung der Netze wäre hier nicht mehr möglich. Deshalb muss nach der *Input Condition* eine *YAWL*-Aktivität *a* eingefügt werden, die einen *AND-split* enthält. Die Knoten b_i bis b_j werden auf einfache *YAWL*-Aktivitäten b_i bis b_j abgebildet. Zwischen der *Input Condition* und der Aktivität *a* wird eine Kontrollflusskante eingefügt. Außerdem wird zwischen der Aktivität *a* und allen Aktivitäten b_i bis b_j eine Kontrollflusskante eingefügt.

Regel R_5: Hat ein Knoten b_n mehr als eine eingehende Kante, muss eine *YAWL*-Aktivität b_n mit einem *AND-join* in das *YAWL*-Netz eingefügt werden. Die Knoten b_i bis b_j werden auf einfache *YAWL*-Aktivitäten b_i bis b_j abgebildet. Außerdem wird zwischen allen Aktivitäten b_i bis b_j und der Aktivität b_n eine Kontrollflusskante eingefügt.

Regel R_6: Hat der Endknoten *end* mehr als eine eingehende Kante, wird eine neue *Output Condition* in das *YAWL*-Netz eingefügt. Um sicherzustellen, dass das *YAWL*-Netz für alle möglichen Ausprägungen korrekt abgearbeitet werden kann, darf die *Output Condition* im generierten Netz nur eine eingehende Kante besitzen. Deshalb muss vor der *Output Condition* eine *YAWL*-Aktivität *a* eingefügt werden, die einem *AND-join* enthält. Die Aktivität führt alle parallel ausgeführten Prozesspfade zusammen. So kann sichergestellt werden, dass das Netz nicht beendet wird, bevor nicht alle parallelen Pfade abgearbeitet wurden. Die Knoten b_i bis b_j werden auf einfache *YAWL*-Aktivitäten b_i bis b_j abgebildet. Außerdem wird zwischen allen Aktivitäten b_i bis b_j und der Aktivität *a* eine Kontrollflusskante eingefügt.

Regel R_7: Ist im Graphen eine Sequenz von zwei Knoten b_i und b_j angegeben, so muss diese auf zwei *YAWL*-Aktivitäten b_i und b_j im *YAWL*-Netz abgebildet werden. Zwischen den Aktivitäten b_i und b_j muss eine Kontrollflusskante generiert werden, die eine sequentielle Abarbeitung sicherstellt.

***YAWL*-Netz generieren**

Im nächsten Schritt, wird der generierte Graph in ein *YAWL*-Netz transformiert. Für die Transformation, wird der Graph nach dem Prinzip der Tiefensuche (engl. depth-first search, DFS) bearbeitet. Für jeden Generator t_i^G wird so ein eigenes *YAWL*-Netz SN_i generiert.

```
1 buildNetGraph(v,pre) {
2  if (v equals startNode && edges.size() > 1)
3     pre=insertTask(pre,"a");
4   //iterate over the adjacent vertieces of the vertex
5   foreach u of Adj(v) {
6     //if v is not the end node of the graph
7     if (u not equals {endNode}){
8       //if the bricklet has to be inserted more than once ...
```

```
 9        if (u.getIterations() > 1){
10          //insert n tasks successively into the root net
11          if (u.getRuletype() equals sequentialNTimes)
12            t_n=createNSeqTasks(u);//insertts n acitities t_m
13            add directed edge between successive t_m
14            buildNetGraph(u,t_n);
15          //insert a multiple instance task into the root net
16          else if (u.getRuletype() equals parallelNTimes)
17            buildNetGraph(u,createMultipleInstanceTask(u));
18        }
19        //insert a single task into the root net
20        else t=insertTask(pre,u); buildNetGraph(u,t);
21      }
22      //if v is the end node of the graph
23      else {
24        if(incomingEdgesOf(v).size()>1)
25          pre=insertTask(pre,"a");
26        //insert end condition into the root net.
27        insertTask(pre,Output Condition);
28        }
29      }
30 }
```

Auflistung 7.2: Algorithmus für die Transformation eines Graphen in ein *YAWL*-Netz

Für die Transformation des Graphen in ein *YAWL*-Netz, wird in Auflistung 7.2 ein rekursiver Algorithmus gezeigt. Die Übergabeparameter der Funktion sind gegeben durch den Knoten v und die *YAWL*-Aktivität *pre*. Die *YAWL*-Aktivität *pre* ist die Aktivität, die für den Vorgängerknoten von v erzeugt wurde. Sie wird benötigt, um den Kontrollfluss korrekt erzeugen zu können. Die Funktion $insert(preTask, task)$ fügt die composite-task *task* in das Netz ein und weist der Dekomposition der Aktivität das Netz des Bricklets zu. Gleichzeitig wird eine Kontrollflusskante zwischen der Aktivität und ihrer Vorgänger-Aktivität *preTask* erzeugt. Als Rückgabewert liefert die Funktion die erzeugte *YAWL*-Aktivität.

Schritt 1 (Startknoten): Wenn der Knoten v der Startknoten vom Graphen ist und mehr als eine ausgehende Kante besitzt, wird eine Aktivität a in das *YAWL*-Netz eingefügt. Diese Aktivität wird danach der Variablen *pre* zugewiesen. (Zeile 2-3)

Schritt 2 (Mehrfachknoten): Wenn der Knoten v nicht der Endknoten des Graphen ist, wird überprüft wie oft der Knoten in das Netz eingehen soll (Zeile 7-9). Gilt für den Wert $u.getIterations() > 1$, wird anhand des Regeltyps ermittelt, welcher Typ *YAWL*-Aktivität in das Netz eingefügt werden muss.

Für $u.getRuleType()$ *equals sequentialNTimes* wird die Funktion $createNSeqTasks(u)$ aufgerufen (Zeile 11-14). Die Funktion erzeugt eine Sequenz von n *YAWL*-Aktivitäten t_m vom

Typ $t \wedge m \in \{1,\ldots,n\}$. Sie werden in das *YAWL*-Netz eingefügt. Die erste Aktivität der Sequenz wird mit der Aktivität *pre* durch eine Kontrollflusskante verbunden. Die letzte Aktivität der Sequenz wird als Rückgabewert der Funktion zurückgegeben. Die Anzahl der Aktivitäten ergibt sich aus dem Wert von $u.getIterations()$. Alle Aktivitäten der Sequenz werden nacheinander mit einer Kontrollflusskante verbunden. Dann wird die Funktion $buildNetGraph(u,t_n)$ mit der letzten Aktivität der Sequenz t_n aufgerufen.

Für $u.getRuleType()$ *equals parallelNTimes* wird die Funktion $buildNetGraph(u, createMultipleInstanceTask(u))$ aufgerufen (Zeile 16-17). Die Funktion $createMultipleInstanceTask(u)$ erzeugt eine *YAWL*-Aktivität vom Typ *Multiple-Instance* (MI) und fügt sie in das *YAWL*-Netz ein. Die Aktivität wird mit der Aktivität *pre* durch eine Kontrollflusskante verbunden. Außerdem wird die Aktivität als Rückgabewert der Funktion zurückgegeben. Die Anzahl der Instanzen der MI-Aktivität, ergibt sich aus dem Wert von $u.getIterations()$.

```
1 //replaces u by n activities t_n  vom Typ u∧m∈{1,...,n}
2 createNTasks(u);
3 insertTask(pre,p) //is an \emph{AND-split}-task
4 insertTask(null,s) //is an \emph{AND-join}-task
5 foreach t_o∈{t_1,...,t_n} add directed edge (p,t_o)
6 foreach t_o∈{t_1,...,t_n} add directed edge (t_o,s)
7 buildNetGraph(u,s);
```

Auflistung 7.3: Erweiterung für Algorithmus 7.2 für n parallele Aktivitäten

Anstelle einer MI-Aktivität ist es hier auch möglich, n Aktivitäten in das Netz einzufügen. In Auflistung 7.3 wird dafür eine Erweiterung gezeigt, welche Zeile 17 im Algorithmus aus Auflistung 7.2 ersetzt. In Zeile 2 (Auflistung 7.3), werden n Aktivitäten vom Typ u erzeugt und in das Netz eingefügt. Dann wird je eine *AND-split* (p) und eine *AND-join*-Aktivität (s)in das Netz eingefügt (Zeile 3-4). Im Anschluss werden alle erzeugten Aktivitäten t_o jeweils mit p und s durch eine Kontrollflusskante verbunden. Am Ende wird die Funktion $buildNetGraph(u,s)$ mit der *AND-join*-Aktivität s aufgerufen.

Schritt 3 (Einfachknoten): Geht ein Knoten nur einmal in das Netz ein, so kann direkt eine *YAWL*-Aktivität erzeugt und in das Netz eingefügt werden (Zeile 20). Dann wird die Funktion $buildNetGraph(u,t)$ mit der Aktivität t aufgerufen.

Schritt 4 (Endknoten): Wenn der Knoten v der Endknoten vom Graphen ist und mehr als eine eingehende Kante besitzt, wird eine Aktivität a in das *YAWL*-Netz eingefügt. Diese Aktivität wird danach der Variablen *pre* zugewiesen. (Zeile 22-24). Am Ende wird eine Kontrollflusskante zwischen der Aktivität a und der *Output Condition* eingefügt.

7.4.5 Beispiel

Wenn die Aktivierungsmenge $actset_i$ eines Generators t_i^G nicht leer ist, muss zur Laufzeit ein Subnetz SN_i generiert werden. Im Beispiel wird erläutert, wie das Subnetz für den Generator *feature extraction*^{G} (der Generator wird mit der ID 2 versehen) generiert wird. Dazu wird als erstes die genaue Aktivierungsmenge $actset_2$ bestimmt. Zusammen mit der Menge der Konstruktionsregeln R_2^C wird dann als erster Zwischenschritt ein Graph erzeugt. Dieser Graph wird durch Anwenden der Transformationsregeln aus Abschnitt 7.4.4 in das *YAWL*-Netz SN_2 umgewandelt. Das *YAWL*-Netz SN_2 wird dann durch den Generator ausgeführt. Bevor der

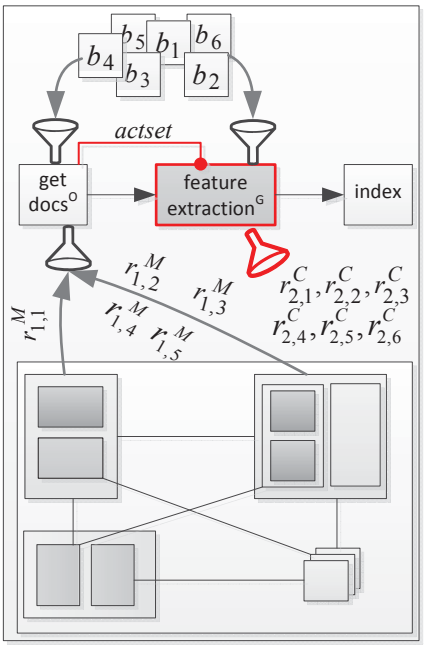


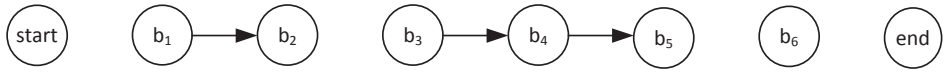
Abbildung 7.10: Beispiel *FlexY*-Prozessfragment, Schritt 4: Generieren und Ausführen eines Subnetzes in einem Generator

Generator ausgeführt wird, hat der Observer *get docs*^{O} die Matching-Regeln $r_{1,1},\ldots,r_{1,5}$ ausgewertet und auf das Beispieldokument aus Anhang B.2 angewendet. Die Regeln $r_{1,1}^M$, $r_{1,2}^M$, $r_{1,3}^M$ (siehe Abschnitt 7.3.3) haben eine Ergebnismenge zurückgeliefert. Für das Beispiel ergibt sich deshalb die Aktivierungsmenge $actset_2 = \{b_1, b_2, b_3, b_4, b_5, b_6\}$.

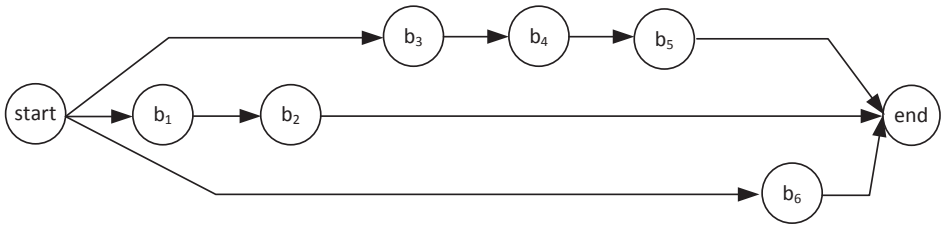
Für den Generator ist die Menge von Konstruktionsregeln $R_2^C = \{r_{2,1}, r_{2,2}, r_{2,3}, r_{2,4}, r_{2,5}, r_{2,6}\}$ definiert. Regel $r_{2,1} = b_1 \prec b_2$ modelliert eine Reihenfolge zwischen den Bricklets b_1 und b_2. Bricklet b_1 muss vor b_2 ausgeführt werden. Regel $r_{2,2} = b_3 \prec b_4$ modelliert eine Reihenfolge zwischen den Bricklets b_3 und b_4. Bricklet b_3 muss vor b_4 ausgeführt werden. Regel $r_{2,3} = b_4 \prec b_5$ modelliert eine Reihenfolge zwischen den Bricklets b_4 und b_5. Bricklet b_4 muss vor b_5 ausgeführt werden. Regel $r_{2,4} = b_1 \overset{n}{\prec}$ beschreibt, dass das Bricklet b_1 n-mal sequentiell in das Prozessmodell eingeht. Regel $r_{2,5} = b_4 \overset{n}{\|}$ beschreibt, dass das Bricklet b_4 n-mal parallel in

das Prozessmodell eingeht. Regel $r_{2,6} = b_6 \overset{n}{\|}$ beschreibt, dass das Bricklet b_6 n-mal parallel in das Prozessmodell eingeht. Für die Anwendung der Regeln, wird im Beispiel davon ausgegangen, dass der Wert für n jeweils 2 ist.

(a) Schritt 1: Graph initialisieren



(b) Schritt 2: Regel für Reihenfolge anwenden



(c) Schritt 3: Parallele Knoten mit Start- und Endknoten verbinden

Abbildung 7.11: Graph aus der Menge der aktivierten Bricklets generieren

Im ersten Schritt wird der Graph mit der Menge der aktivierten Bricklets $actset_2$ initialisiert. Abbildung 7.11a zeigt den resultierenden Graphen. Im zweiten Schritt werden alle Regeln angewendet. Abbildung 7.11b zeigt den erweiterten Graphen. Die Abbildung stellt die Annotation der Knoten b_1, b_4 und b_6 nicht dar. Sie wurden zusätzlich mit der Anzahl der Iterationen n und den Pfadausdrücken erweitert. Im letzten Schritt werden alle Knoten ohne eingehende Kante mit dem Startknoten verbunden. Knoten ohne ausgehende Kante werden mit dem Endknoten verbunden. Abbildung 7.11c zeigt den vollständigen Graphen.

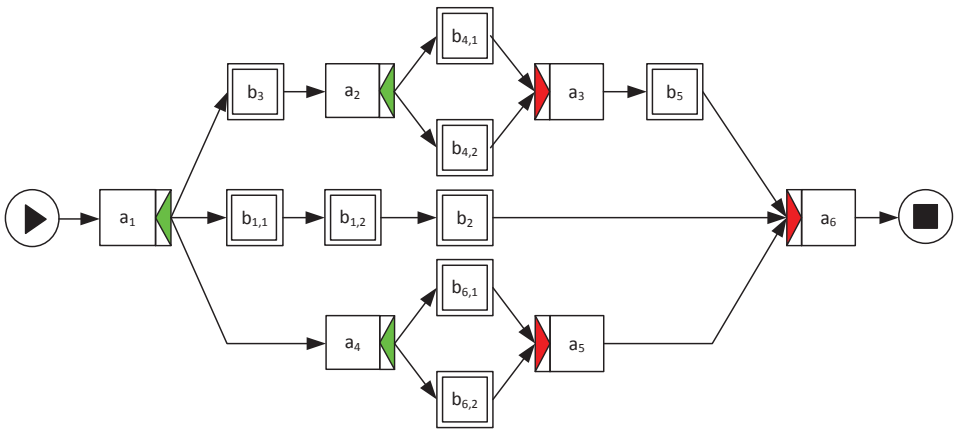


Abbildung 7.12: Geniertes *YAWL*-Subnetz SN_2

Der Graph aus Abbildung 7.11c dient als Ausgangspunkt für die Transformation in ein *YAWL*-Netz. Weil mehrere Knoten im Graphen durch den Startknoten erreichbar sind, wird Transformationsregel R_4 angewendet. Als Ergebnis wird eine Input-Condition zusammen mit der Aktivität a (a ist eine *AND-split*-Aktivität) in das Netz eingefügt. Dann werden alle Knoten expandiert, die mehrfach in das Netz eingehen sollen. Das Bricklet b_1 muss entsprechend der Konstruktionsregel $r_{2,4}$ genau 2-mal sequentiell in das Netz eingehen. Hier werden die Aktivitäten $b_{1,1}$ und $b_{1,2}$ eingefügt und miteinander verbunden. Das Bricklet b_4 wird entsprechend der Konstruktionsregel $r_{2,5}$ genau 2-mal parallel in das Netz eingefügt. Entsprechend dem Algorithmus und der Erweiterung in Auflistung 7.3, werden jeweils eine Aktivität a_2 mit einem *AND-split* und eine Aktivität a_3 mit einem *AND-join* eingefügt und mit den Aktivitäten $b_{4,1}$ und $b_{4,2}$ verbunden. Das Bricklet b_6 wird entsprechend der Konstruktionsregel $r_{2,6}$ auch 2-mal parallel in das Netz eingefügt. Entsprechend dem Algorithmus und der Erweiterung in Auflistung 7.3, werden jeweils eine Aktivität a_4 mit einem *AND-split* und eine Aktivität a_5 mit einem *AND-join* eingefügt und mit den Aktivitäten $b_{6,1}$ und $b_{6,2}$ verbunden. Im letzten Schritt kommt die Transformationsregel R_6 zur Anwendung. Die parallelen Aktivitäten b_5, b_2 und a_5 müssen mit der Aktivität a_6 wider synchronisiert werden. Aktivität a_6 wird deshalb mit einem *AND-join* erweitert und zusätzlich mit der *Output-Condition* verbunden.

7.5 Diskussion

In der Literatur werden unterschiedliche Ansätze diskutiert, die Datenabhängigkeiten von Prozessen betrachten. Die Ansätze werden entsprechend der Einteilung aus Abschnitt 2.2.3 kategorisiert: Flexibilität durch Modellierung und Abweichung, Flexibilität durch Unterspezifikation und Flexibilität durch Modifikation.

7.5.1 Flexibilität durch Modellierung

Datenabhängigkeiten können bereits während der Modellierung von Prozessmodellen genutzt werden, um optimierte Prozessmodelle zu erzeugen (siehe Abschnitt 2.2.3.1 und Abschnitt 2.2.3.2).

Management von Prozessvarianten
Eine Möglichkeit besteht darin, Prozessmodelle kontextabhängig zu Konfigurieren. Die Ansätze *Provop* [HBR08, Hal10] und C-YAWL [GAJVR08, GWJV+09] sind Beispiele für die Umsetzung von Prozessvarianten mittels Konfiguration (siehe Abschnitt 1.3.2.2).

Provop bietet die Möglichkeit ein Basisprozessmodell mittels Änderungsoperationen an den Kontext anzupassen. Die Anwendung von Änderungsoperationen bietet den Vorteil, dass

das Basisprozessmodell nicht vollständig modelliert sein muss. Es können Prozessfragmente hinzugefügt aber auch gelöscht werden. Die Konfiguration von einem Basisprozessmodell wird zusätzlich durch Kontextregeln kontrolliert. Mit Hilfe der Kontextregeln können sehr komplexe Bedingungen ausgedrückt werden, die über die hier vorgestellten Möglichkeiten hinausgehen. Für den Publikationsprozess ist *Provop* trotzdem nur bedingt geeignet. Kann bereits im Vorfeld auf bestimmte Dokumenttypen eingeschränkt werden, so bieten Prozessvarianten einen Vorteil gegenüber einem einzigen Prozessmodell. Keinen Vorteil bietet *Provop*, wenn Autoren die Dokumenttypen während der Bearbeitung frei wählen können. *Provop* bietet keine Möglichkeit, Änderungen am Prozessmodell zur Laufzeit vorzunehmen. Kontextregeln werden in diesem Fall auf Kontrollflussentscheidungen abgebildet. Eine Anwendung der Konfiguration führt hier dazu, dass alle Kontextbedingungen auf den Kontrollfluss abgebildet werden müssen. Die erzeugten Prozessmodelle sind also ähnlich komplex.

C-YAWL erweitert *YAWL* mit einem Ansatz für die Prozesskonfiguration. Im Gegensatz zu *Provop* werden keine Änderungsoperationen angeboten. Es besteht nur die Möglichkeit, mit Hilfe von konfigurierbaren Elementen das Prozessmodell zu reduzieren. Genau wie beim *Provop* Ansatz kann hier nur eine Verbesserung erreicht werden, wenn im Vorfeld auf bestimmte Dokumenttypen eingeschränkt wird. Zur Laufzeit müssen Kontextregeln auch auf Kontrollflussentscheidungen abgebildet werden. Die Konfiguration der Prozessmodelle basiert auf der Auswertung von Fragebögen. Um Kontextinformationen wie die Zustände von Dokumenten abzubilden, müssten eine Reihe von wesentlichen Änderungen vorgenommen werden.

Datenzentriert Prozessmodelle

Der Fokus von datenzentrierten Prozessmodellen liegt auf der Verfügbarkeit von Daten während der Ausführung der Prozessinstanzen. Die Aktivierung von Aktivitäten wird an den Zustand von Daten gebunden (siehe Abschnitt 1.3.2.1).

In Corepro wird die Ableitung von Prozessmodellen aus Datenmodellen ermöglicht [MRH07, MRH08]. Auch hier wird der Prozessfortschritt durch den Zustand von Datenobjekten beeinflusst. Im Gegensatz zu *PHILharmonicFlows* besteht hier die Möglichkeit, dass Prozessmodell in Ausnahmefällen zu ändern. So können Prozessinstanzen an geänderte Dokumente angepasst werden. Der Ansatz kombiniert Techniken der Prozesskonfiguration mit denen der Modifikation von Prozessen. Datenabhängigkeiten, wie sie für den Publikationsprozess benötigt werden, können nicht abgebildet werden. Die Datenabhängigkeiten in *Corepro* werden nur durch Zustandsautomaten beschrieben. Der Zustand ist außerdem nicht abhängig von Attributen und Werten der modellierten Daten. Im Publikationsprozess spielen aber besonders die Attribute und Werte der Dokumente eine wichtige Rolle. Änderungsoperationen am Prozessmodell sollen die Reaktion auf unvorhersehbare Ereignisse ermöglichen.

Änderungen an Dokumenten sind im Publikationsprozess keine unvorhersehbaren Ereignisse und sollten deshalb auch nicht so behandelt werden.

Mit Proclets wird die Möglichkeit, komplexe Prozessstrukturen mit Hilfe von Proclets zu zerlegen, vorgestellt [AMR09, MRA+10]. So kann für den Publikationsprozess beispielsweise jede dokumentabhängige Operation durch ein Proclet umgesetzt werden. Der Geschäftsprozess wird ebenfalls durch ein Proclet umgesetzt, welches die Aktivierung der dokumentabhängigen Proclets koordiniert. So kann z. B. die Indizierung von einem Dokument durch eine Proclet-Klasse umgesetzt sein. Weil erst zur Laufzeit feststeht welchen Zustand ein Dokument hat, muss das Proklet alle Varianten für alle Dokumenttypen umsetzen. Hier kann die weitere Zerlegung in Proclets erfolgen, indem für jeden Dokumenttyp nochmals eine Proclet-Klasse für die Indizierung angelegt wird. Damit steigt aber die Anzahl der Proclet-Klassen sehr stark an. Ein weiterer Nachteil liegt in der Aktivität-zentrierten Umsetzung der Proclets. Externe Datenquellen können nur mit Hilfe von Aktivitäten angesprochen werden. Hier kann der *FlexY*-Ansatz durch den Einsatz von externen Variablen direkt auf Dokumente in externen Datenquellen zugreifen (siehe Kapitel 5 und 6). Damit entfällt auch in den Aktivitäten die Auswertung, ob ein externes Datenelement einen bestimmten Zustand hat oder nicht.

A Framework for Document-Driven Workflow Systems. In [WK05] stellen Wang und Kumar einen Ansatz basierend auf Triggern vor. Ein Kontrollfluss wird nicht beschrieben. Trigger ermöglichen die Aktivierung beliebiger Aktivitäten in Abhängigkeit von Zustandsänderungen an Dokumenten. Die Beschreibung fest vorgegebener Arbeitsabläufe ist so sehr schwer. Im Publikationsprozess werden aber genau diese Arbeitsabläufe für die Modellierung der Geschäftsprozesse der digitalen Bibliotheksanwendungen benötigt. Das Framework bietet außerdem nur die Möglichkeit Daten bzw. Dokumente zu verwenden, die in einer relationalen Datenbank verwaltet werden. Um externe Datenquellen einzubinden, müssten diese Trigger unterstützen und an das System angebunden werden. Mit dem Konzept der Trigger können nur Zustandsübergänge überwacht werden. Im Publikationsprozess muss es aber auch möglich sein, den Zustand von Daten zu einem bestimmten Zeitpunkt zu ermitteln. Die Aktivierung von Triggern ist weiterhin schwer zu kontrollieren. Die Aktivierungsreihenfolge darf jedoch keinen Einfluss auf die Reihenfolge der Ausführung von dokumentabhängigen Operationen haben. Im Publikationsprozess ist das von besonderem Interesse, weil Operationen abhängig voneinander sind.

7.5.2 Flexibilität durch Unterspezifikation

In diesem Abschnitt werden Ansätze diskutiert, die Flexibilität durch Unterspezifikation ermöglichen (siehe Abschnitt 2.2.3.3).

Flexibility as a Service (FAAS) wird durch das WFMS *YAWL* unterstützt [AAH+09]. *YAWL* bietet über ein einheitliches Interface die Möglichkeit, unterschiedliche Services für die Ausführung von Aktivitäten einzubinden. In einem *YAWL*-Prozessmodell können beliebig viele Aktivitäten als Platzhalter definiert werden, die einen solchen Service integrieren. Die Idee, Flexibilität als Service bereitzustellen, wird auch vom *FlexY*-Ansatz ausgenutzt. Eine Beispielhafte Umsetzung wird für Worklets [AHEA06a] und Declare [PSA07, APS09] bereitgestellt. Beide Ansätze wurden bereits allgemein in Abschnitt 1.3.2.3 vorgestellt.

Worklets setzen das Konzept *late-binding* um. Ähnlich wie im *FlexY*-Ansatz, können in einem Prozessmodell beliebig viele Platzhalter für die Ausführung von Worklets verwendet werden. Im Unterschied zum *FlexY*-Ansatz kann für jeden Platzhalter nur ein Worklet ausgeführt werden. Für jeden Dokumentzustand müsste ein eigenes Worklet definiert sein. Zum Beispiel ein Worklet, das die Aktivitäten für die Indizierung von einem Dokument mit Bildern und Videos beschreibt. Ein weiteres Worklet, das die Aktivitäten für die Indizierung von einem Dokument mit Bildern und Text beschreibt. Für jede Variante muss ein eigenes Worklet existieren. Die Auswahl der Worklets ist an Regeln gebunden, die nur auf Prozessvariablen Zugriff haben. Externe Daten können nicht direkt für die Auswahl der Worklets verwendet werden.

***YAWL* und Declare** werden nach dem Prinzip FAAS zusammen verwendet. Durch den Einsatz von Declare wird der FAAS Ansatz mit einem deklarativem Konzept kombiniert. Mit dem Ansatz lassen sich Publikationsprozesse ähnlich umsetzen, wie mit dem hier vorgestellten *FlexY*-Ansatz. Declare bietet wesentlich mehr Modellierungskonzepte als der *FlexY*-Ansatz an. So können weitaus komplexere Regelmengen erzeugt werden, welche die Ausführung dokumentabhängiger Prozessbereiche beschreiben. Mit Declare ist es auch möglich, *YAWL*-Modelle auszuführen (siehe [AAH+09]). Deshalb könnten die dokumentabhängigen Prozessbereiche weiterhin mit *YAWL* modelliert werden. Der *FlexY*-Ansatz bietet dennoch den Vorteil, dass Bedingungen auf externen Datenquellen überprüft werden können. Das Konzept der externen Datenquellen und der Observer steht in *YAWL* und Declare nicht zur Verfügung. Ein weiterer Vorteil von *FlexY* besteht darin, dass nur eine Sprache (*YAWL*) verwendet wird. Alles kann in einem Modell modelliert werden. Für die Ausführung der Prozessmodelle wird deshalb nur ein WFMS benötigt, was den Aufwand für die Modellierung und Administration verringert.

Pockets of Flexibility ist ein Ansatz, der das Konzept *late-modeling* umsetzt [SOS05]. Analog zum *FlexY*-Ansatz werden hier Platzhalter verwendet. Nutzer müssen zur Laufzeit ein Prozessfragment modellieren, welches durch den Platzhalter (Pocket) ausgeführt wird. Im Gegensatz zum *FlexY*-Ansatz wird nur eine manuelle Komposition von Prozessfragmenten durch den Nutzer unterstützt. Weiterhin wird die Regelmenge nicht für die eigentliche Komposition der Prozessfragmente eingesetzt. Die Regelmenge stellt sicher, dass die manuell

erzeugten Prozessfragmente korrekt sind. Außerdem können die Regeln nicht gegenüber externe Datenquellen ausgewertet werden.

In [DYWL04] wird ein Ansatz vorgestellt, der ebenfalls das Konzept *late-modeling* umsetzt. Deng et al. nutzen ein ähnliches Konzept für die Komposition von Platzhaltern, wie im *FlexY*-Ansatz. Für jeden Platzhalter werden drei unterschiedliche Regeln definiert. Es muss angegeben werden, welche Aktivitäten zur Auswahl stehen und welche Ausschlusskriterien zwischen ausgewählten Aktivitäten bestehen. Die dritte Regelmenge beschreibt Bedingungen für die korrekte Komposition der ausgewählten Aktivitäten. Im Gegensatz zum *FlexY*-Ansatz muss die Auswahl der Aktivitäten manuell durch den Nutzer erfolgen. Die Komposition der Subnetze erfolgt entweder automatisch oder durch den Nutzer. Datenabhängigkeiten für die Auswahl und Komposition der Aktivitäten können nicht modelliert werden. Um einen Publikationsprozess mit dem Ansatz umsetzen zu können, müssten Erweiterung für die Integration externer Datenquellen bereitgestellt werden. Außerdem fehlt die Möglichkeit, in Abhängigkeit vom Zustand einzelner Dokumente, Prozessfragmente zu generieren (Konzept Observer).

Flexibilität durch Modifikation

In der Literatur werden verschiedene Ansätze (WASA2 [VW99, Wes00], WIDE [CCPP98] oder CAKE [MSKB07]) vorgestellt, die Flexibilität durch Modifikation umsetzen (siehe Abschnitt 2.2.3.4). In diesem Absatz sollen die beiden Ansätze *ADEPT* und *AgentWork* stellvertretend betrachtet werden.

ADEPT bietet die Möglichkeit, Änderungen an Prozessinstanzen mit Hilfe von Änderungsoperationen umzusetzen [RD98, RRKD05]. Für die Umsetzung von Publikationsprozessen eignet sich ADEPT nur bedingt. An den Prozessinstanzen können komplexe Änderungen vorgenommen werden. Die Änderungsoperationen sind für die Behandlung von Ausnahmen gedacht. Daher müssen sie durch den Nutzer angewendet werden. Eine automatische Komposition von Prozessfragmenten, wie im *FlexY*-Ansatz, ist nicht vorgesehen.

AgentWork ist ein System, das von Müller et al. in [MGR04] vorgestellt wird. Es bietet die Möglichkeit Prozessinstanzen automatisch anzupassen. Für die Umsetzung der Instanzmodifikation nutzt *AgentWork* das WFMS ADEPT. Um mit Ausnahmen während der Ausführung von klinischen Prozessen umgehen zu können, wird ein Ansatz basierend auf ECA-Regeln eingeführt. Eine Erweiterung mit temporaler Logik soll den zeitlichen Aspekt der Ausnahmebehandlung beschreiben. Für die Beschreibung der Ereignisse wird ActiveTFL (Active Temporal Frame Logic) eingeführt, die auf Datenbanktrigger abgebildet werden. Ereignisse stoßen verschiedene Änderungsoperationen an den Prozessinstanzen an. Diese Änderungsoperationen sind auf die Ausnahmebehandlung für einzelne Aktivitäten beschränkt. Diese Einschränkung würde im Publikationsprozess dazu führen, dass eine sehr hohe Anzahl von Änderungsoperationen beschrieben werden müsste. *AgentWork* bietet die Möglichkeit,

Zustandsänderungen mit Hilfe von ActiveTFL zu ermitteln. Für den Ansatz gelten ähnliche Einschränkungen wie für den in [WK05] vorgestellten Ansatz. Der Zustand von Daten wird im Publikationsprozess auch zu einem bestimmten Zeitpunkt benötigt. Zum Zeitpunkt der Auswahl dokumentabhängiger Prozessfragmente, muss der exakte Zustand von einem Dokument bestimmt werden können. Die Aktivierung von Triggern ist weiterhin schwer zu kontrollieren. In AgentWork ist es nicht ausgeschlossen, dass inkompatible Aktionen gleichzeitig ausgelöst werden können. Dann ist ein manueller Eingriff des Nutzers notwendig. Im Publikationsprozess ist ein solcher, manueller Eingriff durch den Nutzer jedoch nicht möglich.

7.6 Zusammenfassung

Mit *FlexY* stellen wir einen Ansatz vor, flexible Prozessmodelle für Publikationsprozesse umzusetzen. Observer-Aktivitäten bieten die Möglichkeit, Dokumentänderungen zu überwachen und Änderungen an Prozessinstanzen umzusetzen. In digitalen Bibliotheksanwendungen sind die Dokumente häufig sehr komplex und werden in spezialisierten Systemen verwaltet. Der *FlexY*-Ansatz bietet deshalb die Möglichkeit, direkt den Zustand von Dokumenten aus externen Datenquellen zu ermitteln. In Abhängigkeit von den Dokumentzuständen können so die Prozessinstanzen angepasst werden.

Im Prozessmodell können anwendungsabhängige Prozessbestandteile durch einen Basisprozess modelliert werden. Dokumentabhängige Prozessbestandteile werden durch Bricklets beschrieben, die als *YAWL*-Netz definiert sind. Bricklets können zur Laufzeit in die Prozessinstanz eingefügt werden. Weil am Basisprozess keine Änderungen zugelassen sind, dürfen Bricklets nur in dafür vorgesehenen Bereichen in den Basisprozess eingefügt werden. Diese Bereiche werden als Generator-Aktivitäten bezeichnet.

Generatoren generieren aus den aktivierten Bricklets ein Subnetz und führen das Subnetz aus. Konstruktionsregeln stellen sicher, dass aus den Bricklets ein korrektes Subnetz generiert wird.

Durch die automatische Anwendung von Matching-Regeln wird eine dynamische Anpassung der Prozessinstanzen ermöglicht. Matching-Regeln beschreiben die Prozessbereiche, die angepasst werden müssen. Generatoren ermöglichen die automatische Komposition und die Ausführung von Subnetzen für die flexiblen Prozessbereiche.

Teil III

Validierung der Konzepte

Kapitel 8

Prototypische Umsetzung

Im Folgenden wird anhand eines Prototypen die Umsetzbarkeit der vorgestellten Konzepte aus den Kapiteln 5, 6 und 7 gezeigt. Für die Realisierung des Prototypen wurde das WFMS *YAWL* in der aktuellen Version 2.2 verwendet.

Das Kapitel gliedert sich wie folgt: In Abschnitt 8.1 wird die allgemeine Architektur des Prototypen vorgestellt. Dafür werden die von YAWL für die Kommunikation bereitgestellten Interfaces beschrieben. Abschnitt 8.2 stellt ein Framework vor, das die transaktionale Integration von externen Datenquellen erlaubt. Es wird ein Überblick über die einzelnen Komponenten des Frameworks gegeben. Für die prototypische Umsetzung mit *YAWL* wird gezeigt, wie das Framework in *YAWL* eingebunden wird und wie *YAWL*-Prozessmodelle mit dem *YAWL*-Editor erstellt werden können. In Abschnitt 8.3 wird die prototypische Umsetzung des *FlexY*-Konzepts vorgestellt. Der Service ist als *YAWL custom service* umgesetzt und besteht aus einer Komponente für die Überwachung (Observer) von Dokumentzuständen und einer Komponente für die Generierung (Generator) von Prozessmodellen. Es wird gezeigt, wie der Service umgesetzt ist und wie er in *YAWL* eingebunden wird. Außerdem wird beschreiben, wie mit dem *YAWL*-Editor *YAWL*-Prozessmodelle für den *FlexY*-Service erstellt werden können.

8.1 Architektur des Prototypen

Für die Umsetzung des Prototypen wurde das *YAWL* WFMS erweitert. Welche Schnittstellen der Prototyp nutzt, wird im Folgenden erläutert.

Die Kernkomponente von *YAWL* stellt die *YAWL*-Engine dar. Sie erzeugt Prozessinstanzen und kontrolliert deren Abarbeitung. Während der Ausführung weist die Engine jeder Aktivität einen *YAWL*-Service zu, der dann ausgeführt wird. Ein wesentliches Merkmal von *YAWL*

besteht darin, dass die *YAWL*-Services völlig unabhängig von der Engine ausgeführt werden. Die Engine hat deshalb kein Wissen und keinen Einfluss über die Operationen eines *YAWL*-Services. Services werden aus Sicht er Engine auch als *black-boxes* betrachtet [ADR10]. In der Standardkonfiguration stellt *YAWL* bereits verschiedene *YAWL*-Services bereit, um z. B. Ressourcen oder Webservices einzubinden (siehe auch [ADR10]). Im Folgenden wird nicht zwischen der Sprache *YAWL* und der Prozess-Engine unterschieden.

Für die Kommunikation mit *YAWL*-Services stellt *YAWL* verschiedene Interfaces bereit. Die Interfaces werden in Abbildung 8.1 gezeigt.

- *Interface A* ermöglicht es, Prozessspezifikationen in *YAWL* hochzuladen oder zu löschen. Das Interface bietet außerdem noch die Möglichkeit, externe Services an- und abzumelden.
- *Interface B* wird verwendet, um *YAWL*-Services aufzurufen. Das Interface stellt außerdem eine Schnittstelle für die Kommunikation zwischen Service und *YAWL* bereit. Über diese Schnittstelle können Prozessinstanzen gestartet und Aktivitäten ein- und ausgecheckt[1] werden. Das Interface stellt außerdem Prozessdaten und Zustandsinformationen über die Prozessinstanz bereit.
- *Interface E* stellt Log-Informationen über die Prozesse bereit.
- *Interface X* stellt *YAWL*-Services Informationen über Fehler bereit, die von den Services für eine Ausnahmebehandlung genutzt werden können.

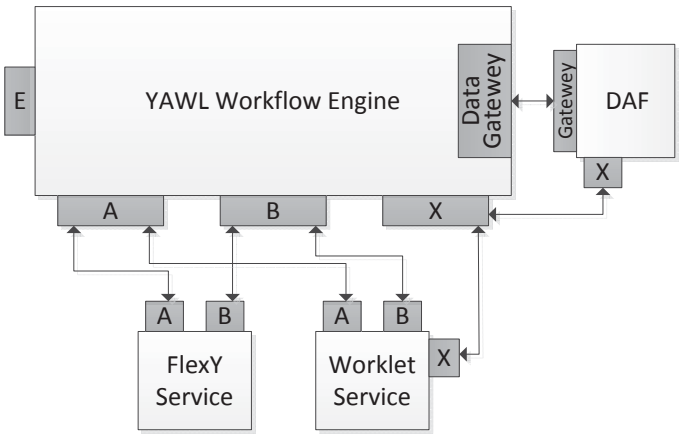


Abbildung 8.1: Allgemeine Architektur von *YAWL* mit den vorgestellten Erweiterungen

Der Prototyp wird in zwei Services aufgeteilt. Das *Data Access Framework* (DAF) implementiert die transaktionale Integration von externen Datenquellen. Dafür stellt *YAWL* ein *Java-Interface* (*DataGateway*) bereit, das vom DAF für den Datenaustausch mit Aktivitäten genutzt wird. Zusätzlich nutzt das DAF das Interface X, um z. B. die Ausführung von Aktivitäten zu unterbrechen. Der *FlexY*-Service nutzt die Interfaces B und A. Das Interface B wird

[1] Mit *check in* und *check out* wird der Vorgang beschrieben, bei dem *YAWL* die Kontrolle für eine Aktivität an einen *YAWL*-Service übergibt bzw. die Kontrolle wieder zurück erhält.

von *YAWL* genutzt, um den *FlexY*-Service aufzurufen. Über das Interface A kann der *FlexY*-Service die generierten Prozessmodelle bereitstellen und starten. Um die beiden Services korrekt nutzen zu können, wurde der *YAWL*-Editor um entsprechende Modellierungskonzepte erweitert.

8.2 Implementierung von *tx+YAWL*

Die in Kapitel 5 und Kapitel 6 vorgestellten Erweiterungen, um externe Datenquellen einzubinden, werden vom *YAWL* WFMS nicht direkt unterstützt. Wir haben deshalb ein Framework entwickelt, mit dessen Hilfe Plug-ins im WFMS bereitgestellt werden können. Außerdem wurde der *YAWL*-Editor so erweitert, dass externe Variablen verwendet werden können.

8.2.1 Zugriff auf externe Datenquellen

Das *Data Access Framework* (DAF) verbindet externe Variablen mit Datenquellen. Hierfür verwaltet das DAF eine Menge von Plug-ins und steuert den Zugriff auf diese.

Connection-Management. Wenn eine hohe Anzahl von Prozessinstanzen die gleichen externen Datenquellen verwenden, muss der Zugriff durch ein geeignetes Management unterstützt werden. Die vorgestellte Architektur ist so ausgelegt, dass der Zugriff auf ein Plug-in nicht gleichzeitig durch mehrere Prozessinstanzen erfolgen darf. Ansonsten können Probleme wie der Ausfall der Datenquelle oder ähnliches auftreten.

Ein *Connection-Management* kann hier Abhilfe schaffen. Das DAF muss Möglichkeiten bieten, Verbindungen zu einer entsprechende Menge von Plug-ins zu verwalten. Der gleichzeitige Zugriff mehrerer Prozessinstanzen auf eine Datenquelle muss ebenfalls unterstützt werden.

Service Integration. Der Zugriff auf eine Datenquelle und die Verarbeitung der Anfragen und Abfrageergebnisse hängt zum einen von der Datenquelle ab, zum anderen von der Anwendung. Statische Datenquellen benötigen eine andere Verarbeitungskette als beispielsweise Datenquellen die kontinuierliche Datenströme liefern. Das DAF muss entsprechend erweiterbar sein. Services ermöglichen eine gezielte Bearbeitung bestimmter Teilaufgaben entlang der Verarbeitungskette.

Flexible Ausführungsreihenfolge von Services. Nicht jedes Prozessmodell verwendet die gleichen Datenquellen und hat die gleichen Anforderungen an die Verarbeitungskette des DAFs. Die statische Aneinanderreihung von verschiedenen Services würde die Integration

von Datenquellen sehr unflexibel machen. Nicht in jedem Fall müssen Integritätsbedingungen überprüft werden. Die Einbindung eines Transaktionslayers ist ebenfalls nur notwendig, wenn dieser explizit modelliert ist. Eine dynamische Anpassung der auszuführenden Services und gegebenenfalls deren Ausführungsreihenfolge ermöglicht es, flexibel auf die Anforderungen der Prozessmodelle zu reagieren. Das DAF muss entsprechend leicht anzupassen sein.

Data Access Framework Architektur

Die oben genannten Anforderungen werden durch die in Abbildung 8.2 gezeigte Architektur

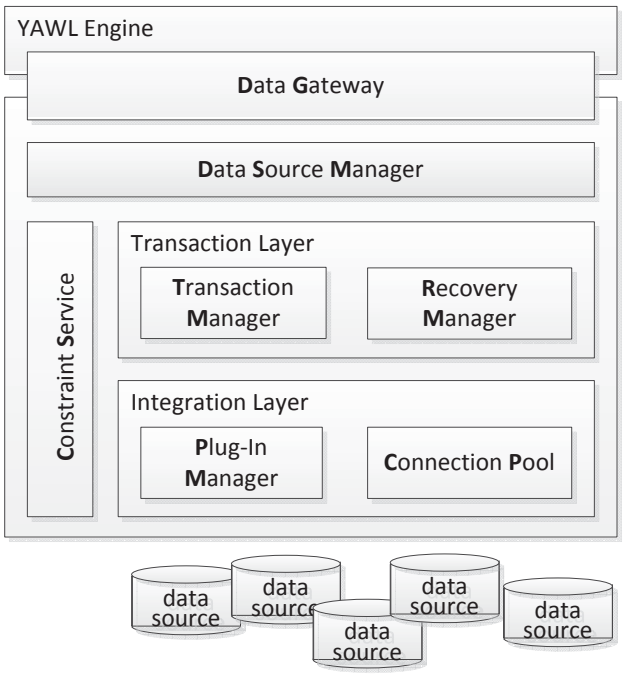


Abbildung 8.2: Data Access Framework Architektur

umgesetzt. Das DAF bietet eine Schnittstelle für Aktivitäten an, über die verschiedene Datenquellen mit Hilfe von Plug-ins integriert werden können. Hierfür verwaltet das DAF eine Menge von Plug-ins.

Innerhalb der verschiedenen Layer werden Services zusammengefasst, die in einer vorgegebenen Reihenfolge abzuarbeiten sind. Die Umsetzung des Frameworks ist inspiriert von dem *Stream Input-Output System*, welches in Unix-Systemen verwendet wird (siehe [Rit84]). Zwischen den Services haben wir eine Zwei-Wege Kommunikation umgesetzt. In [Rit84] wird die Abarbeitung mit der einer *shell pipeline* verglichen, mit dem Unterschied, dass der Datenfluss in beide Richtungen stattfindet. In der gleichen Weise kommunizieren unsere Services miteinander, indem sie Nachrichten zu ihren Nachbarn weiterleiten. Die vom WF-MS verschickte Nachricht oder auch Anfrage wird an die Services weitergereicht, die diese dann der Reihe nach verarbeiten. Die Nachricht wird dabei unabhängig von anderen Services

verarbeitet und teilweise auch verändert. Auch wenn die Services unabhängig voneinander sind, kann eine Nachricht die Verarbeitung durch einen Service konfigurieren.

Die Services kommunizieren dabei ausschließlich durch den Austausch von Nachrichten in Form einer proprietären Datenstruktur. Die Datenstruktur besteht aus den Teilen `DATA`, `MAPPING` und `SYSTEM`. Der Teil `DATA` enthält dabei die eigentlichen Daten, die zwischen Datenquelle und Aktivität ausgetauscht werden. Im Teil `MAPPING` sind Informationen über das zu verwendende Plug-in, die aufrufende Prozessinstanz mit der zugehörigen Instanz-ID der Aktivität und dem eigentlichen externen Mapping *extParam* enthalten. Der Teil `SYSTEM` wird für Statusmeldungen einzelner Services, sowie für die Weitergabe von Fehlernachrichten verwendet.

Die Auswahl der zu durchlaufenden Service-Schichten wird hauptsächlich durch Lese- und Schreibstrategien beeinflusst. Die dynamische Einbindung unterschiedlicher Services ermöglicht es so, auf unterschiedliche Integrationsstrategien für Daten flexibel zu reagieren. Sind beispielsweise Integritätsbedingen im Prozessmodell angegeben, muss ein entsprechender Service eingebunden werden.

8.2.2 Standard-Services für die Datenintegration

Für den Zugriff auf externe Datenquellen muss das DAF verschiedene Standard-Services bereitstellen, die atomare Anfragen auf Plug-ins unterstützen. Im Folgenden werden hierfür mehrere Services und die Anbindung des DAF an *YAWL* vorgestellt.

Framework-Schnittstelle
Als Schnittstelle für *YAWL* wird die Ebene *Data Gateway* (DG) (siehe Abbildung 8.2) eingeführt. Sie kapselt das DAF gegenüber *YAWL*. Das DG wird von *YAWL* direkt aufgerufen, wenn eine externe Variable lesend oder schreibend verwendet wird.

Die eigentliche Konfiguration des Frameworks findet im *Data Source Manager* (DSM) statt (siehe Abbildung 8.2). In Abhängigkeit der Lese- und Schreibstrategien, sowie der Integritätsbedingungen wird eine vorher festgelegte *Strategie* ausgewählt. Eine Strategie beschreibt, welche Services in welcher Ebene für die Abarbeitung der Anfrage benötigt werden. Falls z. B. eine Anfrage mit der Konfiguration $r_p =$ "*immediate read*" angegeben ist, darf die Transaktionsebene die Anfrage nicht bearbeiten. In Abhängigkeit der gewählten Strategie, wird dann die Anfrage durch die jeweiligen Services in den jeweiligen Ebenen bearbeitet. Wird ein Service aufgerufen, muss die Anfrage bearbeitet und an den nachfolgenden Services weitergereicht werden. Dies wird solange fortgeführt, bis am Ende das Plug-in durch den *Plug-in Manager* (PM) aufgerufen wurde (siehe Abbildung 8.2). Nachdem die Lese- oder Schreiboperation im Plug-in ausgeführt wurden, wird die Antwort, in umgekehrter Reihenfolge, an die jeweiligen Services geschickt. Am Ende gibt das DG das Ergebnis der Anfrage an *YAWL* zurück.

Integrationsebene

Die Integrationsebene (engl. *integration layer*) bindet Datenquellen mit Hilfe von Plug-ins ein. Dazu werden der PM und einen *Connection Pool* verwendet. Der PM verwaltet den Zugriff auf alle angemeldeten Plug-ins (siehe Abbildung 5.3). Hierfür nutzt der PM den Driver-Manager, um eine Verbindung für ein Plug-in zu erzeugen. Um die gleichzeitige Nutzung derselben Datenquelle zu ermöglichen, wird der *Connection Pool* eingeführt (siehe Abbildung 8.2). Er ermöglicht ein effektives Verwalten der einzelnen Plug-in-Verbindungen. Je nach Datenquelle wird hier die Anzahl der gleichzeitig nutzbaren Verbindungen kontrolliert. Dies ist besonders wichtig, wenn die Datenquellen oder das Plug-in keine eigenen Mechanismen dafür bereitstellen. Der Pool wird außerdem dafür benutzt, neue Verbindungen zu einem Plug-in zu erzeugen. Wird eine Verbindung zu einem Plug-in durch das WFMS angefragt, stellt der Pool entweder ein vorhandenes Objekt `PlgConnection` bereit, oder erzeugt ein neues. Hierfür wird die *pID* aus dem Mapping verwendet. Nach erfolgreicher Verarbeitung einer Anfrage, wird das Objekt an den Pool zurückgegeben und steht für die nächste Anfrage zur Verfügung.

Constraint Service

Sind im Prozessmodell Integritätsbedingungen modelliert, wird der *Constraint Service* eingebunden (siehe Abbildung 8.2). Wird eine Bedingung verletzt, muss an dieser Stelle eine geeignete Fehlerbehandlung stattfinden.

Weil Integritätsbedingungen insbesondere im Zusammenhang mit Transaktionen wichtig sind, führen wir eine Datenstruktur zum Verwalten ein. Das Konzept der transaktionalen Bereiche in einem Prozessmodell wird in Kapitel 6 vorgestellt. Unabhängig davon, werden die Integritätsbedingungen an ein Netz gebunden. Dafür führen wir den *Data Controller* (DC) ein. Ein Data Controller $DC = (E, C, n, s)$ ist definiert als:

- E ist eine Menge von externen Variablen $(e_1, e_2, \ldots, e_n)$.
- C ist eine Menge von dynamischen Integritätsbedingungen $(c_1, c_2, \ldots, c_n)$, die über E definiert sind.
- n ist das Prozessmodell, zu dem DC gehört.
- s ist der Status von DC mit $s \in \{active, inactive, deactivated\}$.

Der DC gehört zu einem Netz n und wird mit dem initialen Zustand *inactive* erzeugt. Der DC wird aktiviert ($s = active$), wenn das Netz n gestartet wird. Entsprechend wird der DC deaktiviert ($s = deactivated$), wenn das Netz beendet wird. Unabhängig von der Benutzung wird ein DC nicht dynamisch zur Laufzeit erzeugt. Dies kann bereits während der Instantziierung einer Prozessinstanz oder sogar wenn ein Prozessmodell in das WFMS geladen wird, stattfinden. Ist ein DC für eine externe Variable definiert, gilt also $extVar \in DC.E$, dann muss innerhalb des CS eine Überprüfung stattfinden. Die Anfrage ist dann gegenüber allen Integritätsbedingungen $c_i \in DC.C$ zu überprüfen.

Wird durch den CS die Verletzung einer Integritätsbedingung festgestellt, wird dies durch eine Ausnahmebehandlung bearbeitet. Hierfür verwenden wir den von *YAWL* standardmäßig bereitgestellten *Exlet*-Mechanismus (siehe Abschnitt 6.4.3.1). Mit Hilfe von *Exlets* können beispielsweise Fehler einzelner Aktivitäten oder Prozessinstanzen oder einer Menge von Prozessinstanzen, die zum gleichen Prozessmodell gehören, behandelt werden. Dafür bietet *YAWL* unterschiedliche Möglichkeiten an. Im Fehlerfall kann ein Element abgebrochen, angehalten, oder explizit beendet werden. Außerdem kann auch versucht werden, dass Element erneut zu starten.

Plug-in
Für die prototypische Implementierung des DAF wurden zwei Plug-ins implementiert. Ein Plug-in setzt den lesenden und schreiben Zugriff auf eine Oracle-Datenbank um. Intern werden Anfragen über das JDBC-Interface von Oracle gestellt. Ein zweites Plug-in ermöglicht den Zugriff auf ein Repository-System, das auf *MyCoRe*-basiert.

8.2.3 Transaktionale Erweiterung der Data Access Framework Architektur

Um den transaktionalen Datenzugriff vollständig zu unterstützen, wurde im DAF eine Transaktionsebene eingeführt. Die Transaktionsebene in Abbildung 8.2 enthält den Transaktions-Manager (*Transaction Manager*) und einen Recovery-Manager (*Recovery Manager*).

In [Mud12] wird hierfür eine prototypische Umsetzung der Transaktionsebene in Form eines Services vorgestellt, so dass ein transaktionaler Datenzugriff möglich ist. Dafür wird das DAF so erweitert, dass in Abhängigkeit vom Prozesszustand eine lokale Version der externen Variable bereitgestellt wird oder die Anfrage an den Plug-in-Manager weitergeleitet wird. Die lokalen Versionen der einzelnen Variablen werden in einem Cache verwaltet. Jeder Sphäre wird ein eigener Cache zugeordnet, der MVCC ermöglicht (siehe Abschnitt 6.4.1). Der Transaktions-Manager muss außerdem Schreibkonflikte auflösen, die während der Abarbeitung paralleler Sphären auftreten können (siehe Abschnitt 6.4.2.3). In [Mud12] wird dieses Problem nicht vollständig betrachtet. Es besteht zwar die Möglichkeit die Verzögerungen von Aktivitäten zu beschreiben, eine Umsetzung erfolgt aber nur in Form einer Log-Ausschrift.

8.2.4 *YAWL*-Editor mit *tx+YAWL*-Konzepten

Die Verwendung von externen Variablen und der damit eingeführten Konzepten, wird durch den *YAWL*-Editor standardmäßig nicht unterstützt. Deshalb haben wir den Editor so erweitert, dass externe Variablen verwendet werden können. Aktivitätsvariablen werden mit Hilfe von Ein- und Ausgabe-Parametern an eine externe Variable gebunden. Für die Umsetzung werden zwei Varianten zur Verfügung gestellt.

Variante 1 (Direkte Anbindung):
Die erste Variante ermöglicht die direkte Anbindung von Aktivitätsvariablen an eine externe Variable. Auflistung 8.1 zeigt den Eingabe-Parameter für die Aktivitätsvariable *x*. Die ersten beiden Segmente im Ausdruck *#external:DataAccessFrameworkGatewayImpl:author* definieren die Anbindung an das DAF. Das letzte Segment gibt den Namen der externen Variable an, die an die Aktivitätsvariable gebunden wird.

```
1 <startingMappings>
2  <mapping>
3   <expression>
4    query="#external:DataAccessFrameworkGatewayImpl:author" />
5   <mapsTo>x</mapsTo>
6  </mapping>
7 </startingMappings>
```

Auflistung 8.1: Eingabe-Parameter

Auflistung 8.2 zeigt den Ausgabe-Parameter für die Aktivitätsvariable *y*. Die ersten beiden Segmente im Ausdruck *#external:DataAccessFrameworkGatewayImpl:y* definieren die Anbindung an das DAF. Das letzte Segment gibt die Aktivitätsvariable an, die an die externe Variable *author* gebunden wird.

```
1 <completedMappings>
2  <mapping>
3   <expression query="#external:DataAccessFrameworkGatewayImpl:y" />
4   <mapsTo>author</mapsTo>
5  </mapping>
6 </completedMappings>
```

Auflistung 8.2: Ausgabe-Parameter

Abbildung 8.3 zeigt die Eingabemaske für Parameter im Editor. Derzeit ist eine direkte Umsetzung im Editor nur teilweise möglich. Die Zuweisung der externen Variable für die Parameter ist für diese Eingabemaske nicht umgesetzt. Sie muss händisch im *YAWL*-Modell vorgenommen werden. Die Definition externer Variablen wird für diese Variante unabhängig vom *YAWL*-Modell, in einer Konfigurationsdatei vorgenommen. Die Konfigurationsdatei enthält für jede externe Variable Angaben zum Mapping. Der *mappingKey* und das *mapping* beschreiben die Anfrage an das Plug-in. Mit dem Element *mappingKeyVal* kann der Wert für den *distKey* festgelegt werden (siehe Abschnitt 5.3.2). Das Element *pluginID* gibt die ID für das Plug-in an. Die beiden Elemente *readPolicy* und *writePolicy* geben entsprechend Abschnitt 5.3.3 die Strategie für den Lese- und Schreibzugriff vor.

```
1 #Oracle author XML Data Source
2 author_pluginID=XML_Oracle_ADT_Extract
3 author_readPolicy=readOnly
4 author_mapping=//author
5 author_mappingKey=authorDS
```

Auflistung 8.3: Definition für eine externe Variable

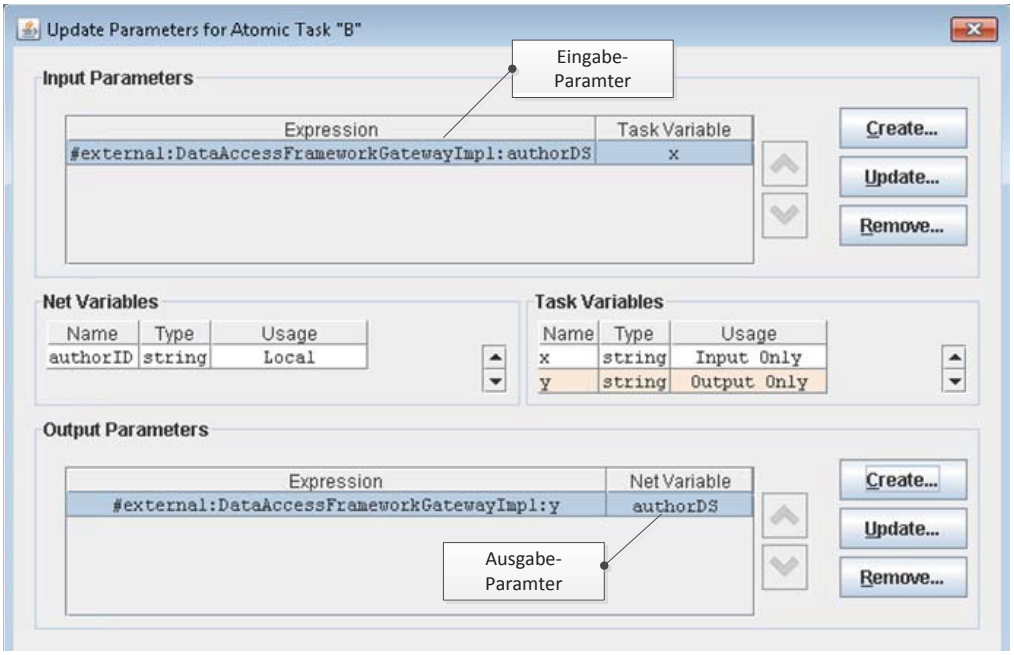


Abbildung 8.3: Parameter-Mapping im *YAWL*-Editor

Auflistung 8.3 zeigt die Definition für die externe Variable *author* (siehe Abschnitt 5.3.2).

Variante 2 (Erweiterte Attribute):
Die zweite Variante für die Verwendung von externen Variablen ermöglicht die vollständige Modellierung im *YAWL*-Editor. Dabei wird die Eingabemaske für Aktivitätsvariablen erweitert, was nicht vollständig den Anforderungen aus Kapitel 5 entspricht.

Für die Umsetzung nutzen wir die Eingabemasken für Aktivitätsvariablen, welche in Abbildung 8.4 gezeigt werden. Mit Hilfe von *Erweiterten Attributen* (engl. extended attributes) wird eine externe Variable direkt an eine Aktivitätsvariable gebunden. Dafür nutzen wir neu eingeführte Attribute, die in Abbildung 8.4 in der rechten Eingabemaske gezeigt werden. Das Attribut *VariableID* verweist dort auf die externe Variable *author*. Die Definition der externen Variable ist in diesem Fall in einer externen Konfigurationsdatei abgelegt, wie sie in Auflistung 8.3 dargestellt ist.

Diese Variante bietet außerdem die Möglichkeit, das Mapping für eine externe Datenquelle direkt an eine Aktivitätsvariable zu binden. Über die Attribute *mappingKey, mapping, mappingKeyVal, pluginID, readPolicy* und *writePolicy* kann das Mapping direkt angegeben werden. Auch für diese Variante müssen die Aktivitätsvariablen an das DAF gebunden werden. Hierfür wird ebenfalls die Eingabemaske für die Eingabe von Parametern verwendet. Im Gegensatz zu Variante 1 wird das Element *expression* im Parameter nur für die Angabe des DAF verwendet (*#external:DataAccessFrameworkGatewayImpl*). Die Angabe der externen Variable entfällt.

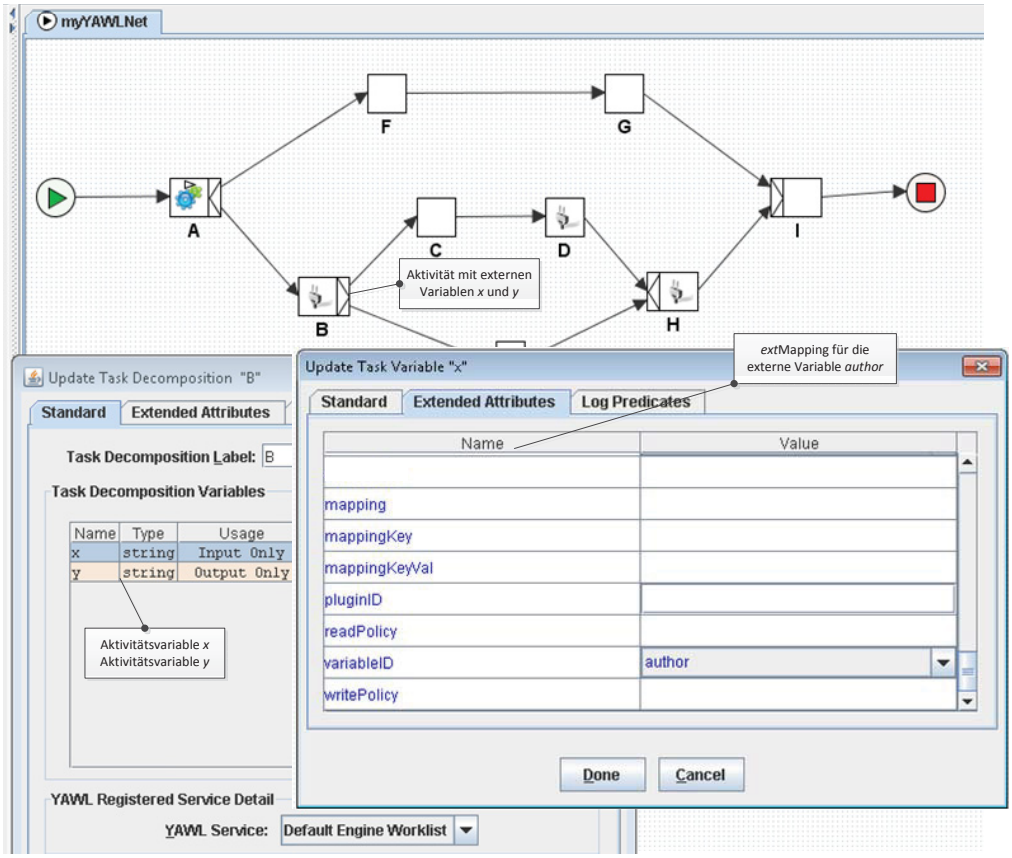


Abbildung 8.4: Ansicht Aktivitätsvariable im *YAWL*-Editor

Transaktionale Sphären

Die Modellierung von transaktionalen Sphären wird derzeit nicht durch den Editor unterstützt. Für die prototypische Umsetzung wird deshalb eine Konfigurationsdatei verwendet, die die Sphären definiert. Die Konfigurationsdatei ordnet außerdem jeder Sphäre die Aktivitäten zu, die in der Sphäre enthalten sind (siehe [Mud12]).

8.3 Implementierung von *FlexY*

Im Folgenden wird die prototypische Umsetzung des *FlexY*-Ansatzes vorgestellt. In Abschnitt 8.3.1 wird die Architektur vom *FlexY*-Service beschrieben. In Abschnitt 8.3.2 wird gezeigt, wie *FlexY* mit dem *YAWL*-Editor verwendet wird.

8.3.1 Der *FlexY*-Service

In Abschnitt 7 wurden Observer- und Generator-Aktivitäten eingeführt, die eine flexible Generierung von Prozessfragmenten ermöglichen. Der Prototyp soll *YAWL* so erweitern, dass

die neuen Aktivitätstypen ohne Einschränkungen in jedem Prozessmodell nutzbar sind. Die Erweiterungen sollen außerdem den Designkriterien des *YAWL*-Systems entsprechen.

Entwurfsentscheidung
YAWL bietet verschiedene Interfaces an, um die Funktionalität zu erweitern. Für die Umsetzung der Observer-Aktivitäten wird ein Mechanismus benötigt, der eine einfache Modellierung in Prozessmodellen ermöglicht und die Nutzung von externen Datenquellen unterstützt. Für Generator-Aktivitäten wird ein Mechanismus benötigt, der eine einfache Modellierung der Prozessmodelle ermöglicht, das Ausführen von Prozessfragmenten unterstützt und Daten über die Prozessinstanz sowie die Aktivitätsinstanz bereitstellen kann.

Für die prototypische Umsetzung wird das Konzept der *YAWL custom services* genutzt. Ein *custom service* ist ein Service, der mit *YAWL* über XML/HTTP Nachrichten kommunizieren kann. *YAWL* gibt die Kontrollen für die eigentliche Ausführung von Aktivitäten an diese Services ab, die von *YAWL* über die verschiedenen Interfaces eingebunden werden. Ein solcher Service muss das Interface B von *YAWL* implementieren. Dadurch kann ein *custom service* einfach in Prozessmodelle von *YAWL* eingebunden werden. Weiterhin stellt ein *custom service* alle Daten über die Prozessinstanz sowie die Aktivitätsinstanz bereit. So wird zum einen der Zugriff auf externe Datenquellen durch den Einsatz des DAF möglich, zum anderen können Generator-Aktivitäten Prozessdaten verwenden. In Kombination mit dem Interface A können *custom services* Prozessmodelle in *YAWL* hochladen.

Weil die Komponenten *Oberserver* und *Generator* stark voneinander abhängen, wird die Implementierung durch einen gemeinsamen Service bereitgestellt. Ein einzelner Service bietet die Vorteile, dass die Wartung erleichtert wird und die Integration in die Prozessmodelle einfacher möglich ist. Beide Komponenten werden deshalb durch den *custom service FlexY* bereitgestellt.

Der *FlexY*-Service wird in drei Hauptkomponenten unterteilt, die im Klassendiagramm in Abbildung 8.5 dargestellt sind. Die Klasse *FlexY* implementiert die beiden *YAWL*-Interfaces A und B. Über das Interface B kann *YAWL* den *FlexY*-Service zur Laufzeit einbinden. Dafür muss der Service zur Modellierungszeit mindestens einer Aktivität zugeordnet werden. Das Interface A wird verwendet, um die erzeugten Prozessmodelle *YAWL* bereitzustellen. Die Klasse *Generator* wird aufgerufen, wenn der Service als Generator im Prozessmodell eingebunden wird. Wird der Service als Observer eingebunden, wird die Klasse *Observer* aufgerufen.

Observer-Aktivitäten
Der *FlexY*-Service muss während der Ausführung von Prozessinstanzen den Zustand von Dokumenten bestimmen, um korrekte Prozessfragmente zu generieren. Die Klasse *Observer* setzt die Funktionalitäten um, die für die Überwachung von Dokumenten und deren Zuständen benötigt werden (siehe Abschnitt 7.3). In Abbildung 8.6 wird das Klassendiagramm für

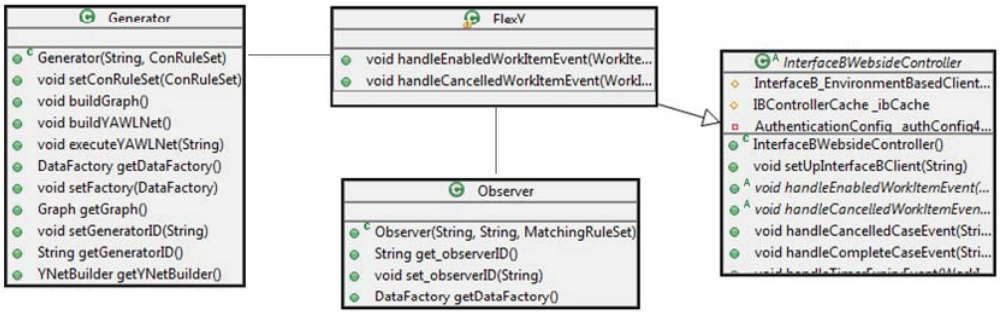


Abbildung 8.5: Klassendiagramm für Basiskomponenten des *FlexY*-Services

den Observer gezeigt.

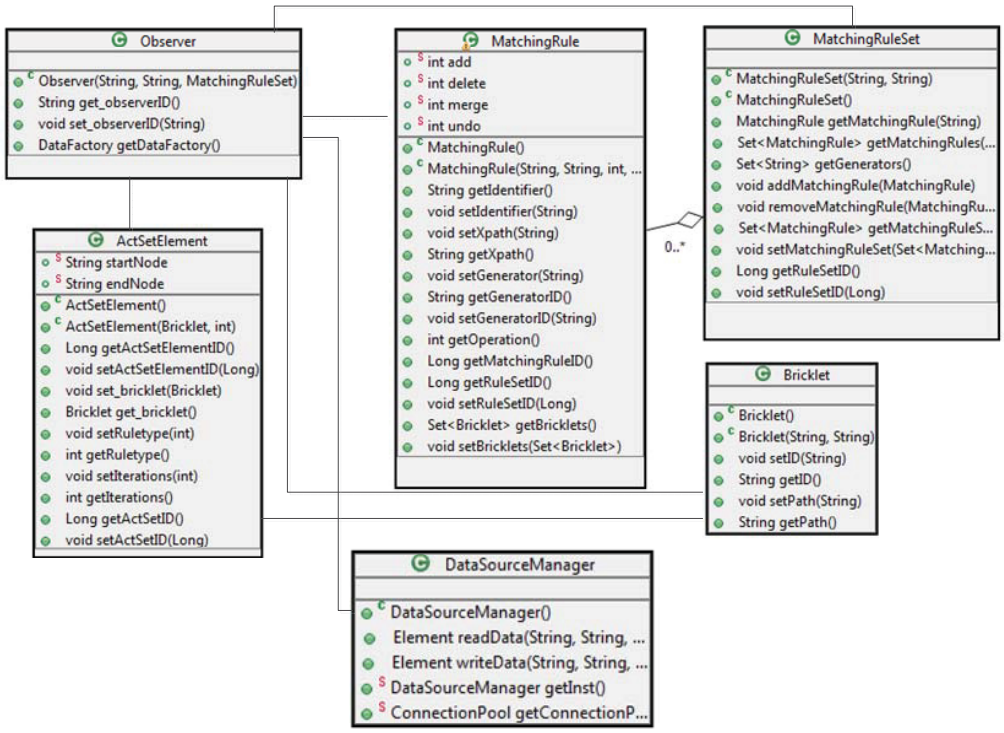


Abbildung 8.6: Klassendiagramm für die *FlexY*-Observer-Komponente

Wird der Observer aufgerufen, muss in einem ersten Schritt die Menge der Matching-Regeln geladen werden. Dafür muss für jedes Prozessmodell eine Konfigurationsdatei zur Modellierungszeit angelegt werden, die die Matching-Regeln für jede Observer-Aktivität im Prozessmodell enthält (in Abschnitt 8.3.2 wird die Struktur der Datei vorgestellt). Nachdem die Menge der Matching-Regeln für die Aktivität geladen wurde, die den Observer aufgerufen hat, werden die Matching-Regeln einzeln ausgewertet. Für die Auswertung der Matching-Regeln nutzt die Klasse *Observer* externe Variablen. Diese müssen im Prozessmodell vorher definiert und auf eine gleichnamige Aktivitätsvariable der Observer-Aktivität abgebildet

werden (siehe Abbildung 8.8). Eine automatische Generierung wird durch den Prototypen derzeit nicht unterstützt. Kann eine Regel angewendet werden, wird das Aktivierungsset des angegebenen Generators verändert (siehe Abschnitt 7.3.2). Dabei wird eine Menge von Bricklets in die Aktivierungsmenge des Generators eingefügt oder aus der Aktivierungsmenge entfernt.

Generator-Aktivitäten
Die flexible Generierung von Prozessfragmenten wird durch Generatoren umgesetzt. Im Prozessmodell können Aktivitäten den *FlexY*-Service als Generator aufrufen (siehe Abbildung 8.8). Dann wird die Klasse *Generator* vom Service verwendet, um aus den aktivierten Bricklets ein Prozessfragment zu erzeugen.

Im ersten Schritt wird durch die Klasse *GraphBuilder* ein Graph erzeugt, der als Zwischenschritt für die Prozessgenerierung dient. Dafür wurde der Algorithmus aus Auflistung 7.1 implementiert. Für die Generierung des Graphen wird die Aktivierungsmenge (*ActSet*) des Generators benötigt. Die Aktivierungsmenge wird in einer eigenen Relation der *YAWL*-Datenbank persistent gespeichert. Wir verwenden dafür die in *YAWL* genutzte Persistenzschicht[2].

Der erzeugte Graph wird an die Klasse *YNetBuilder* übergeben. Dort wird mit Hilfe der Bricklets ein *YAWL*-Netzt erzeugt. Dieses *YAWL*-Netz wird dann an die *FlexY*-Service-Klasse zurückgeliefert. Dort wird das Netz über die Methoden, die das Interface A bereitstellt, in *YAWL* hochgeladen und gestartet.

Die Netze der Bricklets werden im Ordner *repository* des *FlexY*-Services gespeichert. Sie müssen den gleichen Namen wie das Bricklet haben, damit sie korrekt geladen werden können.

8.3.2 *YAWL*-Editor mit *FlexY*-Konzepten

Der *FlexY*-Service kann direkt im *YAWL*-Editor verwendet werden. Im Editor können Observer-Aktivitäten und Generator-Aktivitäten modelliert werden, indem der *FlexY*-Service für eine atomare Aktivität als *custom service* ausgewählt wird (siehe Abbildung 8.8).

Modellierung im Editor
In Abbildung 8.8 wird am Beispiel für die Indizierung gezeigt, wie ein Prozessmodell mit dem *FlexY*-Ansatz modelliert wird. Die Aktivität *get Docs* wird als Generator modelliert. Deshalb muss die externe Variable *document* angegeben werden. Sie wird für die Auswertung der Matching-Regeln benötigt. In der Ansicht *Extended Attributes* muss dann noch der Typ ausgewählt werden. In diesem Fall ist der auszuwählende Typ *observer*. Im Feld *mappingFile* wird der Dateiname der Matching-Regel-Konfigurationsdatei angegeben.

[2] *YAWL* nutzt das Hibernate Framework für die persistente Speicherung.

Abbildung 8.7: Klassendiagramm für die *FlexY*-Generator-Komponente

Soll die Aktivität als Generator eingesetzt werden, muss in der Ansicht *Extended Attributes* der Typ *generator* ausgewählt werden. Im Feld *conRuleFile* muss der Dateiname der Konstruktionsregel-Konfigurationsdatei angegeben werden. Das Feld *mappingFile* muss in diesem Fall leer bleiben. Im Beispiel ist die Aktivität *featureExtraction* als Generator-Aktivität modelliert.

Konfigurationsdateien

Die Regelmengen für Observer- und Generator-Aktivitäten können nicht über den Editor modelliert werden. Dafür werden Konfigurationsdateien bereitgestellt, die die jeweiligen Regelmengen enthalten.

```
1  m_0_xpath=/course/image
2  m_0_bricklets=B6
3  m_0_operation=1
4  m_0_generatorID=featureExtraction
5  m_0_extVarID=document
6
7  m_1_xpath=/course/video
8  m_1_bricklets=B1,B2,B3,B4,B5
9  m_1_operation=1
10 m_1_generatorID=featureExtraction
11 m_1_extVarID=document
12
13 m_2_xpath=/course/text
14 m_2_bricklets=B4,B5
15 m_2_operation=1
16 m_2_generatorID=featureExtraction
17 m_2_extVarID=document
```

Auflistung 8.4: Konfigurationsdatei Observer

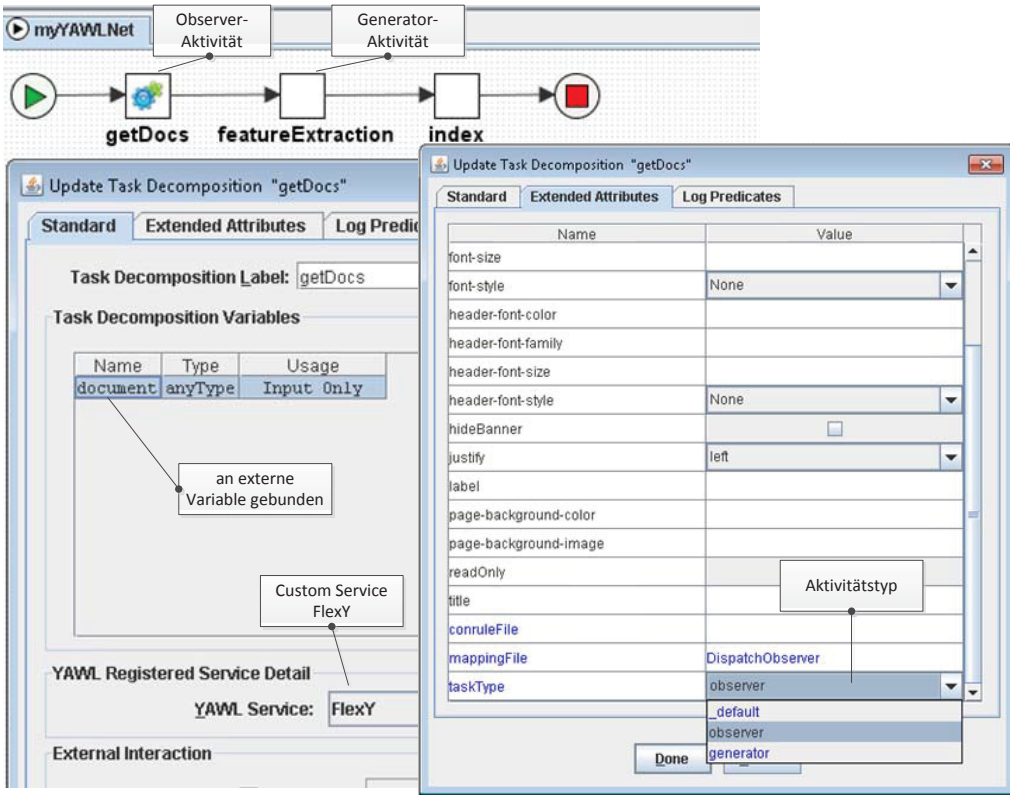


Abbildung 8.8: Observer-Aktivität im *YAWL*-Editor

In Auflistung 8.4 wird die Konfigurationsdatei für den Observer aus Abbildung 8.8 gezeigt. Sie enthält drei Regeln, die jeweils unterschiedliche Bricklets zum Generator *featureExtraction* hinzufügen (Attribut *generatorID*). Das Attribut *operation* gibt die Änderungsoperation der Matching-Regel an (siehe Abschnitt 7.3). Mit dem Attribut *bricklets* werden die hinzuzufügenden Bricklets definiert. Der *XPath*-Ausdruck wird im Attribut *xpath* abgelegt. Das Attribut *extVarID* gibt den Namen der externen Variable an, die für die Auswertung der Regel benutzt werden muss.

In Auflistung 8.5 wird die Konfigurationsdatei für den Generator gezeigt. Sie enthält mehrere Konstruktionsregeln, die vom Generator für die Generierung verwendet werden. Die Regeln sind dabei nach dem Muster aufgebaut, dass immer der Regeltyp, gefolgt von den beteiligten Bricklets, angegeben wird.

```
1 prop1=prec,B1,B2
2 prop2=prec,B3,B4
3 prop3=prec,B4,B5
4 prop4=seq,B1
5 prop5=par,B4
6 prop6=par,B6
```

Auflistung 8.5: Konfigurationsdatei Generator

Die Regeln können wie folgt dargestellt werden:

- *prop1* = *prec, B1, B2* entspricht der Regel $B_1 \prec B_2$
- *prop5* = *par, B4* entspricht der Regel $B_4 \overset{n}{\|}$
- *prop4* = *seq, B1* entspricht der Regel $B_1 \overset{n}{\prec}$

8.4 Zusammenfassung

Der Prototyp setzt die transaktionale Integration von externen Datenquellen und die flexible, datengetriebene Anpassung von Prozessen um. Die transaktionale Integration von externen Datenquellen wird durch das *DAF* implementiert, das direkt an *YAWL* angebunden wird. Die externen Datenquellen können über externe Variablen in Prozessmodelle eingebunden werden. Dafür wurde eine Erweiterung des *YAWL*-Editors vorgenommen. Flexible, datengetriebene Workflows werden durch den *FlexY*-Service unterstützt. Der Service überwacht den Zustand der Dokumente (Observer-Aktivitäten). Dafür werden externe Variablen verwendet, die durch das DAF bereitgestellt werden. In Abhängigkeit der Dokumentzustände werden Prozessinstanzen zur Laufzeit angepasst. Die Generator-Aktivitäten setzten dafür das Konzept des *late modeling* um.

Kapitel 9

Praktische Anwendungsszenarien

Im Folgenden werden zwei Projekte vorgestellt, in denen der Prototyp eingesetzt wurde.

Das Kapitel gliedert sich wie folgt: In Abschnitt 9.1 wird das Projekt *RosDok* vorgestellt, in dem in Zusammenarbeit mit der Universitätsbibliothek Rostock der Prototyp im Rahmen einer digitalen Bibliotheksanwendung eingesetzt wird. In Abschnitt 9.2 wird das Projekt *PERIKLES* vorgestellt. In diesem Projekt wurde der Prototyp in einem klinischen Assistenzsystem eingesetzt.

9.1 Publikationsprozess am Beispiel von MyCoRe

Am Beispiel des Repository-Systems MyCoRe wurde gezeigt, wie die transaktionale Integration von externen Datenquellen und flexible Prozessmodelle für den Publikationsprozess von Dissertationen angewendet werden können. Die Anwendung und die damit verfolgten Ziele werden in Abschnitt 9.1.1 vorgestellt. In Abschnitt 9.1.2 wird beschrieben, wie der Prototyp für diesen Anwendungsfall eingesetzt wurde.

9.1.1 MyCoRe: Digitale Bibliothek RosDok

Die Universitätsbibliothek Rostock stellt hochschuleigene Schriften, mit dem Dissertations- und Publikationsserver *RosDok*[1], digital zur Verfügung. Zu den bereitgestellten Inhalten zählen neben Dissertationen auch historische Sammlungen, Handschriften und Alte Drucke. *RosDok* ist ein Dokumenten Server, der auf dem modularen System *MyCoRe* aufsetzt. *MyCoRe* ist ein digitales Repository-System, auf dessen Basis nutzerdefinierte Anwendungen

[1] `http://rosdok.uni-rostock.de`

entwickelt werden können. Einen Überblick über *MyCoRe* wurde bereits in Abschnitt 1.3.1 gegeben.

In Zusammenarbeit mit der Universitätsbibliothek Rostock wurde beispielhaft gezeigt, wie der Prototyp den Publikationsprozess von Dissertationen unterstützen kann. Für die Umsetzung wurde das Anwendungsszenario „Dissertation Online“ gewählt, das in der Universitätsbibliothek auch im praktischen Einsatz ist. Ziel war es, zu zeigen wie eine beispielhafte Prozessunterstützung für den Publikationsprozess von Dissertationen unter Verwendung des vorgestellten Prototypen umgesetzt werden kann. Als Grundlage wurde der in Abschnitt 3.1 vorgestellte Publikationsprozess „Dissertation Online“ genutzt. Einzelheiten zum Projekt „Dissertation Online“ werden zusätzlich in Anhang B.1 gegeben.

9.1.2 Anwendung des Prototypen am Beispiel von *MyCoRe*

Die wesentlichen Aufgaben, die durch den digitalen Publikationsprozess von Dissertationen an der Universitätsbibliothek Rostock umgesetzt werden, beziehen sich auf die Bereitstellung der elektronischen Pflichtexemplare digitaler Dissertationen und Habilitationen. Dafür bietet das System einen Arbeitsablauf an, der in Abbildung 9.1 dargestellt ist und im Folgenden kurz erläutert wird.

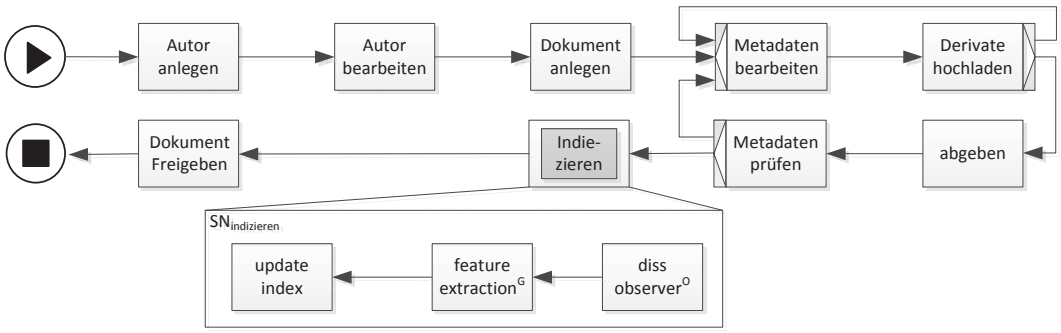


Abbildung 9.1: Beispielprozess für die Publikation von Dissertationen

- Zunächst muss der Autor einen Nutzerzugang für den Dokumentenserver beantragen (Aktivität: *Autor anlegen*). Dafür wird eine Eingabemaske bereitgestellt, die alle relevanten Informationen abfragt.
- In einem zweiten Schritt muss der Nutzer einen Autorendatensatz anlegen, der alle für die Publikation einer Dissertation notwendigen personenbezogenen Daten enthält (Aktivität: *Autor bearbeiten*).
- Danach kann der Nutzer den Abgabeprozess starten, indem die Dissertation angemeldet wird. In diesem Schritt wird eine URN erzeugt, die der Nutzer händisch in die Arbeit einfügen muss (Aktivität: *Dokument anlegen*).

- Außerdem müssen vom Nutzer bibliographische Daten zur Arbeit in ein Formular eingetragen werden (Aktivität: *Metadaten bearbeiten*). Dazu zählen beispielsweise Schlagwörter und die Angabe einer DDC-Kategorie.
- Eine generierte PDF/A1-b Datei kann dann zusammen mit zusätzlichen Dateien, wie z. B. Anhängen mit Messreihen oder Statistiken auf den Dokumentenserver geladen werden (Aktivität: *Derivate hochladen*).
- Am Ende des Prozesses wird ein Abgabeformular erzeugt, das zusammen mit den Ausgedruckten Pflichtexemplaren in der Universitätsbibliothek abzugeben ist (Aktivität: *abgeben*).
- In einem letzten manuellen Schritt wird die Arbeit von einem Bibliothekar überprüft und freigeschaltet (Aktivität: *Metadaten prüfen*) oder für eine Überarbeitung an den Autor übergeben.
- Dann wird die Arbeit in der komplexen Aktivität *Indizieren* für die Suche aufbereitet und indiziert. Dafür wird der *FlexY*-Ansatz verwendet (siehe Kapitel 7). Die Observer-Aktivität *diss observer* konfiguriert in Abhängigkeit des Dokumentes *dissDS* die Generator-Aktivität *feature extraction*. Die Prozessbausteine, die für die Indizierung genutzt werden können, werden in den Abschnitten 3.3.1 und 7 beschrieben. Anschließend wird das Dokument indiziert.

Auf eine weiterführende Analyse des Prozesses wird hier verzichtet, weil die Umsetzung von Publikationsprozessen mit dem hier vorgestellten Ansatz bereits in Kapitel 7 vorgestellt wurde (siehe auch Anhang B.4). Es konnte aber gezeigt werden, dass der Prototyp zusammen mit *MyCoRe* für die Umsetzung eines digitalen Publikationsprozesses sehr gut nutzbar ist.

Zunächst musste die Integration des WFMSs *YAWL* in *MyCoRe* gelöst werden. In enger Zusammenarbeit mit Entwicklern der Universitätsbibliothek wurde eine Lösungsvariante gewählt, die eine flexible Nutzung beider Systeme erlaubt. Um dem Anwender Arbeitslisten und Eingabemasken bereitzustellen, wurden das Web-Interface des Resource-Service von *YAWL* genutzt. Das Anlegen der Autorendatensätze und Dokumente wurde mit Hilfe von *MyCoRe*-Eingabemasken umgesetzt, die direkt von *YAWL* aufgerufen werden. So konnte eine nahtlose Kopplung beider Systeme erreicht werden.

In einer zweiten Phase wurde die flexible Komposition von Prozessfragmenten umgesetzt. Wie in Kapitel 7 beschrieben, benötigt das WFMS für die flexible Komposition von Prozessfragmenten Zugriff auf Dokumente, Metadaten und Dateien, die in *MyCoRe* verwaltet werden.

Die Bereitstellung gestaltete sich schwierig, weil die Dokumente und Daten von *MyCoRe* unterschiedlich verwaltet werden. Die Metadaten von Dokumenten und Derivaten werden in einer Datenbank verwaltet. Die Derivate werden in einem Dateisystem auf dem Server verwaltet. Der Aufwand für die Berechnung von Beziehungen zwischen den einzelnen Daten

ist sehr hoch. Deshalb wird für die Zuordnung zwischen den Dokumenten und den Derivaten das *MyCoRe*-System benötigt.

Aus diesem Grund musste ein eigenes *MyCoRe*-Plug-in für das DAF (siehe Kapitel 5 und 6) bereitgestellt werden, das den Zugriff auf *MyCoRe*-Datensätze innerhalb von *YAWL* ermöglicht.

Das *MyCoRe*-Plug-in ermöglicht mit Hilfe des DAF, beliebige *MyCoRe*-Datensätze in *YAWL*-Prozessen zu nutzen. Das *MyCoRe*-System musste hierfür mit einer eigenen Schnittstelle erweitert werden, die anhand von Dokumentidentifikatoren vollständige XML-Datensätze liefert.

Fazit
Unter Anwendung des *FlexY*-Ansatzes in Kombination mit *tx+YAWL*-Konzepten, konnte die flexible Komposition von Prozessfragmenten beispielhaft umgesetzt werden. An dieser Stelle setzt das *MyCoRe*-System Grenzen, weil verschiedene Funktionalitäten nicht unabhängig vom Rest der Anwendung nutzbar sind. Ein Beispiel dafür ist der Vorgang der Indizierung unterschiedlicher Informationen und die dafür benötigten Operationen. Das System bietet keine komponentenorientierte Architektur. Deshalb können bestimmte Operationen nicht losgelöst vom WFMS aufgerufen und gesteuert werden. Hier ist ein Eingriff in die grundlegende Architektur von *MyCoRe* notwendig, dessen Konzeption und Umsetzung jedoch nicht in dieser Arbeit betrachtet wurden. Unabhängig davon konnte eine erfolgreiche Nutzung des Prototypen in Kombination mit *MyCoRe* realisiert werden.

9.2 Perioperative Prozesse am Beispiel eines klinischen Assistenzsystems

Im Rahmen des Projektes *PERIKLES* wurde ein klinisches Assistenzsystem entwickelt, in dem unter anderem die Konzepte für die transaktionale Integration von externen Datenquellen und auch flexible Prozessmodelle angewendet wurden. Das Projekt und die damit verfolgten Ziele werden in Abschnitt 9.2.1 vorgestellt. In Abschnitt 9.2.2 wird beschrieben, wie der Prototyp für diesen Anwendungsfall eingesetzt wurde.

9.2.1 *PERIKLES*: Perioperative klinische Prozesse

In modernen Kliniken werden häufig mehr als 10 Operationssäle gleichzeitig nebeneinander betrieben. Im Rahmen des Projektes *PERIKLES*[2] wurde ein Assistenzsystem entwickelt,

[2]Der Projektname *PERIKLES* steht für „Unterstützung **peri**operativer klinischer Prozesse durch **k**ooperierende f**le**xible Workflows und AutoID- **S**ensorsysteme“.

das die Planung und Durchführung von perioperativen, klinischen Prozessen unterstützt. Der perioperative Prozess beschreibt alle klinischen Arbeitsschritte, die für die Planung, Durchführung und Nachbehandlung von chirurgischen Eingriffen notwendig sind.

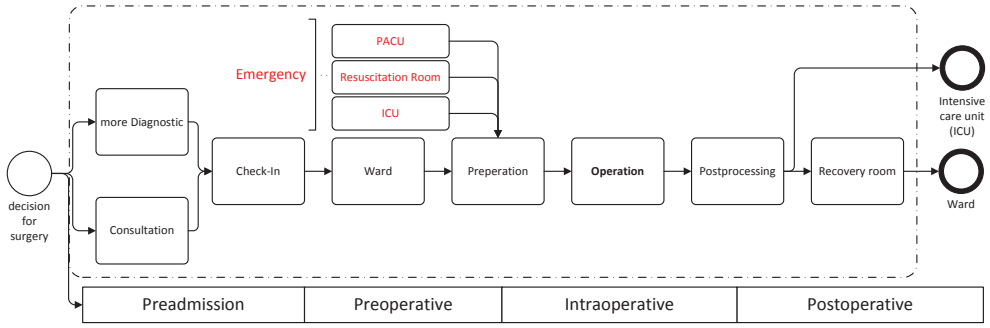


Abbildung 9.2: Perioperativer, klinischer Prozess nach [BKSM11]

In Abbildung 9.2 wird eine vereinfachte Darstellung der Phasen des perioperativen Prozesses gezeigt.

- Der Aufnahmeprozess (engl. *preadmission process*) beginnt mit der Entscheidung, ob eine Operation stattfindet. Der Schritt beinhaltet weiterhin eine präoperative und anästhesiologische Beratung und andere diagnostische Schritte.
- Die präoperativen Prozessschritte (engl. *preoperative process*) enthalten alle Aufgaben für die Planung der Operation und die Vorbereitung der Patienten.
- Der intraoperative Prozess (engl. *intraoperative process*) beschreibt die eigentliche Operation, die in der Regel im Operationssaal erfolgt. Nach der Operation wird der Patient entweder auf die Intensivstation oder in den Aufwachraum gebracht.
- Der postoperative Prozess (engl. *postoperative process*) umfasst die Betreuung der Patienten nach der Operation, bis diese das Krankenhaus verlassen.

Die Herausforderung bei der Umsetzung eines Werkzeugs für die Planung und Durchführung von perioperativen Prozessen liegt insbesondere in dem Management der Operationssäle. Für die Koordinierung der perioperativen Prozesse müssen die Verfügbarkeit von Personal, Operationssälen, vorhandenen Geräten und anderen Materialien beachtete werden. Die Koordinierung wird zusätzlich durch die Nebenläufigkeit und starke Änderungsdynamik der Arbeitsprozesse und einer Ressourcenknappheit erschwert. [KMFS09]

Die perioperativen Prozesse stellen besondere Anforderungen an die Ausführung der Prozesse und an die Integration externer Datenquellen (siehe auch [BKSM11] und [SMBH11]). Bei der Ausführung von Prozessen sind folgende Kontrollfluss-Muster von besonderem Interesse:

- Die partielle Ordnung von Aktivitäten.
- Die optionale Ausführung von Aktivitäten.
- Die mehrfache Ausführung von komplexen Subprozessen.
- Die alternative Ausführung von Aktivitäten.

Damit die Planung der perioperativen Prozesse korrekt durchgeführt werden kann, müssen Prozess- und Patientendaten proaktiv bereitgestellt werden. Das System muss dafür

- eine Integration heterogener Systeme ermöglichen.
- die transaktionale Integrität der Daten sicherstellen.
- die Daten für eine flexible Anpassung der Prozessmodelle bereitstellen.

Im folgenden Abschnitt wird gezeigt, wie diese Anforderungen mit Hilfe der hier vorgestellten Ansätze *tx+YAWL* und *FlexY* umgesetzt wurden.

9.2.2 Anwendung des Prototypen im Projekt Perikles

Anästhesie-Prozess

Perioperative Prozesse zeichnen sich durch ein hohes Maß an Änderungsdynamik aus. Durch eine flexible Ausnahmebehandlung muss auf unvorhergesehene Ereignissen reagiert werden. Unabhängig davon, sind die Prozesse aber durch einen fest vorgegeben Kontrollfluss gekennzeichnet. Am Beispiel des Anästhesie-Prozesses aus Abbildung 9.3 kann gezeigt

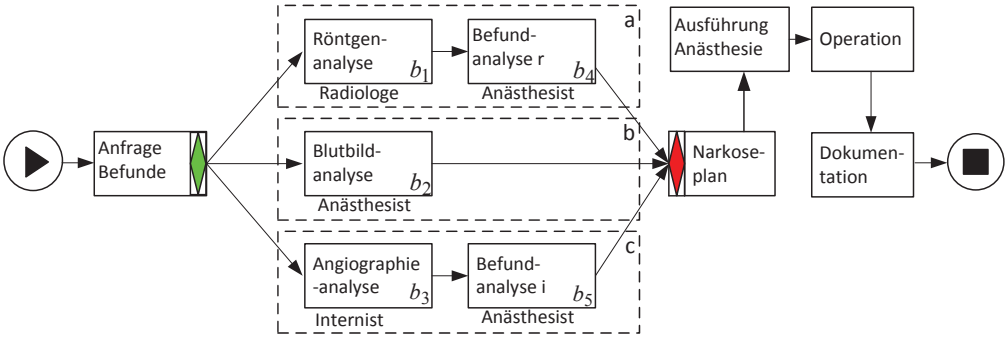


Abbildung 9.3: Beispiel-Prozessfragment für einen Anästhesie-Prozess

werden, welche Prozessbereiche fest vorgegeben sind und welche dynamisch an den Kontext angepasst werden müssen. Dieser Prozess ist Bestandteil des globalen Prozesses entsprechend Abbildung 9.2.

Der eigentliche Arbeitsablauf im Anästhesie-Prozess ist fest vorgegeben. In einem ersten Schritt müssen die Befunde des Patienten angefordert werden (Aktivität: *Anfrage Befunde*). Nachdem die Befunde ausgewertet wurden, kann die Anästhesie geplant und am Tag der Operation ausgeführt werden. Am Ende des Prozesses müssen alle Arbeitsabläufe dokumentiert werden. Die kontextabhängigen Prozessbestandteile ergeben sich bei der Auswertung von Befunden. In Abhängigkeit vom Zustand des Patienten, müssen verschiedene Befunde ausgewertet werden, um die Anästhesie zu planen und durchzuführen. Im Beispiel ist die Röntgenanalyse des Herzens (Aktivität b_1) und die Blutbildanalyse notwendig (Aktivität b_2). Die Angiographieanalyse (Aktivität b_3) ist dagegen ein Beispiel für einen optionalen Schritt.

In jedem Fall müssen für die Röntgenanalyse und die Angiographieanalyse die Befunde von Spezialisten begutachtet werden, bevor die Befunde an den Anästhesisten übergeben werden (Aktivitäten b_4 und b_5).

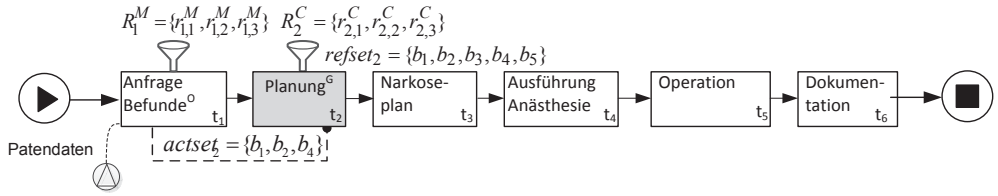


Abbildung 9.4: Anästhesie-Prozess mit *FlexY*-Ansatz umgesetzt

Modellierung

Diese Art von Prozessen können mit dem *FlexY*-Ansatz umgesetzt werden. Der Prozess kann als Basisprozess modelliert werden. Die datenabhängigen Bereiche sind in diesem Fall die Arbeitsabläufe für die Auswertung der Befunde. In Abbildung 9.4 wird das Prozessmodell, indem die Aktivität *Anfrage Befunde* als Observer-Aktivität modelliert ist, beschrieben. Zusätzlich wird die Aktivität *Planung* eingeführt, die als Generator-Aktivität umgesetzt ist. Alle anderen Aktivitäten gehören zum Basisprozess und können übernommen werden.

Die Auswertung der Befunde kann durch Bricklets beschrieben werden, die durch den Observer aktiviert und durch die Generator-Aktivität ausgeführt werden können. Im Beispiel definieren wir die Menge der Bricklets als $refset_2 = \{b_1, b_2, b_3, b_4, b_5\}$ und weisen sie der Generator-Aktivität zu. Die Bezeichnungen der Bricklets entsprechen denen aus Abbildung 9.3. Die Konstruktion der Bricklets kann im Beispiel durch die Menge der Konstruktionsregeln definiert werden: $R_1^C = \{b_1 \prec b_4, b_3 \prec b_5, b_1 \overset{n}{\prec}, b_2 \overset{n}{||}, b_3 \overset{n}{\prec}, b_5 \overset{n}{||}\}$. Für die Observer-Aktivität *Anfrage Befunde* wird die Menge von Matching-Regeln $R_1^M = \{r_{1,1}, r_{1,2}, r_{1,3}\}$ definiert:

- $r_{1,1}^M = add(t_2, b_2,$"`//OBR[../PID/PID.3/CX.1='123'][OBR.4/CWE.1='Blutbilder']`")
  Das Bricklet b_2 wird aktiviert, wenn ein Blutbild vorliegt.
- $r_{1,2}^M = add(t_2, \{b_1, b_4\},$ "`//OBR[../PID/PID.3/CX.1='123'][OBR.4/CWE.1='Roentgenbilder']`")
  Die Bricklets b_1 und b_4 werden aktiviert, wenn Röntgenbilder ausgewertet werden müssen.
- $r_{1,3}^M = add(t_2, \{b_3, b_5\},$ "`//OBR[../PID/PID.3/CX.1='123'][OBR.4/CWE.1='Angiographie']`")"
  Die Bricklets b_3 und b_5 werden aktiviert, wenn eine Angiographie ausgewertet werden muss.

So kann im Beispiel flexibel auf die Daten des Patienten reagiert werden.

Fazit

Die proaktive, transaktional abgesicherte Bereitstellung von Prozess- und Patientendaten kann mit Hilfe des DAFs umgesetzt werden. Im Rahmen des Projektes wurde ein System entwickelt, um materialisierte XML-Sichten auf ereignisorientierten Datenströmen bereitzustellen [Sch12]. Die so gewonnenen Daten können mit Hilfe eines speziell dafür entwickelten Plug-ins, den Prozessen zur Verfügung gestellt werden. Das Plug-in ermöglicht zum Beispiel die Integration von Patientendaten, die für die flexible Anpassung der Prozessinstanzen benötigt werden. In Abbildung 9.4 können so der Observer-Aktivität *Anfrage Befunde* die Patientendaten bereitgestellt werden, anhand derer die Konfiguration stattfinden kann. Die anderen Aktivitäten können über die externen Variablen ebenfalls Patientendaten lesen und schreiben. Um die Übersichtlichkeit im Beispiel zu verbessern, wurden diese Zugriffe nicht modelliert.

9.3 Zusammenfassung

Es wurden zwei Projekte vorgestellt, in denen der Prototyp erfolgreich eingesetzt wurde. Damit konnte gezeigt werden, dass die Konzepte neben der Umsetzung durch den Prototyp auch praktisch eingesetzt werden können.

In Zusammenarbeit mit der Universitätsbibliothek Rostock wurde das Repository-System *MyCoRe* durch eine Integration des Prototypen erweitert. Im Projekt *PERIKLES* konnten die vorgestellten Konzepte und der entwickelte Prototyp im perioperativen Bereich eingesetzt werden. Dabei wurde speziell die Prozessunterstützung für die Planung von perioperativen Prozessen behandelt.

Mit diesen beiden Projekten wird gezeigt, dass die Nutzung der Ansätze *tx+YAWL* und *FlexY* in unterschiedlichen Anwendungsgebieten möglich ist.

Teil IV

Fazit

Kapitel 10

Schlussbetrachtung

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und mögliche Erweiterungen des vorgestellten Ansatzes diskutiert.

10.1 Zusammenfassung

Um die komplexen Arbeitsabläufe in digitalen Bibliothekssystemen adäquat zu unterstützen, kommen zunehmend Workflow-Management-Systeme zum Einsatz. Dabei stellt die Einbeziehung multimedialer Dokumente aufgrund komplexer Dokumentstrukturen und der Vielzahl unterschiedlicher Dokumenttypen besondere Anforderungen an die Prozessmodelle und die genutzten WFMSe. Die Beziehungen zwischen den Dokumenten und den Prozessen können nicht unabhängig voneinander betrachtet werden.

Der Publikationsprozess muss den gesamten Lebenszyklus von Dokumenten berücksichtigen. Innerhalb der Prozesse müssen deshalb Methoden für den Datenzugriff auf externe Datenquellen bereitgestellt werden, da die Dokumente in externen, spezialisierten Medienservern abgelegt sind und auch durch andere Systeme verwendet werden. Die Möglichkeiten, die durch aktuelle Systeme bereitgestellt werden, sind für den Publikationsprozess nicht ausreichend. Vorhandene Ansätze unterstützen den Anwender nicht ausreichend bei der Synchronisation von lokalen Dokumentkopien mit den externen Datenquellen.

Die Herausforderung besteht deshalb darin, den Anwender bei der Integration unterschiedlicher, externer Datenquellen durch einheitliche Schnittstellen und Modellierungskonzepte umfassend zu unterstützen. Nur so können Inkonsistenzen zwischen redundant verwalteten Dokumenten vermieden werden. Wenn die Dokumente bereits während der Abarbeitung von Prozessinstanzen anderen Anwendungen bereitgestellt werden müssen, darf der Datenaustausch mit externen Systemen nicht losgelöst von den konkurrierenden Systemen und

Prozessen betrachtet werden. Neben einer einheitlichen Methode für die Datenintegration, muss das WFMS deshalb auch den Datenaustausch kontrollieren und transaktionale Eigenschaften wie z. B. Rücksetzbarkeit, Konsistenz und Dauerhaftigkeit innerhalb von Prozessen garantieren. Dafür müssen transaktionale Konzepte wie Atomarität und Isolation an die gegebenen Anforderungen angepasst werden.

Der Zusammenhang zwischen Dokumentmodellen und Prozessmodellen ist auch bei der Ausführung von Prozessen zu beachten. Aktuelle Ansätze bieten bereits eine Vielzahl unterschiedlicher Flexibilisierungskonzepte an, die es z. B. ermöglichen flexibel auf Ausnahmesituationen zu reagieren, indem Prozessinstanzen angepasst oder Prozessmodelle kontextabhängig konfiguriert werden. Für den Publikationsprozess sind diese Ansätze nur bedingt anwendbar, weil sich Dokumentstruktur und -inhalt häufig zur Laufzeit ändern. Die Dokumente haben, in Abhängigkeit vom Zustand des Prozesses, Einfluss auf den weiteren Prozessverlauf. Um Prozessinstanzen zur Laufzeit flexibel auf veränderte Dokumentstrukturen und -inhalte anzupassen, werden Strategien benötigt, die die Beziehungen zwischen Prozessmodellen und Dokumenten beschreiben.

Im Rahmen dieser Arbeit wurde deshalb ein Framework entwickelt, das den transaktionalen Zugriff auf externe Datenquellen und die flexible Konfiguration von Prozessinstanzen ermöglicht. Das vorgestellte Lösungskonzept wird im Folgenden zusammengefasst.

Einheitlicher Zugriff auf externe Datenquellen. In Kapitel 5 wurde die Entwicklung einer Erweiterung beschrieben, die die einheitliche Integration externer Datenquellen innerhalb von Prozessen ermöglicht. Dafür wurde das Konzept der *externen Variable* eingeführt, mit deren Hilfe externe Datenquellen innerhalb der Kontrollflussperspektive den Ein-/Ausgabeparameter von Aktivitäten bereitgestellt werden können. Ein einheitlicher Zugriff auf unterschiedliche Datenquellen wird durch das eigens dafür entwickelte *Data Access Framework* ermöglicht. Die verwendete Plug-in-Architektur ermöglicht die Kapselung der Datenquellen und stellt Schnittstellen für den Datenaustausch bereit.

Ein Transaktionsmodell für Prozessmodelle. In Kapitel 6 wurde das Mehrebenen-Transaktionsmodell *tx+YAWL* vorgestellt, das transaktionale Eigenschaften hinsichtlich der integrierten externen Variablen in einem Prozess sicherstellt. Um ACID-Eigenschaften zu erreichen, wurde ein 4-Ebenen-Transaktionsmodell eingeführt. Dafür wurde die Kontrollflussperspektive von *YAWL* mit transaktionalen Konzepten erweitert. Lese- und Schreiboperation auf externen Variablen stellen die atomaren Schritte einer Transaktion dar. Die Isolation paralleler Sphären wurde durch die Einführung eines lokalen Arbeitsbereiches erreicht, der das Konzept des Mehrversionen-Concurrency-Control umsetzt. Es wurden unterschiedliche Transaktionstypen eingeführt, um einerseits die Atomarität für Transaktionen sicherzustellen und andererseits auf Abbrüche von Aktivitäten und Teiltransaktionen flexibler reagieren zu können.

Mit Hilfe externer Variablen und dem vorgestellten Transaktionsmodell kann das WFMS gezielt Inkonsistenzen bei der redundanten Datenhaltung vermeiden.

Dynamische Publikationsprozesse. Neben der transaktionalen Integration von externen Datenquellen wurde in Kapitel 7 ein Konzept für die flexible, datengetriebene Anpassung von Prozessinstanzen vorgestellt. Der verwendete *FlexY*-Ansatz ermöglicht eine dynamische Konstruktion von Prozessbereichen, die in Abhängigkeit vom Dokumentzustand und -inhalt zur Laufzeit vorgenommen wird. Dies wurde erreicht, indem eine Unterteilung in anwendungsabhängige Prozessbestandteile (den Basisprozess) und dokumentabhängige Prozessbestandteile vorgenommen wurde. Der Basisprozess bildet im Wesentlichen den Geschäftsprozess der digitalen Bibliotheksanwendung ab. Die dokumentabhängigen Prozessbestandteile wurden deshalb in Form von Prozessbausteinen (*Bricklets*) bereitgestellt. Bricklets können in Abhängigkeit von den bearbeiteten Dokumenten im Basisprozess integriert werden. Hierfür wurden im Basisprozess Bereiche eingeführt (*Generatoren*), die für die dynamische Komposition und Ausführung von Bricklets genutzt werden können. Die Auswahl notwendiger Bricklets ist direkt vom Zustand der Dokumente und dem aktuellen Zustand der Prozessinstanz abhängig. Um eine geeignete Auswahl von Bricklets vornehmen zu können, wurden spezielle Aktivitäten (*Observer*) implementiert, die den aktuellen Zustand der Dokumente ermitteln und abhängig davon Bricklets aktivieren.

Validierung der Konzepte. Es wurde gezeigt, dass die vorgestellten Konzepte in verschiedenen Anwendungsgebieten praktisch eingesetzt werden können. Neben Publikationsprozessen in digitalen Bibliotheksanwendungen eigenen sich die diskutierten Konzepte z. B. auch für den Einsatz in klinischen Umgebungen. Die Umsetzbarkeit des vorgestellten Ansatzes wurde durch die Implementierung von prototypischen Anwendungen gezeigt.

10.2 Ausblick

Die Auseinandersetzung mit transaktionalen Konzepten für die Datenintegration sowie die flexible, datengetriebene Anpassung von Prozessen haben über die Zielstellung der vorliegenden Arbeit hinausgehende Fragestellungen aufgeworfen, die in weiterführenden Arbeiten betrachtet werden sollten.

Die vorgestellte Plug-in-Architektur bietet einen Rahmen für die manuelle Integration unterschiedlicher Datenquellen. Eine Herausforderung stellt die automatische Integration von Datenquellen dar. Der Nutzer kann bei der Erstellung eigener Plug-ins unterstützt werden, indem z. B. Techniken zum Generieren von einheitlichen XML-Sichten bereitgestellt werden, die auch Änderungsoperationen unterstützen. Der Aufwand für die Integration neuer Datenquellen kann so erheblich verringert werden.

Bei der Umsetzung von *tx+YAWL* wurde die Behandlung von Konflikten auf speziell, für Publikationsprozesse sinnvolle, Prozessstrukturen eingeschränkt. Die Auswertungen von beliebigen Prozessmodellen hinsichtlich möglicher Konflikte war nicht Ziel dieser Arbeit. Damit die vorgestellten Konzepte auch in anderen Anwendungsgebieten uneingeschränkt nutzbar sind, muss eine allgemeine Erweiterung auf komplexe Prozessstrukturen vorgenommen werden. Das kann z. B. unterstützt werden, indem der vorgestellte Mehrversionen-Concurrency-Control-Ansatz mit Techniken erweitert wird, die die auftretenden Anomalien der Snapshot-Isolation vermeiden. Als Ausgangspunkt kann der in [CRF09] vorgestellte Ansatz genutzt werden, der serialisierbare Isolation für Snapshot-Isolation in Datenbanksystemen ermöglicht.

Für den vorgestellten Anwendungsfall ist ein Transaktionsmodell notwendig, das transaktionale Eigenschaften für den verteilten Datenaustausch mit Datenquellen sicherstellt. Eine Integration von Transaktions-Infrastrukturen wie z. B. der *Java Transaction API* (JTA) oder des *CORBA Transaction Service* kann in bestimmten Anwendungsszenarien sinnvoll sein, um eine bessere Integration bzw. Kommunikation mit vorhandenen transaktionalen Ressourcen zu ermöglichen und globale Rücksetzbarkeit zu erreichen. In dieser Arbeit wurden Vorarbeiten für die Umsetzung eines geeigneten Recovery-Konzeptes geleistet, die als Ausgangspunkt für weitere Arbeiten genutzt werden können.

Die Modellierung von transaktionalen Sphären innerhalb der Kontrollflussperspektive von *YAWL* muss durch geeignete Modellierungskonzepte untersetzt werden. Neben neuen Modellierungskonstrukten ist zukünftig auch die Notwendigkeit für die Einführung einer neuen Perspektive für die Modellierung zu untersuchen.

Der vorgestellte Ansatz für die dynamische Komposition von Bricklets nutzt eine Regelmenge, die für den Einsatz in digitalen Bibliotheksanwendungen ausgelegt ist. Es hat sich gezeigt, dass Aktivierungsabhängigkeiten und Datenabhängigkeiten zwischen Bricklets existieren, die derzeit nur manuell über die genutzte Regelmenge abgebildet werden können. Die Eingabevariablen eines Bricklets können z. B. direkt von den Ausgabevariablen eines anderen Bricklets abhängen. Der Nutzer sollte hier durch das automatische Ableiten von Konstruktionsregeln unterstützt werden. Die Herausforderung dabei besteht darin, Methoden für die Analyse der Datenabhängigkeiten zu definieren und diese in Form von Konstruktionsregeln umzusetzen. So kann die Anzahl der manuell zu erzeugenden Regeln reduziert werden und auf mögliche Fehlersituationen früher hingewiesen werden. Wie die rekursive Komposition von Prozessfragmenten genutzt werden kann, ist eine weitere Fragestellung die in weiterführenden Arbeiten betrachtet werden kann. Wenn eine Generator-Aktivität wiederum Generatoren und Observer enthält, ist die Terminierung des Kompositionsprozesses nicht sichergestellt. Außerdem hat die rekursive Definition von Generator-Aktivitäten Einfluss auf Datenabhängigkeiten innerhalb der generierten Prozessfragmente.

Die im Rahmen dieser Arbeit entwickelten Ansätze *tx+YAWL* und *FlexY* realisieren bereits eine weitreichende Unterstützung bei der Umsetzung flexibler, datengetriebener Workflows für verschiedene Anwendungen, wie digitale Bibliotheken oder klinische Umgebungen. Mit den abschließend diskutierten weiterführenden Konzepten könnten Nutzer darüber hinaus noch zielgerichteter bei der Modellierung der Prozesse unterstützt und weitere Anwendungsbereiche erschlossen werden.

Literaturverzeichnis

[AAH+09] AALST, Wil M. P. d. ; ADAMS, Michael ; HOFSTEDE, Arthur H. M. ; PESIC, Maja ; SCHONENBERG, Helen: Flexibility as a Service. In: CHEN, Lei (Hrsg.) ; LIU, Chengfei (Hrsg.) ; LIU, Qing (Hrsg.) ; DENG, Ke (Hrsg.): *DASFAA Workshops* Bd. 5667, Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–04204–1, S. 319–333

[ABEW00] AALST, Wil M. P. d. ; BARTHELMESS, Paulo ; ELLIS, Clarence A. ; WAINER, Jacques: Workflow Modeling Using Proclets. In: ETZION, Opher (Hrsg.) ; SCHEUERMANN, Peter (Hrsg.): *CoopIS* Bd. 1901, Springer, 2000 (Lecture Notes in Computer Science). – ISBN 3–540–41021–X, S. 198–209

[ABEW01] AALST, Wil M. P. d. ; BARTHELMESS, Paulo ; ELLIS, Clarence A. ; WAINER, Jacques: Proclets: A Framework for Lightweight Interacting Workflow Processes. In: *Int. J. Cooperative Inf. Syst.* 10 (2001), Nr. 4, 443-481. http://ejournals.wspc.com.sg/journals/ijcis/10/1004/S0218843001000412.html

[ADR10] ADAMS, Michael ; DUMAS, Marlon ; ROSA, Marcello L.: The Architecture. Version: 2010. http://dx.doi.org/10.1007/978-3-642-03121-2_7. In: HOFSTEDE, Arthur H. M. (Hrsg.) ; AALST, Wil M. P. (Hrsg.) ; ADAMS, Michael (Hrsg.) ; RUSSELL, Nick (Hrsg.): *Modern Business Process Automation*. Springer Berlin Heidelberg, 2010. – DOI 10.1007/978–3–642–03121–2_7. – ISBN 978–3–642–03121–2, S. 205–220

[AH05] AALST, Wil M. P. d. ; HOFSTEDE, Arthur H. M.: YAWL: Yet another workflow language. In: *Information Systems* 30 (2005), Nr. 4, S. 245–275

[AHAE07] ADAMS, Michael ; HOFSTEDE, Arthur H. M. ; AALST, Wil M. P. d. ; EDMOND, David: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: [MT07], S. 95–112

[AHEA06a] ADAMS, Michael ; HOFSTEDE, Arthur H. M. ; EDMOND, David ; AALST, Wil M. P. d.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility

in Workflows. In: [MT06], S. 291–308

[AHEA06b] ADAMS, Michael ; HOFSTEDE, Arthur H. M. ; EDMOND, David ; AALST, Wil M. P. d.: Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In: [MT06], S. 291–308

[AHST97] ALONSO, Gustavo ; HAGEN, Claus ; SCHEK, Hans-Jörg ; TRESCH, Markus: Distributed Processing over Stand-alone Systems and Applications. In: JARKE, Matthias (Hrsg.) ; CAREY, Michael J. (Hrsg.) ; DITTRICH, Klaus R. (Hrsg.) ; LOCHOVSKY, Frederick H. (Hrsg.) ; LOUCOPOULOS, Pericles (Hrsg.) ; JEUSFELD, Manfred A. (Hrsg.): *VLDB*, Morgan Kaufmann, 1997. – ISBN 1–55860–470–7, S. 575–579

[Alo97] ALONSO, Gustavo: Processes + Transactions = Distributed Applications. In: *In Proceedings of the High Performance Transaction Processing (HPTS) Workshop at Asilomar*, 1997, S. 14–17

[AMR09] AALST, Wil M. P. d. ; MANS, R. S. ; RUSSELL, Nick C.: Workflow Support Using Proclets: Divide, Interact, and Conquer. In: *IEEE Data Eng. Bull.* 32 (2009), Nr. 3, S. 16–22

[APS09] AALST, Wil M. P. d. ; PESIC, Maja ; SCHONENBERG, Helen: Declarative workflows: Balancing between flexibility and support. In: *Computer Science - R&D* 23 (2009), Nr. 2, S. 99–113. http://dx.doi.org/10.1007/s00450-009-0057-9. – DOI 10.1007/s00450–009–0057–9

[AWG05] AALST, Wil M. P. d. ; WESKE, Mathias ; GRÜNBAUER, Dolf: Case handling: a new paradigm for business process support. In: *Data Knowl. Eng.* 53 (2005), Nr. 2, S. 129–162. http://dx.doi.org/10.1016/j.datak.2004.07.003. – DOI 10.1016/j.datak.2004.07.003

[AWH92] AIKEN, Alexander ; WIDOM, Jennifer ; HELLERSTEIN, Joseph M.: Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism. In: STONEBRAKER, Michael (Hrsg.): *SIGMOD Conference*, 1992, S. 59–68

[BBDW05] BUCHANAN, George ; BAINBRIDGE, David ; DON, Katherine J. ; WITTEN, Ian H.: A new framework for building digital library collections. In: MARLINO, Mary (Hrsg.) ; SUMNER, Tamara (Hrsg.) ; III, Frank M. S. (Hrsg.): *JCDL*, ACM, 2005. – ISBN 1–58113–876–8, S. 23–31

[BBG+95] BERENSON, Hal ; BERNSTEIN, Philip A. ; GRAY, Jim ; MELTON, Jim ; O'NEIL, Elizabeth J. ; O'NEIL, Patrick E.: A Critique of ANSI SQL Isolation

Levels. In: CAREY, Michael J. (Hrsg.) ; SCHNEIDER, Donovan A. (Hrsg.): *SIGMOD Conference*, ACM Press, 1995, S. 1–10

[BGVO05] BILASCO, Ioan M. ; GENSEL, Jérôme ; VILLANOVA-OLIVER, Marlène: STAMP: A Model for Generating Adaptable Multimedia Presentations. In: *Multimedia Tools Appl.* 25 (2005), Nr. 3, S. 361–375

[Bjo05] BJORK, Bo-Christer: A lifecycle model of the scientific communication process. In: *Learned Publishing* 18 (July 2005), 165-176(12). http://dx.doi.org/doi:10.1087/0953151054636129. – DOI doi:10.1087/0953151054636129

[BJVW10] BUCHMANN, Thomas ; JABLONSKI, Stefan ; VOLZ, Bernhard ; WESTFECHTEL, Bernhard: Towards a Generic Infrastructure for Sustainable Management of Quality Controlled Primary Data. In: *OTM Workshops*, 2010, S. 130–138

[BK01] BOLL, Susanne ; KLAS, Wolfgang: ZYX-A Multimedia Document Model for Reuse and Adaptation of Multimedia Content. In: *IEEE Trans. Knowl. Data Eng.* 13 (2001), Nr. 3, S. 361–382

[BKLW99] BUSSE, Susanne ; KUTSCHE, Ralf-Detlef ; LESER, Ulf ; WEBER, Herbert: Federated Information Systems: Concepts, Terminology and Architectures. 1999. – Forschungsbericht

[BKSM11] BANDT, Markus ; KÜHN, Robert ; SCHICK, Sebastian ; MEYER, Holger: Beyond Flexibility - Workflows in the perioperative Sector of the Healthcare Domain. In: *Electronic Communications of the EASST* 37 (2011), 146-157. http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/530

[BMR+07] BRETTLECKER, Gert ; MILANO, Diego ; RANALDI, Paola ; SCHEK, Hans-Jörg ; SCHULDT, Heiko ; SPRINGMANN, Michael: ISIS and OSIRIS: A Process-Based Digital Library Application on Top of a Distributed Process Support Middleware. In: *DELOS Conference* Bd. 4877, Springer, 2007 (Lecture Notes in Computer Science), S. 46–55

[BOWN06] BAINBRIDGE, David ; OSBORN, Wendy ; WITTEN, Ian ; NICHOLS, David: Extending Greenstone for Institutional Repositories. In: *Digital Libraries: Achievements, Challenges and Opportunities* Bd. 4312. Springer Berlin / Heidelberg, 2006, S. 303–312

[BPM11] *Business Process Modeling Notation (BPMN) Version 2.0.* 2011

[BSW88] BEERI, Catriel ; SCHEK, Hans-Jörg ; WEIKUM, Gerhard: Multi-Level Transaction Management, Theoretical Art or Practical Need ? In: SCHMIDT,

Joachim W. (Hrsg.) ; CERI, Stefano (Hrsg.) ; MISSIKOFF, Michele (Hrsg.): *EDBT* Bd. 303, Springer, 1988 (Lecture Notes in Computer Science). – ISBN 3–540–19074–0, S. 134–154

[CCL+06] CANDELA, L. ; CASTELLI, D. ; LANGGUTH, C. ; PAGANO, P. ; SCHULDT, H. ; SIMI, M. ; VOICU, L.: On-Demand Service Deployment and Process Support in e-Science DLs: the DILIGENT Experience. In: *ECDL Workshop DLSci06: Digital Library Goes e-Science*, 2006, S. 37–51

[CCPP98] CASATI, Fabio ; CERI, Stefano ; PERNICI, Barbara ; POZZI, Giuseppe: Workflow Evolution. In: *Data Knowl. Eng.* 24 (1998), Nr. 3, S. 211–238. http://dx.doi.org/10.1016/s0169-023x(97)00033-5. – DOI 10.1016/s0169–023x(97)00033–5

[CCPS05] CANDELA, Leonardo ; CASTELLI, Donatella ; PAGANO, Pasquale ; SIMI, Manuele: From Heterogeneous Information Spaces to Virtual Documents. In: FOX, Edward A. (Hrsg.) ; NEUHOLD, Erich J. (Hrsg.) ; PREMSMIT, Pimrumpai (Hrsg.) ; WUWONGSE, Vilas (Hrsg.): *ICADL* Bd. 3815, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–30850–4, S. 11–22

[CRF09] CAHILL, Michael J. ; RÖHM, Uwe ; FEKETE, Alan D.: Serializable isolation for snapshot databases. In: *ACM Trans. Database Syst.* 34 (2009), Nr. 4. http://dx.doi.org/10.1145/1620585.1620587. – DOI 10.1145/1620585.1620587

[Dav78] DAVIES, Charles T.: Data Processing Spheres of Control. In: *IBM Systems Journal* 17 (1978), Nr. 2, S. 179–198

[Deu11] DEUTSCHE NATIONALBIBLIOTHEK: *DissOnline.de*. Webseite: www.dissonline.de, 2011. – Zugriff am 01.05.2011

[DYWL04] DENG, ShuiGuang ; YU, Zhen ; WU, Zhaohui ; LICAN, Huang: Enhancement of workflow flexibility by composing activities at run-time. In: HADDAD, Hisham (Hrsg.) ; OMICINI, Andrea (Hrsg.) ; WAINWRIGHT, Roger L. (Hrsg.) ; LIEBROCK, Lorie M. (Hrsg.): *SAC*, ACM, 2004. – ISBN 1–58113–812–1, S. 667–673

[EDGB+12] EKANAYAKE, C. C. ; DUMAS, Marlon ; GARCÍA-BAÑUELOS, Luciano ; ROSA, Marcello L. ; HOFSTEDE, A. H. M.: Approximate clone detection in repositories of business process models. In: *BPM*, 2012, S. 1–16. – http://eprints.qut.edu.au/49409/1/approxclones_main.pdf, Accepted for Publication

[EF00] ENDRES, Albert ; FELLNER, Dieter W.: *Digitale Bibliotheken - Informatik-Lösungen für globale Wissensmärkte*. dpunkt, 2000. – ISBN 3–932588–77–0

[EGL98] EDER, Johann ; GROISS, Herbert ; LIEBHART, Walter: The Workflow Management System Panta Rhei. Version: 1998. http://dx.doi.org/10.1007/978-3-642-58908-9_7. In: DOĞAÇ, Asuman (Hrsg.) ; KALINICHENKO, Leonid (Hrsg.) ; ÖSZU, M. T. (Hrsg.) ; SHETH, Amit (Hrsg.): *Workflow Management Systems and Interoperability* Bd. 164. Springer Berlin Heidelberg, 1998. – DOI 10.1007/978–3–642–58908–9_7. – ISBN 978–3–642–58908–9, S. 129–144

[EL95] EDER, Johann ; LIEBHART, Walter: The Workflow Activity Model WAMO. In: *CoopIS*, 1995, S. 87–98

[EL96] EDER, Johann ; LIEBHART, Walter: Workflow Recovery. In: *CoopIS*, 1996, S. 124–134

[EL04] EDER, Johann ; LEHMANN, Marek: Uniform Access to Data in Workflows. In: *EC-Web* Bd. 3182, Springer, 2004 (Lecture Notes in Computer Science), 66-75

[EL05a] EDER, Johann ; LEHMANN, Marek: Synchronizing Copies of External Data in Workflow Management Systems. In: *CAiSE* Bd. 3520, Springer, 2005 (Lecture Notes in Computer Science), S. 248–261

[EL05b] EDER, Johann ; LEHMANN, Marek: Workflow Data Guards. In: *OTM Conferences (1)* Bd. 3760, Springer, 2005 (Lecture Notes in Computer Science), S. 502–519

[FHM+01] FUHR, Norbert ; HANSEN, Preben ; MABE, Michael ; MICSIK, András ; SØLVBERG, Ingeborg: Digital Libraries: A Generic Classification and Evaluation Scheme. In: CONSTANTOPOULOS, Panos (Hrsg.) ; SØLVBERG, Ingeborg (Hrsg.): *ECDL* Bd. 2163, Springer, 2001 (Lecture Notes in Computer Science). – ISBN 3–540–42537–3, S. 187–199

[FKS+02] FERNÁNDEZ, Mary ; KADIYSKA, Yana ; SUCIU, Dan ; MORISHIMA, Atsuyuki ; TAN, Wang-Chiew: SilkRoute: A framework for publishing relational data in XML. In: *ACM Trans. Database Syst.* 27 (2002), December, S. 438–493. http://dx.doi.org/10.1145/582410.582413. – DOI 10.1145/582410.582413. – ISSN 0362–5915

[FLO+05] FEKETE, Alan ; LIAROKAPIS, Dimitrios ; O'NEIL, Elizabeth J. ; O'NEIL, Patrick E. ; SHASHA, Dennis: Making snapshot isolation serializable. In: *ACM*

Trans. Database Syst. 30 (2005), Nr. 2, S. 492–528. http://dx.doi.org/10.1145/1071610.1071615. – DOI 10.1145/1071610.1071615

[FTA+07] FUHR, Norbert ; TSAKONAS, Giannis ; AALBERG, Trond ; AGOSTI, Maristella ; HANSEN, Preben ; KAPIDAKIS, Sarantos ; KLAS, Claus-Peter ; KOVÁCS, László ; LANDONI, Monica ; MICSIK, Andras ; PAPATHEODOROU, Christos ; PETERS, Carol ; SØLVBERG, Ingeborg: Evaluation of digital libraries. In: *Int. J. on Digital Libraries* 8 (2007), Nr. 1, S. 21–38. http://dx.doi.org/10.1007/s00799-007-0011-z. – DOI 10.1007/s00799–007–0011–z

[GAJVR08] GOTTSCHALK, Florian ; AALST, Wil M. P. d. ; JANSEN-VULLERS, Monique H. ; ROSA, Marcello L.: Configurable Workflow Models. In: *Int. J. Cooperative Inf. Syst.* 17 (2008), Nr. 2, S. 177–221

[GFWK04] GONÇALVES, Marcos A. ; FOX, Edward A. ; WATSON, Layne T. ; KIPP, Neill A.: Streams, structures, spaces, scenarios, societies (5s): A formal model for digital libraries. In: *ACM Trans. Inf. Syst.* 22 (2004), April, S. 270–312. – ISSN 1046–8188

[GMS87] GARCIA-MOLINA, Hector ; SALEM, Kenneth: Sagas. In: DAYAL, Umeshwar (Hrsg.) ; TRAIGER, Irving L. (Hrsg.): *SIGMOD Conference*, ACM Press, 1987, S. 249–259

[Gra81] GRAY, Jim: The Transaction Concept: Virtues and Limitations (Invited Paper). In: *VLDB*, IEEE Computer Society, 1981, S. 144–154

[Gre02] GREFEN, Paul W. P. J.: Transactional Workflows or Workflow Transactions? In: HAMEURLAIN, Abdelkader (Hrsg.) ; CICCHETTI, Rosine (Hrsg.) ; TRAUNMÜLLER, Roland (Hrsg.): *DEXA* Bd. 2453, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–44126–3, S. 60–69

[GSSZ02] GULBINS, Jürgen ; SEYFRIED, Markus ; STRACK-ZIMMERMANN, Hans: *Dokumenten-Management : vom Imaging zum Business-Dokument*. 3., überarb. und erw. Aufl. Berlin [u.a.] : Springer, 2002 (Xpert.press). – XII, 761 S.. – Literaturverz. S. 695 - 710

[GSU08] GUNJAL, Bhojaraju ; SHI, Hao ; URS, Shalini R.: Scholarly Publishing in Australian Digital Libraries: An Overview. In: *ICADL*, 2008, S. 194–202

[GWJV+09] GOTTSCHALK, Florian ; WAGEMAKERS, Teun A. C. ; JANSEN-VULLERS, Monique H. ; AALST, Wil M. P. d. ; ROSA, Marcello L.: Configurable Process Models: Experiences from a Municipality Case Study. In: ECK, Pascal van (Hrsg.) ; GORDIJN, Jaap (Hrsg.) ; WIERINGA, Roel (Hrsg.): *CAiSE* Bd. 5565,

Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–02143–5, S. 486–500

[HA00] HAGEN, Claus ; ALONSO, Gustavo: Exception Handling in Workflow Management Systems. In: *IEEE Trans. Software Eng.* 26 (2000), Nr. 10, 943-958. http://doi.ieeecomputersociety.org/10.1109/32.879818

[HAAR10] HOFSTEDE, Arthur H. M. (Hrsg.) ; AALST, Wil M. P. d. (Hrsg.) ; ADAMS, Michael (Hrsg.) ; RUSSELL, Nick (Hrsg.): *Modern Business Process Automation*. Springer Berlin / Heidelberg, 2010. – ISBN 978–3–642–03120–5

[Hal10] HALLERBACH, Alena: *Management von Prozessvarianten*, Universität Ulm, Diss., 2010

[HBR08] HALLERBACH, Alena ; BAUER, Thomas ; REICHERT, Manfred: Managing Process Variants in the Process Life Cycle. In: *ICEIS (3-2)*, 2008, S. 154–161

[HD02] HEUER, Andreas ; DITTRICH, Klaus R.: Schwerpunktthema: Content Management und digitale Bibliotheken. In: *Datenbank-Spektrum* 4 (2002), S. 7–8

[Heu09] HEUER, Andreas: *Digitale Bibliotheken und Content-Management-Systeme*. 2009. – Vorlesung

[HMR08] HERAVIZADEH, Mitra ; MENDLING, Jan ; ROSEMANN, Michael: Dimensions of Business Processes Quality (QoBP). In: ARDAGNA, Danilo (Hrsg.) ; MECELLA, Massimo (Hrsg.) ; YANG, Jian (Hrsg.): *Business Process Management Workshops* Bd. 17, Springer, 2008 (Lecture Notes in Business Information Processing). – ISBN 978–3–642–00327–1, S. 80–91

[HR83] HÄRDER, Theo ; REUTER, Andreas: Principles of Transaction-Oriented Database Recovery. In: *ACM Comput. Surv.* 15 (1983), Nr. 4, S. 287–317. http://dx.doi.org/10.1145/289.291. – DOI 10.1145/289.291

[Hum12] HUMBOLDT-UNIVERSITÄT ZU BERLIN: *SGML/XML (DiML)*. Webseite: http://edoc.hu-berlin.de/e_autoren/alternativen.php?arbeit=Dissertationen%20%C2%BB&index=index.php&nav=diss, 2012. – Zugriff am 22.01.2012

[IFL10] IFLA: *IFLA Manifesto for Digital Libraries*. http://www.ifla.org/files/hq/documents/digital-library-manifesto-en.pdf. Version: 2010

[IMSS08] IOANNIDIS, Yannis ; MILANO, Diego ; SCHEK, Hans-Jörg ; SCHULDT, Heiko: DelosDLMS. In: *International Journal on Digital Libraries* 9 (2008), S. 101–

114. http://dx.doi.org/10.1007/s00799-008-0044-y. – DOI 10.1007/s00799–008–0044–y

[KM03] KLETTKE, Meike ; MEYER, Holger: *XML & Datenbanken : Konzepte, Sprachen und Systeme*. 1. Aufl. Heidelberg : dpunkt-Verl., 2003 (xml.bibliothek). – XIV, 428 S.

[KMFS09] KUHR, Jan-Christian ; MEYER, Holger ; FORBRIG, Peter ; SATTLER, Robert: *PERIKLES – Vorhabenbeschreibung*. 2009. – Projketantrag

[KR09] KÜNZLE, Vera ; REICHERT, Manfred: Towards Object-Aware Process Management Systems: Issues, Challenges, Benefits. In: *Enterprise, Business-Process and Information Systems Modeling* Bd. 29. Springer Berlin Heidelberg, 2009. – ISBN 978–3–642–01862–6, S. 197–210

[KR11] KÜNZLE, Vera ; REICHERT, Manfred: PHILharmonicFlows: towards a framework for object-aware process management. In: *Journal of Software Maintenance* 23 (2011), Nr. 4, S. 205–244. http://dx.doi.org/10.1002/smr.524. – DOI 10.1002/smr.524

[LAB+06] LUDÄSCHER, Bertram ; ALTINTAS, Ilkay ; BERKLEY, Chad ; HIGGINS, Dan ; JAEGER, Efrat ; JONES, Matthew B. ; LEE, Edward A. ; TAO, Jing ; ZHAO, Yang: Scientific workflow management and the Kepler system. In: *Concurrency and Computation: Practice and Experience* 18 (2006), Nr. 10, S. 1039–1065. http://dx.doi.org/10.1002/cpe.994. – DOI 10.1002/cpe.994

[Ley95] LEYMANN, Frank: Supporting Business Transactions Via Partial Backward Recovery In Workflow Management Systems. In: *BTW*, 1995, S. 51–70

[Lib11] LIBRARY OF CONGRESS: *Metadata Encoding and Transmission Standard (METS)*. Webseite: http://www.loc.gov/standards/mets/, Mai 2011. – Zugriff am 20.05.2011

[LPSW06] LAGOZE, Carl ; PAYETTE, Sandra ; SHIN, Edwin ; WILPER, Chris: Fedora: an architecture for complex objects and their relationships. In: *Int. J. on Digital Libraries* 6 (2006), Nr. 2, S. 124–138. http://dx.doi.org/10.1007/s00799-005-0130-3. – DOI 10.1007/s00799–005–0130–3

[Lüt02] LÜTZENKIRCHEN, Frank: MyCoRe - Ein Open-Source-System zum Aufbau digitaler Bibliotheken. In: *Datenbank-Spektrum* 4 (2002), S. 23–27

[LWMB09] LUDÄSCHER, Bertram ; WESKE, Mathias ; MCPHILLIPS, Timothy M. ; BOWERS, Shawn: Scientific Workflows: Business as Usual? In: DAYAL, Umeshwar (Hrsg.) ; EDER, Johann (Hrsg.) ; KOEHLER, Jana (Hrsg.) ; REIJERS,

Hajo A. (Hrsg.): *BPM* Bd. 5701, Springer, 2009 (Lecture Notes in Computer Science). – ISBN 978–3–642–03847–1, S. 31–47

[MGR04] MÜLLER, Robert ; GREINER, Ulrike ; RAHM, Erhard: $A_{GENT}W_{ORK}$: a workflow system supporting rule-based workflow adaptation. In: *Data Knowl. Eng.* 51 (2004), Nr. 2, S. 223–256. http://dx.doi.org/10.1016/j.datak.2004.03.010. – DOI 10.1016/j.datak.2004.03.010

[MK05] MÜLLER, Uwe ; KLATT, Manuel: SCOPE - A Generic Framework for XML Based Publishing Processes. In: *ECDL*, 2005, S. 104–115

[MMC+10] MANGHI, Paolo ; MIKULICIC, Marko ; CANDELA, Leonardo ; ARTINI, Michele ; BARDI, Alessia: General-Purpose Digital Library Content Laboratory Systems. In: *ECDL*, 2010, S. 14–21

[Mos81] MOSS, J. Eliot B.: Nested Transactions: An Approach to Reliable Distributed Computing / MIT. Version: 1981. http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-260.pdf. 1981 (MIT/LCS/TR-260). – Forschungsbericht

[Mos85] MOSS, J. Eliot B.: *Nested Transactions*. MIT Press, 1985. – ISBN 0–262–13200–1

[MRA+10] MANS, R. S. ; RUSSELL, Nick C. ; AALST, Wil M. P. d. ; BAKKER, Piet J. M. ; MOLEMAN, Arnold J. ; JASPERS, Monique W. M.: Proclets in healthcare. In: *Journal of Biomedical Informatics* 43 (2010), Nr. 4, S. 632–649

[MRH07] MÜLLER, Dominic ; REICHERT, Manfred ; HERBST, Joachim: Data-Driven Modeling and Coordination of Large Process Structures. In: *OTM Conferences (1)*, 2007, S. 131–149

[MRH08] MÜLLER, Dominic ; REICHERT, Manfred ; HERBST, Joachim: A New Paradigm for the Enactment and Dynamic Adaptation of Data-Driven Process Structures. In: *CAiSE*, 2008, S. 48–63

[MSKB07] MINOR, Mirjam ; SCHMALEN, Daniel ; KOLDEHOFF, Andreas ; BERGMANN, Ralph: Structural Adaptation of Workflows Supported by a Suspension Mechanism stand by Case-Based Reasoning. In: *WETICE*, IEEE Computer Society, 2007, 370-375

[MT06] MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.): *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part*

I. Bd. *4275*. Springer, 2006 (Lecture Notes in Computer Science). – ISBN 3–540–48287–3

[MT07] MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.): *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*. Bd. *4803*. Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–76846–3

[Mud12] MUDERACK, Stefan: *Ein Transaktions-Service für tx+YAWL*. Studienarbeit, April 2012

[OWK+11] OUYANG, Chun ; WYNN, Moe T. ; KUHR, Jan-Christian ; ADAMS, Michael ; BECKER, Thomas ; HOFSTEDE, Arthur H. M. ; FIDGE, Colin J.: Workflow support for scheduling in surgical care processes. In: TUUNAINEN, Virpi K. (Hrsg.) ; ROSSI, Matti (Hrsg.) ; NANDHAKUMAR, Joe (Hrsg.): *ECIS*, 2011

[Pes08] PESIC, Maja: *Constraint-Based Workflow Management Systems: Shifting Control to Users*, Technische Universiteit Eindhoven, Diss., 2008

[PML07] POTTINGER, Stefan ; MIETZNER, Ralph ; LEYMANN, Frank: Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework. In: [MT07], S. 336–352

[PSA07] PESIC, Maja ; SCHONENBERG, Helen ; AALST, Wil M. P. d.: DECLARE: Full Support for Loosely–Structured Processes. In: *EDOC*, IEEE Computer Society, 2007, 287-300

[PWZ+11] PICHLER, Paul ; WEBER, Barbara ; ZUGAL, Stefan ; PINGGERA, Jakob ; MENDLING, Jan ; REIJERS, Hajo: Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In: *ER-BPM*, Springer Berlin / Heidelberg, 2011, S. 12. – in press

[RD98] REICHERT, Manfred ; DADAM, Peter: $\text{ADEPT}_{\text{flex}}$ — Supporting Dynamic Changes of Workflows Without Losing Control. In: *J. Intell. Inf. Syst.* 10 (1998), Nr. 2, S. 93–129

[REHA04] RUSSELL, Nick ; EDMOND, David ; HOFSTEDE, Arthur H. M. ; AALST, Wil M. P. d.: Workflow Data Patterns / Queensland University of Technology. Brisbane, Australia, 2004. – Report FIT-TR-2004-01

[Rei00] REICHERT, Manfred: *Dynamische Ablaufänderungen in Workflow-Management-Systemen*, university of Ulm, Diss., Mai 2000. `http://dbis.eprints.uni-ulm.de/433/1/PhD_Thesis_Reichert.pdf`

[Rei10] REISIG, Wolfgang: *Petrinetze : Modellierungstechnik, Analysemethoden, Fallstudien*. Wiesbaden : Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH, Wiesbaden, 2010. – Online–Ressource S.

[RH10] RUSSELL, Nick ; HOFSTEDE, Arthur: The Language: Rationale and Fundamentals. Version: 2010. http://dx.doi.org/10.1007/978-3-642-03121-2_2. In: HOFSTEDE, Arthur H. M. (Hrsg.) ; AALST, Wil M. P. (Hrsg.) ; ADAMS, Michael (Hrsg.) ; RUSSELL, Nick (Hrsg.): *Modern Business Process Automation*. Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–03121–2, 23-102

[RHAM06] RUSSELL, Nick ; HOFSTEDE, Arthur H. M. ; AALST, Wil M. P. d. ; MULYAR, Nataliya: Workflow Control-Flow Patterns : A Revised View / BPM Center. Version: 2006. http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf. 2006 (BPM-06-22). – Forschungsbericht

[RHEA05] RUSSELL, Nick ; HOFSTEDE, Arthur H. M. ; EDMOND, David ; AALST, Wil M. P. d.: Workflow Data Patterns: Identification, Representation and Tool Support. Version: 2005. http://dx.doi.org/10.1007/11568322_23. In: *ER 2005* Bd. 3716. Springer Berlin / Heidelberg, 2005. – DOI 10.1007/11568322_23, S. 353–368

[Rig09] RIGGERT, Wolfgang: *ECM - Enterprise Content Management : Konzepte und Techniken rund um Dokumente ; mit 17 Tabellen*. 1. Aufl. Wiesbaden : Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, 2009. – Online–Ressource S.

[Rit84] RITCHIE, D. M.: A stream input-output system. In: *AT&T Bell Laboratories Technical Journal* 63 (1984), Nr. 8, S. 1897–1910

[RLS+07] ROSA, Marcello L. ; LUX, Johannes ; SEIDEL, Stefan ; DUMAS, Marlon ; HOFSTEDE, Arthur H. M.: Questionnaire-driven Configuration of Reference Process Models. In: KROGSTIE, John (Hrsg.) ; OPDAHL, Andreas L. (Hrsg.) ; SINDRE, Guttorm (Hrsg.): *CAiSE* Bd. 4495, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–72987–7, S. 424–438

[RMRW08] RINDERLE-MA, Stefanie ; REICHERT, Manfred ; WEBER, Barbara: On the Formal Semantics of Change Patterns in Process-Aware Information Systems. In: LI, Qing (Hrsg.) ; SPACCAPIETRA, Stefano (Hrsg.) ; YU, Eric S. K. (Hrsg.) ; OLIVÉ, Antoni (Hrsg.): *ER* Bd. 5231, Springer, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–87876–6, S. 279–293

[RRD03] REICHERT, Manfred ; RINDERLE, Stefanie ; DADAM, Peter: On the Common Support of Workflow Type and Instance Changes under Correctness Constraints. In: MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.) ; SCHMIDT, Douglas C. (Hrsg.): *CoopIS/DOA/ODBASE* Bd. 2888, Springer, 2003 (Lecture Notes in Computer Science). – ISBN 3–540–20498–9, S. 407–425

[RRD04a] RINDERLE, Stefanie ; REICHERT, Manfred ; DADAM, Peter: Correctness criteria for dynamic changes in workflow systems - a survey. In: *Data Knowl. Eng.* 50 (2004), Nr. 1, S. 9–34. http://dx.doi.org/10.1016/j.datak.2004.01.002. – DOI 10.1016/j.datak.2004.01.002

[RRD04b] RINDERLE, Stefanie ; REICHERT, Manfred ; DADAM, Peter: Flexible Support of Team Processes by Adaptive Workflow Systems. In: *Distributed and Parallel Databases* 16 (2004), Nr. 1, S. 91–116

[RRKD05] REICHERT, Manfred ; RINDERLE, Stefanie ; KREHER, Ulrich ; DADAM, Peter: Adaptive Process Management with ADEPT2. In: *ICDE*, IEEE Computer Society, 2005. – ISBN 0–7695–2285–8, 1113-1114

[RRMD09] REICHERT, Manfred ; RINDERLE-MA, Stefanie ; DADAM, Peter: Flexibility in Process-Aware Information Systems. In: *T. Petri Nets and Other Models of Concurrency* 2 (2009), S. 115–135. http://dx.doi.org/10.1007/978-3-642-00899-3_7. – DOI 10.1007/978–3–642–00899–3_7. ISBN 978–3–642–00898–6

[RRS+11] REIMANN, Peter ; REITER, Michael ; SCHWARZ, Holger ; KARASTOYANOVA, Dimka ; LEYMANN, Frank: SIMPL - A Framework for Accessing External Data in Simulation Workflows. In: HÄRDER, Theo (Hrsg.) ; LEHNER, Wolfgang (Hrsg.) ; MITSCHANG, Bernhard (Hrsg.) ; SCHÖNING, Harald (Hrsg.) ; SCHWARZ, Holger (Hrsg.): *BTW* Bd. 180, GI, 2011 (LNI). – ISBN 978–3–88579–274–1, S. 534–553

[RS95] REUTER, Andreas ; SCHWENKREIS, Friedemann: ConTracts - A Low-Level Mechanism for Building General-Purpose Workflow Management-Systems. In: *IEEE Data Eng. Bull.* 18 (1995), Nr. 1, S. 4–10

[RSWH09] RAZUM, Matthias ; SCHWICHTENBERG, Frank ; WAGNER, Steffen ; HOPPE, Michael: eSciDoc Infrastructure: A Fedora-Based e-Research Framework. Version: 2009. http://dx.doi.org/10.1007/978-3-642-04346-8_23. In: *Research and Advanced Technology for Digital Libraries* Bd. 5714. Springer Berlin / Heidelberg, 2009. – DOI 10.1007/978–3–642–04346–8_23, S. 227–238

[SABS02] SCHULDT, Heiko ; ALONSO, Gustavo ; BEERI, Catriel ; SCHEK, Hans-Jörg: Atomicity and isolation for transactional processes. In: *ACM Trans. Database Syst.* 27 (2002), Nr. 1, S. 63–116. http://dx.doi.org/10.1145/507234.507236. – DOI 10.1145/507234.507236

[Sch99] SCHULZ, Matthias: *Dissertation Markup Language (DiML)*. Online. http://edoc.hu-berlin.de/e_autoren/software/diml doc.pdf. Version: Juni 1999. – Zugriff am 22.01.2012

[Sch04] SCHRAITLE, Thomas: *DocBook-XML : medienneutrales und plattformunabhängiges Publizieren*. Nürnberg : SuSE Press, 2004. – XX, 615 S.. – Literatur- und URL-Verz. S. 593 - 599

[Sch05] SCHICK, Sebastian: *Publikationsprozesse in digitalen Bibliotheken - Mittel und Methoden der Autorenunterstützung*, Universität Rostock, Fakultät für Informatik und Elektrotechnik Institut für Informatik, Diplomarbeit, Oktober 2005

[Sch07] SCHIRMBACHER, Peter: Neue Kultur des elektronischen Publizierens unter dem Gesichtspunkt alternativer Publikationsmodelle. In: HAVEMANN, Frank (Hrsg.) ; PARTHEY, Heinrich (Hrsg.) ; UMSTÄTTER, Walther (Hrsg.): *Integrität wissenschaftlicher Publikationen in der Digitalen Bibliothek*. Humboldt-Universität zu Berlin, Gesellschaft für Wissenschaftsforschung e.V. (GeWiF Berlin), 2007 (Wissenschaftsforschung Jahrbuch 2007). – ISBN 3–924682–43–x, S. 51–70. – Online: Stand 2011-06-16

[Sch10] SCHIRMBACHER, Peter: Möglichkeiten und Grenzen des wissenschaftlichen elektronischen Publizierens - mit besonderem Bezug auf Veröffentlichungen nach den Open-Access-Prinzipien. In: *Wissenswerkstatt Staatsbibliothek zu Berlin - Stiftung Preußischer Kulturbesitz*. Berlin : Humboldt-Universität zu Berlin, 2010

[Sch12] SCHULT, Andreas: *Materialisierte Sichten auf ereignisorientierten Datenströmen*. Studienarbeit, Februar 2012

[SGGMR10] SÁNCHEZ-GONZÁLEZ, Laura ; GARCÍA, Félix ; MENDLING, Jan ; RUIZ, Francisco: Quality Assessment of Business Process Models Based on Thresholds. In: MEERSMAN, Robert (Hrsg.) ; DILLON, Tharam S. (Hrsg.) ; HERRERO, Pilar (Hrsg.): *OTM Conferences (1)* Bd. 6426, Springer, 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–16933–5, S. 78–95

[Sha08] SHAPIRO, Robert M.: *XPDL 2.1 Integrating Process Interchange & BPMN*. White Paper. http://www.wfmc.org/Download-document/XPDL-

2.1-Integrating-Process-Interchange-BPMN.html. Version: Januar 2008. – www.wfmc.org

[SKS+01] SHANMUGASUNDARAM, Jayavel ; KIERNAN, Jerry ; SHEKITA, Eugene J. ; FAN, Catalina ; FUNDERBURK, John E.: Querying XML Views of Relational Data. In: APERS, Peter M. G. (Hrsg.) ; ATZENI, Paolo (Hrsg.) ; CERI, Stefano (Hrsg.) ; PARABOSCHI, Stefano (Hrsg.) ; RAMAMOHANARAO, Kotagiri (Hrsg.) ; SNODGRASS, Richard T. (Hrsg.): *VLDB*, Morgan Kaufmann, 2001. – ISBN 1–55860–804–4, 261-270

[SMBH11] SCHICK, Sebastian ; MEYER, Holger ; BANDT, Markus ; HEUER, Andreas: Enabling YAWL to Handle Dynamic Operating Room Management. In: DANIEL, Florian (Hrsg.) ; BARKAOUI, Kamel (Hrsg.) ; DUSTDAR, Schahram (Hrsg.): *Business Process Management Workshops (2)* Bd. 100, Springer, 2011 (Lecture Notes in Business Information Processing). – ISBN 978–3–642–28114–3, S. 249–260

[SMH07] SCHICK, Sebastian ; MEYER, Holger ; HEUER, Andreas: Ein Framework für die Publikation von Multimediadokumenten in digitalen Bibliotheken. In: *Grundlagen von Datenbanken*, 2007, S. 7–11

[SMH11a] SCHICK, Sebastian ; MEYER, Holger ; HEUER, Andreas: Enhancing Workflow Data Interaction Patterns by a Transaction Model. In: EDER, Johann (Hrsg.) ; BIELIKOVA, Maria (Hrsg.) ; TJOA, A M. (Hrsg.): *Proceedings II of the 15th East-European Conference on Advances in Databases and Information Systems, ADBIS 2011, Vienna, Austria* Bd. 789, CEUR, September 2011. – ISSN 1613–0073, 33-44

[SMH11b] SCHICK, Sebastian ; MEYER, Holger ; HEUER, Andreas: Flexible Publication Workflows Using Dynamic Dispatch. In: XING, Chunxiao (Hrsg.) ; CRESTANI, Fabio (Hrsg.) ; RAUBER, Andreas (Hrsg.): *ICADL* Bd. 7008, Springer, 2011 (Lecture Notes in Computer Science). – ISBN 978–3–642–24825–2, S. 257–266

[SMR+08] SCHONENBERG, Helen ; MANS, R. S. ; RUSSELL, Nick ; MULYAR, Nataliya ; AALST, Wil M. P. d.: Process flexibility: A survey of contemporary approaches. In: DIETZ, Jan L. G. (Hrsg.) ; ALBANI, Antonia (Hrsg.) ; BARJIS, Joseph (Hrsg.): *CIAO! / EOMAS* Bd. 10 LNBIP, Springer, 2008 (Lecture Notes in Business Information Processing). – ISBN 978–3–540–68643–9, S. 16–30

[SOS05] SADIQ, Shazia W. ; ORLOWSKA, Maria E. ; SADIQ, Wasim: Specification and validation of process constraints for flexible workflows. In: *Inf. Syst.* 30 (2005), Nr. 5, S. 349–378

[SOSF04] SADIQ, Shazia W. ; ORLOWSKA, Maria E. ; SADIQ, Wasim ; FOULGER, Cameron: Data Flow and Validation in Workflow Modelling. In: SCHEWE, Klaus-Dieter (Hrsg.) ; WILLIAMS, Hugh E. (Hrsg.): *ADC* Bd. 27, Australian Computer Society, 2004 (CRPIT). – ISBN 1–920682–06–6, 207-214

[SSH07] SAAKE, Gunter ; SATTLER, Kai-Uwe ; HEUER, Andreas: *Datenbanken - Konzepte und Sprachen*. MITP-Verlag, Bonn, 2007

[TBB08] TOURÉ, Fodé ; BAÏNA, Karim ; BENALI, Khalid: An Efficient Algorithm for Workflow Graph Structural Verification. Version: 2008. http://dx.doi.org/10.1007/978-3-540-88871-0_26. In: MEERSMAN, Robert (Hrsg.) ; TARI, Zahir (Hrsg.): *On the Move to Meaningful Internet Systems: OTM 2008* Bd. 5331. Springer Berlin / Heidelberg, 2008. – DOI 10.1007/978–3–540–88871–0_26. – ISBN 978–3–540–88870–3, S. 392–408

[Tog11a] TOGETHER: *Together XPDL and BPMN Workflow Editor*, Oktober 2011. http://www.together.at/prod/workflow/twe/manual

[Tog11b] TOGETHER: *Together XPDL and BPMN Workflow Server*, Oktober 2011. http://www.together.at/prod/workflow/tws/manual

[UDGBR11] UBA, Reina ; DUMAS, Marlon ; GARCÍA-BAÑUELOS, Luciano ; ROSA, Marcello L.: Clone Detection in Repositories of Business Process Models. In: RINDERLE-MA, Stefanie (Hrsg.) ; TOUMANI, Farouk (Hrsg.) ; WOLF, Karsten (Hrsg.): *BPM* Bd. 6896, Springer, 2011 (Lecture Notes in Computer Science). – ISBN 978–3–642–23058–5, S. 248–264

[Uni12] UNIVERSITY OF SOUTHAMPTON: *EPrints*. Webseite: http://www.eprints.org/, 2012. – Zugriff am 12.02.2012

[VRA08] VANDERFEESTEN, Irene T. P. ; REIJERS, Hajo A. ; AALST, Wil M. P. d.: Product Based Workflow Support: Dynamic Workflow Execution. In: BELLAHSENE, Zohra (Hrsg.) ; LÉONARD, Michel (Hrsg.): *CAiSE* Bd. 5074, Springer, 2008 (Lecture Notes in Computer Science). – ISBN 978–3–540–69533–2, S. 571–574

[VRA11] VANDERFEESTEN, Irene T. P. ; REIJERS, Hajo A. ; AALST, Wil M. P. d.: Product-based workflow support. In: *Inf. Syst.* 36 (2011), Nr. 2, S. 517–535

[VW99] VOSSEN, Gottfried ; WESKE, Mathias: The WASA2 Object-Oriented Workflow Management System. In: DELIS, Alex (Hrsg.) ; FALOUTSOS, Christos (Hrsg.) ; GHANDEHARIZADEH, Shahram (Hrsg.): *SIGMOD Conference*, ACM Press, 1999. – ISBN 1–58113–084–8, S. 587–589

[W3C11] W3C: *SMIL*. Webseite, Mai 2011. – Zugriff auf www.w3.org/AudioVideo/ am 05.09.2011

[WBN10] WITTEN, Ian H. ; BAINBRIDGE, David ; NICHOLS, David M.: *How to Build a Digital Library*. 2nd Edition. Morgan Kaufmann, 2010. – ISBN 978–0–12–374857–7

[WEAH05] WYNN, Moe T. ; EDMOND, David ; AALST, Wil M. P. d. ; HOFSTEDE, Arthur H. M.: Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In: CIARDO, Gianfranco (Hrsg.) ; DARONDEAU, Philippe (Hrsg.): *ICATPN* Bd. 3536, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–26301–2, S. 423–443

[Wei88] WEIKUM, G.: *Transaktionen in Datenbanksystemen: fehlertolerante Steuerung paralleler Abläufe*. Addison-Wesley, 1988. – ISBN 3–925118–91–8

[Wei91] WEIKUM, Gerhard: Principles and Realization Strategies of Multilevel Transaction Management. In: *ACM Trans. Database Syst.* 16 (1991), Nr. 1, S. 132–180. http://dx.doi.org/10.1145/103140.103145,db/journals/tods/weikum91.html. – DOI 10.1145/103140.103145, db/journals/-tods/weikum91.html

[Wes00] WESKE, Mathias: *Workflow management systems: Formal foundation, conceptual design, implementation aspects*. Habilitationsschrift. http://bpt.hpi.uni-potsdam.de/pub/Public/MathiasWeske/habil.pdf. Version: 2000

[Wes07] WESKE, Mathias: *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007. – ISBN 978–3–540–73521–2

[WfM98] WFMC: *Workflow Management Application Programming Interface (Interface 2&3) Specification*. Version: Juli 1998. www.wfmc.org/Download-document/WFMC-TC-1009-Ver-2-Workflow-Management-API-23-Specification.html. Online PDF. – WFMC-TC-1009

[WfM99] WFMC: *WfMC: Terminology and Glosssary*. Version: 1999. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf. Online PDF. – 65 S. – Glossary. – WFMC-TC-1011

[WK05] WANG, Jianrui ; KUMAR, Akhil: A Framework for Document-Driven Workflow Systems. In: AALST, Wil M. P. d. (Hrsg.) ; BENATALLAH, Boualem (Hrsg.) ; CASATI, Fabio (Hrsg.) ; CURBERA, Francisco (Hrsg.): *Business Process Management* Bd. 3649, 2005. – ISBN 3–540–28238–6, S. 285–301

[WR92] WÄCHTER, Helmut ; REUTER, Andreas: The ConTract Model. In: *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992, S. 219–263

[WRM06] WANG, Ling ; RUNDENSTEINER, Elke A. ; MANI, Murali: Updating XML views published over relational databases: Towards the existence of a correct update mapping. In: *Data Knowl. Eng.* 58 (2006), Nr. 3, S. 263–298. http://dx.doi.org/10.1016/j.datak.2005.07.003. – DOI 10.1016/j.datak.2005.07.003

[WRRM08] WEBER, Barbara ; REICHERT, Manfred ; RINDERLE-MA, Stefanie: Change patterns and change support features - Enhancing flexibility in process-aware information systems. In: *Data Knowl. Eng.* 66 (2008), Nr. 3, S. 438–466

[WS91] WEIKUM, Gerhard ; SCHEK, Hans-Jörg: Multi-Level Transactions and Open Nested Transactions. In: *IEEE Data Eng. Bull.* 14 (1991), Nr. 1, 60-64. http://sites.computer.org/debull/91MAR-CD.pdf

[WSR09] WEBER, Barbara ; SADIQ, Shazia W. ; REICHERT, Manfred: Beyond rigidity - dynamic process lifecycle support. In: *Computer Science - R&D* 23 (2009), Nr. 2, S. 47–65

[WV01] WEIKUM, Gerhard ; VOSSEN, Gottfried: *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. San Francisco, CA, USA : Morgan Kaufmann, 2001

Abbildungsverzeichnis

Tabellenverzeichnis

Teil V

Anhang

Anhang A

YAWL

A.1 Muster die von *YAWL* unterstützt werden

A.1.1 Kontrollflussmuster

In [RHAM06] stellen Russell et al. eine umfangreiche Sammlung von komplexen Kontrollflussmustern vor, von denen *YAWL* 31 unterstützt [RH10]. Tabelle A.1 zeigt auf, welche von diesen Mustern von *YAWL* unterstützt werden.

Tabelle A.1: Von *YAWL* unterstützte Kontrollflussmuster (angepasst aus [RH10])

Muster Kategorie	**Unterstützte Muster (Nummer)**
Verzweigungsmuster	Parallel Split (2), Exclusive Choice (4), Deferred Choice (16), Multi-Choice (6)
Synchronisationsmuster	Synchronization (3), Generalised AND-Join (33), Simple Merge (5), Multi-Merge (8), Structured Synchronizing Merge (7), Local Synchronizing Merge (37), General Synchronizing Merge (38)
Wiederholungsmuster	Arbitrary Cycles (10), Recursion (22)
Multiple Instances (MI) Muster	MI without Synchronization (12), MI with a Priori Design-Time Knowledge (13), MI with a Priori Run-Time Knowledge (14), MI without a Priori Run-Time Knowledge (15), Static Partial Join for Multiple Instances (34), Cancelling Partial Join for Multiple Instances (35)
Muster für Nebenläufigkeit	Sequence (1), Interleaved Parallel Routing (17), Interleaved Routing (40), Critical Section (39), Milestone (18)

Tabelle A.1: Von *YAWL* unterstützte Kontrollflussmuster (angepasst aus [RH10])

Muster Kategorie	**Unterstützte Muster (Nummer)**
Muster für die Ausnahmebehandlung	Cancel Task (19), Cancel Case (20), Cancel Region (25), Cancel Multiple Instance Activity (26)
Muster zum Beenden	Explicit Termination (43)

A.1.2 Datenflussmuster

In [REHA04, RHEA05] stellen Russell et al. eine Reihe von Datenflussmustern vor. Tabelle A.2 listet alle Datenflussmuster auf, die von *YAWL* unterstützt werden.

Tabelle A.2: Von *YAWL* unterstützte Datenflussmuster (angepasst aus [RH10])

Muster Kategorie	**Unterstützte Muster (Nummer)**
Datensichtbarkeit	Task Data (1), Block Data (2), Multiple Instance Data (3)
Datenineraktion Intern	Task to Task (9), Block Task to Sub-Workflow Decomposition (10), Sub-Workflow Decomposition to Block Task (11), to Multiple Instance Task (12), from Multiple Instance Task (13)
Datentransfer	Data Transfer by Value - Incoming (27), Data Transfer by Value - Outgoing (28), Data Transformation - Input (32), Data Transformation - Output (33)
Datenbasiertes Routing	Data-based Routing (40)

A.2 Formale Definition von *YAWL*

In diesem Abschnitt werden die formale Definition einer *YAWL*-Spezifikation, die Definition von einem *YAWL*-Netz und die Definition des *YAWL*-Datenflussmodells gegeben. Die Definitionen sind aus [RH10] übernommen.

Definition A.1 (*YAWL* Spezifikation [RH10])
Eine *YAWL*-Spezifikation ist definiert als tuple = *(NetID, ProcessID, TaskID, MITaskID, VarID, ParamID, TNmap, NYmap, VarName, DataType, VName, DType, VarType, VNmap, VTmap, VMmap)* mit:

— globale Objekte —

- *NetID* ist die Menge aller Netzidentifikatoren (top-level Netz und Subnetze).
- *ProcessID* $\in$ *NetID* ist der Prozessidentifikator.
- *TaskID* ist die Menge aller Aktivitätsidentifikatoren.
- *MITaskID* $\subseteq$ *TaskID* ist die Menge aller Identifikatoren für MI-Aktivitäten.
- *VarID* ist die Menge alle Identifikatoren für Variablen, welche in den Netzen verwendet werden.
- *ParamID* ist die Menge alle Identifikatoren für Parameter, welche in den Netzen verwendet werden.

— Dekomposition —

- *TNmap*: *TaskID* $\nrightarrow$ *NetID* beschreibt die Abbildung von einer zusammengesetzten Aktivität und dem dazugehörigen Subnetz.
- *NYmap*: *NetID* $\rightarrow$ *YAWLnet* gibt die vollständige Beschreibung des Inhaltes aller Netze wieder. Es gilt: n $\in$ *NetID* und *NYmap(n)* entspricht Definition A.2. Dabei ist die Menge aller Aktivitäten T_n eines Netzes, für die gilt $\forall_{m,n \in NetID}[T_m \cap T_n \neq \varnothing \Rightarrow m = n]$.

— Variablen —

- *VarName* ist die Menge aller Variablennamen, die in allen Netzen definiert sind.
- *DataType* ist die Menge aller verwendeten Datentypen.
- *Vname*: *VarID* $\rightarrow$ *VarName* definiert den Namen einer Variable.
- *DType*: *VarID* $\rightarrow$ *DataType* beschreibt den Datentyp einer Variable.
- *VarType*: *VarID* $\rightarrow$ *{Net, Task, MI}* beschreibt die Sichtbarkeitsbereiche von Variablen.
- *VNmap*: $VarID^{Net} \rightarrow NetID$ ordnet eine Netzvariable einem Netz zu.
- *VTmap*: $VarID^{Task} \rightarrow TaskID$ ordnet eine Variable einer Aktivität zu.
- *VMmap*: $VarID^{MI} \rightarrow MITaskID$ ordnet eine MI Variable einer MI-Aktivität zu.

In Definition A.1 wird die Definition für die *YAWL*-Spezifikation gezeigt. Eine *YAWL*-Spezifikation enthält neben *globalen Objekten*, eine *Dekomposition* und *Variablen*.

Definition A.2 (*YAWL*-Netz [RH10])
Ein *YAWL*-Netz ist definiert als tupel=(*nid, C, i, o, T,* T_A, T_C, *M, F, Split, Join, Default,* $<_{XOR}$, *Rem, Nofi, ArcCond*) mit:

— Kontrollflusselemente —

- $nid \in NetId$ definiert die Identität eines *YAWL*-Netzes.
- $i \in C$ ist die input condition.
- $o \in C$ ist die output condition.
- T ist die Menge von Aktivitäten.
- $T_A \in T$ ist die Menge aller atomaren Aktivitäten.
- $T_C \in T$ ist die Menge aller zusammengesetzten Aktivitäten.
- T_A und T_C sind hier aber nur eine Teilmenge von T.
- $M \subseteq T$ ist die Menge aller MI-Aktivitäten.
- $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$ definiert, dass jeder Knoten im Graph $(C \cup T, F)$ auf einem gerichtetem Pfad von i nach o liegt.
- $Split$: $T \nrightarrow \{AND, XOR, OR\}$ definiert die Verzweigungsart einer Aktivität.
- $Join$: $T \nrightarrow \{AND, XOR, OR\}$ definiert die Synchronisierungsart einer Aktivität.
- $Default \subseteq F, Default : dom(Split \rhd \{OR\}) \rightarrow T \cup C$ definiert die Standardkante für einen *OR-Split*. Dabei garantiert $Default(t)$, dass für den Fall das keine Kante zu *true* evaluiert wird, die letzte Kante verwendet wird.
- $<_{XOR} \subseteq \{t \in T \mid Split(t) = XOR\} \times \mathbb{P}((T \cup C) \times (T \cup C))$ definiert die Reihenfolge, in der ein *XOR-Split* ausgewertet wird. Dabei muss folgendes gelten: wenn $(t, V) \in <_{XOR}$, dann $<^t_{XOR} = V$. Außerdem muss gelten, dass V eine strikte Ordnung über $t\bullet = \{x \in T \cup C \mid (t, x) \in F\}$ ist. Hier werden alle Bedingungen evaluiert, solange bis eine das Ergebnis *true* liefert. Ist das nicht der Fall, wird die *Default-Kante* ($\perp_t$) verwendet.
- $Rem : T \nrightarrow \mathbb{P}^+(T \cup C \setminus \{i, o\})$ definiert Token in einem Netz, die an eine *cancelation region* gebunden sind und bei deren Aktivierung entsprechend entfernt werden.
- $Nofi : M \rightarrow \mathbb{N} \times \mathbb{N}^{inf} \times \mathbb{N}^{inf} \times \{dynamic, static\}$ definiert die obere und untere Grenzen einer MI-Aktivität (bzw. die Anzahl der zu erzeugenden Instanzen).

— Bedingungen an Kanten —

- $ArcCond : F \cap (dom(Split \rhd \{XOR, OR\}) \times (T \cup C)) \rightarrow BoolExpr$ definiert die Bedingung, welche an eine Kante mit *XOR* oder *OR* Verzweigung gebunden ist.

Netze enthalten *Kontrollflusselemente* und *Kontrollflussbedingungen* als Bedingungen an Kanten (siehe Definition A.2).

In Definition A.3 wird das *Datenflussmodell* von *YAWL* definiert. In *YAWL* ist es immer an ein Netz gebunden. Es beschreibt die Abbildung von Netzvariablen auf Aktivitätsvariablen unter Verwendung von Parametern.

Definition A.3 (Datenflussmodell [RH10])
Ein Datenflussmodell ist in *YAWL* an ein Netz mit der ID *nid* gebunden und ist definiert als *model = (ParamVar, InPar, OutPar, Accessor, Splitter Instance, Aggregate)*:

- $ParamVar : ParamID \rightarrow VarID \times VarID$ ist eine Funktion, welche die Ein- und Ausgabe Variablen zu einem Parameter bestimmt.
- $InPar : ParamID \times T \rightarrow Expr$ definiert das Mapping für den Eingabe-Parameter, das während der Initialisierung einer Aktivität benutzt wird.
- $OutPar : ParamID \times T \rightarrow Expr$ definiert das Mapping für den Ausgabe-Parameter, das benutzt wird wenn eine Aktivität beendet wird.
- $Accessor : T_M \rightarrow RecExpr$ definiert die *accessor query* einer MI-Aktivität.
- $Splitter : T_M \rightarrow Expr$ definiert die *splitter query* einer MI-Aktivität.
- $Instance : T_M \rightarrow Expr$ definiert die *instance query* einer MI-Aktivität.
- $Aggregate : T_M \rightarrow RecExpr$ definiert die *aggregate query* einer MI-Aktivität.

Anhang B

Dissertation Online

B.1 Das Projekte DissOnline

Das DFG Projekt *Dissertation Online* wurde von 1998 bis 2000 durchgeführt. Ziel war es, Lösungen für die elektronische Publikation von Hochschulschriften, insbesondere von Dissertationen und Habilitationsschriften, zu entwickeln. Die Ergebnisse des Projektes wurden im Projekt *DissOnline* weitergeführt, das durch die Deutsche Nationalbibliothek betreut wird.

DissOnline.de ist ein Informationssystem, das verschiedene Materialien für Autoren, Bibliotheken, wissenschaftliche Einrichtungen und Verlage anbietet [Deu11]. Zu den Dienstleistungen für Autoren zählen:

- Dokumentvorlagen (Formate PDF, WORD und LaTeX), zur Unterstützung von Autoren.
- Ein Veröffentlichungsvertrag, der als Mustervertrag zwischen Autor und Verlag dienen soll. Der Vertrag ist für die Veröffentlichung von Dissertationen im Buchformat ausgelegt.
- Vorschläge für eine Rechts- und Promotionsordnung.
- Informationsmaterial für Autoren, wenn eine Veröffentlichung bei einem Verlag angestrebt wird.
- Datenformate wie z. B. DiML und XDiML, die für die Verarbeitung von elektronischen Hochschulschriften eingesetzt werden können (siehe auch Abschnitt B.2).
- Eine Liste von Institutionellen Repositorien, auf denen Online-Dissertationen angeboten werden.

Außerdem werden zahlreiche Angebote für Bibliotheken und Verlage bereitgestellt. Dazu zählen Metadatenformate, Datenformate, Dokumentserver, Workflow- und Autorenunterstützung.

Der Workflow - DissOnline

In [Deu11] wird für die Einrichtung eines Geschäftsprozesses für die elektronische Publikation von digitalen Dissertationen folgender Workflow vorgeschlagen, der hier übernommen wird:

> „Von der Abgabe bei der UB bis zur Archivierung [...]
>
> - Der vom Doktoranden erstellte Metadatensatz (Dateneingabe meist durch den Doktoranden, entweder online oder durch Einreichung eines Formulars) wird von einem Bibliothekar kontrolliert bzw. erweitert. Entsprechen alle Angaben den Vorgaben der jeweiligen Bibliothek, wird dieser Metadatensatz auf den Server gelegt. Nach diesen Einträgen kann in der Folge gesucht werden.
> - Wenn das Dokument nicht in dem Format abgegeben wurde, in dem es auch archiviert werden soll, erfolgt ggf.eine Dokument-Konvertierung.
> - Die abgegebenen Originaldateien werden ggf. mit einer digitalen Signatur versehen und für diese Signatur ein Zeitstempel erzeugt. Damit ist der exakte Veröffentlichungszeitpunkt (Zeitpunkt der Einlieferung bei der Universitätsbibliothek) rechtskräftig festgehalten. Die Originale werden zusammen mit der Signatur und dem Zeitstempel auf einem gesonderten Archivrechner gespeichert.
> - Sind aus den Originaldateien Dateien in einem Präsentationsformat zur Veröffentlichung der Dissertation erzeugt worden, können sie ebenfalls, wie die Originale, mit einer digitalen Signatur und einem Zeitstempel versehen werden. Wird das Präsentationsformat durch die Bibliothek erstellt, reicht es aus, diese Datei mit einer digitalen Signatur zu versehen. Es sollte in jedem Fall eine Speicherung auf dem Archivserver vorgenommen werden.
> - Der überprüfte und erweiterte Metadatensatz muss an die Deutsche Nationalbibliothek gemeldet werden. (Siehe dazu: Meldung an die Deutsche Nationalbibliothek)
> - Die Deutsche Nationalbibliothek fungiert als Sammelstelle für elektronische Dissertationen. Dazu werden die elektronischen Volltexte zusammen mit den Metadaten an die Deutsche Nationalbibliothek übermittelt bzw. von dieser eingesammelt.“

Die Arbeitsabläufe beschreiben das grundsätzliche Vorgehen, das an die individuellen Voraussetzungen der einzelnen Bibliotheken angepasst werden sollte [Deu11].

B.2 Beispiel-Dokument: Dissertation im DiML-Format

Für die Beschreibung von Dissertationen wurde das *DiML-Format* entwickelt. In Auflistung B.1 wird die Verwendung von *DiML* an einem verkürztem Beispiel gezeigt. Das Dokument ist wie in Abschnitt 3.1.2 beschrieben, in die drei Bereiche *front, body* und *back* unterteilt.

Im Bereich *front* werden Angaben zum Dokument gemacht. Dazu zählen der Titel, Angaben zum Autor und zu den Gutachtern. Außerdem ist dort die Zusammenfassung in deutsch und englisch enthalten.

Der Abschnitt *body* beschreibt die Kapitelstruktur, Abschnitte, Paragraphen und Medienelemente. Das Dokument in Auflistung B.1 enthält drei Kapitel, die jeweils in Abschnitte unterteilt sind. Kapitel 2 enthält Medienelemente vom Typ *Video* und *Audio*, mit unterschiedlichen Formaten (JPEG und TIFF). In Kapitel 3 werden verschiedene Bilder mit unterschiedlichen Bildformaten beschrieben.

Der Abschnitt *back* enthält ein Abkürzungsverzeichnis, ein Literaturverzeichnis, einen Anhang und die Selbständigkeitserklärung.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE etd SYSTEM "xdiml.dtd">
3 <etd>
4     <front>
5         <title>Titel der Dissertation</title>
6         <submission><strong>Dissertation</strong></submission>
7         <degree>zur Erlangung des akademischen Grades</degree>
8         <degree>Dr. Ing</degree>
9         <major>Eingereicht an der Fankultaet ...</major>
10        <author>
11            Von
12            <br/>
13            <given>Max</given>
14            <surname>Mustermann</surname>
15            geb. 7.12.1970 in Rostock
16        </author>
17        <dean>Prof. Dr. Hans Meyer</dean>
18        <approvals>
19            <name>Prof. Dr. ...</name>
20            <name>Prof Dr. ...</name>
21        </approvals>
22        <date>Tag der muendlichen Pruefung: 21.06.2006</date>
23        <abstract lang="de">
24            <head>Zusammenfassung</head>
25            <p>Zusammenfassung der Arbeit</p>
26        </abstract>
27        <keywords lang="de">
28            <keyword>Keyword 1, Keyword 2, Keyword 3</keyword>
```

```
29          </keywords>
30          <abstract lang="en">
31              <head>Abstract</head>
32              <p>
33                  Summary of the thesis
34              </p>
35          </abstract>
36      </front>
37      <body>
38          <chapter id="kapitel1" label="1">
39              <head>Einleitung</head>
40              <p/>
41              <section>
42                  <head>Idee</head>
43                  <p>Text Abschnitt 1, Kapitel 1</p>
44              </section>
45          </chapter>
46          <chapter id="kapitel2" label="2">
47              <head>Video- und Tonaufnahmen</head>
48              <section>
49                  <head>Eine Tonaufnahme</head>
50                  <p>
51                      <mm file="aufnahme_1.wav" format="WAV"/>
52                      Ein Audio-Element
53                  </p>
54              </section>
55              <section>
56                  <head>Videoaufnahmen</head>
57                  <p>Videos</p>
58                  <p>
59                      <mm file="film_1.avi" format="AVI"/>
60                      Ein Video-Element
61                  </p>
62                  <p>
63                      <mm file="film_2.avi" format="AVI"/>
64                      Ein Video-Element
65                  </p>
66              </section>
67          </chapter>
68          <chapter id="kapitel3" label="3">
69              <head>Bilder von Landschaften und Tieren</head>
70              <section>
71                  <head>Blumen</head>
72                  <p><table frame="all" id="tab1" orient="port">
73                      <caption>Blume mit einem Schmetterling</caption>
74                      <tgroup align="left" char="" charoff="50" cols="1
                            ">
75                          <colspec colname="1" colnum="1"/>
```

```
76                          <tbody valign="top">
77                              <row>
78                                  <entry morerows="0" rotate="0" valign
                                        ="top">
79                                      <p><mm file="blume_1.jpg" id="
                                            img1" format="JPEG"/></p>
80                                  </entry>
81                              </row></tbody></tgroup>
82                  </table></p>
83                  <p><table frame="all" id="tab2" orient="port">
84                      <caption>Blume mit einer Hummel</caption>
85                      <tgroup align="left" char="" charoff="50" cols="1
                            ">
86                          <colspec colname="1" colnum="1"/>
87                          <tbody valign="top">
88                              <row>
89                                  <entry morerows="0" rotate="0" valign
                                        ="top">
90                                      <p><mm file="blume_2.tiff" id="
                                            img2" format="TIFF"/></p>
91                                  </entry>
92                              </row></tbody></tgroup>
93                  </table></p>
94              </section>
95              <section>
96                  <head>Landschaften</head>
97                  <p>Eine Hafenansicht</p>
98                  <p><table frame="all" id="tab3" orient="port">
99                      <caption>Stadthafen Rostock</caption>
100                     <tgroup align="left" char="" charoff="50" cols="1
                            ">
101                         <colspec colname="1" colnum="1"/>
102                         <tbody valign="top">
103                             <row>
104                                 <entry morerows="0" rotate="0" valign
                                        ="top">
105                                     <p><mm file="hafen.jpg" id="img3"
                                            format="JPEG"/></p>
106                                 </entry>
107                             </row></tbody></tgroup>
108                 </table></p>
109             </section>
110         </chapter>
111     </body>
112     <back>
113         <acknowledgement><head>Acknowledgements</head><p>
                Acknowledgement</p></acknowledgement>
114         <abbreviation><head>Abkuerzungsverzeichnis</head></
```

```
            abbreviation>
115         <bibliography><head>Literaturverzeichnis</head></bibliography
            >
116         <appendix><head>Anhang</head></appendix>
117         <declaration><head>Selbstaendigkeitserklaerung</head><date/><
            /declaration>
118     </back>
119 </etd>
```

Auflistung B.1: Beispiel für eine Dissertation im DiML-Format

B.3 Beispiel-Dokument: Dissertation als DocBook

Das *DocBook*-Format ist ein Format, mit dem Dokumente medienneutral und plattformunabhängig publiziert werden können [Sch04]. Mit dem Format kann die Struktur von Dissertationen abgebildet werden. Im Gegensatz zum *DiML*-Format sind aber keine spezifischen Elemente für Disserationen vorgesehen (z. B. Elemente für Abkürzungsverzeichnis, Literaturverzeichnis oder Selbständigkeitserklärung). Medienelemente können ebenfalls eingebunden werden, wofür eine Reihe von Elementen zur Verfügung stehen. Auflistung B.2 zeigt ein Beispiel, wie eine Dissertation mit dem DocBook-Format dargestellt werden kann.

Das Dokument setzt sich aus dem Element *info* und mehreren Kapitel-Elementen zusammen. Kapitel sind wiederum in Abschnitte unterteilt. Das erste Kapitel besteht aus einem Abschnitt mit einer Tonaufnahme. Das zweite Kapitel ist in drei Abschnitte unterteilt, die Ton- und Video-Elemente enthalten. Das letzte Kapitel ist in zwei Abschnitte unterteilt, die jeweils Bild-Elemente enthalten.

```
 1 <?xml version='1.0'?>
 2 <!DOCTYPE book PUBLIC "-//OASIS//DTD␣DocBook␣V5.0//EN" "http://
       docbook.org/xml/5.0/dtd/docbook.dtd">
 3 <book xmlns="http://docbook.org/ns/docbook"
 4       xmlns:xlink="http://www.w3.org/1999/xlink" version="5.0">
 5       <info>
 6           <title>Dissertation</title>
 7           <author>
 8               <personname>
 9                   <honorific>Herr</honorific>
10                   <firstname>Max</firstname>
11                   <surname>Mustermann</surname>
12               </personname>
13               <affiliation>
14                   <shortaffil>Universitaet Rostock</shortaffil>
15                   <orgname>Universitaet Rostock</orgname>
16               </affiliation>
```

```
17              <address>
18                  <city>Rostock</city>
19                  <street>Albert-Einstein-Strasse 22</street>
20                  <postcode>18055 Rostock</postcode>
21                  <country>Deutschland</country>
22              </address>
23              <email>max.mustermann@uni-rostock.de</email>
24
25
26          </author>
27      </info>
28        <chapter>
29              <title>Einleitung</title>
30              <sect1>
31                  <title>Idee</title>
32                  <para>Text Abschnitt 1, Kapitel 1</para>
33              </sect1>
34          </chapter>
35
36          <chapter>
37              <title>Bilder von Landschaften und Tieren</title>
38              <sect1>
39                  <title>Blumen</title>
40                  <para>Blume mit Schmetterling
41                      <mediaobject>
42                          <imageobject>
43                              <imagedata width="6in" fileref="blume_1
                                    .jpg" format="JPG"/>
44                          </imageobject>
45                          <caption>Schmetterling auf Blume</caption>
46                      </mediaobject>
47                  </para>
48                  <para>Blume mit Hummel
49                      <mediaobject>
50                          <imageobject>
51                              <imagedata width="6in" fileref="blume_2.
                                    jpg" format="JPG"/>
52                          </imageobject>
53                          <caption>Hummel auf Blume</caption>
54                      </mediaobject>
55                  </para>
56              </sect1>
57              <sect1>
58                  <title>Landschaften</title>
59                  <para>Eine Hafenansicht
60                      <mediaobject>
61                          <imageobject>
62                              <imagedata align="right" width="6in"
```

```
                                        fileref="hafen.jpg" format="JPG"/>
63                                  </imageobject>
64                              </mediaobject>
65                          </para>
66                      </sect1>
67                  </chapter>
68 </book>
```

Auflistung B.2: Beispiel für eine Dissertation im DocBook-Format

B.4 Beispiel-Prozess: Dissertation veröffentlichen

Das folgende Beispiel zeigt an einem Prozess für die Veröffentlichung von Dissertationen, wie die Konzepte *tx+YAWL* und *FlexY* angewendet werden. Dafür wird zunächst das Prozessmodell in Abbildung B.1 und die darin enthaltenen Aktivitäten beschrieben. Anschließend wird gezeigt, wie das Konzept der externen Variablen (siehe Kapitel 5 und 6) genutzt wird. Die Anwendung externer Variablen wird außerdem im Zusammenhang mit der flexiblen Komposition von Prozessfragmenten vorgestellt.

Der in Abbildung B.1 dargestellte Publikationsprozess setzt im Wesentlichen Aufgaben um, die für die Bereitstellung elektronischer Pflichtexemplare digitaler Dissertationen oder Habilitationen auf einem Dokumentenserver notwendig sind. Der Prozess orientiert sich an der prototypischen Umsetzung des Publikationsprozesses für Dissertationen, die in Zusammenarbeit mit der Universitätsbibliothek Rostock erstellt wurde (siehe Abschnitt 9.1.1).

- Zunächst muss der Autor einen Nutzerzugang für den Dokumentenserver beantragen (Aktivität: *Autor anlegen*). Dafür wird eine Eingabemaske bereitgestellt, die alle relevanten Informationen abfragt.
- In einem zweiten Schritt muss der Nutzer einen Autorendatensatz anlegen, der alle personenbezogenen Daten enthält, die für die Publikation einer Dissertation notwendig sind (Aktivität: *Autor bearbeiten*).
- Danach kann der Nutzer den Abgabeprozess starten, indem die Dissertation angemeldet wird. In diesem Schritt wird eine URN erzeugt, die der Nutzer händisch in die Arbeit einfügen muss (Aktivität: *Dokument anlegen*).
- Außerdem müssen vom Nutzer bibliographische Daten zur Arbeit in ein Formular eingetragen werden (Aktivität: *Metadaten bearbeiten*). Dazu zählen beispielsweise Schlagwörter und die Angabe einer DDC-Kategorie.
- Eine generierte PDF/A1-b Datei kann dann zusammen mit zusätzlichen Dateien, wie z. B. Anhängen mit Messreihen oder Statistiken auf den Dokumentenserver geladen werden (Aktivität: *Derivate hochladen*).

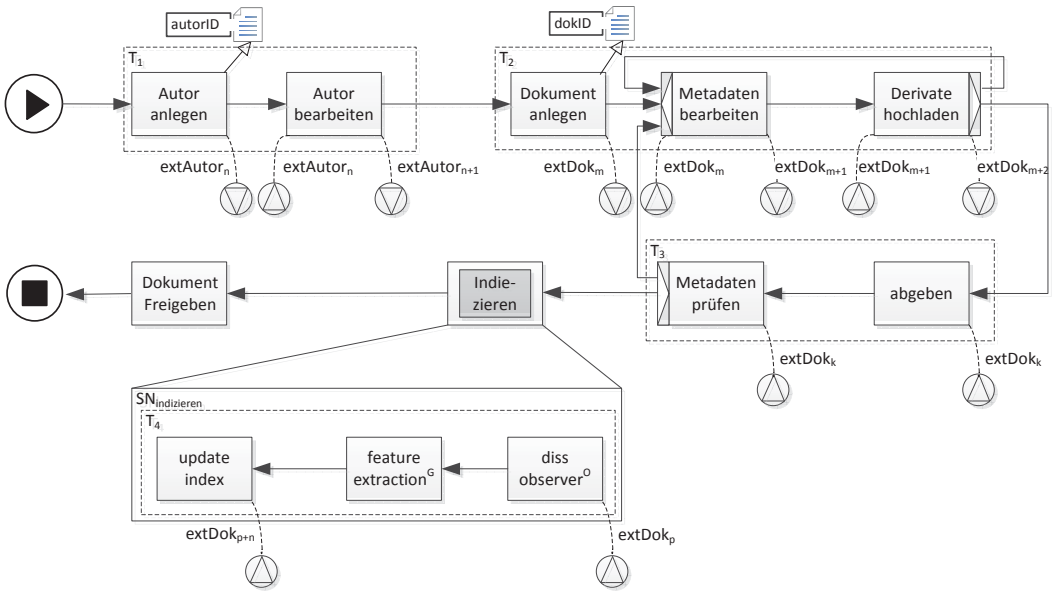


Abbildung B.1: Beispielprozess für den Publikationsprozess von Dissertationen

- Am Ende des Prozesses wird ein Abgabeformular erzeugt, das zusammen mit den Ausgedruckten Pflichtexemplaren in der Universitätsbibliothek abzugeben ist (Aktivität: *abgeben*).
- In einem letzten manuellen Schritt wird die Arbeit von einem Bibliothekar überprüft und freigeschaltet (Aktivität: *Metadaten prüfen*) oder für eine Überarbeitung an den Autor übergeben.
- Dann wird die Arbeit in der komplexen Aktivität *Indizieren* für die Suche aufbereitet und indiziert. Dafür wird der *FlexY*-Ansatz verwendet (siehe Kapitel 7). Die Observer-Aktivität *diss observer* konfiguriert in Abhängigkeit des Dokumentes *dissDS* die Generator-Aktivität *feature extraction*. Die Prozessbausteine, die für die Indizierung genutzt werden können, werden in den Abschnitten 3.3.1 und 7 beschrieben. Anschließend wird das Dokument indiziert.

Externe Variablen

Für die Umsetzung werden im Prozess unter anderem zwei externe Variablen verwendet. Die externe Variable *extAutor* hält einen Autorendatensatz, der in einer externen Quelle verwaltet wird. Die zweite externe Variable *extDok* hält den Dokumentdatensatz, der ebenfalls in einer externen Datenquelle gespeichert wird. Für die Identifizierung einzelner Datensätze in einer externen Datenquelle wird ein Schlüsselattribut benötigt. Das Schlüsselattribut für den aktuell bearbeiteten Autor wird in der globalen Netzvariablen *autorID* abgespeichert. Das Schlüsselattribut für das aktuell bearbeitete Dokument wird in der globalen Netzvariable *dokID* abgespeichert.

Wie in Kapitel 6 beschrieben, sollte der Zugriff auf externe Variablen nur innerhalb transaktionaler Sphären (siehe Abschnitt 6.3.3) erfolgen. Hierfür werden im Prozess die vier transaktionalen Sphären T_1 bis T_4 definiert.

- **Sphäre** T_1 umschließt die Aktivitäten zum Anlegen und Bearbeiten eines Autorendatensatzes. Dabei wird für beide Aktivitäten die Aktivitätsvariable *varAutor* angelegt, die an die externe Variable *extAutor* gebunden ist.
- **Sphäre** T_2 umschließt die Aktivitäten zum Anlegen und Bearbeiten eines Dokumentendatensatzes. Dazu zählen neben der Angabe von Metadaten auch das Bereitstellen von Derivaten. Innerhalb der Transaktion wird für jede Aktivität jeweils eine eigene Aktivitätsvariable *varDok* angelegt, die an die externe Variable *extDok* gebunden ist.
- **Sphäre** T_3 umschließt Aktivitäten, die den Abgabeprozess für ein Dokument beschreiben. Für den Abgabeprozess wird ebenfalls die Aktivitätsvariable *varDok* benötigt, die an die externe Variable *extDok* gebunden ist. Sie wird für die beiden enthaltenen Aktivitäten eingeführt.
- **Sphäre** T_4 ist innerhalb der komplexen Aktivität *Indizieren* definiert. Dort umschließt sie drei Aktivitäten, die für die Indizierung eines Dokumentes benötigt werden. Da Observer-Aktivitäten mit Hilfe externer Variablen die Konfiguration von Generator-Aktivitäten umsetzen, wird hier ein lesender Zugriff auf das externe Dokument benötigt. Dafür wird die Aktivitätsvariable *varDok* benötigt, die an die externe Variable *extDok* gebunden ist. Die Variable wird für die Aktivitäten *diss observer* und *update index* benötigt.

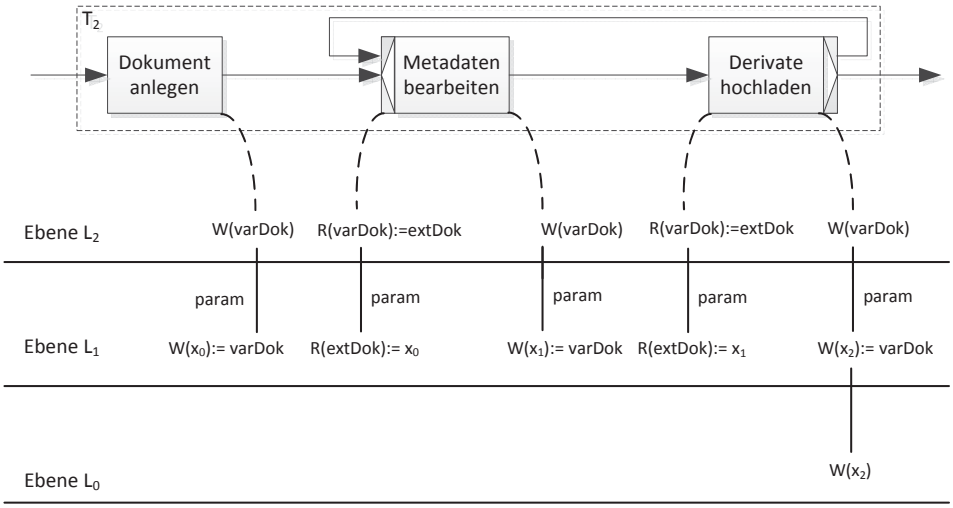


Abbildung B.2: Transaktionale Sphäre T_2: Dokument anlegen und bearbeiten

Am Beispiel der Sphäre T_2 wird im Folgenden die Nutzung des 4-Ebenen-Transaktionsmodells verdeutlicht. Abbildung B.2 zeigt, wie der Datenzugriff auf externe Variablen in einer transaktionalen Sphäre entsprechend dem Mehrebenen-Transaktionsmodell *tx+YAWL* umgesetzt wird (siehe Kapitel 6).

Aktivität *Dokument anlegen*:

- Es wird ein Dokument erzeugt und in die Aktivitätsvariable *varDok* geschrieben.
- Die Aktivitätsvariable *varDok* ist an die externe Variable *extDok* gebunden.
- Durch die Operation $W(varDok)$ wird in Ebene L_2 der Wert von *varDok* an die externe Variable übergeben.
- In Ebene L_1 wird eine neue, lokale Version x_0 angelegt, die nicht in die externe Datenquelle geschrieben wird (siehe Ebene L_0).

Aktivität *Metadaten bearbeiten*:

- Die Aktivität greift auf die externe Variable *extDok* zu, indem in Ebene L_2 ein Lesezugriff durch die Aktivitätsvariable *varDok* stattfindet.
- In Ebene L_1 wird die aktuelle Version x_0 bereitgestellt, die lokal zur Verfügung steht.
- Es wird eine neue, lokale Version x_1 in Ebene L_1 angelegt, die nicht an die Datenquelle übergeben wird.

Aktivität *Derivate hochladen*:

- Die Aktivität greift auf die externe Variable *extDok* zu, indem in Ebene L_2 ein Lesezugriff durch die Aktivitätsvariable *varDok* stattfindet.
- In Ebene L_1 wird die aktuelle Version x_1 bereitgestellt, die lokal zur Verfügung steht.
- Es wird eine neue, lokale Version x_2 in Ebene L_1 angelegt.
- In Ebene L_0 wird die lokale Version x_2 der externen Variable in die externe Datenquelle geschrieben, weil die transaktionale Sphäre beendet ist.

Flexible Generierung von Prozessen

Innerhalb der Sphäre T_4 wird die externe Variable *extDok* von der Observer-Aktivität *diss observer*^{O} benötigt. Sie wird für die Auswertung verschiedener Matching-Regeln verwendet. Die Dokumentversion wird für diesen Fall direkt aus der externen Datenquelle gelesen, weil für diese Sphäre noch keine lokale Version existiert.

Im Beispiel werden drei Regeln für die Aktivierung von Bricklets eingeführt:

- $r^M_{1,1}$=`add(feat.extr.,`b_6`,"exists(//mm[@format='JPEG' or @format='GIF'])")`
- $r^M_{1,2}$=`add(feat.extr.,`$\{b_1,b_2,b_3,b_4,b_5\}$`,"exists(//mm[@format='AVI' or @format='WMV'])")`
- $r^M_{1,3}$=`add(feat.extr.,`$\{b_4,b_5\}$`,"exists(//p[not(child::mm)])")`

Die Bricklets werden für den Generator mit der ID $feat.extr.$ aktiviert ($feat.extr.$ entspricht dem Generator *feature extraction*^{G}).

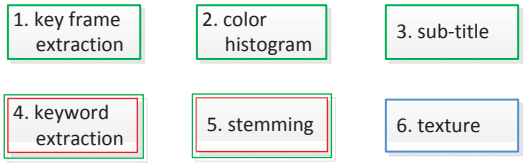


Abbildung B.3: Bricklets für die Indizierung einer Dissertation

Die verwendeten Bricklets entsprechen den Bricklets aus Abbildung B.3[1]. Wie mit Hilfe der Generator-Aktivität $feature\ extraction^G$ ein Prozessfragment generiert wird, wird in Abschnitt 7.4 beschrieben. Die Konfiguration der Observer- und Generator-Aktivitäten wird in Abschnitt 8.3.2 beschrieben.

[1] Dasselbe Beispiel wird auch in Abschnitt 7.2.2 verwendet.