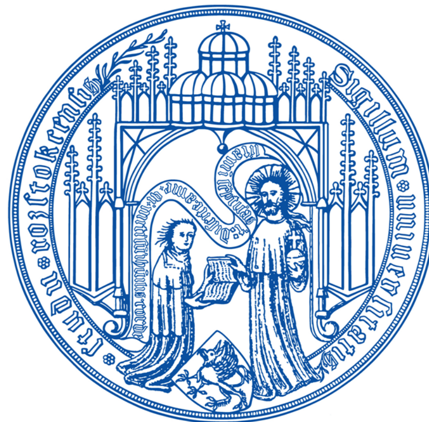
Datentypgetriebene Dynamische Komposition von Prozessbeschreibungen

Diplomarbeit

Universität Rostock
Fakultät für Informatik und Elektrotechnik
Institut für Informatik



vorgelegt von:	Marcel Paleit
Matrikelnummer:	5203648.
geboren am:	20.11.1984 in Rostock
Erstgutachter:	Dr.-Ing. Holger Meyer
Zweitgutachter:	Prof. Dr.-Ing. habil. Peter Forbrig
Betreuer:	Dr.-Ing. Holger Meyer
	Dipl.-Inf. Sebastian Schick
Abgabedatum:	19. April 2011

Kurzfassung

In dieser Diplomarbeit werden Ansätze für die effiziente Handhabung von Prozessvarianten durch ein Workflow-Managementsystem untersucht. Als Grundlage dient ein Publikationsprozess für die Erstellung und Veröffentlichung von Multimediadokumenten in digaitalen Bibliotheken, dessen Modellierung und Verwaltung einer Vielzahl von Anforderungen genügen muss. Die besten Ideen aus den untersuchten Ansätzen, die den Anforderungen genügen, werden in einem neuen Konzept vereint. Es stützt sich auf die dynamische Komposition von Prozessbeschreibungen. Anhand eines geeigneten Workflow-Managementsystems wird die Realisierbarkeit der Konzeption nachgewiesen.

Abstract

This diploma thesis examines approaches for the efficient handling of process variants by a workflow management system. The foundation is a publication process for the creation and publishment of a multimedia document in digital libraries. Its modeling and management has to satisfy a plurality of requirements. Using the best ideas from the examined approaches satisfying the requirements, a new concept is created. It is based on the dynamic composition of process description. A proof-of.concept demonstrates the feasibility on the example of an appropriate workflow management system.

CR Classification

H.3.3 Information Search and Retrieval
H.3.6 Library Automation
H.4.1 Office Automation - Workflow Management

Keywords

multimedia document, publication process, workflow management, YAWL, process variants

Inhaltsverzeichnis

1 Einleitung **2**

1.1 Motivation . . . 2

1.2 Zielstellung . . . 3

1.3 Aufbau der Arbeit . . . 3

2 Multimediadokumente und digitale Bibliotheken **5**

2.1 Das digitale Dokument . . . 5

2.2 Das Multimediadokument . . . 6

2.3 Das Dokumentenmodell . . . 7

2.4 Metadaten . . . 8

2.5 Content Management . . . 9

2.6 Die digitale Bibliothek . . . 10

2.7 Der Publikationsprozess . . . 11

3 Der Publikationsprozess als Workflow **14**

3.1 Serive-orientierte Architekturen . . . 14

3.2 Workflow Management . . . 16

3.3 Workflow Patterns . . . 18

3.4 Das Modellierungsproblem: Prozessvarianten . . . 20

3.5 Anforderungsanalyse . . . 22

4 Modellierungsansätze für den Publikationsprozess **26**

4.1 Der deklarative Ansatz . . . 26

4.1.1 Deklarative gegenüber Imperativer Modellierung . . . 26

4.1.2 Die deklarative Sprache ConDec . . . 28

4.1.3 Beurteilung des deklarativen Ansatzes . . . 30

4.2 Der PROVOP Ansatz . . . 31
4.2.1 Modellierung von Prozessvarianten in PROVOP . . . 31
4.2.2 Prozesskonfiguration in PROVOP . . . 33
4.2.3 Beurteilung des Ansatzes . . . 34
4.3 Configurable YAWL . . . 35
4.3.1 YAWL - Die Prozessmodellierungssprache . . . 35
4.3.2 Die Prozesskonfiguration . . . 38
4.3.3 Beurteilung der Prozesskonfiguration . . . 44
4.4 Die Unterspezifikation . . . 44
4.4.1 Realisierung der Unterspezifikation . . . 45
4.4.2 Pockets of Flexibility . . . 46
4.4.3 Beurteilung der Unterspezifikation . . . 48
4.5 Kombination von Unterspezifikation und Prozesskonfiguration . . . 49
4.5.1 Der Worklet Service . . . 49
4.5.2 Beurteilung der Kombination . . . 50

5 Konzept für die Umsetzung des Publikationsprozesses 52
5.1 Das Konzept . . . 53
5.1.1 Validierung von Regelmengen . . . 55
5.1.2 Konfluenz von Regelmengen . . . 58
5.1.3 Komposition von Aktivitäten . . . 59
5.2 Umsetzbarkeit der Constraints mit YAWL . . . 60

6 Dynamische Komposition von Prozessbeschreibungen mit YAWL 63
6.1 YAWL – Das Workflow Management System . . . 63
6.2 Custom YAWL Services . . . 65
6.3 Umsetzung des Konzepts . . . 67
6.3.1 Bestimmung der Komponenten . . . 68
6.3.2 Implementation des Composition Service . . . 70
6.3.3 Die Klasse „Merger“ . . . 71

7 Schlussbetrachtung 80
7.1 Zusammenfassung . . . 80
7.2 Ausblick . . . 81

Abkürzungsverzeichnis **A**

Literatur **B**

Selbstständigkeitserklärung **E**

Thesen **G**

Abbildungsverzeichnis

2.1 Dokumentenmodell eines Multimediadokuments 8
2.2 Klassifikation von Metadaten 9
2.3 Phasen eines Publikationsprozesses 11

3.1 Grundkonzept einer Service-orientierten Architektur 15
3.2 Das Workflow-Referenz Modell 18
3.3 Prozessvarianten der Aggregation 21
3.4 Process Life Cycle 23

4.1 Vergleich zwischen imperativer und deklariver Modellierung 27
4.2 Grafische Notation für $\Box(A \longrightarrow \Diamond B)$ 29
4.3 Ein einfaches ConDec-Modell 30
4.4 Beurteilung des deklarativen Ansatzes 30
4.5 Prozessvarianten in PROVOP 32
4.6 Modellierung von Kontext in PROVOP 34
4.7 Beurteilung des PROVOP Ansatzes 35
4.8 Symbole für den Kontrollfluss in YAWL 37
4.9 Input- und Output Ports einer YAWL-Task 40
4.10 Konfigurierter YAWL Prozess 41
4.11 Auswirkung der Konfiguration auf YAWL-Tasks 43
4.12 Beurteilung von C-YAWL 44
4.13 Flexibilität durch Unterspezifikation 45
4.14 Spezifikation einer Pocket of Flexibility 46
4.15 Kompositionsmöglichkeiten einer Build-Aktivität 47
4.16 Beurteilung der Unterspezifikation 48

5.1 Prozess mit vier Konfigurationspunkten 52

5.2 Gerichteter Graph mit Zyklus 57
5.3 Konfluenz von Regelmengen 58
5.4 Das Milestone-Pattern 59
5.5 Sequenzielle und parallele Modellierung mit YAWL 61
5.6 Modellierung des Milestone-Patterns mit YAWL 62

6.1 Services eines YAWL Systems 64
6.2 Kommunikation zwischen YAWL Engine und Composition Service 67
6.3 Dokumentenmodell einer Vorlesung 69
6.4 Klassendiagramm: Composition Service 70
6.5 YAWL-Spezifikationen in XML-Syntax 71
6.6 Komponierte YAWL-Spezifikation in XML-Syntax 73
6.7 Parallele Komposition: Verbinden eines Fragments mit AND-Split 76
6.8 Parallele Komposition: Verbinden eines Fragments mit AND-Join 77
6.9 Sequenzielle Komposition: Direktes Verbinden zweier Fragmente 78

Kapitel 1

Einleitung

1.1 Motivation

Am Lehrstuhl für Datenbank- und Informationssysteme der Universität Rostock wird ein Framework entwickelt, welches Autoren bei der Erstellung und Veröffentlichung von komplex strukturierten multimedialen Dokumenten in digitalen Bibliotheken unterstützt. Im Vordergrund steht hierbei die Dokumentierung des Ablaufs als Geschäftsprozess und die anschließende Automatisierung des Publikationsprozesses durch ein Workflow-Managementsystem (WfMS).
Hierzu wird ein Modell des Geschäftsprozesses angefertigt, welches von dem WfMS instanziiert wird. Das System übernimmt die Ablaufsteuerung der Teilaktivitäten in der modellierten Reihenfolge, weist den am Prozess beteiligten Personen die Aktivitäten automatisch zu und stellt ihnen die erforderlichen Daten und Anwendungen für die Abarbeitung zur Verfügung.
Viele Unternehmen, öffentliche Einrichtungen und auch der medizinische Sektor haben bereits die Vorteile von Geschäftsprozessen erkannt. Jedoch werden Prozesse heutzutage in vielen parallel existierenden Varianten definiert, die Anpassungen an unterschiedliche Rahmenbedingungen darstellen. Die Zusammensetzung eines Multimediadokuments ist eine derartige Bedingung für den Publikationsprozess. Reine Textdokumente werden anders durch den Prozess bearbeitet, als ein Dokument mit mehreren Bildern und Videos. Es gibt folglich keinen Publikationsprozess, der für jedes Multimediadokument allgemeingültig ist. Vielmehr existieren eine Vielzahl von Prozessvarianten, welche die Veröffentlichung in Abhängigkeit von den Dokumentbestandteilen steuern.

1.2 Zielstellung

Die effiziente Handhabung von Prozessvarianten ist eine herausfordernde Aufgabe für heutige Workflow-Managementsysteme. Das Ziel dieser Arbeit ist die Untersuchung existierender Ansätze für die Modellierung und Verwaltung von Prozessvarianten in einem WfMS. Die Ansätze werden anhand von selbst aufgestellten Anforderungen verglichen. Die Definition der Anforderungen basiert auf den Ansprüchen, die an eine effizienten Modellierung und Verwaltung des Publikationsprozesses gestellt werden. In der Folge wird ein Konzept präsentiert, welches die besten Ideen der untersuchten Methoden vereint und ein Prototyp vorgestellt, der dieses Konzept mit dem WfMS YAWL (*Yet Another Workflow Language*) umsetzt.

1.3 Aufbau der Arbeit

- Kapitel 2 führt den Publikationsprozess für Multimediadokumente ein. Es beschreibt die einzelnen Phasen des Prozesses und erläutert vorab die Begrifflichkeiten, wie das Multimediadokument, Metadaten oder die digitale Bibliothek, welche mit der Veröffentlichung in Verbindung stehen.

- Kapitel 3 gibt eine Einführung in die Themen „Service-orientierte Architektur“ ,sowie „Workflow Management“ und definiert in diesem Zusammenhang eine Vielzahl von Begriffen für den weiteren Verlauf der Arbeit. Weiter wird das Problem der unterschiedlichen Prozessvarianten am Beispiel des Publikationsprozesses beschrieben. Abschließend erfolgt eine Anforderungsanalyse, welche eine Vielzahl von Forderungen an die Modellierung und Verwaltung des Prozesses in einem WfMS stellt.

- Kapitel 4 stellt mehrere konkrete Ansätze für die effektive Handhabung von Prozessvarianten vor. Jeder Ansatz wird abschließend auf Grundlage der zuvor aufgestellten Anforderungen beurteilt. Das Kapitel endet mit der Aufstellung von zwei Möglichkeiten, die für die Umsetzung des Publikationsprozesses zur Auswahl stehen.

- Kapitel 5 präsentiert in Anlehnung an die untersuchten Ansätze ein theoretisches Konzept, mit dem der Publikationsprozess entsprechend den Anforderungen umgesetzt werden kann. Im Vordergrund steht hierbei die dynamische Komposition von Prozessbeschreibungen anhand von vordefinierten Regeln. Weiter wird überprüft,

ob die Verknüpfung von Prozessbeschreibungen auf diese Weise mit der Modellierungssprache YAWL umsetzbar ist.

- Kapitel 6 beschreibt die Umsetzung des in Kapitel 5 vorgestellten Konzepts mit dem WfMS YAWL. Vorab wird das System YAWL erläutert. Im Anschluss folgt eine Einführung in die Implementation eines eigenentwickelten Dienstes für YAWL. Dieser realisiert das Konzept für die Umsetzung des Publikationsprozesses. Der Fokus liegt hier auf der dynamischen Komposition von YAWL-Prozessmodellen.
- Kapitel 7 fasst die wesentlichen Aspekte dieser Arbeit zusammen. Zusätzlich werden weiterführende Gedanken zum präsentierten Konzept dargestellt.

Kapitel 2

Multimediadokumente und digitale Bibliotheken

In diesem Kapitel werden die Hintergründe der Diplomarbeit vorgestellt. Im Fokus steht ein Publikationsprozess für Multimediadokumente. Er umfasst alle Phasen von der Erstellung eines Multimediadokuments durch den Autor bis zu seiner Speicherung und Verwaltung in einer digitalen Bibliothek. In den nachfolgenden Abschnitten werden kurz Begriffe und Sachverhalte erläutert, die mit dem Publikationsprozess verknüpft sind. Sie bilden die Voraussetzung für das Verständnis nachfolgender Kapitel.

2.1 Das digitale Dokument

> „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 genutzt werden kann." [EF00]

Analoge Medien werden zunehmend durch digitale Dokumente verdrängt oder ergänzt. Es gibt eine Vielzahl von Gründen, die diesen Wandel begründen. Nach [EF00] sind einige wichtige die Folgenden:

1. **Speicherkapazität**: Bücher füllen in Bibliotheken unzählige Regale. Gleicher Inhalt hat hingegen auf einigen wenigen DVDs Platz.

2. **Weltweite Verfügbarkeit**: Dokumente müssen auf Grund ihrer geographischen

Lage nicht mehr reproduziert werden, sondern sind über Netzwerke überall verfügbar.

3. **Gleichzeitige Nutzung des selben Exemplars**: Sofern ein elektronisches Dokument auf einem Rechner angeboten wird, der es laden kann, haben hunderte Nutzer die Möglichkeit das Dokument gleichzeitig zu nutzen.

4. **Die Integration verschiedener Medien** bezeichnet die Verknüpfung von Texten, Grafiken, Videos oder Audio-Clips in einem Dokument.

5. **Erschließbarkeit**: Digitale Dokumente können weitgehender erschlossen werden als konventionelle Dokumente. Beispielsweise es möglich, den Inhalt basierend auf der Dokumentenstruktur zu selektieren

Das Sammeln, Speichern, Verteilen, Finden, Archivieren und Nutzen von digitalen Dokumenten ist Aufgabe einer *digitalen Bibliothek*. Der Grund für den Einsatz einer digitalen Bibliothek im Publikationsprozess wird in Abschnitt 2.6 erläutert.

2.2 Das Multimediadokument

Digitale Dokumente „sind zunehmend keine statischen, sequenziell organisierten Objekte mehr. Vielmehr sind es komplex organisierte Ansammlungen von dynamischen Objekten“ [RR01], die zusätzlich Verweisungsbeziehungen untereinander besitzen können. Beispielsweise ermöglichen Hyperlinks und Menüs dem Nutzer eigenständig durch ein Dokument zu navigieren.
Das Resultat dieser Entwicklung sind Multimediadokumente, die neben Textdaten zusätzlich Daten anderer Medien enthalten können. Diese Daten werden im Folgenden als Medienobjekte bezeichnet. Beispiele für Medienobjekte sind unter anderem:

- Unstrukturierte Texte
- Strukturierte und Formatierte Dokumente
- 2D- und 3D-Bilder
- 2D- und 3D-Animationen
- Audio-Clips

- Videos

Alle Medienobjekte können in verschiedenen Dateiformaten vorliegen. Ein Video kann z. B. das häufig verwendete Containerformat *AVI (Audio Video Interleave)*[1] oder das unter Windows übliche *WMV (Windows Media Video)*[2] Format nutzen. Ferner können formatierte Dokumente in einem der vielen MS-Word-Formate vorliegen oder unstrukturierte Texte unterschiedliche Zeichensätze verwenden. [RR01]
Die Verarbeitung von gleichen Medienobjekten im Publikationsprozess unterscheidet sich in Abhängigkeit vom genutzten Format. Folglich macht die Komplexität der Zusammensetzung das Veröffentlichen von Multimediadokumenten in digitalen Bibliotheken zu einer Herausforderung.

2.3 Das Dokumentenmodell

Anders als bei Suchanfragen in relationalen Datenbanken ist es in digitalen Bibliotheken erwünscht, dass neben identischen Suchtreffern (Dokumente) auch besonders ähnliche Dokumente gefunden werden. Die Suchanfrage nach Dokumenten mit dem Titel „Raubkatzen“ soll dem Nutzer zusätzlich Ergebnisse über Publikationen zu Tigern oder Geparden liefern, da diese für ihn ebenfalls interessant sein könnten. Das System muss folglich mehr über ein Dokument wissen, als nur den Titel und Autor.

Zu diesem Zweck wird ein Multimediadokument beim Einstellen in die Datenbank in nicht komplexe Medienobjekte zerlegt. Im Anschluss erfolgt eine Extraktion von Feature-Werten, die in Form von Metadaten (siehe Abschnitt 2.4) zusätzlich gespeichert werden. Feature-Werte sind von Medium zu Medium unterschiedlich. Texte enthalten Schlüsselwörter, Bilder bestimmte Texturen und Farben und Audio-Clips haben eine Melodie.
Der Vorgang der Informationsgewinnung aus Medienobjekten wird *Information Retrieval* [Sch02] genannt.

Damit das Multimediadokument wieder zusammengesetzt werden kann, müssen zusätzlich zu jedem Dokument Strukturdaten gespeichert werden. Laut [EF00] spezifiziert ein *Dokumentenmodell* die logische Struktur eines Dokuments. Es kann grafisch als Baum-

[1] http://www.e-teaching.org/glossar/avi
[2] http://lehrerfortbildung-bw.de/werkstatt/video/formate/

struktur dargestellt werden. Abbildung 2.1 zeigt einen frei erfundenen Beispielbaum für ein Multimediadokument.

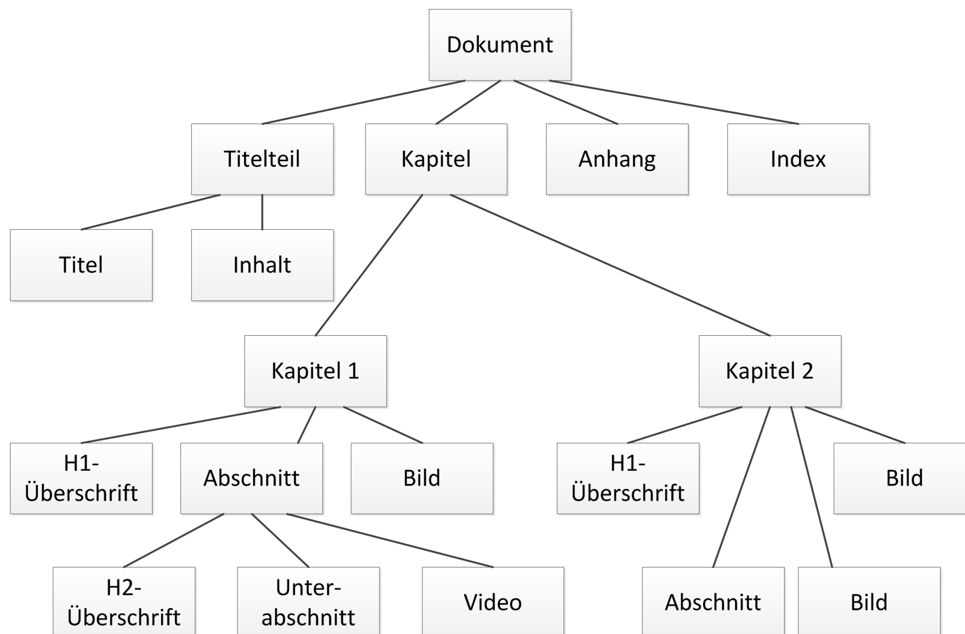


Abbildung 2.1: Dokumentenmodell eines Multimediadokuments, vgl. [EF00]

Es gibt viele Anforderungen an Multimedia-Dokumentenmodelle. Dazu gehören u. a. die Spezifikation eines zeitlichen und räumlichen Modells, sowie die Beschreibung der Interaktionen zwischen Medienobjekten. Ein wesentlicher Aspekt ist zudem die Wiederverwendbarkeit und Anpassbarkeit der Dokumentbestandteile. In [Sch05] beschreibt S. Schick die Anforderungen im Detail.

2.4 Metadaten

Metadaten sind „Daten über Daten“. Sie beschreiben folglich andere Daten und bilden die Grundlage für Suchanfragen in digitalen Bibliotheken.
Inhaltsbezogene Metadaten, wie Farbhistogramme, die Auflösung von Bildern oder die Spieldauer von Filmen, können automatisch aus Dokumenten extrahiert werden.
Hingegen ist die automatische Gewinnung von *inhaltsbeschreibenden* Metadaten schwierig. Sie müssen häufig aufwendig durch Personen manuell erzeugt werden. Zu inhaltsbeschreibenden Metadaten zählen unter anderem Angaben über den Autor, Titel, Verlag oder das Veröffentlichungsdatum eines Dokuments.

Weiterhin existieren *inhaltsunabhängige* Metadaten. Das Speicherformat eines Objektes zählt zu dieser Kategorie von Daten. Abbildung 2.2 fasst die Vielfalt der Metadaten für Multimediadokumente zusammen.

Klassifikation	Metadaten	Beispiele
inhaltbeschreibend (interpretierend)	kontextbeschreibend	Indexvokabular, Ontologien, Thesaurus
	kontextbezogen	Identifikation, Raum- und Zeitdaten
	objektbeschreibend nicht textuell	Gegenstände, Personen, Eindrücke, Aktivitäten, Titel
	objektbeschreibend textuell	Annotation, Drehbuch, Untertitel
inhaltsbezogen (nicht interpretierend)	Feature	Farbverteilung, Textur, Klangdynamik, Form
	Segmentspezifikation	Anfang und Ende einer Videoszene, Umriss eines Bildausschnitts
inhaltsunabhängig	präsentationsbezogen	QoS, Auflösung, Layout
	aufnahmebezogen	Urheber, Aufnahmegerät
	speicherungsbezogen	Medientyp, Speicherformat, Speicherort

Abbildung 2.2: Klassifikation von Metadaten, vgl. [Sch02]

Eine genaue Beschreibung der Klassifikation von Metadaten ist in [Sch02] gegeben. Für den weiteren Verlauf der Arbeit ist wichtig, dass Metadaten im Verlauf des Publikationsprozesses sowohl manuell als auch automatisch erzeugt werden müssen.

2.5 Content Management

Die bisherigen Abschnitte machen deutlich, dass im Rahmen des Publikationsprozesses ein Dokumentenmodell spezifiziert und Metadaten gewonnen werden müssen. Im Folgenden werden die Phasen der Veröffentlichung betrachtet. Der Publikationsprozess für Multimediadokumente setzt sich aus den Phasen für Publikationsprozesse in Content-Managementsystemen (CMS) und den Phasen aus digitalen Bibliothekssystemen zusammen.

Zu dem Prozess des Content Managements gehören das Erstellen, Bearbeiten, Freigeben, Wiederfinden, Speichern und das langfristige Archivieren von *Inhalten (Content)*. In [HD02] definiert A. Heuer den Begriff Content als Informationen, wie strukturierte Daten oder multimediale Dokumente, die in Rechnernetzen bereitgestellt werden sollen.
Es wird unterschieden zwischen elementaren und zusammengesetzten Content. Texte, Bilder oder Audiosignale sind elementarer Content. Sie können nicht in weitere Komponenten zerlegt werden. Hingegen bestehen Filme aus einer Sequenz von Einzelbildern und Ton. Videos sind folglich zusammengesetzter Content. Das Multimediadokument selbst wird ebenfalls in diese Kategorie gezählt.

Content-Managementsysteme automatisieren die einzelnen Phasen des Content Managements. Besonders verbreitet sind Web-Content-Managementsysteme, wie Wordpress[3] oder Drupal[4]. Sie ermöglichen Nutzern die Bereitstellung von Web-Inhalten und werden ausschließlich für Web-Anwendungen genutzt.
Im Fokus dieser Arbeit steht die Veröffentlichung von Dokumenten in einer digitalen Bibliothek. Der folgende Abschnitt beschreibt, warum der Publikationsprozess im Rahmen des Content Management nicht ausreichend ist.

2.6 Die digitale Bibliothek

Bei dem Publizieren von Multimediadokumenten (z. B. Vorlesungsfolien) wird häufig das Ziel verfolgt, dass diese für Kunden (hier Studenten) auffindbar sind. Content-Management-Systeme bieten die Möglichkeit Inhalte zu archivieren, aber ohne kundenseitige Content-Verwaltung. *Digitale Bibliotheken* hingegen erfüllen dieses Kriterium. [Sch09]

A. Heuer definiert eine digitale Bibliothek in [HD02] wie folgt: „Eine digitale Bibliothek ist

- eine Sammlung von Dokumenten mit bleibendem Wert,
- deren Dokumente mit Metadaten beschrieben (erschlossen) werden,
- die langfristig zitierbar bleiben müssen,

[3] http://wordpress-deutschland.org
[4] http://www.drupal.de

- 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').“

Zusammengefasst ermöglicht eine digitale Bibliothek das Verteilen, Finden und Archivieren von digitalen Dokumenten, sowie deren Nutzung durch Anbieter und Kunden.

An der Universität Rostock wird das Open-Source-System *MyCoRe* eingesetzt, welches auf Java- und XML-Technologien aufbaut und alle Grundfunktionen einer digitalen Bibliothek implementiert. In [Lüt02] wird das System ausführlich beschrieben.

2.7 Der Publikationsprozess

Bisher wurden die Phasen eines Publikationsprozesses nur grob benannt. Im Folgenden werden die Phasen genauer beschrieben. Abbildung 2.3 präsentiert den sequenziellen Ablauf einer Publikation.

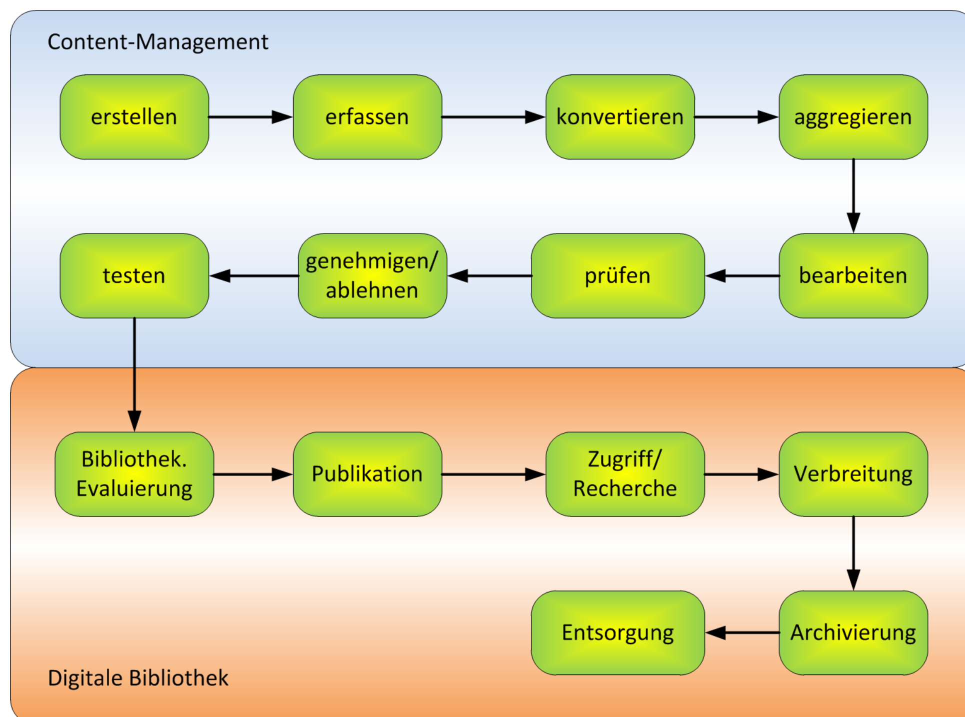


Abbildung 2.3: Phasen eines Publikationsprozesses, vgl. [Sch05]

1. **Erstellen**: In dieser Phase wird dem Autor die Möglichkeit gegeben, eigenen Content für sein Dokument zu erschaffen. Das jeweilige Content-Managementsystem stellt hierfür Schnittstellen bereit, um externe Autorenwerkzeuge anzubinden. Zusätzlich können von dem Werkzeug verwaltete Metadaten mit an das System übergeben werden. Eine andere Möglichkeit für das Erstellen von Inhalten ist die vollständige Integration einer Autoren-Umgebung in das CMS selbst.

2. **Erfassen**: Das Importieren von Content ist eine weitere Methode, um Daten in das Publikationssystem zu laden. Das CMS muss in diesem Fall Upload-Möglichkeiten bereitstellen. Der Datenaustausch kann beispielsweise über Schnittstellen, wie FTP[5] und WebDAV[6], oder komplett über ein Web-Interface erfolgen.

3. **Konvertieren**: Durch das Konvertieren wird erfasster Content in das interne Format des Content Management Systems überführt. Weiterhin erfolgt ein Mapping der Struktur des Contents auf die Standardstruktur des CMS.

4. **Aggregieren**: Diese Phase besteht aus zwei Schritten. Zuerst erfolgt die Segmentierung. Dadurch wird zusammengesetzter Content in elementaren Content zerlegt. Im zweiten Schritt benötigen die neu entstandenen Komponenten Metadaten. Sie werden entweder automatisch extrahiert oder manuell angegeben.

5. **Bearbeiten**: Auf Basis des zugrunde liegenden Dokumentenmodells wird beim Bearbeiten das Multimediadokument erzeugt. Dies umfasst die Modellierung der zeitlichen, räumlichen und interaktiven Beziehungen des erfassten Contents. Für die Umsetzung muss das CMS Editoren bereitstellen, um strukturelle Daten zu erfassen. Zusätzlich wird das Multimediadokument mit beschreibenden Metadaten erweitert.

6. **Prüfen**: Nach dem Bearbeiten prüft der Autor das Dokument ein letztes Mal. Der Fokus liegt auf der richtigen Darstellung der verschiedenen Medienobjekte. Danach gibt der Autor das Dokument für die Veröffentlichung frei.

7. **Genehmigen/Ablehen**: In diesem Schritt wird das zu publizierende Dokument ein weiteres Mal geprüft. Dies geschieht durch eine Instanz außerhalb der Bibliothek,

[5]Das File Tranfer Protocol (kurz FTP) ist ein Netzwerkprotokoll zur Übertragung von Dateien über IP-Netzwerke.

[6]Web-based Distributed Authoring and Versioning (kurz WebDAV) ist ein offender Standard zur Bereitstellung von Dateien im Internet.

beispielsweise durch einen Vorgesetzten des Autors. Diese Stufe stellt die Qualitätssicherung dar.

8. **Testen**: Das Dokument wird zum letzten Mal geprüft und an die digitale Bibliothek übergeben.

9. **Bibliothekarische Evaluierung**: Die Bibliothek überprüft zuerst, ob das Dokument valide ist. Dies geschieht anhand bestimmter Kriterien, die in jeder Bibliothek individuell festlegt sind.

10. **Publikation**: Das zu publizierende Dokument befindet sich zu diesem Zeitpunkt bereits im Repository[7] der Bibliothek. In diesem Schritt wird es für Suchmaschinen zugänglich gemacht.

11. **Zugriff/Recherche**: Eine Bibliothek stellt eine Vielzahl von Suchmechanismen bereit, um dem Nutzer die Suche zu erleichtern. Für deren Realisierung sind Maßnahmen, wie das Vergeben von Schlagworten, erforderlich.

12. **Verbreitung**: Hierzu zählen Aufgaben, wie die Bereitstellung verschiedener Dateiformate für ein Dokument und die Unterstützung von Internetprotokollen, um Dokumente zu dem Nutzer zu transportieren.

13. **Die Archivierung** umfasst alle Aufgaben im Zusammenhang mit der langfristigen Aufbewahrung von Dokumenten.

In Abbildung 2.3 ist ein optimaler Publikationsprozess abgebildet. Er beinhaltet keine Rücksprünge. Es ist denkbar, dass Phasen wiederholt ausgeführt werden müssen. Zum Beispiel kann die bibliothekarische Evaluierung fehlschlagen, da das Dokument einem Kriterium nicht genügt. In diesem Fall hat der Autor die Möglichkeit sein Dokument nachträglich zu bearbeiten. Weiter kann der Autor jederzeit zwischen den Phasen „Erstellen“ und „Prüfen“ wechseln. Aus Gründen der Übersichtlichkeit wurden die entsprechenden Kanten solcher Fälle in der Abbildung nicht berücksichtigt.
Zuletzt sei noch erwähnt, dass die Reihenfolge der Phasen leicht variieren kann. Es ist möglich, dass die Phasen „Publizieren“ und „Zugriff/Recherche“ je nach Umsetzung des Publikationsprozesses vertauscht ausgeführt werden.

[7] Ein Repository ist ein Verzeichnis für die Speicherung von digitalen Objekten

Kapitel 3

Der Publikationsprozess als Workflow

Am Lehrstuhl für Datenbank- und Informationstechnik der Universität Rostock wird ein Java-Framework entwickelt, um den in Kapitel 2 vorgestellten Publikationsprozess für Multimediadokumente umzusetzen. Es baut auf der Konzeption einer *Service-orientierten Architektur (SOA)* auf. Das Framework selbst wird in dieser Arbeit nicht betrachtet. Der Schwerpunkt liegt auf der Modellierung und Ausführung des Publikationsprozesses mit einem *Workflow-Managementsystem.*
In diesem Kapitel erfolgt eine kurze Einführung in die Themen Service-orientierte Architekturen und Workflow Management. Anschließend wird das Kernproblem der Arbeit am Beispiel des Publikationsprozesses beschrieben.

3.1 Serive-orientierte Architekturen

Eine SOA ist ein Architekturmuster, mit denen Dienste (Services) in verteilen Systemen genutzt werden können. Für den Begriff „Service-orientierte Architektur“ existiert keine einheitliche Definition. Es lassen sich aber einige grundlegende Merkmale finden, um die SOA zu charakterisieren. Die folgende Aufzählung benennt die wesentlichsten Charakteristika, wie sie in [Mel07] hervorgehoben werden:

- Die **Verwendung von Standards** ist ein wesentliches Merkmal einer SOA. Dies beinhaltet die Beschreibung sämtlicher Schnittstellen, die für die Kommunikation benötigt werden. Offene Standards spielen eine besondere Rolle. Sie ermöglichen es, dass ein Nutzer einen Dienst von einem unbekannten Anbieter verstehen kann.

Weiterhin erhöht die Verwendung von Standards die Akzeptanz der jeweiligen Architektur.

- **Sicherheit** ist eines der Kernthemen in Netzwerken und verteilen Systemen. Dazu zählen Aspekte wie Vertraulichkeit, Berechtigung, Glaubwürdigkeit und Verbindlichkeit von Informationen, die zwischen Sender und Empfänger transportiert werden. Für eine genaue Beschreibung sei hier auf [Mel07] verwiesen.
- **Einfachheit** bedeutet, dass Dienste leicht austauschbar sind und in verschiedenen Umgebungen mehrfach wiederverwendet werden können.
- Die **lose Kopplung von Diensten** bedeutet, dass Dienste zur Laufzeit bei Bedarf von Anwendungen oder anderen Diensten dynamisch gesucht, gefunden und eingebunden werden können.
- Ein **Verzeichnisdienst** oder **Repository** ermöglicht das Suchen der Methoden eines Dienstes, die von einer Anwendung benötigt werden. Dazu müssen alle zur Verfügung stehenden Dienste im Verzeichnisdienst registriert sein.

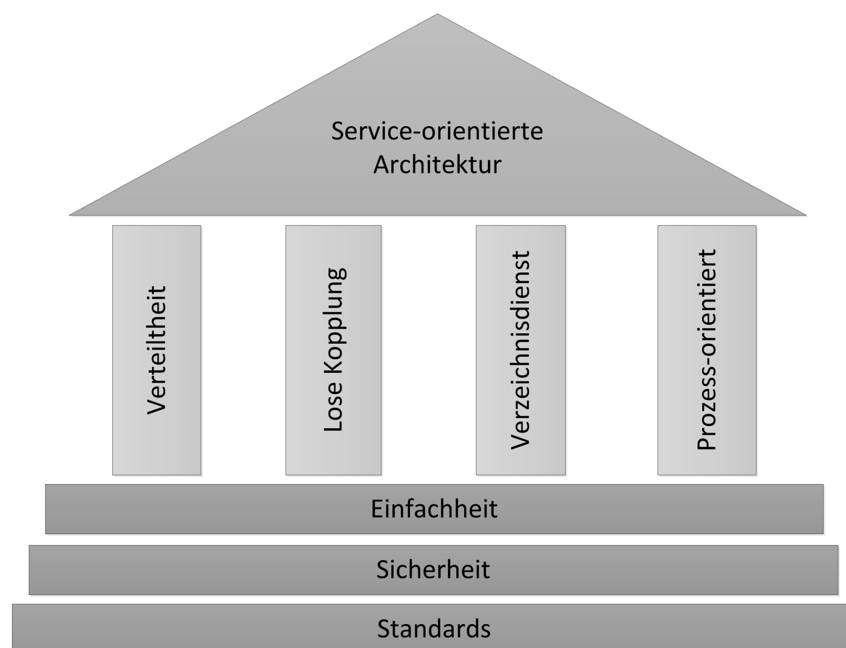


Abbildung 3.1: Grundkonzept einer Service-orientierten Architektur, vgl. [Mel07]

Abbildung 3.1 stellt den sogenannten *SOA-Tempel* dar, wie er durch I. Melzer in [Mel07] vorgeschlagen wird. Das Fundament benennt die notwendigen Voraussetzungen für eine

Service-orientierte Architektur.
Hinzu kommen vier tragende Säulen, die lose Kopplung, die Verteiltheit bzw. Plattformunabhändigkeit, der Verzeichnisdienst und die prozessorientierte Struktur. Letzteres wird im nachfolgenden Abschnitt genauer beschrieben. In diesem Bereich ist das Kernproblem dieser Arbeit angesiedelt.

3.2 Workflow Management

Die Prozessorientierung ist eine der tragenden Säulen und ein wesentlicher Aspekt einer Service-orientierten Architektur. Die Grundlage bildet die Modellierung von *Geschäftsprozessen*, welche in vielen heutigen Unternehmen fest integriert sind. Sie beschreiben eine Menge von Aktivitäten, die ausgeführt werden, um ein bestimmtes betriebliches Ziel zu erreichen.
Der Bezug zur SOA ergibt sich aus der Tatsache, dass Geschäftsprozesse vorhandene Dienste integrieren können. Somit ist es möglich, beliebige Dienste zu einem neuen komplexen Dienst zu komponieren.
Daraus ergibt sich der Vorteil, dass Änderungen im Prozessablauf keine Neuimplementation des komplexen Dienstes erfordern. Es genügt, die vorhandenen Dienste neu zusammenzustellen. [Mel07]
Der Publikationsprozess stellt einen Geschäftsprozess dar. Er beinhaltet alle Schritte, um ein Multimediadokument zu erstellen und in einer digitalen Bibliothek zu veröffentlichen. Dienste sind in diesem Fall alle Programme, die für die Realisierung benötigt werden. Die Integration diverser Autorenprogramme für die Erstellung von verschiedenem Content sind Beispiele derartiger eingebrachter Dienste.

Mit der Einführung von Geschäftsprozesses entstand das Gebiet des Workflow Managements, welches eine große Herausforderung für die Softwaretechnik darstellt. Seit der Gründung der *Workflow Management Coalition (WfMC)* im Jahr 1993 wurden Standards für diesen Bereich eingeführt, auf denen sich nachfolgende Begriffsdefinitionen stützen. [Wor99]

Ein *Workflow* ist ein vollständig oder teilweise automatisierter Geschäftsprozess, der Informationen oder Aktivitäten anhand von prozeduralen Regeln von Workflow-Teilnehmer zu Workflow-Teilnehmer weiterreichen kann.

Ein *Workflow-Managementsystem (WfMS)* ist ein Softwaresystem, welches die Definition und Erzeugung von Workflows ermöglicht. Weiterhin verwaltet und überwacht es deren Ausführung, interagiert mit am Workflow beteiligten Personen und kann benötigte Tools oder Applikationen aufrufen.

Die *Prozess-Definition* ist die Darstellung eines Geschäftsprozesses in einer Form, welche die automatische Manipulation durch ein WfMS unterstützt. Eine Spezifikation besteht aus einem Netzwerk von Aktivitäten und ihren Beziehungen, welche den Start und das Ende eines Prozesses festlegen. Zusätzlich enthält sie Informationen zu jeder Aktivität, wie beispielsweise Teilnehmer, zugewiesene Dienste und Daten.

Aktivitäten (Tasks) beschreiben einen Arbeitsschritt und bilden einen logischen Schritt innerhalb des Prozesses. Es wird zwischen manuellen und automatischen Tasks unterschieden. Erstere unterstützen keine automatisierte Abarbeitung und sind auf das Eingreifen von Personen angewiesen. Hingegen benötigt eine automatisierte Aktivität menschliche und/oder maschinelle Ressourcen.

Eine *Prozessinstanz (Case)* ist ein konkreter Ausführungsstrang, welcher sich aus der Prozess-Definition ergibt. Jede Aktivität einer Prozessinstanz wird als *Task-Instanz* bezeichnet.

Workflow-Teilnehmer sind Ressourcen, welche Task-Instanzen abarbeiten. Ein Teilnehmer erhält die auszuführende Arbeit einer Task-Instanz (inklusive Daten) in Form eines *Work Items*. Sämtliche Work Items, die mit einem Workflow-Teilnehmer verknüpft sind, werden in einer *Worklist* gehalten.

Die WfMC hat weiterhin ein *Workflow Reference Model (WRM)* vorgestellt, an dem sich heutige Worflow-Managementsysteme orientieren. Es identifiziert die wichtigsten Interfaces, die für ein WfMS relevant sind. In Abbildung 3.2 wird das WRM abgebildet.

Das WRM beschreibt unter anderem ein Interface, um Prozess-Definitionen in die *Workflow Engine* zu im- bzw. exportieren. Das Interface 2 dient zum Austausch von Work Items zwischen der Engine und Applikationen. Die Abarbeitung eines Work Items beginnt mit dessem „Auschecken“ durch eine Workflow-Applikation. In der Folge wird das Work Item von der jeweiligen Applikation abgearbeitet und schließlich wieder an die Engine über-

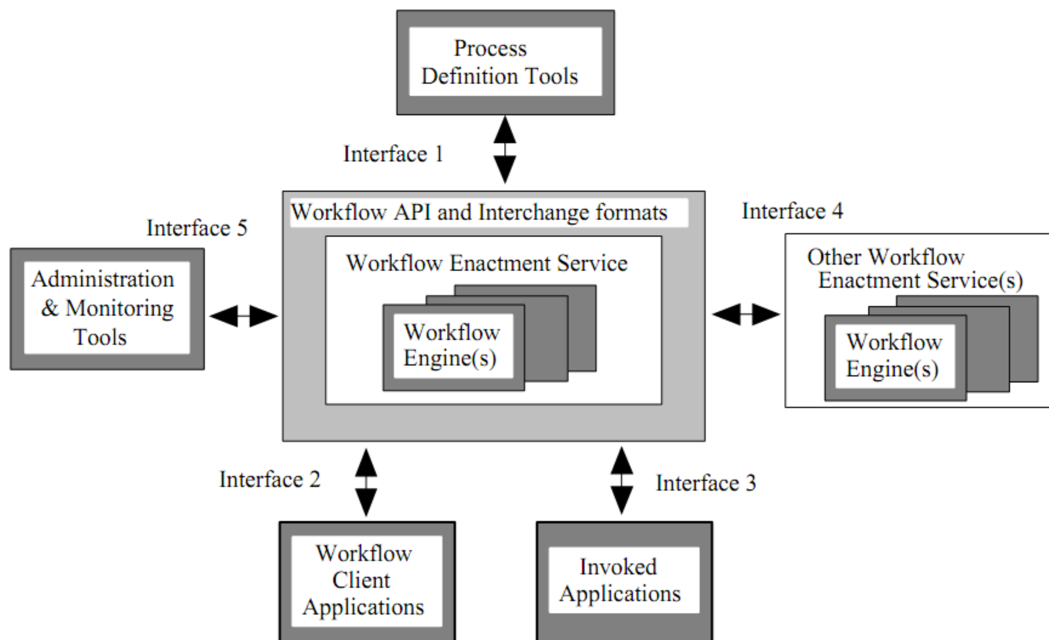


Abbildung 3.2: Das Workflow-Referenz Modell, vgl. [Wor99]

geben. Diesen Vorgang wird „Einchecken“ genannt. Über das Interface 3 ist die Engine in der Lage, externe Applikationen direkt anzusprechen, um die Abarbeitung eines Work Items direkt anzustoßen. Das Interface 4 beschreibt eine Schnittstelle für die Verbindung der Engine des aktuellen WfMS mit Engines anderer Workflow Management Systeme. Mit dem letzten Interface werden Administrations- und Monitoring-Tools an das System angebunden, welche die Verwaltung von laufenden Prozessinstanzen in der Engine ermöglichen. [TVAR10]

3.3 Workflow Patterns

Im Jahr 1999 startete die *Workflow Patterns Initiative* mit dem Ziel der Identifikation typischer, wiederkehrender Konstrukte, wie sie bei der Modellierung von Geschäftsprozessen immer wieder benötigt werden. Seitdem wurden diese Muster ständig erweitert und stellen heute eine umfassende Sammlung von Konstrukten bereit, ohne sich auf eine konkrete Prozessmodellierungssprache festzulegen. Die Unterstützung von *Workflow Patterns* ist ein allgemein akzeptierter Maßstab, um die Tauglichkeit von Modellierungssprachen zu bewerten. Bis heute wurden über 120 Workflow Patterns beschrieben. Sie werden vier

verschiedenen Kategorien zugeordnet, die an dieser Stelle kurz eingeführt werden:

1. **Control-Flow Patterns** beschreiben, wie der Kontrollfluss zwischen Aktivitäten umgesetzt werden kann. Diese Kategorie definiert verschiedene Konstrukte für die Verzweigung, Zusammenführung, Wiederholung, Mehrfach-Instanziierung, Auslösung, Abbruch und Beendigung von Tasks. In [RTM06] geben die Autoren einen umfassenden Überblick über mehr als 40 Control-Flow Patterns.

2. **Data Patterns** befassen sich mit der Definition und Verwaltung von Daten während der Ausführung von Workflows. Dies beinhaltet die Beschreibung von Gültigkeitsbereichen für Datenelemente oder deren Transport zwischen verschiedenen Prozessteilen. Data Patterns werden in [RTE04a] genau behandelt.

3. **Ressource Patterns** erfassen die verschiedenen Möglichkeiten für die Verteilung von Arbeitseinheiten an Ressourcen, die mit dem Prozess verbunden sind. Ein Beispiel ist die Erstellung von Rollen für Personen und Geräte. Eine Aktivität wird folglich einer Rolle zugewiesen, welche deren Ausführung übernimmt. Ein Gesamtüberlick dieser Kategorie wird in [RTE04b] gegeben.

4. **Exception Handling Patterns** definieren „Exception Handling“ Strategien und bieten somit Lösungen für Ausnahmesituationen bei der Ausführung von Workflows. In [RTE06] fassen die Autoren die existierenden Exception Handling Patterns zusammen.

YAWL (*Yet Another Workflow Language*) ist heute eine der Prozessmodellierungssprache, die nahezu alle Workflow Patterns unterstützt. In [Ter10] beschreibt A. Ter Hofstede YAWL als die mächtigste Sprache in Bezug auf die Erfassung des Betriebsmittelbedarfs und die Modellierung von Kontrollflussabhängigkeiten. Sie ist in der Lage erwartete Exceptions zur Modellierungszeit, sowie unerwartete Exceptions zur Laufzeit zu behandeln. Weiterhin erfasst YAWL die Sicht auf Daten mit Hilfe von XML Schema[1], XPath[2] und XQuery[3] und realisiert auf diese Weise die Data Patterns.

Nicht zuletzt aufgrund dieser Vorteile wurde am Lehrstuhl für Datenbank- und Informa-

[1]XML Schema ermöglicht die Definition von Strukturen für XML-Dokumente.
[2]Xpath ist eine Abfragesprache für die Adressierung von Teilen eines XML-Dokuments.
[3]XQuery ist eine Abfragesprache für XML-Datenbanken, welche XML Schema und XPath nutzt.

tionssysteme der Universität Rostock entschieden, dass der Publikationsprozess mit der Prozessmodellierungssprache YAWL umgesetzt wird.

3.4 Das Modellierungsproblem: Prozessvarianten

Ein Prozesstyp tritt oftmals in zahlreichen Varianten auf, welche Anpassungen an bestimmte Rahmenbedingungen darstellen. Ein Beispiel sind Prozesse für den Bereich der Produktentstehung in der Automobilindustrie. Die Arbeitsabläufe bei der Herstellung eines Fahrzeugs variieren, da der Prozess an einen bestimmten Produkttyp (beispielsweise PKW, Bus oder LKW) gebunden ist.
Ähnliche Betrachtungen können für die Veröffentlichung von Multimediadokumenten angewendet werden. Der grobe Arbeitablauf ist für jede Publikation gleich. Hingegen variiert der Prozess durch die unterschiedlichen Zusammensetzungen eines Multimediadokuments.

Die Phase der Aggregation eignet sich gut, um *Prozessvarianten* am Beispiel des Publikationsprozesses zu verdeutlichen. Die Aggregation besteht aus zwei Teilschritten. Zuerst wird zusammengesetzter Content in elementaren Content zerlegt. Im Anschluss erfolgt die *Indexierung*. Sie bezeichnet den Vorgang des Schreibens von Metadaten und unterteilt sich weiter in die Feature-Extraktion und manuelle Angabe von Metadaten. Abbildung 3.3A stellt einen Standardprozess für die Aggregation eines reinen Textdokument dar.
Der Prozess beginnt mit der Zerlegung des Content-Typs „Text“ in Abschnitte und Unterabschnitte. Zudem sind eventuell Grafiken aus der Textdatei extrahierbar. Im Anschluss folgt eine *Volltextindexierung*, welche Schlüsselwörter automatisch aus dem Text entnimmt. Die Volltextindexierung speichert alle im Text vorkommenden Wörter in einem Index. Ausgenommen sind sogenannte Stopwörter. Dazu zählen bestimmte und unbestimmte Artikel, sowie Konjunktionen und Präpositionen jeder Art, da solche Wörter keine Relevanz zum Inhalt eines Dokuments besitzen. Danach erfolgt die Angabe von manuellen Metadaten.

Ändert sich der Kontext des in Abbildung 3.3A abgebildeten Prozesses von einem reinen Textdokument zu einem Dokument mit Text und Bild ab, ergibt sich die Prozessvariante aus Abbildung 3.3B. Ein Bild ist bereits elementarer Content. Daher wird es nur gespeichert und muss nicht in weitere Bestandteile zerlegt werden. Die Feature-Extraktion

hingegen benötigt zusätzliche Arbeitsschritte für die Gewinnung von Bildinformationen. Es wird unterschieden zwischen *low-level-* und *high-level*-Features. Ersteres beschreibt die elementaren Bildcharakteristika, wie Farben, Formen und Texturen. Low-level Features sind als Bildbeschreibung nicht ausreichend. Das Problem ist, dass sich beispielsweise Bilder mit ähnlichen Farben nicht inhaltlich ähneln müssen. Umgekehrt können Bilder mit völlig unterschiedlichen Farben den gleichen Inhalt darstellen [Bar10].

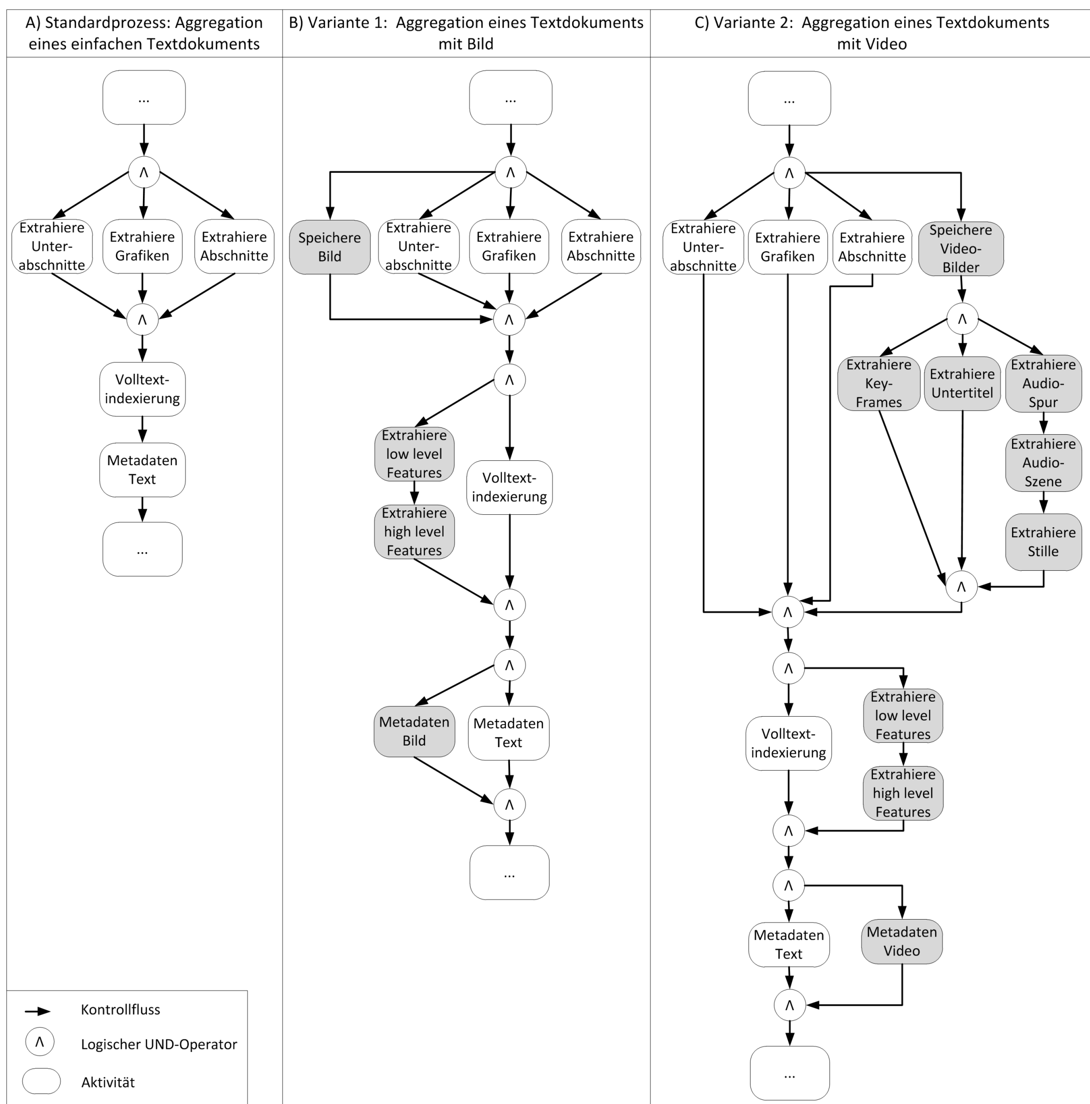


Abbildung 3.3: Prozessvarianten der Aggregation (Idee der Grafik aus [HBR08])

Erst high-level Features ermöglichen die inhaltsbasierte Suche von Bildern. Sie identifizieren den abgebildeten Sachverhalt (z. B. Menschen, Tiere, Bäume, ...) [ES10]. Zuletzt wird

der Prozess um eine Aktivität erweitert, welche die manuelle Eingabe von Metadaten für das Bild ermöglicht.

Abbildung 3.3C illustriert eine weitere Variante der Aggregation. In diesem Fall besteht ein Dokument aus den Bestandteilen „Text“ und „Video“. Ein Video ist zusammengesetzter Content und muss somit in elementaren Content zerlegt werden. Hierzu zählt die Speicherung aller Einzelbilder aus denen das Video besteht, sowie die Extraktion von Untertiteln und Audiospur. Audiosignale wiederum setzen sich aus Audioszenen und geräuschfreien Momenten zusammen, welche gesondert abgespeichert werden. Zusätzlich erfolgt eine Sicherung der Key-Frames, die den kontinuierlichen Übergang von einer Szene zu einer anderen definieren. Die Feature-Extraktion aus einem Video beschränkt sich im Wesentlichen auf die Gewinnung von low-level und high-level Informationen aus den Einzelbildern. Zuletzt stellt der Prozess in Analogie zur Prozessvariante 3.3B eine Möglichkeit für die manuelle Eingabe von Video-bezogenen Metadaten zur Verfügung.

Es können zusätzlich Informationen aus Audio-Clips gewonnen werden. An dieser Stelle wurde auf die Modellierung von Audio-Features verzichtet. Weiter werden in den Prozessabläufen Iterationen und Rücksprünge nicht berücksichtigt. Die wiederholte Ausführung von Aktivitäten ist nicht ausgeschlossen. Iterationen werden beispielsweise bei der Abarbeitung mehrerer Bilder benötigt.

3.5 Anforderungsanalyse

Die effiziente Modellierung und Ausführung von Prozessvarianten stellt für heutige Geschäftsprozessmodellierungswerkzeuge eine große Herausforderung dar. Herkömmliche Ansätze definieren und halten Prozessvarianten in separaten Prozessmodellen. Die Folge sind eine Vielzahl von eigenständigen Prozessbeschreibungen, die einzeln gewartet und gepflegt werden müssen. In [HB08] und [HBR08] beschreibt A. Hallerbach, warum diese Art der Modellierung nicht geeignet ist, um Prozessvarianten umzusetzen. Im Wesentlichen existieren drei Probleme:

1. Eine große Anzahl von Prozessmodellen sind für den Modellierer schlecht handhabbar bzw. wartbar.

2. Hoher Änderungsaufwand: Alle Prozessvarianten müssen separat geändert werden.

3. Redundante Modellierung: Viele Prozessteile werden immer wieder modelliert.

Um obige Probleme zu vermeiden, besteht die Möglichkeit sämtliche Prozessvarianten in nur einem Modell zu modellieren. Hierfür werden die verschiedenen Varianten durch bedingte Verzweigungen miteinander verbunden. Die nachfolgenden Gründe zeigen, warum diese Art der Modellierung ebenfalls nicht ausreichend ist:

1. „Der Modellierer kann nicht mehr erkennen, ob eine bedingte Verzweigung eine Prozessvariante darstellt oder eine ‚normale' Verzweigung." [HB08]

2. Ab eine bestimmten Anzahl von Verzweigungen ergibt sich ein sehr großes und unübersichtliches Prozessmodell.

3. Es wird immer das gesamte Modell instanziiert, obwohl nur ein Pfad des Modells ausgeführt wird.

Obige Nachteile zeigen die Schwierigkeit im Umgang mit Prozessvarianten. Gleichzeitig ist es mit ihrer Hilfe möglich, Anforderungen an die Modellierung und Ausführung des Publikationsprozesses zu definieren. Die Anforderungen lassen sich gut in die Phasen des *Process Life Cycle* einordnen. Hierzu stellt Abbildung 3.4 zuerst den „Standard-Lebenszyklus" eines Prozesses dar.

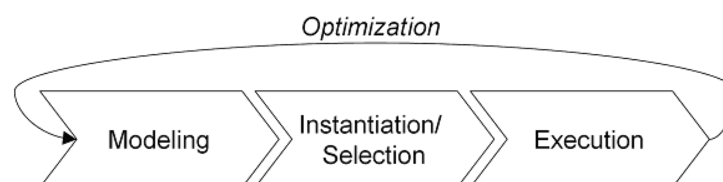


Abbildung 3.4: Process Life Cycle, vgl. [HBR08]

Er besteht aus drei Phasen. Zuerst wird ein Prozess hinsichtlich bestimmter Zielstellungen geplant und modelliert. Anschließend folgt die Instanziierung des Prozessmodells. Prozessvarianten werden in dieser Phase anhand von Kontextinformationen ausgewählt. Die dritte Phase ist die Ausführung der Prozessinstanz in einem WfMS.

Für die **Modellierungsphase** ergeben sich für den Publikationsprozesses drei erforderliche Anforderungen, welche obige Nachteile vermeiden:

Anforderung 1a: Der Modellierungs-, Änderungs- und Wartungsaufwand von Prozessvarianten muss minimal sein. Demzufolge entfallen herkömmliche Ansätze für die Handhabung von Prozessvarianten.

Anforderung 1b: Die Modellierung von Prozessvarianten muss frei von Redundanzen sein.

Anforderung 1c: Die Modellierung von Prozessvarianten muss übersichtlich sein.

Herkömmliche Ansätze verwalten Prozessvarianten unter anderem in nur einem Modell. Der große Nachteil ist, dass immer das gesamte Modell instanziiert werden muss, obwohl große Teile des Modells bei der Ausführung ungenutzt bleiben. Um solche Probleme zu vermeiden, wird für die Phase der **Instanziierung** die folgende Anforderung definiert:

Anforderung 2: Es dürfen nur Prozessteile instanziiert werden, die zur Laufzeit tatsächlich benötigt werden.

Im Verlaufe des Publikationsprozesses erzeugt der Autor ein Multimediadokument, welches er veröffentlichen möchte. Entsprechend existieren erst zur Laufzeit Informationen über die konkrete Zusammensetzung eines Dokuments (der enthaltene Content). Eine Prozessvariante ist somit zum Zeitpunkt der Instanziierung nicht auswählbar. Vielmehr muss der Prozess in der Lage sein, auf dynamische Kontextänderungen zu reagieren. Hieraus ergeben sich die Anforderungen 3a und 3b für die **Ausführungsphase** des Process Life Cycle:

Anforderung 3a: Prozessvarianten müssen zur Laufzeit anhand von Kontextinformationen auswählbar sein.

Anforderung 3b: Prozessvarianten müssen zur Laufzeit aufgrund von Kontextänderungen ausgetauscht werden können.

Die letzte Anforderung ergibt sich aus der Tatsache, dass am Lehrstuhl für Datenbank- und Informationssysteme der Universität Rostock mit der Prozessmodellierungssprache YAWL gearbeitet wird.

Anforderung 4: Ein Ansatz zur Handhabung von Prozessvarianten muss mit der Prozessmodellierungssprache YAWL umsetzbar sein.

Im nächsten Kapitel werden vier konkrete Ansätze vorgestellt, die sich mit der effizienten Handhabung von Prozessvarianten beschäftigen. Im Fokus steht hierbei die Beurteilung dieser Ansätze hinsichtlich der aufgestellten Anforderungen. Ziel ist es, einen Ansatz zu bestimmen, welcher sich am besten für die Umsetzung des Publikationsprozesses eignet.

Kapitel 4

Modellierungsansätze für den Publikationsprozess

4.1 Der deklarative Ansatz

In der Vergangenheit gab es viele Diskussionen über die Überlegenheit einer bestimmten Programmiersprache im Vergleich zu einer anderen. Thematisiert wurden Eigenschaften wie Ausdrucksstärke oder Leistungsfähigkeit. Das Ergebnis dieser Debatten war, dass eine Programmiersprache für die Lösung einer bestimmten Aufgabe anderen überlegen ist, aber die selbe Sprache bei der Lösung eines anderen Problems von alternativen Programmiersprachen übertroffen wird. Folglich hat jede Programmiersprache seine Stärken und Schwächen.
Gleiches kann auf Prozessmodellierungssprachen angewendet werden. Es existieren verschiedene Vorgehensweisen für die Modellierung von Geschäftsprozessen, welche unterschiedliche Ansätze verfolgen. Sie unterscheiden sich in Aspekten wie Wartbarkeit, Verständlichkeit bzw. Lesbarkeit und vor allem bei der Modifizierbarkeit bzw. Änderbarkeit von Prozessbeschreibungen.

4.1.1 Deklarative gegenüber Imperativer Modellierung

Heutige Prozessmodellierungssprachen sind vornehmlich *imperativer* Natur. Imperative Prozessmodelle beschreiben exakt *wie* gearbeitet wird. Sie definieren einen Algorithmus für die Prozessausführung nach dem das System entscheidet, in welcher Reihenfolge die Aktivitäten ausgeführt werden. [PV06] Imperative Prozessmodelle verwenden den *inside-*

to-outside Ansatz. Alle Ausführungsalternativen werden explizit im Modell spezifiziert und das Modell muss für jede neue Alternative explizit erweitert werden. [FMRW10] Ein imperatives Prozessmodell besteht demnach aus einer Menge von Aktivitäten und Kontrollflüssen, welche die einzelnen Tasks miteinander verbinden und die Ausführungsreihenfolge und -alternativen festlegen.
Im Gegensatz zu imperativen Sprachen spezifizieren *deklarative* Modellierungssprachen nur das *was*, ohne sich auf das *wie* festzulegen. Wenn ein Nutzer mit einem derartigen Prozessmodell arbeitet, wird er vom System angeleitet, ein bestimmtes Ergebnis zu erzielen. Die Art und Weise wie das Ziel erreicht wird, hängt nur von den Präferenzen des Nutzers ab. [PV06]
Deklarative Prozessmodelle verwenden den *outside-to-inside* Ansatz. Ein minimales deklaratives Modell besteht nur aus einer Menge von Aktivitäten. Der Nutzer kann selbst entscheiden, wie oft er Aktivitäten ausführt und in welcher Reihenfolge er dies tut.
Die Angabe von sogenannten *Constraints (Einschränkungen)* ermöglicht die Spezifikation bestimmter Ausführungsalternativen. Im Gegenzug gehen mit der Definition von Constraints für gewöhnlich einige Ausführungsalternativen verloren und es bleiben jene Alternativen, die dem Constraint genügen. Durch Einschränkungen erhält das deklarative Prozessmodell einen imperativen Charakter. Mit einer entsprechenden Anzahl von Constraints kann somit das Verhalten eines imperativen Prozessmodells simuliert werden.

Das folgende Beispiel verdeutlicht den Unterschied zwischen imperativer und deklarativer Modellierung. Angenommen, für eine gegebene Prozessinstanz soll entweder nur Aktivität A oder Aktivität B ausgeführt werden, aber niemals beide. Abbildung 4.1a zeigt eine entsprechende Modellierung mit einer imperativen Sprache.

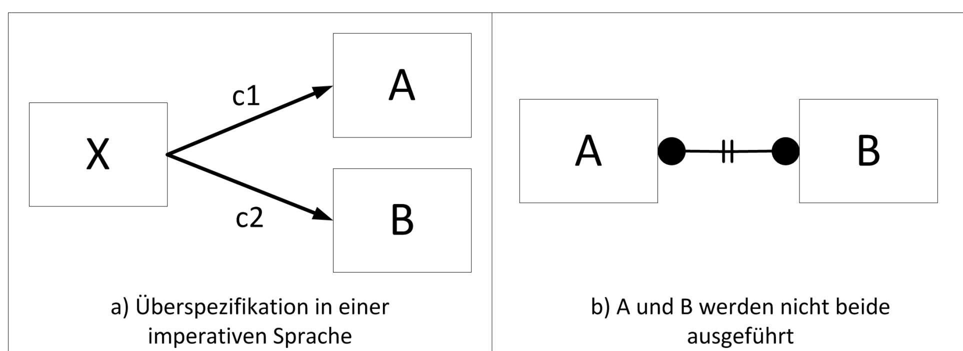


Abbildung 4.1: Vergleich zwischen imperativer und deklariver Modellierung, vgl. [PV06]

Für die Umsetzung muss eine „Entscheidungsaktivität“ X existieren, welche zu einem Zeitpunkt vor Task A oder B ausgeführt wird. Anhand von Regeln (c1 und c2) ermittelt Aktivität X den weiteren Kontrollfluss. Es wird implizit angenommen, dass A bzw. B nur einmal ausgeführt werden.
Durch diese Art der Modellierung haben Nutzer keine Möglichkeit den Kontrollfluss eines imperativen Modells zu beeinflussen. Es wird auch von einer Überspezifikation des Prozessmodells gesprochen. Die Folge sind sehr starre Prozessabläufe, welche unflexibel gegenüber Änderungen sind. Anders verhält sich die in Abbildung 4.1b dargestellte deklarative Modellierung mit der Sprache *ConDec* (siehe Abschn. 4.1.2).
ConDec stellt für diesen Sachverhalt ein Constraint zu Verfügung, welches für zwei Aktivitäten spezifiziert werden kann. Es ermöglicht die Ausführung von Aktivität A beliebig oft, solange Aktivität B nicht ausgeführt wird. Gleiches gilt für den umgekehrten Fall. Eine zusätzliche Entscheidungsaktivität wird nicht benötigt. Einzig der Nutzer entscheidet, zu welchem Zeitpunkt und wie oft er Aktivität A oder B ausführt. Der anschließende Unterabschnitt beschreibt kurz die deklarative Prozessmodellierungsprache ConDec.

4.1.2 Die deklarative Sprache ConDec

Wie bereits erwähnt, definiert ConDec im einfachsten Fall nur die Aktivitäten, welche nötig sind, um ein gewünschtes Ziel zu erreichen. In [PV06] beschreibt M. Pesic et. al, dass zu jedem ConDec-Modell zusätzlich Regeln hinzugefügt werden können, falls die Abarbeitung nach bestimmten Vorgaben erfolgen soll.
Regeln definieren Beziehungen zwischen Aktivitäten und werden als Constraints bezeichnet. Constraints besitzen entweder den boolschen Wert „*true*“ oder „*false*“. Dieser kann sich während der Ausführung des Modells jederzeit ändern. Wenn ein Constraint den Wert „true“ hat, dann ist die assoziierte Regel erfüllt. Der Wert „false“ hingegen kennzeichnet eine Verletzung der entsprechenden Regel.
Die Korrektheit des Modells wird zu jedem Zeitpunkt der Ausführung evaluiert. Die Ausführung eines Modells ist genau in dem Moment korrekt, wenn alle definierten Constraints den Wert „true“ besitzen. Weiterhin können Constraints ebenfalls nur vorübergehend verletzt sein. Während der Ausführung eines ConDec-Modells ist es möglich, dass eine Regel mit dem Wert „false“ evaluiert wird, aber jene Regel zu einem späteren Zeitpunkt wieder erfüllt wird. Demnach bedeutet ein verletztes Constraint nicht zwangläufig ein Fehler in der Ausführung. Erst am Ende der Abarbeitung muss sich das ConDec-Modell in einem

Zustand befinden, in dem alle Constraints den Wert „true“ besitzen.

Für die deklarative Modellierung von Beziehungen zwischen Tasks wird in [PV06] die *Linear Temporal Logic (LTL)* verwendet. Dies ermöglicht die Definition von zeitlichen Anforderungen. Hierzu beinhaltet die LTL neben den logischen Standardoperatoren die temporalen Operatoren: **nexttime** ($\bigcirc F$), **eventually** ($\Diamond F$), **always** ($\Box F$) und **until** ($\sqcup F$).
Für die Erstellung von Constraints werden *Constraint Templates* verwendet. Jedes Template besteht aus einer LTL-Formel und ihrer grafischen Notation. Gegeben sei beispielsweise die folgende LTL-Formel, welche eine Beziehung zwischen zwei Aktivitäten definiert:

$$\Box(A \longrightarrow \Diamond B)$$

Sie beschreibt den Sachverhalt, dass auf jeder Ausführung von Aktivität A schließlich die Ausführung von Task B folgen wird. Die entsprechende grafische Notation wird in Abbildung 4.2 dargestellt.

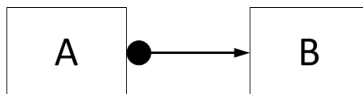


Abbildung 4.2: Grafische Notation für $\Box(A \longrightarrow \Diamond B)$, vgl. [VP06]

Constraint Templates sind einfach zu ändern, entfernen, erweitern und wiederzuverwenden und aus diesem Grund ideal, um Beziehungen zwischen beliebigen Aktivitäten zu definieren. Templates werden in drei Gruppen eingeteilt:

1. **Existence Templates** definieren die Kardinalitäten einer **einzelnen** Aktivität.

2. **Relation Templates** definieren die Beziehung zwischen **zwei** Aktivitäten.

3. **Negation Templates** sind negierte Versionen der Relation Templates.

In [VP06] wird ein detaillierter Überblick in Tabellenform über die verschiedenen Templates gegeben. Zum Abschluss wird in Abbildung 4.3 ein Beispiel für ein einfaches ConDec-Modell mit drei Aktivitäten präsentiert. Insgesamt sind vier Constraints definiert, jeweils zwei Relation- und Existence Templates. Die Abbildung verdeutlicht die Verwendung von Constraints in einem deklarativen Prozessmodell.

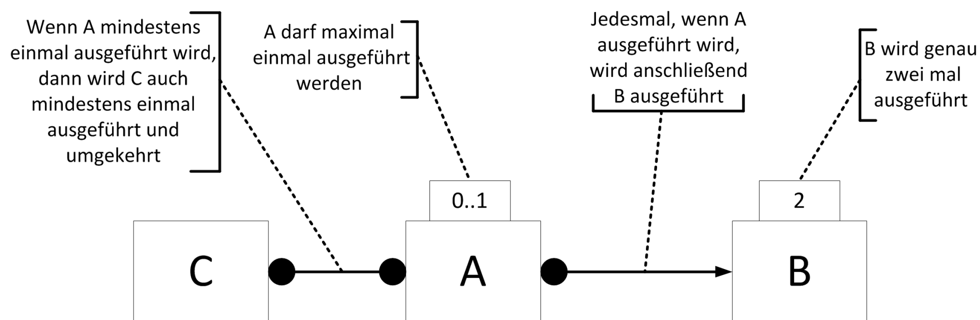


Abbildung 4.3: Ein einfaches ConDec-Modell, vgl. [PV06]

4.1.3 Beurteilung des deklarativen Ansatzes

Der deklarative Ansatz ermöglicht die Modellierung von Prozessvarianten in nur einem Modell. Einzig die Übersichtlichkeit kann bei bei einer großen Anzahl von Varianten verloren gehen. Der Vorteil eines deklarativen Prozessmodells ist, dass zur Laufzeit die Abarbeitungsreihenfolge der Aktivitäten beeinflusst bzw. bestimmt werden kann. Es existieren keine einschränkenden Kontrollflüsse. Somit kann in einem deklarativen Prozessmodell eine Prozessvariante zu jeder Zeit anhand von Kontextinformationen ausgewählt (Anforderung 3a) und ebenfalls ausgetauscht werden (Anforderung 3b). Ein Nachteil des deklarativen Ansatzes ist, dass immer das gesamte Modell instanziiert wird (Anforderung 2). Daher erzielt diese Art der Umsetzung lediglich den gleichen Effekt, wie ein imperatives Modell, indem Prozessvarianten zur Laufzeit anhand von bedingten Verzweigungen ausgewählt werden. Abbildung 4.4 präsentiert die Beurteilung des deklarativen Ansatzes tabellarisch.

Anforderung / Ansatz	Minimaler Modellierungs-, Änderungs-, Wartungsaufwand	Redundanzfreie Modellierung	Übersichtlichkeit der Prozessvarianten	Instanziierung nur erwünschter Prozessteile	Prozessvarianten zur Laufzeit auswählbar	Prozessvarianten zur Laufzeit austauschbar	Umsetzbar in YAWL
Deklarativer Ansatz	✓ (erfüllt)	✓ (erfüllt)	teilweise erfüllt	nicht erfüllt	✓ (erfüllt)	✓ (erfüllt)	teilweise erfüllt

✓ erfüllt — teilweise erfüllt — nicht erfüllt

Abbildung 4.4: Beurteilung des deklarativen Ansatzes

YAWL ist eine rein imperative Prozessmodellierungssprache. Dies widerspricht dem deklarativen Ansatz. Der *Declare Service* ist ein Custom YAWL Service (siehe Abschn. 6.2), der es ermöglicht, YAWL mit dem deklarativen WfMS *Declare* zu verbinden. Auf diese

Weise können deklarative und imperative Ansätze vereint werden. Declare verwendet unter anderem die Sprache ConDec, um Constraints zu definieren. Der gleichnamige YAWL Service erlaubt die Dekomposition einer Declare Task zu einen YAWL Workflow und umgekehrt. Letzteres ermöglicht das Auslagern variantenspezifischer Prozessteile in deklarative Modelle, deren Abarbeitung durch eine YAWL-Task ausgelöst wird. Diese Richtung der Verbindung ist ist für den Publikationsprozess nicht anwendbar. Ein entsprechender Declare Workflow wird zur Modellierungszeit mit einer YAWL-Task verknüpft. Folglich ist die Auswahl einer Prozessvariante zur Laufzeit unmöglich.
Der umgekehrte Weg ist die Modellierung variantenspezifischer Prozessteile in verschiedenen YAWL Netzen, welche durch assoziierte Declare Aktivitäten angestoßen werden. Diese Variante ist für die Umsetzung des Publikationsprozesses durchaus denkbar, da der deklarative Hauptprozess die variantenspezifischen YAWL-Netze zur Laufzeit auswählen kann. Dieser Weg widerspricht der Anforderung, den Publikationsprozess mit der Prozessmodellierungssprache YAWL umzusetzen (Anforderung 4). Weiterhin bleibt der Nachteil von Anforderung 4.

4.2 Der PROVOP Ansatz

In diesem Abschnitt wird der PROVOP Ansatz vorgestellt. Er stellt eine andere Möglichkeit dar, große Sammlungen von Prozessvarianten solide in einem Modell zu verwalten. PROVOP (**PRO**cess **V**ariants by **OP**tions) unterstützt viele Anforderungen für die Verwaltung von Prozessvarianten in allen Phasen des Process Life Cycle.

4.2.1 Modellierung von Prozessvarianten in PROVOP

Die Grundidee von PROVOP ist, alle Prozessvarianten in einem einzigen Modell zu erfassen. Prozessvarianten werden dazu von einem *Basisprozessmodell* abgeleitet. Die Erzeugung einer neuen Variante realisiert PROVOP durch die Anwendung sogenannter *Change Operations (COs)* auf das Basisprozessmodell. Change Operations beschreiben die Unterschiede zwischen dem Basisprozess und einer Prozessvariante. Jede Variante wird demnach durch eine Menge von COs repräsentiert. In [HBR08] stellt A. Hallerbach die vier grundlegenden Operationen vor:

1. **INSERT** fügt ein Prozessfragment bzw. -element zum Basisprozess hinzu. Die Operation kennt den Start- bzw. Endpunkt des Fragments und mappt diese auf entspre-

chende Elemente im Basisprozess.

2. **DELETE** entfernt Prozesselemente. Das Löschen erfordert die Anpassung der Elemente im Basisprozess, die den Start- und Endpunkt des Prozessfragments darstellen.

3. **MOVE** verändert die Ausführungsreihenfolge von Aktivitäten. Ein Prozessfragment kann auf diese Weise an eine gewünschte Position verschoben werden.

4. **MODIFY** verändert Attribute von ausgewählten Prozesselementen.

Die Erzeugung von Prozessvarianten erfordert oft die Anwendung mehrerer Change Operations. PROVOP definiert zu diesem Zweck *Options*, die mehrere COs in einem Objekt gruppieren. Abbildung 4.5A stellt den Basisprozess aus Abschnitt 3.4 für die Aggregation eines reinen Textdokumentes grafisch dar. Durch Anwendung der drei Change Operations von Option A in Abbildung 4.5B wird der Basisprozess an die Prozessvariante aus Abbildung 3.3B angepasst.

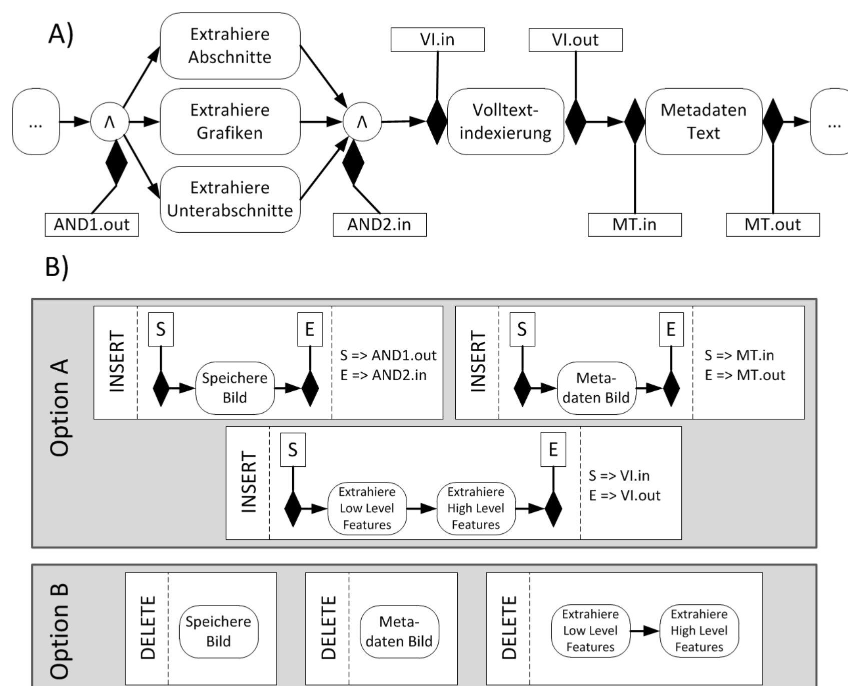


Abbildung 4.5: Prozessvarianten in PROVOP (Idee der Grafik aus [HB08])

Option A gruppiert zu diesem Zweck drei INSERT Operationen. Die Start- und Endpunkte der Prozessfragmente werden im Basisprozess durch schwarz ausgefüllte Rautensymbole

gekennzeichnet.
PROVOP bietet zusätzlich die Möglichkeit die Anwendung von Options auf ein Modell einzuschränken. Zwischen Options können folgende Beziehungen über Constraints realisiert werden:

1. **Abhängigkeit**: Zwei Options werden entweder beide zusammen oder gar nicht auf ein Modell angewendet.

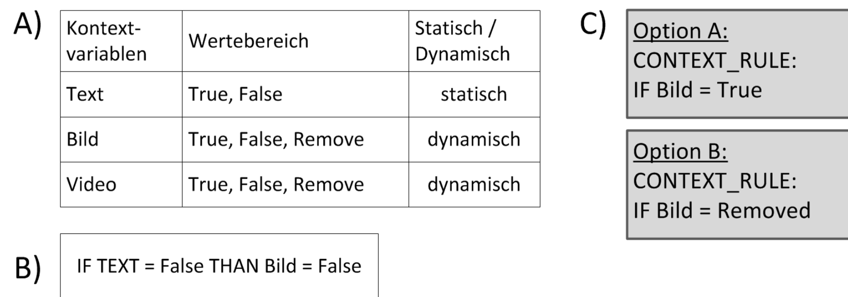
2. **Gegenseitiger Ausschluss**: Zwei Options dürfen jeweils einzeln, aber nicht beide zusammen auf ein Modell angewendet werden.

3. **Anwendungsreihenfolge**: Die Reihenfolge der Ausführung von Options ist in manchen Fällen entscheidend und kann entsprechend festgelegt werden. Option B aus Abbildung 4.5B macht die Änderungen von Option A rückgängig. Entsprechend sollte Option B erst ausgeführt werden, nachdem Option A angewendet wurde.

4.2.2 Prozesskonfiguration in PROVOP

PROVOP unterstützt die kontextbezogene Prozesskonfiguration. Der Kontext eines Prozesses wird über Kontextvariablen definiert, die einen vorgegebenen Wertebereich haben. Eine Variable ist entweder statisch oder dynamisch. Statische Kontextvariablen besitzen während der Ausführung des Prozesses einen festen Wert. Dynamische Kontextvariablen hingegen können sich zur Laufzeit ändern.
Ein Beispiel für die Definition von Kontext in PROVOP wird in Abbildung 4.6A gegeben. Die Variablen „Bild“ und „Video“ sind dynamisch und können ihren Wert zur Laufzeit ändern. Zusätzlich bietet PROVOP die Möglichkeit Beziehungen zwischen bestimmten Kontextvariablen zu definieren. Diese sind nach dem *IF-THEN-ELSE*-Prinzip aufgebaut. Die Regel aus Abbildung 4.6B definiert, dass wenn die statische Variable „Text“ zum Instanziierungszeitpunkt mit dem Wert „False“ belegt ist, die dynamische Variable „Bild“ ebenfalls den Wert „False“ besitzen muss. Der Variablenwert von „Bild“ kann sich zur Laufzeit wieder ändern. Hingegen bleibt der Wert von „Text“ immer gleich.

Kontextvariablen lösen die Anwendung von Options auf den Prozess aus. Options werden entweder bei der Instanziierung auf den Basisprozess oder zur Laufzeit auf eine Prozessvariante angewendet. Um den Prozesskontext mit Options zu verbinden, werden sogenannte *Context Rules* verwendet. Abbildung 4.6 bildet zwei Beispielregeln ab.



Kontext-variablen	Wertebereich	Statisch / Dynamisch
Text	True, False	statisch
Bild	True, False, Remove	dynamisch
Video	True, False, Remove	dynamisch

Abbildung 4.6: Modellierung von Kontext in PROVOP, vgl. [HB08]

Für Option A und B aus Abbildung 4.5B ist jeweils eine Regel definiert. Option A wird ausgelöst, wenn die Kontextvariable „Bild“ den Wert „True“ hat. Ist die Variable mit dem Wert „Removed“ belegt, löst dies die Anwendung von Option B aus. Die dynamische Variable Bild steuert somit die Anwendung beider Options auf eine laufende Prozessinstanz.

A. Hallerbach beschreibt in [HB08] die Instanziierung eines PROVOP Modells in vier Schritten. Der erste Schritt ist die Auswahl der Options für die Anwendung auf den Basisprozess. Diese werden entweder direkt vom User bestimmt oder anhand der aktuellen Werte der Kontextvariablen ausgewählt. Anschließend erfolgt die Auswertung der Abhängigkeiten zwischen den Options. An dieser Stelle wird der Nutzer über Inkonsistenzen informiert. Schließen sich beispielsweise zwei Options aus, muss der Nutzer den Konflikt beseitigen und eine der beiden Options entfernen.
Der vorletzte Schritt ist die Anwendung der Options auf den Basisprozess. Im letzten Schritt wird die entstandene Prozessvariante nach *Deadlocks* oder Dateninkonsistenzen untersucht.
Für weiterführende Information über die Re-Konfiguration einer Prozessvariante aufgrund von Kontextänderungen zur Laufzeit wird an dieser Stelle auf [HBR10] verwiesen.

4.2.3 Beurteilung des Ansatzes

Der PROVOP Ansatz erfüllt nahezu alle Anforderungen, die in Abschnitt 3.5 aufgestellt wurden. Das Modellierungskonzept minimiert den Modellierungs-, Änderungs- und Wartungsaufwand, vermeidet Redundanzen und gewährleistet die Übersichtlichkeit der Prozessbeschreibung (Anforderungen 1a,b und 1c). Prozesse können nicht nur vor der In-

stanziierung konfiguriert werden, sondern auch zur Laufzeit (Anforderungen 3a und 3b). Abbildung 4.7 präsentiert die Beurteilung von PROVOP.

Anforderung / Ansatz	Minimaler Modellierungs-, Änderungs-, Wartungsaufwand	Redundanzfreie Modellierung	Übersichtlichkeit der Prozessvarianten	Instanziierung nur erwünschter Prozessteile	Prozessvarianten zur Laufzeit auswählbar	Prozessvarianten zur Laufzeit austauschbar	Umsetzbar in YAWL
PROVOP	✓	✓	✓	✓	✓	✓	⛔

✓ erfüllt — teilweise erfüllt — ⛔ nicht erfüllt

Abbildung 4.7: Beurteilung des PROVOP Ansatzes

Die Umsetzbarkeit mit dem WfMS YAWL ist der einzige Kritikpunkt von PROVOP. Der Ansatz passt Prozessinstanzen zur Laufzeit an ein neues Modell an. YAWL verfügt über keine Möglichkeiten laufende Prozessinstanzen zu verändern.
Für die Umsetzung des Publikationsprozesses ist der PROVOP Ansatz aus einem weiteren Grund nicht ideal. Der Publikationsprozess kann sich zur Laufzeit sehr oft ändern, da ein Autor die Möglichkeit hat, sein Dokument immer wieder zu ändern. Das Anpassen von Prozessvarianten zur Laufzeit ist hauptsächlich für das Behandeln von Ausnahmesituationen gedacht. Beispielweise muss ein Unternehmen, aufgrund von Gesetzesänderungen, seinen Geschäftsprozess für die Bearbeitung von Versicherungsanträgen angleichen. Da es mitunter tausende laufende Anträge geben kann, ist es sinnvoll, laufende Prozessinstanzen anzupassen und nicht abzubrechen. Solche Maßnahmen stellen Ausnahmefälle dar und derartige Änderungen von Prozessinstanzen erforderen viel Aufwand. Daher wird der PROVOP Ansatz für die Umsetzung des Publikationsprozesses nicht berücksichtigt.

4.3 Configurable YAWL

Bevor die Prozesskonfiguration am Beispiel von YAWL beschrieben wird, wird im folgenden Unterabschnitt eine Einführung in die Prozessmodellierungssprache YAWL gegeben.

4.3.1 YAWL - Die Prozessmodellierungssprache

Die Sprache YAWL erweitert das Konzept von *Petrinetzen*. Petrinetze sind sehr ausdrucksstark in Bezug auf die Modellierung von Kontrollflüssen. Jedoch sind selbst *high-level-Petrinetze* als Workflow-Sprache nicht ausreichend.

Zum Beispiel besitzen high-level-Petrinetze keine effiziente Unterstützung von Mehrfach-Instanzen für Aktivitäten oder Cases.
Weiterhin ist die Modellierung von *Joins* schwierig. Ein Join bezeichnet hier die Zusammenführung meherer Zweige eines YAWL-Netzes in einen einzelnen Knoten (Aktivität). Die Zweige können entweder gar nicht (XOR-Join), teilweise (OR-Join) oder vollständig (AND-Join) synchronisiert werden. Petrinetze unterstzützen nur zwei Typen von Joins – den AND-Join einer Transition und den XOR-Join einer Stelle.
Als letztes Beispiel dient der Abbruch der Abarbeitung eines Prozesses aufgrund eines Fehlers. Die Modellierung eines *Cancellation Pattern* mit Petrinetzen ist mühsam, da sämtliche Tokens aus dem Netz entfernt werden müssen.[TV05]

Die Prozessmodellierungssprache YAWL bietet die effektive Handhabung obiger Sachverhalte. Abbildung 4.8 identifiziert sämtliche Sprachelemente mit denen der Kontrollfluss in YAWL umgesetzt wird.
Eine Workflow-Spezifikation in YAWL besteht aus einer Menge von YAWL-Netzen, welche in einer hierarchischen Struktur angeordnet sind. Es existieren *Atomic Tasks* und *Composite Tasks*. Eine Task kann als Transitionen in Petrinetzen interpretiert werden. Jede Composite Task verweist auf ein YAWL-Netz in einer tieferen Ebene der Hierarchie. Die letzte Ebene bilden die Atomic Tasks.
Weiterhin existieren *Conditions*, welche das Gegenstück zu Stellen in Petrinetzen repräsentieren. Jedes Workflow-Netz besitzt einmalig eine *Input Condition* als Startpunkt und eine *Output Condition* als Endpunkt. Im Gegensatz zu Petrinetzen können Tasks in einem YAWL-Netz direkt miteinander verbunden werden, ohne optisch eine Condition als Zwischenknoten zu passieren. Verbindet man zwei Aktivitäten auf direktem Weg, wird zwischen beiden Tasks eine versteckte Condition eingefügt.
Jede Task (entweder atomar oder zusammengesetzt) kann mehrfach instanziiert werden. Die Anzahl der Instanzen einer Multiple Instance Task kann zum Beispiel durch eine untere und optional eine oberen Grenze spezifiziert werden.
Eine andere Möglichkeit bietet die Angabe eines Grenzwertes. In dem Moment, in dem der Grenzwert überschritten wird, erfolgt die Terminierung aller laufender Instanzen und die Multiple Instance Task ist abgearbeitet. Für den Fall, dass kein Grenzwert spezifiziert wird, ist die Multiple Instance Task dann abgearbeitet, wenn alle ihre Instancen erfolgreich abgearbeitet wurden.
Ferner sei noch erwähnt, dass die Angabe über die Anzahl der Instanzen statisch und dy-

namisch erfolgen kann. Dazu existiert ein Parameter. Ist dieser dynamisch, ist es möglich, neue Instanzen hinzuzufügen solange laufende Instanzen existieren. Ein statischer Parameter erlaubt nur die Erzeugung der ursprünglich spezifizierten Anzahl von Instanzen.

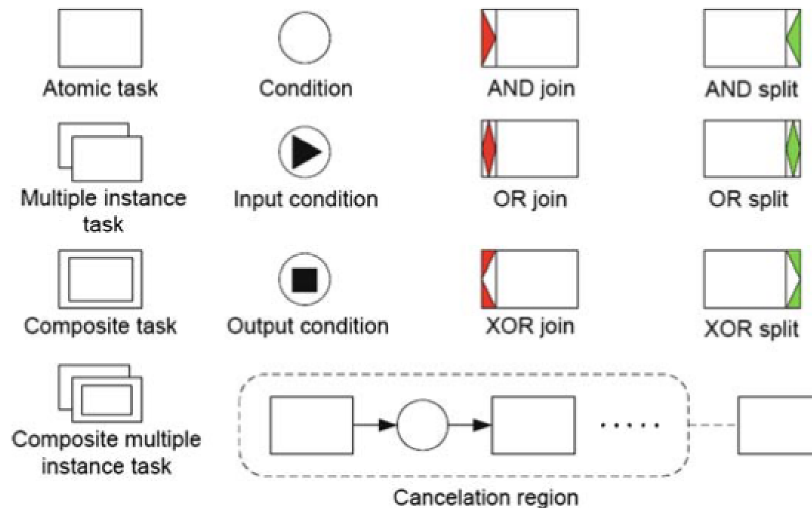


Abbildung 4.8: Symbole für den Kontrollfluss in YAWL, vgl. [TVAR10]

Für das Routing durch ein YAWL-Netz bietet die Sprache die Möglichkeit das *Join*- und *Split*-Verhalten einer Task zu spezifizieren. YAWL unterstützt die Konstrukte AND-Join, OR-Join, XOR-Join, AND-Split, OR-Split, und XOR-Split, welche wie folgt operieren [TVAR10]:

- **AND-Join** – Das Teilnetz hinter dem AND-Join wird erst dann ausgeführt, wenn alle in den AND-Join eingehenden Zweige eines Cases aktiviert wurden.

- **OR-Join** – Das Teilnetz hinter dem OR-Join wird erst dann ausgeführt, wenn entweder (1) jeder eingehende Zweig des Cases aktiviert wurde oder (2) es nicht möglich ist, dass jeder momentan noch nicht aktivierte eingehende Zweig zu einem Zeitpunkt in der Zukunft noch aktiviert wird.

- **XOR-Join** – Das Teilnetz hinter dem XOR-Join wird erst dann ausgeführt, wenn einer der in den XOR-Join eingehenden Zweige aktiviert wurde.

- **AND-Split** – Wenn der in den AND-Split eingehende Zweig aktiviert ist, dann werden alle dem AND-Split folgenden Zweige ausgeführt.

- **OR-Split** – Wenn der in den OR-Split eingehende Zweig aktiviert ist, dann werden einer oder viele der dem OR-Split folgenden Zweige ausgeführt. Die Auswahl erfolgt durch die Auswertung von Bedingungen, die mit jedem der ausgehenden Zweige des OR-Splits verknüpft sind.

- **XOR-Split** – Wenn der in den XOR-Split eingehende Zweig aktiviert ist, dann wird genau ein ausgehender Zweig des XOR-Splits ausgeführt. Auch hier erfolgt die Auswahl durch die Auswertung von Bedingungen, die mit jedem der ausgehenden Zweige des XOR-Splits verknüpft sind.

Zusätzlich realisiert YAWL die Idee von *Cancellation Regions*. Diese umspannen eine Gruppe von Conditions und Tasks in einem YAWL-Netz. Jede Cancellation Region ist mit einer bestimmten Task des selben YAWL-Netzes verknüpft. Die Task, welche mit der Cancellation Region verbunden ist, wird zur Laufzeit ausgeführt. Ist die Abarbeitung erfolgreich, werden alle aktuell laufenden Aktivitäten der entsprechenden Cancellation Region abgebrochen.

Bisher wurden nur Control-Flow-Elemente beschrieben. In einer Prozess-Spezifikation können zusätzlich Daten in Form von *Netzvariablen* und *Taskvariablen* definiert werden. Netzvariablen speichern Informationen für ein gesamtes Netz und können von beliebigen Tasks im gleichen Netz gelesen oder aktualisiert werden.
Taskvariablen hingegen nehmen Informationen bezüglich einer einzelnen Aktivität auf. Sie werden für mehrere Zwecke benötigt. Zum einen ermöglichen Taskvariablen die Übertragung von Informationen (Case-spezifischen Daten) zwischen Workflow-Teilnehmern und der Workflow Engine. Zum anderen dienten sie zum Datenaustausch zwischen einer Composite Task und dem assoziierten Subnetz. Hierbei werden die Taskvariablen der Composite Task auf die entsprechende Netzvariablen des gleichen Typs im Subnetz abgebildet.[Ter10]

4.3.2 Die Prozesskonfiguration

Die Prozesskonfiguration ist eine Methode, um Geschäftsprozesse an bestimmte Rahmenbedingungen anzupassen. Hierzu wird das mögliche Verhalten eines Prozessmodells vor der Ausführung eingeschränkt. Dies geschieht in einer *Konfigurationsphase* zwischen Modellierung und Ausführung. Auf diese Weise haben Designer zur Modellierungszeit

die Möglichkeit verschiedene Prozessvarianten in einem einzelnen Modell zu integrieren. Anschließend selektieren die Nutzer des Modells die relevanten Teile und passen das Prozessmodell an die eigenen Bedürfnisse an. Alle nicht benötigten Prozessteile werden vor der Ausführung automatisch aus dem Modell entfernt.

In [GVJVLR08] führen die Autoren zwei Operationen ein, um ein Prozessmodell zu konfigurieren bzw. einzuschränken. Sie ermöglichen entweder das *Blocken* oder das *Verstecken* einer Aktion in einem Workflow. Ist eine Aktion geblockt, kann sie nicht mehr ausgeführt werden. Weiterhin stoppt der Prozess die Abarbeitung des nachfolgenden Prozessteils und erreicht somit keine Folgeaktionen oder Folgezustände mehr.
Das Verstecken einer Aktion bedeutet, dass sie ausgelassen bzw. nicht ausgeführt wird. Eine versteckte Aktion (Silent Action) verbraucht weder Zeit noch Ressourcen. Im Gegensatz zur geblockten Aktion setzt der Prozess die Abarbeitung nach einer Silent Action fort und erreicht Folgezustände, die hinter der versteckten Aktion liegen.

YAWL repräsentiert Aktionen in einem Prozessmodell durch Aktivitäten (Tasks). Die Konfiguration eines YAWL-Modells erfolgt demnach über die Konfiguration der einzelnen Aktivitäten. YAWL-Tasks erlauben die Spezifikation von verschiedenen Verhaltensweisen, wie die Anwendung verschiedener „Split“ und „Join“ Patterns, den Start von „Mehrfachinstanzen“ (Multiple Instances) und den Abbruch von anderen Aktivitäten durch „Cancellation Regions“. Folglich ist das Blocken oder Verstecken der kompletten Task nicht ausreichend. Vielmehr muss identifiziert werden, wann und wie eine Aktivität den Zustand eines Prozesses ändern kann. Hierzu werden die Alternativen, um eine Task auszulösen und zu beenden, genauer betrachtet. Der nächste Unterabschnitt stellt das Prinzip der Input- und Output-Ports vor.

Konfiguration von Input- und Output-Ports

Die Abarbeitung einer Task wird durch eine Kombination ihrer eingehenden Kanten ausgelöst. Beispielsweise besitzt eine Task mit einem XOR-Join mehrere eingehende Kanten von davorliegenden Conditions (Pre-Conditions). Für die Auslösung dieser Task braucht nur eine der Conditions markiert sein. Folglich repräsentiert jede eingehende Kante eine andere Alternative, um den Zustand des Prozesses zu verändern. Jede Alternative, mit der eine Task gestartet werden kann, wird *Input Port* genannt. Die Input Ports einer Task mit XOR-Join sind folglich konfigurierbar. Die linke Seite von Abbildung 4.9B bildet die

Input Ports für eine Task mit XOR-Join ab.

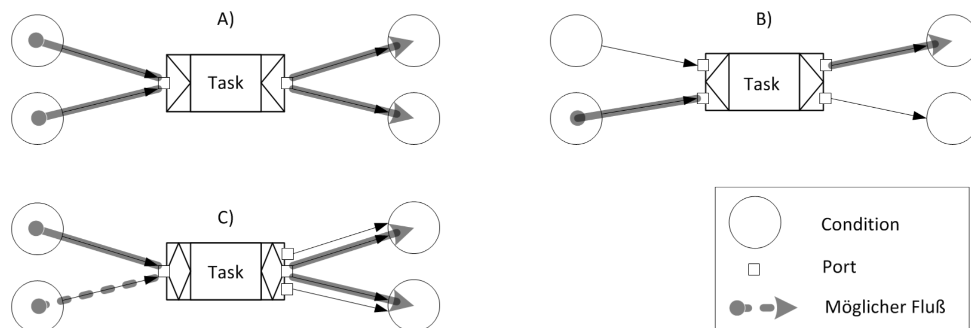


Abbildung 4.9: Input- und Output Ports einer YAWL-Task, vgl. [TVAR10]

Aktivitäten mit einem AND-Join besitzen immer nur einen Input Port, da die eingehenden Kontrollflüsse synchronisiert werden. Nur die Markierung aller Pre-Conditions (linke Seite von Abb. 4.9A) löst die Abarbeitung der Task aus. Es existiert demnach nur eine Kombination eingehender Kanten, um den Zustand des Prozesses zu verändern. Für eine Task mit AND-Join ist nur ein einzelner Input Port konfigurierbar. Gleiches gilt für eine Aktivität mit OR-Join, mit dem Unterschied, dass nicht alle Pre-Conditions markiert sein müssen (siehe linke Seite von Abb. 4.9C).
Das Blockieren eines Input Ports bedeutet, dass die Task durch diesen Port nicht ausgeführt werden kann. Die Task verbraucht keine Markierungen von Pre-Conditions. Ein versteckter Input Port hingegen ist in der Lage Markierungen zu konsumieren und weiterzuleiten. Die Aktivität selbst wird nicht ausgeführt. Es ändert sich dennoch der Zustand des Prozesses. [TVAR10]

Input Ports sind verantwortlich für Zustandsänderungen des Prozesses, da sie die Ausführung einer Task ermöglichen. Auf welche Weise sich der Prozess verändert ist abhängig vom Ergebnis der Ausführung einer Aktivität, welches von dem „Split"-Verhalten einer Task bestimmt wird. Beispielsweise markiert ein XOR-Split nur eine nachfolgende Condition (Post-Condition). Folglich entspricht jede Post-Condition einer unterschiedlichen Zustandsänderung des Prozesses. Ein *Output Port* entspricht einer konkreten Markierung von Post-Conditions. Eine Aktivität mit XOR-Split besitzt für jede Folgestelle einen Output-Port (siehe rechte Seite von Abb. 4.9B).
Ein OR-Split hingegen ist in der Lage jede mögliche Kombination von Post-Conditions zu markieren. Das Resultat ist ein immer anderer Prozesszustand. Zum Beispiel besitzt eine

Task mit OR-Split sieben Ouput Ports, wenn drei Post-Conditions zur Auswahl stehen. Ein AND-Split markiert sämtliche Folgestellen nach seiner Ausführung. Folglich hat eine Task mit AND-Split nur einen einzigen Output Port, der eine Zustandsänderung hervorruft. Die rechte Seite von Abbildung 4.9A stellt diesen Sachverhalt dar.
Output Ports können ebenfalls konfiguriert werden. Ein geblockter Port verhindert die Markierung einer entsprechenden Menge von Post-Conditions. Im Gegensatz zu Input Ports können Output Ports nicht versteckt werden. Für Informationen über die Konfiguration von Mehrfachinstanzen und Cancellation Regions wird auf [TVAR10] verwiesen.

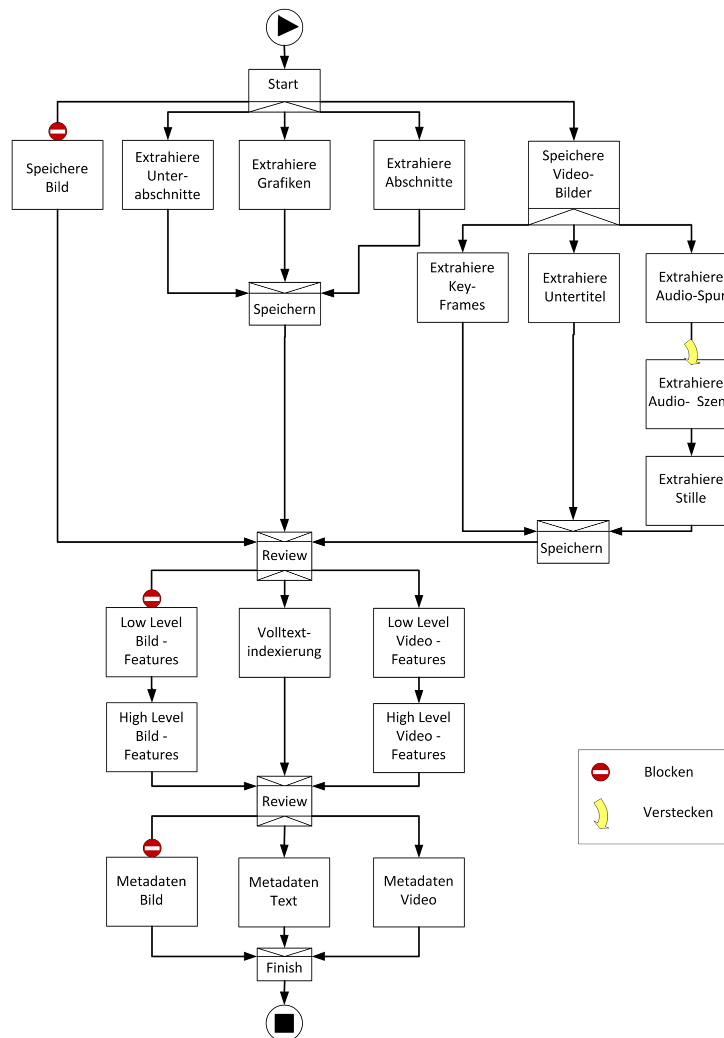


Abbildung 4.10: Konfigurierter YAWL Prozess, vgl. [TVAR10]

Abbildung 4.10 illustriert einen Beispielprozess in YAWL für die Phase der Aggregation.

Der Prozess modelliert die Prozessvarianten aus Abbildung 3.3B und 3.3C in einem Prozessmodell. Durch die Konfiguration ist die Auswahl einer bestimmten Prozessvariante möglich. In diesem Fall ist der Prozess für die Bearbeitung eines Textdokuments mit integriertem Video konfiguriert. Folglich werden die Input Ports der Aktivitäten „Speichere Bild“, „Low Level Bild – Features“ und „Metadaten Bild“ geblockt.
Abbildung 4.10 spezifiziert kein Join-Verhalten für mehrere Aktivitäten. Als Standard besitzt jede YAWL-Task ein XOR-Join, wenn kein anderes Join-Verhalten ausdrücklich erwünscht ist. Weiterhin ist die Aktivität „Extrahiere Audio-Szene“ versteckt. Aus einer Audiospur werden somit nur geräuschfreie Momente extrahiert und keine Audio-Szenen.

Die Prozesskonfiguration verfolgt die Absicht, Teile aus dem Prozessmodell zu entfernen, die für einen bestimmten Kontext irrelevant sind. Bisher wurde nur beschrieben, wie diese Absichten durch Konfiguration ausgedrückt werden können. Das Entfernen unerwünschter Teile aus einem Prozessmodell anhand von Konfigurationsentscheidungen wird im nachfolgenden Abschnitt kurz erläutert.

Das Entfernen unerwünschter Prozessteile

Unerwünschte Prozessteile können automatisch in zwei Phasen aus einem YAWL-Modell entfernt werden. Der erste Schritt betrachtet direkte Folgerungen der Konfiguration auf Tasks und Kontrollflüsse. Diese unterscheiden sich je nach Join- bzw. Split-Verhalten einer Aktivität.
Das Split-Verhalten einer Aktivität wird durch die Output Ports beeinflusst. Eine Task mit XOR-Split besitzt für jede Post-Condition einen Port. Für den Fall, dass einer der Output Ports geblockt ist, kann die ausgehende Kante entfernt werden. Abbildung 4.11B stellt diesen Sachverhalt grafisch dar.
Im Gegensatz zu einem XOR-Split besitzt eine Aktivität mit AND-Split nur einen einzigen Output Port. Ist dieser geblockt, werden alle Kanten zu den Folgestellen der Task entfernt (siehe Abb. 4.11A). Ein OR-Split hat für jede Kombination der Post-Conditions einen Output Port. Mehrere Output Ports verweisen folglich auf eine ausgehende Kante. Das Entfernen einer Kante ist demnach nur möglich, wenn alle Ports geblockt sind, die auf diese Kante verweisen.
Input Ports können entweder aktiv, geblockt oder versteckt sein. Abbildung 4.11C bildet die Auswirkung eines geblockten Input Ports eines AND-Joins ab. Entsprechend werden alle eingehenden Kanten von Pro-Conditions entfernt. Analog zum XOR-Split verhält sich

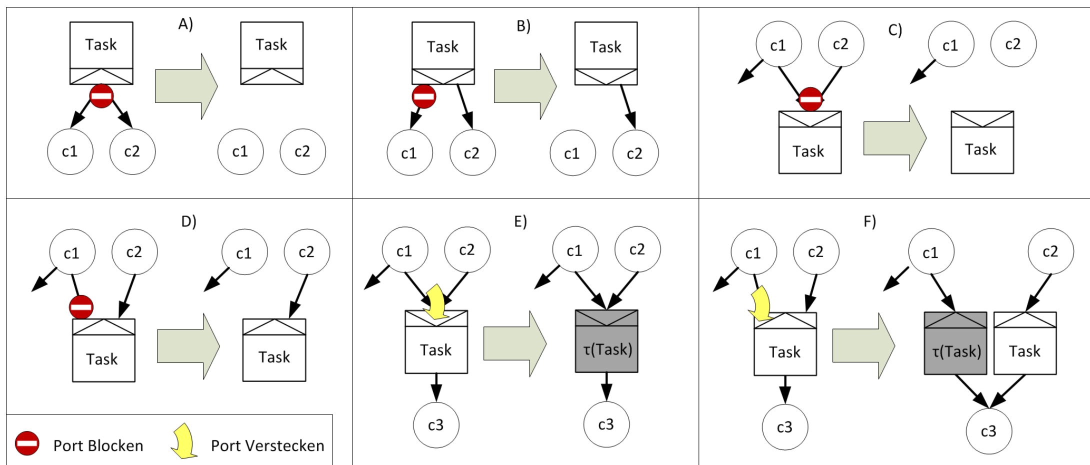


Abbildung 4.11: Auswirkung der Konfiguration auf YAWL-Tasks, vgl. [TVAR10]

der XOR-Join, wenn ein Port geblockt ist (siehe Abb. 4.11D).
Bei einem versteckten Input Port dürfen die eingehenden Kanten nicht entfernt werden, da der Prozess nach der Aktivität fortsetzen soll. Bei einer YAWL-Task mit AND-Join reicht es aus, die *Decomposition* der Aktivität zu entfernen. Eine solche *Silent Task* wird in der Folge nur durchlaufen und nicht ausgeführt. Abbildung 4.11E symbolisiert die Silent Task durch den Bezeichner τ. Ein OR-Join verhält sich identisch mit einem AND-Join. Hingegen hat ein XOR-Join mehrere Input Ports. Ist einer der Ports versteckt, wird eine Kopie der Task benötigt. Alle aktiven eingehenden Kanten werden in diese Kopie geleitet. Zuletzt wird die Decomposition der Task entfernt (siehe Abb. 4.11F). Die τ-Task und die Kopie müssen beide die selben Post-Conditions markieren.

In der zweiten Phase werden alle Folgerungen der Konfiguration auf andere Elemente in dem Workflow aufgelöst. Liegt ein Element nach dem Entfernen der Kanten auf einem Pfad zwischen der Input- und Output Condition, bleibt es im Prozessmodell enthalten. In Abbildung 4.10 befinden sich die Aktivitäten „Speichere Bild", „Low Level Bild – Features", „High Level Bild – Features" und „Metadaten Bild" nicht auf einem derartigen Pfad und werden aus dem finalen Modell entfernt. Für eine weiterführende Beschreibung dieser Phase findet sich in [TVAR10].

4.3.3 Beurteilung der Prozesskonfiguration

Die Prozesskonfiguration ist für die Umsetzung des Publikationsprozesses ungeeignet, da sie mehreren Anforderungen nicht genügt. Die Konfiguration findet in einer Phase zwischen Modellierung und Instanziierung statt. Diese Tatsache macht das Auswählen von Prozessvarianten zur Laufzeit unmöglich (Anforderung 3a). Dementsprechend können während der Ausführung des Prozesses ebenfalls keine Varianten ausgetauscht werden (Anforderung 3b). Durch das Entfernen nicht erwünschter Prozessteile erfüllt die Prozeskonfiguration in YAWL Anforderung 2.
Abbildung 4.12 fasst die Tauglichkeit von configurable YAWL tabellarisch zusammen.

Anforderung / Ansatz	Minimaler Modellierungs-, Änderungs-, Wartungsaufwand	Redundanzfreie Modellierung	Übersichtlichkeit der Prozessvarianten	Instanziierung nur erwünschter Prozessteile	Prozessvarianten zur Laufzeit auswählbar	Prozessvarianten zur Laufzeit austauschbar	Umsetzbar in YAWL
C-YAWL	erfüllt	erfüllt	teilweise erfüllt	erfüllt	nicht erfüllt	nicht erfüllt	erfüllt

Legende: erfüllt · teilweise erfüllt · nicht erfüllt

Abbildung 4.12: Beurteilung von C-YAWL

4.4 Die Unterspezifikation

Während PROVOP die Anpassung von Prozessinstanzen zur Laufzeit ermöglicht, verfolgt die Unterspezifikation einen gegenteiligen Ansatz. M. Schoneberg definiert die Unterspezifikation in [SMR+07] wie folgt:

> *"Flexibility by underspecification* is the ability to execute an incomplete process specification at run-time, i.e. one which does not contain sufficient information to allow it to be executed to completion."

Ein unterspezifiziertes Prozessmodell benötigt keine Anpassung zur Laufzeit. Vielmehr komplettieren konkret realisierte Prozessteile die undefinierten Stellen laufender Prozessinstanzen.
Verwendet wird die Unterspezifikaiton für Prozessbeschreibungen, bei denen bekannt ist, dass sie an bestimmten Punkten angepasst werden müssen. Der genaue Inhalt dieser Punkte ist nicht zur Modellierungszeit, aber zur Laufzeit bekannt. Die Unterspezifikation

ermöglicht das Entwerfen unterschiedlicher Prozessteile durch verschiedene Arbeitsgruppen, während die Grundstruktur des Basisprozesses fest ist.

4.4.1 Realisierung der Unterspezifikation

Ein unvollständiger Prozess besitzt nicht für jede Aktivität eine genaue Realisierung. Diese Aktivitäten werden *Placeholders* genannt. Placeholders stellen die unspezifizierten Teile des Basisprozesses dar, deren Inhalt erst während der Ausfühung spezifiziert wird. Es existieren zwei Methoden, um Placeholders auszuführen.

- Das **Prinzip des Late Binding** ist die Realisierung eines Placeholders durch Auswahl eines Prozessfragments aus einer Menge vollständig verdefinierter Prozessfragmente. Diese Methode beschränkt sich auf die Auswahl und erlaubt keine Konstruktion neuer Prozessfragmente.

- Das **Prinzip des Late Modelling** ermöglicht die Konstruktion eines neuen Prozessfragments, um einen gegebenen Placeholder zu realisieren. Es schließt das Prinzip des Late Bindings ein. Komplexe Prozesselemente können sowohl aus einer vordefinierten Menge von Prozessfragmenten konstruiert, als auch völlig neu entwickelt werden.

Abbildung 4.13 stellt das Prinzip der Unterspezifikation dar. Die linke Seite der Abbildung zeigt einen unterspezifizierten Prozess mit einem Placeholder (gekennzeichnet durch die *Black Box*). Auf der rechten Seite befindet sich der Prozess nach der Vervollständigung durch das Einbinden eines Prozessfragments.

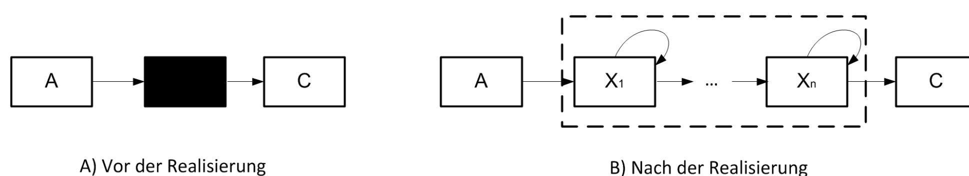


Abbildung 4.13: Flexibilität durch Unterspezifikation, vgl. [SMR+07]

Ein Prozess kann an mehreren Stellen unterspezifiziert sein. Entsprechend oft muss ein Prozessfragment zur Laufzeit an die entsprechenden Stellen eingebunden werden. Es gibt zwei unterschiedliche Arten, um eine unterspezifizierte Aktivität zu realisieren:

- **Before Placeholder Execution**: Der Placeholder wird am Start der Prozessinstanz oder während der Ausführung realisiert, aber bevor der Placeholder das erste mal abgearbeitet wird.

- **At Placeholder Execution**: Der Placeholder wird bei seiner Ausführung realisiert.

Weiterhin kann die Realisierung eines Placeholder nach dem Late Binding oder Late Modelling Prinzip statisch oder dynamisch erfolgen. Ersteres legt das Prinzip bei der ersten Placeholdersausführung für alle nachfolgenden unterspezifizierten Punkte fest. Eine dynamische Realisierung hingegen ermöglicht die Auswahl des Prinzips für jeden Placeholder im Prozessmodell von neuem.

4.4.2 Pockets of Flexibility

In [SSO01] stellen die Autoren ein Konzept vor, welches weiterführende Ideen zur Unterspezifikation durch Late Modelling präsentiert. Ausgangspunkt der Überlegungen ist ein Basisprozess bestehend aus Workflow-Aktivitäten und sogenannten *Pockets of Flexibility.* Jedem Pocket wird eine Menge von Prozessfragmenten und eine spezielle Aktivität mit dem Namen *Build-Aktivität* zugeordnet. Build-Aktivitäten stellen detailliertere Placeholders dar. Sie enthalten zusätzliche Regeln, um eine unspezifizierte Stelle mit einer konkreten Komposition verschiedener Prozessfragmente zu ersetzen. In Abbildung 4.14A wird die Build-Aktivität grafisch dargestellt. Sie enthält eine Menge von Prozessfragementen symbolisiert durch mehrere Rechtecke mit den Beschriftungen X_1 bis X_n. Hinter jedem Rechteck verbirgt sich eine einzelne Aktivität oder ein komplettes Subnetz (Fragment).

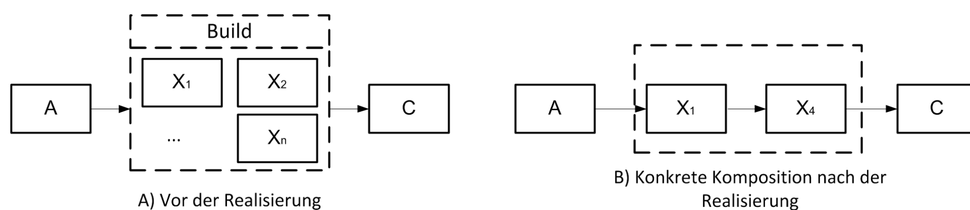


Abbildung 4.14: Spezifikation einer Pocket of Flexibility, vgl. [SSO01]

Abbildung 4.14B zeigt eine mögliche konkrete Realisierung der Build-Aktivität. Sie besteht aus einer sequenziellen Komposition der Prozessfragmente X_1 und X_4. Ein derartig vervollständigter Basisprozess wird als *Instance Template* bezeichnet. Instance Templates stellen die unterschiedlichen Prozessvarianten dar .

Es existieren drei verschiedene Methoden, mit denen eine Build-Aktivität ein Instance Template erzeugen kann. Sie werden in der nachfolgenden Aufzählung vorgestellt.

1. Eine Build-Aktivität baut ein Instance Template automatisch aus einer gegebenen Menge von Prozessfragmenten und nur auf der Basis von Prozessdaten und vordefinierten Nebenbedingungen (Constraints).
2. Eine Build-Aktivität ruft eine Applikation auf, die es einem Workflow-Clienten ermöglicht, Instance Templates aus einer gegeben Menge von Prozessfragmenten und anhand von Constraints zu erschaffen.
3. Die Build-Aktivität erlaubt einem Workflow-Client die Definition neuer Prozessfragmente und somit die Erschaffung von Instance Template aus neuen und existierenden Fragmenten.

Wie bereits erwähnt, bestimmen Constraints die Art und Weise, wie neue Prozessfragmente aus Bestehenden zusammengesetzt werden können. Die Art der Komposition kann von vielen Gründen abhängen, wie zum Beispiel von prozessbezogenen Daten, dem Prozesszustand, zeitlichen Einschränkungen oder aktuell verfügbaren Prozessfragmenten. In [SOS05] werden **strukturelle** und **Containment** Constraints unterschieden. Erstere bestimmen die Anordnung von Prozessfragmenten. Abbildung 4.15 gibt einen Überblick über die Möglichkeiten, um die Komposition der drei Prozessfragmente A, B und C festzulegen.

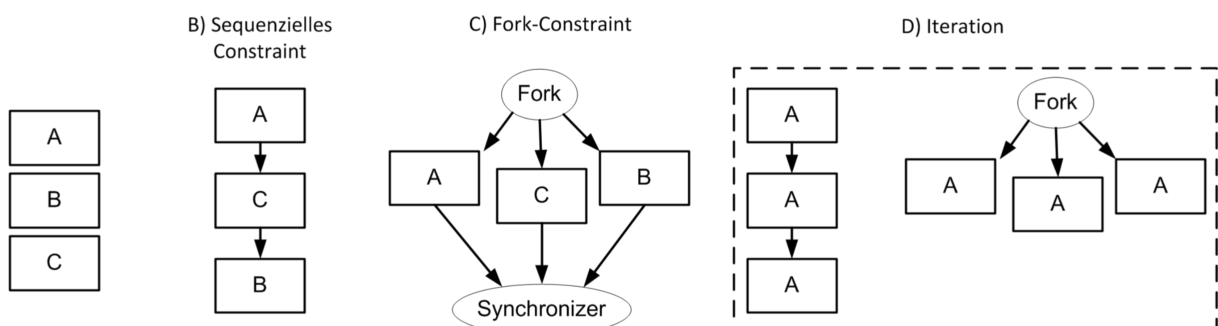


Abbildung 4.15: Kompositionsmöglichkeiten einer Build-Aktivität, vgl. [SSO01]

Eine sequenzielles Constraint erzwingt die serielle Anordnung von Prozessfragmenten. Die Reihenfolge der Abarbeitung ist nicht fest vorgegeben. Zusätzlich existiert das Order-Constraint als Spezialfall der sequenziellen Abarbeitung. In diesem Fall ist die Ausführungsreihenfolge der Fragmente entscheidend und muss eingehalten werden. Hierbei wird

nicht impliziert, dass die beteiligten Fragmente direkt hintereinander platziert werden müssen.
Das *Fork*-Constraint ordnet Prozessfragmente nebenläufig an. Parallele Zweige müssen an einem bestimmten Punkt wieder synchronisiert werden. Hierzu dient der *Synchronizer*. Iterationen können ebenfalls als Sequenz oder Fork realisiert werden.
Sogenannte „Choice“ and „Merge“ Konstrukte schließen die Autoren in [SOS05] aus, um ein Instance Template zu konstruieren. Grundsätzlich werden alle Entscheidungen während der Ausführung der Build-Aktivität getroffen. Es besteht folglich nicht die Notwendigkeit derartige Entscheidungen in ein Instance Template zu verlagern.
Containment Constraints beschreiben Bedingungen, unter denen ein Prozessfragement in einem Instance Template vorkommen darf oder nicht. Es wird unterschieden zwischen *Inclusion* und *Exclusion* Constraints. Ersteres beschreibt eine Abhängigkeit zwischen zwei Fragmentmengen, nach der die Präsenz (oder Absenz) von Fragmenten der einen Menge die Anwesenheit (oder Abwesenheit) der Fragmente der zweiten Menge bestimmt. Das Exclusion Constraint beschreibt ähnlich zum Inclusion Constraint das gegenteilige Verhalten.

4.4.3 Beurteilung der Unterspezifikation

Die Unterspezifikation nach dem Late Modelling Prinzip ist ideal geeignet, um den Publikationprozess umzusetzen. Prozessvarianten werden zur Laufzeit durch Vervollständigung eines lose spezifizierten Prozessmodells erschaffen (Anforderung 3a). Durch Loops können Placeholder Aktivitäten mehrfach ausgeführt und somit Prozessvarianten ausgetauscht werden (Anforderung 3b). Die Vervollständigung ermöglich das Instanziieren der wirklich benötigten Prozessteile anhand von Kontextinformationen (Anforderung 2). Eine abschließende Beurteilung der Unterspezifikation wird in Abbildung 4.16 gegeben.

Anforderung / Ansatz	Minimaler Modellierungs-, Änderungs-, Wartungsaufwand	Redundanzfreie Modellierung	Übersichtlichkeit der Prozessvarianten	Instanziierung nur erwünschter Prozessteile	Prozessvarianten zur Laufzeit auswählbar	Prozessvarianten zur Laufzeit austauschbar	Umsetzbar in YAWL
Unterspezifikation	✓	✓	✓	✓	✓	✓	teilweise erfüllt

✓ erfüllt · teilweise erfüllt · nicht erfüllt

Abbildung 4.16: Beurteilung der Unterspezifikation

Die Umsetzung mit dem WfMS YAWL ist wieder der einzige Krititpunkt. YAWL implementiert die Unterspezifikation nach dem Late Binding Prinzip durch den Worklet Service (siehe Abschn. 4.5.1).
Das Late Binding von Prozessfragmenten ist für den Publikationsprozess nicht geeignet. Dies würde eine vordefinierte Menge aller möglichen Prozessvarianten für jeden Placeholder erfordern. Bezogen auf ein Multimediadokument wird ein Prozessfragment für jeden Placeholder und jede mögliche Zusammensetzung des Dokuments benötigt. Dies bedeutet, es existieren separate Prozessvarianten für Textdokumente mit Bild, Video und beiden Komponenten. Letzteres besteht aus den Komponenten Bild und Video und wäre demnach redundant vertreten. Dies stellt einen Wiederspruch zu Anforderung 1b dar.
Vielmehr sollen komplexe Prozessvarianten aus elementaren Fragmenten zusammengebaut werden können. Diese Fähigkeit besitzt das WfMS YAWL nicht. Es kann aber mit einem Custom YAWL Service (siehe Abschn. 6.2 und [DF]) um eine derartige Funktion erweitert werden.

4.5 Kombination von Unterspezifikation und Prozesskonfiguration

Sowohl die Prozesskonfiguration, als auch die Unterspezifikation nach dem Late Binding Prinzip können mit YAWL realisiert werden. Daher stellt sich die Frage, ob die Kombination beider Ansätze eine Möglichkeit darstellt, den Publikationsprozess umzusetzen ohne die Notwendigkeit eines Custom YAWL Services. Vorab wird zu diesen Zweck der Worklet Service kurz erläutert.

4.5.1 Der Worklet Service

Der *Worklet Dynamic Process Selection & Exception Handling Service* umfasst zwei verschiedene Dienste [Ter10]:

- Der **Selection Service** ermöglicht die Substitution eines Work Items zur Laufzeit durch ein dynamisch ausgewähltes *Worklet*. Ein Worklet ist ein einzelner YAWL-Prozess, welcher als Subnetz für das Work Item agiert.

- Mit dem **Exception Handling Service** ist YAWL in der Lage erwartete und unerwartete Ausnahmesituationen (Exceptions) zur Laufzeit zu behandeln. Unerwartete

Exceptions sind Ereignisse oder Vorkommnisse, die während der Lebenszeit einer Prozessinstanz auftreten können und nicht im Prozess modelliert wurden.

Für die Behandlung von Prozessvarianten ist der Selection Service von Interesse. Wie bereits erwähnt, implementiert der Worklet- bzw. der Selection Service die Unterspezifikation nach dem Late Binding Prinzip. Eine Placeholder-Task wird zur Modellierungszeit mit dem Worklet Service assoziiert. Zur Laufzeit übernimmt der Service als Workflow-Teilnehmer die Ausführung der Aktivität.
Der Service verwaltet eine Menge von Worklets. Die Auswahl eines bestimmten Worklets erfolgt auf Basis von Kontextdaten innerhalb des Work Items, welches vom Worklet Service entgegen genommen wird.
Der Worklet Service erlaubt die Substitution von atomaren Tasks und „Multiple-Instance“ Tasks. Für letzteres wird zu jedem „Kind Work Item“ ein separates Worklet gestartet, da jedes Kind unterschiedliche Daten enthalten kann. Für detailierter Infomationen zur Auswahl von Worklets bzw. über den Exception Service wird an dieser Stelle auf [TVAR10] und [Ter10] verwiesen.

4.5.2 Beurteilung der Kombination

Eine Kombination beider Verfahren bedeutet, dass in einem unterspezifizierten Prozess konfigurierbare Worklets eingesetzt werden können. Die Konfiguration der Worklets erfolgt zum Zeitpunkt der Substitution durch den Workflow-Teilnehmer anhand von Kontextinformationen.
In der Theorie bietet diese Möglichkeit alle Vorraussetzungen, um den Publikationsprozess umzusetzen. Das Problem ist die automatische Konfiguration der Worklets zur Laufzeit, ohne die Einbeziehung menschlicher Workflow-Teilnehmer. Diese Funktionalität bietet die Prozesskonfiguration in YAWL nicht. Folglich wird ein Custom Service benötigt, der diese Aufgabe realisiert.

In diesem Kapitel wurde herausgestellt, dass die Umsetzung des Publikationsprozesses mit zwei verschiedenen Verfahren möglich ist.

- Entwicklung eines Custom YAWL Services, der die Unterspezifikation nach dem Late Modelling Prinzip umsetzt.

- Entwicklung eines Custom YAWL Services, der die automatisierte Konfiguration von Worklets ermöglicht.

Beide Verfahren erfordern die Erweiterung des WfMS YAWL um einen zusätzlichen Custom YAWL Service. Im nachfolgenden Kapitel wird ein Konzept für die Umsetzung des ersten Verfahren vorgestellt.

Kapitel 5

Konzept für die Umsetzung des Publikationsprozesses

Im vorherigen Kapitel wurden Ansätze vorgestellt, mit denen Prozessvarianten effektiv modelliert und ausgeführt werden können. Für die Umsetzung des Publikationsprozesses wird der Ansatz der Unterspezifikation nach dem Late Modelling Prinzip gewählt, da dieser den in Abschnitt 3.5 aufgestellten Anforderungen am besten genügt.

Dieses Kapitel stellt ein Konzept für die Umsetzung vor, welches sich an dem Konzept der „Pockets of Flexibility“ (siehe Abschn. 4.4.2) orientiert. Ausgangspunkt der Unterspezifikation ist ein lose spezifiziertes Basis-Prozessmodell. Es modelliert alle Prozessteile des Publikationsprozesses, die für sämtliche Prozessvarianten gleich sind. Weiter enthält das Basis-Prozessmodell bestimmte Punkte, die sich von Variante zu Variante unterscheiden. In [SOS05] verwenden die Autoren den Begriff „Build-Aktivität“, um derartige Stellen zu benennen. Im Folgenden wird hierfür die Bezeichnung *Konfigurationspunkt* gewählt. Abbildung 5.1 bildet einen beliebigen Prozess mit vier Konfigurationspunkten ab.

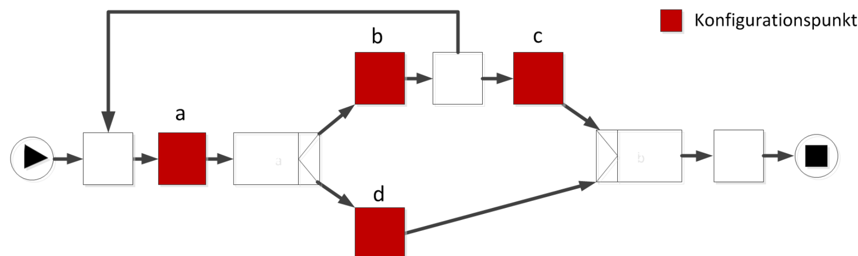


Abbildung 5.1: Prozess mit vier Konfigurationspunkten

Jeder Punkt steht im Publikationsprozess für eine content-abhängige Bearbeitung, wie

z. B. die Konvertierung oder Aggregation verschiedener Objekte (z.B. Text, Bild, ...).

5.1 Das Konzept

Das Basis-Prozessmodell enthält eine Menge C von Konfigurationspunkten. Jeder Konfigurationspunkt $c \in C$ ermöglicht das Einbinden von beliebig vielen Komponenten. Eine Komponente $k \in K$ ist ein gültiges Prozessmodell (Fragment). Konfigurationspunkte beschreiben ein Zweitupel:

$$c = (K, R)$$

K ist eine Menge von Komponenten und enthält alle Prozessfragmente, die an dem assoziierten Konfigurationspunkt eingebunden werden können. Im Publikationsprozess müssen häufig mehrere Komponenten in einen Konfigurationspunkt eingesetzt werden. Zum Beispiel benötigt die Aggregation eines Textdokuments mit Bild und Video insgesamt drei Prozessfragmente, welche die Aggregation der drei Content-Typen realisieren. Um mehrere Komponenten in einen Konfigurationspunkt einzubinden, müssen diese vorher miteinander komponiert werden. Die Menge R behinhaltet verschiedene Regeln (Constraints), welche den Zusammenschluss der Prozessfragmente einschränken. Für die Umsetzung des Publikationsprozesses werden zwei mögliche Constraints definiert. Die „**serielle Ordnung**“ beschreibt eine Abhängigkeit zwischen n unterschiedlichen Prozessfragmenten durch:

$$S(k_1, ..., k_n), \quad n \in \mathbb{N}$$

Das Constraint erzwingt die Hintereinanderausführung der n Prozessfragmente in Leserichtung. Derartige Regeln werden beispielsweise benötigt, wenn eine Komponente auf die Vorarbeit oder zusätzliche Daten eines anderen Prozessmodells angewiesen ist. Die serielle Ordnung entspricht dem in [SOS05] vorgestellten Order-Constraint mit der Einschränkung, dass die Komponenten direkt hintereinander angeordnet werden.
Das zweite Constraint für den Publikationsprozess ist das Fork-Constraint, das im Folgenden die Bezeichnung „**parallele Komposition**“ erhält. Es erzwingt die nebenläufige Anordnung der Fragmente, für die es definiert wird. Die Notation für eine parallele Komposition zwischen n Komponenten ist:

$$P(k_1, ..., k_n), \quad n \in \mathbb{N}$$

Die parallele Anordnung von Prozessfragmenten ist *kommutativ*. Jedes Fragment entspricht einem nebenläufigen Zweig im komponierten Prozessmodell. Eine beliebige Vertauschung der Komponenten k_1 bis k_n bewirkt, dass die Zweige untereinander verschoben werden. Es existiert keine Ordnung zwischen den Zweigen, da das Ergebnis der Ausführung des komponierten Prozessmodells für jede mögliche Anordnung gleich ist. Für zwei Komponenten k_1 und k_2 gilt:

$$P(k_1, k_2) = P(k_2, k_1)$$

Drei Prozessfragmente können bereits in sechs Varianten angeordnet werden, welche alle das äquivalente Ausführungsergebnis erzielen.
Eine leere Regelmenge R bewirkt automatisch die parallele Komposition aller ausgewählten Fragmente. Gleichermaßen wird eine Komponente, die in keiner Regel vertreten ist, nebenläufig zu allen anderen bestimmten Prozessmodellen angeordnet. Die parallele Komposition kann demnach explizit gefordert werden, um beispielsweise die sequenzielle Komposition zu verhindern.

Das anschließende Beispiel beschreibt die Anwendung von Constraints genauer. Für einen Konfigurationspunkt $c \in C$ seien die Mengen K und R definiert durch:

$$K = \{k_1, ..., k_n\}, \quad n \in \mathbb{N}$$

$$R = \{S(k_1, k_2), P(k_1, k_3), S(k_4, k_2), ...\}$$

Zur Laufzeit wurde ermittelt, dass die Komponenten k_1 bis k_4 in den Konfigurationspunkt c eingebunden werden sollen. Die Menge R spezifiziert für diese Prozessmodelle drei Regeln. Alle weiteren Regeln beziehen sich auf andere Komponenten und werden nicht berücksichtigt (deaktiviert).
In diesem Beispiel kommt jede ermittelte Komponente mindestens einmal in einer Regel vor. Für die Komposition werden die Constraints der Reihe nach angewendet. Die erste Bedingung impliziert die sequenzielle Anordnung der Komponenten k_1 und k_2. Die zweite Regel definiert das parallele Abarbeiten von k_1 und k_3. Da die Komponente k_1 bereits sequenziell mit k_2 verbunden ist, wird das Prozessmodell k_3 parallel zu diesem Konstrukt komponiert. Die letzte Bedingung schiebt k_4 vor das Konstrukt von k_1, k_2 und k_3. Das Ergebnis ist die folgende eindeutige Abarbeitungssequenz:

$$k_{neu} = S(k_4, P(k_3, S(k_1, k_2)))$$

Mit einer steigenden Anzahl von Komponenten und Regeln erhöht sich die Komplexität von Abarbeitungssequenzen. Weiter ist eine Abarbeitungssequenz nicht in jedem Fall eindeutig. Beispielsweise sei ein Konfigurationspunkt $c = (\{k_1, k_2, k_3\}, \{S(k_1, k_2)\})$ definiert. Zur Laufzeit sollen alle drei Komponenten eingebunden werden. In diesem Fall sind drei Kompositionen möglich:

1. Die Sequenz von k_1 und k_2 wird parallel zur Komponente k_3 ausgeführt. Es ergibt sich: $k_{neu} = P(S(k_1, k_2), k_3)$
2. Es ist möglich zuerst Komponente k_1 und k_3 parallel auszuführen und anschließend k_2. Es ergibt sich: $k_{neu} = S(P(k_1, k_3), k_2)$
3. In Analogie zur zweiten Möglichkeit wird k_1 zuerst ausgeführt und danach folgt die parallele Abarbeitung von k_2 und k_3. Es ergibt sich: $k_{neu} = S(k_1, P(k_2, k_3))$

Uneindeutige Abarbeitungssequenzen sind vermeidbar, indem die Regelmenge angepasst wird. Es existiert keine Regel für die Komponente k_3. In der Folge wird k_3 standardmäßig parallel abgearbeitet, aber in welcher Art und Weise bleibt unspezifiziert. Die Erweiterung der Regelmenge um eine zweite Regel für die Komponente k_3 löst das Problem. Diese könnte wie folgt aussehen:

$$r_2 = P(k_1, k_3)$$

Es entsteht eine eindeutige Anordnung der Prozessmodelle k_1, k_2 und k_3, welche die erste obige Abarbeitungssequenz darstellt. Stellvertretend für das Prozessmodell k_1 kann ebenfalls die Komponente k_2 in die Regel r_2 eingesetzt werden. Für den weiteren Verlauf der Arbeit wird definiert, dass Komponenten, die keiner Regel angehören, auf diese Weise eine Regel zugewiesen bekommen. Das Einfügen einer derartigen Regel für die Erzeugung eindeutiger Abarbeitungssequenzen erfolgt am Ende der Regelmenge.

5.1.1 Validierung von Regelmengen

Inkonsistente Regelmengen stellen ein Problem bei der Komposition von Prozessbeschreibungen dar. Inkonsistenzen entstehen u. a. durch Regeln, die sich gegenseitig ausschließen. Das einfachste Beispiel ist die Regelmenge $R_1 = \{S(k_1, k_2), S(k_2, k_1)\}$, welche die zwei

Prozessfragmente k_1 und k_2 in beide Richtungen seriell anordnet. Gleichermaßen ergeben die Constraints der Regelmenge $R_2 = \{S(k_1, k_2), P(k_1, k_2)\}$ einen direkten Widerspruch, da sie für zwei Komponenten die serielle und parallele Komposition definieren. In [SOS05] beschreiben die Autoren drei wesentliche Probleme von Regelmengen:

1. **Wiedersprüchliche Constraints**: Die einfachsten Beispiele wurden bereits vorgestellt.

2. **Transitive Constraints**: Die Regeln $r_1 = S(k_1, k_2)$ und $r_2 = S(k_2, k_3)$ implizieren die Regel $r = S(k_1, k_2, k_3)$. Die Regeln r_1 und r_2 können demnach durch das Constraint r ersetzt werden.

3. **Redundante Constraints**: Redundante Regeln sind zum Beispiel die Constraints $r_1 = S(k_1, k_2, k_3)$ und $r_2 = S(k_2, k_3)$. Die erste Regel schließt Constraint r_2 mit ein. Gleiches gilt für die parallele Komposition.

Wiedersprüchliche Constraints werden auch als *Konflikte* in Regelmengen bezeichnet. Konflikte können zwischen zwei seriellen Ordnungen oder zwischen einem seriellen und parallelen Constraint auftreten. Ein Konflikt zwischen zwei seriellen Ordnungen $S(K_n)$ und $S(K_m)$ mit $K_n, K_m \subseteq K$ besteht, wenn

$$\exists k_i, k_j \in K_n, \text{ sodass } k_i \text{ zu } k_j \text{ führt und}$$

$$\exists k_s, k_t \in K_m, \text{ sodass } k_s \text{ zu } k_t \text{ führt und}$$

$$k_i = k_t \text{ und } k_j = k_s$$

Enthält die Regel $S(K_n)$ ein Paar von Fragmenten, dass in $S(K_m)$ in umgekehrter Reihenfolge vorkommt, ergibt dies einen Widerspruch. Serielle Ordnungen können als Graph $G = (V, E)$ dargestellt werden. Komponenten repräsentieren die Knoten $v \in V$ und jeweils zwei benachbarte Fragmente in einer Regel bilden eine Kante des Graphen. Sie werden als Tupel in die Menge E aufgenommen. Als Beispiel sei folgende Regelmenge ohne Transitivitäten und Redundanzen gegeben:

$$R = \{S(k_1, k_2, k_3), S(k_3, k_1)\}$$

R spezifiziert die sequenzielle Anordnung von k_1 *vor* k_2 *vor* k_3. Die anschließende Anwendung der zweiten Regel ergibt einen Widerspruch. Aus den Constraints der Regelmenge

lässt sich der Graph $G = (\{k_1, k_2, k_3\}, \{(k_1, k_2), (k_2, k_3), (k_3, k_1)\})$ formal definieren. Abbildung 5.2 bildet die Notation des Graphen G ab. Die Abbildung macht deutlich, dass G einen Zyklus enthält. Jeder Zyklus repräsentiert einen Konflikt in einer Regelmenge.

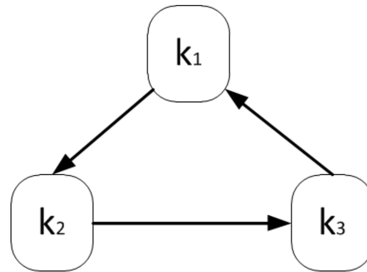


Abbildung 5.2: Gerichteter Graph mit Zyklus

Ein Konflikt zwischen einer seriellen Regel $S(K_n)$ und einer parallelen Ordnung $P(K_m)$ besteht, wenn die folgende Bedingung zutrifft:

$$|K_n \cap K_m| > 1$$

Es dürfen folglich nicht zwei Prozessfragmente in beiden Regeln vertreten sein. Die Beseitigung von Konflikten muss durch den Nutzer des Systems erfolgen, da nur er die durch die Constraints bezweckte Semantik kennt. Hingegen kann die Identifikation von Widersprüchen mit softwaretechnischer Unterstützung erreicht werden. Z. B. ermöglicht die Verwendung der Tiefensuche die Erkennung von Zyklen in gerichteten Graphen. Die Tiefensuche schlägt fehl, wenn ein Knoten erreicht wird, der vorher schon einmal passiert wurde. Die Durchführung eines derartigen Tests könnte für jedes Constraint erfolgen, welches neu in eine Regelmenge aufgenommen wird. Auf diese Weise lassen sich Konflikte zwischen seriellen Ordnungen bei dem Aufbau der Regelmenge erkennen. In [Tur09] beschreibt V. Turau einen Algorithmus für die Tiefensuche.
Potenzielle Redundanzen zwischen zwei seriellen Constraints $S(K_n)$ und $S(K_m)$ können auf ähnliche Weise erkannt werden, wie Konflikte zwischen seriellen und partiellen Ordnungen.

Weitere Informationen über die Beseitigung von Inkonsistenzen in Regelmengen werden in [SOS05] beschrieben. Im Folgenden wird angenommen, dass Regelmengen frei von Redundanzen, Transitivitäten und Konflikten sind.

5.1.2 Konfluenz von Regelmengen

Offen bleibt die Frage, welche Bedeutung die Anwendungsreihenfolge der Regeln hat. In [HSS08] definiert A. Heuer den Begriff der „Konfluenz“ für *ECA*-Regelmengen, welche das Trigger-Konzept in Datenbanken um zusätzliche Aspekte erweitern:

> „Ein Regelsystem heißt konfluent, wenn der Effekt auf die Datenbank bei gleichzeitig aktivierten Regeln immer unabhängig von der Reihenfolge der Abarbeitung dieser Regeln ist.“

Die Definition ist ebenfalls auf Regelmengen für die Komposition von Prozessmodellen anwendbar. Aktiviert sind immer alle Regeln, die mindestens zwei Prozessfragmente betreffen, die zur Laufzeit in den Konfigurationspunkt $c \in C$ eingebunden werden sollen. Für zwei Komponenten k_1 und k_2 bedeutet dies, dass ebenfalls Regeln wie $r = S(k_1, k_3, k_2)$ aktiviert sein können. Da das Fragment k_3 nicht in $c \in C$ eingebunden werden soll, dürfen die irrelevanten Komponenten einer Regel nicht berücksichtigt werden. Alle übrigen Regeln entfallen für die Betrachtung der Konfluenz.

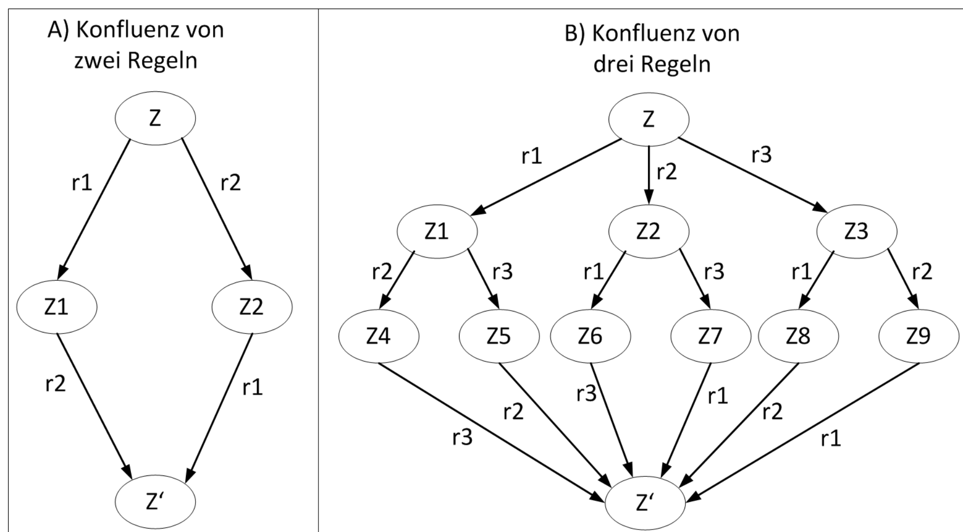


Abbildung 5.3: Konfluenz von Regelmengen, vgl. [Hom]

Die Konfluenz beschreibt die Kommutativität von Regelmengen. Zwei Regeln r_1 und r_2 sind kommutativ, wenn bei der Auführung von r_1 vor r_2 dasselbe Endergebnis erreicht wird, wie bei der umgekehrten Ausführungsreihenfolge. Abbildung 5.3A zeigt einen Zustand Z. Die Anwendung beider Regeln führt zu zwei unterschiedlichen Zuständen $Z1$ und $Z2$. Die Anwendung der Regel r_2 bewirkt einen Zustandswechsel von $Z1$ nach Z'.

Gleiches gilt für die Ausführung der Regel r_1 im Zustand $Z2$. Abbildung 5.3 bildet das gleiche Prinzip für eine Regelmenge mit drei aktiven Constraints ab.

Regelmengen, die nur Constraints der parallelen und seriellen Ordnung beinhalten und keine Redundanzen, Transitivitäten und Konflikte besitzen, sind konfluent. Unterschiedliche Abarbeitungssequenzen bewirken unterschiedliche Kompositionen der einzubindenden Prozessfragmente. Die Abhängigkeiten werden dennoch eingehalten und somit führt jede Komposition zum selben Endergebnis der Ausführung.

5.1.3 Komposition von Aktivitäten

Bei der Komposition von Prozessmodellen ist es grundsätzlich erwünscht, möglichst viele Komponenten parallel abzuarbeiten. Besteht eine Abhängigkeit zwischen zwei Prozessmodellen, müssen diese jedoch sequenziell angeordnet werden, selbst wenn die Abhängigkeit nur eine einzelne Aktivität in einer der beiden Komponenten betrifft. Dieser Unterabschnitt stellt eine Methode vor, um für derartige Fälle die beteiligten Komponenten dennoch parallel anzuordnen.
In [RTM06] beschreiben die Autoren das *Milestone*-Control-Flow-Pattern. Es beschreibt die Möglichkeit der Synchronisation von zwei parallel angeordneten Prozessteilen. Genau bedeutet dies, dass eine Aktivität nur aktiviert werden kann, wenn sich die Prozessinstanz in einem bestimmten Zustand, dem Milestone, befindet. Hiermit ist der Zustand des parallel angeordneten Prozessteils gemeint. Hat die Prozessinstanz bereits einen Zustand jenseits des Milestone erreicht, kann diese Aktivität nicht mehr aktiviert werden. Abbildung 5.4 bildet das Milestone-Pattern als Petrinetz-Modell ab.

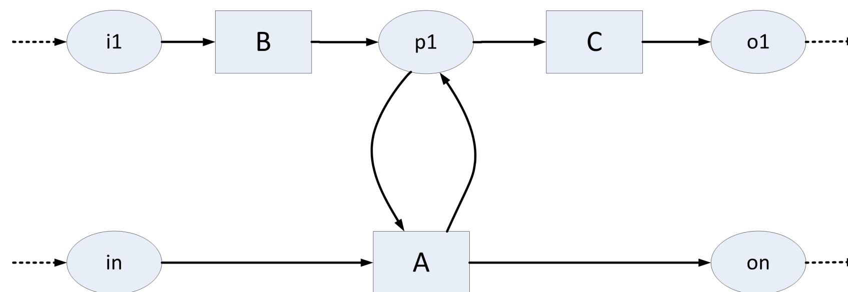


Abbildung 5.4: Das Milestone-Pattern, vgl. [RTM06]

Die Aktivierung von A ist nicht möglich, solange sich der obige Abarbeitungszweig nicht im Zustand p1 befindet (Es ist ein Token auf der Stelle p1).

Nachfolgend wird eine Komponente $k \in K$ für einen Konfigurationspunkt $c \in C$ als Menge von Aktivitäten betrachtet, welche folgendermaßen definiert ist:

$$k_1 = \{a_{11}, a_{12}, ..., a_{1n}\}, \quad n \in \mathbb{N}$$

Jede Aktivität besitzt einen Index bestehend aus zwei Zahlen. Die erste Zahl repräsentiert den Index der zugehörigen Komponente und dient zur Unterscheidung von Aktivitäten verschiedener Prozessmodelle. Alle Tasks einer Komponente sind zusätzlich mit der zweiten Ziffer durchnummeriert, um einzelne Aktivitäten desselben Prozessfragments zu referenzieren.
Das Milestone-Pattern erfordert ein neue Regel, die im Folgenden die Bezeichnung „*Mile-Constraint*" trägt. Diese Regel erzwingt die parallele Anordnung von *zwei* Komponenten und modelliert zusätzlich das „Milestone-Konstrukt" zwischen die abhängigen Aktivitäten. Die formale Notation der mile-Regel für zwei Prozessfragmente $k_1 = \{a_{11}, ..., a_{1n}\}$ und $k_2 = \{a_{21}, ..., a_{2m}\}$ mit $n, m \in \mathbb{N}$ wird wie folgt definiert:

$$\text{Mile}(a_{1i}, a_{2j})$$

$$1 \leq i \leq n, 1 \leq j \leq m$$

Obiges Mile-Constraint bedeutet, dass die Aktivität a_{1i} der Komponente k_1 zwingend vor der Aktivität a_{2j} des Prozessmodells k_2 ausgeführt werden muss.
Der nachfolgende Abschnitt untersucht, inwiefern die eingeführten Constraints mit der Prozessmodellierungssprache YAWL umsetzbar sind.

5.2 Umsetzbarkeit der Constraints mit YAWL

Nachdem die möglichen Regeln für die Komposition von Prozessfragmenten eingeführt wurden, wird in diesem Unterabschnitt die Frage geklärt, ob die Modellierungssprache YAWL die entsprechenden Konstrukte zur Verfügung stellt, um die Constraints zu modellieren.
Die sequenzielle und parallele Anordnung von Prozesselementen sind grundlegende Control-Flow Patterns, die mit YAWL realisierbar sind. Abbildung 5.5A illustriert die serielle Ordnung für die Beispielregeln $r_1 = S(k_1, k_2, ..., k_n)$ und $r_2 = S(k_2, k_1)$.
Die sequenzielle Komposition benötigt keine zusätzlichen Konstrukte. Die letzte Task der

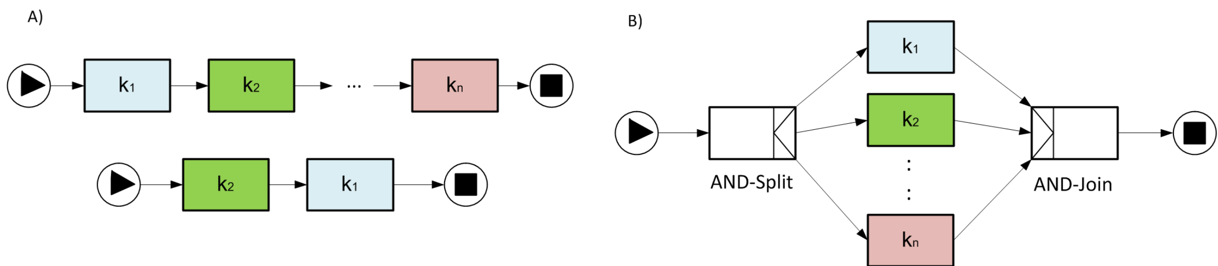


Abbildung 5.5: Sequenzielle und parallele Modellierung mit YAWL

ersten Komponente „fließt“ in die erste Aktivität des letzten Fragments.
Für die Umsetzung der parallelen Anordnung stellt YAWL den „AND-Split“ (siehe Abschnitt 4.3.1) zur Verfügung, welcher den Kontrollfluss einer Prozessmodells in n unabhängige Zweige teilt. Jeder Zweig stellt eine Komponente dar. Die Modellierung der parallelen Ordnung wird in Abbildung 5.5B für n Fragmente dargestellt.
Für die Synchronisation von n parallelen Zweigen bietet die Sprache YAWL das Prinzip von „AND-Join“ an. Obwohl „Choice and Merge“-Konstrukte für die Umsetzung des Publikationsprozesses nicht berücksichtigt wurden, sei an dieser Stelle erwähnt, das YAWL ebenfalls die Konstrukte des „OR-Splits“ und „OR-Joins“ bereitstellt.

Die Modellierung von Mile-Constraints erfordert im Gegensatz zur seriellen und parallelen Ordnung mehr Aufwand. Für die Umsetzung eines Mile-Konstrukts zwischen zwei Prozessfragmenten werden zusätzliche „AND-Splits“ und „AND-Joins“ benötigt. Abbildung 5.6A bildet die konkrete Modellierung des Milestone-Patterns für zwei Komponenten $k_1 = \{a_{11}, ..., a_{1n}\}$ und $k_2 = \{a_{21}, ..., a_{2m}\}$ mit $n, m \in \mathbb{N}$ in der Prozessmodellierungssprache YAWL ab.
Es wird ein Prozessverlauf dargestellt, indem die Aktivität a_{2j} erst ausgeführt werden kann, nachdem die Aktivitäten a_{11} bis a_{1i} sowie die Tasks a_{21} bis a_{2i-1} abgearbeitet wurden. Diese Art der Modellierung besitzt eine Einschränkung. Der obere Zweig des Modells kann erst mit der Bearbeitung von Aktivität a_{1i+1} fortfahren, wenn die Ausführung der Task a_{2j} abgeschlossen wurde. Eine derartige Einschränkung lässt sich vermeiden. Abbildung 5.6 zeigt eine alternative Modellierung des Constraints $r = \text{Mile}(a_{2j}, a_{1i+1})$.
Diese Umsetzung weicht von der ursprünglichen Definition des Milestone-Pattern ab. Sie entspricht dennoch den Anforderungen an den Publikationsprozess. Wie in Abbildung 5.6A ist die Ausführung von der Aktivität a_{2j} erst möglich, nachdem die Aktivitäten a_{1i-1} und a_{2j-1} abgearbeitet wurden. Im Unterschied zum Milestone Pattern kann der

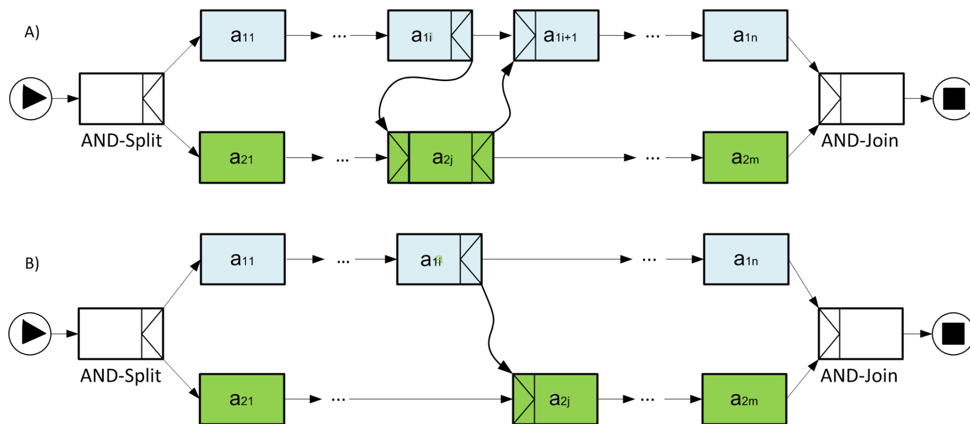


Abbildung 5.6: Modellierung des Milestone-Patterns mit YAWL

obere Prozesszweig seine Ausführung ohne Verzögerung fortsetzen und die Aktivitäten a_{2j} und a_{1i+1} werden parallel abgearbeitet.

Kapitel 6

Dynamische Komposition von Prozessbeschreibungen mit YAWL

6.1 YAWL – Das Workflow Management System

Das YAWL-System ist wie eine Service-orientierte Architektur aufgebaut. Es besteht aus einer erweiterbaren Menge von YAWL Services, die ein oder mehrere Interfaces offerieren. Einige Services stellen ihre Interfaces Endnutzern zur Verfügung, andere wiederrum Applikationen oder weiteren Diensten.
Der Kerndienst eines YAWL-Systems ist die *Workflow-Engine*. Alle Prozess-Spezifikationen werden zur Ausführung in die Engine geladen und in einem Repository gespeichert, von wo aus sie instanziiert werden können. Für die Verwaltung laufender Prozessinstanzen (Cases) erzeugt und verteilt die Workflow Engine Work Items entsprechend der Workflow-Spezifikation.
Jeder andere Service muss erst eine Sitzung mit der Workflow Engine aufbauen, um mit ihr zu kommunizieren. Im Gegenzug auf eine derartige Verbindungsanfrage verteilt die Engine ein sogenanntes *Session Handle Token*, welches von dort an bei allen nachfolgenden Anfragen eines Services an die Workflow Engine ausgetauscht wird. Abbildung 6.1 zeigt die wesentlichen Dienste eines YAWL Systems und ihre dazugehörigen Interfaces.

Services können zwei verschiedene Wege nutzen, um mit der YAWL Engine zu kommunizieren. In [DF] beschreibt M. Dumas folgende Kommunikationsmöglichkeiten:

1. Ein Service kann mit der Workflow Engine **direkt** durch den Austausch von XML-

Nachrichten über HTTP kommunizieren.

2. Ein Service kann mit der Workflow Engine **indirekt** über den *Web Service Invoker (WSI)* kommunizieren. Der WSI macht die Ausführung von Operationen eines externen SOAP-Web Services[1] möglich. Hierzu wird eine Task zur Modellierungszeit mit einer derartigen Operation verbunden. Zur Laufzeit verteilt der WSI das entsprechende Work Item an den Web Service, der die Operation anschließend ausführt.

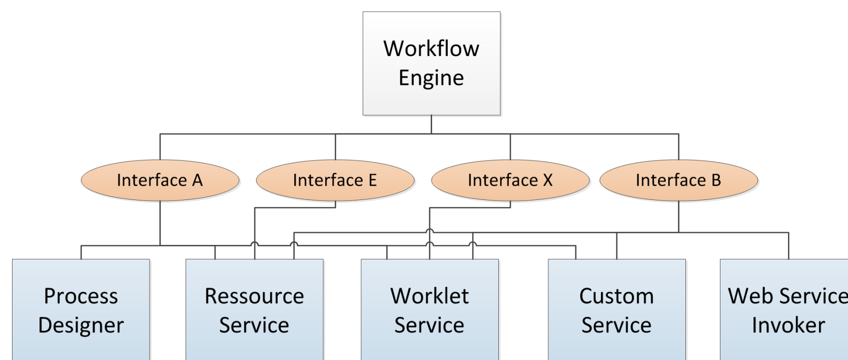


Abbildung 6.1: Services eines YAWL Systems, vgl. [TVAR10]

Die Workflow Engine interagiert mit anderen Diensten im YAWL System über vier verschiedene Interfaces. Drei von ihnen entsprechen den Interfaces des Workflow Reference Model aus Abschnitt 3.2. In [TVAR10] beschreiben die Autoren die Interfaces wie folgt:

- **Interface A** wird genutzt, um Prozessspezifikationen in die Workflow Engine zu laden bzw. diese wieder zu entfernen. Weiterhin ermöglicht das Interface das Registrieren und Löschen von Referenzen zu externen Diensten, sowie den Auf- und Abbau von Nutzerverbindungen.

- **Interface B** dient Services zum Aufbau einer Sitzung mit der Workflow Engine, für den Start von Prozessinstanzen, zum Ein- und Auschecken von Work Items und zum Beziehen von Prozessdaten und Zustandsinformationen.

- **Interface E** ermöglicht das Abrufen und Auswerten von Prozessprotokollierungen (Logs).

[1]SOAP ist ein Protokoll zum Austausch XML-basierter Nachrichten.

- Über **Interface X** können Services Ausnahmeerscheinungen (Exceptions) zur Laufzeit erkennen und behandeln. Dieses Interface kann keinem Interface des Workflow Reference Model zugeordnet werden.

Der YAWL Editor (*Process Designer*) ist eine Entwurfsumgebung für die Entwicklung und Verifikation von YAWL Prozessspezifikationen. Der Process Designer ist als Java-Desktopanwendung realisiert und kommuniziert mit der Workflow Engine über das Interface A, um eine Liste der momentan registrierten Services zu erhalten. Diese Liste wird benötigt, um sogenannte **Task-related Services** an eine bestimmte Aktivität zu binden. Zur Laufzeit übernimmt ein derartiger Dienst die Ausführung der assoziierten Taskinstanz. Der Funktionsumfang des YAWL Editors bietet sämtliche Möglichkeiten, um einen Workflow grafisch mit der Prozessbeschreibungssprache YAWL zu modellieren. Der Editor ermöglicht ebenfalls die Zuweisung von Ressourcen zu einzelnen Aktivitäten und die Definition von Daten in Form von Netz- und Taskvariablen. Hierzu werden eine Vielzahl von primitiven Datentypen zur Verfügung gestellt.

Sofern einer Aktivität kein Task-related Service zugeordnet wurde, verwaltet der *Ressource Service* die Verteilung von Work Items an Workflow-Teilnehmer. Es wird unterschieden zwischen „manuellen“ und „automatisierten“ Work Items. Erstes benötigt personelle Ressourcen und wird an den *Ressource Manager* übergeben. Der Unterdienst bestimmt eine Menge von Teilnehmern, denen das Work Item für die Bearbeitung angeboten werden darf. Ein weiterer Unterdienst mit dem Namen *Worklist Handler* ist schließlich für das Anbieten und Verteilen von manuellen Work Items an die Endnutzer verantwortlich. Über eine Web-Oberfläche können menschliche Workflow-Teilnehmer Work Items annehmen, starten, abarbeiten (Variablen ändern) und beenden. Der Vorgang des Startens eines Work Items wird „Auschecken“, das Beenden „Einchecken“ genannt. Der Ressource Service besitzt zusätzlich den *Codelet Manager*, der die Bearbeitung von automatisierten Work Items übernimmt. Er ist in der Lage sogenannte Codelets auszuführen. [TVAR10]

6.2 Custom YAWL Services

Custom YAWL Services erweitern das YAWL System um zusätzliche Funktionen. Die Anbindung externer Applikationen an die YAWL Engine ermöglicht die Kommunikation laufender Prozessinstanzen mit ihrer Umwelt. Auf diese Weise können Custom Services

Arbeitseinheiten (Work Items) entgegennehmen, über ein bestimmtes Ereignis informiert werden oder den Status von Work Items verändern. [Ter10]
Custom YAWL Services kommunizieren mit der YAWL Engine durch das Austauschen von XML-Nachrichten über HTTP. Die YAWL-Distribution stellt mehrere APIs zu Verfügung, um die Kommunikation zwischen Custom Service und Engine aufzubauen. Die gebräuchliste API ist das Interface B. Im Wesentlichen sind für die Abarbeitung eines Work Items durch einen Custom Service vier Interaktionen relevant [TCA10]:

1. Die YAWL Engine informiert den Custom Service über die Aktivierung einer mit dem Service verbundenen Task. Anschließend erzeugt die Engine ein Work Item und setzt dessen Status auf „Aktiviert“.

2. Der Custom Service informiert die YAWL Engine, dass er bereit ist, das Work Item zu bearbeiten. Dies ist der Vorgang des Auscheckens. Die YAWL Engine setzt daraufhin den Status des Work Items auf „Wird bearbeitet“.

3. Die YAWL Engine kann den Custom Service über den Abbruch der Bearbeitung eines Work Items informieren.

4. Der Custom Service informiert die YAWL Engine über die erfolgreiche Bearbeitung des Work Item. Der Status wird auf „Abgearbeitet“ gesetzt.

Für die Implementierung eines Custom YAWL Services wird eine neue Klasse benötigt, die vom *InterfaceBWebSideController* erbt. Der InterfaceBWebSideController besitzt zwei abstrakte Methoden, die jeder Custom Service in der Hauptklasse implementieren muss [TCA10]:

- `public void handleEnabledWorkItemEvent(WorkItemRecord wr)` – Die Methode wird aufgerufen, wenn die YAWL Engine ein Work Item erzeugt und seinen Status auf „aktiviert“ setzt. Sie enthält den Code für die Abarbeitung des Work Items durch den Custom Service, der zur Modellierungszeit mit der entsprechenden Task assoziiert wurde.

- `public void handleCancelledWorkItemEvent(WorkItemRecord wr)` – Wird die Bearbeitung eines Work Items durch die YAWL Engine abgebrochen, wird diese Methode im Custom YAWL Service aufgerufen. Nach der Ausführung der Methode unternimmt der Custom Service keinen Versuch mehr, das Work Item ein- bzw. auszuchecken.

6.3 Umsetzung des Konzepts

Das WfMS YAWL besitzt keinen Dienst, der das in Kapitel 5 vorgestellte Konzept umsetzen kann. Entsprechend wird das Konzept durch einen Custom YAWL Service mit dem Namen *Composition Service (CS)* realisiert. Dieser kommuniziert auf ähnliche Weise mit der YAWL Engine, wie der Worklet Service. Durch das Interface B wird der Composition Service über die Aktivierung einer mit ihm assoziierten Task informiert und checkt das entsprechende Work Item aus. Das Aufrufen des CS bedeutet, dass sich der Basisprozess an einem Konfigurationspunkt befindet. Konfigurationspunkte werden folglich durch die Zuweisung des Composition Service als selbige ausgezeichnet. Abbildung 6.2 illustriert die Kommunikation zwischen der YAWL Engine und dem Composition Service.

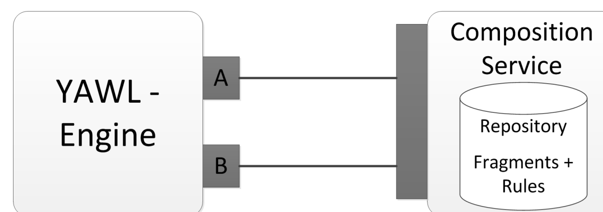


Abbildung 6.2: Kommunikation zwischen YAWL Engine und Composition Service

Alle Fragmente die in das Basisprozessmodell eingesetzt werden können, befinden sich in einem *Repository*. Jeder Konfigurationspunkt hat einen separaten Ordner, indem sich alle potenziellen Prozessfragmente befinden, die in diesem Punkt eingebunden werden können. Weiter enthält jeder Ordner zusätzlich eine Textdatei. Sie dient als Ablage für sämtliche Regeln, die bei der Komposition der Prozessfragmente eingehalten werden müssen. Diese Ordner- und Dateistruktur entspricht der praktischen Umsetzung der theoretischen Definition eines Konfigurationspunktes $c = (K, R)$.
Konfigurationspunkte besitzen einen Identifier (ID), welcher mit dem Work Item an den Composition Service übermittelt wird. Diese Variable beinhaltet die Information, welchen Fragmentordner der CS für die aktuelle Komposition nutzen muss.

Der Composition Serive führt nach dem Auschecken eines Work Items folgende Teilschritte aus:

1. Die Bestimmung der Komponenten, die in den Konfigurationspunkt eingebunden werden müssen.

2. Die Komposition der ermittelten Komponenten zu einem neuen Prozessmodell.

3. Das Uploaden des komponierten Prozessmodells in die YAWL Engine über das Interface A.

4. Das Starten des komponierten Prozessmodell als separaten Case nebenläufig zum Basisprozess

5. Das Variablen-Mapping zwischen Basisprozess und komponiertem Case.

6. Das Variablen-Mapping zwischen komponiertem Case und Basisprozess nach Abarbeitung der Prozessinstanz.

Im Folgenden werden die ersten beiden Teilschritte erläutert. Der Fokus liegt hierbei vor allem auf der Komposition von beliebigen YAWL-Netzen. Die Teilschritte 3 bis 6 werden analog zur Funktionsweise des Worklet Services durchgeführt und nicht genauer betrachtet.

6.3.1 Bestimmung der Komponenten

Für die Veröffentlichung von Multimediadokumenten stehen dem Autor mehrere Dokumentenmodelle zur Verfügung. Sie spezifizieren die Struktur und den erlaubten Content für verschiedene Dokumenttypen, wie beispielsweise Vorlesungsdokumente oder Dissertationen.
Abbildung 6.3 zeigt ein Beispiel für ein Dokumentenmodell von Vorlesungsfolien einer Universitätsveranstaltung. Das Modell ist als *Dokumenttypdefinition (DTD)*[2] spezifiziert und deklariert die Struktur eines Vorlesungsdokuments ohne Anspruch auf Vollständigkeit. Aus Gründen der Übersichtlichkeit wurde in Abbildung 6.3 auf die Deklaration von *Parsed Character Data (PCDATA)* und *Character Data (CDATA)* verzichtet.
Die DTD beschreibt eine Vorlesung als Menge von Vorlesungsfolien, welche aus Komponenten, Metadaten und weiteren Vorlesungsfolien bestehen kann. Eine Komponente ist entweder ein Bild, ein Text oder ein Video.
Jede Komponente sowie das Dokument selbst besitzen Metadaten, deren Angabe durch die Verwendung von Standards bestimmt ist. Der *MPEG-7-Standard* ermöglicht es die

[2] http://www.w3schools.com/dtd/default.asp

Beschreibung audiovisueller Inhalte aus verschiedenen Bereichen auf gleiche Art und Weise. [Sch05]. *Dublin Core (DC)*[3] ist eine Konvention, die Metadaten für eine Vielzahl von unterschiedlichen Objekten definiert. Zudem können physikalische Metadaten zu einem Dokument angegeben werden, welche keinen Standard besitzen müssen. Die Komponenten Bild und Video bestehen wiederum aus einer Reihe von Bestandteilen.

```
1  <?xml version="1.0"?>
2  <!DOCTYPE Course [
3
4  <!ELEMENT Course (Lecture+)>
5  <!ELEMENT Lecture (Lecture*, Component+, Metadata+)>
6  <!ELEMENT Component (Image | Text | Video)>
7  <!ELEMENT Metadata (DC | physicalMetadata | MPEG7)>
8
9  #Text
10 <!ELEMENT Text (Data, Metadata+)>
11
12 #Image
13 <!ELEMENT Image (Region+, Metadata+)>
14 <!ELEMENT Region (Region*, FeatureVector*)>
15 <!ELEMENT FeatureVector (Feature+)>
16 <!ELEMENT Feature (AverageColor | Histogram | Texture )>
17
18 #Video
19 <!ELEMENT Video (VideoStream+, Metadata+)>
20 <!ELEMENT VideoStream (FrameSequence+)>
21 <!ELEMENT FrameSequence (StructuralComponent+, Metadata+)>
22 <!ELEMENT StructuralComponent (Scene | Topic | Slide | KeyFrame)>
23
24 ...
25
26 ]>
```

Abbildung 6.3: Dokumentenmodell einer Vorlesung

Wenn der Autor im Rahmen des Publikationsprozesses ein Dokument erstellt, erschafft er eine konkrete Instanz des verwendeten Dokumentenmodells. Diese ergibt sich aus der beliebigen Komposition von Content (in der DTD als Komponenten deklariert). Das Parsen der aktuellen Dokumentinstanz ermöglicht dem Composition Service die Bestimmung der Komponenten, die in einen Konfigurationspunkt eingebunden werden müssen. Beinhaltet die Dokumentinstanz die XML-Elemente „Text“ und „Video“, werden nur die entsprechenden Kompositionsregeln aktiviert und folglich nur die dazugehörigen Komponenten miteinander verbunden.

Die nachfolgenden Abschnitte beschreiben ausgehend von der Implementation des Composition Serivce die Komposition von YAWL-Netzen zu einem neuen Prozessmodell.

[3] http://dublincore.org

6.3.2 Implementation des Composition Service

Nachdem die für die Komposition benötigen Prozessfragmente bestimmt wurden, werden die Komponenten unter Berücksichtigung der Kompositionsregeln zusammengesetzt. Abbildung 6.4 zeigt vorab die Implementation des Composition Services als UML-Klassendiagramm. Dieses dient als Vorlage für die Erläuterung der Komposition von YAWL-Spezifikationen.

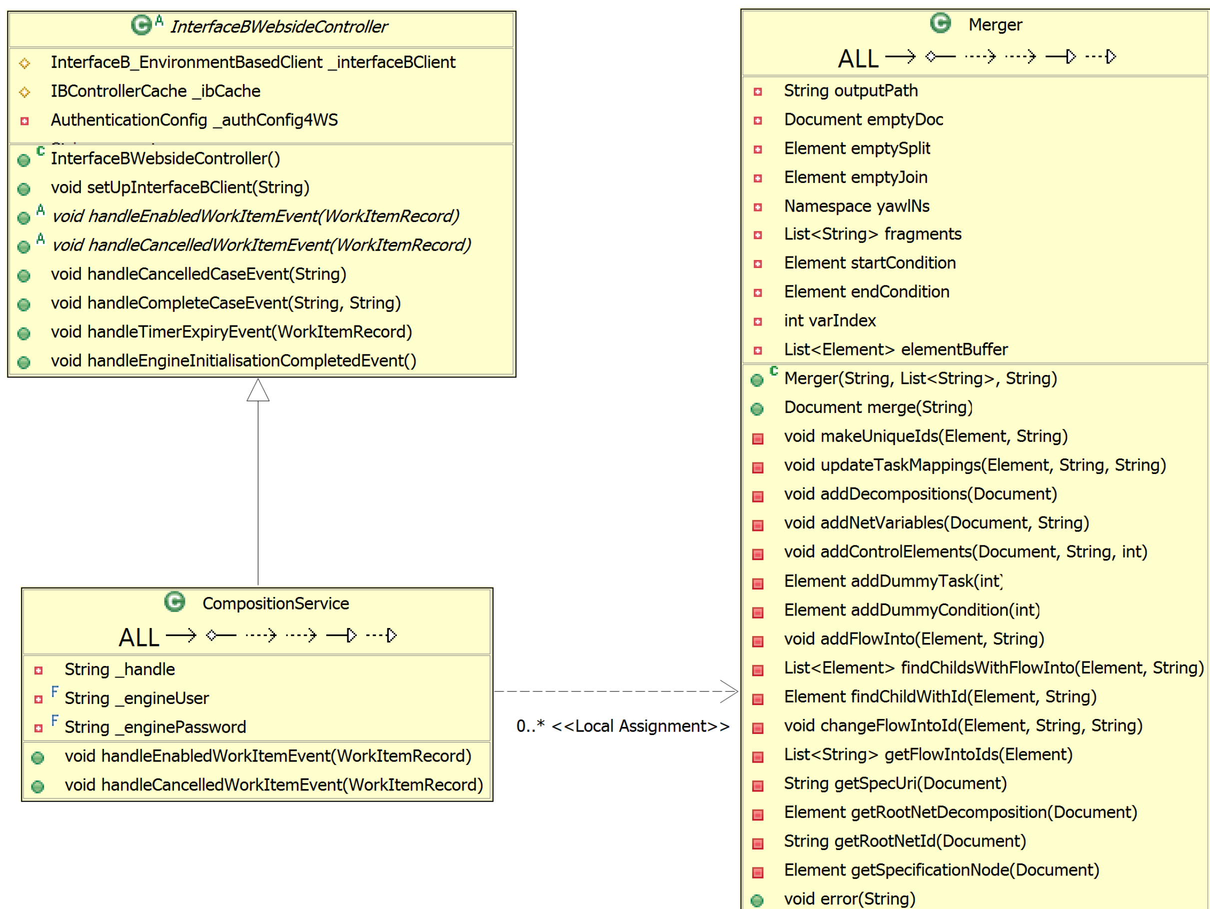


Abbildung 6.4: Klassendiagramm: Composition Service

Das UML-Diagramm besteht aus zwei grundlegenden Klassen. Die Hauptklasse mit dem Namen „CompositionService“ leitet, wie in Abschnitt 6.2 beschrieben, von der Klasse „InterfaceBWebsideController“ ab und implementiert die abstrakten Methoden. Nachdem die einzubindenden Komponenten im ersten Teilschritt bestimmt wurden, arbeitet die „handleEnabledWorkItemEvent“-Methode alle aktivierten Constraints aus der Regeldatei der Reihe nach ab. Hierzu werden das Kompositionsverfahren (parallel oder seriell), sowie

die in dem Constraint enthaltenen Prozessfragmente aus der Regel geparst. Eine Regel sieht beispielsweise wie folgt aus:

$$P(Bild, Video)$$

Das „P“ gibt das Verfahren an. In diesem Fall sollen die Prozessfragmente für ein Bild und ein Video parallel angeordnet werden. Die Komposition übernimmt die Klasse „Merger“. Die jeweilige Instanz der „CompositionSerivce“-Klasse erzeugt für jede Regel ein neues „Merger“-Objekt, welches die geparsten Komponenten unter Berücksichtigung des Kompositionsverfahrens kombiniert.

6.3.3 Die Klasse „Merger“

Die Aufgabe der Klasse „Merger“ ist die Komposition von einer übergebenen Anzahl von Prozessfragementen (aus der Regel). Komponenten bzw. Prozess-Spezifikationen verwaltet YAWL innerhalb von XML-Dateien. Abbildung 6.5 zeigt vereinfacht an zwei Beispielen, in welcher Form eine YAWL-Spezifikation als XML-Datei abgespeichert wird.

```
<specification uri="Spec1">
  ....
  <decomposition id="Netz1">
    <inputParam>
      <index>0</index>
      <name>Metadata</name>
      <type>string</type>
    </inputParam>
    <processControlElements>
      <inputCondition id="InputCondition_1">
        <flowsInto>
          <nextElementRef id="Bild_3" />
        </flowsInto>
      </inputCondition>
      <task id="Bild_3">
        <name>Metadaten Bild</name>
        <flowsInto>
          <nextElementRef id="OutputCondition_2" />
        </flowsInto>
        ...
      </task>
      <outputCondition id="OutputCondition_2" />
    </processControlElements>
  </decomposition>
  ...
</specification>
```

```
<specification uri="Spec2">
  ....
  <decomposition id="Netz2">
    <inputParam>
      <index>0</index>
      <name>Metadata</name>
      <type>string</type>
    </inputParam>
    <processControlElements>
      <inputCondition id="InputCondition_1">
        <flowsInto>
          <nextElementRef id="Video_3" />
        </flowsInto>
      </inputCondition>
      <task id="Video_3">
        <name>Metadaten Video</name>
        <flowsInto>
          <nextElementRef id="OutputCondition_2" />
        </flowsInto>
        ...
      </task>
      <outputCondition id="OutputCondition_2" />
    </processControlElements>
  </decomposition>
  ...
</specification>
```

Abbildung 6.5: YAWL-Spezifikationen in XML-Syntax

Das äußerste XML-Element ist das „specification"-Tag, welches alle weiteren Bestandteile einer YAWL-Spezifikation einschließt. Es besitzt ein Attribut mit dem Namen „uri". Der entsprechende Attributwert identifiziert jede Spezifikation eindeutig. Ein „specification"-Tag enthält mehrere „decomposition"-Elemente. Es existieren *Dekompositionen (Zerlegungen)* für YAWL-Netze und YAWL-Tasks. Das „decompostion"-Element eines YAWL-Netzes enthält dessen Netzvariablen und die sogenannten *ProcessControlElements*. Letzteres umfasst die Definition des Tasks, Conditions und ihrer Kontrollflüsse.
In Abbildung 6.5 besitzt die linke Prozessbeschreibung eine Input-Netzvariable vom Typ „String", definiert im „inputParam"-Tag. Weiter beinhaltet das „processControlElements"-Tag eine Input- und Output-Condition, sowie eine Task mit dem Namen „Metadaten Bild". Kontrollflüsse werden innerhalb der Condition- und Task-Elemente durch das „flowsInto"-Tag angegeben. ProcessControlElements besitzen einen eindeutigen Identifier, um den Kontrollfluss ohne Konflikte zu steuern.
Die Dekomposition einer Task beinhaltet die lokalen Variablen, welche für diese Aktivität spezifiziert wurden.

Für die Komposition der beiden YAWL-Spezifikationen aus Abbildung 6.5 werden die entscheidenden Elemente beider XML-Datei in ein XML-Grundgerüst übernommen und je nach Kompositionsverfahren (parallel oder seriell) entsprechend angepasst. Ein solches Grundgerüst enthält bereits den Aufbau einer Prozess-Spezifikation. Es beinhaltet bereits die Input- und Output-Condition ohne die Definition Definition der „flowsInto"-Elemente. Abbildung 6.6 zeigt die „Vereinigung" der XML-Datei aus Abbildung 6.5 als serielle Komposition.

Der Konstruktor der Klasse „Merger" erfordert die Angabe von drei Übergabeparametern. Sämtliche Parameter spezifizieren relative Pfade. Hierzu gehören der Pfad zur XML-Grundgerüstdatei, Pfadinformationen über die zu komponierenden Prozessfragmente und ein Zielort für die Speicherung des neu entstandenen Prozessmodells.
Die wichtigste Funktion der Klasse ist die „merge"-Methode, welche einen *Modus* erfordert. Über den Modus wird angegeben, ob die Komponenten parallel oder seriell angeordnet werden sollen. Für die Unterscheidung dienen die Werte „Parallel" und „Sequential". Die „merge"-Methode iteriert über alle übergebenen Prozessfragmente. *Jeder* Iterationschritt (die Abarbeitung *jeder* Komponente) umfasst fünf Schritte, welche im Folgenden erläutert werden.

```
<specification uri="Komposition">
  ….
  <decomposition id="KompNetz">
    <inputParam>
      <index>0</index>
      <name>Metadata_Spec1</name>
      <type>string</type>
    </inputParam>
    <inputParam>
      <index>1</index>
      <name>Metadata_Spec2</name>
      <type>string</type>
    </inputParam>
    <processControlElements>
      <inputCondition id="InputCondition_1">
        <flowsInto>
          <nextElementRef id="Bild_3_Spec1" />
        </flowsInto>
      </inputCondition>
      <task id="Bild_3_Spec1">
        <name>Metadaten Bild</name>
        <flowsInto>
          <nextElementRef id="Video_3_Spec2" />
        </flowsInto>
      </task>
      <task id="Video_3_Spec2">
        <name>Metadaten Video</name>
        <flowsInto>
          <nextElementRef id="OutputCondition_2" />
        </flowsInto>
      </task>
      …
```

Abbildung 6.6: Komponierte YAWL-Spezifikation in XML-Syntax

Angleichung der Identifikatoren

Der erste Schritt für eine erfolgreiche Komposition von Prozessfragmenten ist das Angleichen der Identifikatoren. Abbildung 6.5 macht deutlich, dass nicht nur die ProcessControllelements der jeweiligen Netzdekomposition einen Identifier besitzen. Zusätzlich besitzen die Netzdekomposition selbst, sowie alle Taskdekompositionen eine ID. Die Methode „makeUniqueIds“ ergänzt die Werte aller „id“-Attribute, welche im aktuellen Fragment vorkommen, um einen Unterstrich „_“ und die „Specification-URI“. Zum Beispiel ändert sich die ID einer Task wie folgt ab:

$$\text{TaskID} \rightarrow \text{TaskID_SpecificationID}$$

Dieser Schritt ist aus mehreren Gründen erforderlich. In unterschiedlichen Dokumenten können ProcessControlElements den gleichen Identifier besitzen. Ohne die Angleichung

der IDs entstehen nach der „Mischung“ der XML-Elemente zweideutige Referenzen, die in „flowsInto“-Elementen Konflikte verursachen. Zum anderen dürfen zwei Netz- bzw. Taskdekompositionen nicht den gleichen Namen besitzen. Die „makeUniqueIds“-Methode beseitigt derartige Probleme für den weiteren Kompositionsverlauf.

Angleichen und Kopieren der Netzvariablen

Identische Netzvariablen dürfen nicht mehrfach in einer YAWL-Spezifikation auftreten. Um dies beim „Mischen“ der Fragmente zu gewährleisten, werden sämtliche Netzvariablen nach dem selben Prinzip, wie die Identifikatoren, umbenannt. Sind alle Netzvariablen eindeutig, können die entsprechenden XML-Elemente bereits in das das neue Prozessmodell kopiert werden. Es ist zu beachten, dass die Reihenfolge beim Kopieren wichtig ist. Netzvariablen können entweder Eingang-, Ausgangs- oder lokale Variablen sein. Die Reihenfolge der entsprechenden Elemente wird durch die YAWL-Spezifikation vorgegeben. Sie spezifiziert die Anordnung von „Input- vor Output- vor lokalen Variablen“.
Das erste Prozessfragment ist ein Spezialfall, da sich noch keine Netzvariablen im Grundgerüst befinden. Die Reihenfolge muss folglich nicht berücksichtigt werden. Alle XML-Elemente folgender Fragmente, welche Netzvariablen definieren, müssen an der richtigen Stelle eingefügt werden.
Die Methode „addNetVariables“ fügt alle „inputParam“-Elemente als erstes Kindelement der Netzdekomposition ein. Lokale Variablen werden als vorletztes Element hinzugefügt, da das letzte Element der Netzdekomposition das „ProcessControlElements“-Tag ist. Das Einsetzen von Ausgangsvariablen erfolgt hinter dem letzen „inputParam“-Element.

Angleichen der Variablen-Mappings

In YAWL werden Abbildungen von Netzvariablen auf Taskvariablen und umgekehrt durch XQuery-Ausdrücke beschrieben. Zur Unterscheidung dienen zwei XML-Elemente. Das „startingMappings“-Element beinhaltet alle Mappings, in denen eine Input-Netzvariable auf eine Input-Taskvariable abgebildet wird. Der folgende XML-Ausschnitt zeigt die Syntax einer derartigen Abbildung.

```
<mapping><expression query =
    “&lt;Taskvariable&gt;{/Netzdekomposition/Netzvariable/text()}&lt;Taskvariable&gt;“/>
</mapping>
```

Das Attribut „query“ enthält den für das Mapping benötigten XQuery-Ausdruck. Der Audruck nutzt sowohl die Identifier der Task- und Netzvariable, sowie die ID der entsprechenden Netzdekomposition. Diese Identifikatoren wurden in den ersten beiden Abarbeitungsschritten umbenannt, um Konflikte zu vermeiden. Das Variablen-Mapping muss entsprechend an diese Konvention angepasst werden. Die Methode „updateTaskMappings“ parst die XQuery-Ausdrücke sämtlicher Mappings des aktuellen Prozessfragments und fügt den Unterstrich „_“, sowie die „Specification-URI“ an den richtigen Stellen an.
Alle Abbildungen von Output-Taskvariablen auf Output-Netzvariablen werden für eine Task in dem XML-Element „completedMappings“ gesammelt.

```
<mapping><expression query =
    “&lt;Netzvariable&gt;{/Taskdekomposition/Taskvariable/text()}&lt;Netzvariable&gt;“/>
</mapping>
```

Der XQuery-Ausdruck unterscheidet sich leicht vom ersten Ausdruck. Anstatt der Netzdekomposition ist in diesem Fall die Umbenennung einer Taskdekomposition erforderlich.

Kopieren der Dekompositionen

Die ersten drei Schritte haben die Elemente eines Prozessfragments für das Kopieren in das neue Prozessmodell (XML-Dokument) vorbereitet. Die vierte Maßnahme wird von der Methode „addDecompositions“ realisiert. Sie überträgt sämtlicher Netz- und Taskdekompositionen in das Zieldokument. Von diesem Schritt ist die Dekomposition des *Wurzelnetzes* ausgeschlossen. Das Grundgerüst des neuen Prozessmodells besitzt bereits eine *Root-Net-Decomposition*. Weiterhin wurden die „flowsInto“-Elemente der ProcessControlElements von der Wurzelnetzdekomposition bisher nicht an den spezifizierten Modus (parallel oder sequenziell) angeglichen. Dies erfolgt im letzten Schritt der Iteration.

Kopieren und Angleichen der ProcessControlElements

Die Bearbeitung des letzten Schrittes unterscheidet sich in Abhängigkeit vom übergebenen Modus. Die parallele Anordnung des aktuellen Prozessfragments erfordert das Einhängen der Komponente in einen AND-Split bzw. AND-Join. Die Input-Condition des Fragments wird durch den AND-Split ersetzt und die Output-Condition entsprechend durch den AND-Join.
Ein Spezialfall ist wieder die erste Komponente. Mit ihr müssen der AND-Split und der

AND-Join in das neue Prozessmodell integriert werden. Hierzu erschafft die Methode „addControlElements“ zwei neue Task-Elemente mit dem Namen „Dummy“ und spezifizert deren Split- und Join-Verhalten entsprechend. Anschließend werden die Dummy-Elemente zu dem neuen Prozessmodell hinzugefügt und mit der Input- bzw. Output-Condition verbunden. Dies geschieht, indem die Input-Condition des neuen Prozessmodells ein „flowsInto“-Element mit der ID des Dummy-Splits erhält. Die Output-Condition muss nicht angeglichen werden. Hingegen benötigt der Dummy-Join ein neues „flowsInto“-Tag, welches die ID der Output-Condition beinhaltet.
Zusätzlich werden die Dummy-Elemente in den globalen Variablen „emptySplit“ und „emptyJoin“ vermerkt. Dies begründet sich auf der Tatsache, dass die Elemente für jeden Iterationsschritt der parallelen Komposition benötigt werden.

Für das Verbinden des aktuellen Prozessfragments mit dem AND-Split gibt es zwei Fälle, die beachtet werden müssen. Der einfachste Fall ist, dass die Input-Condition des Prozessfragments nur eine nachfolgende Task besitzt. In diesem Fall erhält der Dummy-Split im neuen Prozessmodell ein zusätzliches „flowsInto“-Element, welches auf den einzigen Nachfolger der Input-Condition der Komponente zeigt (siehe Abb. 6.7A).

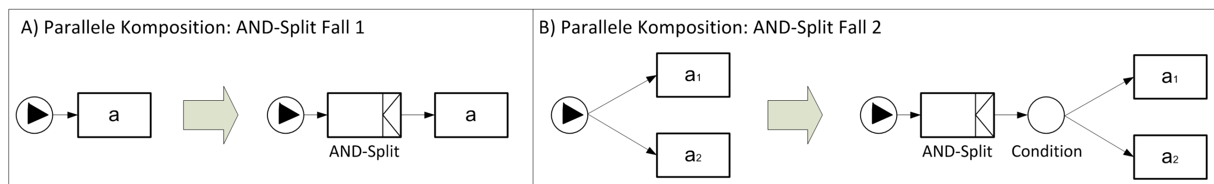


Abbildung 6.7: Parallele Komposition: Verbinden eines Fragments mit AND-Split

Referenziert die Input-Condition hingegen mehrere Tasks, muss die Bedingung erhalten bleiben. Erst zur Laufzeit entscheidet sich der Nutzer des Systems für einen konkreten Prozesszweig, den er ausführen möchte. Folglich dürfen die Nachfolger nicht direkt mit dem AND-Split verbunden werden. Abbildung 6.7B bildet die richtige Anordnung ab.
Die Methode „addControlElements“ fügt zu diesem Zweck eine Condition mit dem Namen „DummyCondition“ in das neue Prozessmodell ein. Diese wird mit dem AND-Split verbunden und erhält die gleichen „flowsInto“-Elemente wie die Input-Condition des aktuellen Prozessfragments.

Das Verbinden einer Komponente mit dem AND-Join funktioniert nach dem ähnlichen

Prinzip. Im einfachsten Fall existiert nur eine Aktivität, welche mit der Output-Condition der aktuellen Komponente verbunden ist. Die „addControlElements“-Methode sucht diese Task, welche die Output-Condition des Prozessfragments referenziert. Das Austauschen der Referenz (Abändern des „flowsInto“-Elements) durch den Identifier des Dummy-Joins passt den Kontrollfluss entsprechend an. Abbildung 6.8A illustriert das Verbinden einer Komponente mit dem Dummy-Join.

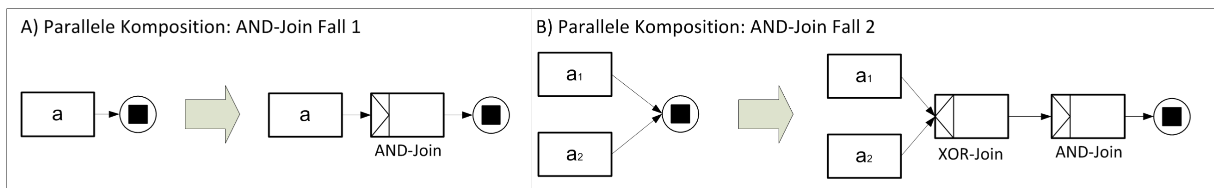


Abbildung 6.8: Parallele Komposition: Verbinden eines Fragments mit AND-Join

Eine weitere Möglichkeit ist, dass mehrere Tasks im Prozessfragment existieren, welche auf die Output-Condition zeigen. Laufen mehrere Kontrollflüsse in einer Output-Condition zusammen, bedeutet dies, dass der Kontrollfluss vorab durch eine Condition verzweigt wurde. Es kann sich in diesem Fall nicht um nebenläufige Zweige handeln, da parallele Kontrollflüsse durch einen AND-Join vor der Output-Condition synchronisiert werden müssen. Folglich funktioniert die Output-Condition des aktuellen Fragments wie ein XOR-Join. Die bedingten Zweige werden in einer zusätzlichen Dummy-Task vor dem Dummy-Join zusammengeführt (siehe Abb. 6.8B). Dies erfordert das Abändern der „flowsInto“-Elemente der letzten Tasks der bedingten Zweige. Diese verweisen anschließend auf den neuen XOR-Dummy, der eine Referenz auf den AND-Split besitzt.

Nach dem Angleichen sämtlicher Referenzen können die ProcessControlElements der aktuellen Komponente ins Zieldokument kopiert werden. Für die Übertragung ist die Reihenfolge von Bedeutung. Die YAWL-Spezifikation schreibt vor, dass das erste Kindelement des „processControlElements“-Tags die Input-Condition ist. Entsprechend ist das letzte Kindelement die Output-Condition. Die Task- und Condition-Elemente können beliebig dazwischen eingefügt werden.

Erhält die „merge“-Methode den Modus „Sequential“, ändert sich der Ablauf des letzten Abarbeitungsschritts. Das erste Prozessfragment wird an die Input-Condition im XML-Grundgerüst gehängt. Eine in der Klasse „Merger“ global definiert Liste mit dem Na-

men *ElementBuffer* speichert zudem die Task-Elemente, welche die Output-Condition der Komponente referenzieren. Für das Verbinden der aktuellen Komponente mit einem Vorgängerfragment existieren drei Fälle.

1. **Fall**: Der ElementBuffer besitzt nur ein Element und das aktuelle Fragment enthält nur eine Task, welche von der Input-Condition referenziert wird. Dies ist der trivialste Fall. Das Angleichen des „flowsInto"-Elements der Task im ElementBuffer genügt. Die neue Referenz-ID ist der Identifier der ersten Aktivität der aktuellen Komponente (siehe Abb.6.9A).

2. **Fall**: Der ElementBuffer besitzt mehrere Elemente und das aktuelle Fragment enthält nur eine Task, welche von der Input-Condition referenziert wird. Ähnlich wie bei der parallelen Komposition bedeutet das Zusammenlaufen mehrerer Zweige in einer Output-Condition, dass diese Zweige im Vorgängerfragment durch eine Condition entstanden sind. In diesem Fall müssen die bedingten Zweige durch einen XOR-Split zusammengeführt werden. Die erste Task des aktuellen Fragments besitzt als Standard ein solches Join-Verhalten. Folglich werden die „flowsInto"-Tags der Aktivitäten im ElementBuffer an die ID der ersten Task des aktuellen Fragments angeglichen. (siehe Abb. 6.9B)

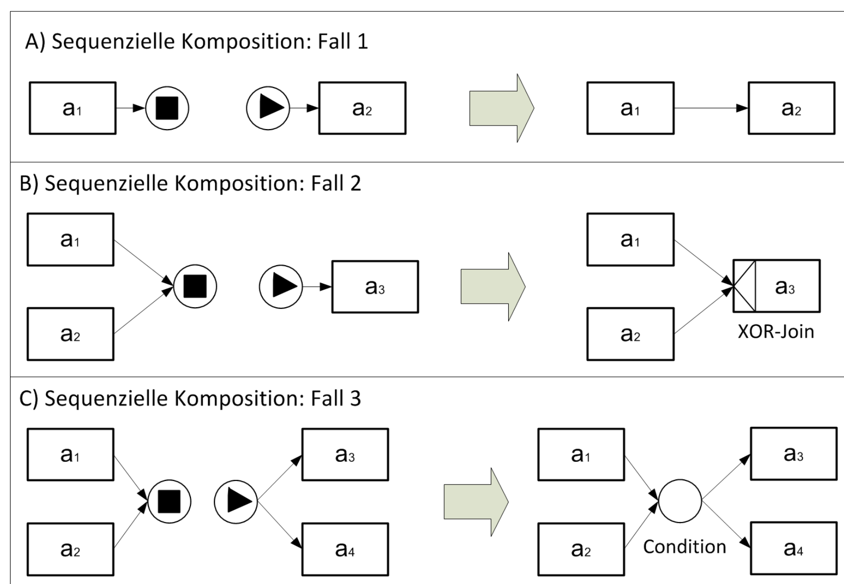


Abbildung 6.9: Sequenzielle Komposition: Direktes Verbinden zweier Fragmente

3. **Fall**: Der ElementBuffer besitzt eins oder mehrere Elemente und das aktuelle Fragment enthält mehrere Aktivitäten, welche von der Input-Condition referenziert werden. Mehrere ausgehende Zweige einer Input-Condition bedeuten immer die bedingte Ausführung eines *einzelnen* Zweiges. Folglich muss die Condition erhalten bleiben. Die Output-Condition des Vorgängerfragments und die Input-Condition der aktuellen Komponente werden praktisch zusammengefasst. Hierzu erschafft die Methode „addControlElements“ eine Dummy-Condition, welche die identischen „flowsInto“-Elemente erhält, wie die Input-Condition der aktuellen Komponente. Weiter werden alle Aktivitäten im ElementBuffer an die neue Dummy-Condition angeglichen. (siehe Abb. 6.9C)

Das letzte Prozessfragment wird zusätzlich mit der Output-Condition des neuen Prozessmodell verbunden. Das Kopieren der Elemente muss für die serielle Komposition nicht zwangsläufig am Ende erfolgen, da während der Iteration der Komponenten nur Dummy-Conditions bzw. Elemente des ElementBuffers verändert werden.

Kapitel 7

Schlussbetrachtung

7.1 Zusammenfassung

Es existieren mehrere Ansätze für die effiziente Modellierung und Verwaltung von Prozessvarianten in Workflow-Managementsystemen. Für die Umsetzung des Publikationsprozesses erwies sich die Unterspezifikation nach dem Late Modelling Prinzip als geeignet. Das Verfahren ermöglicht die Auswahl und den Austausch von Varianten zur Laufzeit des Prozesses. Ferner erfüllt die Unterspezifikation weitere wichtige Anforderungen, wie ein übersichtliches und redundanzfreies Prozessmodell und die Instanziierung nur benötigter Prozessteile.
Für die Realisierung der Unterspezifikation wurde das WfMS YAWL gewählt. Das System ist wie eine Service-orientierte Architektur aufgebaut und kann durch Custom YAWL Services einfach um zusätzliche Funktionalitäten erweitert werden. Es wurde ein Service mit dem Namen Composition Service vorgestellt, der durch die dynamische Komposition von Prozessfragmenten das Einbinden mehrerer Komponenten an einen Konfigurationspunkt ermöglicht. Der Service ist in der Lage, Prozessmodelle seriell und parallel anzuordnen. Die Auswahl der Prozessfragmente erfolgt dynamisch auf Basis der Dokumentinstanz, welche gerade für den Publikationsvorgang verwendet wird.
Mit der Implementation des Composition Service wurde nachgewiesen, dass das Konzept der Unterspezifikation nach dem Late Modelling Prinzip mit dem WfMS YAWL realisierbar und anwendbar ist.

7.2 Ausblick

Der Composition Service wurde nur prototypisch umgesetzt. Er ist in der Lage, Netze seriell oder parallel zu verbinden. Hierzu werden die Namen von Netzvariablen, sowie die Identifikatoren von Task- und Netzdekompositionen manipuliert, sodass diese im komponierten Prozessmodell eindeutig sind. Es erfolgt weiterhin eine Angleichung der Mappings von Netzvariablen auf Taskvariablen und umgekehrt.
Diese Schritte ermöglichen die Komposition einfacher Netze. Viele Aspekte, die YAWL anbietet, wurden bisher nicht betrachtet. Dazu zählen z.B Mehrfachinstanzen von YAWL-Netzen und -Tasks. *Multiple instance*-Variablen kann der Composition Service nicht verarbeiten. Ein anderes Beispiel sind *Timeout-Tasks*. Der Composition Service bietet folglich viel Potenzial für Verbesserungen, die umgesetzt werden können.
Ferner lag der Fokus dieser Arbeit auf der Komposition von Prozessbeschreibungen. Es bleiben Fragen offen, wie beispielsweise die Behandlung von Abbrüchen oder Änderungen des Publikationsvorgangs zu einer Zeit, in der bereits mehrere Konfigurationspunkte durchlaufen wurden. In derartigen Fällen muss der Publikationprozess vorgenommene Änderungen zurücknehmen können.
Ein Beispiel ist die Veröffentlichung eines Textdokuments mit Bildern und einem Video welches der Autor nachträglich ändert und das Video entfernt, nachdem bereits einige Video-bezogenen Fragmente abgearbeitet wurden. Eine Lösung für das Problem ist die Anwendung von *Undo-Fragmenten*, die alle Änderungen einer bestimmten Komponente rückgängig machen. Derartige Aspekte wurden nicht betrachtet, sind aber für die Umsetzung des Publikationsprozesses relevant.
Zuletzt wurden die Entscheidungen für bzw. gegen einen Ansatz für die Umsetzung des Publikationsprozesses in Abhängigkeit von der Prozessmodellierungssprache YAWL getroffen. Es ist ebenso möglich, dass die Unterspezifikation nach dem Late Modelling Prinzip von anderen Workflow-Managementsystemen realisierbar ist. Der PROVOP-Ansatz stellte sich zudem als sehr vielversprechend heraus. Dieser passt Prozessinstanzen zur Laufzeit an. Ein WfMS, welches den Ansatz umsetzt, ist ADEPT [RD09] und bildet eine Alternative zu YAWL.

Abkürzungsverzeichnis

Abb	Abbildung
API	Application Programming Interface
CDATA	Character Data
CMS	Content Management System
CO	Change Operation
CS	Composition Service
DC	Dublin Core
DTD	Dokumenttypdefinition
ID	Identifier
LTL	Linear Temporal Logic
PCDATA	Parsed Character Data
PROVOP	Process Variants by Options
SOA	Service-orientierte Architektur
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WRM	Workflow Reference Model
WSI	Web Service Invoker
YAWL	Yet Another Workflow Language

Literaturverzeichnis

[Bar10] Barthel, Kai U.: *Visual Information Retrieval: Features und Metriken.* Vorlesung, Hochschule für Technik und Wirtschaft in Berlin, Letzter Zugriff: 12.03.2011. `http://www.f4.htw-berlin.de/~barthel/veranstaltungen/SS10/IR/02_IR_Features_und_Metriken.ppt.pdf`. Version: 2010

[DF] Dumas, Marlon ; Fjellheim, Tore: *Connecting Custom Services to the YAWL Engine*

[EF00] Endres, Albert ; Feller, Dieter W.: *Digitale Bibliotheken : Informatik-Lösungen für globale Wissensmärkte.* Bd. 1. dpunkt.verlag, 2000

[ES10] Ekenel, Hazim K. ; Stiefelhagen, Rainer: *Content-based Image and Video Analysis - High-Level Feature Detection.* Vorlesung, Universität Karlsruhe, Letzter Zugriff: 12.03.2011. `http://cvhci.anthropomatik.kit.edu/download/cbir10/CBIR_06_HLF.pdf`. Version: 2010

[FMRW10] Fahland, Dirk ; Mendling, Jan ; Reijers, H.A. ; Weber, Barbara: Declarative versus Imperative Process Modeling Languages: The Issue of Maintainability. In: *Business Process* (2010), S. 477–488

[GVJVLR08] Gottschalk, F. ; Van Der Aalst, W.M.P. ; Jansen-Vullers, M.H. ; La Rosa, M.: Configurable workflow models. In: *International Journal of Cooperative Information Systems* 17 (2008), Nr. 2, S. 177–221

[HB08] Hallerbach, Alena ; Bauer, Thomas: Anforderungen an die Modellierung und Ausführung von Prozessvarianten. In: *Datenbank Spektrum* (2008), S. 48–58

[HBR08] Hallerbach, Alena ; Bauer, Thomas ; Reichert, Manfred: Managing process variants in the process life cycle. In: *ICEIS 2008 - Proceedings of the 10th International Conference on Enterprise Information Systems* 2 ISAS (2008), S. 154–161

[HBR10] Hallerbach, Alena ; Bauer, Thomas ; Reichert, Manfred: Configuration and management of process variants. In: *Handbook on Business Process Management 1* (2010), S. 237–255

[HD02] Heuer, Andreas ; Dittrich, Klaus R.: Schwerpunktthema : Content Management und digitale Bibliotheken. In: *Datenbank-Spektrum* 2 (2002), Nr. 4, S. 7–8

[Hom] Hombach, Laura E.: *Vortragsreihe zum Thema „Aktive Datenbanken“*. Lehrstuhl für Datenbanken und Informationssysteme der Friedrich-Schiller-Universität Jena, Letzter Zugriff: 09.04.2011. `http://www.slidefinder.net/k/konfluenz_aktiven_vortragsreihe_thema_aktive/16953007`

[HSS08] Heuer, A. ; Saake, G. ; Sattler, K.U.: *Datenbanken: Konzepte und Sprachen*. Mitp bei Redline, 2008. – ISBN 9783826616648

[Lüt02] Lützenkirchen, Frank: MyCoRe - Ein Open-Source-System zum Aufbau digitaler Bibliotheken. In: *Datenbank-Spektrum* 2 (2002), Nr. 4, S. 23–27

[Mel07] Melzer, Ingo: *Service-orientierte Architekturen mit Web Services : Konzepte - Standards - Praxis*. Bd. 2. Spektrum Akad. Verl., 2007

[PV06] Pesic, M. ; Van der Aalst, Wil M.P.: A Declarative Approach for Flexible Business Processes Management. In: *Business Process Management Workshops*, Springer Verlag, 2006, S. 169–180

[RD09] Reichert, Manfred ; Dadam, Peter: Enabling Adaptive Process-aware Information Systems with ADEPT2. In: *Information Systems* (2009), S. 173–203

[RR01] Rothfuss, Gunther ; Ried, Christian: *Content Management mit XML : Grundlagen und Anwendungen*. Bd. 1. Springer-Verlag, 2001

[RTE04a] Russell, Nick ; Ter Hofstede, Arthur H. M. ; Edmond, David: Workflow Data Patterns. (2004)

[RTE04b] Russell, Nick ; Ter Hofstede, Arthur H. M. ; Edmond, David: Workflow Resource Patterns. (2004)

[RTE06] Russell, Nick ; Ter Hofstede, Arthur H. M. ; Edmond, David: Exception handling patterns in process-aware information systems. In: *BPM Center Report BPM-06-04, BPMcenter. org* (2006), S. 06–04

[RTM06] Russell, Nick ; Ter Hofstede, Arthur H. M. ; Mulyar, Nataliya: Workflow Control-Flow Patterns A Revised View. In: *BPM Center Report BPM-06-04, BPMcenter. org* (2006)

[Sch02] Schmitt, Ingo: Retrieval in Multimedia - Datenbanksystemen. In: *Datenbank-Spektrum* 2 (2002), Nr. 4, S. 28–35

[Sch05] Schick, Sebastian: *Publikationsprozesse in digitalen Bibliotheken - Mittel und Methoden der Autorenunterstützung.* Diplomarbeit, 2005

[Sch09] Schick, Sebastian: *Digitale Bibliotheken und Content Management.* Vorlesungsskript, 2009

[SMR+07] Schonenberg, M.H. ; Mans, R.S. ; Russell, N.C. ; Mulyar, N.A. ; van Der Aalst, W.M.P: Towards a taxonomy of process flexibility (extended version). In: *BPM Center Report BPM-07-11, BPMcenter. org* (2007)

[SOS05] Sadiq, S ; Orlowska, M ; Sadiq, W: Specification and validation of process constraints for flexible workflows. In: *Information Systems* 30 (2005), Juli, Nr. 5, S. 349–378

[SSO01] Sadiq, Shazia ; Sadiq, Wasim ; Orlowska, Maria: Pockets of Flexibility in Workflow Specification Dimensions of Change in Workflows. In: *Science* (2001), S. 513–526

[TCA10] Ter Hofstede, Arthur H. M. ; Clemens, Stephan ; Adams, Michael: YAWL Technical Manual. In: *The YAWL Foundation* (2010)

[Ter10] Ter Hofstede, Arthur H. M.: YAWL User Manual. In: *The YAWL Foundation (October 2008)* (2010)

[Tur09] Turau, Volker: *Algorithmische Graphentheorie*. Oldenbourg Verlag, 2009

[TV05] Ter Hofstede, Arthur H.M. ; Van der Aalst, Wil M.P.: YAWL : Yet Another Workflow Language (Revised version). In: *Information Systems* (2005), S. 245–275

[TVAR10] Ter Hofstede, Arthur H.M. ; Van der Aalst, Wil M.P. ; Adams, M. ; Russell, N.: *Modern Business Process Automation: YAWL and its Support Environment*. Springer-Verlag, 2010

[VP06] Van der Aalst, Wil M.P. ; Pesic, M.: Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware. In: *BPM Center Report BPM-06-09, BPMcenter. org* (2006)

[Wor99] Workflow Management Coalition: *Terminology & Glossary*. Document Number: WFMC-TC-1011, 1999

Selbsterklärung

Mit meiner Unterschrift versichere ich gemäß §26(1) der Prüfungsordnung der Universität Rostock, dass ich diese Arbeit selbständig und nur mit Hilfe der angegebenen Quellen und Hilfsmittel verfasst habe.

______________________________ ________________________

Ort, Datum Unterschrift

Thesen

1. Autoren müssen immer häufiger multimediale Lehrmaterialien oder Multimediadokumente erstellen und diese in digitalen Bibliotheken zur Verfügung stellen. Die Dokumentierung dieses Ablaufs als Geschäftsprozess und die anschließende Automatisierung in einem Workflow-Managementsysteme ersparen Autoren viel Aufwand.

2. Herkömmliche Ansätze zur Modellierung und Verwaltung von Prozessvarianten eignen sich nicht für die Umsetzung des Publikationsprozesses.

3. Die deklarative Modellierung ermöglicht die Spezifikation von flexiblen Prozessmodellen. Für die Realisierung von Prozessvarianten ist der deklarative Ansatz dennoch nicht geeignet.

4. Die Prozesskonfiguration kann Prozessvarianten in einer Phase zwischen Modellierung und Ausführung bestimmen. Die Auswahl zur Laufzeit anhand von Kontextinformationen ist mit der Konfiguration von Prozessbeschreibungen nicht möglich.

5. Die Unterspezifikation vervollständigt Prozess-Spezifikationen zur Laufzeit und vermeidet, im Gegensatz zum PROVOP-Ansatz, die Anpassung von laufenden Prozessinstanzen. Weiterhin bietet das Late Modelling Prinzip die Möglichkeit der dynamischen Komposition von Prozessmodellen. Das Verfahren eignet sich gut für die Realisierung des Publikationsvorgangs.

6. Die Kombination von der Unterspezifikation nach dem Late Binding Prinzip und der Prozesskonfiguration erfüllt alle Anforderungen für die Umsetzung des Publikationsprozesses.

7. Das Workflow-Managementsystem YAWL kann für die Realisierung der Unterspezifikation nach den Prinzipien des Late Bindings und Late Modellings und somit ebenfalls für die Implementierung des Publikationsprozesses eingesetzt werden.