Universität Koblenz
Fachbereich 4, Informatik
Institut für Softwaretechnik
AG Lautenbach

iProcess

- Geschäftsprozessmodellierung mittels YAWL -

Studienarbeit
im Studiengang Informatik

vorgelegt von
Mario Demuth
202120809

Betreuer:	Prof. Dr. Kurt Lautenbach / AG Lautenbach
	Dr. Stephan Philippi / AG Lautenbach
Erstgutachter:	Prof. Dr. Kurt Lautenbach / AG Lautenbach
Zweitgutachter:	Dr. Stephan Philippi / AG Lautenbach

Koblenz, im September 2006

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	[ ]	[ ]
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	[ ]	[ ]

. . . . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

„Modelle sind nichts anderes als ein Mittel, um sich an die wirtschaftliche Wirklichkeit heranzutasten. Sie sind gewissermaßen Bilder, um eben diese Wirklichkeit zu begreifen. [2]“ Im spezielleren Falle der Prozess- bzw. der Geschäftsprozessmodellierung sind diese eine Hilfe, um unternehmensinterne Abläufe bzw. Geschäftsprozesse zu identifizieren, dokumentieren, optimieren und /oder zu steuern. Das Modellieren selbst kann verschiedene Ausprägungen haben. Häufig findet man eine Kombination aus informalen, semiformalen und formalen Bestandteilen. Informale textuelle Beschreibungen werden eingesetzt, um Abläufe zu erläutern. Semiformale graphische Elemente dienen der Visualisierung, wohingegen formale Darstellungen als Basis zur Simulation oder der automatischen Codegenerierung Verwendung finden.
Es können verschiedene Sichten auf die zu erstellenden Arbeitsablaufspezifikationen (*workflow specifications*) existieren: eine Kontrollfluss- (bzw. Prozess-), Daten-, Ressourcen- und operationale Perspektive. Die Kontrollflusssicht zeigt beispielsweise Aktionen und deren Abarbeitungsreihenfolge, die durch verschiedene Kontrollkonstrukte verändert werden können: z.B. Strukturen für Folgen (*sequence*), Wahlmöglichkeiten (*choice*), Parallelität (*parallelism*) und Synchronisation (*synchronisation*). Die Datenperspektive beleuchtet den Fluss von Unternehmensobjekten bzw. -dokumenten zwischen einzelnen Aktionen, und stellt somit auch die Vor- bzw. Nachbedingungen zur Ausführung einer Aktion. Aus Ressourcenperspektive werden die Rollen von Unternehmensressourcen, wie z. B. Arbeiter oder Maschinen, definiert, die bei der Abarbeitung einer bestimmten Aktion eingesetzt werden. Die operationale Perspektive gibt schlussendlich an, wie die soeben genannten Dokumente be- bzw. verarbeitet werden müssen.
Zahlreiche sogenannte Arbeitsablaufmanagementsysteme (*workflow management systems*) werden für diese Aufgaben entwickelt. Aufgrund von fehlender universeller Organisationstheorie und standardisiertem Modellierungskonzept gibt es eine Vielzahl verschiedener, meist proprietärer, systemabhängiger Modellierungssprachen. Eine Studie von W.M.P. van der Aalst und A.H.M. ter Hofstede von den Universitäten Eindhoven (Niederlande) und Queensland (Australien) zeigt, dass es große Diskrepanzen zwischen diesen Modellierungssprachen gibt. Fünfzehn gängige Arbeitsablaufmanagementsysteme wurden auf ihre Verträglichkeit mit diversen *workflow patterns*[1] getestet. Man ist zu dem Schluss gekommen, dass keines der getesteten Systeme alle geprüften Patterns mit zumutbarem Modellierungsaufwand realisieren konnte.
Aus dieser Unzulänglichkeit der Modellierungsmöglichkeiten entsteht unter van der Aalst und ter Hofstede die formale, aber dennoch graphische Modellierungssprache YAWL (*Yet Another Workflow Languange*). YAWL bietet die Möglichkeit, jedes der workflow patterns mit geringem Aufwand zu modellieren und bietet somit ausreichende Mittel, eine Kontrollflussperspektive für ein Arbeitsablaufmanagementsystem zu realisieren.
Das Implementieren einer Basis für ein workflow management system soll Ziel dieser Studienarbeit sein. Da Kontrollflüsse eine äußerst wichtige Rolle in einem solchen System spielen,

wird dieses Programmgrundgerüst die Kontrollflussstrukturen von YAWL umsetzen, um als Grundlage für die anderen beschriebenen Perspektiven zu dienen.

2 YAWL - Yet Another Workflow Language

YAWL (*Yet Another Workflow Language*) ist eine Open-Source und dementsprechend toolunabhängige, formale und dennoch graphische Prozessmodellierungssprache und der gelungene Versuch von W.M.P. van der Aalst und A.H.M. ter Hofstede, ein Modellierungskonzept zu schaffen, welches die verschiedenen Workflow Patterns [1] mit zumutbarem Modellierungsaufwand realisieren kann.
Als Basis dienen Petri-Netze (PN), deren Vorteil es ist, trotz ihrer graphischen Notation dennoch einer formalen Semantik zu folgen. Diese wurden um verschiedene Elemente erweitert, um den Anforderungen der Patterns gerecht zu werden. Konstrukte für multiple Instanzen fortgeschrittener Synchronisation und für die Annullierungs Patterns (*cancellation patterns*) mussten geschaffen werden. Die folgenden Ausführungen beziehen sich auf die in [4] beschriebene Spezifikation von YAWL.

2.1 Visuelle Elemente

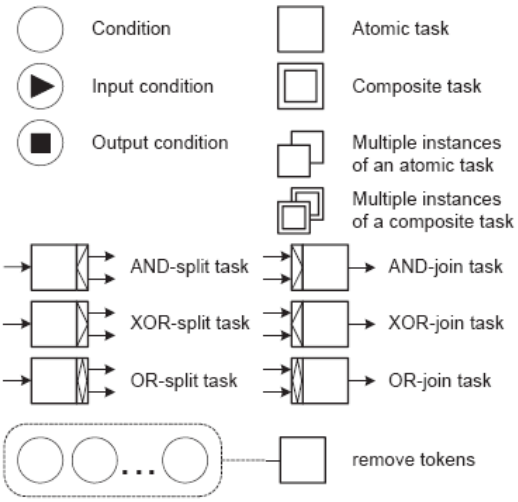

Abbildung 1: Symbole in der Modellierungssprache YAWL (Quelle: [4] S. 11)

Abbildung 1 zeigt Symbole, die in der Prozessmodellierungsspache YAWL Verwendung finden. Zu beachten ist, dass durch Rechtecke symbolisierte Aktionen (*tasks*) den ebenso dargestellten Transitionen der Petri-Netze entsprechen. Analoges gilt auch für die durch Kreise symbolisierten Bedingungen (*conditions*), welche mit Stellen in Petri-Netzen zu identifizieren sind. Weitere Symbole dienen der Veranschaulichung der Erweiterungskonstrukte von YAWL gegenüber der Petri-Netz-Basis, deren Erläuterung in den YAWL Unterkapiteln 2.2 und 2.3 zu finden ist.

2.2 Informale Beschreibung

Eine Arbeitsablaufspezifikation (*workflow specification*) in YAWL ist eine Menge von erweiterten Arbeitsablaufnetzen (*extended workflow nets (EWF-Nets*[1]*)*), die eine baumartige Hierarchie bilden. Im Folgenden sollen diese gemäß der englischen Abkürzung auch EWF-Netze genannt werden. In EWF-Netzen existieren wie in den Petri-Netzen zwei Kontentypen, Aktionen (*Tasks*)[vgl. Transitionen in PN] und Bedingungen (*Conditions*)[vgl. Stellen in PN]. Aktionen können atomar (*atomic*) oder zusammengesetzt (*composit*) sein, wobei eine zusammengesetzte Aktion immer ein eindeutiges EWF-Netz aus einem niedrigerem Hierarchielevel referenziert. Dementsprechend können atomare Aktionen als Blätter der baumartigen Struktur betrachtet werden. Der oberste Arbeitsablauf (*top level workflow*) ist das einzige EWF-Netz, das nicht von einer zusammengesetzten Aktion referenziert wird und bildet somit die Wurzel.

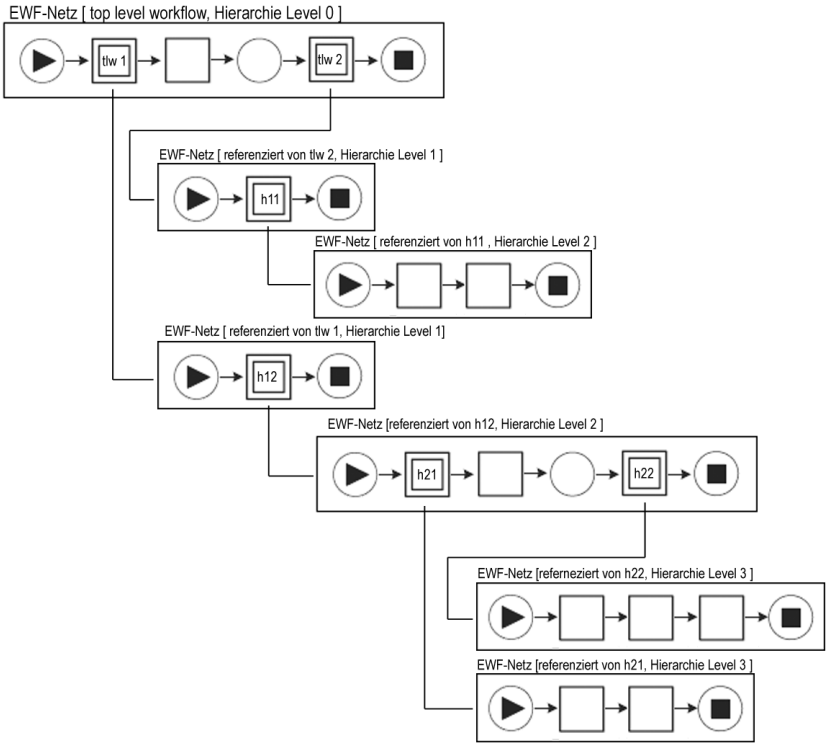

Abbildung 2: Beispielhafte Darstellung der Struktur einer Arbeitsablaufspezifikation (*workflow specification*)

Als Veranschaulichung der soeben beschriebenen Workflow Spezifikation soll Abbildung 2 dienen. Auf Hierarchielevel 0 existiert ein einziges EWF-Netz, das *top level workflow*, welches nicht durch eine der zusammengesetzten Aktionen referenziert wird. Zusammengesetzte Aktionen (z.B. $tlw1$ und $tlw2$) verweisen auf EWF-Netze einer tieferliegenden Hierarchieebene (z.B. $tlw1 \rightarrow$ Hierarchieebene 1) wodurch eine Baumstruktur entsteht. EWF-Netze, die Blätter dieses Baumes, dürfen somit ausschließlich atomare Aktionen beinhalten (z.B. EWF-Netz referenziert von *composit task* $h21$).

[1] Ein EWF-Netz ist eine Erweiterung der in [3, 5] von W.M.P van der Aalst beschriebenen *workflow nets*.

Ein EWF-Netz besteht aus Aktionen (*tasks*), Bedingungen (*conditions*) und Flüssen (*flows*) zwischen diesen. Da zwischen zusammengesetzter und atomarer Aktion prinzipiell nur ein Unterschied darin besteht, dass bei zusammengesetzter Aktion eine Verfeinerung[2] existiert, wird im Folgenden zusammenfassend nur noch der Begriff der Aktion verwendet. Sollte es dennoch nötig sein, zu unterscheiden, so geschieht dieses explizit. EWF-Netze definieren jeweils genau eine Eingangs-(*input*) und eine Ausgangsbedingung (*output condition*). Gerichtete Kanten zwischen Knoten symbolisieren einen Fluss. Im Gegensatz zu Petri-Netzen dürfen in einem EWF-Netz zwei Aktionen direkt miteinander verbunden werden. Semantisch gesehen kann eine implizite Bedingung zwischen direkt miteinander verbundenen Aktionen angenommen werden.
Aktionen werden um diverse Attribute erweitert, um den Modellierungsaufwand für einige Workflow-Patterns zu verkleinern. Aktionen können multiple Instanzen haben. Dementsprechend werden ihnen eine minimale und maximale Anzahl zu generierender Instanzen beim Anlaufen der Aktion zugeschrieben. Eine Abschlussschwelle (*termination threshold*) gibt an, wie viele der generierten Instanzen abgeschlossen haben müssen, um die gesamte Aktion zu beenden. Ein letzter Parameter bestimmt, ob weitere Instanzen nach der ursprünglichen Instanzgenerierung angelegt werden dürfen. Wert für diesen Parameter ist entweder „statisch“ (*static*), wenn nach der ursprünglichen Initialisierung keine weiteren Instanzen mehr generieren, „dynamisch“ (*dynamic*) falls weitere Instanzen angelegt werden dürfen. Aktionen sollen eine Vereinigungs- (*join*) sowie eine Spalteigenschaft (*split*) besitzen, die eine verbesserte Parallelisierung und Synchronisation erlauben. Werte für diese Attribute können „und“ (*AND*), „exklusives oder“ (*XOR*) sowie „oder“ (*OR*) sein. Um die Annullierungs-Patterns zu unterstützen, existiert schlussendlich eine Notation, die es einer Aktion erlaubt, alle Tokenobjekte (*Token* vgl. PN) in einem bestimmten Netzbereich zu entfernen.

2.3 Formale Definition

Definition 2.1 (EWF-Netz)**:**
Ein Erweitertes Arbeitsablaufnetz (*extended workflow net*) (EWF-Netz) N ist ein Tupel $(C, i, o, T, F, split, join, rem, nofi)$ derart, dass:

- C eine Menge von Bedingungen (*conditions*),
- $i \in C$ die Eingangsbedingung (*input condition*),
- $o \in C$ die Ausgabebedingung (*output condition*),
- T eine Menge von Aktionen (*tasks*),
- $F \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup (T \times T)$ die Flussrelation,

[2] Eine Aktion wird in meherere Teilaktionen zerlegt, symbolisiert durch ein weiteres EWF-Netz.

- jeder Knoten im Graph $(C \cup T, F)$ sich auf einem gerichteten Pfad von i nach o befindet,
- $split : T \rightarrow \{AND, XOR, OR\}$ das Spaltungsverhalten jeder Aktion spezifiziert,
- $join : T \rightarrow \{AND, XOR, OR\}$ das Vereinigungsverhalten jeder Aktion spezifiziert,
- $rem : T \not\rightarrow \mathbb{P}(T \cup C \setminus \{i, o\})$ spezifiziert, welche Teile des Netzes anulliert werden sollen,
- $nofi : T \not\rightarrow \mathbb{N} \times \mathbb{N}^{inf} \times \mathbb{N}^{inf} \times \{dynamic, static\}$ die Multiplizitätseigenschaften jeder Aktion (minimale- , maximale Anzahl zu generierenden Instanzen, Abschlussschwelle, und die dynamische/statische Instanzierungseigenschaft) spezifiziert.

Dem klassischen Petri-Netz entspricht das Tupel (C, T, F). Dabei ist C als Menge der Bedingungen gleichzusetzen mit den Stellen, T ist als Menge der Aktionen mit den Transitionen zu identifizieren und F als die Flussrelation zu betrachten. Es gibt allerdings zwei Besonderheiten: zum einen existieren zwei spezielle Bedingungen / Stellen i und o und zum anderen können auch direkte Verbindungen innerhalb der Flussrelation zwischen zwei Aktionen / Transitionen bestehen. Die Funktionen $split, join, rem$ und $nofi$ repräsentieren die erweiterten Attribute von Aktionen. $Split$ und $join$ sind Funktionen, die das Spalt- bzw. Vereinigungsverhalten der Aktionen bestimmen. Die partielle Funktion rem spezifiziert, welche Teile des Netzes bei Ausführung einer Aktion von Tokens befreit werden sollen. Zu beachten ist, dass sowohl von Bedingungen ($C \setminus i, o$) als auch von Aktionen Token entleert werden können. Das Entleeren einer Aktion entspricht dem Abbruch der Ausführung dieser Aktion. Für zusammengesetzte Aktionen ergibt sich dementsprechend die Entleerung aller Tokens im referenzierten EWF-Netz.

Definition 2.2 (Arbeitsablaufspezifikation)**:**
Eine Arbeitsablaufspezifikation (*workflow specification*) S ist ein Tupel (Q, top, T°, map) derart, dass:

- Q eine Menge von EWF-Netzen,
- $top \in Q$ der oberste Arbeitsablauf ist (*top level workflow*),
- $T^{\circ} = \cup_{N \in Q} T_N$ die Menge aller Aktionen (*tasks*) ist und es nach
- $\forall_{N_1, N_2 \in Q} N_1 \neq N_2 \Rightarrow (C_{N_1} \cup T_{N_1}) \cap (C_{N_2} \cup T_{N_2}) = \emptyset$ keine Namenskollisionen gibt.
- $map : T^{\circ} \not\rightarrow Q \setminus \{top\}$ ist eine surjektive, injektive und partielle Funktion, welche jeder zusammengesetzten Aktion (*composite task*) ein EWF-Netz zuweist.
- Die Relation $\{(N_1, N_2) \in Q \times Q | \exists_{t \in dom(map_{N_1})} map_{N_1}(t) = N_2\}$ ist ein Baum.

Q ist eine nicht leere Menge von EWF-Netzen mit einem speziellen obersten Arbeitsablauf top. T° ist die Menge aller Aktionen T aus Q. Ferner gibt es keine Namenskollisionen unter Bedingungen oder Aktionen aus Q; sollte es dennoch Namensüberlappungen geben, so werden diese durch Umbenennung behoben. Als Domäne für die Relation map gilt die Menge aller zusammengesetzten Aktionen, welchen jeweils ein EWF-Netz zugeteilt wird, das diese verfeinert.

3 iProcess - Geschäftsprozessmodellierung mittels YAWL

Im Rahmen der Studienarbeit entsteht ein Arbeitsablaufmanagementsystem (*workflow management system*) unter dem Namen *iProcess*. Als Modellierungssprache soll die in Kapitel 2 beschriebene Sprache YAWL verwendet werden, welche erlaubt, komplexe Geschäftsprozesse und deren Abarbeitungsreihenfolge zu modellieren. Somit kann iProcess einem Geschäftsprozessmanagement als Werkzeug dienen, die meist komplexen unternehmensinternen Sachverhalte zu identifiziern, dokumentieren, optimieren und / oder zu steuern. Die aus Kapitel 2 bekannten *workflow specifications* und EWF-Netze bilden die Grundlage für eine Kontrollflussperspektive. Daten-, Ressourcen und operationale Sichten können auf diese aufgesetzt werden, sind aber nicht Teil dieser Studienarbeit.
Die AG Lautenbach unter Prof. Dr. Kurt Lautenbach an der Universität Koblenz entwickelt zur Zeit eine Graphmodellierungsumgebung (*graph modeling framework*) NeMo, welche es ermöglicht, verschiedene Graphen zu zeichnen und diverse Algorithmen auf diesen abzuarbeiten. Unter anderem beschäftigt sich die AG hauptsächlich mit Petri-Netzen und deren Anwendungen. Da YAWL sowohl eine graphische als auch petri-netz-erweiternde Sprache ist, liegt es nahe, iProcess als eine Spezialisierung bzw. Weiterentwicklung des Basisframeworks NeMo anzulegen.
Die folgenden Unterkapitel werden erläutern, wie Kontrollflüsse mittels iProcess modelliert und diese im iProcess-Datenmodell gespeichert bzw. in der GUI[3] sichtbar gemacht werden.

3.1 Beispielmodellierung mit iProcess

Anhand eines Beispiels soll erläutert werden, wie man mit iProcess anhand YAWL aus Kapitel 2 modelliert. Abbildung 3 zeigt einen Screenshot von iProcess nach dem Systemstart.

[3] *graphical user interface.*

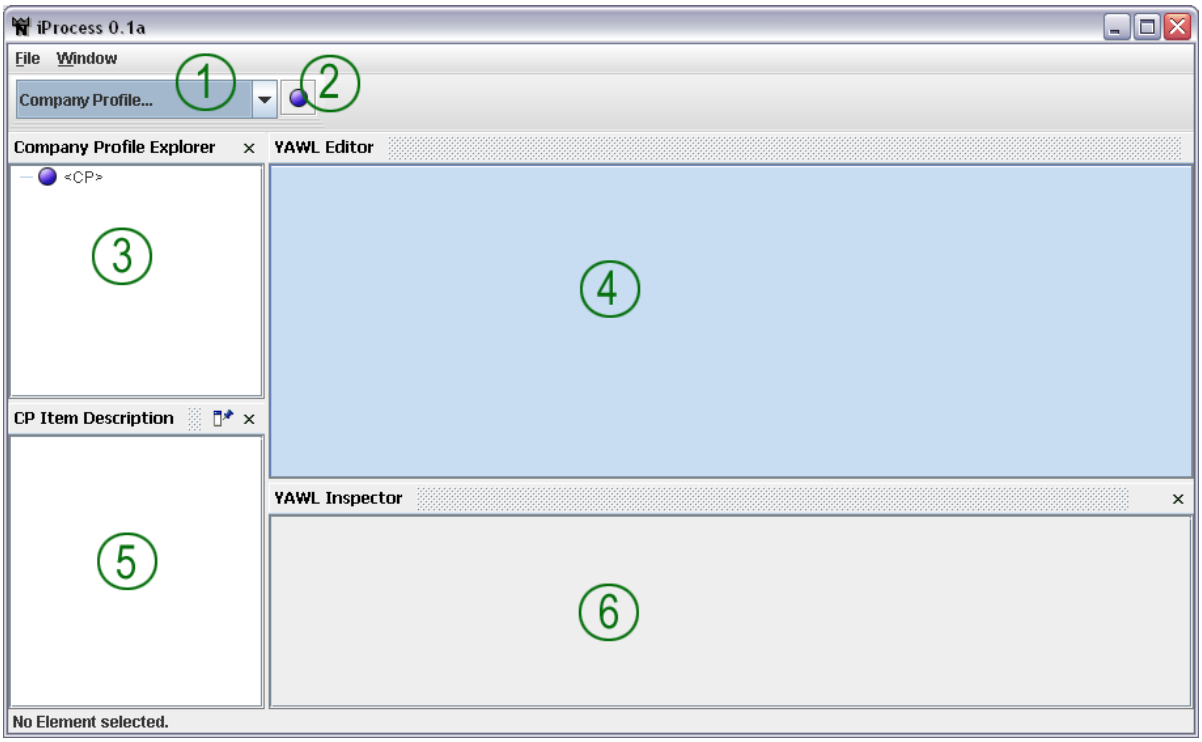

Abbildung 3: Screenshot nach dem Start von iProcess

Zu sehen sind sechs interaktive Bereiche, gekennzeichnet durch die Nummern eins bis sechs. Diese Nummern werden im Folgenden immer wieder kennzeichnend für die von ihnen markierten Bereiche genannt.

1. In Bereich eins befindet sich eine *ComboBox*, mit deren Hilfe ein Geschäftsprozessmodellierer die zuvor modellierten Daten zu einem Unternehmen öffnen kann. Diese Daten werden fortan zusammenfassend als „Company Profile“ bezeichnet. Es ist also möglich, zu mehreren verschiedenen Unternehmen ein solches Company Profile anzulegen.

2. Bereich zwei ist ein Schalter, um ein neues Company Profile anzulegen. Das Anlegen erfordert einen eindeutigen Unternehmensnamen und generiert eine Reihe von voreingestellten (*default*) Dokumenten die vielen der zu modellierenden Unternehmen gemein sind.

3. Der dritte Bereich, der Company Profile Explorer, erlaubt es dem Modellierer, durch die verschiedenen Dokumente des Company Profiles zu navigieren.

4. Der YAWL-Editor in Bereich vier ermöglicht es, YAWL-Graphen zu zeichnen.

5. Die Nummer fünf markiert einen Bereich, in dem zu dem gerade zur Bearbeitung geöffneten YAWL-Graphen passende, textuelle Beschreibungen angezeigt werden, welche selbst auch editiert werden können.

6. Bereich sechs, der YAWL-Inspector, zeigt Informationen zu Modellspezifika von im YAWL-Editor selektierten Elementen.

Wird ein Company-Profile zu einem „BeispielUnternehmen“ angelegt (z.B. mit Schalter (2)), so werden diverse default Dokumente in einer logischen Hierarchie generiert. Aus Kapitel 2 ist bekannt, dass für ein Unternehmen Arbeitsablaufspezifikationen (*workflow specifications*) von Bedeutung sind, welche die gesamten Arbeitsabläufe eines Unternehmens modellieren. In iProcess entspricht diese Arbeitsablaufspezifikation für ein Unternehmen dem Aufgabenrahmenplan.

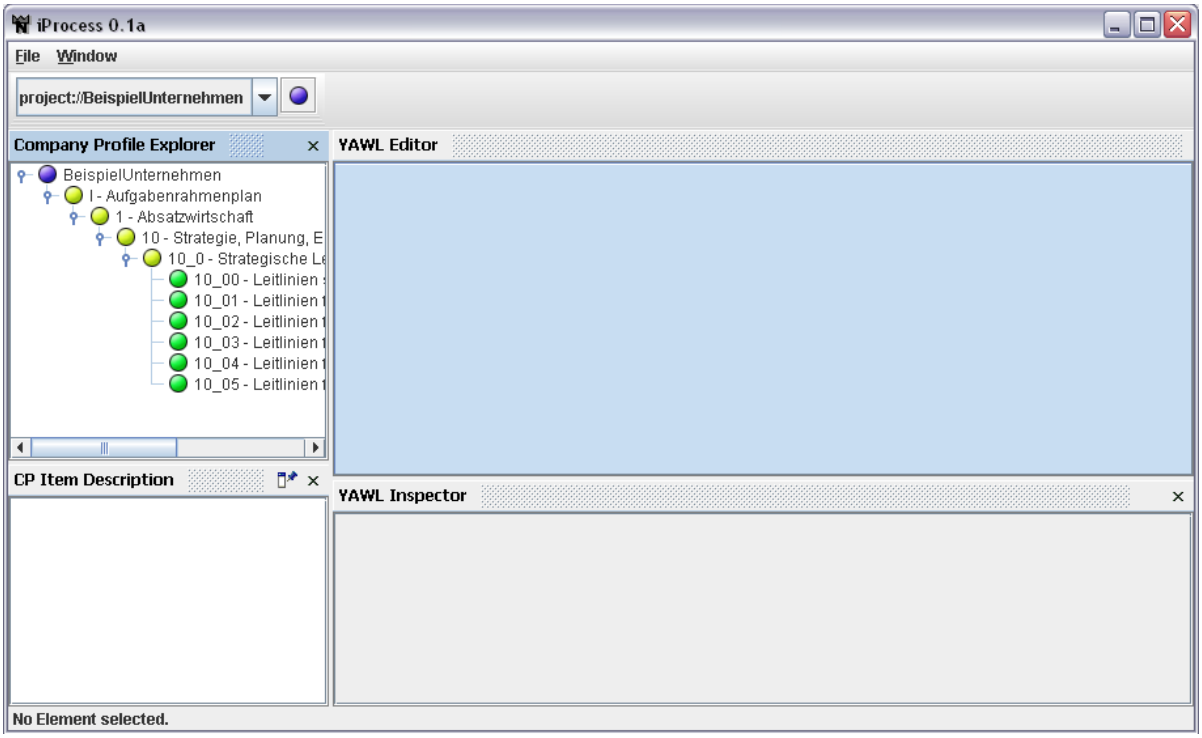

Abbildung 4: Screenshot nach dem Anlegen eines „BeispielUnternehmens“

Abbildung 4 zeigt iProcess nach dem Anlegen des BeispielUnternehmens. Bereich (1) gibt den Namen des aktuell geöffneten Company Profiles an: „BeispielUnternehmen“. Der Company Profile Explorer (3) zeigt den voreingestellten Aufgabenrahmenplan des Unternehmens. Durch doppeltes Klicken auf einen der Unterpunkte des Aufgabenrahmenplans können YAWL-Graphen (vgl. EWF-Netz aus Kapitel 2) erstellt und diesem Knoten zugeordnet werden. Beispielhaft soll der „Knoten 10_00 - Leitlinien Strategischer Planung Absatzwirtschaft“ modelliert werden.
Mit dem Doppelklick auf den Knoten „10_00 - Leitlinien Strategischer Planung Absatzwirtschaft“ (siehe Abbildung 5) im Company Profile Explorer wird ein neuer YAWL-Graph angelegt und diesem Knoten zugeordnet. Der neue YAWL-Graph bzw. das neue EWF-Netz wird initialisiert mit der obligatorischen Eingangsbedingung (*input condition*) und der Ausgangsbedingung (*output condition*). Im iProcess-Programm wird ein siebter Bereich oberhalb des YAWL-Editors sichtbar, der als Toolbar zum Zoomen im Graph, Gruppieren von Knoten und Einfügen weiterer Knoten bestimmt ist. Der YAWL-Editor zeigt den neu generierten Graph, der Beschreibungseditor in Bereich (5) eine zugehörige informale textuelle Beschreibung und

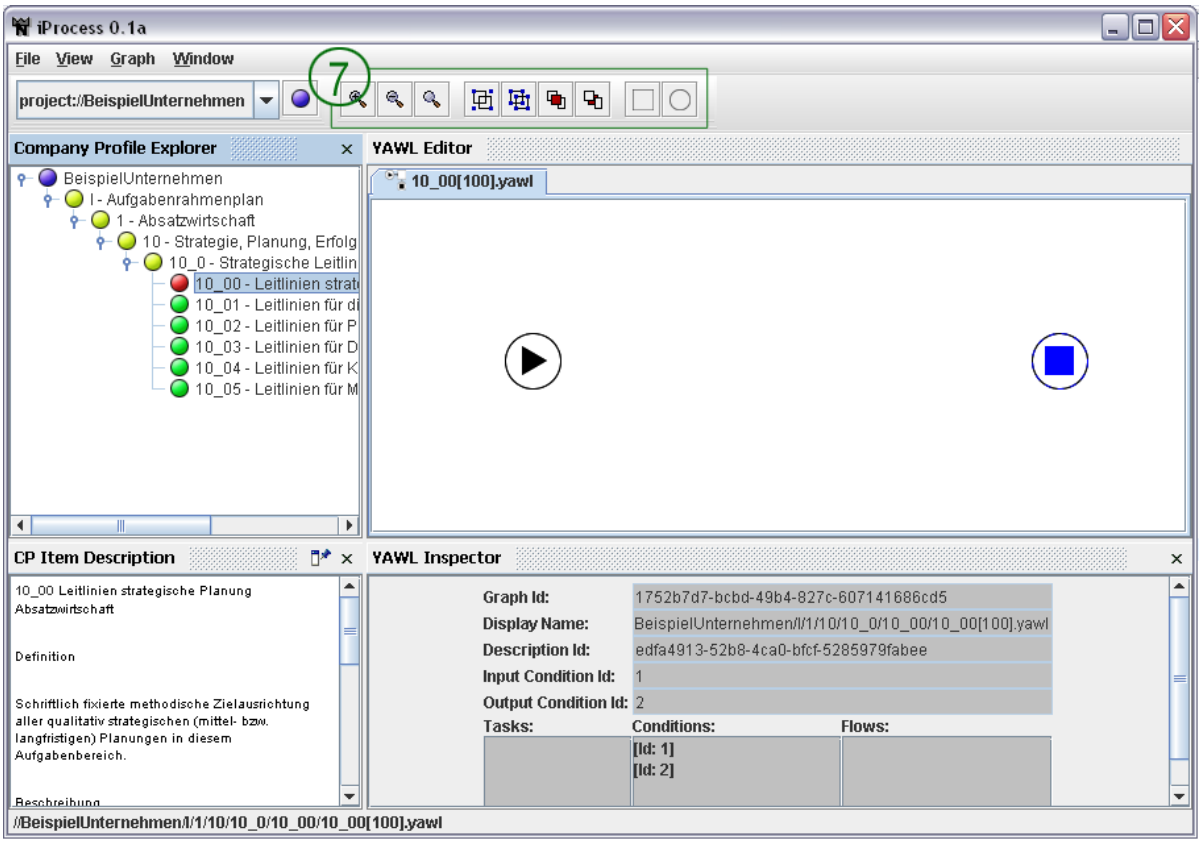

Abbildung 5: Screenshot Beispielmodellierung 10_00 - Leitlinien Strategische Planung Absatzwirtschaft

im YAWL-Inspector erscheinen dessen modellspezifische Eigenschaften.
Für die folgenden Erläuterungen sollen nur noch die Bereiche (7), (4) und (6) eine Rolle spielen, alle anderen Bereiche werden nicht mehr von den Screenshots erfasst.
Das Einfügen von neuen Knoten geschieht über die rechten zwei Schalter in Bereich (7). Diese Schalter haben zwei verschiedene Modi. Ein einziger Klick auf den rechtesten Schalter aktiviert das Einfügen einer Bedingung; ist diese mit einem Klick in den YAWL-Editor (Bereich (4)), eingefügt so wird die Aktivierung rückgängig gemacht. Durch einen zweiten Klick auf den Schalter (vor dem Einfügen) wird eine Einrastfunktion aktiviert. Das heißt, dass ab sofort jeder Klick in den YAWL Editor ein Einfügen einer neuen Bedingung zur Folge hat. Gleiches gilt analog für den Aktionseinfügeschalter links neben dem soeben beschriebenen.
Abbildung 6 zeigt einen Screenshot nach dem Einfügen einer Aktion. Aktionen werden in iProcess allerdings ein wenig anders gezeichnet als aus der YAWL-Spezifikation von Kapitel 2 bekannt. Nach dem Einfügen wurde diese Aktion über den YAWL-Inspector ein wenig von den Standardvorgaben abgeändert, so dass diese ein AND-Vereinigungsverhalten (*join behaviour*), ein OR-Spaltverhalten (*split behaviour*) und einen neuen Namen „Beispiel Aktion“ bekam. Im YAWL Inspektor sind ebenfalls alle anderen modellspezifischen Informationen bezüglich dieser Aktion zu sehen.

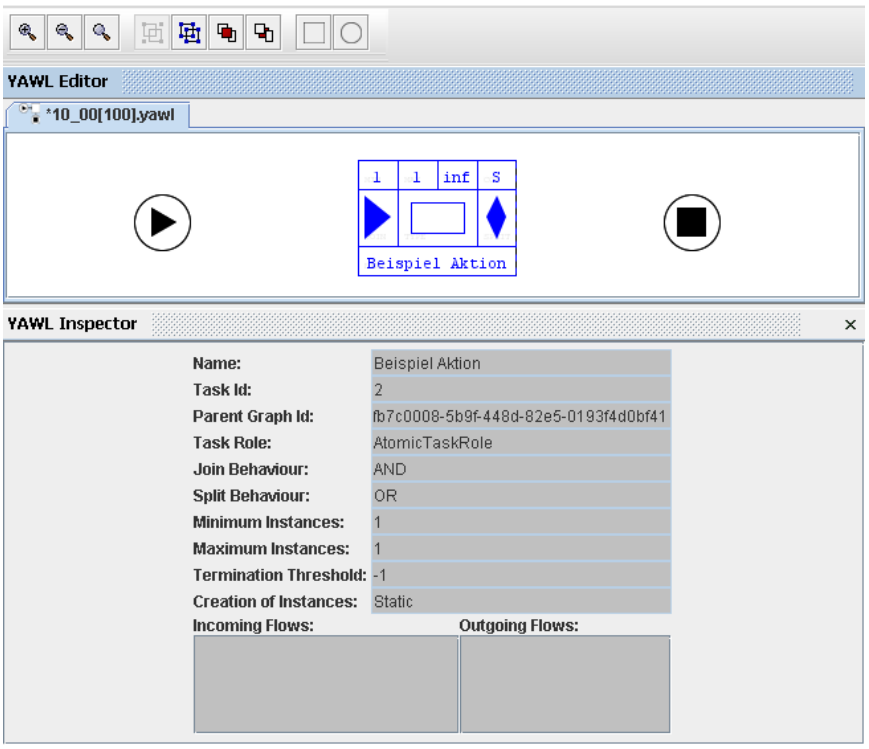

Abbildung 6: Screenshot nach dem Einfügen einer Aktion

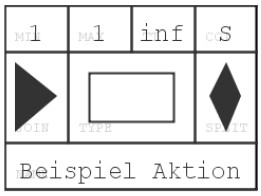

Abbildung 7: Vergrößerung der eingefügten Aktion

In Abbildung 7 ist eine Vergrößerung der soeben eingefügten Aktion zu erkennen. Die einzelnen Bereiche dieses Symbols können wie folgt interpretiert werden :
Die vier Werte der ersten Zeile $1, 1, inf, s$ stehen für die Multiplizitätseigenschaften von Aktionen. Von links nach rechts sind diese die minimale und maximale Anzahl zu generierender Instanzen beim Anlaufen der Aktion, die Abschlussschwelle (*termination threshold*), wieviele der generierten Instanzen abgeschlossen haben müssen, um die gesamte Aktion zu beenden und der Parameter, der angibt, ob weitere Instanzen nach der ursprünglichen Instanzgenerierung angelegt werden dürfen. Für die gezeigte Aktion gilt also: Es **muss** mindestens eine Aktion instanziert werden, es **darf** höchstens eine Aktion instanziert werden ($\Rightarrow$ keine multiplen Instanzen), es **müssen** alle Instanzen abgeschlossen haben, um die Aktion zu beenden, und weitere Instanzen **dürfen nicht** nach der ursprünglichen Instanzgenerierung erstellt werden. Die drei Piktogramme der zweiten Zeile machen sichtbar, dass es sich bei dieser Aktion um eine solche handelt, die ein AND-Vereinigungsverhalten aufweist, vom Typ eine atomare Aktion ohne multiple Instanzen ist und ein OR-Spaltverhalten inne hat. In der dritten und letzten

Zeile in der Aktionszeichnung ist der Aktionsname zu erkennen.
Bewegt man den Mauszeiger über einen der Knoten, so erscheint in dessen Mitte ein graues Quadrat und der Mauszeiger wird zu einer Hand. Ist dies der Fall, so kann eine Kante bzw. ein Fluss zwischen zwei verschiedenen Knoten gezeichnet werden.

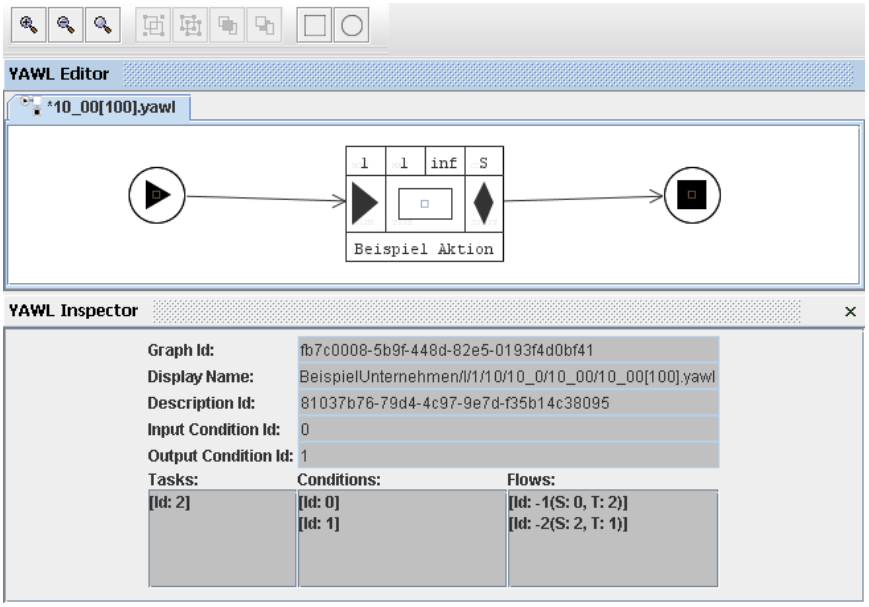

Abbildung 8: Screenshot nach dem Einfügen von Kanten

Abbildung 8 zeigt den Screenshot nach dem Einfügen von zwei Flüssen: ein Fluss von der Eingangsbedingung zur Beispielaktion und ein weiterer von der Beispielaktion zur Ausgangsbedingung. Im YAWL-Inspector sind alle modellspezifischen Daten dieses Graphen zu sehen: die Menge an Bedingungen, die speziellen Eingangs- und Ausgangsbedingungen und die Flüsse zwischen diesen Knoten.

3.2 YAWL-Model

Wie in Kapitel 3 beschrieben, ist iProcess eine Erweiterung des Modellierungsframeworks NeMo. Um eine gemeinsame Grundlage für alle in NeMo zu modellierenden Graphen zu schaffen, gibt es im Paket *nemo.graph.framework* verschiedene Klassen, die als Schnittstelle zur von NeMo verwendeten Graphbibliothek JGraph dienen.
Abbildung 9 zeigt die Ableitungshierarchie der umgesetzten Klassen des YAWL-Models in einem UML-Klassendiagramm. Weniger für das Verständnis wichtige Klassen wurden bewusst - der Übersichtlichkeit willen - nicht modelliert. Für die folgenden Ausführungen sei angemerkt, dass in allen Diagrammen nur die essenziellen Charakteristika des YAWL-Models beschrieben, nicht etwa auf JGraph spezifische Eigenheiten oder die Propagationen von Attributs- bzw. Modelländerungen eingegangen wird.

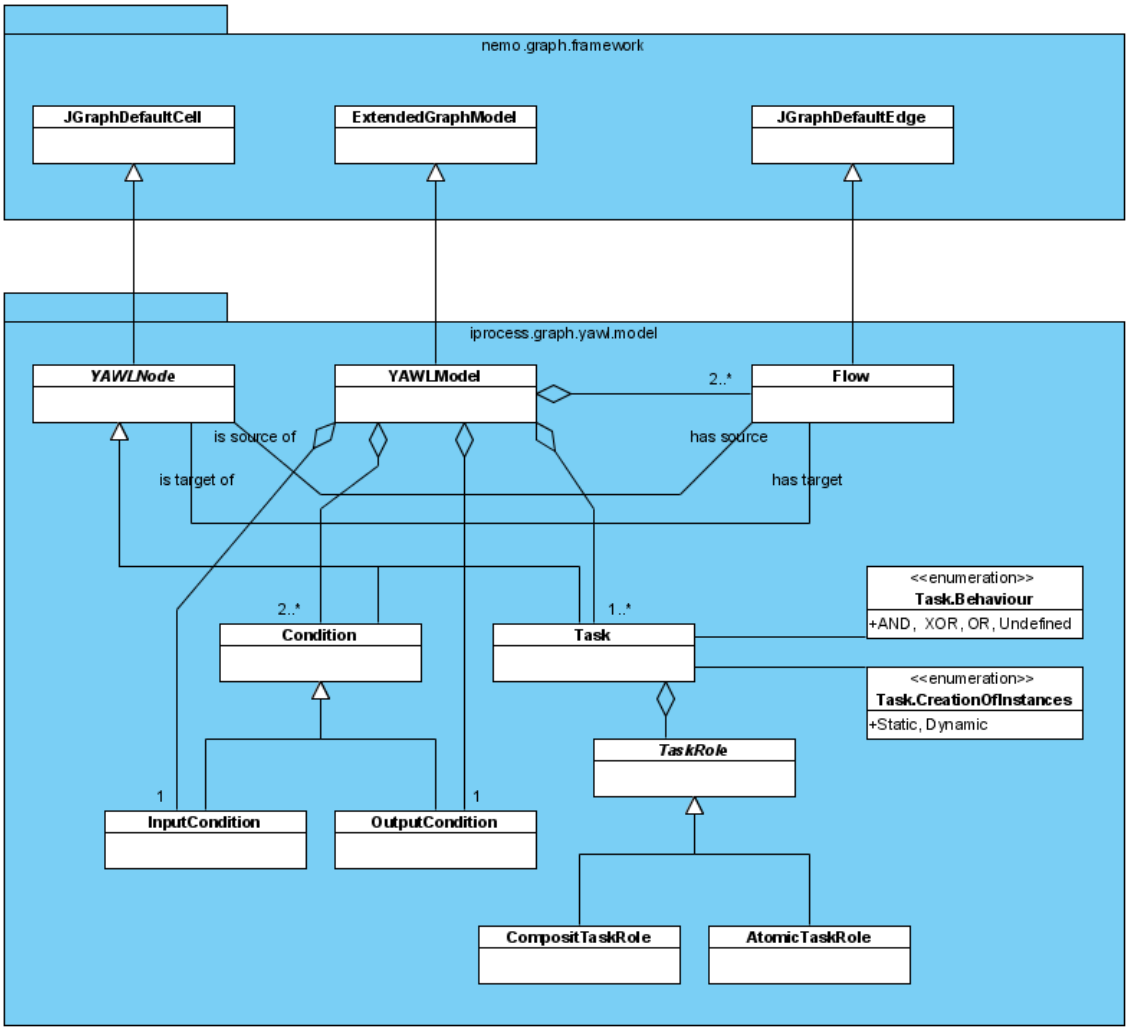

Abbildung 9: UML-Klassendiagramm zum iprocess.graph.yawl.model Paket

Wie üblich in Graphen, gibt es im YAWLModel Knoten und Kanten. Die abstrakte Klasse, YAWLNode, dient den beiden yawlspezifischen Knotentypen, Condition (Bedingung) und Task (Aktion) als Superklasse. YAWLNode ist gemäß der Erweiterungsmöglichkeiten von NeMo eine direkte Ableitung der Schnittstellenklasse JGraphDefaultCell. InputCondition (Eingangsbedingung) und OutputCondition (Ausgangsbedingung) sind bekanntlich spezielle-re Conditionen, die sowohl beim Rendern als auch bei Flüssen gesondert behandelt werden müssen. Daraus ergibt sich hier eine zusätzliche Hierarchiestufe.

Bei Tasks wird in einer etwas anderen Manier verfahren: Auch hier gibt es zwei verschiedene Möglichkeiten: Entweder ist ein Task eine atomare oder aber eine zusammengesetzte Aktion, beide müssen, analog zu den speziellen Bedingungen, anders gerendert werden; aus Modellsicht unterscheiden sich diese aber jedoch fast nicht[4]. Infolgedessen wird der Aktion eine Rolle zugewiesen, welche diese entweder als atomar oder zusammengesetzt auszeichnet. Diese Konstruktion erleichtert ein Umschalten bzw. Austauschen[5]. Aufzählungstypen für das

[4]Zusammengesetzte Aktionen verweisen zusätzlich noch auf in der Hierarchie tiefer liegende YAWL-Graphen.

[5]Man betrachte den Fall, dass eine bestimmte Aufgabe als atomar angesehen wurde, sie sich aber dennoch

Vereinigungs- bzw. Spaltverhalten und für die statische oder dynamische Instanzengenerierung werden in der Task-Klasse eingeführt. Flüsse gibt es in YAWL nur eine Sorte, deshalb ist hier eine mögliche weitere Hierarchiestufe, analog zum YAWLNodes, nicht nötig, so dass die Flussklasse Flow eine direkte Ableitung der NeMo-Schnittstellenklasse JGraphDefaultEdge sein kann. Flüsse sind gerichtet und haben somit sowohl einen Start- als auch einen Zielknoten. Als Schnittstelle für das YAWL-Model dient die NeMoklasse ExtendedGraphModel. Das YAWL-Model soll bekanntlich ein EWF-Netz darstellen und muss infolgedessen mindestens eine InputCondition und eine OutputCondition aggregieren. Da laut Spezifikation alle Knoten auf einem gerichteten Pfad von Eingangs- zur Ausgangsbedingung liegen müssen, ergibt sich daraus, dass ein Modell auch mindestens einen Task beinhalten muss, da keine Flüsse zwischen zwei Bedingungen erlaubt sind. Daraus wiederum entsteht die Notwendigkeit für mindestens zwei Flüsse, die eine Verbindung von Eingangs- über den notwendigen Knoten zur Ausgangsbedingung herstellen[6].

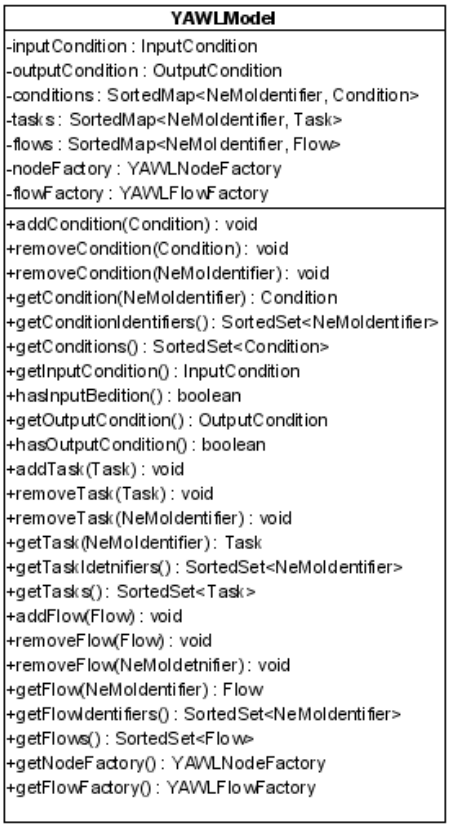

Abbildung 10: UML-Klassendiagramm zur Klasse iprocess.graph.yawl.model.YAWLModel

als etwas Komplexeres herausstellt, so kann man durch Wechseln der Rolle direkt ohne große Veränderung im Modell diese Aktion in einem neuen YAWL-Graphen verfeinern.

[6] Anmerkung: Es mag verwunderlich sein, dass in Abbildung 5 beim Graphgenerieren nur die Eingangs- und Ausgangsbedingungen standardmäßig eingefügt werden. Dieses wird dennoch so gehandhabt, da es sich bei diesen Knoten um zwei besonders wichtige und in einem Graph einzigartige Knoten handelt. Werden diese schon beim Anlegen eines neuen YAWL-Graphen eingefügt, so kann auf spezielle Einfügebuttons der GUI verzichtet werden, welches wiederum zur Benutzerfreundlichkeit von iProcess beiträgt.

Abbildung 10 zeigt die wesentlichen Attribute und Methoden des umgesetzten YAWLmodells. In einem YAWLModel-Object werden jeweils ein InputCondition- als auch OutputCondition-Objekt, jeweils eine Menge von Condition-Objekten (soeben genannte Input- und OutputCondition sind ebenfalls Teil dieser Menge), Task-Objekten und Flow-Objekten jeweils indiziert von deren einzigartigem NeMoIdentifier. Ferner stellt das YAWLModel ebenfalls sowohl eine YAWLNodeFactory als auch eine YAWLFlowFactory, zur Generierung von neuen Knoten bzw. Kanten zur Verfügung, um zu garantieren, dass alle Knoten und Kanten in einem Modell einen einzigartigen NeMoIdentifier zugewiesen bekommen.
Um Zugriff auf die vom Modell gespeicherten Daten zu bekommen, werden entsprechende Methoden angeboten. Diese sollen beispielhaft für Bedingungen (conditions) erläutert werden, funktionieren aber analog bei Aktionen (tasks) und Flüssen (flows). Die gebotenen Funktionen können in zwei Bereiche eingeteilt werden. Der erste Bereich ist zuständig für die Manipulation des Modells und beinhaltet dementsprechend die Methoden: (1) addCondition(Condition): void[7], (2) removeCondition(Condition):void und (3) removeCondition(NeMoIdentifier). Erstere Funktion erlaubt das Einfügen von neuen Bedingungen in das Modell. Um die Eindeutigkeit der Bezeichner von Knoten im Modell zu wahren, sollten neu einzufügende Knoten über die dem Modell zugehörige YAWLNodeFactory generiert werden. Methoden zwei und drei dienen dem Löschen von Bedingungen. Dieses kann entweder durch Angabe der tatsächlichen Bedingung oder aber deren Bezeichner geschehen. Der zweite Bereich ist zuständig für das Auslesen von Modellinhalten. Dieser stellt die Methoden: (1) getCondition(NeMoIdentifier): Condition, (2) getConditionIdentifiers(): SortedSet<NeMoIdentifier> und (3) getConditions(): SortedSet<Condition> bereit. Methode (1) gibt die mit ihrem Bezeichner indizierte Bedingung zurück. Funktion (2) liefert alle Bezeichner der im Modell enthaltenen Bedingungen und (3) liefert diese als Menge. Wie schon erwähnt, gibt es für Aktionen und Flüsse entsprechend analoge Methoden. Des weiteren gibt es jeweils zwei Methoden für die Eingangs- und Ausgangsbedingung. Eine um festzustellen, ob das Modell schon eine dieser besitzt (Bsp. hasInputCondition(): boolean) und eine, um diese direkt zu referenzieren (Bsp. getInputCondition(): InputCondition). Schlussendlich gibt es noch Methoden, um Zugriff auf die dem Modell zugeordneten Factories zu erlangen: getNodeFactory(): YAWLNodeFactory und getFlowFactory(): YAWLFlowFactory.

[7] Besonderheit: Bei Bedingungen wird überprüft, ob es sich um eine Eingangs- oder Ausgangsbedingung handelt und dementsprechend dem Modell als solche zugewiesen. Da ein Modell zusätzlich nur jeweils eine der genannten Bedingungen beinhalten darf, wird vorher überprüft, ob nicht schon eine derartige vorhanden ist.

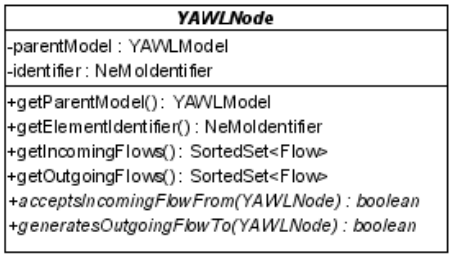

Abbildung 11: UML-Klassendiagramm zur Klasse iprocess.graph.yawl.model.YAWLNode

In Abbildung 11 sind die Gemeinsamkeiten aller Knoten im YAWLModel ersichtlich. Knoten gehören zu einem „Elternmodell“(YAWLModel) und haben einen eindeutigen Bezeichner (NeMoIdentifier). Dementsprechend werden auch Methoden bereitgestellt, mit denen man diese Eigenschaften einsehen kann: getParentModel(): YAWLModel und getElementIdentifier(): NeMoIdentifier. Sowohl eingehende Flüsse zum Knoten als auch ausgehende Flüsse vom Knoten können über die Methoden: getIncomingFlows(): SortedSet<Flow> und getOutgoingFlows(): SortedSet<Flow> referenziert werden. Die beiden abstrakten Funktionen acceptsIncomingFlowsFrom(YAWLNode): boolean und generatesOutgoingFlowsTo(YAWLNode): boolean müssen in den abgeleiteten Klassen überschrieben werden, um bestimmen zu können, ob diese, in Abhängigkeit der angegebenen Knoten, eingehende Flüsse akzeptieren bzw. ausgehende Flüsse generieren können.

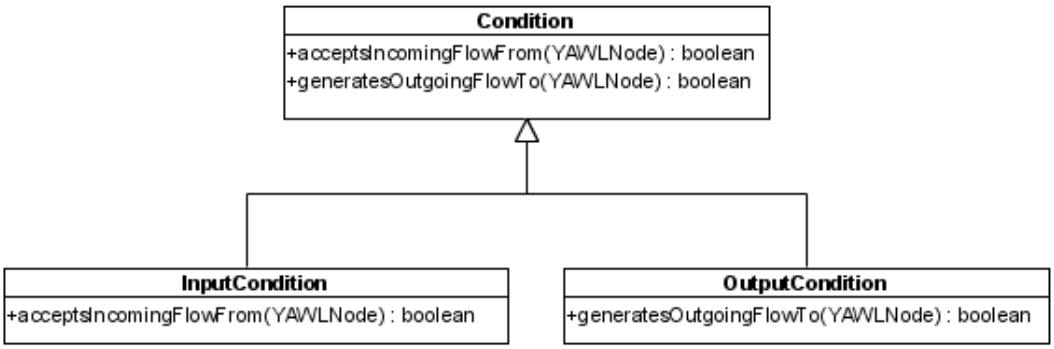

Abbildung 12: UML-Klassendiagramm zur Klasse iprocess.graph.yawl.model.Condition und dessen Ableitungen

Conditions und deren speziellere Ausführungen Input- bzw OutputConditions speichern keine weitere Information als die von YAWLNode vorgegebenen. Es müssen also nur die genannten abstrakten Funktionen aus Abbildung 11 überschrieben werden. Allgemein gilt für Bedingungen, dass sie nur eingehende / ausgehende Flüsse haben können, die nicht von / zu anderen Bedingungen kommen / gehen. Im spezielleren Fall der Eingangsbedingung kann diese keine eingehenden Flüsse besitzen, so muss die allgemeinere Methode der Superklasse entsprechend angepasst werden. Analoges gilt für ausgehende Flüsse bei Ausgangsbedingungen.

Abbildung 13: UML-Klassendiagramm zur Klasse iprocess.graph.yawl.model.Task

Abbildung 13 zeigt die Klasseninterna einer Aktion. Diese implementiert, analog zur Bedingung, die Methoden, die feststellen, ob von / zu bestimmten Knoten eingehende / ausgehende Flüsse existieren können. Auf diese folgen eine Reihe von statischen Klassenmethoden, die in einer Map unter einem bestimmten Key einen Wert ablegen und diesen dann auch wieder aus einer solchen Map extrahieren können. Dies hängt damit zusammen, dass diese Werte zum einen veränderbar sind, aber viel mehr noch das Rendering eines Tasks beeinflussen. Um die Renderer über eventuelle Änderungen zu informieren, werden solche von JGraph genannten AttributeMaps überprüft. Um Annullierungs Patterns zu unterstützen, muss eine Aktion ebenfalls wissen, von welchen Knoten die vorhandenen Token gelöscht werden müssen. Infolgedessen werden Knoten in einem SortedSet<YAWLNode> gespeichert. Diese Menge kann über

die fünf folgenden Methoden geändert bzw. erhalten werden[8]. Da das von NeMo verwendete JGraph als „transaktionsbasiert“ bezeichnet werden kann, wird der Methode setAttributeChanges(Map): void eine besondere Aufgabe zugeschrieben: Die Methode kann alle Änderungen in einer, von den soeben genannten statischen Methoden gefüllten, Map auf einmal durchführen. Das würde beim Ändern von z.B. fünf Werten nur ein Undo- / Redo Schritt erzeugen, anstatt fünf für jede einzelne Änderung. Alle nachfolgenden Methoden können benutzt werden, um Aktionsattribute auszulesen oder diese einzeln zu ändern (man beachte, dass jede einzelne Änderung von einer dieser set-Methoden einen Undo- / Redo Schritt generiert).

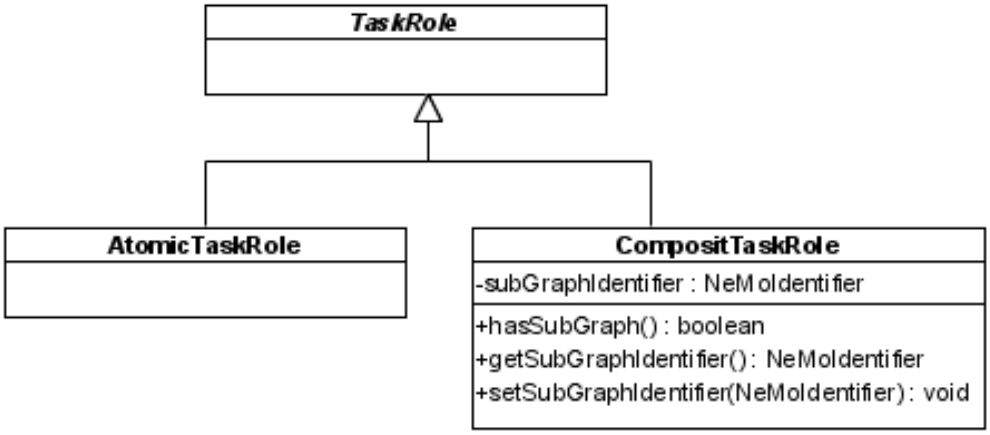

Abbildung 14: UML-Klassendiagramm zur Klasse iprocess.graph.yawl.model.TaskRole und dessen Ableitungen

Abbildung 14 zeigt die Hierarchie der verschiedenen Aktionsrollen. Es gibt eine abstrakte Oberklasse und die jeweils möglichen Rollen atomar (atomic) oder zusammengesetzt (composit). Auffallend ist, dass Superklasse und atomare Rolle ohne wirklichen Inhalt sind. Eine zusammengesetzte Rolle bedeutet hingegen, dass von der dazugehörigen Aktion ein Subgraph referenziert wird. Der eindeutige Bezeichner dieses Subgraphen wird in der CompositTaskRole gespeichert. Motivation für ein solches Vorgehen: Alle Aktionen können von einem eventuellen Simulator gleich behandelt werden; trifft dieser jedoch auf eine zusammengesetzte Aktion, kann festgestellt werden, welcher Subgraph durch diese symbolisiert und somit verfeinert wird, dort kann dementsprechend mit der Simulation fortgefahren werden.

3.3 YAWL-View

Verschiedene Sichten (*views*) auf Modelle bieten in iProcess derzeit Bereiche (3),(4) und (6), bekannt aus Abbildung 3. Der Company Profile Explorer (Bereich (3)) bildet eine Sicht auf

[8]Da es sich in dieser Studienarbeit nur um den Grundstein für ein workflow management system handelt (nicht aber um ein komplettes) wurde die Beeinflussung dieses cancellation sets durch die GUI noch nicht implementiert.

eine Arbeitsablaufspezifikation mit dem Aufgabenrahmenplan. Jeder Knoten dieses Plans repräsentiert einen YAWL-Graphen, also ein EWF-Netz aus Sicht der YAWL-Spezifikation. Die Ausführungen aus Kapitel 2 weisen auf eine baumartige Struktur einer Workflowspezifikation hin, welche durch die Eltern-Kind-Beziehungen dieser Knoten sichtbar wird.
Aufgabe des YAWL-Editors (Bereich (4)) als View ist das Visualisieren des darunterliegenden YAWLModels. Da dieser View die wichtigste Rolle für die graphische Modellierung spielt, soll dieser auch etwas ausführlicher beschrieben werden. In Abbildung 15 sind die für das Zeichnen verantwortlichen Klassen zu erkennen.

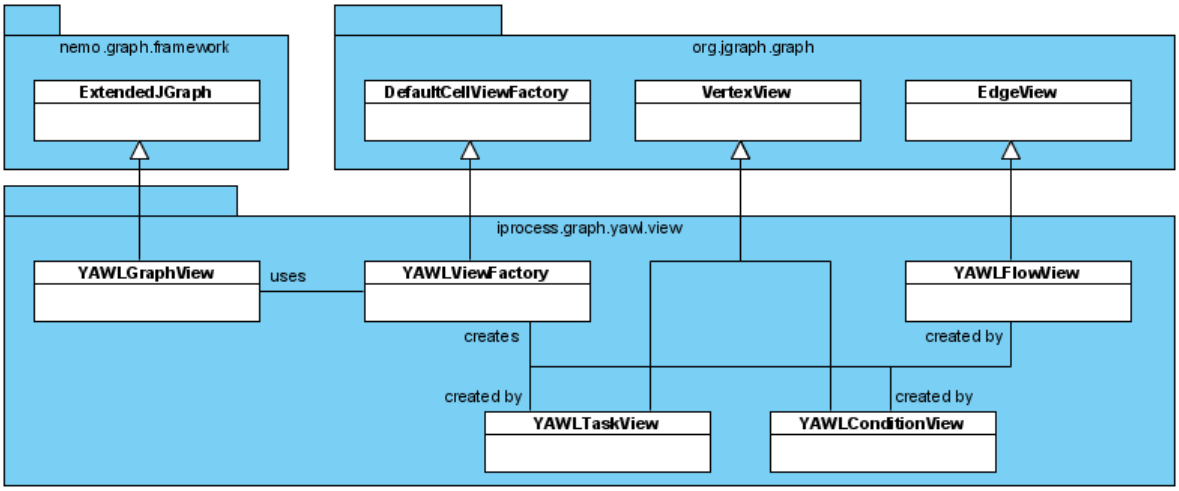

Abbildung 15: UML-Klassendiagramm zum iprocess.graph.yawl.view Paket

In Kapitel 3.4 wird YAWLEditor als die wichtigste Controller-Klasse bezeichnet werden. Diese generiert einen YAWLGraphView, eine Spezialisierung eines JGraphs, der die verschiedenen Knoten und Kanten des YAWL-Models visualisiert. Ein JGraph besitzt einen sogenannten GraphLayoutCache, der mittels einer ViewFactory entscheidet, wie verschiedene Knoten- und Kantentypen gezeichnet werden. Im Falle des YAWLGraphViews übernimmt die Rolle der ViewFactory die speziellere YAWLViewFactory, welche je nach Art des Knotens oder der Kante einen speziellen View generiert. Für Aktionen wird ein YAWLTaskView, für Bedingungen ein YAWLConditionView und für Flüsse ein YAWLFlowView angelegt. Diese Views besitzen intern einen ViewRenderer, welcher dann schlussendlich anhand der jeweiligen Eigenschaften des Knotens bzw. der Kante entscheidet, was, wie und wo zu zeichnen ist. Zu diesem Zweck generieren sie ein JComponent dessen paint()-Methode dementspechend angepasst wird.
Dritter und letzter derzeit implementierter View ist mit Bereich (6) der YAWL-Inspector. Dieser zeigt, wie aus Kapitel 3.1 bekannt, aktuell selektierte Objekte mit Hilfe textueller Mittel an. Für nähere Beschreibungen dieses Views, sowie für den Company Profile Explorer, sei auf den Quellcode im Paket iprocess.gui.components verwiesen.

3.4 YAWL-Controller

Die Controller für das YAWL-Model sind alle bekannt durch die Ausführungen in Kapitel 3.1. Daher wird in diesem Kapitel auch nur zusammenfassend berichtet. Derzeit gibt es vier Elemente in IProcess, die eine solche Funktion verkörpern: Bereiche (3), (4), (6) und (7) aus Abbildungen 3 und 5. Mit Hilfe des Company Profile Explorers (3) (Klassen im Paket: iprocess.gui.components.cpexplorer) können neue YAWLGraphen angelegt werden und diese in einer Hierarchie angeordnet werden gemäß der Definition einer Workflowspezifikation bekannt aus Kapitel 2.
Durch Mausaktionen im Bereich (4) dem YAWL-Editor (in Kombination mit Bereich (7)) können Knotenpositionen verändert, dem Modell weitere Knoten und Kanten hinzugefügt, oder diese durch das Selektieren und Drücken von Entfernen aus dem Modell gelöscht werden. Dieser Controller spielt die wichtigste Rolle für die graphische Modellierung und soll dementspechend etwas ausführlicher beschrieben werden. In Abbildung 16 ist unter anderem die wichtigste Controller-Klasse YAWLEditor zu sehen.

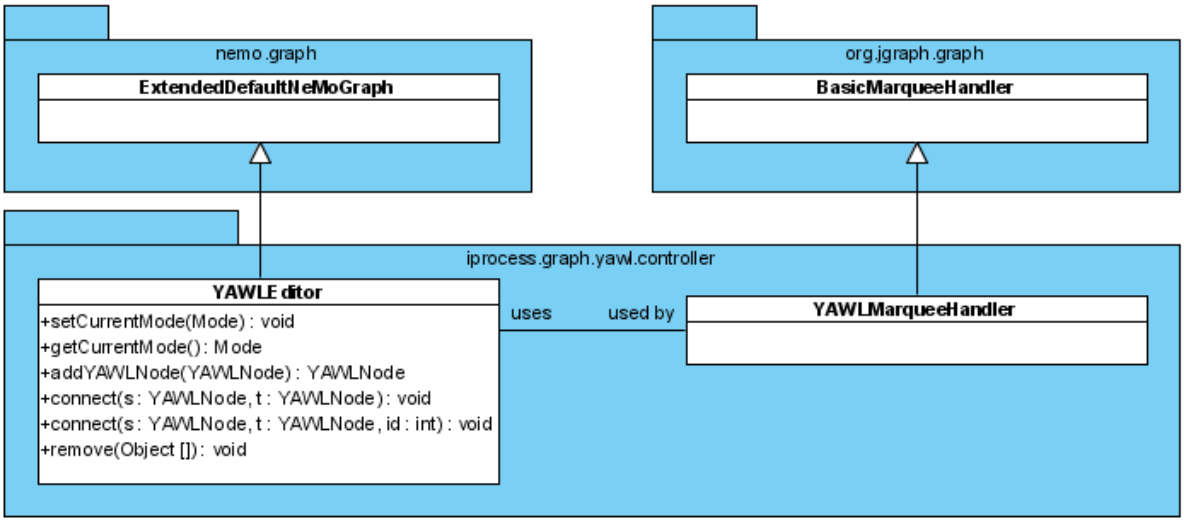

Abbildung 16: UML-Klassendiagramm zum iprocess.graph.yawl.controller Paket

Die Methoden setCurrentMode() und getCurrentMode() werden von den Modi-Schaltern in Contoller-Bereich (7) benötigt, um den YAWL-Editor in einen bestimmten Editiermodus zu setzen. Editiermodi existieren zur Zeit sechs: Selektions-, Aktionseinfüge-, Multipleaktioneneinfüge-, Bedingungseinfüge-, Multiplebedingungeneinfüge- und Fluss-Modus. Diese werden allerdings - der Benutzerfreundlichkeitwillen - nur von zwei Schaltern beeinflusst. Ist von diesen zwei Schaltern keiner selektiert, so befindet sich der Editor im Selektionsmodus. Wird der Aktionseinfügeschalter einmal betätigt, so wechselt man in den Aktionseinfüge-Modus. Betätigt der Modellierer diesen ein zweites Mal, so befindet sich der Editor im Multipleaktioneneinfüge-Modus. Ein dritter Klick ohne Mausklick im Editor selbst wechselt wieder in den Selektionsmodus. Analoges gilt natürlich für den Bedingungseinfügeschalter. Sollen z.B. zwei Bedingungen hinzugefügt werden, so klickt der

Modellierer zwei Mal auf den Bedingungseinfügeschalter und versetzt den Editor in den Multiplebedingungeneinfüge-Modus. Erfolgt nun ein Mausklick im Editor selbst, so erscheint eine Bedingung. Ein zweiter Klick generiert eine weitere Bedingung. Klickt der Modellierer nun nochmals auf den Bedingungseinfügeschalter, so schaltet der Editor zurück in den Selektiermodus. Durch Klicks in den Editor in einem der bestimmten Eifübe-Modi wird die Methode addYAWLNode() ausgeführt und ein neuer Knoten in das Modell eingefügt. Das Modell selbst unterscheidet, ob es sich um eine Aktion oder Bedingung handelt und sortiert diese dann dementsprechend. Die beiden Methoden connect() fügen einen neuen Fluss dem Modell zu. Bewegt sich der Mauscurser über einen Knoten, so erscheint in dessen Mitte ein sogenannter Port und der Mauszeiger wird zu einer Hand. Mit dem Drücken der Maus auf einen solchen Port beginnt der mit dem Editor verbundene YAWLMarqueeHandler das Warten auf das Loslassen des Mausbuttons. Wird diese auf einem Port eines anderen Knoten losgelassen, so entsteht über die erste connect() Methode die Verbindung und der Fluss im Modell. Leztere connect() Methode hingegen findet Verwendung beim Einfügen von zuvor gespeicherten Kanten, die schon einen Identifier besitzen. Eine letzte für den Controllerteil wichtige Funktion remove(Object[]) löscht schlussendlich alle Knoten und Kanten aus dem Modell, die im Parameter spezifiziert werden.
Der YAWL-Inspector (Bereich (6)) bietet momentan die Möglichkeit, bei ausgewähltem Task dessen Multiplizitätseigenschaften, Split- und Joinverhalten als auch dessen Namen zu verändern. Näheres zum Inspector kann im iprocess.gui.components.yawlinspector Paket eingesehen werden.

4 Ausblick

Mit der Implementierung der durch die Modellierungssprache YAWL gegebenen Kontrollflussperspektive können in iProcess, zum Zeitpunkt der Fertigstellung dieser Studienarbeit, Arbeitsabläufe spezifiziert werden. Modelliert werden können Aktionen und deren Abarbeitungsreihenfolge, die durch die Konstrukte für Folgen (*sequence*), Wahlmöglichkeiten (*choice*), Parallelität (*parallelism*) und Synchronisation (*synchronisation*) verändert werden können. Mit YAWL verwendet iProcess eine übersichtliche graphische Sprache, mit deren Hilfe komplexe Geschäftsprozesse und deren Abarbeitungsreihenfolge veranschaulicht werden können und somit einem Geschäftsprozessmanagement als Werkzeug zum Identifizieren und Dokumentieren unternehmenseigener Prozesse dienen kann. Als formale Sprache eignet sich YAWL ebenfalls als optimale Grundlage für Simulationen. Es gibt eine vordefinierte Workflowspezifikation: den Aufgabenrahmenplan[9], der für die meisten der zu modellierenden Unternehmen Anwendung finden kann. Er zeigt einen Vorschlag für eine mögliche Kategorisierung bzw.

[9]Der implementierte Aufgabenrahmenplan ist keineswegs vollständig und bietet daher nur einen kleinen Ausschnitt des kompletten Unternehmensumfelds. Der implementierte Ausschnitt soll lediglich als Beispiel für die komplette Arbeitsablaufspezifikation stehen.

Verfeinerung der Tätigkeiten der einzelnen Bereiche, die zu modellieren sind, ähnlich dem Kontenrahmen der Buchhaltung eines Unternehmens.
Es steht nun aus, das Grundgerüst bzw. die Kontrollflusssicht von iProcess um Daten-, Ressourcen- und operationale Perspektive zu erweitern, um aus diesem ein vollwertiges workflow management system zu erschaffen. In der Datensicht müssen Strukturen geschaffen werden, die es ermöglichen, für Aktionen diejenigen Daten bzw. Dokumenttypen zu definieren, die von solchen be- bzw. verarbeitet werden sollen. Da diese Daten höchstwahrscheinlich aus verschiedensten, zur Zeit der Implementierung des Systems unbekannten, Informationen bestehen werden, muss ein geeignetes Modellierungskonzept gefunden werden, Datentypen dynamisch zu generieren und einzelnen Prozessen zuzuweisen. Da Datentypen häufig mehrmals Verwendung finden, wäre es von Vorteil, generierte Datentypen in einer Art Datentypenbibliothek vorzuhalten, welche dann von den jeweiligen Prozessen instanziert werden können. Dementspechend müssen sowohl das Modell als auch der YAWL-Inspektor um die Möglichkeit der Dokumentassozierung erweitert werden. Im YAWL-Inspektor könnte es z.B. einen Bereich bei Aktionen geben, mit dem man von der Datentypenbibliothek einen Typ per Drag-And-Drop dieser zuweist. Viele Aktionen bedürfen der Abarbeitung unter Einhaltung von verschiedenen Normen. Analog zu einer Datentypenbibliothek sollte ein Normenindex geschaffen werden, der diese Normen bündelt und aus dem gegebenenfalls einzelne Normen von Aktionen referenziert werden können.
Die instanzierten Daten sollen nach Möglichkeit auch gespeichert werden. In den meisten Unternehmen werden eine Vielzahl solcher Daten und Dokumente von einer größeren Anzahl an Benutzern be- bzw. verarbeitet. Dementsprechend könnte es von Vorteil sein, das Persistieren dieser riesigen Datenmengen nicht etwa wie bisher auf einem Filesystem, sondern in einer dafür vorgesehenen Datenbank anzustreben. Nicht allein der Zugiff auf diese Informationen würde gegebenenfalls beschleunigt, sondern ein Mehrbenutzerbetrieb, bei dem mehrere Modellierer gleichzeitig verschiedene Prozesse bearbeiten, würde erleichtert. Durch eine Datenbankanbindung entsteht natürlich der Bedarf für eine Benutzerverwaltung, die einen Mehrbenutzerbetrieb des Systems überhaupt erst ermöglichen kann. Es soll mit diesem System nicht nur modelliert und optimiert werden, sondern auch eventuellen Mitarbeitern eines Unternehmens ermöglicht werden, die zu leistenden Arbeitsschritte zu verstehen bzw. als Arbeitsanleitung (-anweisung) dienen. Diesen Mitarbeitern hingegen sollte jedoch nur ein Lesezugriff auf die Modelle gewährt werden, da das Verändern von Prozessmodellen von solchen Nichtmodellierern keinesfalls wünschenswert ist.
Da das System in Unternehmen Verwendung finden soll, in denen es verschiedene Nutzer mit unterschiedlichen Rollen geben wird, könnte eine integrierte Modellierung der Unternehmens- bzw. Organisationsstruktur wünschenswert sein. Es sollte dementsprechend eine Organisationsmodellierungssprache gefunden werden, die Hierarchien und Stellungen einzelner Personen bzw. Personengruppen untereinander und deren Rollen im Unternehmen definiert. Diese können dann eine Ressourcen-Perspektive für das Prozessmodell ergänzen, indem man Ver-

bindungen zwischen den Rollen der einzelnen Personen und den zur Abarbeitung einer Aktion nötigen, als Produktionsmittel anzusehenden, Menschen erstellt.
Die Ausführungen des letzten Abschnitts zeigen, dass das Implementierte keinesfalls ein vollständiges Arbeitsablaufsmanagementsystem ist, wohl aber als Grundgerüst für ein solches dienen kann. Wie in jeder Software gilt es, eventuelle Fehler zu beheben, den Funktionsumfang zu erhöhen und den Bedienungskomfort zu steigern.

Literatur

[1] Workflow patterns website. http://is.tm.tue.nl/research/patterns/.

[2] H. Ruchti. Bilanz und investition. *Der Industriebetrieb und sein Rechnungswesen (Festschrift für M.R. Lehmann)*, page 36, 1956.

[3] W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers, 8(1)*, pages 21–66, 1998.

[4] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: Yet another workflow language. http://www.yawl.fit.qut.edu.au/yawldocs/yawlrevtech.pdf, 2005.

[5] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, 2002.