

Bachelor Thesis

Execution Perspective for Ad-Hoc Business Processes

Lutz Gericke

Betreuer

Prof. Dr. Mathias Weske, Hasso-Plattner-Institute, Potsdam, Germany

M.Sc. Gero Decker, Hasso-Plattner-Institute, Potsdam, Germany

M.Sc. Harald Schubert, SAP AG, Walldorf, Germany

30. Juni 2008

Abstract

Workflow Management Systems are becoming increasingly important for planning and executing business operations. Unfortunately, the existing solutions offer only little flexibility in execution and process design, especially when handling processes with a minor degree of automation. Within the scope of the bachelor project 'Ad-Hoc Business Processes' at the Business Process Technology chair at the Hasso-Plattner-Institut Potsdam in cooperation with SAP AG Walldorf, several prototypes were developed to show the elaborated concepts.

This paper deals with the Oryx platform of the Hasso-Plattner-Institut. It documents, which changes on the existing components had to be done in order to support flexible workflows. Before, some essential concepts have to be clarified to show selected problems and their solutions on the implementation layer.

Zusammenfassung

Workflowmanagementsysteme werden immer bedeutender bei der Planung und Ausführung von Arbeitsabläufen in Unternehmen. Leider bieten die existierenden Lösungen gerade für Prozesse mit einem geringen Automatisierungsgrad und großer Beteiligung von menschlichen Ressourcen eine zu geringe Flexibilität, sowohl bei der Ausführung als auch beim Prozessdesign. Im Rahmen des Bachelorprojekts 'Ad-Hoc Business Processes' am Lehrstuhl Business Process Technology des Hasso-Plattner-Instituts Potsdam in Kooperation mit der SAP AG Walldorf entstanden mehrere Prototypen, die die ausgearbeiteten Konzepte verdeutlichen.

Diese Arbeit beschäftigt sich mit Oryx, dem Prototyp am Hasso-Plattner-Institut. Dabei soll dokumentiert werden, welche Änderungen an den bestehenden Komponenten vorgenommen wurden, um flexible Workflows zu unterstützen. Zuvor müssen allerdings grundlegende Begriffe geklärt werden, um dann auf der Implementierungsebene ausgewählte Probleme und ihre Lösung zu erläutern.

Inhaltsverzeichnis

1	Einführung	1
2	Grundlegende Begriffe	3
2.1	Was ist Ad-Hoc?	3
2.2	Führendes Szenario "Einstellung eines neuen Mitarbeiters"	5
2.3	Abgeleitete Anforderungen	6
2.4	Gefärbte Petrinetze mit Guard Conditions	7
2.5	Umwandlung von BPMN in Petrinetze	8
3	Entwicklungen an der Oryx-Plattform	11
3.1	Existierende Prototypen	12
3.2	Verwendete Technologien	13
3.3	Integration der Komponenten	14
3.4	Entwicklungsmethodik	15
4	Implementierungen	17
4.1	Schnittstellendefinition	17
4.2	Oryx	20
4.3	Transformationskomponente BPMN2PN	22
4.4	Petrinetzengine	25
4.5	Worklist	30

5	Analyse und Monitoring	35
5.1	Integrierte Analysen	35
5.2	Weitergehende Analysen	37
6	Zusammenfassung	41
	Literaturverzeichnis	43

1. Einführung

Heutige Business Process Management Systeme sind sehr gut dafür geeignet, Geschäftsprozesse in Unternehmen zu visualisieren und zur Ausführung zu bringen. Zuständigkeiten werden klar aufgeteilt und Abläufe definiert, um letztlich effektiver zu wirtschaften und die Prozesse im Unternehmen effizienter zu gestalten [Wes07]. Diese Art der Softwareunterstützung ist gerade für Abläufe mit einem hohen Grad an Automatisierung sehr zweckmäßig. Problematisch kann es allerdings werden, wenn viele Nutzer an und mit dem System arbeiten. Es kommt vor, dass Computersysteme umgangen werden. Probleme werden teilweise lieber direkt am Schreibtisch besprochen und sind somit nicht in der Software nachvollziehbar. Hier muss die zu entwickelnde Lösung ansetzen und Anreize schaffen, die Abläufe im System zu dokumentieren. So sind einfache Möglichkeiten zur Zusammenarbeit von Prozessbearbeitern von Nöten, um spontanes Verhalten durch Software abzubilden.

Diese Arbeit ist im Rahmen des Bachelorprojekts "Ad-Hoc Business Processes" am Lehrstuhl Business Process Technology des Hasso-Plattner Instituts in Potsdam entstanden. Industriepartner war die SAP AG in Walldorf, die es uns ermöglichte, am Prototypen des SAP Netweaver Business Process Management zu entwickeln. Konzepte und Methoden, die Ad-Hoc-Änderungen an Prozessen erlauben, wurden sowohl theoretisch als auch praktisch untersucht. Großes Augenmerk wurde auf die grundlegenden Konzepte gelegt und diese an zwei Prototypen evaluiert. Beim SAP-Prototypen wurden vor allem Modellierungsaspekte beleuchtet, wobei bei der Oryx-Plattform [DGKW08] die Laufzeitaspekte im Vordergrund standen.

Im Rahmen der Arbeiten an Oryx, sind vier Bachelorarbeiten entstanden. Mike Nagora hat sich in [Nag08] mit Nutzerkonzepten wie Rollen und Delegation beschäftigt. In der Bachelorarbeit von Philipp Maschke [Mas08] wird das Thema der Reevaluierung beleuchtet. Alexander Koglin widmet sich in [Kog08] Konzepten zu Benutzeroberflächen mittels XForms, wie sie in der Oryx-Plattform umgesetzt sind.

Die Implementierungen an SAP Galaxy sind das Thema der Arbeit von Matthias Kleine [Kle08]. Die Arbeit von Stefan Krumnow [Kru08] schlägt dann die Brücke zwischen den Entwicklungen an SAP Galaxy und der Oryx-Plattform.

Diese Arbeit beschäftigt sich mit den Modifikationen, die an der Oryx-Plattform gemacht wurden, den nötigen Integrationsschritten und den Erweiterungsmöglichkeiten, die die Architektur bietet. Idee dabei ist, den Weg eines Prozessmodells durch alle Stufen einer Business Process Management Lösung zu verfolgen und punktuell in die Tiefe zu gehen.

Zuerst werden in Kapitel 2 grundlegende Begriffe, die im Rahmen dieser Arbeit wichtig sind, erklärt. Dementsprechend soll vor allem der Begriff *Adhocness* im Rahmen dieses Projekts geklärt werden. Hierzu müssen Annahmen darüber getroffen werden, welche Anwendungsfälle unterstützt werden sollen und welche generellen Aspekte des Business Process Managements zu beleuchten sind.

Anschließend wird in Kapitel 3 die Herangehensweise an das Projekt gezeigt, welche Softwarelandschaft zu Projektbeginn existierte und wie die Komponenten miteinander integriert wurden. In Kapitel 4 sollen konkrete Implementierungen aufgezeigt werden. Dabei wird auf Definitionen und Realisierungen von Schnittstellen eingegangen. Weiterhin soll jede Komponente im Detail beleuchtet werden. Die Ansätze, die im Bereich der Analyse und des Monitorings umgesetzt wurden, werden in Kapitel 5 thematisiert. Eine Zusammenfassung in Kapitel 6 schließt diese Arbeit.

2. Grundlegende Begriffe

Der konkreten Arbeit an den Prototypen war eine lange Phase der Ideen- und Begriffsfindung vorangestellt. Hier wurde geklärt, welche Arten von Adhocness es gibt und in welchen Use-Cases sich diese Erkenntnisse nutzen ließen. Die Ergebnisse dieser Planungsphase sind in diesem Kapitel beschrieben.

Die beschreibenden Szenarien wurden zu einem großen Teil von Seiten der SAP AG geprägt und durch die Projektgruppe ausgearbeitet. Es finden sich typische Anforderungen eines Unternehmens wieder, dessen Software täglich Millionen von Nutzer unterstützt. Hierbei treten Probleme auf, die es zu erkennen und durch geeignete Maßnahmen in Software abzubilden gilt.

2.1 Was ist Ad-Hoc?

Der Mensch – um den es bei den von uns betrachteten Unternehmensprozessen zentral geht – besitzt eine nicht zu unterschätzende spontane Komponente. Vielerlei Vorgänge funktionieren in der Praxis so unterschiedlich, dass sie kaum durch eine Business Process Management Lösung abgedeckt werden können. An dieser Stelle soll der Entwicklung von Softwarelösungen für Geschäftsprozessmodelle vorangetrieben werden und bewusst Freiräume sowohl in der Modellierung als auch in der Ausführung geschaffen werden.

Die Wissenschaft hat dieses Problem ebenso erkannt und sich der Thematik gewidmet. So wurden mehrere Arten von Veränderbarkeit in Workflows identifiziert. Folgt man [SSO01], so sind es genau 3:

- Dynamik
- Adaptivität
- Flexibilität

Diese Klassifizierungen sollen in erster Linie helfen, ein Maß für die Dimensionen der Änderungen von Workflows zu finden. Dabei ist die Unterscheidung von Modell und Instanz von immanenter Bedeutung. Ein Modell beschreibt die Definition von – um im Wortschatz der BPMN¹ zu bleiben – Aktivitäten, Sequenzflüssen und Datenobjekten usw., wogegen die Instanz eine konkrete Ausprägung dieses Modells ist.

Dynamik beschreibt die Fähigkeit eines Prozess(-modells), an die Änderungen im Geschäftsprozess anpassbar zu sein. Dies bewirkt, dass Modelle zur Laufzeit abgeändert werden können. Hier stellt sich die Frage nach bereits laufenden Instanzen, die möglicherweise in dieses geänderte Modell migriert werden müssten.

Adaptivität bezeichnet die Fähigkeit von Workflows, auf Ausnahmen zu reagieren. Dabei ist es nahezu unmöglich auf Ausnahmen zu reagieren, die nicht vorhersehbar sind. Um zu vermeiden, dass in unvorhergesehenen Situationen 'um das System herum' gearbeitet wird, sollen in gewissen Maßen Ausnahmen antizipiert und geeignet behandelt werden, indem Möglichkeiten geschaffen werden, auf Ausnahmen mit dem System zu reagieren.

Flexibilität bezeichnet die Fähigkeit eines Workflows, auf Grundlage eines möglicherweise nur teilweise spezifizierten Modells zu laufen. Die endgültige Spezifikation des Prozesses passiert hier erst zur Laufzeit und kann von Fall zu Fall anders sein.

Der BPMN Ad-Hoc Subprozess ist ein Beispiel für einen nicht strikt vordefinierten Prozess, da es viele nicht explizit spezifizierte Ausführungspfade gibt. Auch die Möglichkeit, Aktivitäten zu überspringen, würde in diesen Bereich zählen. Dies wäre auch durch die Modellierung eines XORs um die Task herum realisierbar, doch ist das ein wenig intuitiver Weg. Delegation (siehe [Nag08]) kann ein Weg sein, Adaptivität – in einem gewissen Rahmen – zu realisieren. Tritt beispielsweise die Ausnahme auf, dass ein Mitarbeiter eine Aufgabe nicht oder nur teilweise erledigen kann, so kann durch Delegation von einzelnen Formularfeldern entsprechend reagiert werden. Desweiteren sollte an der Oryx-Plattform das Konzept der Reevaluierung als weiterer Aspekt der Adaptivität betrachtet werden (siehe [Mas08]).

Wie soeben angedeutet, war die Untersuchung von Aspekten zur Flexibilität und Adaptivität zentrales Ziel dieses Bachelorprojektes. Der Nachverfolgung von Änderungen an Prozessmodellen sollte am wenigsten Beachtung geschenkt werden.

¹Business Process Modeling Notation

Ein übergreifender Aspekt, wenn auch nicht losgelöst von den vorgestellten Dimensionen, ist die Unterscheidung von Änderungen am Prozess, die zur Laufzeit oder zur Modellierzeit passieren. Die Implementationsarbeit an der SAP Suite beschränkte sich vor allem auf Modellierungsaspekte. Am Beispiel der Oryx-Plattform sollte sich zusätzlich auf Aspekte zur Laufzeit beschäftigt werden.

2.2 Führendes Szenario "Einstellung eines neuen Mitarbeiters"

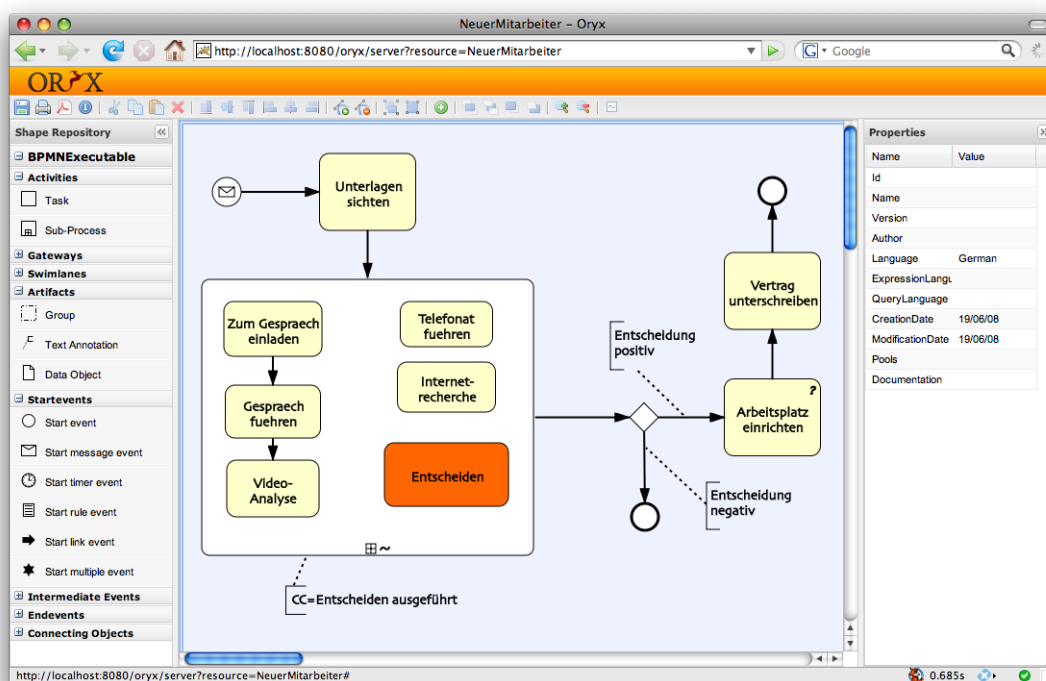


Abbildung 2.1: Beispielprozess in BPMN modelliert im Oryx

Um die Konzepte zu illustrieren, soll ein Beispiel eingeführt werden. Betrachtet wird ein einfacher Prozess aus dem Personalwesen (Abbildung 2.1). Dieser läuft grundsätzlich in 3 Stufen ab:

1. Die Bewerbungsunterlagen werden empfangen und gesichtet
2. In diversen Schritten wird die Tauglichkeit des Bewerbers ermittelt

3. Bewerbung war erfolgreich und Bewerber wird zum Mitarbeiter

Das Einreichen der Bewerbungsunterlagen kann durch den Bewerber selbst erfolgen, z.B. indem er ein Formular auf der Website des potentiellen Arbeitgebers ausfüllt. Besonderes Merkmal ist dabei, dass der Bewerber selbst eine Prozessinstanz erstellt und eine Reihe anderer Tasks damit anstößt. Das ist vereinfacht durch ein Message-Event dargestellt. Da die komplette Prüfung innerhalb einer Abteilung erfolgen soll, wurde auf die Modellierung von Pools und Lanes verzichtet.

Wichtigster Teil ist die Prüfung der Eignung des Bewerbers. Dieses Testverfahren wird auf die Bewerbung hin gestartet. Hier gibt es eine Menge an Aktivitäten, die in wenig geordneter Reihenfolge stattfinden können. Auch sind nicht alle Aufgaben tatsächlich auszuführen. Beispielsweise muss hier nicht zwingend eine Internetrecherche zum Bewerber stattfinden. Auf der anderen Seite sind auch einige Aktivitäten tatsächlich geordnet. So muss die Einladung zum Gespräch vor der Gespräch selbst erfolgen. Diese Abhängigkeiten innerhalb des Ad-Hoc Subprozesses sind mit normalem Sequenzfluss gekennzeichnet.

Weiteres wichtiges Merkmal ist, dass sobald die Entscheidung gefallen ist, ob der Bewerber eingestellt wird, alle weiteren Aktivitäten innerhalb des Bewertungssubprozesses hinfällig werden. Dementsprechend ist die Aktivität *Entscheiden* hervorgehoben, da sie über das Abschließen des kompletten Ad-Hoc Subprozesses entscheidet.

Wenn die Entscheidung für den Bewerber gefallen ist, dann kann ein Arbeitsplatz für neuen Mitarbeiter eingerichtet werden. Dieser Schritt kann auch unterbleiben, beispielsweise wenn der Mitarbeiter eine Neubesetzung für einen bestehenden Arbeitsplatz ist. Wenn alle Aktionen abgeschlossen sind, kann der Vertrag unterzeichnet werden und der Arbeitsalltag beginnen. Sollte die Entscheidung gegen den Bewerber gefallen sein, dann wird der Prozess ohne weitere Aktivität beendet.

2.3 Abgeleitete Anforderungen

Das Szenario führt zu folgenden Anforderungen an das Softwaresystem:

- **Multi-User Fähigkeit**
Mehrere User müssen gleichzeitig am System arbeiten können und eine personalisierte Ansicht der zu bearbeitenden Aufgaben erhalten. Darüber hinaus sollen auch Prozessinstanzen durch Einwirkung von außen erzeugt werden können.
- **Delegation**
Durch die starke Spezialisierung in Unternehmen sind einzelne Aufgaben oft nur durch Mithilfe von anderen Mitarbeitern zu lösen. Beispielsweise kann zur Videoanalyse der Bewerbung von der Personalabteilung ein weiterer Experte zu Rate gezogen werden.

- **Skipping**
Es soll vorgesehen werden, dass Tasks zur Laufzeit übersprungen werden, wenn ihre Bearbeitung vom Prozessverlauf nicht zwingend erfordert wird. Dafür kann es den Grund geben, dass eine Task zwar nicht immer, aber wenn, dann an einer bestimmten Stelle im Prozess ausgeführt werden soll. So möchte das Unternehmen dem Arbeitnehmer den Arbeitsplatz vor Arbeitsantritt zur Verfügung stellen, aber nicht zwingend muss dieser neu eingerichtet werden, weil er vielleicht schon existiert.
- **BPMN Ad-Hoc-Subprozess mit Completion Condition**
Um Freiheitsgrade für den Modellierer zu schaffen, soll die BPMN-Ad-Hoc-Task inklusive Abhängigkeiten zwischen einzelnen Tasks zur Ausführung gebracht werden und mittels einer Completion Condition die Überprüfung auf einen zur Beendigung geeigneten Zustand bewerkstelligt werden. Hierbei handelt es sich mehr oder weniger um eine nicht-funktionale Eigenschaft, die allerdings das Modellieren in diesem Fall wesentlich vereinfacht.

2.4 Gefärbte Petrinetze mit Guard Conditions

Um die Anforderungen im System ausführbar zu machen, soll es ermöglicht werden, die gut lesbaren Prozessmodelle in leicht abwickelbare Strukturen abzubilden. Dabei wurde sich für gefärbte Petrinetze mit Guard Conditions entschieden.

Petrinetze gehen als theoretisches Konstrukt zurück auf Carl Adam Petri, der in seiner Dissertation 1962 dieses Modell erstmals vorschlägt [Pet62]. Da sich mit einem solchen Formalismus schnell Probleme aufzeigen und Konzepte visualisieren lassen, wurde es als theoretische Grundlage für die Ausführungsumgebung in der Oryx-Plattform verwendet. Darüber hinaus gibt es inzwischen nahezu ein halbes Jahrhundert an Erfahrungen mit Petrinetzen und dementsprechend viele Abhandlungen, die sich intensiv damit beschäftigen.

Gefärbte Petrinetze [Jen96] sind Stellen-Transitionsnetze. Es handelt sich hierbei um bipartite Graphen, die aus Transitionen, Stellen und gerichteten Kanten bestehen. Auf einer Stelle können beliebig viele Tokens liegen. Beim Schalten einer Transition wird über jede eingehende Kante genau ein Token konsumiert und über jede ausgehende Kante genau ein Token produziert. Bei Stellen-Transitionsnetzen sind alle Token jedoch gleich beschaffen. Gefärbte Petrinetze gehen darüber hinaus und kodieren in jedes Token eine Farbe. Im Beispiel der für diese Arbeit relevanten Netze entspricht die Farbe eines Tokens einem kompletten XML-Dokument. Guard Conditions können dabei nicht nur wie in normalen gefärbten Petrinetzen als Arc Expressions an Kanten funktionieren, sondern auch direkt an Transitionen, also alle eingehenden Kanten zur gleichen Zeit evaluieren.

```

<?xml version="1.0"?>
<data>
  <metadata>
    <startTime>2008-06-10T16:41:35+00:00</startTime>
    <endTime xmlns:exsl="http://exslt.org/common"/>
    <status>running</status>
    <owner>testuser</owner>
    <actions>
      <action>
        <name>allocate</name>
        <time>2008-06-10T16:41:35+00:00</time>
        <user>testuser</user>
      </action>
      <action xmlns:exsl="http://exslt.org/common">
        <name>enable</name>
        <time>2008-06-10T15:23:50+00:00</time>
        <user />
      </action>
    </actions>
  </metadata>
</data>

```

Ein möglicher Tokenwert, wie er in der Implementierung benutzt wird, ist hier abgebildet. Grundsätzlich wird das Token in Nutzdaten und Metadaten aufgeteilt. Im Beispiel ist ein Token einer Kontextstelle zu sehen, das nur Metadaten enthält. Es enthält hier unter anderem den aktuellen Status und den Nutzer, der momentan das Token besitzt.

2.5 Umwandlung von BPMN in Petrinetze

Prozessmodelle können in der vorliegenden Modellierungsumgebung Oryx durch viele Notationen ausgedrückt werden. Die ausgereifteste davon ist die BPMN. Um die Modelle zur Ausführung zu bringen, wird zuerst deren Struktur erfasst und dann mit einem Task Lifecycle ein Zustandsmodell für jede Aktivität erstellt. Der Formalismus, in den umgewandelt wird, sind die zuvor beschriebenen gefärbten Petrinetze.

Strukturelle Umwandlung

Im einfachsten Fall werden Aktivitäten zu Transitionen mit einer angeschlossenen Stelle. Zudem werden zum Beispiel AND-Splits zu einer Transition mit zwei ausgehenden Stellen. Dies ist in Abbildung 2.2 visualisiert. Alle weiteren Kontrollflusskonstrukte lassen sich auf ähnliche Weise in einem Petrinetz abbilden und orientieren sich an [DDO08].

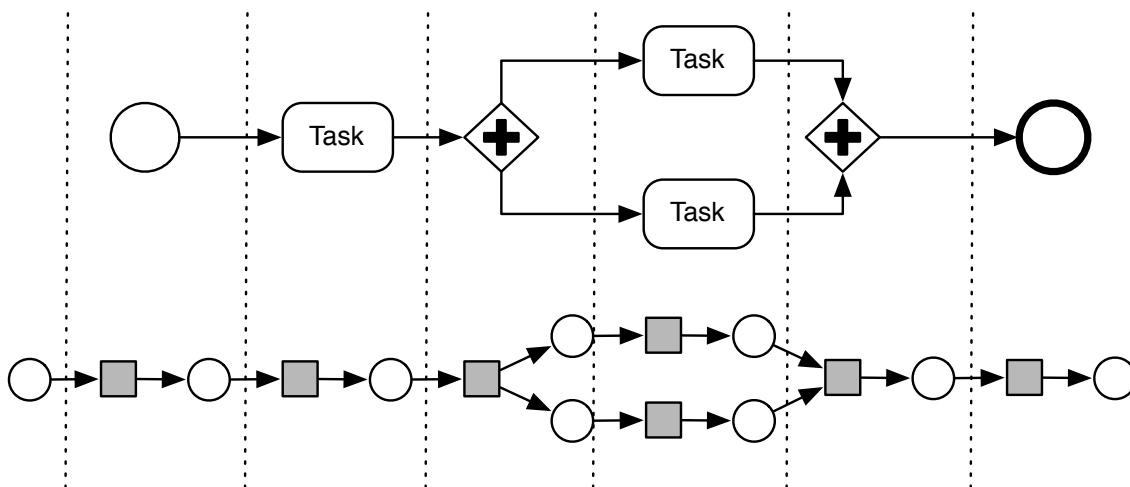


Abbildung 2.2: Grundlegende Umwandlung BPMN nach Petrinetz

Task Lifecycle

Das Schalten einer Transition passiert modellhaft betrachtet in einer vernachlässigbar kurzen Zeit. So müssen die Zustände der einzelnen Aktivitäten in BPMN als Petrinetz abgebildet werden. Aus den Markierungen sind dann die Zustände ablesbar. So soll das Allokieren einer Task dem Schalten einer Transition entsprechen. Aus Gründen der Übersichtlichkeit, wird in Abbildung 2.3 eine vereinfachte Version des Lifecycle gezeigt. Die finale Version verfügt zusätzlich über Transitions zur Delegation und zum Review. Vorteil dieses standardisierten Lifecycle ist, dass die Task, wie sie aus dem BPMN-Modell stammt, als Einheit im Petrinetz bestehen bleibt und eine definierte Schnittstelle nach außen liefert. So kann schnell festgestellt werden, welche Transition welche Bedeutung hat und dies kann im User Interface entsprechend dargestellt werden.

Auffallend ist hier, dass jede Task mit einer Transition beginnt. Das rührt daher, dass das Instanzieren einer Prozessinstanz in der Ausführungsumgebung durch Schalten einer Transition passiert. Das Belegen einer Stelle mit einem Token (siehe 4.4) und der damit verbundenen Instanzerstellung wird bei der Umwandlung nicht vorgesehen.

Ad-Hoc Subprozess

BPMN sieht ein Attribut *AdhocOrdering* für den Ad-Hoc Subprozess vor. Es kann zwei Werte annehmen: *'sequential'* und *'parallel'*. Wir wollen hier exemplarisch den sequentiellen Fall betrachten.

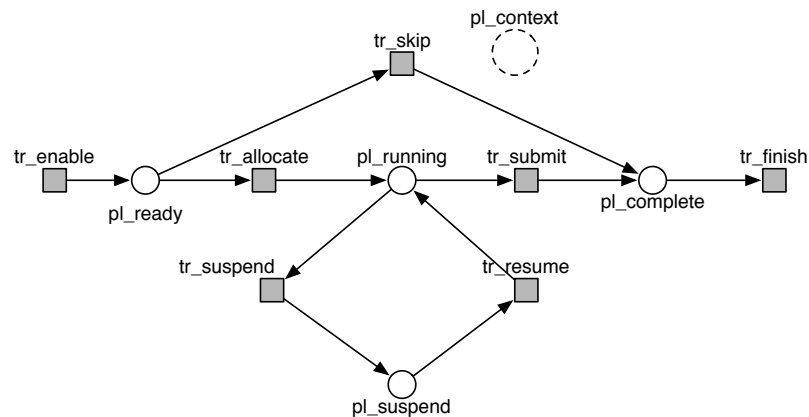


Abbildung 2.3: Task Lifecycle im Petrinetz

Grundsätzlich ist die Transformation so aufgebaut, dass es eine Verwaltungsstruktur gibt, die den Status des Ad-Hoc Subprozesses selbst trägt und jede enthaltene Task ist über ihre feste Schnittstelle, den Task Lifecycle, angebunden.

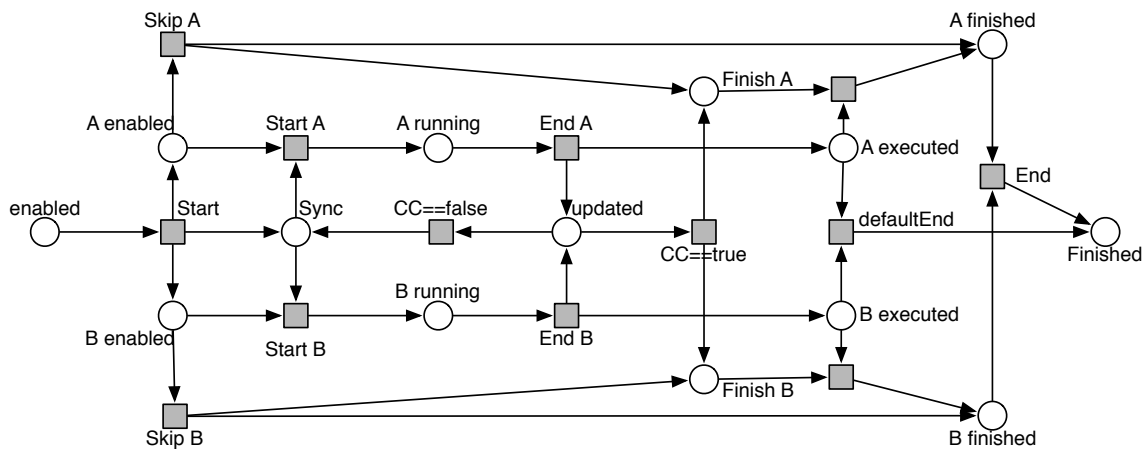


Abbildung 2.4: Sequentieller Ad-Hoc Subprozess als Petrinetz

Über eine Sync-Stelle wird realisiert, dass jeweils nur eine Task gleichzeitig laufen kann. Nach dem Ausführen einer Task wird die Completion Condition überprüft und entweder wieder neu ein Token auf die Sync-Stelle gelegt oder alle noch nicht ausgeführten Tasks geskippt und der Subprozess verlassen. Sehr viel detaillierter beschrieben ist die Umwandlung des Ad-Hoc Subprozesses in [Kru08].

3. Entwicklungen an der Oryx-Plattform

Nachdem die theoretischen Grundlagen beleuchtet wurden, soll in diesem Kapitel ein Überblick über die Softwarelandschaft am Hasso-Plattner-Institut gegeben werden. Dabei soll vor allem herausgestellt werden, welche Teile zu Projektbeginn existierten, wie diese miteinander integriert wurden und was zusätzlich entwickelt wurde. Auch soll die Entwicklungsmethodik umrissen werden und wie diese die Arbeitsergebnisse beeinflusst hat.

Die Business Process Management Umgebung am HPI besteht aus 4 abgrenzbaren Bestandteilen (Abbildung 3.1):

- Modellierung
- Modelltransformation
- Petrinetzausführung
- Worklist als Nutzeroberfläche zur Fallbearbeitung

Jede Komponente stammt dabei aus unterschiedlichen Vorläuferprojekten, die aufgrund der neuen Anforderungen weiterentwickelt und miteinander verbunden werden mussten. Einzig die Worklist ist ein komplett neu entstandener Bestandteil des Toolsets.

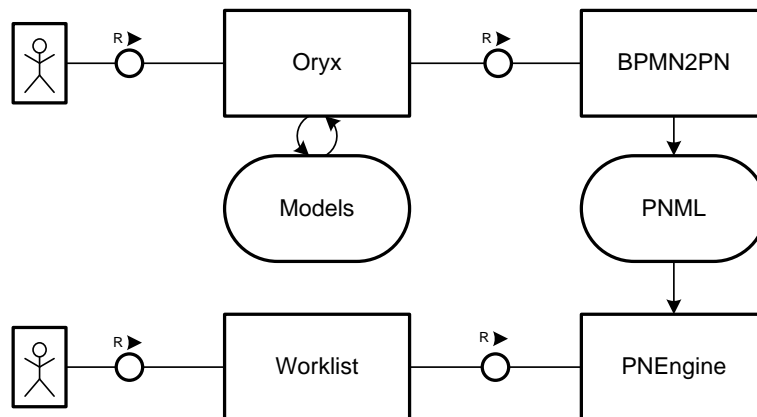


Abbildung 3.1: Architektur der Oryx-Plattform

3.1 Existierende Prototypen

Den Entwicklungen an den bestehenden Prototypen ging eine Evaluierung der Umgebungen voraus. Es wurde überprüft, inwieweit die benötigten Fähigkeiten schon im vorliegenden Stand erfüllt werden konnten. Hier offenbarte sich, dass einige Komponenten schon sehr ausgereift waren und kaum Anpassungen benötigten und andere grundsätzliche Erweiterungen brauchten.

Oryx ist die zentrale Modellierungsumgebung, die das kollaborative Erstellen und Bearbeiten von Modellen im Web ermöglicht. Ausgangslage für das Bachelorprojekt war, dass es ein Stencil-Set gab, das es ermöglicht, den kompletten Sprachstandard von BPMN zu nutzen. Durch die erweiterbare Plugin-Schnittstelle war auch die Möglichkeit offen, hier Erweiterungen vorzunehmen. Entstanden ist Oryx durch das vorangegangene Bachelorprojekt und wird seitdem intensiv weiterentwickelt. Es ist die ausgereifteste Komponente des Toolsets und erfüllte weitgehend die Anforderungen, die an sie gestellt wurden.

Die Petrinetzengine dient als Ausführungscontainer. Theoretische Grundlage sind gefärbte Petrinetze mit Guard-Conditions, die in diesem Container abgewickelt werden. Die Netze können in Petri Net Markup Language (PNML [BCvH⁺03]) beschrieben werden. Entstanden ist das Projekt aus einem Seminar am Lehrstuhl. Die Erweiterbarkeit ist durch die Architektur zum einen und die Nutzung des stark generalisierten Konzepts 'Petrinetz' zum anderen gegeben. Allerdings sind grundlegende Konzepte wie Fälle oder Nutzer weder in Petrinetzen, als auch in der Engine selbst vorgesehen gewesen. Fälle sind hierbei explizit als solche benannte Instanzen von Prozessmodellen. Hierzu bedurfte es umfangreichen Ergänzungen.

Die Transformation lag als prototypisches Framework vor und sollte dazu dienen BPMN in Petrinetze umzuwandeln. Starke Erweiterungen mussten in dieser Kom-

ponente gemacht werden, da in der Umwandlung und der damit verbundenen Erstellung von z.B. Guard-Conditions große Teile der Ausführungslogik liegen. Konzeptionell war hier ein solider Rahmen gegeben, der durch die entsprechende Implementierung gefüllt wurde.

3.2 Verwendete Technologien

Es war von Anfang an keine Wahl zu treffen, welche Programmiersprachen oder andere Technologien benutzt werden sollen. Die Entscheidungen waren jeweils in den existierenden Komponenten schon getroffen. Aufgrund der losgelösten Entwicklung der einzelnen Komponenten, sind auch mehrere unterschiedliche Technologien zum Einsatz gekommen. Gemein ist ihnen allen, dass die nahe Anlehnung an Webtechnologien ein zentraler Aspekt ist.

So nutzt Oryx das browserbasierte JavaScript-Framework ExtJS¹ um für jeden verfügbar unterschiedlichste Arten von Modellen einfach bearbeitbar zu machen. Zur Darstellung der Zeichenfläche wird SVG (Scaleable Vector Graphics) benutzt, was zur Folge hat, dass sich die Browser-Unterstützung momentan auf den Firefox² beschränkt. Das Backend ist teilweise in Java, teilweise in Ruby programmiert. Es gibt zwei Backends, das ältere und in Zukunft nicht mehr unterstützte, nutzt als Datenbank MySQL, das zweite hält die Daten in PostgreSQL, wobei die Daten mittels POEM³ serialisiert werden.

Die Petrinetengine wurde in Ruby geschrieben und sieht sich als Implementierung des REST-Architekturstils [Fie00]. Zur Anzeige der Formulare wird die browserbasierte Formularsprache XForms [xfo07] benutzt, die Teil der noch nicht verabschiedeten Version 2.0 der XHTML-Spezifikation sein wird. Die Browser-Unterstützung ist auch hier bisher nicht sehr weitreichend, allerdings existiert für den Firefox ein Addon, das das Rendern der XForms übernimmt. Die Beschränkung auf diesen Browser ist akzeptabel, da der Oryx auch exklusiv auf dieser Plattform läuft.

Die Transformation ist durch ein Java-Servlet realisiert und nach außen bereitgestellt. Momentan ist dieses in den Webcontainer des Oryx eingebettet, ließe sich aber auch jederzeit auf einem anderen Host deployen, sodass es als externer Dienst zur Verfügung gestellt bzw. genutzt werden könnte.

Die Worklistkomponente nutzt ebenso ExtJS als Framework zur Darstellung von Oberflächenelementen und ist somit in guter Gesellschaft zum Oryx, da beide Teile sich einfach miteinander integrieren ließen. Zur Kommunikation mit der Engine wird HTTP verwendet, die im Sinne von AJAX⁴ stattfindet

¹<http://www.extjs.com>

²<http://www.mozilla-europe.org/de/firefox/>

³POEM = Potsdam Encoding for Models

⁴Asynchronous Javascript And Xml

3.3 Integration der Komponenten

Damit die Möglichkeit geschaffen werden konnte, dem Nutzer das gesamte Toolset als Lösung 'aus einem Guss' zu präsentieren, mussten die voneinander unabhängigen Einzelsysteme verbunden werden. Zur Festlegung von geeigneten Schnittstellen sollen zuerst die Kommunikationswege herausgestellt werden:

- Modellierungsumgebung mit Transformationskomponente
- Transformationskomponente mit Petrinetz-Engine
- Worklist mit Petrinetz-Engine

Zur nahtlosen Integration in Oryx ist ein Plugin entstanden, das die als RDF (Resource Description Framework) vorgehaltenen Modelldaten an den Umwandler weiterreicht. Das Modell ist in die ausgelieferten HTML-Seiten als eRDF (embedded RDF) eingebettet, um die Darstellung mittels Oryx direkt an die Ressourcen-URL des jeweiligen Modells zu koppeln. Zur Extraktion der Daten als RDF wird XSLT benutzt, das das eRDF transformiert. Die Schnittstelle zur Transformationskomponente (im Folgenden BPMN2PN) wird momentan als JavaServlet implementiert, gibt das erzeugte PNML aus und deployt es auf Wunsch automatisch an die Engine.

Das Transformationsservlet kann die Daten automatisch an eine dazu konfigurierte Engine weiterleiten. Hierbei wird die Eigenschaft der Engine benutzt, eine *RESTful* Schnittstelle nach außen zu bilden.

Der Endanwender nutzt zur Bearbeitung seiner Fälle die Worklist. Hier werden alle Vorgänge inklusive der dazugehörigen Formulare dargestellt. Die für den Nutzer große Komplexität der dahinterliegenden Petrinetze wird durch diese Abstraktionsschicht in Fälle, Aufgaben und Aktionen untergliedert. Dies ist eine Abbildung der Zustände im zugrundeliegenden Petrinetz. Die Daten, die hier zur Anzeige gebracht werden, kommen von der Petrinetz-Engine und aggregieren Laufzeitinformationen, wie aktivierte Transitionen und die dazugehörigen Tokenkombinationen.

Gerade durch die Entwicklungsmethodik, die stark auf Prototypen setzte, war der Integrationsaufwand recht erheblich. Vielfach mussten Anpassungen an den Teilprojekten vorgenommen werden, um vereinbarte Schnittstellen einzuhalten. Beispielsweise mussten Änderungen zu speziellen Ad-Hoc Attributen im Stencilset oftmals in allen anderen Komponenten reflektiert werden. So soll der Umwandler genau diese Ad-Hoc Attribute geeignet ins PNML übertragen und die Engine kann die Attribute auch für die Ausführung benutzen. Gerade bei den Datenmodelldefinitionen wurden weitreichende Änderungen in allen Projektteilen durchgeführt.

3.4 Entwicklungsmethodik

Durch die agile Entwicklungsmethodik, die von Seiten der SAP AG im Bachelorprojekt etabliert wurde, sind sowohl konzeptionelle Arbeiten, als auch Implementierungen in kurzen Zyklen entstanden und wurden dann zu einem Ganzen zusammengefügt. In weiten Teilen wurde dabei Scrum (Abbildung 3.2), als Verfahren den Softwareentwicklungsprozess zu planen, durchzuführen und zu evaluieren, angewandt.

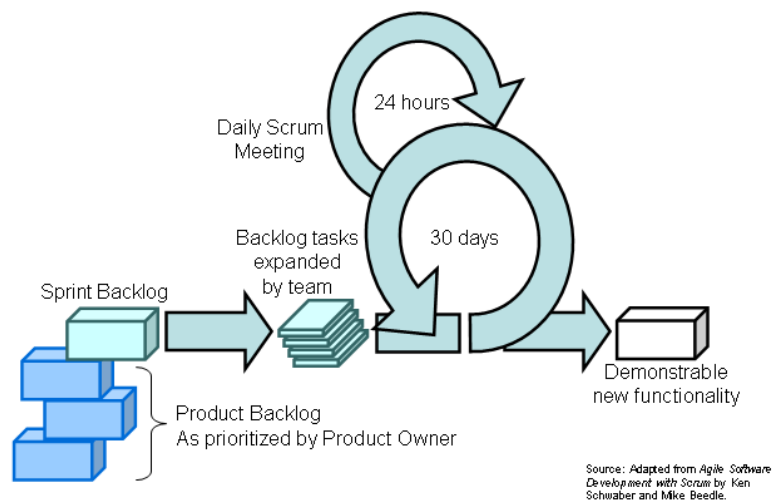


Abbildung 3.2: Übersicht Scrum Softwareentwicklungsprozess

Scrum erfüllt als agile Methode die Werte, wie sie 2001 im 'Manifest der agilen Softwareentwicklung'⁵ definiert wurden:

- Individuen und Interaktionen gelten mehr als Prozesse und Tools
- Funktionierende Programme gelten mehr als ausführliche Spezifikation
- Zusammenarbeit mit dem Kunden steht über Verträgen
- Reaktion auf Änderungen steht über dem Befolgen eines festgelegten Plans

Dieses Manifest wurde neben den Begründern von Scrum – Ken Schwaber und Jeff Sutherland – von 15 weiteren Personen entwickelt und findet täglich weitere Unterzeichner⁶.

⁵<http://agilemanifesto.org/>

⁶<http://agilemanifesto.org/sign/display.cgi?ms=all>

Vor allem war es wichtig, kleinere Prototypen zu erstellen, die dann nach und nach in die mehrschichtige Toolchain eingesetzt wurden. Am Beispiel der Delegation, wird dies besonders deutlich. Ganz am Anfang der Entwicklung stand hier ein Prototyp, der – im Zuge des Vertrautmachens mit SAP WebDynpro – auf einfache Weise die Handhabung von Delegation im Kontext der Taskbearbeitung aufzeigte. Dieser wurde für gut befunden und nachdem die strategische Entscheidung fiel, die laufzeitorientierten Aspekte an der Oryx-Plattform umzusetzen, in XForms reimplementiert. Nach der Evaluation dieses zweiten Prototypen konnte die Integration in das Standardrepertoire der BPMN-nach-Petrinetz Transformation erfolgen. Auf ähnliche Weise wurde auch mit weiteren Teilkomponenten der Oryx-Plattform verfahren, wie z.B. dem Task-Lifecycle und der Worklist-Komponente.

Das ständige Feedback, wie es durch Scrum vorgegeben wird, führte zu einer ständigen Verbesserung des Entwicklungsprozesses und erhöhte zudem die Qualität der Entwicklung. Auch konnte so täglich nachvollzogen werden, ob Entwicklungen in die richtige Richtung gehen und gegebenenfalls korrigierend eingegriffen werden.

Neben der Benutzung von Scrum, vor allem zur Abstimmung mit den Projektpartnern und untereinander, wurden weitere agile Softwareentwicklungsmethoden angewandt. Darunter fällt zum Beispiel das Praktizieren von Pair Programming, das gerade bei der komplexen Umsetzung der Transformationskomponente half, den Überblick zu behalten. Weiterhin ließen uns regelmäßige Code-Reviews und die Orientierung vor allem an Use-Cases, zielgerichtet zu Ergebnissen kommen.

4. Implementierungen

Die besondere Herausforderung bei der Entwicklung bestand darin, dass nicht an einem einzigen Produkt entwickelt wurde, sondern an vier Teilprojekten. Trotzdem sollte die Software nach außen hin ein geschlossenes Bild vermitteln. Dazu war viel Abstimmung auf der einen und klare Definition von Schnittstellen auf der anderen Seite notwendig.

In diesem Kapitel soll gezeigt werden, welche Komponenten wie weiterentwickelt wurden und welche Teile komplett neu entstanden sind.

4.1 Schnittstellendefinition

Schnittstellendefinition bezieht sich speziell auf die Definition von Datenaustauschformaten. Als Protokoll dient über sämtliche Kommunikationswege HTTP. Aufgezeigt werden die übermittelten Nutzdaten und deren Bedeutung für die Kommunikationspartner.

Zur Eingabe in den Transformationsprozess dient RDF. Dieses Format und deren Verwendung im Kontext von Oryx ist in [Czu07] genau beschrieben. Wichtiger für die Realisierungen im Rahmen dieses Projekts ist die Beschreibung der Petrinetze, die in PetriNet Markup Language(PNML) erfolgt.

Deployment von Petrinetzen

Grundsätzlich ist PNML in [BCvH⁺03] erklärt und dessen Schema durch RelaxNG klar reglementiert¹. PNML ist ein XML-Dialekt, der die Speicherung von Transitionen, Stellen und Kanten ermöglicht. Zusätzlich dazu ist es möglich in einem

¹<http://www2.informatik.hu-berlin.de/top/pnml/pnml.html>

`toolspecific`-Teil beliebigen weiteren XML-Content anzugeben, der nur von der entsprechenden Software ausgewertet werden soll. So war eine Festlegung, dass die Engine nur `toolspecific`-Tags wie `<toolspecific tool="Petri Net Engine" version="1.0">` auswertet. Dort können dann zusätzlich Angaben gemacht werden:

- Anzeigoptionen für die Worklist
- Locators und Guards
- Rollendefinitionen
- Bezug auf die Kontextstelle
- Transformationen mittels XSLT-Dokumenten
- Datenmodell- und Formulardefinitionen

Ein (nicht vollständiges) Beispiel für den `toolspecific`-Teil einer Transition könnte wie folgt aussehen:

```
<toolspecific tool="Petri Net Engine" version="1.0">
  <fire href="http://localhost:3000/examples/contextPlace/copy_xslt.xsl" />
  <worklist action="resume" task="Unterlagen sichten" />
  <role>Sachbearbeiter</role>
  <contextPlaceID>place_pl_context_resource0</contextPlaceID>
</toolspecific>
```

Hierbei handelt es sich um die ausführungsspezifischen Angaben, zu einer Transformation-Transition. Dabei besteht die Nutzeraktion beim Schalten nur darin, dass 'Absenden' gedrückt wird. Dadurch ausgelöst wird auf die Eingangsdaten eine XSL-Transformation angewendet. In diesem Fall besteht die Transformation darin, dass lediglich der Inhalt der eingehenden auf die ausgehenden Token kopiert wird.

Die Angaben zu `worklist` sagen aus, welche Semantik die Transition im Kontext der Taskausführung hat. Sie gehört in dem Beispiel zur Task 'Unterlagen sichten' und führt die 'resume'-Aktion entsprechend des Lifecycles 2.5 der Task aus. So wird ermöglicht, dass Transitionen direkt in die Darstellung der Worklist gemappt werden, und zwar als zentrale Aktionsbuttons für jede Task.

Die Rollendefinition dient dazu, den potentiellen Taskbearbeiter einzuschränken. Das Sichten der Unterlagen darf also nur von einem Nutzer der Rolle Sachbearbeiter ausgeführt werden. Das Element `contextPlaceID` dient zur Angabe der zur Transition gehörigen Kontextstelle und wird benutzt, um die Metadaten korrekt zu den jeweiligen Transitionsgruppen, also Tasks, zuzuordnen.

Anzeige von Ausführungsinformationen

Das abzuwickelnde Petrinetz soll durch eine Worklist verwaltet werden. Das ExtJS-Framework bietet die Möglichkeit, Views durch Daten aus XML-Quellen zu füllen. Genau diese Möglichkeit wurde verwendet, um Ausführungsinformationen zur Anzeige zu bringen. Es wurde eine Case-Übersicht, eine Case-Detailansicht und ein Log-Overview implementiert, die die Anzeige der benötigten Informationen für Worklist und Monitoring darstellen. Als Vorbild zum Worklist-Design dient die klassische dreigeteilte Ansicht eines E-Mail-Clients.

```
<?xml version="1.0" ?>
<tasks>
  <task>
    <case_id>6</case_id>
    <net_name>NeuerMitarbeiter</net_name>
  </task>
</tasks>
```

Im Listing ist eine mögliche Ausgabe der Case-Übersicht dargestellt, hier mit genau einem aktivierten Case, der durch eine Task repräsentiert wird. Dieses Dokument wird von der Worklist ausgewertet, um die Ansicht der Cases zu erstellen. Es dient dazu, aktuell laufende Instanzen der jeweiligen Netze darzustellen und sie für die Case-spezifische Task-Ansicht auswählbar zu machen.

```
<?xml version="1.0" ?>
<tasks>
  <task>
    <transition_id>179/tokens/179;180</transition_id>
    <task_name>Unterlagen sichten</transition_name>
    <action>allocate</action>
    <case_id>6</case_id>
    <pending_time>298.76</pending_time>
    <actual_owner>testuser</actual_owner>
    <role>Sachbearbeiter</role>
    <net_name>NeuerMitarbeiter</net_name>
    <user_enabled>true</user_enabled>
    <delegation_ratio>0.0</delegation_ratio>
  </task>
</tasks>
```

Die hier gezeigte detaillierte Case-Ansicht wird für die Anzeige aller Tasks, die zu einem Case gehören, benutzt. Die `transition_id` gibt hier nicht nur die Transition an, die die Aktion repräsentiert, sondern auch die Token-Kombination, mit der die Transition für den jeweiligen Fall schaltbereit ist. Die `action` bezieht sich auf die im PNML beim Deployment angegebene Bedeutung der Transition. `pending_time` bezeichnet die Zeitdauer (in Sekunden), die bereits verstrichen ist, seitdem die Transition aktiviert wurde. Angezeigt werden alle Tasks, die im Sinne der Petrinetz-Schaltsemantik aktiviert sind, die Angabe `user_enabled` bezieht sich zusätzlich auf den Umstand, welcher User im anliegenden Token eingetragen ist und ob

dieser mit dem aktuell eingeloggten Benutzer übereinstimmt. `delegation_ratio` ist eine statistische Angabe, die ausdrückt in wievielen Fällen der Ausführung der jeweiligen Task, die Task delegiert wurde. Mehrfachdelegation zählt dabei als einmaliger Delegierungsvorgang.

4.2 Oryx

Die Modellierungsumgebung Oryx dient als zentrales Repository für Modelle. Diese sollen über die zuvor beschriebenen Schnittstellen direkt deployed werden. Dazu wurde die Modellierplattform um ein Plugin für die Petrinetzumwandlung und eine Parallelversion `bpmnexecutable` des BPMN-Stencilsets für ausführbare Modelle erweitert. Die Kernfunktionalität des Oryx blieb unangetastet, da die dazu vorgesehenen und vorhandenen Schnittstellen zur Erweiterung genutzt werden konnten.

Export-Plugin

Die Diagramme, die im Oryx in BPMN erstellt wurden, liegen in eRDF [Dav06] vor. Das hat den Grund, dass die Modelldaten beim Abrufen direkt in die Modellierungsumgebung eingebettet werden. Die Antwort, die das Oryx-Backend auf ein GET an die Modellressource liefert, ist eine HTML-Seite inklusive Modelldaten. Damit die Daten für die Transformation nutzbar werden, müssen diese extrahiert werden. Dies geschieht als Transformation von eRDF in RDF [DB04] mittels einer XSL-Transformation. Dies funktioniert immer, da: *'all HTML Embeddable RDF is valid RDF, not all RDF is Embeddable RDF.'*².

Das RDF wird im Transformator analysiert und die Ausgabe in PNML an die Engine weitergereicht. Dazu muss in der Konfigurationsdatei `'pnengine.properties'` eine zugehörige Petrinetzengine eingestellt sein, an die deployed werden soll. Die Kommunikation von Oryx und Engine kann an dieser Stelle nicht direkt im Browser funktionieren, da die *'Same-origin policy'* nicht zulässt, per JavaScript auf einen fremden Server zuzugreifen.

Auch wenn beide Systeme auf dem gleichen Host laufen, verwenden sie typischerweise einen anderen Port, was auch gegen ebendiese Richtlinie verstößt. Somit muss das Deployment vom Plugin-Servlet an die Engine erfolgen, da es hier keine solche Restriktionen gibt. Im Falle eines Fehlers beim Deployment wird eine Fehlermeldung wieder im Browser-Plugin ausgegeben oder – im Normalfall des Erfolges – ein Link zur Worklist angegeben (Abbildung 4.1). Man gelangt also durch genau einen Mausklick zu einer lauffähigen Version des modellierten Ablaufs.

²<http://www.siatec.net/erdfparser/>

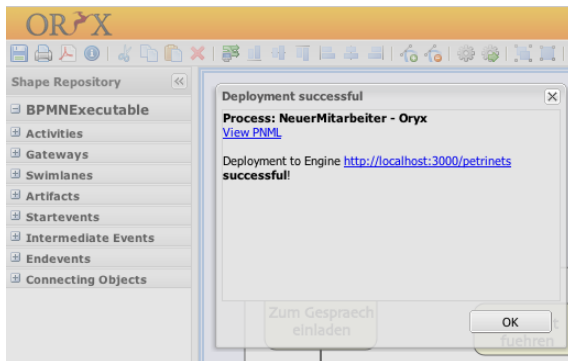


Abbildung 4.1: Export Plugin

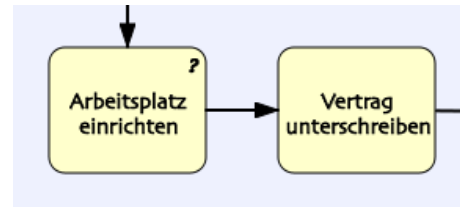


Abbildung 4.2: Überspringbare Aktivität

Skippable Task

Die Erweiterung der überspringbaren Aktivität ist zwar hauptsächlich eine Fähigkeit der Ausführungsumgebung, soll aber auch im Oryx als sichtbares Attribut erscheinen.

`is_skippable` wurde als Boolean-Attribut mit dem Default-Wert `false` zum Stencilset hinzugefügt, damit die Möglichkeit besteht, dass eine Aufgabe erledigt werden muss und nicht jede Aktivität ungewollt dieses 'Feature' erhält. Signalisiert wird es in der Oberfläche als kleines Fragezeichen, das an die Task annotiert ist, wie in Abbildung 4.2 zu sehen.

Erweiterungen zum Ad-Hoc Subprozess

Im Rahmen der Modellierung des Ad-Hoc Subprozesses (Abbildung 4.3) mussten zusätzlich Änderungen am Stencilset gemacht werden. Die Wichtigsten sind hier die zusätzlichen Attribute `AdHocOrdering` und `AdHocCompletionCondition`. Während ersteres durch eine einfache Auswahl zwischen `Sequential` und `Parallel` zu erledigen ist, schien es schwer, den User eine Completion Condition schreiben zu lassen, die die Syntax der Guard-Expressions – also die Ruby-Syntax – nutzt. Dazu wurde ein Dialog entwickelt, der das komfortable Editieren auch komplizierter Ausdrücke erlaubt. Genauer beschrieben wird diese Entwicklung in [Kru08].

Datenmodelldefinition

Es wurde ein Attribut `DataModel` zum Metamodell der Datenobjekte (Abbildung 4.4) hinzugefügt in das XSD-Content eingetragen werden kann. Dieses Dokument dient als Vorlage für die Werte der Datenobjekte und als Modell für die

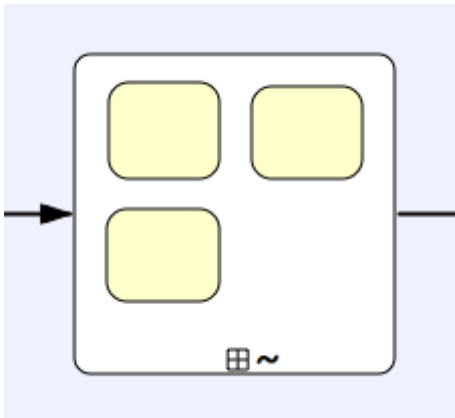


Abbildung 4.3: Ad-Hoc Subprozess

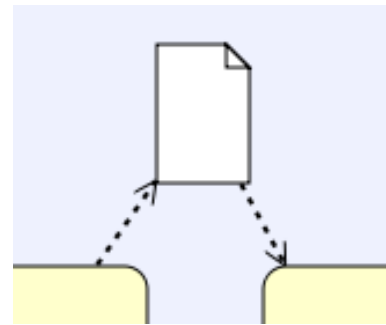


Abbildung 4.4: Datenobjekt

Datentoken im Petrinetz. Auf Basis dessen werden Formulare erzeugt, die das Datenmodell wiederum als Ein- bzw. Ausgabemodell nutzen und durch XForms beschrieben werden. Eine solche Formulardefinition besteht aus: Modell, Instanz(en), Bindings und Steuerungsinformationen. Es wird ein MVC-Pattern benutzt, das eine klare Trennung zwischen Daten, deren Darstellung und Steuerungsinformationen schafft. Die Verarbeitung der Datenmodelle ist Teil der Transformation und wird in [Kog08] genauer erklärt.

4.3 Transformationskomponente BPMN2PN

Um die Brücke zu schlagen, zwischen einfach lesbaren Modellen in BPMN und strikt formalisierten und damit maschinell gut abwickelbaren Petrinetzen, wurde der Umwandler entwickelt, um diese Modellarten ineinander zu überführen. Diese Transformation läuft dabei in 3 Schritten ab:

1. Import der Daten aus RDF und Erzeugung einer Instanz des BPMN-Metamodells
2. Umwandlung der Datenstruktur von BPMN in Petrinetz
3. Export der Petrinetz Metamodell-Instanz im PNML-Format

Das Einlesen der RDF-Modelldaten geschieht im `BPMNRDFImporter`, der die Subjekt-Prädikat-Objekt-Beziehung in die Klassenhierarchie übernimmt und Instanzen erstellt. Das BPMN-Metamodell ist in Java implementiert und bildet das vollständige BPMN-Stencilset aus dem Oryx ab. Die Erweiterungen im Stencilset `bpmnexecutable` wurden ebenso auch in das Java-Metamodell übernommen.

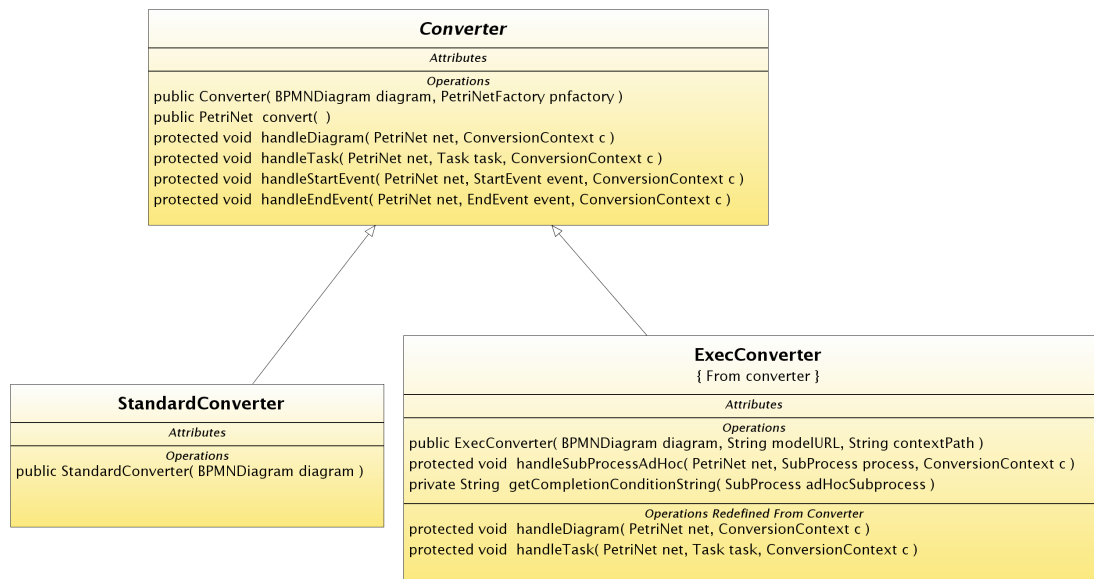


Abbildung 4.5: Klassendiagramm Konverter

Eine Umwandlerklasse ist dafür verantwortlich, die BPMN-Objektstruktur in eine Petrinetz-Objektstruktur zu überführen. Da ein BPMN-Modell geschachtelt aufgebaut sein kann (Diagramm enthält Subprozess, Subprozess enthält Tasks, usw.), geschieht dieser Umwandlungsprozess rekursiv. Es gibt mehrere Umwandlerimplementierungen. Zwei davon sind der **StandardConverter**, der eine Umwandlung ohne Task-Lifecycle und ohne ausführungsspezifische Attribute vornimmt, und der **ExecConverter**, der eine komplett ausführbare Version des Prozessmodells erzeugt (siehe Abbildung 4.5).

Die spezialisierte Version der Umwandlung nutzt auch ein spezielles Petrinetz-Metamodell in dem die zur Ausführung benötigten Attribute reflektiert werden. Beim **StandardConverter** hingegen genügt die Petrinetzstruktur selbst.

Daraus ergibt sich für die unterschiedlichen Transitionen eine Vererbungshierarchie, wie sie in Abbildung 4.6 dargestellt ist. Es gibt grundsätzlich zwei Arten von Transitionen: **TauTransitions** und **LabeledTransitions**.

Eine **TauTransition** wird bei der Umwandlung in ausführbares PNML zu einer **AutomaticTransition**. Wichtigstes Merkmal ist das automatische Schalten dieser Transitionen durch die Engine. Diesen Transitionen ist jeweils ein XSLT-Dokument hinterlegt, das zur Transformation der Token-Werte dient. Im default-Fall ist das `cp_one_token.xsl`, das lediglich die eingehenden Token-Werte auf die ausgehenden

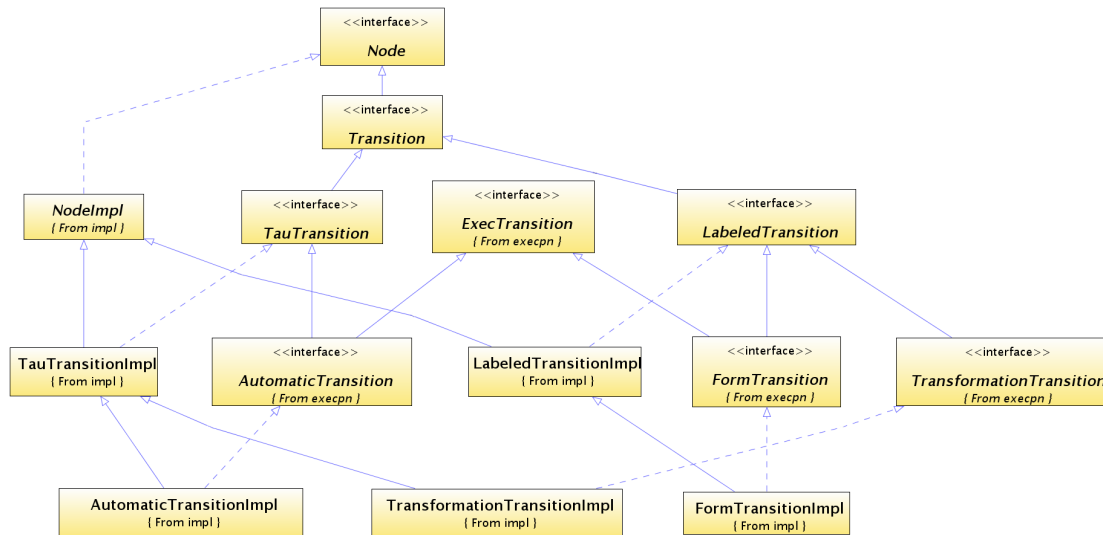


Abbildung 4.6: Klassendiagramm Transitionstypen

kopiert. Das funktioniert aber nur bei genau einer eingehenden Kante. Im Fall der Datenzusammenführung, bei mehreren eingehenden Kanten, muss es durch speziellere Transformationen behandelt werden.

Die `LabeledTransition` und alle Unterklassen, schalten auf Verlangen des Benutzers. Grundsätzlich werden auch hier zwei Arten unterschieden: `FormTransition` und `TransformationTransition`. Letztere ist der `AutomaticTransition` sehr ähnlich, nur wird sie nicht automatisch geschaltet, sondern durch Nutzerinteraktion. In `FormTransition` findet sich eine zentrale Einrichtung der Engine wieder. Es ermöglicht das Zuweisen von Formularen zu Transitionen. Das Schalten der Transition ist dann also gleichbedeutend mit dem Abschicken des Formulars.

Der Vorteil der Erstellung einer Datenstruktur, sowohl für BPMN als auch für Petrinetze, liegt darin, dass auf den Modellen Analysen betrieben werden können. So kann beispielsweise ein einfacher Syntax-Check prüfen, ob tatsächlich alle Verbindungen plausibel angegeben sind. Beispielsweise sollte ein Endevent einen eingehenden Sequenzfluss haben. Interessant wäre hier auch ein Soundness-Check, der die erzeugten Petrinetze überprüft. Weiterer Vorteil ist, dass der Umwandler durch dieses Design wesentlich besser zu erweitern ist, wenn mehrere Umwandlerimplementierungen das selbe Metamodell benutzen können und dieses nicht explizit Teil der Umwandlungslogik ist.

Der letzte Schritt der Umwandlung besteht in der Serialisierung der Petrinetz-Objektstruktur nach PNML. Dazu wird wieder unterschieden zwischen `ExecPNPML-`

`Exporter` und `PetriNetPNMLExporter`, die jeweils ein XML-Dokument mit oder ohne ausführungsspezifische Attribute erzeugen.

Das entstandene PNML-Dokument wird im Falle der Ausführung an die Engine weitergereicht und im Browser zur Anzeige gebracht.

4.4 Petrinetzengine

Ziel bei der Entwicklung der Engine war es, den ressourcenorientierten Architekturstil in Einklang mit prozessorientierten Informationssystemen zu bringen. Dabei lehnt sich das Design sehr stark an das Konzept an, wie es von R. Fielding in seiner Dissertation [Fie00] beschrieben wird. Dieses steht in engem Bezug zum HTTP-Protokoll.

REST

REST steht für Representational State Transfer und bezeichnet einen Architekturstil, der auf zustandsloser Kommunikation zwischen Client und Server basiert. Größtes Beispiel ist das World Wide Web selbst. Die Größe ist dabei auch das massivste Problem, das Fielding adressiert. Lösung sieht er in der Schaffung von Constraints:

- Ressourcenorientierung inkl. deren Repräsentation
- einheitliche Schnittstelle
- Client/Server (cacheable, stateless, layered)
- Hypermedia

'Anything you can find a noun for' kann nach Fielding als Ressource bezeichnet werden. Dabei kann es sich um die Entsprechung eines realen Objekts oder aber eines Businessobjekts handeln, das rein zur Verwaltung dient. Jede Ressource kann eine beliebige Anzahl an Repräsentationen haben. Beispielweise könnte ein Text als HTML-Seite gerendert werden, aber auch als PDF-Dokument oder sogar vorgelesen als MP3-Datei.

Die einheitliche Schnittstelle ist dabei schon auf Protokollebene – nämlich HTTP – gegeben. Entgegen einer serviceorientierten Schnittstelle, gibt es nicht wenige Endpunkte mit vielen Methoden, die an ihnen aufgerufen werden können, sondern potentiell viele Endpunkte mit einem festen Satz an Methoden. Diese Methoden definiert HTTP in RFC2616 [FGM⁺99] als: GET, POST, PUT, DELETE, OPTIONS, HEAD, TRACE, CONNECT. Die ersten 4 Methoden bilden bereits den Wortschatz, um eine Ressource hinreichend granular zu verwalten. Die Petrinetz-Ressource wurde realisiert, wie in Tabelle 4.7 beispielhaft beschrieben. Da die Petrinetz-Ressourcen

HTTP Methode	Bedeutung	Eigenschaften
GET	Petrinetzbeschreibung ausgeben	seiteneffektfrei, idempotent
POST	neues Petrinetz erstellen und URL zurückgeben	-
PUT	Petrinetz mit neuer Definition überschreiben	idempotent
DELETE	Petrinetz löschen	idempotent

Abbildung 4.7: REST-konforme Petrinetz-Schnittstelle

durch ebendiese uniforme Schnittstelle beschrieben sind, lässt sich mit einem HTTP-POST an eine festgelegte URL ein neues Netz einrichten. Auf gleiche Weise werden Transitionen, Tokens, Stellen usw. in der Engine verwaltet.

Die Client/Server-Architektur ist wesentlicher Bestandteil des WWW. Problematisch dabei ist, dass aber keinerlei Annahmen darüber getroffen werden können, was sich zwischen Server und Client befindet. Es sind beliebige Zwischenstufen denkbar, wie Caching Proxies oder Load Balancer. Deswegen definiert HTTP die Verben als atomare Operationen um unabhängig davon zu sein, welchen Weg die Aufrufe zum Server hin nehmen und nahezu unendliche Skalierbarkeit zu erreichen.

Das Prinzip der hypermedialen Darstellung von Ressourcen-Repräsentationen findet sich in HTML beispielsweise wieder. Es genügt eine URL der Engine und man kann darauf aufbauend jede weitere Repräsentation über Hyperlinks erreichen. Das Prinzip des 'URI-Passings' ist zudem in der Engine realisiert. Man kann eine URL verschicken, die genau auf eine Instanz eines Prozesses deutet und somit sich Prozessabläufe als Bookmark setzen.

Aufbau

Der Aufbau der Petrinetz-Engine ist in Abbildung 4.8 visualisiert. Entsprechend der REST-Architektur wurde die Repräsentation der Ressourcen auf unterschiedlichste Weise realisiert. So kommuniziert der Oryx mittels PNML mit der Engine, wenn Petrinetze deployed werden sollen. Eine weitere Engine würde mittels Atom Daten austauschen, um verteilte Prozessausführung zu realisieren. Der Standardfall, bei dem der Nutzer mittels Browser die Engine anspricht, gibt XHTML aus.

Als Webcontainer dient der Ruby-basierte Webserver Mongrel³. Als Webframework wurde Merb⁴ benutzt. Es wird ein striktes Modell-View-Controller Prinzip voraus-

³<http://mongrel.rubyforge.org/>

⁴<http://merbivore.com/>

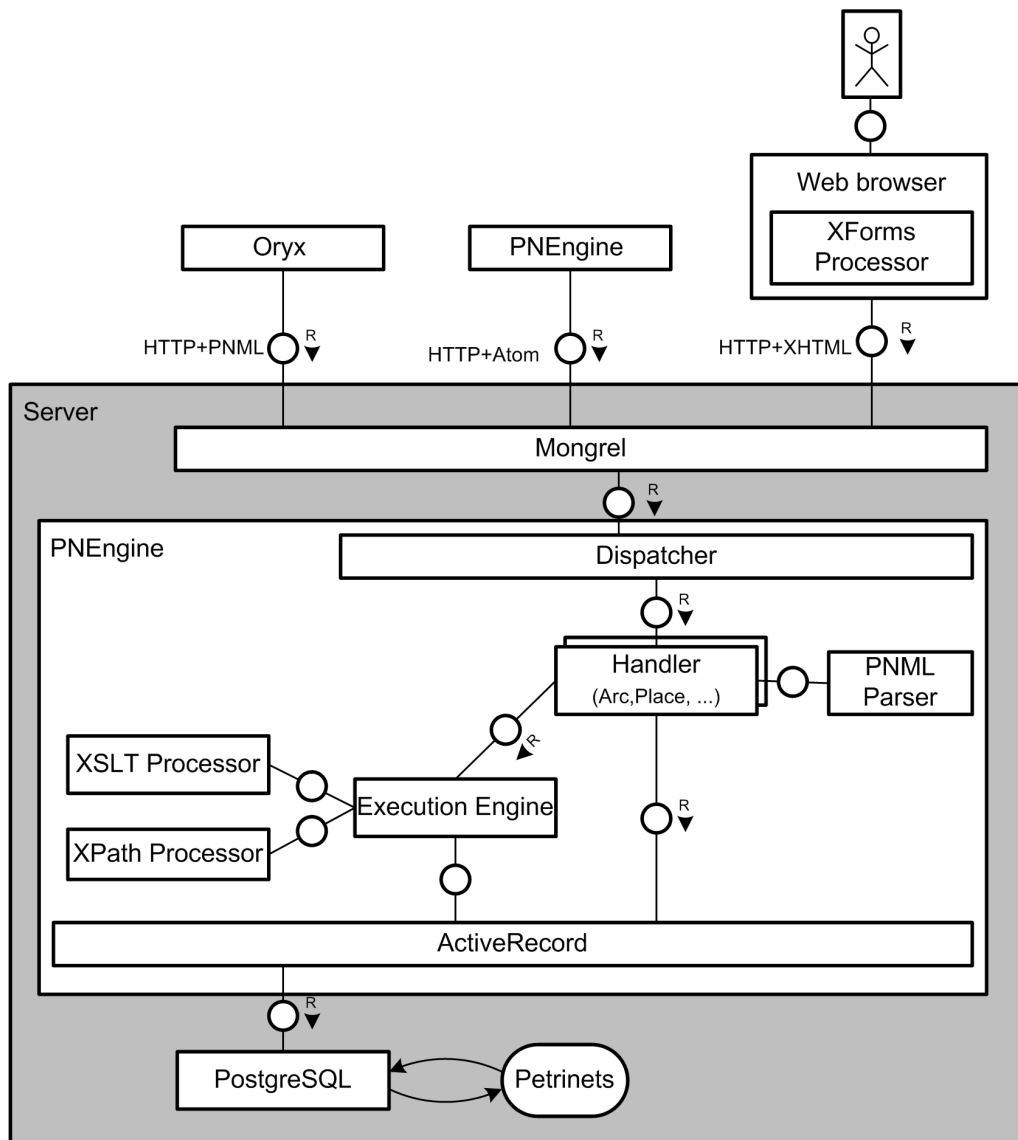


Abbildung 4.8: Architektur Petrinetzengine

gesetzt, das die Anwendungsteile in Datenmodell, Repräsentation und Steuerungskomponente aufteilt. Das Datenmodell wird mittels ActiveRecord – einer Objekt-relationalen Mapping-Bibliothek – persistent in einer PostgreSQL-Datenbank gespeichert. Im Controller-Teil werden Aktionen wie das Schalten von Transitionen erledigt. Ein PNML-Parser nimmt Petrinetze entgegen. Zusätzlich sind Prozessoren für XSLT und XPath eingebunden um Transformationen von Tokenwerten bzw. die Addressierung von Daten innerhalb von Token mittels Locators zu realisieren.

Funktionsweise

Die Hauptfunktion der Engine ist das Schalten von Transitionen. Die PNEngine kennt drei Arten von ihnen:

- AutomaticTransition
- ReceiveTransition
- SendTransition

AutomaticTransitions entsprechen genau dem, wie sie auch im Umwandler verwendet werden. Es ist eine Transition, die automatisch schaltet, sobald sie aktiviert ist. Hierbei kann der Token-Inhalt mittels XSLT transformiert werden. **Receive-Transitions** werden auf Verlangen des Nutzers geschaltet und können entweder formularbasiert oder zur automatisierten Datentransformation dienen. **SendTransitions** sind für die Interaktion der Engine nach außen verantwortlich und kann HTTP-Request mit der Umwelt anstoßen und auf die Antworten warten.

Problematisch ist es speziell bei den automatisch schaltenden Transitionen, wenn sie ohne eingehende Kanten eingesetzt werden und dementsprechend immer aktiviert sind. Deswegen werden Netze, wie sie der Umwandler erzeugt, typischerweise eine manuelle Transition ohne eingehende Kanten (das Startevent) haben, wodurch explizit durch den Nutzer neue Prozessinstanzen erzeugt werden können. Auch wird hier ein Problem der Petrinetze deutlich. Es gibt nicht direkt das Konzept einer Prozessinstanz. Dieses muss zusätzlich aus den Token interpretiert werden. Deswegen werden 'Cases' jetzt zusätzlich für jedes Token gespeichert. Die Case-Erstellung erfolgt dann, wenn ein Token von einer Transition erstellt wurde, die keine Eingangsstelle hat, also bei den Netzen aus dem Umwandler, jeweils das Startevent. Wenn eine Transition schaltet, die eingehende Stellen besitzt, dann wird auf die Ausgabtoken der eingehende Case kopiert.

Möglicherweise gibt es aber nicht genau einen eingehenden Case (Abbildung 4.9). Im Beispiel liegen jeweils zwei Cases an, somit gibt es 4 mögliche `token_combinations`. Demnach wäre in der Engine `transition.token_combinations` ein Array bestehend aus vier Arrays à zwei Token-Elementen. Diese Kombinationen sind die Einheit in der überprüft wird, ob eine Transition schaltbereit ist. Eine Bedingung basierend auf den anliegenden Cases, wird wie folgt überprüft:

```
def eval_guard(tokens)
  case_ids = []
  tokens.each do |token|
    if !token.case.nil?
      case_ids << token.case.id
    end
  end
  return false if case_ids.uniq.size > 1
  ...
end
```

Diese Überprüfung geschieht, bevor die Guard-Conditions kontrolliert werden, damit unnötige Combinations von vornherein ausscheiden. Von dieser Überprüfung wird auch der Sonderfall abgedeckt, wenn eine Kombination aus einem Token mit und einem ohne Case besteht, also beispielweise durch ein von außen ins System eingebrachtes Token. Die Regel ist hier, dass ein Token ohne Case sich mit allen Token eines beliebigen Cases vereinigen darf.

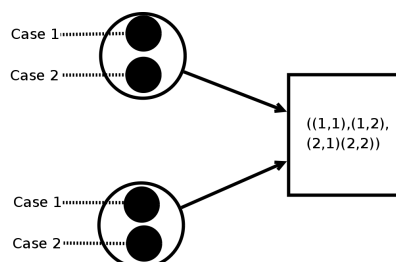


Abbildung 4.9: Tokencombinations

Anfänglich war das Case-Konstrukt nicht Teil der Engine, sondern wurde mittels Token-Werten und Guard-Conditions überprüft, jedoch wurde der Begriff des Cases zu einem essentiellen Konzept in der Verwaltung der Aufgaben, sodass es jetzt direkt in der Engine festgehalten ist.

Das Schalten einer Transition kann also mit einer `active_combination` erfolgen. Ob eine Kombination aktiv ist, wird in `eval_guard` überprüft. Die Guards, die dazu überprüft werden, werden in der PNML-Beschreibung angegeben. Guards sind boolesche Ausdrücke, die in Ruby-Syntax verfasst werden können. Mittels `eval`-Funktion werden diese Ausdrücke ausgewertet, wobei das – böswillige Absicht vorausgesetzt – das System kompromittieren könnte. Die Nutzung der Ruby Security Level bringt hier kein Mehr an Sicherheit, da hiermit nur die `eval`-Funktion vollständig verboten werden könnte. Es wird aber von Seiten A.Lüders und K.Schlichting weiter daran gearbeitet eine spezielle Guard-Syntax zu entwickeln, die diese Sicherheitsprobleme aus der Welt schafft.

Die Daten, die von Guards adressiert werden können, stammen aus den Nutzdaten der Token und werden mittels sogenannter Locators von Bezeichnern an XPath-Ausdrücke⁵, die auf das XML-Dokument der Token angewendet werden, gebunden. Durch das Konzept der Locators ist es möglich, die XPath-Abfragen zu cachieren und damit aufwendige Stringverarbeitungsroutinen zu minimieren. Ein Beispiel für einen Locator könnte in der PNML-Beschreibung wie folgt aussehen:

```
<locator>
  <name>isDelegated</name>
  <type>xsd:boolean</type>
  <expr>/data/metadata/isDelegated</expr>
</locator>
```

⁵<http://www.w3.org/TR/xpath>

Damit könnte eine Guard-Expression, die diesen Locator benutzt dann lauten: `[place_id].isDelegated == true`.

Wenn das Schalten der Transition erlaubt ist, dann werden die zu konsumierenden Tokens genommen, gegebenenfalls XSL-Transformationen angewandt und auf jede ausgehende Stelle ein Token produziert. Damit nicht in jedem Token der ausgehenden Stellen der gleiche Inhalt liegt, kann auf den ausgehenden Kanten eine Arc-Transformation hinterlegt werden, die beispielweise Metadaten von Prozessdaten trennt, wie es für die Kontextstellen benutzt wird.

4.5 Worklist

Die Engine wickelt unter Umständen sehr komplexe Modelle ab, die aus dem Umwandler erzeugt werden. Die Schaltsemantik muss von einem Menschen, der die Prozesse bearbeitet, nicht verstanden werden. Dazu soll es eine einfache Oberfläche geben, die die Bearbeitung von Aktivitäten gestattet: die Worklist.

Implementiert wurde die Worklist mit ExtJS. Dabei handelt es sich um eine JavaScript-Bibliothek, zur Darstellung von Formular- und Layoutelementen im Browser. ExtJS hegt dabei den Anspruch, das Look&Feel von Desktop-Anwendungen ins Web zu bringen. Es soll dabei über *'High performance, customizable UI widgets'* und ein *'Well designed, documented and extensible Component model'* verfügen.⁶

Benutzt wurde Version 2.0 der Bibliothek, die eine Reihe von Änderungen gegenüber 1.x mitbringt, wie zum Beispiel die Umstrukturierung der Layout Architektur (Abbildung 4.10). Unter ExtJS 1.x war man auf das BorderLayout fokussiert und alle anderen Layouts davon abgeleitet. Da in ExtJS 1.x die Layout-Logik beim BorderLayout selbst lag, ist es jetzt in die Zuständigkeit der einzelnen Layout-Klassen verschoben, wobei die reinen Container nichts über ihre Repräsentation wissen müssen. ExtJS 2.0 folgt damit dem sonst auch in allen Komponenten durchgehenden Component/Container-Prinzip, wobei Layout-Elemente dynamisch hinzugefügt oder entfernt werden können. Die Umsetzung eines solchen Layouts sieht man in der Worklist, wie in Abbildung 4.11 gezeigt.

Als umspannender Rahmen um die einzelnen Bereiche des Layouts wird ein `BorderLayout` benutzt. Dieses reglementiert die Darstellung in der Art der dreigeteilten Ansicht eines E-Mail-Clients. Im auf der linken Seite befindlichen Viewport wird ein `AccordionLayout` aufgespannt. Das macht es möglich, dass die Unterseiten *'Cases'*, *'Monitoring & Analysis'*, *'User Management'* und *'Spawn Processes'* durch animiertes Hochschieben voneinander getrennt sind. Diese Aufteilung orientiert sich an der Gruppierung eines Stencilsets im Oryx.

⁶<http://extjs.com/>

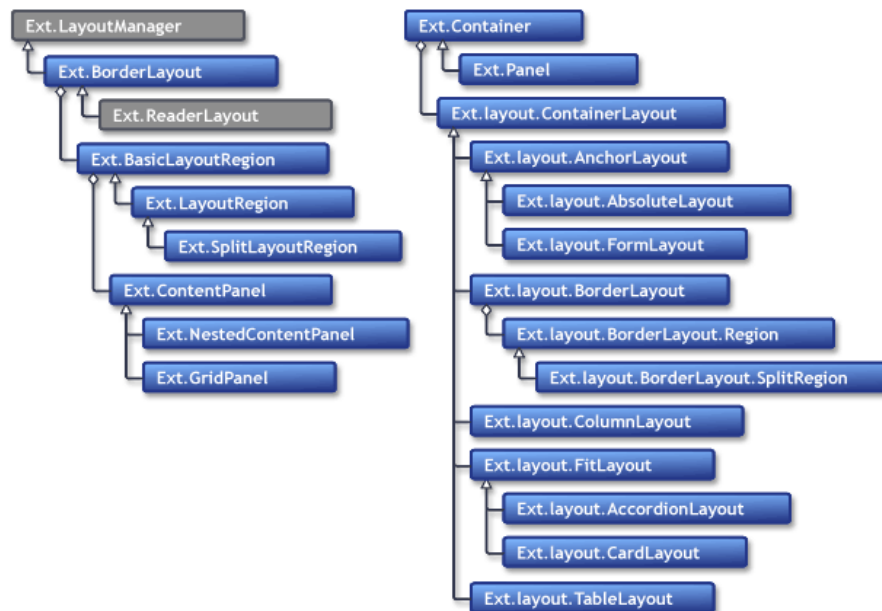


Abbildung 4.10: ExtJS Layout-Klassen Version 1.0/2.0

Die Caselist wird durch eine Toolbar nach unten begrenzt, in der durch acht Buttons die Aktionen der Tasks aufgerufen werden können. Diese werden je nach Status der Task als aktiv oder inaktiv markiert. Darunter befindet sich das eigentliche Bearbeitungsfenster. Dieses besteht aus einem IFrame, der die Formulare, wie sie durch die Engine ausgegeben werden zur Anzeige bringt. Problematisch dabei ist, dass diese Formulare potentiell von einem anderen Host als dem der Worklist ausgegeben werden.

Zur intuitiveren Bedienung wird bei einer Task, die als einzige Aktion ein Allocate erlaubt, direkt die Allocate-Transition in der Bearbeitungsansicht angezeigt. Weiterhin sind Aktionen ohne Formular, wie allocate, resume, suspend und skip durch Klick auf den jeweiligen Button direkt schaltbar. Wenn Submit aktiviert ist, wird diese Seite ebenso sofort aufgerufen, da es die potentiell wichtigste Aktion im Lifecycle darstellt. So braucht es im günstigsten Fall drei Klicks um den kompletten Lifecycle einer Task zu durchlaufen.

Innerhalb der Layoutregionen befinden sich sogenannte Grids. Dieses Konzept dient im einfachsten Fall zur Darstellung von Tabellen. Allerdings sind diese durch die Verwendung von Proxies, Readers, Stores, Column Models, Selection Models und Renderer sehr generisch und flexibel benutzbar. Ein Proxy kann zur Umgehung der same-origin-policy genutzt werden um Daten über den gleichen Server verfügbar zu machen, der auch das JavaScript-Dokument ausliefert. Reader sind dazu da, Daten aus diversen Quellen zu lesen, im vorliegenden Beispiel ist das ein XML-Dokument.

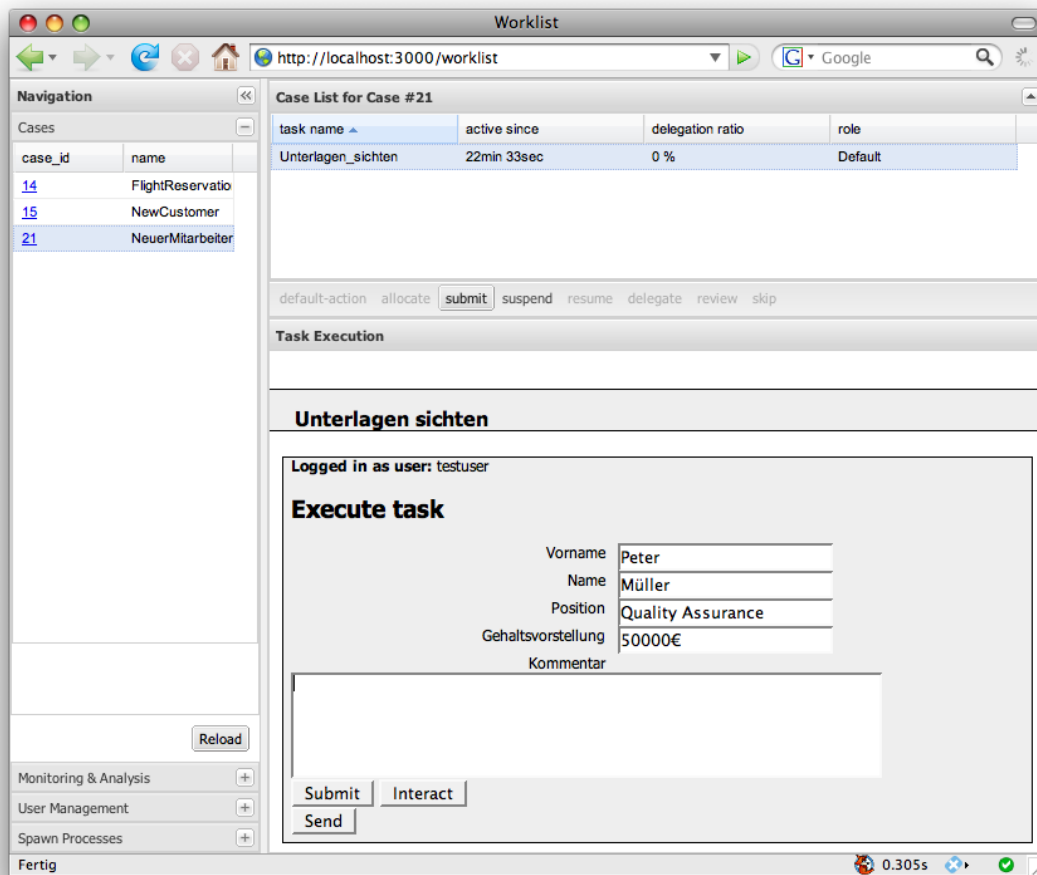


Abbildung 4.11: Worklist

Ein Store übernimmt das Client-seitige Caching der Daten. Ein Column Model wird genutzt, um die Datensätze des Stores auf Spalten zu mappen. Das Selection Model wird für Selektionsvorgänge innerhalb eines Grids benutzt. Dabei wird standardmäßig unterschieden zwischen zellen-, spalten- oder reihenweisen Selektion. Das Konzept des Renderers dient dazu, die Werte einer Zelle mit Formatanweisungen auszustatten, um die reinen Daten mit einem gewissen semantischen Mehrwert anzureichern (z.B. Delegation Ratio soll rot dargestellt werden, wenn ein Wert über 50% erreicht wird).

Zur Illustration dieser Konzepte soll folgendes (gekürztes) Code-Beispiel dienen:

```
Ext.store_overview = new Ext.data.GroupingStore({
  url: '/overview',
  groupField: 'net_id',
  sortInfo: {field: 'case_id', direction: 'ASC'},
```

```

    reader: new Ext.data.XmlReader({
        record: 'instance',
        ...
    }, [
        'net_id', 'case_id', 'transition_count', ...
    ])
});

Ext.overview_grid = new Ext.grid.GridPanel({
    store: Ext.store_overview,
    columns: [
        {id: 'case_id', header: "case_id",
         sortable: true, dataIndex: 'case_id'},
        ...
        {header: "duration", dataIndex: 'duration',
         renderer: pendingTimeRenderer},
        ...
    ],

    view: new Ext.grid.GroupingView({
        forceFit: true,
        groupTextTpl: '{text} ({[values.rs.length]}
        {[values.rs.length > 1 ? "Items" : "Item"]})'
    }),
    ...
});

```

Hierbei wird eine Kombination aus Grouping-Store und -View benutzt. Diese hat die zusätzliche Fähigkeit, dass sie Daten anhand einer vorgegebenen Spalte gruppiert. Die Gruppierung resultiert in einer aufklappenden Ansicht wie in Abbildung 5.1. Da die Worklist vom selben Host wie die Engine ausgeliefert wird, kann hier auf die zusätzliche Proxy-Schicht verzichtet werden. Sollte die Worklist beispielsweise in die Oryx-Oberfläche integriert werden, so müsste ein ProxyServlet oder ähnliches benutzt werden (das zu Testzwecken auch schon im Oryx-SVN vorliegt). Die Darstellung findet in einem `GridPanel` statt, das als Rahmen dient und den `GroupingView` nutzt, um die einzelnen Zellen darzustellen. Eine weitere Besonderheit der Gruppierung ist das Template, das zur Formatierung der Kopfzeile eines jeden gruppierten Eintrags dient. Das Konzept des `renderers` wurde hier ebenso benutzt. Es kann eine JavaScript-Methode angegeben werden, die den Wert entgegennimmt und einen formatierten String zurückgibt. In dem Beispiel würde '123.45' als `pending_time` in '2min 3sec' nach Bearbeitung durch den `pendingTimeRenderer` resultieren. Dieser rundet die Werte auf volle Sekunden, Minuten, Stunden und Tage.

Durch die Flexibilität des Layoutings in ExtJS kann die Worklist schnell um zusätzliche Anzeigen oder Funktionalitäten erweitert werden. Eine interessante Erweiterung wäre, dass Cases nicht anonym durch eine Nummer repräsentiert werden, sondern tatsächlich Daten, die im Prozess eingegeben wurden, angezeigt werden. Beispiels-

weise wäre es denkbar, dass ein Ausdruck bei der Modellierung angegeben wird der beim Bewerbungsprozess auf den Namen des Bewerbers zeigt.

5. Analyse und Monitoring

Die Worklist in der existierenden Version ist als Oberfläche gedacht, die dem Nutzer auf einfache Weise die Bearbeitung von Fällen ermöglicht. Um die Arbeit weiterhin intuitiver zu gestalten, wurden zusätzliche Informationen eingepflegt, die auf historischen Daten Analysen betreiben und die ermittelten Werte zur Anzeige bringen. Dieser Abschnitt soll zudem einen sehr kurzen Ausblick darüber geben, was durch die Oryx-Plattform in Zukunft geleistet werden könnte und wo die Potentiale stecken, die durch Mining ergründet werden können.

5.1 Integrierte Analysen

Die Worklist soll zusätzlich zu der reinen Aufgabenbearbeitung auch Oberfläche für Prozessdesigner oder Administratoren sein, mögliche Flaschenhalse aufzuspüren. Dazu wurde die Task-Darstellung mit statistischen Informationen angereichert, wie schon in 4.5 zu sehen. So kann der Nutzer direkt von den zusätzlichen Informationen profitieren, die durch Analyse der vorangehenden Prozessinstanzen entstanden sind. Hilfreich wäre beispielsweise zu sehen, wie lange eine Task schon auf ihre Bearbeitung wartet oder wieviele Mitarbeiter vor ihm diese Task delegiert haben.

Weiterhin wurde eine Übersicht für den Prozesseigner geschaffen (Abbildung 5.1). Damit soll ein Gesamtüberblick geschaffen und beantwortet werden, wie oft welche Prozesse durchlaufen werden und wie lange dafür gebraucht wird.

Zur Beantwortung dieser Fragestellungen wurde ein zusätzliches Log eingeführt. Hier werden genau drei Aktionen erfasst: Transition aktivieren, automatisch schalten oder manuell schalten. Zudem wird zu jedem dieser Ereignisse die Case-ID, der Benutzer, die Transitions-ID usw. erfasst. Ein Beispiel ist in Tabelle 5.1 aufgeführt. Realisiert

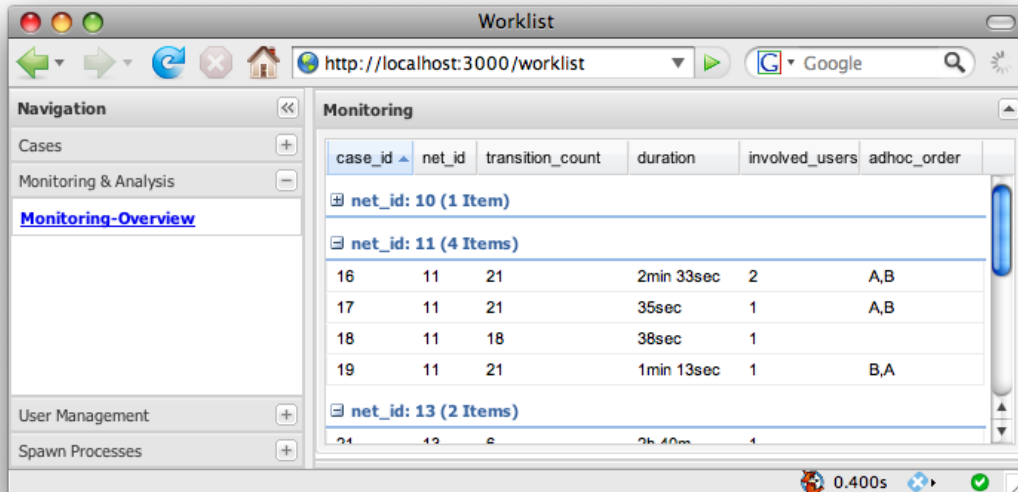


Abbildung 5.1: Monitoring-Ansicht

action	time	user	task_id	task_name	op	case_id	net_id
manual_fire	Fri Jun 13 11:38:21 2008	testuser	304	resource0		8	4
activate	Fri Jun 13 11:38:22 2008	testuser	308	transition_tr_init_resource1		8	4
automatic_fire	Fri Jun 13 11:38:22 2008	testuser	308	transition_tr_init_resource1		8	4
activate	Fri Jun 13 11:38:22 2008	testuser	341	transition_tr_enable_resource1		8	4
automatic_fire	Fri Jun 13 11:38:23 2008	testuser	341	transition_tr_enable_resource1		8	4
activate	Fri Jun 13 11:38:23 2008	testuser	290	Unterlagen_sichten	allocate	8	4
manual_fire	Fri Jun 13 11:39:02 2008	testuser	290	Unterlagen_sichten	allocate	8	4

Tabelle 5.1: Ausschnitt des Logs – Ausführung Ad-Hoc Subprozess

wurde das Log mit einer CSV-Datei, die zusätzlich zum schnelleren Zugriff permanent im Speicher gehalten wird. Viele der Anforderungen, die sich aus Analysefragestellungen ergeben, sind mehr oder weniger komplexe Anfragen an eine Datenbasis. Statt eine Datenbank zu benutzen (was bei größeren Datenmengen möglicherweise sinnvoll wäre), wurde dabei auf die sehr flexiblen Ruby-Collection Methoden zurückgegriffen. Einige der Analysefragestellungen sind:

Delegation ratio

Um dem Bearbeiter zum einen einen Anhaltspunkt zu geben, ob er in einer unsicheren Situation vielleicht delegieren sollte und dem Prozessdesigner zum anderen, der eventuell über Nachbesserung am Modell entscheiden kann, wurde der 'delegation ratio' eingeführt. Dieser wird in der Worklist zu jeder Task angegeben. Die Anzahl der bisher gelaufenen Instanzen einer Task, bei denen eine Delegation stattgefunden hat, wird zu der Anzahl aller Instanzen der Task ins Verhältnis gesetzt. Der Quotient daraus ist der 'delegation ratio'.

Active since

Dies ist der Zeitabschnitt, den eine Task schon als aktiv gekennzeichnet ist, d.h. wann das erste mal eine Transition aktiviert wurde, die zu der Gruppe von Transitionen gehört, die die Task repräsentieren. Es stellt also die Zeit dar, wie lange eine Aufgabe schon unerledigt in der Worklist liegt. Diese Abfrage wird auf ähnliche Weise auch auf Prozessinstanzen benutzt, die bereits abgeschlossen sind. Hier wird die Zeit zwischen letztem und erstem Event ausgegeben und als `duration` in der Monitoring-Overview angezeigt.

Ad-Hoc Order

Durch die feste Benennung der Transitionen, die einen Ad-Hoc-Subprozess aktivieren, kann die Reihenfolge der Ausführung der Aktivitäten pro Instanz angegeben werden. Diese Analyse kann einem Administrator dazu dienen, das Prozessmodell noch einmal zu überdenken und beispielsweise eine Sequenz statt einem Ad-Hoc-Prozess einzuführen.

5.2 Weitergehende Analysen

Die bisherigen Analysefragestellungen konzentrieren sich nur auf sehr lokale Probleme, wie die Laufzeit eines Prozesses oder den `delegation ratio`. Um diese Aufgaben zu bewältigen, kann das Log als einfache Abfragebasis genutzt werden, ohne eine Gesamtsicht auf den Prozess herzustellen. Um ebendiese Gesamtsicht zu gewinnen und aus historischen Daten neue – eventuell bessere – Prozessmodelle zu gewinnen, muss Process Mining betrieben werden. Es war nicht Ziel, im Rahmen dieses Projektes Mining-Funktionalität umzusetzen. Vielmehr soll hier ein Ausblick gegeben werden, wie ein externes Tool für diese Aufgabe genutzt werden kann.

Das erzeugte Log kann dazu durch eine zusätzliche Export-Funktion in MiningXML (MXML) ausgegeben werden. Eine exemplarische Ausgabe der Logdaten im MXML-Format zeigt die Ähnlichkeit zur CSV-Datei.

```
<?xml version="1.0" ?>
<WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
"http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd">
  <Process id="6">
    <ProcessInstance id="5">
      <AuditTrailEntry>
        <WorkflowModelElement>resource0</WorkflowModelElement>
        <EventType></EventType>
        <Timestamp>Tue Jun 10 17:22:18 +0200 2008</Timestamp>
        <Originator>test</Originator>
      </AuditTrailEntry>
    </AuditTrailEntry>
  </ProcessInstance>
</Process>

```

```

    <WorkflowModelElement>A</WorkflowModelElement>
    <EventType>allocate</EventType>
    <Timestamp>Tue Jun 10 17:22:34 +0200 2008</Timestamp>
    <Originator>test</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <WorkflowModelElement>A</WorkflowModelElement>
    <EventType>submit</EventType>
    <Timestamp>Tue Jun 10 17:22:38 +0200 2008</Timestamp>
    <Originator>test</Originator>
  </AuditTrailEntry>
  ...

```

Mit diesem MXML-Dokument wird es ermöglicht, die Analyse und Mining-Funktionalität in ProM¹ zu nutzen. Dieses an der TU Eindhoven entwickelte Framework, bietet eine Fülle an Funktionen, wie "control-flow mining, performance analysis, resource analysis, conformance checking, verification"².

Im Folgenden veranschaulicht ein einfaches Beispiel zwei der Algorithmen, die ProM zur Verfügung stellt. Das Beispiel ist ein Prozess mit einem Ad-Hoc-Subprozess, der zwei Tasks (A,B) beinhaltet. Dieser Prozess wurde für das Test-Log 4 Mal durchlaufen, einmal in der Reihenfolge $B \rightarrow A$ und dreimal $A \rightarrow B$. Aus dem Task-Lifecycle wurden nur allocate und submit ausgeführt. Die erzeugten Modelle geben die durch Mining entstandenen Prozessmodelle wieder und geben einen Eindruck von der Oberfläche von ProM. Zum Test wurden folgende Algorithmen benutzt: Alpha, Alpha++ und ein Fuzzy Miner [vdAG07].

Der Alpha-Algorithmus [vdAWM04] ist dazu gedacht, alle Abläufe, die im Log aufgezeigt werden, im erzeugten Petrinetz wiederzugeben. Die Traces, die der Alpha-Algorithmus analysiert, entsprechen den geloggtten Cases, also den erzeugten Prozessinstanzen und ihrem jeweiligen Ablauf. Die Ausgabe (Abbildung 5.2), die das Ergebnis des Alpha++-Algorithmus auf das Log inklusive automatischer Transitionen darstellt, ist sehr genau das, was auch als Grundlage der Prozessausführung aus dem Umwandler deployed wurde. Selbstverständlich fehlt hier die Kontextstelle, die ja nicht als aktiver Teil des Prozessablaufes sichtbar wird. Weiterhin würden automatisches Skipping innerhalb von Ad-Hoc Subprozessen bei erfolgreicher Completion Condition nur dargestellt, wenn diese auch ausgeführt werden. Ein Merkmal, das gar nicht generiert werden kann, sind Guard Conditions.

Die Ergebnisse von Alpha und Alpha++ unterscheiden sich in diesem einfachen Beispiel erwartungsgemäß nicht. Es sind keine Problemfälle, wie zum Beispiel kur-

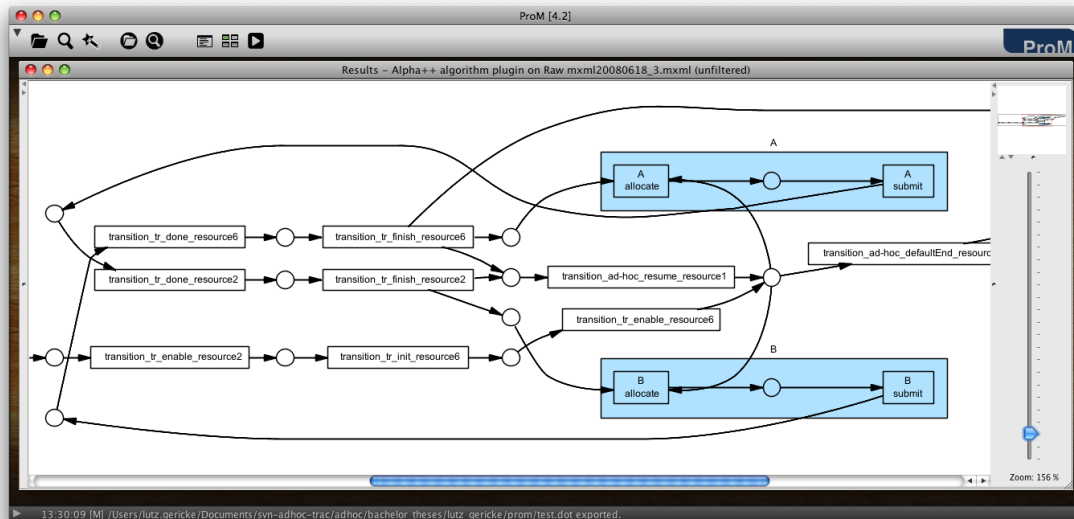


Abbildung 5.2: Ergebnis Alpha++ Algorithmus in ProM

ze Loops, in dem Beispiel vorhanden, die von Alpha++ zusätzlich erkannt werden würden.

Das Fuzzy Mining (Abbildung 5.3) gibt einen zusätzlichen Blickwinkel auf die Loganalyse. So kann durch unterschiedliche Werte beim 'Significance cutoff' eingestellt werden, wieviele Details des Logs wiedergegeben werden sollen. In der Abbildung dargestellt sind von links nach rechts immer kleinere Werte, also größere 'Genauigkeit' mit der das Modell erstellt wird. Dabei fällt auf, dass ein Hauptmerkmal des Prozesses relativ genau abgebildet wird. So wird der Ad-Hoc Subprozess so abgebildet, dass er durch Start und Ende abgegrenzt ist. Dazwischen findet das Fuzzy Mining eine starke Varianz der Abfolgen.

Diese Ansätze werfen vor dem Hintergrund der Ad-Hoc Subprozesse Fragen auf. So könnte es möglich werden, einen Grad der Varianz auszumachen, ab dem es sinnvoller ist, einem Ad-Hoc Subprozess die explizite Modellierung vorzuziehen.

¹<http://www.processmining.org/>

²<http://tabu.tm.tue.nl/wiki/publications/bnaic2007>

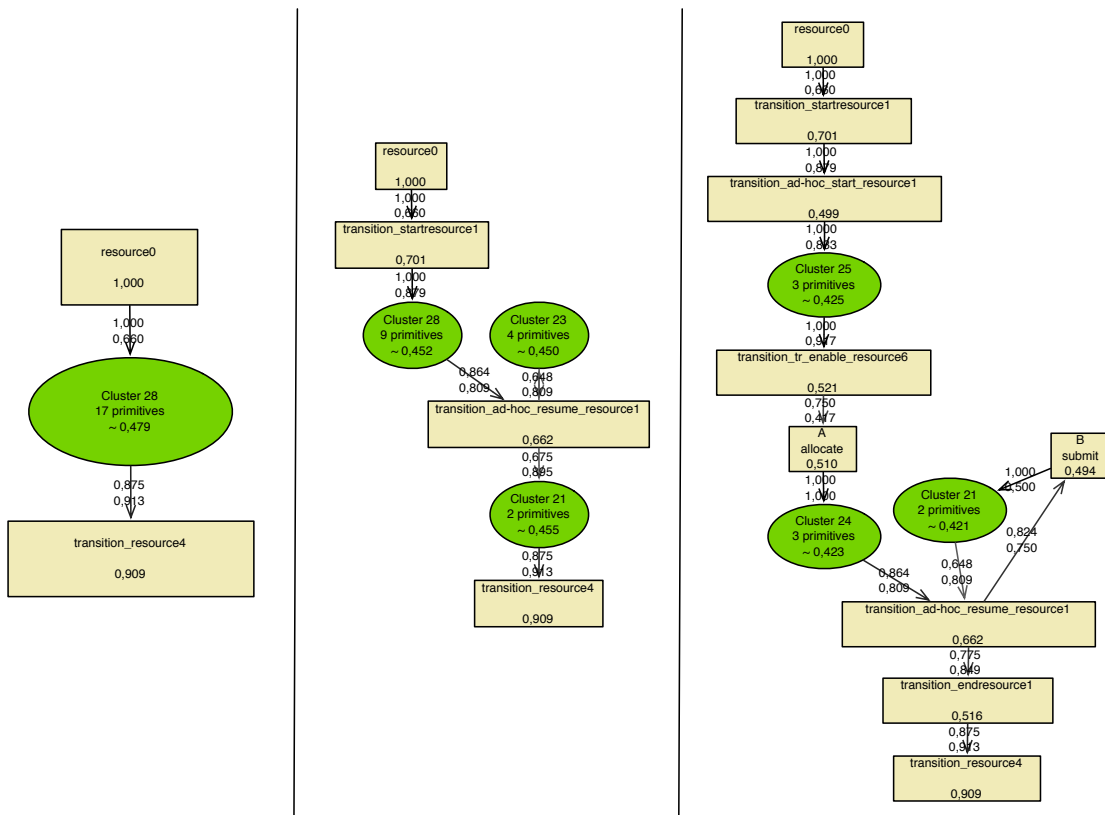


Abbildung 5.3: Ergebnisse Fuzzy Mining

6. Zusammenfassung

Viele Entscheidungen, die während der Entwicklung getroffen wurden, haben sich letzten Endes als richtig herausgestellt. Die Anforderungen, die an das System gestellt wurden, sind erfüllt worden.

Zwar war die Aufteilung der Zuständigkeiten zwischen den Komponenten nicht immer einfach, wurde aber durch die Architektur erzwungen und konnte so konsequent realisiert werden. So sind durch die Schaffung von wohldefinierten Schnittstellen (siehe 4.1) die Verantwortlichkeiten klar getrennt. Die Konsequenz in der Aufteilung hat aber auch seine Schattenseite. So ist es nicht möglich, dass die Worklist auf Ereignisse reagiert und daraufhin beispielsweise Schaltflächen in der Oberfläche ein-/ausblendet. Durch die Verwendung eines Pull-Verfahrens müsste die Engine ständig neu angefragt werden, um auf Änderungen zu reagieren.

Ebenso in der Architektur begründet liegt die Möglichkeit schnell Änderungen am Modell in der Laufzeit zu reflektieren. Gerade dadurch ist die Entwicklung am Task-Lifecycle bzw. auch am BPMN2PN-Umwandler effektiv möglich. Durch diese Entwicklungen war es beispielsweise erst möglich, schnell zu prototypischen Teilanwendungen zu kommen, die dann nach Tests in das Standardrepertoire des Umwandlers übernommen werden konnten.

Der entwickelte Lifecycle selbst, wie er im Umwandler implementiert ist, ermöglicht es, die Konzepte generalisiert zu betrachten, da für potentiell jede Task die Möglichkeit geschaffen wird, sie zu delegieren oder zu skippen. Diese Flexibilität ist auch in der Worklist visualisiert und offenbart die gleiche Schnittstelle und ein einheitliches Look&Feel für alle Aktivitäten. Die Kontextstelle – als Teil des Lifecycles – hat es erst ermöglicht, dass eine Completion Condition auswertbar ist.

Ein Nachteil davon ist, dass der Tokeninhalt, speziell bei den Kontextstellen, im Laufe der Implementierungsarbeiten recht stark angewachsen ist. Viele Verwaltungs-

informationen werden gerade in den Kontextstellen-Token vorgehalten. Da für die Reevaluierung auch historische Daten aufbewahrt werden, wächst die Datenmenge kontinuierlich an. In zukünftigen Versionen sollte hier eine Lösung geschaffen werden, die nicht mehr benötigte Daten ausfindig macht und löscht.

Erweiterungen/Ausblick

Gerade durch den prototypischen Charakter geprägt, war Performance vor allem bei der Entwicklung der Engine kein primäres Ziel. Hier könnte an vielen Stellen noch verbessert werden. Die Konzepte sind klar herausgearbeitet worden, allerdings offenbart die Umsetzung unter Umständen, gerade bezüglich der Fehlerbehandlung und Verarbeitungsgeschwindigkeit, noch Potential zur Optimierung.

Die Umwandlerkomponente erfüllt für einfache Prozessmodelle die gestellten Anforderungen, allerdings gibts es noch einige BPMN-Konzepte, die so noch nicht vollständig implementiert wurden. Dazu zählt der nicht unproblematische OR-Gateway. Wichtig wird hier vor allem die geeignete Darstellung im User Interface sein. SAP Galaxy hat diese Problematik bisher so beantwortet, dass OR-Joins nicht zu modellieren sind. Boundary Events, wie sie angelehnt an [DDO08] im Umwandler implementiert sind, sollten im Rahmen der Ausführung angestoßen werden können. Beispielsweise könnten Timer Events, gekoppelt an Subprozesse ein interessantes Feature sein. So könnte ein Ad-Hoc Subprozess automatisch abgebrochen werden, wenn er zu lange unbearbeitet bleibt.

Interessant ist unter Umständen auch das zur Ausführen von weiteren im Oryx modellierbaren Modellarten. Möglich wäre hier die Umwandlung von Ereignisgesteuerten Prozessketten in Petrinetze. Auch denkbar ist die Unterstützung von Methoden zum Mining von Prozessmodellen innerhalb der Oryx-Plattform ohne den Umweg eines externen Tools.

Literaturverzeichnis

- [BCvH⁺03] Jonathan Billington, Søren Christensen, Kees M. van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *24th International Conference on the Applications and Theory of Petri Nets (ICATPN)*, LNCS, pages 483–505, Eindhoven, The Netherlands, June 2003.
- [Czu07] Martin Czuchra. Oryx - Embedding Business Process Data Into the Web. Bachelor's Thesis, Hasso Plattner Institute at the University of Potsdam, July 2007.
- [Dav06] Ian Davis. An Introduction to Embedded RDF. In *XTech 2006*, 2006. <http://research.talis.com/2005/erdf/xtech2006.html>.
- [DB04] Brian McBride Dan Brickley, R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, W3C, February 2004.
- [DDO08] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology (IST)*, 2008.
- [DGKW08] Gero Decker, Lutz Gericke, Stefan Krumnow, and Mathias Weske. Prozessmodellierung und -ausführung im Web. Technical report, Hasso-Plattner Institute at the University of Potsdam, Potsdam, 2008.
- [FGM⁺99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, The Internet Engineering Task Force, 1999. <http://www.ietf.org/rfc/rfc2616>.
- [Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair-Richard N. Taylor, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

- [Jen96] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Volume 1*. Springer, 1996.
- [Kle08] Matthias Kleine. Extending Galaxy Composer with Ad-Hoc Constructs. Bachelor's Thesis, Hasso-Plattner Institute at the University of Potsdam, June 2008.
- [Kog08] Alexander Koglin. Scenarios, Usability and XForms in Ad-Hoc Business Processes. Bachelor's Thesis, Hasso-Plattner Institute at the University of Potsdam, June 2008.
- [Kru08] Stefan Krumnow. Modeling and Executing Ad-Hoc Subprocesses in Different Environments. Bachelor's Thesis, Hasso-Plattner Institute at the University of Potsdam, June 2008.
- [Mas08] Philipp Maschke. Execution and Re-evaluation of BPMN Processes. Bachelor's Thesis, Hasso-Plattner Institute at the University of Potsdam, June 2008.
- [Nag08] Mike Nagora. Roles and Delegation in Ad-Hoc Business Processes. Bachelor's Thesis, Hasso-Plattner Institute at the University of Potsdam, June 2008.
- [Pet62] Carl Adam Petri. *Communication with Automata (in German)*. PhD thesis, Universität Bonn, Institut für Instrumentelle Mathematik, Schriften IIM Nr.2, 1962.
- [SSO01] Shazia W. Sadiq, Wasim Sadiq, and Maria E. Orłowska. Pockets of Flexibility in Workflow Specification. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 513–526, London, UK, 2001. Springer-Verlag.
- [vdAG07] W.M.P. van der Aalst and C.W. Günther. Finding structure in unstructured processes: The case for process mining. *Proceedings the 7th International Conference on Applications of Concurrency to System Design (ACSD 2007)*, pages 3–12, 2007.
- [vdAWM04] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, 2004.
- [Wes07] Mathias Weske. *Business Process Management*. Springer, 2007.
- [xfo07] XForms 1.0 (Third Edition). Technical report, W3C, 2007. <http://www.w3.org/TR/xforms/>.