Diplomarbeit

# Softwarekomponenten zur Definition und Ausführung von Geschäftsprozessen in Informationsportalen

Jan Eelbo
Matrikelnummer: 12160
Studiengang Informatik-Ingenieurwesen
Technische Universität Hamburg-Harburg

Hamburg, im Juni 2004

Erstbetreuung
Prof. Dr. Joachim W. Schmidt
Arbeitsbereich Softwaresysteme
Technische Universität Hamburg-Harburg

Zweitbetreuung
Prof. Dr. Ralf Möller
Arbeitsbereich Softwaresysteme
Technische Universität Hamburg-Harburg

TUHH
Technische Universität Hamburg-Harburg

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

## Danksagung

Ich möchte mich für das interessante Thema und die sehr gute Betreuung am Arbeitsbereich Softwaresysteme bedanken. Insbesondere danke ich Herrn Prof. Dr. Joachim W. Schmidt und Herrn Prof. Dr. Ralf Möller für die Betreuung der Arbeit. Ganz besonderer Dank geht an Patrick Hupe, der mich während der Anfertigung der Arbeit persönlich betreut hat.

# Kapitel 1

# Einleitung

In den letzten Jahren sind die Möglichkeiten der Ausführung von Geschäftsvorgängen mithilfe von Computersystemen aufgrund der stetig steigenden Leistung der Systeme und der in Forschungsarbeiten entwickelten Konzepte größer geworden. Zur Verwaltung und Ausführung solch automatisierter Prozesse durch Computersysteme werden Workflow-Management-Systeme eingesetzt. Dabei wurden in den letzten zehn Jahren in diesem Bereich Standards durch die Workflow Management Coalition [WfMC] definiert, die immer mehr Akzeptanz finden. So wurde ein Referenzmodell definiert, welches den allgemeinen Aufbau von Workflow-Management-Systemen beschreibt [WfMC95].
Prozessunterstützungssysteme werden allgemein als Untermenge von Workflow-Management-Systemen betrachtet [Ahm03]. Sie verfügen nicht über den vollen Umfang der von der WfMC definierten Funktionen und Schnittstellen. Sie werden zum Ausführen und Verwalten von Prozessen eingesetzt, die aus einer Reihe von manuellen oder automatisierten Aufgaben bestehen, die entweder durch eine Person oder einen Computer ausgeführt werden. Auf solche Workflow-Mangement-Systeme werden die Geschäftsprozesse eines Unternehmens abgebildet und somit von den verwendeten Anwendungsprogrammen entkoppelt. Der Ablauf und Fortschritt eines Prozesses wird separat durch eine eigenes System gesteuert und nicht durch die Anwendungsprogramme. Dadurch können Geschäftsprozesse flexibler modelliert werden und sind besser zu kontrollieren.
Auf der anderen Seite werden Portale in den letzten Jahren vermehrt in Unternehmen eingesetzt. So werden Informationsportale in vielen Unternehmen eingesetzt, um in Unternehmen vorhandenes Wissen und Informationen aus verschiedenen Medien zu kategorisieren und für die Mitarbeiter oder Kunden des Unternehmens verfügbar zu machen. Die dabei durch das Portal bereitgestellten Informationen und Dienste werden von den Mitarbeitern bei der täglichen Arbeit genutzt. Diese Informationen sollen Kunden und Mitarbeitern gezielt im Arbeitskontext bereitgestellt werden. Zu diesem Zweck kann ein Prozessunterstützungssystem eingesetzt werden.
Eine Plattform zur Realisierung von Informationsportalen ist die infoAssetBroker-Plattform [IA04]. Im Rahmen dieser Arbeit soll ein Prozessunterstützungssystem in diese Portal-Plattform integriert werden, um die Dienste eines Prozessunterstützungssystems bei der Realisierung von Portalen mit dieser Plattform nutzen zu können. Im Fokus liegen die Bereitstellung von Informationen und Diensten zur Unterstützung von Geschäftsprozessen, sowie ihre Verknüpfung zur Prozess-Identifikation und Optimierung.

## 1.1 Ziele der Arbeit

In der vorliegenden Arbeit sollen die fachlichen Anforderungen an ein Prozessunterstützungssystem zusammengetragen werden. Dazu werden unter anderem zwei Beispielprozesse hinsichtlich der durch ein Prozessunterstützungssystem zu unterstützenden Konzepte untersucht. Bei den beiden Beispielprozessen handelt es sich um den Studienarbeiter-Prozess, der am Arbeitsbereich Softwaresysteme der Technischen Universität Hamburg-Harburg [Sts04] definiert wurde und um den Schiffsbesichtigungs-Prozess, der im Rahmen des Forschungsprojektes WIPS IT1.1 [Wips04] zwischen dem Germanischen Lloyd [Gl04] und dem Arbeitsbereich Softwaresysteme definiert wurde.
Aufgrund der Anforderungsanalyse soll dann ein konkretes Prozessunterstützungssystem für die infoAssetBroker-Plattform entworfen und realisiert werden, wobei auf den Ergebnissen der Arbeit [Ahm03], in der ein Prototyp eines Prozessunterstützungssystems entworfen und realisiert wurde, aufgebaut werden soll.
Mit dem in dieser Arbeit realisierten Prozessunterstützungssystem soll dann ein Beispielprozess umgesetzt werden, um die Funktion des Systems zu demonstrieren und zu evaluieren.

## 1.2 Gliederung

In Kapitel 2 wird in die thematischen Grundlagen dieser Arbeit eingeführt, wobei zunächst der Begriff des Portals sowie die verschiedenen Portalarten erläutert werden. Anschließend folgt eine kurze Einführung der infoAssetBroker-Plattform, mit der die Architektur und wichtige Konzepte vorgestellt werden. Zum Abschluss wird die Motivation für die Integration von Prozessunterstützungssystemen erläutert.
Kapitel 3 stellt zwei existierende Arbeiten auf dem Gebiet der Prozessunterstützungssysteme vor: Den in der Studienarbeit von Vanda Lehel [Leh01] entwickelten Job-Dienst und den von Ayaz Ahmed in Rahmen seiner Master Thesis [Ahm03] realisierten Prototyp für ein Prozessunterstützungssystem.
Die Analyse der beiden erwähnten Beispielprozesse wird in Kapitel 4 beschrieben. Die beiden Prozesse werden kurz vorgestellt und die identifizierten Konzepte als Anforderungen zusammen getragen.
Kapitel 5 beschreibt die fachlichen Anforderungen an ein Prozessunterstützungssystem, wobei die bereits im Kapitel 4 identifizierten Konzepte noch einmal kurz erwähnt werden. Im Wesentlichen werden hier die allgemeinen Anforderungen betrachtet und diese als Anwendungsfälle beschrieben.
In Kapitel 6 wird der Entwurf und die Realisierung des Prozessmodells sowie der Prozessmaschine für das Prozessunterstützungssystem behandelt. Es werden der Entwurf und die Realisierung des Ereignismodells beschrieben, welches als Resultat der Anforderungsanalyse in dieser Arbeit definiert und realisiert wurde.
Kapitel 7 behandelt den Entwurf und die Realisierung der Softwarekomponenten zur Definition und Verwaltung von Prozessen. Im Rahmen dieses Kapitels wird beschrieben, wie ein externes Modellierungswerkzeug für UML-Diagramme zur Definition von Prozessen in die Portalplattform integriert und eingesetzt wird.
In Kapitel 8 werden der Entwurf und die Realisierung der Softwarekomponenten zur Prozessverwaltung behandelt, wobei zwischen der Verwaltung von laufenden Prozessen, der Konfiguration von Prozessdefinitionen und der Prozessüberwachung unterschieden wird.
Nachdem in den vorangegangenen Kapiteln das Prozessunterstützungssystem entworfen und realisiert wurde, wird in Kapitel 9 eine allgemeine Vorgehensweise zur Umsetzung von Prozessen mit dem entwickelten Prozessunterstützungssystem beschrieben und die konkrete Umsetzung eines der beiden Beispielprozesse aus Kapitel 4 behandelt.
In Kapitel 10 wird eine Evaluation der in dieser Arbeit erreichten Ziele geliefert.
Zum Abschluss wird in Kapitel 11 eine Zusammenfassung dieser Arbeit präsentiert und ein Ausblick über Erweiterungs- und Verbesserungsmöglichkeiten des entwickelten Prozessunterstützungssystems beschrieben.

# Kapitel 2

# Thematische Einführung

Nach der allgemeinen Einleitung in Kapitel 1 soll in diesem Kapitel nun eine Einführung in die thematischen Grundlagen dieser Diplomarbeit geliefert werden. Dazu wird im Abschnitt 2.1 der Begriff des Portals erklärt. Im Abschnitt 2.2 folgt eine kurze Vorstellung der infoAssetBroker-Plattform, die in dieser Diplomarbeit verwendet wird. Im Abschnitt 2.3 wird eine kurze Einführung in Workflow-Management-Systeme präsentiert, während im Anschluss in Kapitel 2.4 in die Prozessunterstützung in Informationsportalen eingeführt wird. Es wird eine Abgrenzung zu Workflow-Management geliefert und die Motivation für die Integration von Prozessunterstützungssystemen in Informationsportale erläutert.

## 2.1 Portale

In der Informatik bezeichnet ein Portal einen einheitlichen Zugang („Tor“) zu Informationen und Diensten, der im Internet als Website realisiert wird. Eine weitaus genauere Definition ist die folgende, die [KLT00] entnommen ist:

> „Ein *Web-Portal* ist eine Website im World-Wide-Web, die Informationen aus verschiedenen, ausgewählten Quellen zusammenfasst und ihren Nutzern über einen Standard-Web-Browser einen (personalisierten) Zugang mittels Suche und/oder Navigation von Verzeichnisstrukturen bietet, gegebenenfalls ergänzt um redaktionellen Inhalt, Funktionalität zur Kommunikation und/oder Informationsverarbeitung.“

Es kann ein Standard-Web-Browser genutzt werden, um auf die Inhalte eines Portals zuzugreifen. Des Weiteren gibt es personalisierte Zugänge, die Nutzern des Portals Inhalte und Dienste, die den persönlichen Referenzen entsprechen, anbieten.
Portale können weiter unterteilt werden in horizontale und vertikale Portale. Horizontale Portale decken einen weiten Bereich von Themen ab, wohingegen vertikale Portale auf einen eingeschränkten Themenbereich in größerer Detailliertheit fokussieren. Als Standardbeispiel für ein vertikales Portal wird häufig **Amazon.de** [Amz04] genannt. Es ist spezialisiert auf den Verkauf von Büchern, Tonträgern und Software. Als Beispiele für

horizontale Portale werden oft **T-Online**, **AOL**, **Lycos**, **Yahoo** oder **MSN** [Ton04, Aol04, Lyc04, Yah04, Msn04] genannt. Sie sprechen eine breite Zielgruppe an, indem sie über die präsentierten Inhalte viele Themenbereiche abdecken. Schwerpunkt ist nicht die Zielgruppe, sondern die Breite der Abdeckung durch Inhalte und Dienste.
Ein weiterer Begriff, der im Zusammenhang mit Portalen von Bedeutung ist, ist der Begriff des *Unternehmensportals*. Ein *Unternehmensportal* wird in [Weg02] als ein spezielles vertikales Portal eingeführt, dessen Nutzergruppe sowohl die Öffentlichkeit (Medien, Investoren) als auch Kunden, Partner und eigene Mitarbeiter eines Unternehmens sind. Ein Unternehmensportal bietet seinen Nutzern Inhalte an, die für das Geschäftsziel des Unternehmens von Bedeutung sind. Dabei wird in [Weg02] und [Eel03] die Art von Unternehmensportalen weiter nach der Zielgruppe differenziert, die das Unternehmensportal anspricht:

1. Allgemeine Unternehmens-Websites, die der bloßen Information der Öffentlichkeit (Medien, Investoren und Interessenten) über das Unternehmen und der Vorstellung seiner Produkte oder Leistungen, sowie Kontaktmöglichkeiten dienen.

2. Kundenportale, die der engeren Bindung von Kunden – meistens Firmenkunden – an das Unternehmen dienen, um die Verkaufsprozesse zu unterstützen und zu verbessern. Sie bieten meist eine personalisierte Informationsversorgung. Als Schlagwort wird in [Weg02] hierfür das Kundenbeziehungsmanagement (Customer Relationship Management, CRM) angeführt.

3. Partnerportale, die als Zielgruppe eigene Mitarbeiter des Einkaufs oder Verkaufs, etablierte Geschäftspartner und Zulieferer adressieren. Sie dienen der Unterstützung und Verbesserung des elektronisch unterstützten Lieferketten-Management (Supply Chain Management, SCM), sowie von elektronischen Marktplätzen und Anwendungsvermietung (Application Service Providing, ASP).

4. Wissensportale, die dem Wissensmanagement dienen und als Zielgruppe die Mitarbeiter des Unternehmens haben.

Als letzter Begriff wird das *Informationsportal* erläutert. Bei einem Informationsportal handelt es sich ebenfalls um ein Unternehmensportal. In [Weg02] wird der Begriff synonym mit *Enterprise Information Portal* gesehen. Dabei dient ein Informationsportal der Erschließung, Präsentation und Verwaltung von Wissen. Dieses Wissen wird als Information und Inhalt aus verschiedenen Quellen gewonnen. (z.B. Dateien, Dokumenten, Internet, Fax) Auch Informationsportale bieten ihren Nutzern einen personalisierten und rollenbasierten Zugang über das Web. Eine genauere Klassifizierung von Portalen findet sich in [KLT00, Weg02 und Eel03].

## 2.2 Die Portal-Plattform infoAssetBroker

In diesem Abschnitt soll die Portal-Plattform infoAssetBroker [IA04] vorgestellt werden. Dazu werden die verwendete Architektur und Konzepte der Plattform erläutert. Die Plattform wird in dieser Diplomarbeit verwendet. In sie soll das in dieser Arbeit zu entwerfende und zu realisierende Prozessunterstützungssystem integriert werden.
Nach [Weg00] ist der infoAssetBroker eine Standardsoftware zur Realisierung von Informationsportalen, die Informationen ungeachtet vom Format oder Medium anbieten. Benutzer können mit einem Web-Browser oder einem mobilen WAP-Browser auf die Dienste des Portals zugreifen. Dabei können die Informationen des Portals in verschiedenen Systemen, wie Datei-Systemen, Datenbanken oder Content-Managment-Systemen gespeichert sein. Eine Kombination dieser Systeme ist nach [IAWP03] auch möglich.

**Architektur**

Als Standardsoftware zur Realisierung von Informationsportalen ist die infoAssetBroker-Plattform in einer Mehrschichten-Architektur realisiert. Die Abbildung 2.1 zeigt die Architektur-Übersicht. Es werden drei Ebenen gezeigt: Die Präsentationsschicht (P), die Dienstschicht (D) und die Speicherungsschicht (S). Die unterste Ebene bildet die Speicherungsschicht. Aufgabe dieser Schicht ist es, den Zugriff auf die Informationsspeicher des Portals zu realisieren. Dazu werden verschiedene Adapter für unterschiedliche Informationsspeichertypen realisiert. (Datenbank, Datei-System, usw.) Diese Schicht wird nicht weiter erläutert, da ihr Aufbau für diese Arbeit nicht relevant ist.

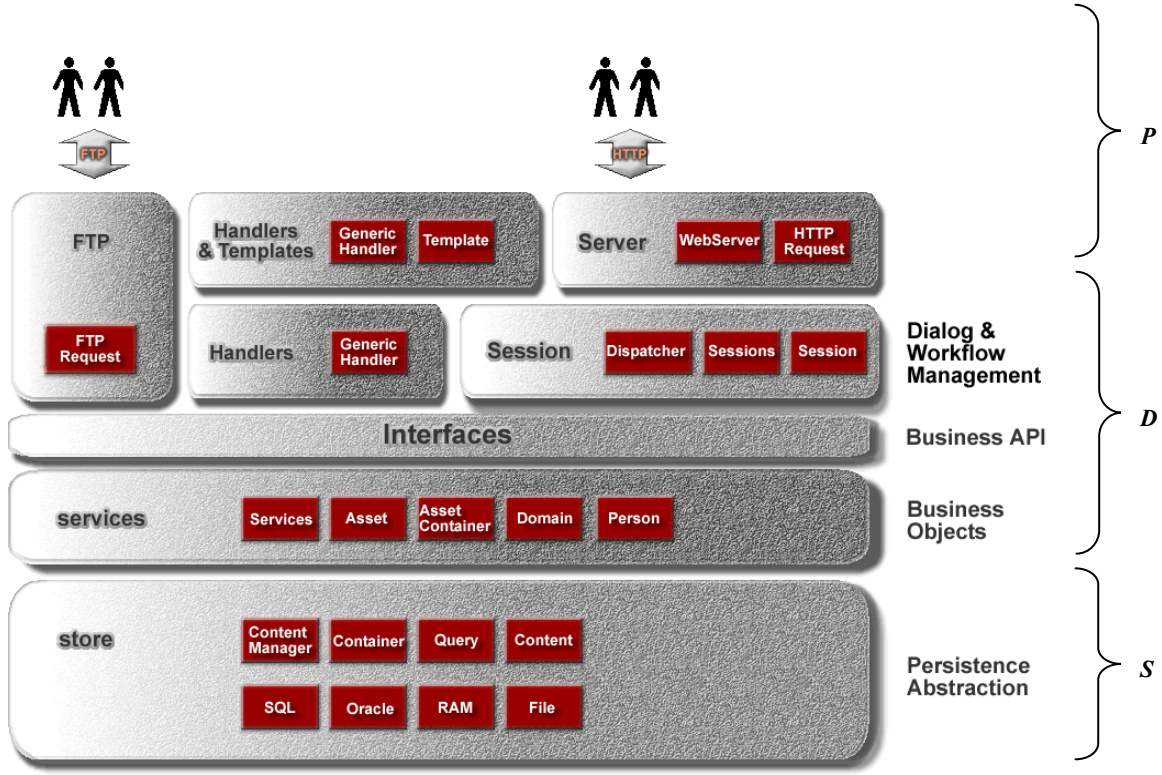


Abbildung 2.1 Architektur der infoAssetBroker-Plattform aus [Ge03]

**Präsentationsschicht**

Die Präsentationsschicht realisiert eine uniforme Benutzerschnittstelle, die Benutzer zur Interaktion mit dem Portal nutzen können. Dazu wird den Benutzern eine geeignete Web-Oberfläche zur Verfügung gestellt. Benutzerinteraktionen werden auf Dienste der darunter liegenden Schicht abgebildet. Dies betrifft allerdings nur die Interaktion mit dem Portal über einen Web-Browser. Der FTP-Zugang funktioniert nach einem anderen Prinzip und wird in dieser Arbeit nicht vorgestellt. Zur Realisierung der Benutzerschnittstelle werden zwei Konzepte eingesetzt:

Handler

Handler bearbeiten Anfragen an den infoAsset Broker in der Präsentationsschicht. Es gibt zwei Arten von Handlern: sichtbare und unsichtbare Handler. Sichtbare Handler können den Systemzustand ändern und erzeugen eine Ausgabe durch Template-Substitution. Unsichtbare Händler können nur den Systemzustand ändern und die Anfrage weiterleiten. Allen Handlern aber ist gemein, dass sie von einer gemeinsamen Oberklasse GenericHandler erben und die Methode handleRequest() implementieren, die vom System aufgerufen wird, damit der jeweilige Handler eine Anfrage bearbeitet.

Templates

Templates sind Output-Formatvorlagen. Sie treten in Form von HTML- oder WML-Vorlagen für die Präsentation auf. Sie können Platzhalter für den dynamischen Inhalt von Seiten enthalten. Dabei sind drei Arten von Platzhaltern möglich: Platzhalter für textuelle Ersetzung, bedingte Platzhalter, die eine Ausgabe nur in bestimmten Fällen durchführen, und Listenplatzhalter (Wiederholungen). Jedem Template ist ein Handler zugeordnet, der die Platzhalter mit Inhalten substituiert. Außerdem können Handler Eingabedaten als Request-Parameter von den Nutzern (aus Formularen) verarbeiten.

Die Abbildung 2.2 zeigt die Verarbeitung einer Benutzeranfrage in mit der infoAssetBroker-Plattform realisierten Portalen.

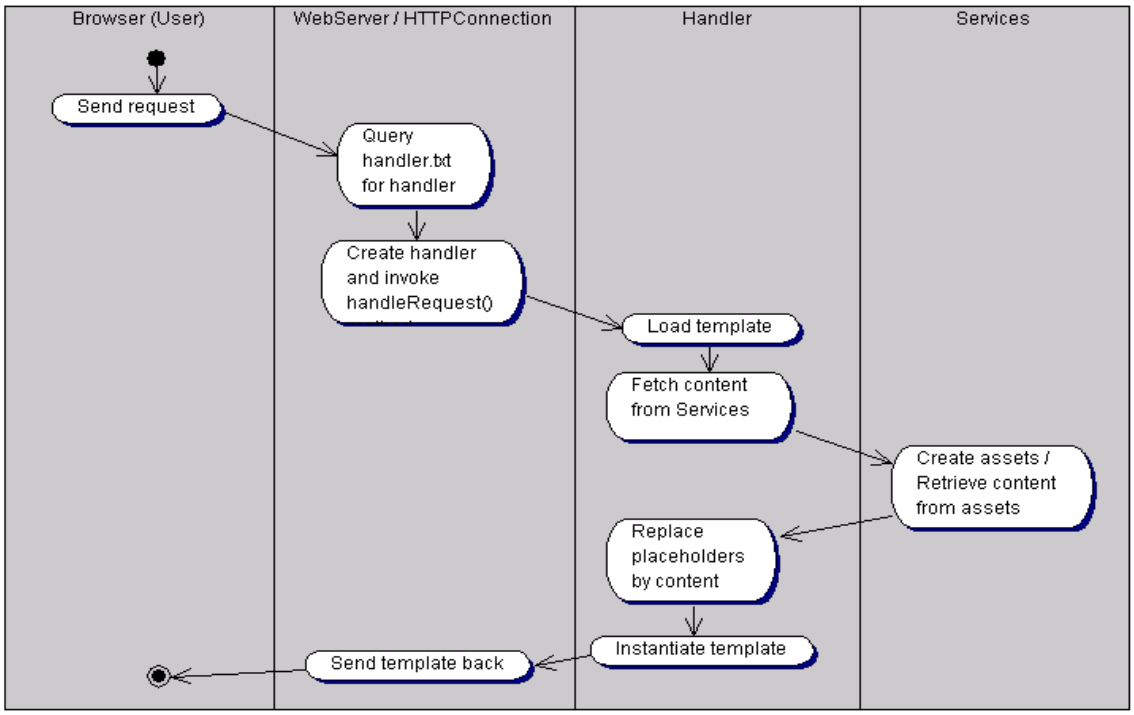


Abbildung 2.2 Anfrageverarbeitung in der infoAssetBroker-Plattform [MH02]

Ein Benutzer sendet eine Anfrage nach einer Portalseite. (*Send request*) Der Web-Server des Portals bestimmt aus der Datei handler.txt welche Handler-Klasse für diese Anfrage zuständig ist (*Query handler.txt for handler*). Dazu wird in der Datei handler.txt eine Abbildung der Anfragen auf Handler-Klassennamen definiert. Anschließend wird der entsprechende Handler erzeugt und die Verarbeitung an ihn übergeben (*Create handler and invoke handleRequest()*). Der Handler lädt zunächst das Template der anzuzeigenden Portalseite. (*load template*). Dann fragt er über die Dienste des Portals die benötigten Informationen ab und ersetzt die Platzhalter des Templates durch Inhalte (*Fetch content from services*, *create assets/retrieve content from assets* und *Replace placeholders by content*). Das Template wird nun instanziiert (*Instantiate template*), d.h. es wird eine konkrete HTML-Seite davon erzeugt. Diese wird vom Web-Server an den Browser des Benutzers als Antwort zurückgesendet (*Send template back*).

**Dienstschicht**

Aufgabe der Dienstschicht ist es, die Dienste zu realisieren, die das Portal bereitstellen soll. Des Weiteren werden in der Dienstschicht auch die Schnittstellen zur Nutzung der Dienste durch Systemkomponenten (z.B. Handler) realisiert. Dazu wird in der Dienstschicht das Konzept der Assets verwendet:

Assets

Assets sind Objekte der Anwendungslogik. Sie repräsentieren Informationsobjekte verschiedener Entitäten (z.B. Dokumente, Personen, Aufträge). Technisch gesehen realisieren Assets das Interface Asset und werden automatisch persistent gespeichert. Alle Assets eines Typs werden von einem sog. AssetContainer verwaltet und besitzen ein Attribut „id“, das als Wert eine Zeichenfolge enthält, die ein Asset eindeutig unter allen Assets eines Typs identifiziert. Dieses Attribut wird auch als *AssetId* bezeichnet.

Alle für diese Arbeit relevanten Konzepte wurden damit eingeführt. Weitere Informationen und Einführungen können aus [Weg02] sowie [Weg00] entnommen werden.

**Das WIPS IT1.1 Informationsportal**

Das WIPS IT1.1 Informationsportal entstand im Rahmen des WIPS IT1.1 Projektes [Wips04], das vom Arbeitsbereich STS der Technischen Universität Hamburg-Harburg [Sts04] in Zusammenarbeit mit dem Germanischen Lloyd in Hamburg [Gl04] durchgeführt wurde. WIPS ist eine Abkürzung für „Wettbewerbsvorteile durch informationstechnische Produktsimulation im Schiffbau“. Das WIPS-Portal wurde mit der infoAssetBroker-Plattform realisiert. Das Portal bietet den potentiellen Kunden des Germanischen Lloyd (Schiffsreedern, Werften) Dienste an, wie z.B. die Abfrage von Schiffsinformationen, Besichtigungsbestellung durch den Reeder. Die eigenen Mitarbeiter des Germanischen Lloyd unterstützt es bei der Verwaltung und Durchführung von Besichtigungsaufträgen und bei der Verwaltung von Schiffsneubau-Projekten. Das WIPS IT1.1 Informationsportal wird in dieser Arbeit als konkretes Beispielportal eingesetzt, welches um ein Prozessunterstützungssystem erweitert wird.

## 2.3 Systeme zur Steuerung und Ausführung von Geschäfts-Prozessen

Nach der Erläuterung des Portal-Begriffs und der Einführung der in dieser Arbeit verwendeten Portalplattform sollen in diesem Kapitel die Grundlagen von Systemen zur Steuerung und Ausführung von Geschäftsprozessen vorgestellt werden. Zur Steuerung und Ausführung von Geschäftsprozessen werden Workflow–Management-Systeme eingesetzt. Der Begriff Workflow kann in diesem Zusammenhang synonym zum Begriff des automatisierten Geschäftsprozesses gesehen werden. Im Bereich der Workflow-Management-Systeme ist es die Workflow Management Coalition [WfMC], die es sich zum Ziel gesetzt hat, Standards zu definieren. Die Workflow Management Coaltion ist eine 1993 gegründete Organisation. Ihr gehören Softwarehersteller, Kunden und Forschungsinstitutionen aus dem Bereich der Workflow-Management-Software an. Ziel dieser Organisation ist es, durch eine breite Basis an Mitgliedern, Standards zu entwickeln, die allgemein akzeptiert werden. Dadurch soll eine vermehrte Nutzung von Workflow-Management-Software erreicht werden.
Um den Begriff des Workflow-Management-Systems zu definieren, wird zunächst eine Definition des Begriffs Workflow präsentiert, die durch die WfMC verfasst wurde (aus dem Englischen übersetzt):

> „Ein *Workflow* beschreibt die Automatisierung eines Geschäftsprozesses im Ganzen oder als Teil, während der Dokumente, Informationen oder Aufgaben von einem Teilnehmer einem anderen übergeben werden mit dem Zweck, nach einer Menge von Verfahrens-Regeln verarbeitet zu werden.“ [WfMC99]

Um diese Definition verstehen zu können, soll der Begriff des Geschäftsprozesses (engl. business process) oder synonym Geschäftsvorgangs ebenfalls definiert werden (aus dem Englischen übersetzt):

> „Ein *Geschäftsvorgang* ist eine Menge einer oder mehrerer verbundener Aktivitäten, deren Ausführung innerhalb einer betrieblichen Organisation mit verschiedenen Rollen und Funktionen einem Geschäftsziel dient.“ [WfMC99]

Nach der Vorstellung der WfMC und der Definition der Begriffe Workflow und Geschäftsvorgang bzw. Geschäftsprozess soll nun der Begriff des Workflow-Management-Systems erläutert werden. Die WfMC definiert ein Workflow-Management-System (WFMS) wie folgt (aus dem Englischen übersetzt):

> „Ein *Workflow-Management-System* ist ein System, das die Ausführung von Workflows durch Verwendung von Software definiert, erstellt und verwaltet. Es wird auf einer oder mehrerer Workflow-Maschinen ausgeführt, die in der Lage sind Prozessdefinitionen zu interpretieren, mit Workflow-Teilnehmern zu interagieren und, wo es nötig ist, andere Softwarewerkzeuge und Anwendungen aufzurufen.“ [WfMC99]

Ein Workflow-Management-System besteht aus Softwarekomponenten, die Prozessdefinitionen speichern und interpretieren können, Instanzen von Workflows erzeugen und ausführen können. Dabei übernehmen die Workflow-Management-Systeme die Interaktion mit den am Prozess beteiligten Personen und steuern den Aufruf von Anwendungen im Prozessverlauf. Workflow-Management-Systeme verfügen typischerweise über Funktionen zur Verwaltung der ausgeführten Prozesse. Dadurch wird es möglich, den Prozessfluss zu ändern, wenn sich Änderungen ergeben. Es können aber auch Informationen über den Zustand des Gesamt-Systems sowie einzelner Prozesse abgefragt werden.
Die WfMC hat ein Referenzmodell entworfen, das den Aufbau von Workflow-Management-Systemen standardisieren soll, so dass Software-Produkte verschiedener Hersteller im Bereich der Workflow-Management-Software besser miteinander kommunizieren und sich gegenseitig ergänzen können. Das Referenzmodell ist in Abbildung 2.3 gezeigt.

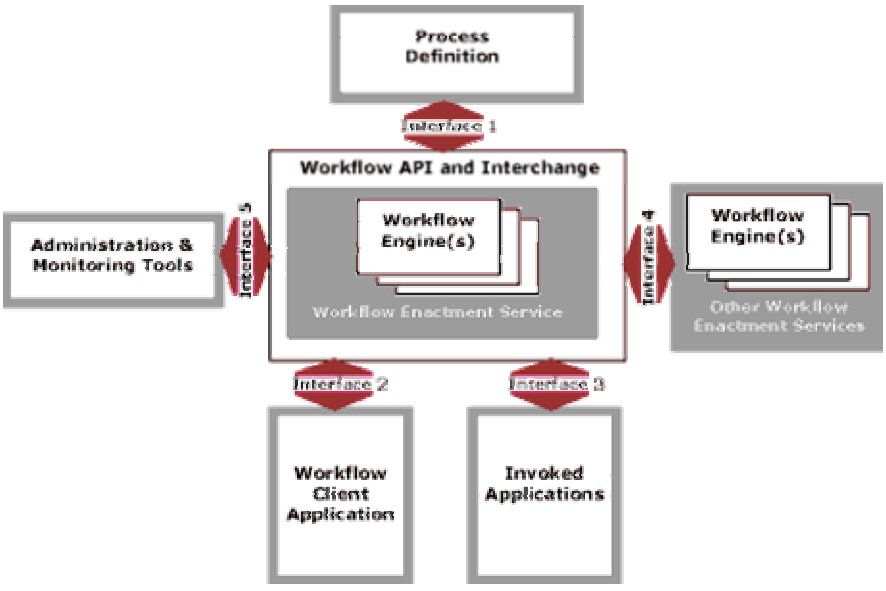


Abbildung 2.3 Referenzmodell der WfMC für Workflow-Management-Systeme

Der zentrale Dienst eines jeden Workflow-Management-Systems ist danach der „*Workflow Enactment Service*“. Die WfMC beschreibt in [WfMC95] den Workflow Enactment-Service als Software-Dienst, der aus einer oder mehr Workflow-Maschinen besteht, um Workflow-Instanzen zu erzeugen, auszuführen und zu verwalten. Anwendungen können auf diesen Dienst über das *Workflow Application Programming Interface* (WAPI) zugreifen.

Es werden im Referenzmodell zwar fünf verschiedene Schnittstellen gezeigt, doch in [WfMC95] wird erläutert, dass diese nicht separate Schnittstellen eines Systems sind, sondern eher fünf unterschiedliche Funktionalitäten.

Nach dem Referenzmodell sollte ein Workflow-Management-System über fünf Funktionalitäten verfügen, die jeweils von der WfMC standardisiert werden. Diese Funktionalitäten sind:

- Definition eines Austauschformates (Interface 1): Prozessdefinitions-Werkzeuge können über dieses Austauschformat Prozessdefinitionen sowohl im- als auch exportieren.
- Zugriff auf Dienste der Workflow-Maschine (Interface 2): Client-Anwendungen können diese Funktionalität nutzen, um auf Dienste der Workflow-Maschine zuzugreifen. Unter anderem können sie den Fortschritt der Prozesse und Aktivitäten steuern.
- Aufruf Externer Anwendungen (Interface 3): Durch diese Funktionalität ist es der Workflow-Maschine möglich verschiedene externe Anwendungen aufzurufen und zu steuern, falls der Prozess dies erfordert.
- Kommunikation zwischen Workflow-Maschinen (Interface 4): Komponenten, die diese Funktionalität realisieren, definieren ein Workflow-Interoperabilitätsmodell, das von Workflow-Maschinen verwendet werden kann, um miteinander zu kommunizieren.
- Verwalten und Überwachen von Prozessen (Interface 5): Diese Funktionalität umfasst typischerweise unter anderem Möglichkeiten zur Benutzerverwaltung, Rollenverwaltung, Ressourcen-Steuerung und Prozessüberwachung.

Nach der Vorstellung der Workflow-Management-Systeme aus Sicht der WfMC soll abschließend noch eine kurze Zusammenfassung über Workflow-Management-Systeme aus einer anderen Quelle geliefert werden. In [RM03] werden Workflow-Management-Systeme beschrieben als eine Technologie zur aktiven und zuverlässigen Ausführung von eventuell verteilten Geschäftsprozessen. Voraussetzung dafür ist, dass die Geschäftsprozesse stark strukturiert sind. Dabei werden in Workflow-Management-Systemen Ablauflogik und Anwendungscode getrennt. Der Datenfluss wird in solchen Systemen explizit modelliert. Workflow-Management-Systeme integrieren manuelle und automatisierbare Arbeitsschritte eines Geschäftsprozesses und bieten zur Ausführungssicherheit verschiedene Transaktionsmodelle. Arbeitschritte werden nicht durch das System selbst ausgeführt, sondern vielmehr werden Anwendungen ferngesteuert, um diese Aufgabe zu erledigen.

## 2.4 Prozessunterstützung in Informationsportalen

In diesem Abschnitt soll die Motivation für die Integration von Prozessunterstützungssystemen in Informationsportale erläutert werde. Anschließend werden die Funktionalitäten, die durch ein Prozessunterstützungssystem eines Informationsportals realisiert werden müssen, präsentiert und der Begriff des Prozessunterstützungssystems erläutert.

### 2.4.1 Motivation

In diesem Abschnitt soll die Motivation für die Integration eines Prozessunterstützungssystems in ein Informationsportal geliefert werden. Betreibt ein Unternehmen ein Informationsportal, so sollte dieses die zentrale Informationsquelle für die Mitarbeiter eines Unternehmens darstellen. Wichtige Informationen, die die tägliche Arbeit eines Mitarbeiters betreffen, werden durch das Informationsportal bereitgestellt. Genauso könnten Kunden den Status einer Bestellung abfragen. Um die Arbeitsabläufe noch effektiver zu gestalten, ist eine Unterstützung von strukturierten, definierten Geschäftsprozessen des Unternehmens durch das Portal wünschenswert. Als Beispiel für einen Geschäftsprozess, der innerhalb eines Portals abläuft, sei das Bearbeiten einer Bestellung genannt. Jede Bestellung wird formal auf Korrektheit geprüft, anschließend wird die Verfügbarkeit der bestellten Dienstleistungen oder Produkte geprüft und die Bereitstellung bzw. der Versand angeordnet. Parallel dazu wird die Rech-

nung erstellt. Diese und andere Aufgaben können durch ein in das Portal integriertes Prozessunterstützungssystem unterstützt werden. Die Vorteile eines integrierten Prozessunterstützungssystems sind:

- Gezielte Bereitstellung von Informationen im Verlauf eines Geschäftsprozesses für Mitarbeiter, Lieferanten oder Kunden des Unternehmens zu dem Zeitpunkt, zu dem sie benötigt werden.
- Sich wiederholende Arbeitsabläufe können strukturiert, automatisiert und optimiert werden.
- Informationen über den Fortschritt der einzelnen Prozesse können durch das Portal bereitgestellt werden. (Controlling, Kundeninformationen)
- Es wird eine Möglichkeit zur automatischen Bearbeitung von Informationen durch das Portal-System geschaffen.

Den Prozessteilnehmern wird vor allem die Suche von Informationen abgenommen, da diese im Verlauf des Prozesses gezielt durch das Prozessunterstützungssystem bereitgestellt werden können. Auf der anderen Seite werden durch Prozessunterstützungssysteme neue Informationen wie Zustand und Fortschritt der Geschäftsprozesse bereitgestellt, die zur Überwachung und Steuerung der Geschäftstätigkeit eingesetzt werden oder Kunden zur Verfolgung ihrer Aufträge zur Verfügung gestellt werden können.

### 2.4.2 Prozessunterstützungssysteme in Informationsportalen

Nachdem die Motivation für die Integration von Prozessunterstützungssystemen in Informationsportale erläutert wurde, soll in diesem Abschnitt nun der Begriff Prozessunterstützungssystem erläutert werden, wie er im Folgenden in dieser Arbeit verwendet wird. Ein Prozessunterstützungssystem bezeichnet in dieser Arbeit eine Untermenge der Funktionalität von Workflow-Management-Systemen. Ein Prozessunterstützungssystem realisiert nicht alle der durch das Referenzmodell (Abbildung 2.3) beschriebenen Funktionalitäten. Für ein Prozessunterstützungssystem, das in ein Informationsportal integriert wird, sollen in dieser Arbeit die folgenden Funktionalitäten berücksichtigt werden:

- Definition von Prozessen (WfMC: Interface 1): Es sollen Prozesse definiert, bzw. ihre Definition geändert und angepasst werden können.
- Administration und Überwachung des Systems (WfMC: Interface 5): Dies dient der Verwaltung von ausgeführten Prozessen durch den zuständigen Administrator. Des Weiteren können Prozesse überwacht werden, d.h. der Prozessfortschritt von laufenden Prozessen kann verfolgt werden.
- Ausführung von manuellen Aktivitäten (WfMC: Untermenge von 2): Benutzer führen über die Web-Schnittstelle des Portals manuelle Aktivitäten aus.

Damit werden die folgenden Funktionalitäten des Referenzmodells nicht unterstützt:

- Aufruf externer Anwendungen (WfMC: Interface 3): Eine Unterstützung dieser Funktionalität ist nicht notwendig. Informationsportale arbeiten web-basiert, d.h. die Benutzer benutzen einen Standard-Web-Browser, um an die durch das Informationsportal bereitgestellten Informationen zu gelangen. Genauso muss die Interaktion mit dem Prozessunterstützungssystem als Teil des Portals über den Web-Browser laufen. Es werden also keine externen Anwendungen zur Ausführung von Aktivitäten durch das Prozessunterstützungssystem aufgerufen.
- Interoperabilität mit anderen Prozessmaschinen (WfMC Interface 4): Im Rahmen dieser Arbeit soll diese Funktionalität nicht betrachtet werden. In Kapitel 6 wird allerdings ein kurzer Vorschlag eines Kommunikationsverfahrens zwischen Prozessmaschinen gemacht.

In [Ahm03] wurde ein Prototyp eines Prozessunterstützungssystems für die infoAssetBroker-Plattform realisiert, der die hier beschriebenen Funktionalitäten für Prozessunterstützungssysteme teilweise unterstützt. Er wird in Kapitel 3.2 vorgestellt.

## 2.5 Zusammenfassung

In diesem Kapitel wurde in die thematischen Grundlagen dieser Arbeit eingeführt. Es wurden die verschiedenen Portalarten kurz vorgestellt. Anschließend wurde die infoAssetBroker-Plattform als eine konkrete Plattform zur Realisierung von Informationsportalen vorgestellt. Abschließend wurden die Begriffe Workflow-Management-System und Prozessunterstützungssystem eingeführt, sowie die Motivation für die Integration von Prozessunterstützungssystemen in Informationsportale geliefert.

# Kapitel 3

# Existierende Arbeiten

Dieses Kapitel teilt sich in zwei Unterabschnitte auf. Es sollen existierende Arbeiten zur Umsetzung einer Prozessunterstützung für die Portalplattform infoAsset Broker vorgestellt werden. Im ersten Abschnitt wird der Job-Dienst präsentiert, während im zweiten Abschnitt dieses Kapitels ein Prototyp eines Prozessunterstützungssystems präsentiert wird. Die Ergebnisse stammen aus drei Arbeiten: der Job-Dienst wurde in der Studienarbeit von Vanda Lehel [Leh01] entwickelt, während die Analyse des Prozessunterstützungssystems von Ayaz Ahmed im Rahmen eines Student Project [Ahm02] durchgeführt wurde und Design und Implementierung im Rahmen der Master Thesis [Ahm03] realisiert wurden.

## 3.1 Der Job-Dienst für den infoAsset Broker

Ziel der Studienarbeit [Leh01] war es, einen generischen Basisdienst, genannt Job-Dienst, zur Unterstützung arbeitsteiliger Geschäftsvorgänge in auf der Plattform infoAssetBroker basierenden Unternehmensportalen prototypisch zu realisieren. Dafür sollten die Anforderungen an arbeitsteilige Geschäftsvorgänge anhand existierender Systeme ermittelt und berücksichtigt werden. Weiter sollte exemplarisch ein arbeitsteiliger Geschäftsvorgang mit dem neu entwickelten Job-Dienst realisiert werden.
Ein arbeitsteiliger Geschäftsvorgang wird in der Studienarbeit als ein Geschäftsvorgang definiert, an dessen Abwicklung mehrere unterschiedliche Organisationseinheiten eines Unternehmens beteiligt sein können, wobei als Organisationseinheiten Personen und Gruppen angesehen werden.

### 3.1.1 Anforderungen

Zur Anforderungsermittlung wurden im Rahmen der Studienarbeit vier verschiedene Geschäftsvorgänge betrachtet. Der erste Geschäftsvorgang behandelte die Freischaltung einer Gruppenmitgliedschaft in einem Informationsportal.
Ein im Portal registrierter Nutzer beantragt die Aufnahme in eine Gruppe. Die Administratoren des Portals bearbeiten den Auftrag und schalten die Gruppenmitgliedschaft frei, wenn alle Voraussetzungen dafür erfüllt sind.

Im zweiten Geschäftsvorgang wird der Vorgang der Fehlerdokumentation und –behebung (issue management), wie er aus der Softwareentwicklung bekannt ist, mittels des BugTrackers des infoAsset-Brokers betrachtet. Eine Kunde erfasst eine neue Meldung (BugReport), die von einem Koordinator an den zuständigen Entwickler weitergeleitet wird. Dieser Entwickler kann dann die Meldung annehmen oder an einen anderen Entwickler weiterleiten. Der Geschäftsvorgang ist abgeschlossen, sobald der Fehler behoben ist.
Als dritter Geschäftsvorgang wird der Bestellvorgang im Portal der Firma Contentmap betrachtet. Dieses Portal ist mit einer erweiterten Version der infoAssetBroker-Plattform realisiert worden, die zur Realisierung von Portalen zur Vermittlung und zum Vertrieb von digitalen Inhalten im Internet verwendet werden kann. [Info01]. Contentpakete stellen digitale Inhalte dar, die in einer bestimmten Form zu speziellen Lizenzbedingungen vertrieben werden. Der Bestellvorgang läuft so ab: Der Kunde gibt eine Bestellung ab. Die verschiedenen Bestellpositionen werden von einem oder mehreren Mitarbeitern auf Korrektheit geprüft, der Bestellauftrag wird anschließend akzeptiert oder abgelehnt und die digitalen Inhalte werden ausgeliefert, falls der Auftrag akzeptiert wurde.
Als letzter Geschäftsvorgang wird die Bestellung einer Schiffsbesichtigung im WIPS-Portal betrachtet [HLAS04]. Dieser Geschäftsvorgang wird nicht weiter betrachtet, da er schon im vorigen Kapitel erläutert wurde und im Kapitel 4.1 detailliert beschrieben wird.
Aus den oben vorgestellten Geschäftvorgängen wurden in der Studienarbeit von Vanda Lehel Anforderungen an den Job-Dienst ermittelt. Allen vorgestellten Geschäftsvorgängen ist demnach gemein, dass sie auf ein abstraktes Prozessmodell abgebildet werden können. Es finden sich die folgenden Konzepte:

- *Auftraggeber* - der Initiator des Geschäftsvorganges
- *Auftragnehmer* - die den Geschäftsvorgang abwickelnden Akteure
- *Auftrag* - der Geschäftsvorgang selbst
- *Auftragsbearbeitung* - Abwicklung des Geschäftsvorganges
- *Auftragstyp* - Charakterisierung eines Geschäftsvorganges nach Abfolge der Bearbeitungsschritte und der zugehörigen Tätigkeiten
- *Auftragsmappe* - enthält die Daten, die zur Bearbeitung des Geschäftsvorganges erforderlich sind (z.B. Dokumente).

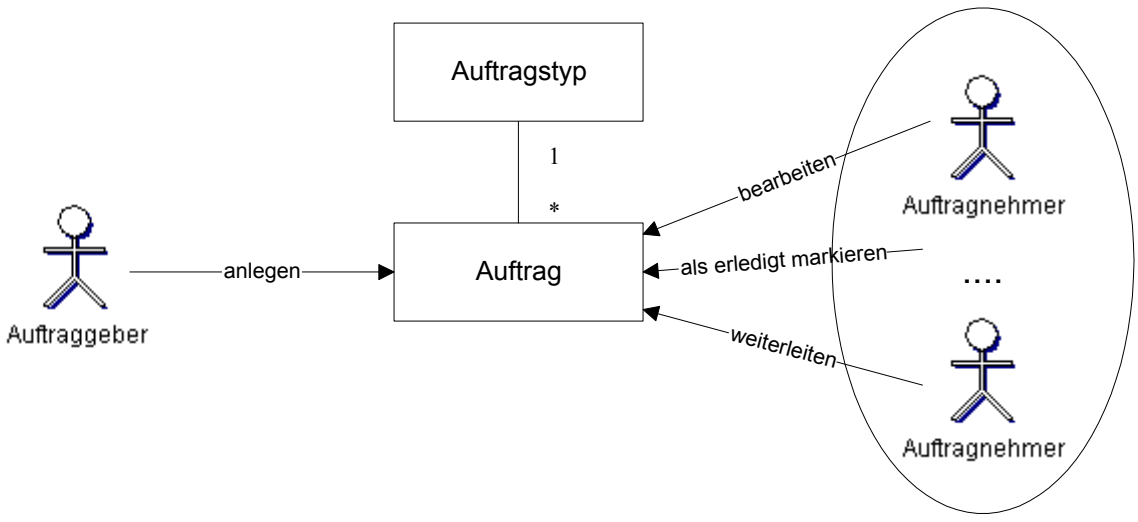


Abbildung 3.1 Akteure und Operationen an einem Auftrag (aus [Leh01])

Die Abbildung 3.1 zeigt die an einem Auftrag beteiligten Akteure und ihre Aktionsmöglichkeiten.
Zur Abwicklung von Aufträgen werden noch die Konzepte der *Auftragsverwaltung* und *Auftragslisten* eingeführt. Die Auftragsverwaltung soll es erlauben, auch auf Mengen von Aufträgen Operationen auszuführen. Ferner ist jedem Auftragsgeber eine Auftragsliste zugeordnet, die Aufträge enthält, deren Auftraggeber er ist. Jedem Auftragnehmer ist gleichfalls eine Auftragsliste zugeordnet, die die Aufträge enthält, die ihm zum gegebenen Zeitpunkt zugewiesen sind.
Als funktionale Anforderungen ergeben sich dabei auf einzelnen Aufträgen die folgenden Operationen:

- Neuen Auftrag anlegen
- Auftrag mit Details anzeigen
- Einem Auftrag Anlagen zuordnen
- Vom Auftrag Anlagen entfernen
- Anlagen eines Auftrags anzeigen
- Auftragshistorie anzeigen
- Auftrag annehmen
- Auftrag ablehnen
- Auftrag bearbeiten
- Auftrag weiterleiten
- Auftrag zur Seite legen
- Auftrag als erledigt markieren
- Zu einem Auftrag Unteraufträge erstellen
- Auftrag löschen

Funktionalitäten der Auftragsverwaltung sind:

- Liste der initiierten Aufträge anzeigen
- Liste der zu bearbeitenden Aufträge anzeigen
- Auftragsliste der Gruppen anzeigen
- Filtern von Aufträgen in der Anzeige der Auftragslisten
- Aufträge zu einer Anlage suchen

Genauere Beschreibungen zu den einzelnen Funktionen finden sich in [Leh01].

### 3.1.2 Das Modell des Job-Dienstes

Nachdem die funktionalen Anforderungen kurz dargestellt wurden, wird nun der daraus entwickelte Job-Dienst kurz präsentiert. Es werden die wichtigsten Klassen vorgestellt und ihre Funktion erläutert. Ein konzeptuelles Klassendiagramm der Klassen findet sich im Anhang A. Die Tabelle 3.1 ordnet den festgestellten Konzepten Begriffe des entwickelten Job-Dienstes zu.
Die Klassen *Job*, *JobType*, und *JobStep* implementieren das Asset-Interface und zählen deshalb zu den Services der Broker-Plattform. Als Historie eines Auftrages wird die Folge der JobSteps des Auftrags modelliert. Die Menge der Anlagen wird als ein PortfolioItem modelliert.

| **Konzept** | **Begriff im Job-Dienst** |
|---|---|
| Auftrag | Job |
| Auftragstyp | JobType |
| Bearbeitungsschritt | JobStep |
| Auftragshistorie | History |

| | |
|---|---|
| Anlage | Asset |
| Auftragsmappe | Portfolio |
| Unterauftrag | Subjob |

Tabelle 3.1: Entsprechungen zwischen den Konzepten und den Begriffen des Job-Dienstes (aus [Leh01])

Die wichtigsten Klassen sind:

- *Job*: Instanzen dieser Klasse repräsentieren Aufträge, also konkrete Geschäftsvorgänge. Ein Auftrag kann während der Auftragsbearbeitung die vier Zustände durchlaufen: ‚Gruppe zugewiesen', ‚Person zugewiesen', ‚in Bearbeitung' und ‚erledigt'. Beim Anlegen eines Auftrages müssen Auftraggeber, Erstellungsdatum, Titel und die Gruppe der nächsten Bearbeiter angegeben werden. Der Auftrag befindet sich nun im Zustand ‚Gruppe zugewiesen', erscheint in der zugehörigen Auftragsliste und kann von einem Auftragnehmer aus der Gruppe angenommen werden.
- *JobType*: Instanzen dieser Klasse repräsentieren verschiedene konkrete Auftragstypen. Dabei wird die Abfolge der vom System aufzurufenden Bearbeitungsschritte in den Instanzen gespeichert. Ein Bearbeitungsschritt wird dabei durch den Aufruf eines Handlers realisiert (s. Kapitel 2).
- *JobStep*: Instanzen dieser Klasse repräsentieren bereits ausgeführte Bearbeitungsschritte. Sie speichern Informationen zum Bearbeiter, Beginn der Bearbeitung, Ende der Bearbeitung u.a. Die Folge der JobSteps eines Auftrages stellt seine Historie dar.

Auf die weitere Präsentation der Studienarbeit soll verzichtet werden, da die weiteren Ergebnisse für diese Arbeit unrelevant sind.

### 3.1.3 Zusammenfassung

An dieser Stelle sollen noch einmal die entscheidenden Eigenschaften des Job-Dienstes zusammengetragen werden. Der Job-Dienst unterstützt Prozesse, die streng sequentiell ablaufen. Das bedeutet der Job-Dienst lässt keinerlei Definition von Nebenläufigkeit, Verzweigungen oder Schleifen zu. Nebenläufigkeit kann nur erreicht werden, indem Unteraufträge angelegt werden. Dies kann aber nur zur Laufzeit geschehen und kann nicht vorab definiert werden.
Der Job-Dienst besitzt keinen grafischen Editor, mit dem sich die Bearbeitungsreihenfolge einfach definieren ließe. Dies kann nur durch Anlegen einer XML-Datei geschehen, die die Abfolge definiert.
Die Bearbeitungsreihenfolge ist unabhängig von der Abwicklung des Auftrages fest definiert. Sie lässt keine Verzweigungen und somit auch kein variables, von der Abwicklung abhängiges Verhalten zu. Ein Abweichen vom sequentiellen Ablauf muss in der Logik der Handler codiert werden.

## 3.2 Das Prozessunterstützungssystem für den infoAsset Broker

Ziel der Studienarbeit [Ahm02] war es, basierend auf dem Job-Dienst und zwei ausgewählten Geschäftsvorgängen Anforderungen an ein Prozessunterstützungssystem (*Processsupport-System*) zu analysieren, wobei das Prozessunterstützungssystem an die gängigen Standards der [WfMC] angelehnt sein sollte. Ferner sollten die Konzepte präsentiert werden, nach welchen ein solches System aufgebaut werden konnte.
Die Master Thesis [Ahm03] beschreibt die prototypische Realisierung eines solchen Systems mit einigen der festgestellten Konzepte für die Plattform infoAsset Broker.

An dieser Stelle sollen nur die in [Ahm02] festgestellten Konzepte, ihre technische Realisierung [Ahm03] und die sich daraus ergebenden Eigenschaften dieses Prozessunterstützungssystems präsentiert werden. Es werden aber dabei nur die Konzepte präsentiert, die auch in [Ahm03] realisiert wurden.

### 3.2.1 Konzepte für Prozessunterstützung

Bei der Präsentation der Konzepte orientiert sich [Ahm02] eng an den durch die WfMC vorgegebenen Standard aus [WfMC99]. Die folgenden generellen Konzepte auf der Workflow-Ebene sind demnach relevant für ein Prozessunterstützungssystem:

- *Business Process* (dt. Geschäftsvorgang): Wird von der WfMC als Menge einer oder mehrerer verbundener Aktivitäten betrachtet, deren Ausführung innerhalb einer betrieblichen Organisation mit verschiedenen Rollen und Funktionen einem Geschäftsziel dient.
- *Process Definition*: Ist die digitale Repräsentation eines Geschäftsvorganges, die von einem Prozessunterstützungssystem verstanden und ausgeführt werden kann. Sie enthält Angaben über alle auszuführenden Aktivitäten, Kriterien für deren Ausführungsbeginn und die ausführenden Organisationseinheiten.

Unter den Kontrollfluss-Konzepten können die folgenden eingegliedert werden:

- *Process Instance*: Eine Prozessinstanz stellt eine Ausführung einer Prozessdefinition, also einen konkreten Geschäftsvorgang, dar und beinhaltet alle der Ausführung zugeordneten Informationen und Daten.
- *Sub Process Instance*: Stellt eine Prozessinstanz dar, die von einer laufenden Prozessausführung aus gestartet wurde und somit Teil der aufrufenden Prozessausführung ist.
- *ProcessThread*: Ein Prozessthread wird innerhalb einer Prozessinstanz ausgeführt und führt eine sequentielle Bearbeitung von Aktivitäten durch. Sollen mehrere Aktivitäten nebenläufig ausgeführt werden, so ist für jede nebenläufig auszuführende Aktivitätenfolge ein eigener Prozessthread erforderlich. Jeder Prozessthread hat einen Zustand, der angibt, ob der Thread gerade ausgeführt wird, pausiert, abgebrochen oder (erfolgreich) beendet wurde [CDK01].
- *Activity*: Eine Aktivität ist die kleinste Einheit eines Geschäftsvorganges. Sie beschreibt einen einzelnen logischen (Arbeits-)Schritt. Dabei wird zwischen *manuellen* Aktivitäten und *automatischen* Aktivitäten unterschieden. Manuelle Aktivitäten werden durch Menschen, die dem System als Benutzer bekannt sind, ausgeführt, automatische Aktivitäten durch das System selbst.
- *Transition*: Eine Transition beschreibt den Übergang von einer Aktivität zur nächsten. Einer Transition kann eine Bedingung (s. unten) zugeordnet sein. In diesem Falle kann der durch diese Transition beschriebene Übergang nur stattfinden, wenn die Bedingung erfüllt ist.
- *Process Start*: Die Umstände, die zu einem Prozessstart führen, werden durch die Prozessdefinition beschrieben. Dabei kann ein Prozess sowohl manuell durch Benutzeranfrage als auch automatisch (z.B. durch ein Ereignis) ausgelöst werden. Einzige Bedingung ist, dass jeder Prozess einen eindeutigen Einstiegspunkt hat.
- *Process End*: Ein Prozess wird dann als beendet betrachtet, sobald alle seine Unterprozesse sowie die durch die jeweilige Prozessinstanz ausgeführten Prozessthreads beendet sind.

Zu den Konzepten für die Modellierung des Datenflusses zählen nach [Ahm03] diese:

- *Conditions*: Bedingungen können an verschiedenen Stellen verwendet werden. Sie dienen als Vor- und Nachbedingung (Pre-/Post-Condition) einer Aktivität. Sie stellen einen logischen Ausdruck dar, der vom System ausgewertet werden kann. Nur wenn die Vorbedingung erfüllt ist, kann die Aktivität ausgeführt werden. Sollte die Nachbedingung nach Beendigung nicht erfüllt sein, so kann die Aktivität

nicht abgeschlossen werden. Weiter treten Bedingungen als Kontrollflussbedingungen an Transitionen auf.

- *Data Dictionary*: Ist der zentrale Speicherplatz für Prozessinstanzen, der Informationen und Prozessvariablen persistent speichert. Im Kapitel 3.2.2 ist dies der Prozesskontext.

### 3.2.2 Das Prozessmodell

Aus den hier präsentierten Konzepten wurde in [Ahm03] ein Prozessunterstützungssystem realisiert, dass aus drei Teilen besteht. Den ersten Teil stellt ein *Prozesseditor* dar, der zur grafischen Definition von Prozessen verwendet werden kann. Den zweiten Teil stellt das Prozessmodell selbst dar und der dritte Teil ist die Prozessmaschine (*ProcessEngine*) an sich. Die Abbildung 3.2 zeigt die Komponenten in der Architekturübersicht.
An die Stelle des in Kapitel 2.2 vorgestellten Workflow-Enactment-Service von Workflow-Management-Systemen tritt ein Prozessmodell, welches keine strikte Trennung von Daten- und Kontrollfluss vorsieht. Des Weiteren werden auch im Gegensatz zu Workflow-Management-Systemen keine Transaktionen unterstützt, da dieses Konzept in der zu Grunde liegenden infoAssetBroker-Plattform nicht vorgesehen ist.
Es werden alle drei Teile vorgestellt, wobei zunächst das entwickelte Prozessmodell erläutert werden soll. Dazu wird zunächst das konzeptuelle Klassendiagramm des Modells in Abb. 3.3 dargestellt und die dort auftretenden Klassen näher beschrieben. Hier wird deutlich, dass das Modell sehr stark an die erläuterten Konzepten angelehnt ist.

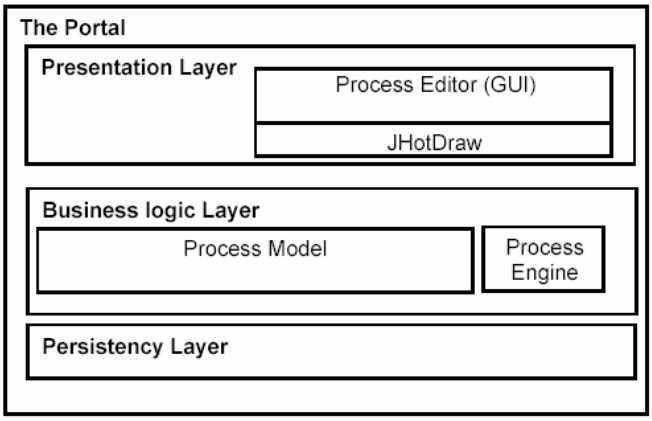


Abbildung 3.2 Architektur des prototypisch realisierten Prozessunterstützungssystems aus [Ahm03]

**ProcessDefinition**
Instanzen dieser Klasse repräsentieren das Konzept der Prozessdefinition. Einer ProcessDefinition ist die Menge der Transitionen zugeordnet, die den Kontrollfluss eines Geschäftsvorgangs darstellen. Eine ProcessDefinition besitzt eine eindeutig definierte Start-Aktivität (*StartActivity*) und definiert mögliche Endzustände (*Endstates*). Zu den Attributen einer ProcessDefinition, zählen der Name, eine Beschreibung, sowie die Information über den Ersteller dieser Definition.

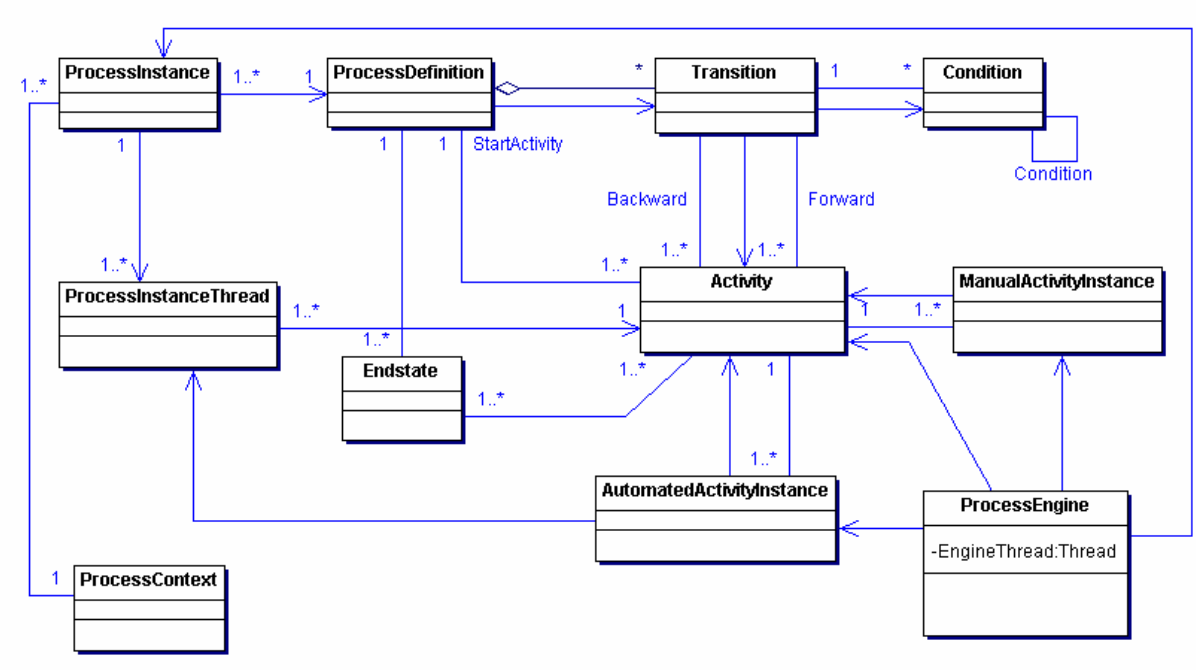


Abb. 3.3 Konzeptuelles Klassendiagramm des Prozessmodells (aus [Ahm03])

**ProcessInstance**

Sie stellt die Ausführung eines Geschäftsvorganges dar. Jeder ProcessInstance ist die Prozessdefinition zugeordnet, deren Ausführung sie darstellt. Sie wird in einem oder mehreren ProcessInstanceThreads ausgeführt und besitzt einen ProcessContext, sowie Informationen darüber, wer die Ausführung gestartet hat. Ferner besitzt eine ProcesssInstance einen Zustand, der sich aus dem Zustand ihrer *ProcessInstanceThreads* ergibt. Die Abbildung 3.4 zeigt die möglichen Zustände eines *ProcessInstanceThread*. Auf die Beschreibung der Zustände soll verzichtet werden, sie kann bei Bedarf in [Ahm03] nachgelesen werden.

Es soll ferner angegeben werden, wie sich der Zustand der ProcessInstance ergibt: Eine ProcessInstance befindet sich im Zustand

- *Inactive*, wenn alle ProcessInstanceThreads *Inactive* sind
- *Hold,* wenn alle ProcessInstanceThreads sich im Zustand *Hold* befinden
- *Aborted,* wenn alle ProcessInstanceThreads abgebrochen wurden
- *Finished,* wenn alle ProcessInstanceThreads beendet sind
- *Active,* falls keiner der vorigen Zustände zutrifft.

**ProcessInstanceThread**

Das Konzept der ProcessThreads wird von dieser Klasse realisiert. Eine Instanz dieser Klasse entspricht genau einem Thread, welcher in der ProcessEngine (s. u.) ausgeführt wird. Anfänglich startet jede Prozessinstanz in einem Thread. Durch Starten neuer Threads wird Nebenläufigkeit erzeugt. Jeder ProcessInstanceThread hat einen Zustand und kennt die ProcessInstance, innerhalb der er ausgeführt wird. Jeder Thread hat eine Referenz auf die augenblickliche, durch ihn auszuführende Tätigkeit.

**Activity**

Eine Instanz dieser Klasse repräsentiert eine Aktivität eines Geschäftsprozesses. Jede Aktivität besitzt einen Namen, eine Beschreibung, sowie eine Information darüber, welcher Handler zu ihrer Ausführung durch das System aufzurufen ist (siehe Kapitel 2.2, Handler).

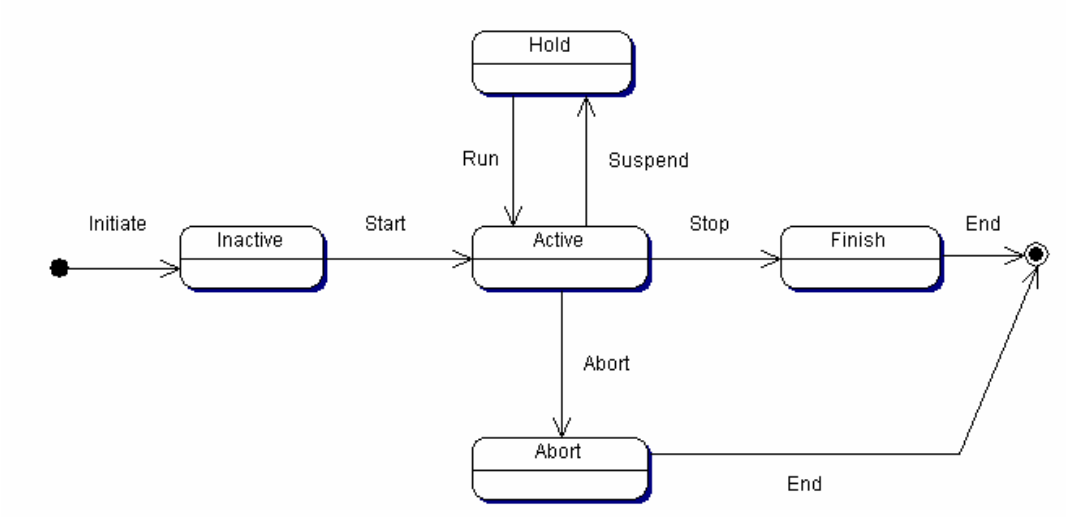


Abbildung 3.4 Zustände eines ProcessInstanceThreads (aus [Ahm03])

**ActivityInstance**

Es ist wichtig zwischen Aktivitäten und ihrer Ausführung zu unterscheiden. Instanzen der Klasse Activity repräsentieren eine Aktivität als solche, während eine ActivityInstance die Ausführung der Aktivität beschreibt. Sie speichern Informationen über den Beginn, das Ende sowie die ausführende Rolle innerhalb des Systems. Es gibt dabei **ManualActivityInstances** und **AutomatedActivityInstances**. ManualActivityInstances beschreiben Ausführungen manueller Aktivitäten und AutomatedActivityInstances beschreiben Ausführungen automatischer Aktivitäten. Ähnlich wie ProcessInstances besitzen alle ActivityInstances einen Zustand. Abbildung 3.5 zeigt die möglichen Zustände von ActivityInstances.

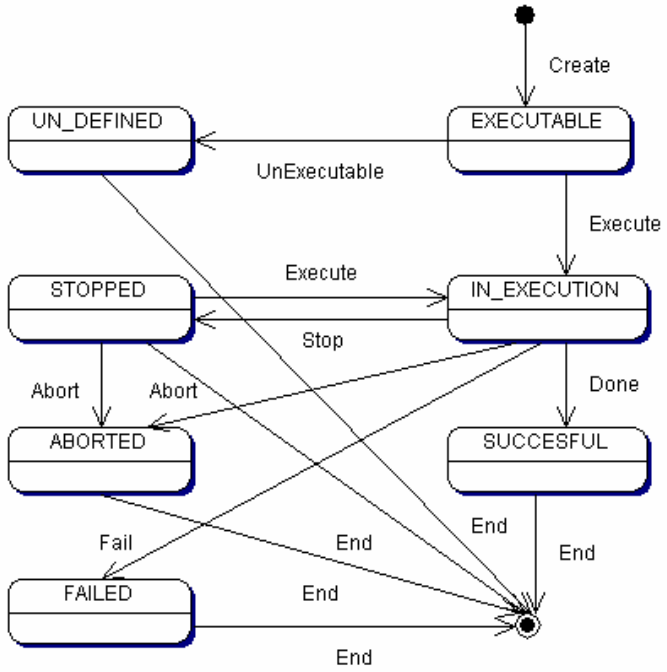


Abbildung 3.5 Zustände einer ActivityInstance ([Ahm03])

**Endstate**
Diese Klasse realisiert das Konzept Prozessende. Jede Instanz dieser Klasse beschreibt einen möglichen Endzustand einer Prozessdefinition. Erreicht eine Prozessinstanz einen solchen Endzustand, so gilt sie als beendet, wenn alle ProzessInstanceThreads beendet sind.

**Transition**
Jede Instanz repräsentiert den Übergang von einer Aktivität zur nächsten. Jede Transition hat einen optionalen Namen, kennt ihre Ursprungsaktivität und die Zielaktivität. Des Weiteren kann eine Transition eine Kontrollfluss-Bedingung (*Condition*) haben, die bestimmt, ob die Transition durchgeführt werden kann.

**ProcessContext**
Jeder ProcessInstance ist ein Prozesskontext zugeordnet. Dies ist ein Repository für die einzelnen Informationsobjekte, die eine Prozessausführung benötigt. Ein Prozess kann nach belieben Werte und Objekte im ProcessContext persistent speichern.

**Condition**
Über dieses Klasse wird das Konzept der Conditions realisiert. Dabei werden bereits für den infoAsset Broker entworfene Klassen wieder verwendet. In [Gic03] wurde eine Regelmaschine entwickelt und dabei das Konzept der Conditions realisiert. Aus dieser Arbeit wurden die Klassen *Condition* und *EvaluationContext* übernommen. Nach [Gic03] beschreibt eine Condition einen logischen Ausdruck basierend auf Konstanten und Variablen, der gegen eine Menge von Variablenbindungen ausgewertet werden kann. Die Variablenbindungen des Prozesskontextes werden hierzu verwendet.
Eine Bedingung (Condition) ist eine abstrakte Klasse, und jede ihrer Subklassen definiert einen speziellen Bedingungstyp (SimpleCondition, ComplexCondition und NOTCondition). Einfache Bedingungen (SimpleConditions) werden zum Vergleich von Termen und für den Test auf Inklusion verwendet. Bedingungen lassen sich durch UND, ODER verknüpfen oder durch NOT invertieren. Weitere Details dazu lassen sich in [Gic03] nachlesen.

### 3.2.3 Die Prozessmaschine

Die Prozessmaschine (*ProcessEngine*) stellt eine Komponente dar, die die Ausführung von Prozessen steuert. Dazu leitet sie die Ausführung von manuellen Aktivitäten ein und führt automatische Aktivitäten selbst aus. Die wichtigsten Eigenschaften der Prozessmaschine sind: Sie

- führt Prozesse durch Generieren einer Prozessinstanz mit einem anfänglichen ProcessInstanceThread aus,
- kapselt einen ProcessInstanceThread durch ProcessEngineThreads während der Ausführung,
- ermöglicht die Ausführung von einem oder mehreren Prozessen (Eine Begrenzung ist nur durch die Leistungsmöglichkeiten des System gegeben),
- ist verantwortlich für die Erstellung neuer ProcessInstanceThreads, falls Nebenläufigkeiten auftreten (nicht realisiert),
- stoppt die Ausführung eines Threads, wenn ein Endzustand erreicht wurde,
- ist verantwortlich für die Auswertung der Bedingungen (Kontrollfluss-Bedingungen, Pre- und Post-Conditions).

Die Prozessmaschine ist nach dem Singleton-Pattern ([GHJV95, Sun04a]) implementiert, d.h. es gibt nur eine Prozessmaschine im gesamten System.

**Ausführung einer Aktivität**

Es soll hier nur kurz beschrieben werden, wie eine Aktivität durch die ProcessEngine ausgeführt wird. Eine ausführliche Beschreibung findet sich in [Ahm03]. Die Ausführung einer Aktivität besteht aus den folgenden Schritten:

- Erstellen einer ActivityInstance zur auszuführenden Aktivität
- Bestimmen des auszuführenden Handlers
- Bereitstellen der Ablaufumgebung (Session) des Handlers (bei manuellen Aktivitäten wird die Session des Benutzer, der die Aktivität aufruft, verwendet; bei automatischen Aktivitäten wird eine Session generiert)
- Verfügbar machen des Processkontextes des Prozesses in der Session
- Evaluation der Pre-Condition einer Aktivität. Sollte das Ergebnis „falsch“ sein, so kann die Aktivität nicht gestartet werden und die Ausführung bricht ab.
- Aufruf von vorher bestimmten Handlern im Falle von automatischen Aktivitäten. Bei manuellen Aktivitäten wartet der zugehörige ProcessEngineThread auf die Ausführung der manuellen Aktivität (der Zustand ändert sich von IN_EXECUTION auf SUCCESSFUL). Ausgeführt wird die Aktivität durch einen Benutzer des Systems.
- Evaluation der Post-Condition nach der Ausführung einer Aktivität. Sollte die Bedingung hier nicht erfüllt sein, so bricht die Ausführung ab.
- Ermitteln der nächsten Aktivität durch Bestimmen der von der Aktivität ausgehenden Transitionen und Evaluation der zugehörigen Kontrollfluss-Bedingungen.

### 3.2.4 Die Präsentationsschicht

Die Präsentationsschicht besteht aus zwei Teilen: Einer Benutzerschnittstelle zum Anzeigen und Ausführen manueller Aktivitäten und dem Prozesseditor (*ProcessEditor*). Die Benutzerschnittstelle wird zum Ausführungszeitpunkt durch Prozessteilnehmer verwendet, während der Prozesseditor zum Modellierungszeitpunkt durch einen Modellierer verwendet wird.

Die Benutzerschnittstelle ist vom Job-Dienst übernommen worden. Sie besteht aus zwei Listen, die jeweils die dem Benutzer zugeordneten Aktivitäten und die den Gruppen des Benutzers zugeordneten Aktivitäten anzeigt. Der Benutzer hat dann die Möglichkeit, persönliche Aktivitäten an die Gruppe zurückzugeben, oder an eine andere Person weiterzuleiten. Aufträge der Gruppe können zu den persönlichen Aktivitäten hinzugefügt werden oder an eine andere Gruppe delegiert werden. Von dieser Liste aus kann auch die Abwicklung einer manuellen Aktivität gestartet werden. Es wird dann der der Aktivität zugeordnete Handler aufgerufen, der wiederum weitere Handler aufrufen kann, oder als Resultat ein Template mit sichtbarem Inhalt anzeigt.

Der Prozesseditor ist eine Neuerung gegenüber dem Job-Dienst. Mit ihm lassen sich grafisch Prozessdefinitionen erzeugen. Dazu werden Aktivitäten als Rechteck mit dem Namen der Aktivität in der Mitte des Rechtecks dargestellt. Transitionen werden als Pfeile von der Ursprungs- zur Zielaktivität modelliert.

Allerdings sind das auch die einzigen Funktionen, die der Prototyp des Prozesseditors in der Version von [Ahm03] bietet. Es lassen sich also keine Bedingungen, keine Aktivitätstypen, keine ausführenden Rollen und keine Handler angeben. Damit ist dieser Prototyp eigentlich nicht mehr als ein Ansatz eines Prozesseditors.

In der technischen Realisierung basiert der ProcessEditor auf dem *JHotDraw*-Framework [JH03], welches eine Basis für die Entwicklung von Applikationen zum Erstellen technischer Zeichnungen bilden soll. Das Mapping zwischen der gezeichneten Prozessdefinition, also der Darstellung im Editor, und den Klassen des Prozessmodells geschieht über einen XML-Mapper, der allerdings nicht realisiert wurde. Das JHotDraw-Framework besitzt die Möglichkeit, die Zeichnung in ein spezielles, durch das Framework definiertes XML-Format zu serialisieren. Dies wäre die Grundlage, auf der der XML-Mapper die Prozessdefinition auf die Klassen des Prozessmodells abbilden könnte.

Der Prozesseditor ist eine eigene Applikation, die per Java Web Start-Technologie [Sun04c] auf dem Client-PC installiert wird. Dies hat den Vorteil, dass die Applikation auch offline, d.h. ohne Internetverbindung zum Portal, verwendet werden kann. Die Kommunikation mit dem Portal (Laden und speichern der Prozessdefinitionen) geschieht per Simple Object Access Protocol (SOAP, [WC04]). Implementierungsdetails dazu finden sich in [Ahm03].

### 3.2.5 Zusammenfassung

Das entwickelte Prozessunterstützungssystem ist auf Basis des Workflow-Standards der Workflow Management Coalition entwickelt worden. Es bietet die Möglichkeit, Prozessdefinitionen grafisch zu definieren und durch das System selber ausführen zu lassen. Es gibt aber Defizite im funktionellen Umfang dieses Systems. Es lassen sich weder Nebenläufigkeiten innerhalb eines Prozesses modellieren, noch ist die ProcessEngine in ihrer jetzigen Version fähig, diese zu verarbeiten. Des Weiteren ist die Ausdrucksstärke, d.h. die Modellierungsvielfalt des ProcessEditors stark eingeschränkt. Es lassen sich nur Aktivitäten und Transitionen in sequentieller Reihenfolge zeichnen. Konstrukte zur Modellierung von Choice bzw. Merge-Strukturen oder Fork- bzw. Join-Strukturen fehlen genauso, wie die Möglichkeiten weitere Informationen in der Prozessdefinition anzugeben (Rollen, Bedingungen, Aktivitätstypen, Handler). Auch ein standardisiertes Format zum Austausch von Prozessdefinitionen mit anderen Modellierungstools fehlt.

# Kapitel 4

# Zwei Fallstudien

In diesem Kapitel sollen zwei ausgewählte Geschäftsvorgänge vorgestellt und Konzepte in diesen Vorgängen identifiziert werden, die von einem Prozessunterstützungssystem realisiert werden müssen, um diese Geschäftsvorgänge abwickeln zu können. Die beiden Geschäftsvorgänge dienten zum Teil schon in einigen Arbeiten als Fallstudie und sind deshalb sehr gut verstanden, weshalb sie auch in dieser Arbeit als Fallstudie dienen können. Es handelt sich dabei um den Schiffsbesichtigungs-Prozess, der im WIPS IT1.1 Projekt [Wips04] entwickelt wurde, sowie den Studienarbeiter-Prozess, der von Shirley Yang in ihrem Student Project definiert wurde [Ya02]. Im dritten Abschnitt werden dann die in diesen beiden Fallstudien identifizierten Konzepte präsentiert, die von einem Prozessunterstützungssystem unterstützt werden sollen.

## 4.1 Der Schiffsbesichtigungs-Prozess

Der Schiffsbesichtigungsprozess tritt bei einem Schiffsklassifizierungs-Unternehmen wie dem Germanischen Lloyd [Gl04] in ähnlicher wie der hier beschrieben Form auf. Aufgabe dieser Unternehmen ist es, Schiffe in regelmäßigen Abständen zu besichtigen und anhand der festgestellten Ergebnisse nach festen Kriterien einer Klasse zuzuordnen. Mit Zuteilung zu einer Klasse erhält der Schiffseigentümer ein Zertifikat ausgestellt, welches Aufschluss über die Klasse gibt. Ohne Zertifikat ist der Betrieb eines Schiffes nahezu unmöglich.
Die hier vorgestellte Form des Prozesses wurde in einem Forschungsprojekt zwischen dem Arbeitsbereich Softwaresysteme [Sts04] der TUHH und dem Germanischen Lloyd definiert. Sie wurde durch den Autor remodelliert und in einigen Teilen geändert. Die Änderungen und Gründe der Remodellierung werden hier ebenfalls präsentiert. Zunächst aber sollen Ausschnitte von UML-Aktivitätsdiagrammen [JBR99], die den Prozess modellieren, dargestellt und der Prozess erläutert werden (Die vollständigen UML-Aktivitätsdiagramme werden im Anhang D dargestellt).

### 4.1.1 Beschreibung des Prozesses

In Abbildung 4.1 wird ein Auschnitt des UML-Aktivitätsdiagramms zum Schiffsbesichtigungs-Prozess gezeigt. Das vollständige Aktivitätsdiagramm wird in Abbildung D.2 präsentiert. Die Abbildung 4.2 zeigt einen Aus-

schnitt des UML-Aktivitätsdiagramms zum Subprozess „Besichtigungsdurchführung“, der Teil des eigentlichen Schiffsbesichtigungs-Prozesses ist. Das vollständige Aktivitätsdiagramm ist in Abbildung D.1 dargestellt.
Die Kunden eines Schiffsklassifizierungs-Unternehmens sind Reeder, die im Diagramm D.1 durch die Swimlane „shipOwner“ repräsentiert werden. Als „employee at GL“ werden Angestellte des Unternehmens bezeichnet, die in der Verwaltung arbeiten und die eingehenden Kundenanfragen bearbeiten. „Inspector“ sind schließlich die Angestellten, die die eigentliche Schiffsbesichtigung in einem Hafen durchführen (auch Besichtiger genannt).
Es soll noch erwähnt werden, dass es natürlich die Möglichkeiten gibt, den Prozess abzubrechen. Dies ist aber zur besseren Übersichtlichkeit in den Diagrammen nicht modelliert. Allerdings kann der „ShipOwner“ einen Prozess nicht mehr abbrechen, wenn die Besichtigung durch den Schiffsklassifizierer bereits ausgeführt wurde. In diesem Falle muss die erbrachte Leistung bezahlt werden. Dies ist eine wichtige Eigenschaft von Geschäftsprozessen. Erbrachte Leistungen müssen abgerechnet werden.

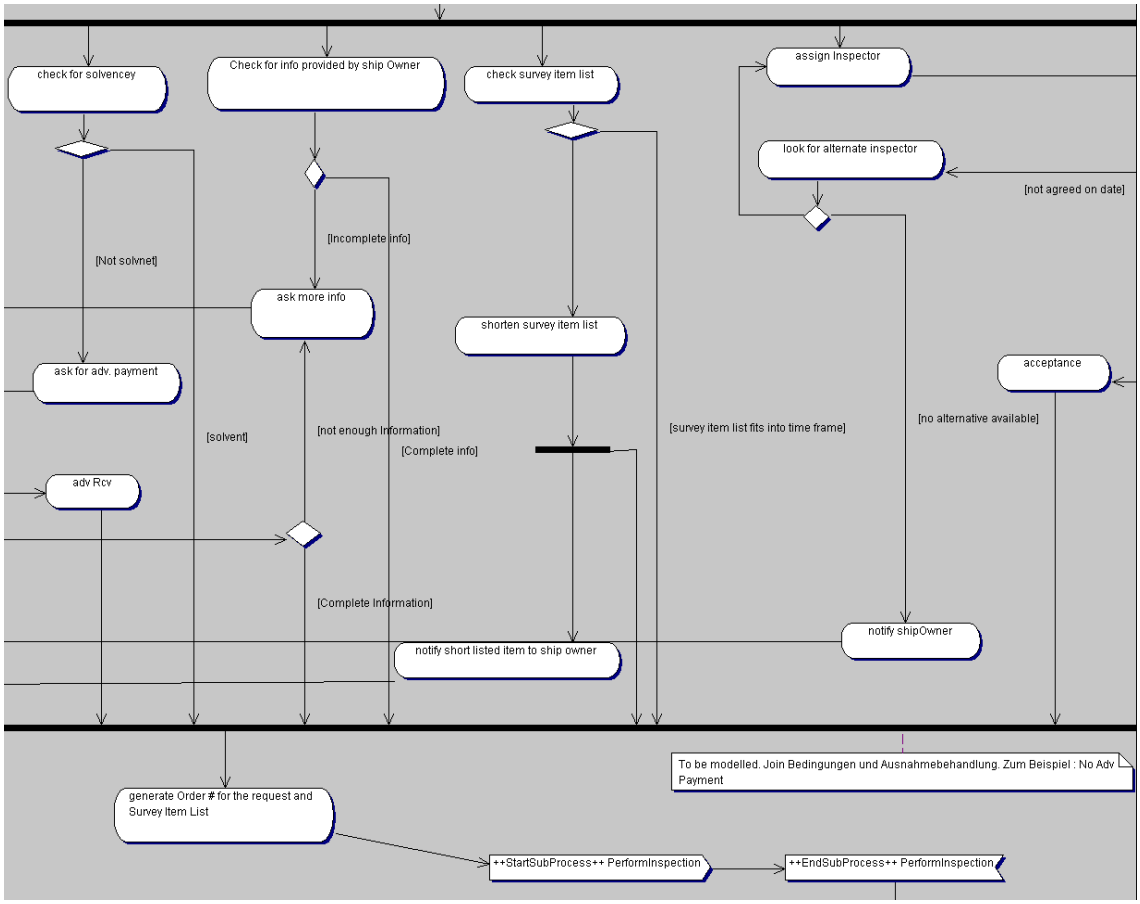


Abbildung 4.1 Ausschnitt des UML-Aktivitätsdiagramms zum Schiffsbesichtigungs-Prozess

**Der Schiffsbesichtigungs-Prozess**

Der Schiffsbesichtigungs-Prozess beginnt mit der Beauftragung durch den Kunden. Dieser wählt die Besichtigungsarten aus (Aktivität: *choose survey type an send request*“) und legt so einen Besichtigungsauftrag an. Ein Verwaltungsangestellter bearbeitet diese Anfrage weiter und prüft die Anfrage auf formelle Korrektheit (Aktivität: *employee takes care of request handling*). Sollte der Auftrag formell korrekt sein, wird er zur weiteren Bearbeitung freigegeben. Es folgt nun eine Reihe nebenläufiger Aktivitäten, die ebenfalls von einem oder mehreren Verwaltungsangestellten bearbeitet werden. Der linke Ast beginnt mit der Aktivität *check for solvency*. Sollte der Kunde nicht zahlungsfähig sein, so wird die Gebühr für die Durchführung der Besichtigung im Voraus verlangt.
Der Ast rechts daneben beginnt mit der Aktivität *Check for info provided by ship owner*“. Hier wird die inhaltliche Vollständigkeit des Auftrages geprüft. Sollten Informationen fehlen, müssen diese vom Kunden nachgereicht werden.

Der dritte Ast beginnt mit der Aktivität *check survey item List*. Dabei wird überprüft, ob die Länge der Besichtigungsliste, d.h. die Menge an Besichtigungspunkten, in den Zeitrahmen passt, oder ob die Liste gekürzt werden muss. Im Fall, dass eine Kürzung vorgenommen wird, wird der Kunde über die Besichtigungsarten informiert, die nicht durchgeführt werden können. Der vierte Ast beginnt mit der Aktivität *assign Inspector*. Es wird ein Besichtiger bestimmt, der die Besichtigung durchführen soll. Dieser hat anschließend die Möglichkeit den Auftrag abzulehnen, z.B. weil er zum genannten Zeitpunkt keine Zeit hat. Wird kein Besichtiger gewählt, so kann die Besichtigung nicht durchgeführt werden. Der Kunde wird darüber informiert und der Prozess bricht damit ab. Nachdem alle nebenläufigen Aktivitäten ausgeführt wurden, wird in der Aktivität *generate Order # for the request and Survey Item List*" eine Auftragsnummer vergeben. Der Besichtigungsauftrag ist damit angenommen und die Besichtigung kann durchgeführt werden. Dazu wird der Subprozess *PerformInspection* gestartet. Dieser wird im Anschluss erläutert. Nach Ende des Subprozesses wird aus den Ergebnissen, ein Zertifikat generiert, übergeben und die Rechnung erstellt. Sollte der Kunde die Rechnung nicht bezahlen, wird die Rechtsabteilung informiert, die alles weitere übernimmt.

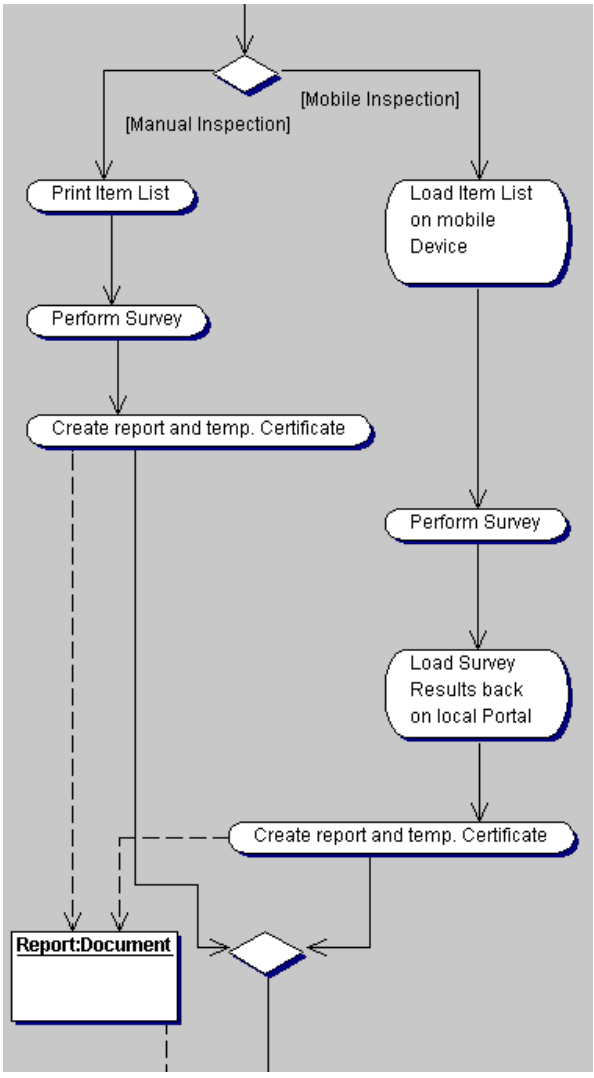


Abbildung 4.2 Ausschnitt des UML-Aktivitätsdiagramms zum Subprozess Besichtigungsdurchführung

**Der Subprozess „Besichtigungsdurchführung" (PerformInspection)**

Dieser Subprozess ist ein eigenständiger Prozess, der innerhalb des Schiffsbesichtigungsprozesses als Subprozess auftritt. Er beginnt mit der Synchronisation des persönlichen Besichtigerportals mit dem zentralen Portal (siehe Abschnitt 4.1.2 und Abbildung 4.3). Anschließend kann der Besichtiger selbst noch einmal die Liste der

Besichtigungspunkte kürzen, falls dies notwendig sein sollte. Die Verwaltung sowie der Kunde werden darüber informiert. (Aktivitäten: *Check SurveyItemList, Remove Survey Items/Types, Notify GL, Notify to ShipOwner).*
Nach diesem Teil gibt es zwei mögliche Alternativen: die manuelle Inspektion und die Inspektion mit Hilfe eines mobilen Gerätes. In der manuellen Besichtigung druckt sich der Besichtiger die Liste der Besichtigungspunkte aus, führt die Besichtigung durch und erstellt anschließend den Report und ein temporäres Zertifikat. (Aktivitäten: *Print Item List*, *Perform Survey*, *Create report and temp. Certificate*). In der mobilen Variante lädt sich der Besichtiger die Liste der Besichtigungspunkte auf ein mobiles Gerät, wie z.B. ein Handy oder Personal Digital Assistant (PDA) führt dann die Besichtigung durch, wobei er die Ergebnisse in dem mobilen Gerät speichert (siehe Abbildung 4.4). Im Anschluss werden die Ergebnisse wieder vom mobilen Gerät auf das Portal des Besichtigers geladen und anschließend wird der Report und ein temporäres Zertifikat erstellt (Aktivitäten: *Load Item List on mobile device*, *Perform Survey*, *Load Survey Results back on local Portal*, *Create report and temp. Certificate*). Den Abschluss bildet die Synchronisation mit dem zentralen Portal.

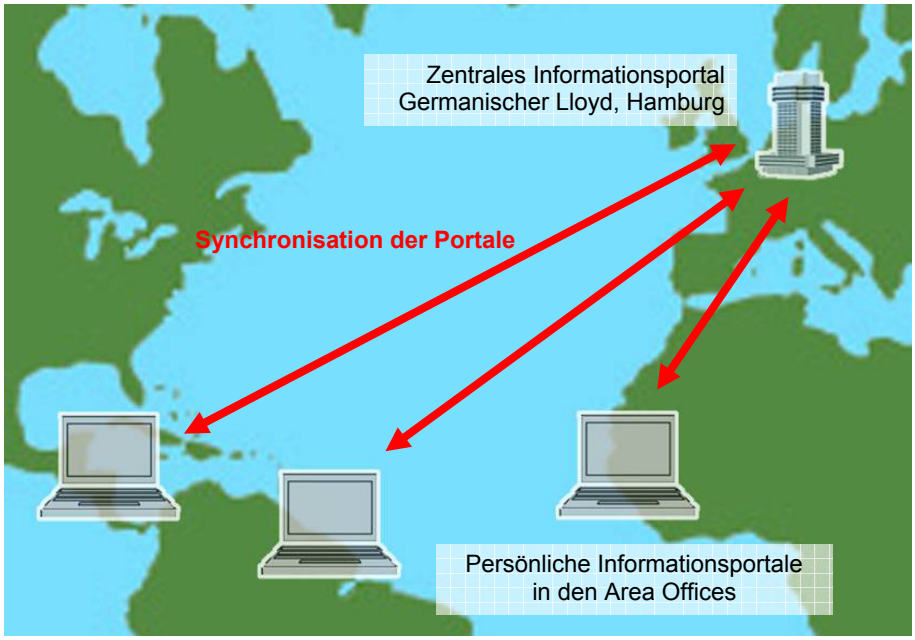


Abbildung 4.3 Persönliche Informationsportale stellen einen lokalen Arbeitskontext bereit, der mit dem zentralen Portal abgeglichen wird [HMS04].

### 4.1.2 Änderungen gegenüber der ursprünglichen Modellierung

Wie schon anfangs in Kapitel 4 erwähnt, wurde der Schiffsbesichtigungsprozess zur Verwendung in dieser Arbeit remodelliert. Es werden nun die Änderungen gegenüber der ursprünglichen Modellierung präsentiert. Das sind im Wesentlichen fünf Änderungen:

1. In der ursprünglichen Fassung fand sich noch eine Modellierung dafür, dass das Portalsystem den Eigentümer eines Schiffes, dessen Besichtigung überfällig ist, über diesen Zustand informiert, woraus ein Besichtigungsauftrag durch den Kunden eingeleitet werden kann. Diese Modellierung ist entfernt worden, da sie einen in sich eigenen Prozess darstellt, der für ein Prozessunterstützungssystem eigenständig modelliert werden kann. Der Schiffsbesichtigungsprozess soll in dieser Arbeit mit der Beauftragung durch den Kunden beginnen.
2. Es wurde eine Notation eingeführt, die es erlaubt Aktivitäten zu modellieren, die aber von einem konkret umgesetzten Geschäftsprozess nicht realisiert werden. Damit lassen sich Aktivitäten zum besseren Verständnis des Prozesses in Aktivitätsdiagrammen modellieren. In den Diagrammen D.1 und D.2 sind dies alle Aktivitäten die mit zwei Fragezeichen-Symbolen gekennzeichnet sind.

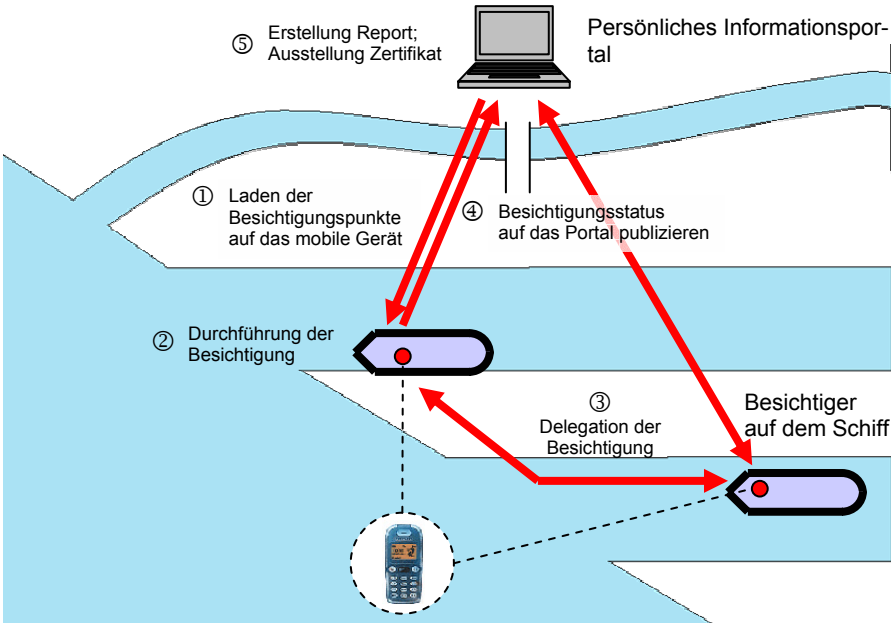


Abbildung 4.4 Durchführung der Schiffsbesichtigung mit mobilen Diensten [HMS04].

3. Als weitere Änderung gegenüber der ursprünglichen Fassung kann die Unterteilung des Schiffsbesichtigungsprozesses in den Hauptprozess und den Subprozess „Besichtigungsdurchführung“ gesehen werden. Für diese Teilung gab es zwei Gründe: Zum einen wird dadurch die räumliche Teilung zwischen Verwaltung und Ort der Besichtigungsdurchführung deutlich, die auch in der technischen Realisierung von Bedeutung ist. Jeder Besichtiger betreibt ein persönliches Informationsportal neben dem zentralen Portal, welches sich in der Hauptverwaltung befindet, am jeweiligen Niederlassungsort des Schiffsklassifizierungsunternehmens. (siehe Abbildung 4.3) Informationen müssen zwischen diesen Portalen synchronisiert werden, weshalb auch die Aktivität *Synchronize with WIPS IT 1.1 Portal* mit in den Subprozess „Besichtigungsdurchführung“ aufgenommen wurde. Die Synchronisierung zwischen den Portalen ist Gegenstand der Studienarbeit des Autors [Eel03].
   Als zweiter Grund für die Teilung kann aber auch die bessere Wiederverwendbarkeit der neuen Modellierung gesehen werden. Es ist denkbar, dass Schiffsbesichtigungen auch aus einem anderen Anlass durchgeführt werden, z.B. als Inspektion während eines Neubaus oder nach einem Schiffsunfall. So kann der Subprozess unabhängig vom Hauptprozess in verschiedenen Szenarien zum Einsatz kommen.
4. Eine weitere Änderung betrifft die Modellierung des Subprozesses. Gegenüber der ursprünglichen Variante ist die Möglichkeit der mobilen Besichtigung neu hinzugekommen. Im Forschungsprojekt WIPS IT1.1 [HLAS04, HMS04] ist für das WIPS IT1.1 Portal eine technische Realisierung dieser Besichtigungsmöglichkeit mit einem Java-fähigen Handy realisiert worden, so dass diese Möglichkeit nun auch von dem Prozess unterstützt werden soll. Näheres zur technischen Realisierung findet sich in der Master Thesis [Aziz03].
5. Die letzte Änderung ergibt sich am Ende des Schiffsbesichtigungsprozesses. Es werden nun auch ansatzweise die Rechnungserstellung und das weitere Vorgehen im Falle einer Verweigerung der Rechnungsbegleichung modelliert. Dies war vorher nicht der Fall. Allerdings ist die Modellierung nicht sehr ausführlich, denn der Schwerpunkt soll in der Besichtigungsdurchführung liegen.

## 4.2 Der Studienarbeitsprozess

Der Studienarbeitsprozess wurde von Shirley Yang in ihrem Student Project [Ya02] modelliert. Er beschreibt die Bewerbung eines Studenten auf ein Thema für eine Studien- oder Diplomarbeit am Arbeitsbereich Softwaresysteme der TUHH. Dabei wird davon ausgegangen, dass die aktuellen Themen auf der Portalseite des Arbeitsbereiches [STS04] ausgeschrieben sind und ein Interessent dort online ein Bewerbungsverfahren starten kann. Auch dieser Prozess wurde für diese Arbeit remodelliert. Im Abschnitt 4.2.1 wird der Prozess beschrieben und im Abschnitt 4.2.2 die Änderungen gegenüber der ursprünglichen Fassung angegeben.

### 4.2.1 Beschreibung des Prozesses

Der Prozessablauf ist im Diagramm D.3 zu sehen. Die Beschreibung ist aus [Ya02] entnommen worden. Danach besteht der Prozess aus 6 Phasen:

1. Bewerbung auf einen Studien-/Diplomarbeitsplatz
2. Bearbeiten der Bewerbung
3. Gespräch mit der verantwortlichen Person
4. Auswahl des Themas
5. Erstellen eines Benutzerzugangs
6. Start der Arbeit

**Bewerbung auf einen Studien-/Diplomarbeitsplatz**
Um sich auf einen Studien-/Diplomarbeitsplatz bewerben zu können, müssen alle Studenten das zweite Semester des Master-Programms vollendet bzw. das Grundstudium beendet haben (für eine Diplomarbeit alle Prüfungen ihres Studiums). Anderenfalls verwehrt das Prüfungsamt die Zulassung.
Interessiert sich ein Student für ein auf der Portalseite angegebenes Thema, so kann er sich darauf online bewerben. Dazu beantragt er einen registrierten Portalzugang und kann anschließend ein Online-Bewerbungsformular ausfüllen, seinen Lebenslauf anfügen und die Bewerbung abschicken.

**Bearbeitung der Bewerbung**
Alle Bewerbungen werden von dem Portalsystem gespeichert. Zu jeder Themenausschreibung ist mindestens eine verantwortliche Person definiert. Sobald eine Bewerbung eingeht, wird ein Subprozess gestartet, der die verantwortliche Person in vordefinierten Zeitintervallen über den Eingang benachrichtigt. Sollte die verantwortliche Person nicht reagieren, so wird die nächste verantwortliche Person in der Liste informiert. Sobald die Bewerbung durch eine verantwortliche Person bearbeitet wird, wird der Subprozess beendet.

**Gespräch mit der verantwortlichen Person**
Während der Auswahl eines Bewerbers kann die verantwortliche Person, falls notwendig, ein Gespräch mit anderen mit dem Thema vertrauten Personen arrangieren. Dazu kann der Subprozess „*Make a meeting appointment*“ genutzt werden, welcher in [Ya02] beschrieben ist. Um ein Gespräch mit einem Bewerber zu vereinbaren wird der Subprozess „*Confirm Time*“ verwendet, der zur Einigung über einen Gesprächstermin dient. Während des Gespräches wird das Thema der Studien-/Diplomarbeit besprochen und ein Zeitplan diskutiert.

**Auswahl des Themas**
Alle verfügbaren Themen werden durch das Portal aufgelistet. Natürlich können auch neue Themen angelegt werden, falls die existierenden Themen nicht ausreichen. Sobald eine grobe Einigung zwischen Bewerber und verantwortlicher Person zustande gekommen ist, muss der Bewerber einen offiziellen Antrag auf Durchführung einer Studienarbeit beim Prüfungsamt stellen. Dazu bekommt er ein Formular ausgehändigt, welches er in den

Arbeitsbereich bringen muss und auf dem später Thema und Note festgehalten werden. Sobald dieses Dokument der verantwortlichen Person vorliegt, werden Zeitplan und Thema endgültig festgelegt.

**Erstellen eines Benutzerzuganges**
Wenn alle Formalitäten erledigt sind, kann die verantwortliche Person über das Portal automatisch einen Benutzerzugang für die Rechenanlagen des Arbeitsbereiches frei schalten lassen. Die Zugangsdaten werden per Email an den Bewerber gesendet.

**Start der Arbeit**
Sobald ein Rechnerzugang existiert und das Einverständnis des Bewerbers vorliegt, startet die verantwortliche Person über das Portal den Zeitraum der Bearbeitung der Diplomarbeit bzw. Studienarbeit. Damit verbunden werden dem Bewerber auch mehr Rechte im Portal gewährt. Er kann eine persönliche Homepage anlegen, Dokumente ins Portal laden, geschützte Bereiche einsehen und die fertige Arbeit im Portal publizieren.

### 4.2.2 Änderungen gegenüber der ursprünglichen Modellierung

Es gibt nur zwei Änderungen gegenüber der ursprünglichen Fassung von Shirley Yang. Es befand sich ursprünglich noch eine Modellierung für das Publizieren von Themen für Studien- und Diplomarbeiten im Diagramm. Diese wurde entfernt, da es einen eigenen Prozess darstellt. Es soll hier nur das reine Bewerbungsverfahren betrachtet werden.
Die zweite Änderung ist die Einführung der Fragezeichen-Notation. Sie wurden an einigen Aktivitätsbezeichnungen eingefügt. Ihre Bedeutung wurde schon im Abschnitt 4.1 erläutert.

## 4.3 Identifizierte Konzepte auf Modellierungsebene

Die beiden vorgestellten Fallstudien wurden im Rahmen dieser Arbeit analysiert. Das Ziel war, Konzepte zu identifizieren, die in einem Prozessunterstützungssystem umgesetzt werden müssen, damit sich dieses Prozessunterstützungssystem zur Realisierung der Prozesse einsetzen lässt. Die identifizierten Konzepte werden nun präsentiert und erläutert. Im Einzelnen konnten folgende Konzepte identifiziert werden:

- Transitionen
- Aktivitäten
- Bedingungen
- Endstate, InitialState
- Choice
- Merge
- Fork
- Join
- Ereignisse
- „NotInSystem“-Modellierung
- Verteilte Ausführung von Prozessen
- Subprozesse
- Timeout-Strategie
- Modellierung des Objektflusses

**Transitionen und Aktivitäten**
Diese Konzepte sind bekannt. Sie wurden schon in der Arbeit [Ahm03] erwähnt und werden auch in der gängigen Fachliteratur genannt. In [WfMC99] wird eine Aktivität als logischer Schritt innerhalb eines Prozesses definiert, der eine Aufgabe darstellt. Er benötigt menschliche (manuelle Aktivität) oder maschinelle (automatische Aktivität) Ressourcen, um ausgeführt zu werden. Als Transition wird in [WfMC99] der Punkt in der Ausführung eines Prozesses definiert, in der die Ausführung einer Aktivität endet und eine neue Aktivität beginnt. Sie stellen damit also den Übergang von einer zur nächsten Aktivität dar.

**Bedingungen**
Sie stellen auch kein neues Konzept dar. Eine Bedingung ist ein logischer Ausdruck, der ausgewertet werden kann (siehe Kapitel 3.2). Es wurden Pre-Conditions, Post-Conditions und Kontrollfluss-Bedingungen an Transitionen identifiziert.

**EndState und InitialState**
Diese Konzepte dienen zur Modellierung eines definierten Anfangspunktes für einen Prozess (*InitialState*), in dem die Prozessausführung startet, und zur Modellierung eines Endpunktes (*EndState*) für die Ausführung eines Prozesses.

**Choice**
Ein Choice wird in der Fachliteratur auch als *Branch,* oder in [WfMC99] als *OR-Split* bezeichnet. Er modelliert den Umstand, dass nach einer Aktivität, die mehrere ausgehende Transitionen hat, eine Entscheidung getroffen werden muss, welche Transition weiterverfolgt wird. In den beiden Prozessen wird deutlich, dass dies sowohl Entscheidungen sein können, die nur durch eine am Prozess beteiligte Person entschieden werden können (manueller Choice), aber auch Entscheidungen, die vom System selbst getroffen werden können (automatischer Choice), nämlich durch Auswertung von Kontrollflussbedingungen an Transitionen.

**Merge**
Ein Merge wird auch als *OR-Join* bezeichnet [WfMC99]. Er modelliert die Zusammenführung von mehreren alternativen Ausführungszweigen eines Prozesses. Dabei sollte ein Merge nur verwendet werden, um tatsächlich alternative Ausführungszweige zu vereinigen. Das bedeutet, dass die Zweige sich aus einem Choice ergeben haben. Dies ist wichtig, da der Merge keine Synchronisation (Warten auf andere Ausführungszweige) vorsieht.

**Fork**
Synonyme für Fork sind auch Gabelung, oder *AND-Split*. Durch das Konzept Fork können Nebenläufigkeiten innerhalb eines Prozesses modelliert werden. Es kommt immer dann zum Einsatz, wenn sich ein Ausführungszweig in zwei oder mehr verzweigen soll, die nebenläufig ausgeführt werden sollen.

**Join**
Das Konzept Join stellt das Gegenstück zum Fork dar. Es modelliert die Zusammenführung mehrerer nebenläufiger Ausführungszweige mit Synchronisation. Dabei sind grundsätzlich zwei Synchronisations-Strategien denkbar: AND und OR. Bei der AND-Strategie wird gewartet, bis jeder der nebenläufigen Ausführungszweige die Join-Stelle erreicht hat, bei der OR-Strategie wird der Prozess weitergeführt, sobald einer diese Stelle erreicht hat, wobei die anderen Zweige beim Erreichen der Stelle dann beendet werden.

**Ereignisse**
Bei der Analyse der beiden Prozesse wurde klar, dass das Konzept der Ereignisse eine wichtige Rolle spielt, z.B. als Bedingung für einen Prozessstart. Der Schiffsbesichtigungsprozess soll gestartet werden, wenn ein neuer Besichtigungsauftrag eines Kunden vorliegt. Das kann über ein Ereignis gesteuert werden. Genauso verhält es

sich für den Studienarbeitsprozess. Er wird gestartet, sobald eine neue Bewerbung vorliegt. Aber auch in der Inter-Prozesskommunikation können Ereignisse hilfreich sein: So können die Prozesse untereinander über Ereignisse kommunizieren und ihren Fortschritt steuern [CDK01].

**„NotInSystem"-Modellierung**
Das Konzept der „NotInSystem"-Modellierung wurde auch in den beiden präsentierten Prozessen identifiziert. Es soll erlauben, Aktivitäten und Transitionen zwar zu modellieren, aber so zu kennzeichnen, dass sie nicht Teil des vom System unterstützten Prozesses sind. In den beiden Prozessen sind dies Aktivitäten, die mit zwei Fragezeichen-Symbolen gekennzeichnet sind. Dies erlaubt es, in der Modellierung auch Sachverhalte zum besseren Verständnis zu modellieren, ohne dass ein Prozessunterstützungsystem diese später realisiert. Damit sind diese Aktivitäten zwar im Modell enthalten, werden im System aber nicht berücksichtigt.

**Verteilte Ausführung von Prozessen**
Dieses Konzept wurde im Schiffsbesichtigungsprozess identifiziert. In diesem Prozess ist die Besonderheit, dass ein Teil des Prozesses auf einem anderen System, nämlich dem lokalen Portal des Besichtigers, ausgeführt wird. Dieses lokale Portal ist nicht ständig mit dem zentralen Portal verbunden und muss daher mit einem eigenen Prozessunterstützungsystem ausgestattet sein, um Prozesse steuern zu können. Im Diagramm 4.1 wird deutlich, dass dieses Problem der verteilten Ausführung gelöst werden kann, indem der Teil eines Prozesses, der von einem fremden System gesteuert werden soll, als Unterprozess modelliert wird. Es muss dann nur noch modelliert werden können, auf welchen Systemen ein Unterprozess ausgeführt werden kann.

**Subprozesse**
Subprozesse (synonym Unterprozesse) werden in beiden gegebenen Fallstudien verwendet. Sowohl der Teilprozess *Besichtigungsdurchführung*, als auch die Subprozesse *Notification*, *Appointment for meeting*, *Confirm time* sind ein Beispiel für die Verwendung des Konzeptes Subprozess. Sie unterscheiden sich allerdings in der Form der grafischen Notation. Aufgabe hier ist es eine einheitliche Notation zu finden.

**Timeout-Strategie**
Dieses Konzept wurde im Studienarbeitsprozess identifiziert, kann aber verallgemeinert werden. Dort wurde gefordert, dass bei nicht rechtzeitiger Bearbeitung einer Bewerbung eine weitere für das Projekt verantwortliche Person informiert wird. Verallgemeinert bedeutet dies, dass man jeder Aktivität einen Zeitraum zuordnen kann, innerhalb dem die Aktivität bearbeitet werden muss. Anderenfalls wird eine vorher definierte Ausnahmebehandlung ausgelöst.

**Modellierung des Objektflusses**
Durch Transitionen wird der Kontrollfluss beschrieben. Im Diagramm 4.2 des Teilprozesses *Besichtigungsdurchführung* wird ein Objekt vom Typ Report modelliert. Die Pfeile repräsentieren den Fluss der Informationen von einer Aktivität zur nächsten. Auf Modellierungsebene ist dieses Konzept, den Objektfluss separat vom Kontrollfluss zu modellieren, auch zum besseren Verständnis des Prozesses wichtig.

In den nächsten Kapiteln wird nun erläutert, wie die hier vorgestellten Konzepte durch ein Prozessunterstützungssystem realisiert werden können.

# Kapitel 5

# Anforderungen an das Prozessunterstützungssystem

In diesem Kapitel sollen weitere Anforderungen an ein Prozessunterstützungssystem zusammengetragen werden. Die Anforderungen sind gegenüber den im Kapitel 4.3 präsentierten allgemeiner, da sie nicht speziell aus der Analyse von Geschäftsvorgängen resultieren, sondern vielmehr für jedes Prozessunterstützungssystem gelten. Es wird in Kapitel 5.1 der generelle Aufbau eines solchen Systems vorgestellt, während in den folgenden Unterabschnitten die Anforderungen an die einzelnen Komponenten dieses Systems definiert werden. Dazu werden auch UML-Anwendungsfalldiagramme präsentiert und erläutert.

## 5.1 Aufbau eines Prozessunterstützungssystems

In diesem Abschnitt soll der Aufbau eines Prozessunterstützungssystems präsentiert werden. Im Kapitel 2 wurde das Referenzmodel der WfMC vorgestellt, welches die Schnittstellen eines Workflow Management Systems definiert. Ausgehend von diesem Referenzmodell werden einige Schnittstellen, die den Einschränkungen aus Kapitel 2 bezüglich der portalbasierten Prozessunterstützung entsprechen (es werden keine externen Applikationen aufgerufen), nicht berücksichtigt. Dies führt dann zu dem in Abbildung 5.1 dargestellten Aufbau eines Prozessunterstützungssystems. Im Folgenden werden nun die Aufgaben der dargestellten Komponenten erläutert.

**Prozessdefinitionswerkzeug**
Ein Benutzer kann diese Komponente dazu verwenden, eine Definition eines Geschäftsprozesses zu erstellen und dem Prozessunterstützungssystem bekannt zu machen. Da die Verwendung dieser Komponente die Interaktion mit einem Benutzer erfordert, wird sie über eine Benutzerschnittstelle verfügen müssen, das die die graphische oder textuelle Definition von Prozessdefinitionen erlaubt. Natürlich kann diese Komponente auch dazu verwendet werden, bereits erstellte Definitionen zu ändern.

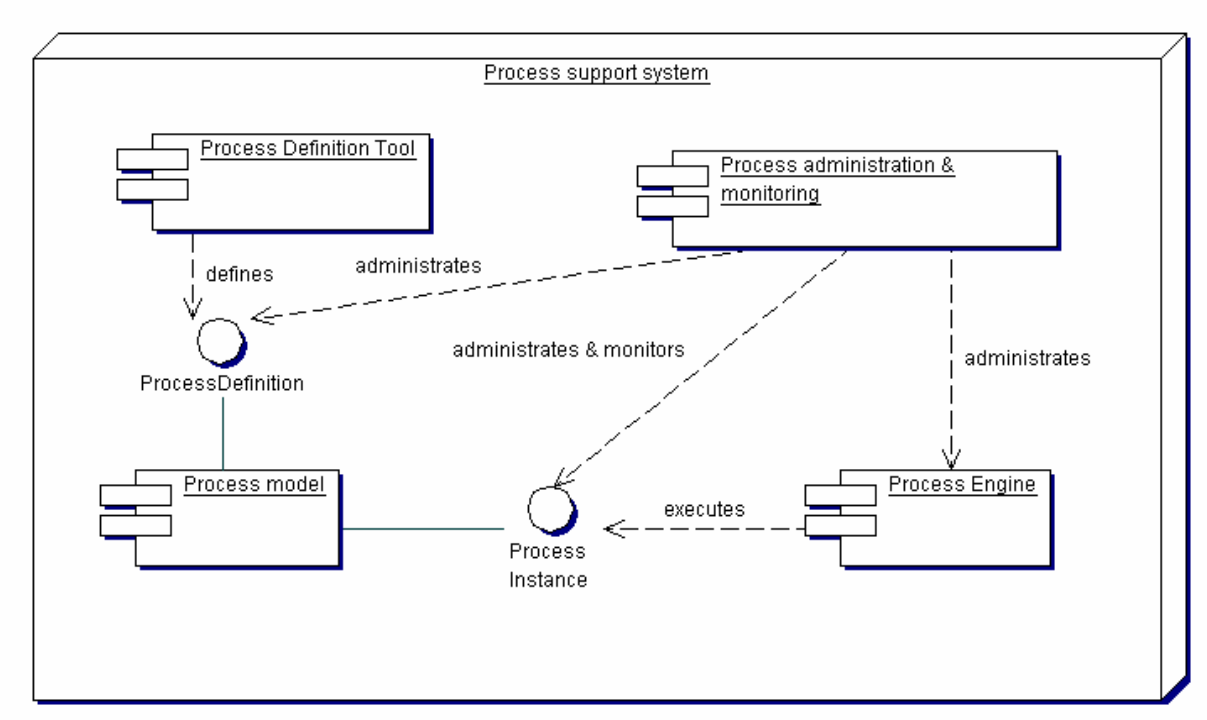


Abbildung 5.1 Komponenten eines Prozessunterstützungssystems

**Prozessmodell**
Aufgabe dieser Komponente ist es, eine technische Unterstützung für die in der Prozessdefinition modellierten Konzepte zu geben. (z.B. Aktivitäten, Bedingungen, Join/Fork-Strukturen). Die Konzepte werden auf Klassen des Systems abgebildet, für das eine Prozessunterstützung realisiert wird. Ferner werden die Schnittstellen definiert, die andere Komponenten des Systems zur Erfüllung ihrer Aufgaben benötigen. Damit wird durch diese Komponente die Basis des Systems gebildet, auf der andere Komponenten aufsetzen können.

**Prozessmaschine**
Die Prozessmaschine (*Process Engine*) stellt die Ausführungsumgebung für Prozesse dar. Sie steuert den Ablauf des Prozesses, indem sie Prozessinstanzen initiiert, die folgenden Aktivitäten bestimmt, Bedingungen auswertet, automatische Aktivitäten ausführt und manuelle Aktivitäten zur Ausführung bereitstellt. Der Ablauf des Prozesses wird durch die Prozessdefinition vorgegeben und kann durch die Prozessmaschine daher bestimmt werden.

**Prozessverwaltung**
Diese Komponente soll es den Systemadministratoren erlauben, in die Ausführung von Prozessen einzugreifen. Das umfasst das Anhalten von Prozessen, die Konfigurierung der Startparameter eines Prozesses, das Abbrechen fehlerhafter Prozesse oder Aktivitäten, sowie das Fortführen angehaltener Prozesse. Ferner soll die Komponente die Prozessverfolgung erlauben, also Informationen darüber bereitstellen, wer gerade eine Aktivität durchführt, bzw. durchgeführt hat. Dies erlaubt es den Prozessteilnehmern (Akteuren), Informationen über das augenblickliche und vergangene Prozessgeschehen abzufragen.

## 5.2 Prozessmodell und Prozessmaschine

Die Anforderungen an diese beiden Komponenten wurden weitestgehend schon in vorigen Kapiteln in anderem Zusammenhang genannt. Sie sollen an dieser Stelle noch einmal zusammengetragen werden und um ein UML-Anwendungsfalldiagramm ergänzt werden. Wie schon erwähnt stellt die Prozessmaschine die Ausführungsumgebung für Prozesse dar. Das Prozessmodell hingegen stellt die Basis für Prozessunterstützung bereit, indem es

die notwendigen Konzepte abbildet und als Klassen zur Verfügung stellt. Diese Konzepte wurden im Kapitel 3.2.1 und im Kapitel 4.3 präsentiert. Es handelt sich dabei um allgemein gültige Konzepte zur Prozessunterstützung, sowie speziell aus der Analyse der beiden Fallstudien identifizierte Konzepte auf Modellierungsebene. Dabei muss durch das Prozessmodell zu jedem dieser modellierbaren Konzepte ein Gegenstück(z.B. eine Klasse) auf Systemebene definiert werden, dass durch die Prozessmaschine verarbeitet werden kann. Die folgende Tabelle 5.1 zeigt nun noch einmal alle identifizierten Konzepte:

| Konzept | Semantik |
|---|---|
| Process Definition | Digitale Repräsentation einer Geschäftsprozessdefinition |
| Process Instance | Ausführung eines Prozesses auf Basis einer Process Definition |
| Sub Process Instance | Unterprozess einer Process Instance |
| ProcessThread | Übernimmt die sequentielle Abarbeitung von Aktivitäten innerhalb einer Process Instance |
| Activity | Definition einer (manuellen oder automatischen) Aktivität |
| ActivityInstance | Ausführung einer Aktivität |
| Transition | Gerichteter Übergang zwischen Aktivitäten |
| Process Start | Start eines Prozesses |
| Process End | Ende eine Prozesses |
| Conditions | Bedingung |
| Data Dictionary | Repository für den Prozess betreffende Informationen |
| Initial State | Anfangszustand eines Prozesses |
| Endstate | Endzustand eines Prozesses |
| Choice | Auswahl zwischen zwei oder mehr alternativen Kontrollflusszweigen (manuell oder automatisch) |
| Merge | Vereinigung mehrerer alternativer Kontrollflusszweige |
| Fork | Gabelung; es werden mehrere Kontrollflusszweige parallel ausgeführt |
| Join | (synchronisierte) Vereinigung mehrer paralleler Kontrollflusszweige |
| Ereignisse | Modelliert ein durch System oder Prozess ausgelöstes Ereignis, auf das andere Systemkomponenten, oder Prozesse reagieren können. |
| Verteilte Ausführung von Prozessen | Möglichkeit, Prozesse auf einem entfernten Prozessunterstützungssystem zu starten. |
| Timeout-Strategie | Ausnahmebehandlung bei zeitkritischen Aktivitäten |
| Objektfluss | Verwaltung des Informationsflusses separat vom Kontrollfluss |

Tabelle 5.1 – Konzepte des Prozessmodells

Die Anforderung für das Prozessmodell ist, die in der Tabelle gezeigten Konzepte in (Modell-)Klassen abzubilden. Anforderung für die Prozessmaschine ist, die durch das Prozessmodell realisierten Klassen verarbeiten zu können und so Prozesse auszuführen. Das folgende UML-Anwendungsfalldiagramm (Abbildung 5.2) beschreibt die während der Prozessausführung auftretenden Anwendungsfälle aus der Sicht der Prozessmaschine, die als (externer) Benutzer des Prozessmodells auftritt, und der Prozessteilnehmer (*User*).

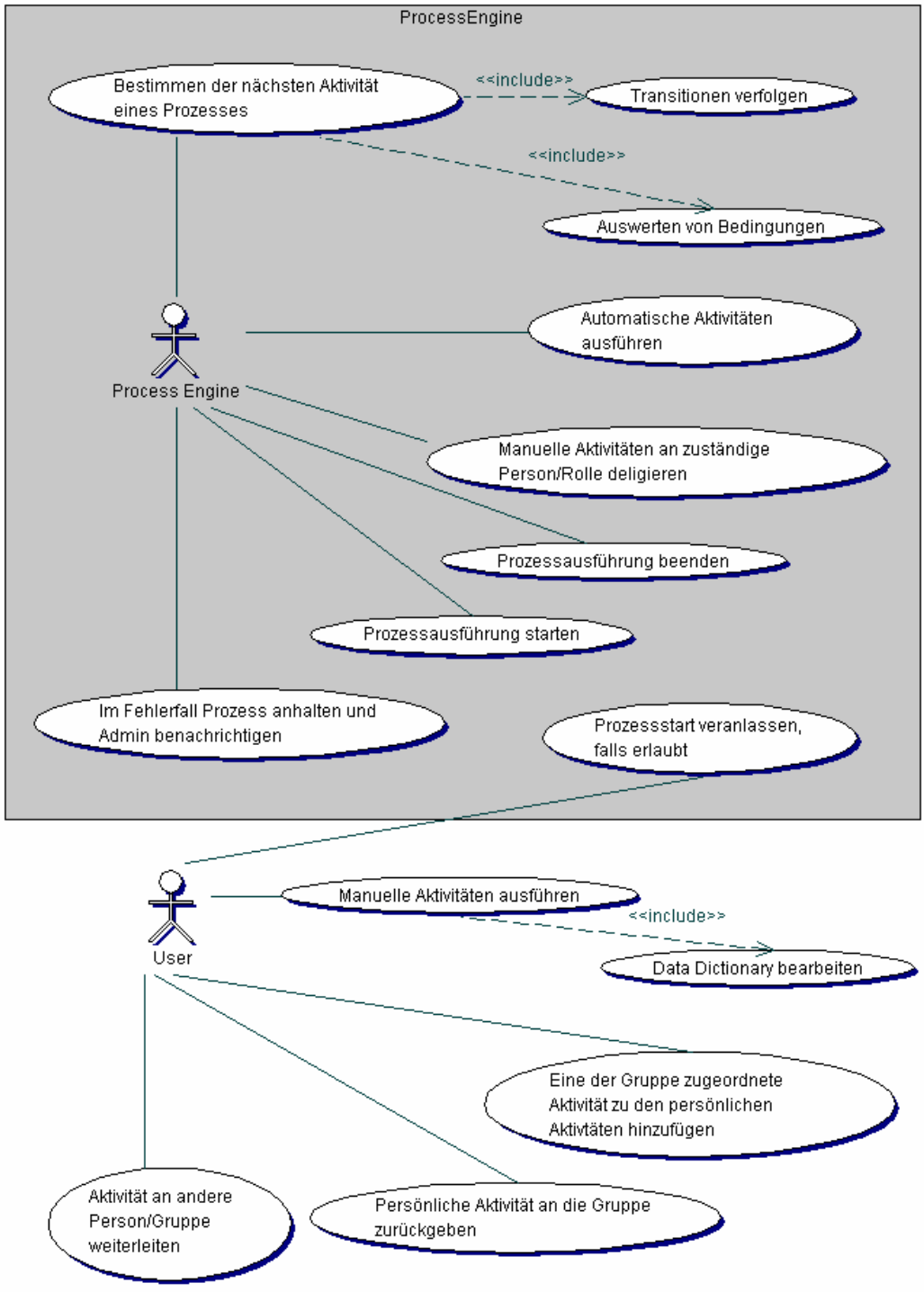


Abbildung 5.2 – UML-Anwendungsfalldiagramm für die Rollen Process Engine und User

Die auftretenden Anwendungsfälle werden im Folgenden beschrieben. Dabei ergeben sich für den Benutzer (*User*) des Prozessunterstützungssystems in Zusammenhang mit Prozessmodell und Prozessmaschine drei Anwendungsfälle (eine Aktivität bezeichnet hier stets eine manuelle Aktivität):

- **Prozessstart veranlassen, falls erlaubt:** Ein Benutzer kann einen Prozessstart veranlassen, sofern der Prozessstart durch den Administrator freigegeben ist. (siehe Abschnitt 5.4) Die Prozessmaschine würde daraufhin die *Prozessausführung starten.*
- **Manuelle Aktivitäten ausführen:** Der Benutzer führt eine manuelle Aktivität aus. Beinhaltet den Anwendungsfall *Data Dictionary bearbeiten*

- **Data Dictionary bearbeiten:** Der Benutzer kann das zentrale Informations-Repository des Prozesses einsehen und Informationen hinzufügen oder löschen. (z.B. ein PDF-Dokument hinzufügen)
- **Eine der Gruppe zugeordnete Aktivität zu den persönlichen Aktivitäten hinzufügen:** Ein Benutzer kann eine Aktivität, die der Rolle oder Gruppe, deren Mitglied er ist, zugeordnet ist, zu seinen persönlichen Aktivitäten hinzufügen. Das bedeutet, dass er die exklusive Bearbeitung dieser Aktivität übernimmt. Die anderen Gruppen- oder Rollenmitglieder können diese Aktivität nicht mehr bearbeiten.
- **Persönliche Aktivität an die Gruppe zurückgeben:** Ein Benutzer kann persönliche Aktivitäten an die Rolle/Gruppe zurückgeben, der sie zuletzt zugeordnet waren. Wenn es keine letzte Gruppe/Rolle gibt und sich auch keine Standard-Gruppe/Rolle aus der Prozessdefinition ergibt, dann steht dieser Anwendungsfall nicht zur Verfügung.
- **Aktivität an andere Person/Gruppe weiterleiten:** Ein Benutzer kann Aktivitäten, die der Rolle/Gruppe, deren Mitglied er ist, zugeordnet sind, an andere Gruppen/Rollen weiterleiten. Aktivitäten, die persönlich von ihm selbst bearbeitet werden, kann er an andere Personen weiterleiten.

Als Anforderung aus diesen Anwendungsfällen ergibt sich, dass das Prozessmodell eine oder mehrere Schnittstellen besitzen muss, die es Benutzern erlaubt, die oben genannten Anwendungsfälle auszuführen.

Für die Prozessmaschine ergeben sich die folgenden Anwendungsfälle:

- **Prozessausführung starten:** Die Prozessmaschine startet die Ausführung eines neuen Prozesses. Sie erstellt eine Prozessinstanz, einen zugehörigen Prozessthread und Data Dictionary und bestimmt die Startaktivität des Prozesses aus der Prozessdefinition. Anschließend wird der Prozessthread angewiesen, mit der Ausführung dieser Startaktivität zu beginnen. Die weitere Ausführung wird ebenfalls von der Prozessmaschine kontrolliert.
- **Automatische Aktivitäten ausführen:** Die Prozessmaschine führt eine automatische Aktivität aus.
- **Manuelle Aktivitäten an zuständige Person/Rolle delegieren:** Die Prozessmaschine erstellt für die auszuführende Aktivität die zugehörende ActivityInstance und weist diese der Gruppe/Rolle zu, die durch die Informationen des Data Dictionary vorgegeben wird, oder anderenfalls durch die Standard-Gruppe/Rolle, die durch die Prozessdefinition für diese Aktivität definiert wird. Hinter diesem Anwendungsfall steht die Forderung, dass durch Setzen von Variablen im Data Dictionary die Standard-Rolle, die für eine Aktivität in der Prozessdefinition definiert ist, umgangen werden kann. So kann durch den Prozess selber zur Laufzeit die ausführende Rolle oder Person der nächsten Aktivität gesetzt werden.
- **Bestimmen der nächsten Aktivität eines Prozesses:** Nachdem eine Aktivität ausgeführt wurde, bestimmt die Prozessmaschine die nächste Aktivität. Dazu werden die beiden folgenden Anwendungsfälle verwendet :
    - **Transitionen verfolgen:** Die Prozessmaschine bestimmt aus der Prozessdefinition ausgehend von der aktuellen Aktivität die abgehenden Transitionen und deren Ziele, welches wiederum Aktivitäten sind.
    - **Auswerten von Bedingungen:** Aus den so bestimmten Transitionen wird über die Auswertung der zugeordneten Kontrollflussbedingungen eine Transition ausgewählt, deren Bedingung erfüllt ist.
- **Prozessausführung beenden:** Sobald alle Prozessthreads einer Prozessinstanz ihre Ausführung erfolgreich beendet haben, wird die Prozessausführung beendet. Der Zustand des Prozesses wird von „aktiv“ auf „beendet“ gesetzt und das Data Dictionary des Prozesses wird zur späteren Einsicht gespeichert.
- **Im Fehlerfall Prozess anhalten und Admin benachrichtigen:** Wenn während der Prozessausführung ein Fehler auftritt, so wird der betreffende Prozess durch die Prozessmaschine angehalten. Der Zustand des Prozesses wird eingefroren. Manuelle Aktivitäten können nicht länger bearbeitet werden. Die Pro-

zessthread befinden sich im Warte-Zustand. Der zuständige Administrator wird mit einer Fehlermeldung informiert und kann die weitere Ausnahmebehandlung übernehmen.

Die hier beschriebenen Anwendungsfälle stellen neben den weiter oben präsentierten Konzepten wichtige Anforderungen für Prozessmodell und Prozessmaschine dar. Es gibt aber noch zwei weitere grundsätzliche Anforderungen an das Prozessmodell und die Prozessmaschine:

1. **Persistenz:** Der Zustand der ausgeführten Prozesse muss sich in jedem Fall nach einem Systemabsturz wiederherstellen lassen. Zu diesem Zweck ist eine persistente Speicherung der Informationen zu einem Prozess und des Prozesszustandes notwendig.

2. **Stabilität gegenüber Änderungen der Prozess-Definition:** Die Prozess-Definition definiert den Kontrollfluss für einen Prozess und ist somit die Grundlage für jeden Prozess. Falls Änderungen an der Prozessdefinition vorgenommen werden (z.B. bei einer Remodellierung), so darf diese Änderung auf keinen Fall die korrekte Ausführung bereits laufender Prozesse beeinflussen.

Damit sind alle Anforderungen an das Prozessmodell und die Prozessmaschine präsentiert worden. Sie bilden die Grundlage für die Entwicklung dieser Komponenten im Kapitel 6.

## 5.3 Prozesseditor

In diesem Abschnitt sollen die Anforderungen an einen Prozesseditor (*Process Definition Tool*) zusammengetragen werden. Sie setzen sich zusammen aus funktionalen Anforderungen und technischen Anforderungen. Die funktionalen Anforderungen ergeben sich aus den in Abschnitt 4.3 identifizierten Konzepten und werden durch das folgende Anwendungsfalldiagramm in Abbildung 5.3 beschrieben. Nach der Erläuterung der Anwendungsfälle folgt die Präsentation der technischen Anforderungen am Ende dieses Abschnittes.
Als Modellierer (*Business Process Modeler*) wird in Abbildung 5.1 ein Akteur bezeichnet, dessen Aufgabe die Erstellung und Anpassung von Geschäftsprozessdefinitionen an die realen Arbeitsabläufe eines Unternehmens ist. Die von ihm erstellten Geschäftsprozessdefinitionen sollen von dem zu entwickelnden Prozessunterstützungssystem genutzt werden können. Die dabei auftretenden Anwendungsfälle sind in Abbildung 5.3 gezeigt.
Folgende Anwendungsfälle sind in dem Diagramm gezeigt:

- **Erstellen einer Geschäftsprozess-Definition:** Der Modellierer erstellt eine neue Geschäftsprozess-Definition mit dem Prozesseditor.
- **Laden einer Geschäftsprozess-Definition vom Portal:** Der Modellierer kann eine im Portal gespeicherte Geschäftsprozess-Definition zur Bearbeitung in den Prozesseditor laden.
- **Speichern einer Geschäftsprozess-Definition im Portal:** Der Modellierer speichert eine im Prozesseditor bearbeitete Geschäftsprozess-Definition im Portal. Eine Instanz der Klasse *ProcessDefinition*, die diese Geschäftsprozess-Definition repräsentiert, wird dann im Portal neu erstellt.

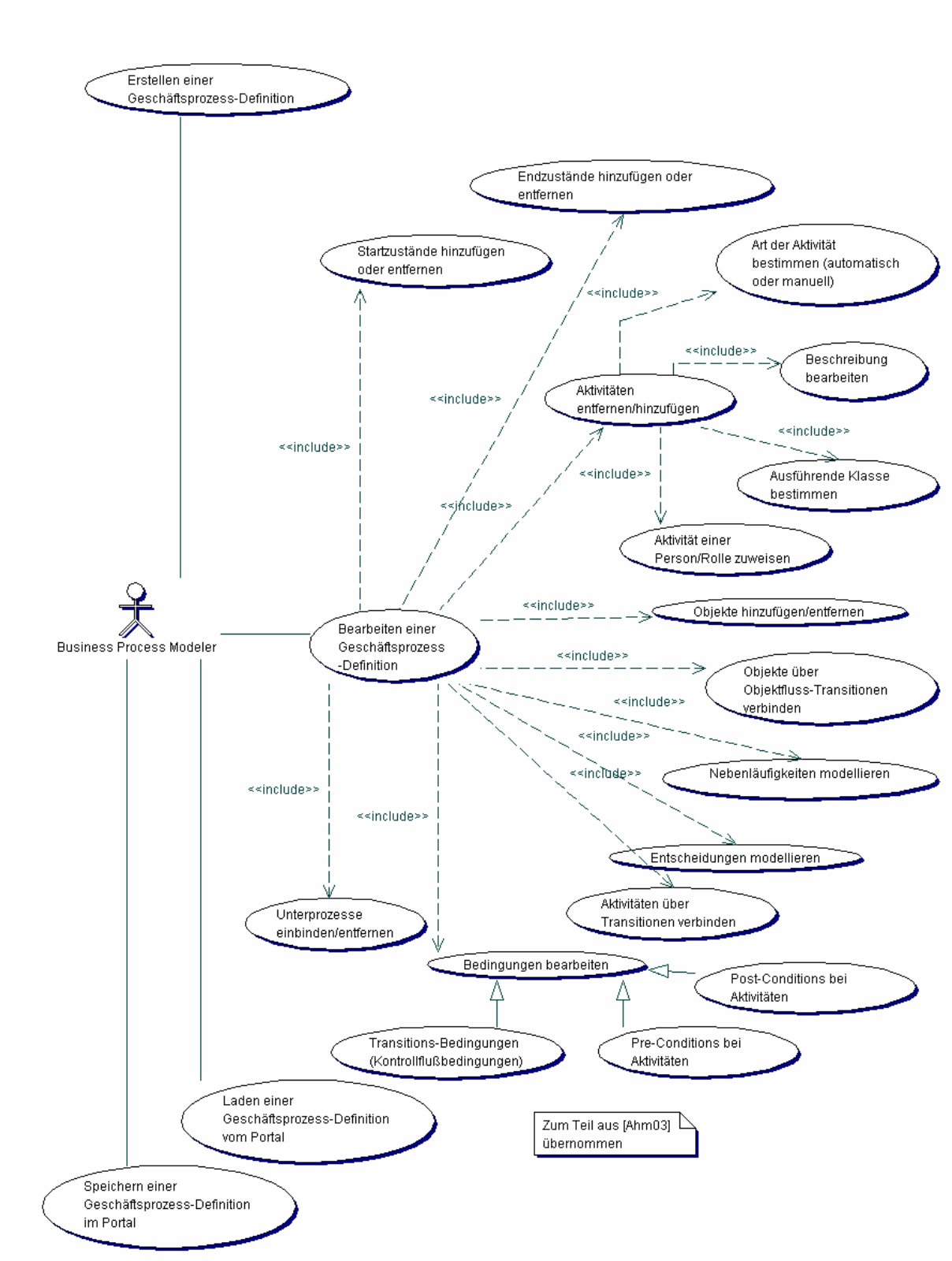


Abbildung 5.3 UML-Anwendungsfalldiagramm des Prozessmodellierers

- **Bearbeiten einer Geschäftsprozess-Definition:** Der Modellierer bearbeitet eine Geschäftsprozess-Definition mit dem Prozesseditor. Dabei können die folgenden Anwendungsfälle auftreten, die in diesem Anwendungsfall enthalten sind :
    - **Startzustände hinzufügen oder entfernen:** Der Modellierer fügt der Geschäftsprozessdefinition einen Startzustand hinzu oder entfernt diesen. Der Startzustand definiert den Beginn des Kontrollflusses für einen Prozess.
    - **Endzustände hinzufügen oder entfernen:** Der Modellierer fügt der Geschäftsprozessdefinition einen Endzustand hinzu oder entfernt diesen. Der Endzustand definiert das Ende eines Kontrollflusszweiges eines Prozesses.
    - **Aktivitäten entfernen/hinzufügen:** Der Modellierer fügt einer Geschäftsprozessdefinition eine Aktivität hinzu, oder entfernt diese. Aktivitäten werden über ihren Namen identifiziert. Das beinhaltet die folgenden Anwendungsfälle:
        - **Art der Aktivität bestimmen (automatisch oder manuell):** Der Modellierer bestimmt den Typ der Aktivität. Eine Aktivität kann manuell oder automatisch sein. Manuelle Aktivitäten werden durch Benutzer des Portals ausgeführt, automatische durch das Prozessunterstützungssystem
        - **Beschreibung bearbeiten:** Der Modellierer kann zu jeder Aktivität eine Beschreibung angeben, die die Aktivität genauer beschreibt.
        - **Aktivität einer Person/Rolle zuweisen:** Der Modellierer bestimmt die Rolle oder Person innerhalb des Portals, die diese Aktivität standardmäßig ausführen soll.
        - **Ausführenden Handler bestimmen:** Der Modellierer hinterlegt in der Prozessdefinition welcher Handler für die Ausführung dieser Aktivität zuständig ist, damit das Prozessunterstützungssystem zur Laufzeit die ausführende Komponente bestimmen kann.
    - **Aktivitäten über Transitionen verbinden:** Der Modellierer verbindet in der Geschäftsprozess-Definition definierte Aktivitäten über Transitionen und bestimmt so den Kontrollfluss von einer zur nächsten Aktivität.
    - **Bedingungen bearbeiten:** Der Modellierer kann in der Geschäftsprozess-Definition Bedingungen angeben. Diese Bedingungen lassen sich als Pre- oder Post-Conditions bei Aktivitäten, oder als Kontrollflussbedingungen an Transitionen definieren.
    - **Unterprozesse einbinden/entfernen:** Der Modellierer kann in einer Geschäftsprozess-Definition Verweise auf andere Prozess-Definitionen definieren oder entfernen. Diese werden dann als Subprozess mit in die Geschäftsprozessdefinition aufgenommen.
    - **Nebenläufigkeiten modellieren:** Der Modellierer kann in einer Geschäftsprozess-Definition Nebenläufigkeiten definieren, d.h. durch Verwendung von Fork und Join mehrere Kontrollflusszweige schaffen, die nebenläufig ausgeführt werden sollen. Dazu gehört auch die Modellierung der Synchronisation dieser parallelen Zweige.
    - **Entscheidungen modellieren:** Der Modellierer kann in einer Geschäftsprozess-Definition Entscheidungen definieren, d.h. durch Verwendung von Choice und Merge mehrere Kontrollflusszweige schaffen, die alternativ ausgeführt werden sollen. Die Entscheidung, welcher Zweig gewählt werden soll, kann manuell durch einen Benutzer oder automatisch durch das Prozessunterstützungssystem zur Laufzeit des Prozesses erfolgen. Deshalb gibt es manuelle und automatische Entscheidungen.
    - **Objekte hinzufügen/entfernen:** Zur expliziten Modellierung des Informations-/Datenflusses kann der Modellierer in einer Geschäftsprozessdefinition Informationsobjekte definieren.
    - **Objekte über Objektfluss-Transitionen verbinden:** Die Objekte innerhalb einer Prozessdefinition können durch den Modellierer über Objektfluss-Transitionen mit Aktivitäten verbun-

den werden. Durch diese Transitionen wird der Fluss der Informationen von einer zur nächsten Aktivität explizit definiert.

Neben diesen funktionalen Anforderungen gibt es noch eine Reihe technischer Anforderungen an einen Prozesseditor, die nun präsentiert werden:

1. **Erweiterbarkeit des Werkzeugs:** Diese Anforderung ist nur dann von Bedeutung, wenn als Prozesseditor ein externes Tool zum Einsatz kommt, also keine Neuentwicklung stattfindet. In diesem Fall sollte das fremde Tool idealer Weise durch selbst entwickelte Funktionen erweiterbar sein (z.B. über einen Plug-In-Mechanismus), um benötigte Funktionalität ergänzen zu können.

2. **Integrierbarkeit:** Der Prozesseditor muss in das Portal integriert werden können, so dass er bei Nutzung des Portals mit einem Internet-Browser bei Bedarf gestartet werden kann.

3. **Unterstützung von Offline-Arbeit:** Damit der Modellierer während der Modellierungsphase einer Geschäftsprozessdefinition nicht permanent mit dem Portal verbunden sein muss, um den Prozesseditor nutzen zu können, ist es außerdem von Bedeutung, dass der Prozesseditor, nachdem er einmal gestartet wurde, ohne Verbindung zum Portal genutzt werden kann. Die Verbindung zum Portal sollte nur zum Laden und Speichern von Geschäftsprozess-Definitionen und zum Starten des Prozesseditors notwendig sein. Das bedeutet, dass die Prozessdefinitionen zwischenzeitlich lokal abgelegt werden können.

4. **Unterstützung gängiger Formate:** Der Prozesseditor sollte die Möglichkeit besitzen, die erstellte Geschäftsprozess-Definition in einem bekannten Dateiformat abzulegen, um sie mit anderen Tools weiterzubearbeiten. Für die grafische Repräsentation bieten sich Grafikformate wie JPEG, GIF oder SVG an. Für eine textuelle Repräsentation kommt eine XML-Darstellung in Frage. Hier bieten sich die Standards *XML Process Definition Language* [Xpdl] oder *XML Metadata Interchange* [Xmi] an. Die Unterstützung gängiger Dateiformate erleichtert auch die Bearbeitung von Prozessdefinitionen, die mit anderen Tools erstellt worden sind.

5. **Konvertierung in Prozessmodell:** Die mit dem Prozesseditor erstellte Prozess-Definition muss von einem *Konvertierer* (entweder eine eigene Softwarekomponente oder ein Teil des Prozesseditors) in die Klassen des Prozessmodells konvertiert werden, denn die Prozessmaschine kann nur die Instanzen dieser Klassen verarbeiten. Eine grafisch dargestellte Aktivität muss also in ein Objekt umgewandelt werden, welches vom Prozessunterstützungssystem verarbeitet werden kann. Dabei treten die im UML-Anwendungsfalldiagramm in Abbildung 5.4 dargestellten Anwendungsfälle auf.
    - **Überprüfen der Geschäftsprozess-Definition auf Korrektheit:** Der Konvertierer prüft eine mit dem Prozesseditor erstellte Prozessdefinition auf semantische Korrektheit. Damit wird sichergestellt, dass die Geschäftsprozessdefinition konvertiert werden kann.
    - **Geschäftsprozess-Definition in korrespondierende Klassen des Prozessmodells konvertieren:** Der Konvertierer bildet die Prozessdefinition auf die Klassen des Prozessmodells ab und erstellt die notwendigen Objekte.

Die in diesem Abschnitt präsentierten Anforderungen werden an einen Prozesseditor gestellt, der als Teil des zu entwickelnden Prozessunterstützungssystems eingesetzt werden soll. Die Wahl bzw. Realisierung eines Prozesseditors wird in Kapitel 7 behandelt.

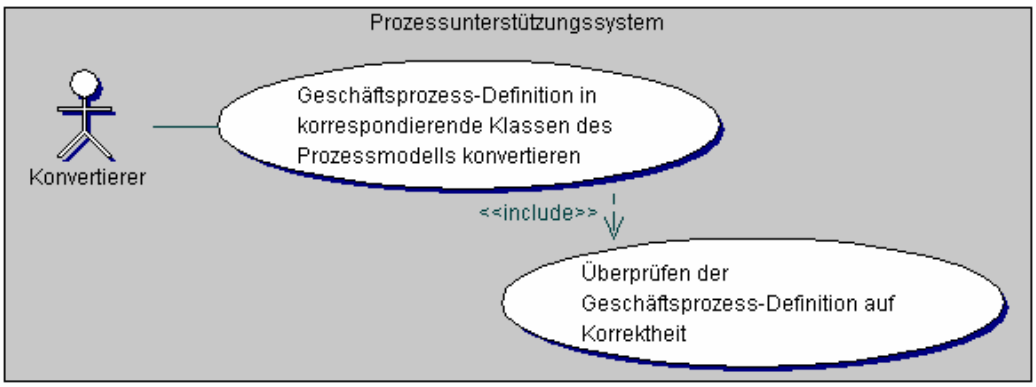


Abbildung 5.4 UML-Anwendungsfälle Konvertierer

## 5.4 Prozessverwaltung

Dieser Abschnitt beschreibt die Anforderungen, die an die Prozessverwaltung gestellt werden. Dabei umfasst die Prozessverwaltung sowohl die Monitoring- also auch die Administrationsdienste. Mit Monitoring-Diensten werden Informationen über das aktuelle Prozessgeschehen gewonnen. Die Administrationsdienste erlauben es dem Systemadministrator, Einstellungen, die die Ausführung von Prozessen betreffen, anzupassen. Die für beide Fälle präsentierten Anforderungen sind funktional und werden durch in den Abbildungen 5.5 und 5.6 dargestellten UML-Anwendungsfalldiagramme beschrieben.
Abbildung 5.5 zeigt die Anwendungsfälle, die ein zu entwickelnder Administrationsdienst für das Prozessunterstützungssystem realisieren muss. Die Nutzer dieses Dienstes sind die Administratoren des Prozessunterstützungssystems. Die Anwendungsfälle werden im Folgenden beschrieben:

- **Prozessstart konfigurieren:** Der Administrator konfiguriert für eine im Portal gespeicherte Geschäftsprozess-Definition den Prozessstart und legt so die Startparameter fest. Dabei können die folgenden enthaltenen Anwendungsfälle genutzt werden:
    - **Processstart freigeben / verbieten:** Der Administrator gibt für eine Prozess-Definition den Prozessstart frei oder verbietet ihn. Prozesse einer Prozess-Definition können nur gestartet werden, wenn der Prozessstart freigegeben ist.
    - **Anzahl parallel laufender Prozesse einer Prozessdefinition wählen:** Der Administrator bestimmt die Anzahl maximal gleichzeitiger Ausführungen einer Prozessdefinition als Prozess. Ist diese erreicht, so können keine weiteren Prozesse dieser Definition gestartet werden.
    - **Startart wählen:** Der Administrator bestimmt die Art, wie ein Prozess gestartet werden soll. Mögliche Wahlmöglichkeiten sind automatischer Start, ereignisgesteuerter Start und/oder manueller Start. Beim automatischen Start werden von dem Prozessunterstützungssystem solange Prozessinstanzen einer Prozessdefinition gestartet, bis die Anzahl maximal parallel laufender Prozesse erreicht ist. Wird eine Prozessinstanz beendet, so wird automatisch eine neue gestartet. Beispiel: Prozessdefinition A ist so konfiguriert, dass maximal fünf Prozessinstanzen gleichzeitig laufen dürfen. Die automatische Startart ist gewählt.
      Das Prozessunterstützungssystem startet nun fünf Prozessinstanzen von A. Sobald die Anzahl der gleichzeitig ausgeführten Prozessinstanzen von A unter fünf sinkt, startet das Prozessunterstützungssystem wieder so viele Prozessinstanzen von A, dass die Anzahl wieder fünf beträgt.
      Wird der ereignisgesteuerte Start gewählt, so muss ein Ereignis für den Start angegeben werden (Anwendungsfall *Ereignis für ereignisgesteuerten Start angeben*). Die Prozessinstanz wird erst dann gestartet, wenn das angegebene Ereignis eingetreten ist. Zu jedem Ereigniseintritt

wird eine Prozessinstanz gestartet, bis die Anzahl maximal parallel laufender Instanzen erreicht ist.

Bei Auswahl des manuellen Starts muss eine Rolle angegeben werden (Anwendungsfall *Rolle für manuellen Start angeben*). Portalnutzer, die dieser Rolle angehören, können dann eine Prozessinstanz starten, solange die Anzahl maximal parallel laufender Ausführungen eines Prozesses nicht überschritten wird.

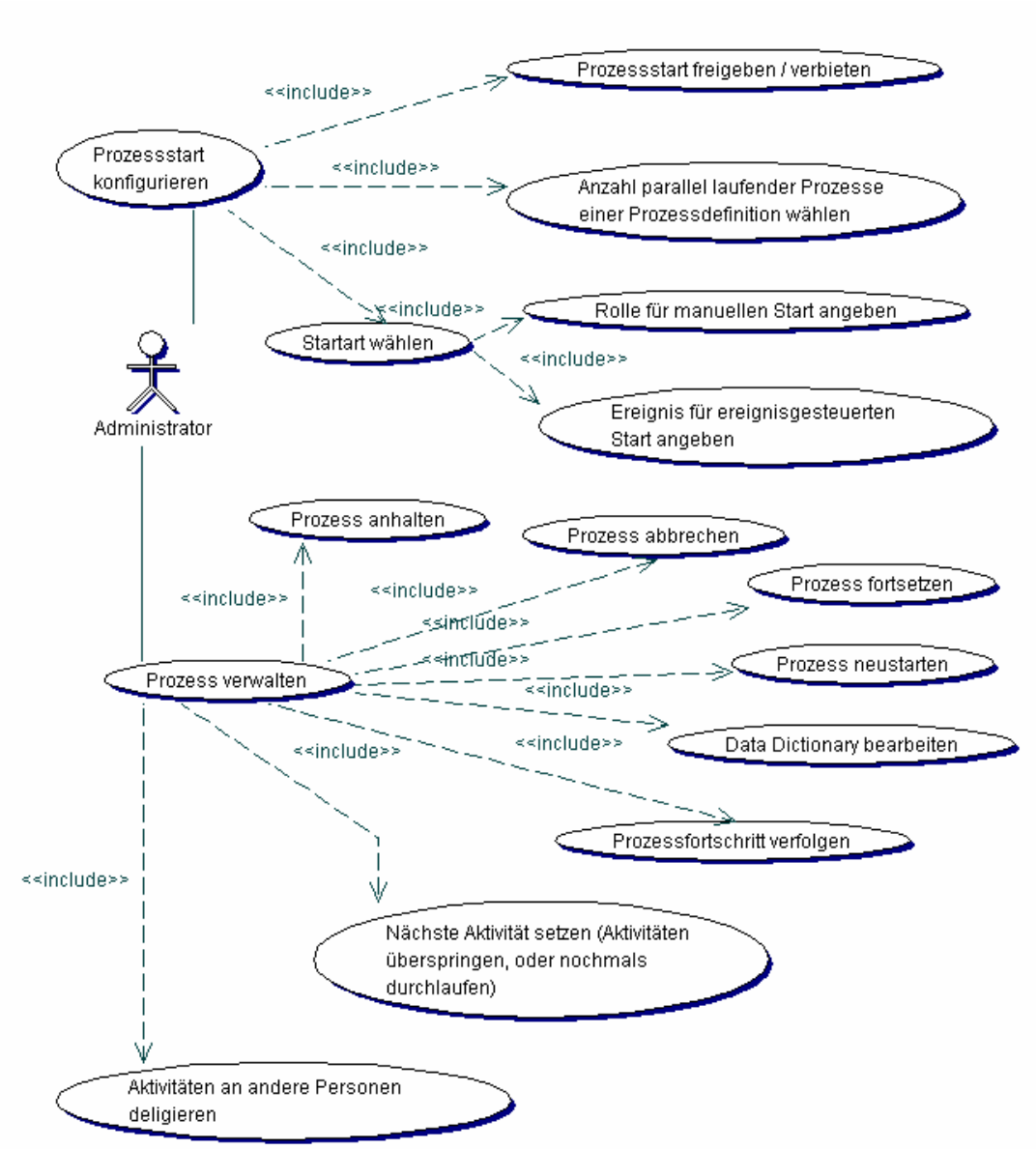


Abbildung 5.5 UML-Anwendungsfälle für die Administration von Prozessen

- **Prozess verwalten:** Der Administrator bearbeitet mit diesem Anwendungsfall laufende, abgebrochene, fehlerhafte oder beendete Prozesse einer Prozessdefinition. Dazu kommen die folgenden enthaltenen Anwendungsfälle zum Einsatz (Die möglichen Zustände einer Prozessinstanz sind identisch mit den in Abbildung 3.4 dargestellten):
    - **Prozess anhalten:** Mit diesem Anwendungsfall wird die Ausführung einer Prozessinstanz gestoppt, d.h. die Prozessmaschine bearbeitet den Prozess nicht länger. Der Zustand wird eingefroren.

- **Prozess abbrechen:** Der Administrator bricht laufende oder angehaltene Prozesse ab. Dieser Abbruch ist endgültig und kann nicht mehr rückgängig gemacht werden.
- **Prozess fortsetzen:** Mit diesem Anwendungsfall wird die Ausführung eines angehaltenen Prozesses wieder aufgenommen, d.h. die Prozessmaschine nimmt die Bearbeitung dieses Prozesses wieder auf.
- **Prozess neu starten:** Der Administrator kann eine Prozessinstanz erneut ausführen. Die Ausführung eines Prozesses wird abgebrochen und beginnt erneut mit Ausführung der Startaktivität.
- **Data Dictionary bearbeiten:** Der Administrator kann das Data Dictionary eines Prozesses einsehen und verändern. Dieser Anwendungsfall ist insbesondere dann sinnvoll, wenn aufgrund falscher Informationen im Data Dictionary Fehler in der Prozessausführung auftreten. Der Administrator kann die Informationen korrigieren.
- **Prozessfortschritt verfolgen:** Dieser Anwendungsfall stellt Informationen über das aktuelle Prozessgeschehen zur Verfügung. Dies sind Insbesondere Informationen über die gerade ausgeführten Aktivitäten eines Prozesses, die Benutzer, die diese Aktivitäten bearbeiten, den Typ der Aktivitäten, usw.
- **Nächste Aktivität setzen:** Mit diesem Anwendungsfall kann der Administrator die nächste Aktivität eines Prozessthreads setzen, und so direkt Einfluss auf den Kontrollfluss nehmen.
- **Aktivitäten an andere Personen delegieren:** Der Administrator weist persönliche manuelle Aktivitäten eines Portalnutzers einem anderem Nutzer zu, so dass dieser nun exklusiv diese Aktivitäten bearbeitet.

Das UML-Anwendungsfalldiagramm in Abbildung 5.6 beschreibt die Anforderungen an einen Monitoring-Dienst. Der Monitoring-Dienst soll es den Teilnehmern eines Prozesses während der Prozessausführung ermöglichen, mehr Informationen über den Prozess zu gewinnen. Die Anwendungsfälle überschneiden sich zum Teil mit den oben dargestellten Anwendungsfällen, nur dass die Informationen in diesem Fall direkt während einer Prozessbearbeitung zur Verfügung stehen sollen.

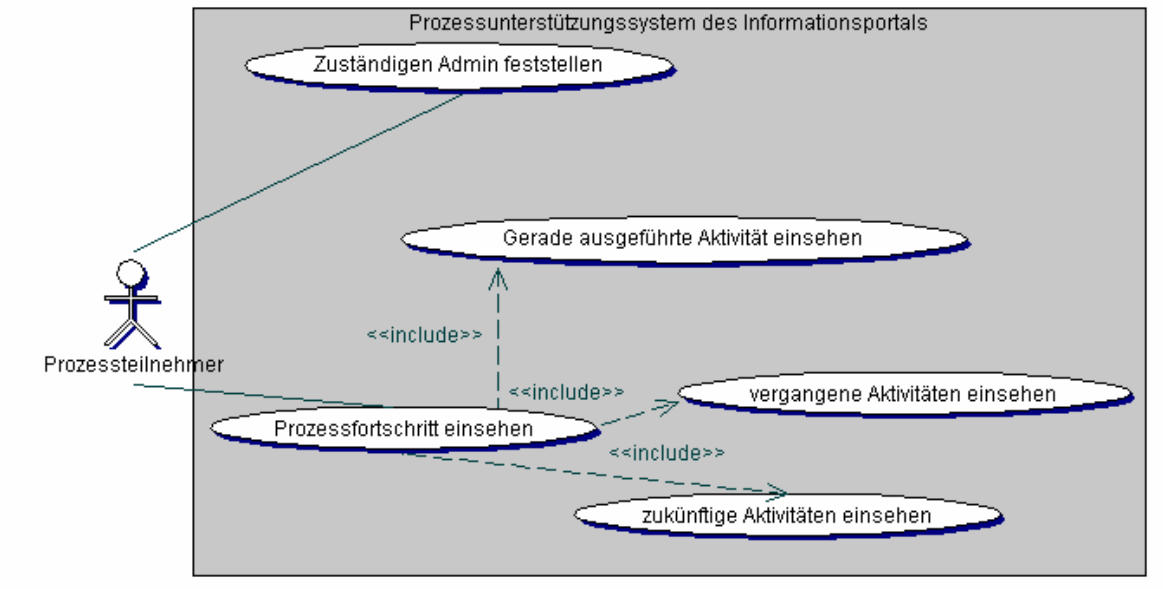


Abbildung 5.6 UML-Anwendungsfalldiagramm Monitoring

Die dabei auftretenden Anwendungsfälle werden nun beschrieben:

- **Zuständigen Admin feststellen:** Der Prozessteilnehmer kann feststellen, welcher Administrator für die Ausführung dieses Prozesses verantwortlich ist. Dadurch kann im Problemfall ein Ansprechpartner gefunden werden.
- **Prozessfortschritt abfragen:** Der Prozessteilnehmer kann alle dem System bekannten Informationen über einen Prozess einsehen, insbesondere den Zustand des Prozesses, ausgeführte Aktivitäten, aktuelle Aktivitäten, bisherige Prozessteilnehmer, aktive Prozessthreads, nächste Prozessschritte. Die enthaltenen Anwendungsfälle verdeutlichen dies noch einmal, sollen aber nicht näher erklärt werden.

Damit sind auch alle Anforderungen an den Monitoring- und Administrationsdienst bzw. die Softwarekomponente, die diesen Dienst realisiert, zusammengetragen. Sie bilden damit die Grundlage für die Entwicklung dieser Dienste, die im Kapitel 8 beschrieben wird.

# Kapitel 6

# Prozessmodell und Prozessmaschine

Nachdem in den vorangegangenen Kapiteln die Anforderungen an das Prozessmodell und die Prozessmaschine des zu entwickelnden Prozessunterstützungssystems präsentiert worden sind, soll in diesem Kapitel nun der Entwurf und die Realisierung dieser Komponenten präsentiert werden. In der Master Thesis von Ayaz Ahmed [Ahm03] wurde ein erster Prototyp für Prozessmodell und Prozessmaschine mit eingeschränkter Funktionalität entworfen. Dieser Prototyp soll in diesem Kapitel weiterentwickelt werden, so dass fehlende Funktionalität ergänzt wird und die Anforderungen der vorangegangenen Kapitel erfüllt werden. Dazu wird im ersten Abschnitt 6.1 dieses Kapitels präsentiert, welche der Anforderungen an Prozessmodell und Prozessmaschine der Prototyp in seiner bisherigen Version erfüllt, welche er nicht erfüllt und wie das Modell erweitert wird, um fehlende Funktionalität zu realisieren. Im Kapitel 6.2 werden dann die Realisierung der Erweiterung und die Prozesssemantik der implementierten Klassen besprochen. Das Kapitel 6.3 behandelt die Erweiterung der Schnittstelle zur Ausführung manueller Aktivitäten. Im Abschnitt 6.4 werden schließlich das Design und die Implementierung eines Ereignis-Modells als Teil des Prozessmodells besprochen. Im Abschnitt 6.5 wird die Erweiterung der Prozessmaschine präsentiert.

## 6.1 Erweiterung des existierenden Prozessmodells und der Prozessmaschine

Das in [Ahm03] entwickelte Prozessmodell und die ebenfalls entwickelte Prozessmaschine wurden im Kapitel 3.2 bereits vorgestellt. Die beiden Komponenten sind für die infoAssetBroker–Plattform (siehe Kapitel 2.2) realisiert worden, weshalb diese Plattform zur Realisierung von Informationsportalen auch in dieser Arbeit als Entwicklungsbasis dient. In diesem Abschnitt soll nun eine Erweiterung des existierenden Prozessmodells und der Prozessmaschine entworfen werden, wobei zunächst erläutert wird, warum diese Erweiterung notwendig ist. Die folgende Tabelle 6.1, zeigt welche der Anforderungen aus Kapitel 5.2 das bisherige Prozessmodell nicht erfüllen kann:

| Nicht erfüllte Anforderungen |
|---|
| |
| Entscheidungen werden nicht unterstützt (Konzepte Choice und Merge nicht realisiert) |
| Nebenläufige Aktivitäten werden nicht unterstützt (Konzepte Fork und Join nicht realisiert) |
| Ereignis-Behandlung wird nicht unterstützt (Auslösen von und Warten auf Ereignisse) |
| Verteilte Ausführung von Prozessen wird nicht unterstützt |
| Subprozesse werden nicht unterstützt |
| Timeout-Strategie ist nicht vorhanden |
| Anwendungsfall „DataDictionary bearbeiten“ ist nicht realisiert. |
| Anwendungsfälle zum Weiterleiten an einen anderen Benutzer, oder eine andere Rolle sind nicht realisiert. |
| Stabilität gegen über Änderungen der Prozess-Definition ist nicht gesichert |
| Dynamisches Zuweisen von Aktivitäten an Personen und Rollen zur Laufzeit nicht möglich |

Tabelle 6.1 Nicht erfüllte Anforderungen des Prototyps aus [Ahm03]

Die oben dargestellten Defizite des Prozessunterstützungssystems in seiner bisherigen Version betreffen gleichermaßen Prozessmodell wie Prozessmaschine. Das Prozessmodell stellt verschiedene Konzepte nicht zur Verfügung, während die Prozessmaschine nicht darauf ausgerichtet ist, diese Konzepte zu verstehen. Die fehlenden Anwendungsfälle sowie die Stabilität gegenüber Änderungen der Prozess-Definition betreffen allein das Prozessmodell.

**Keine Modellierung des Objektflusses**
Einen Sonderfall stellt das in der Tabelle nicht erwähnte Konzept der Objektfluss-Modellierung dar, welches aber in den Anforderungen aus Kapitel 5 erwähnt wird. Mit diesem Konzept kann der Objektfluss separat vom Kontrollfluss modelliert und definiert werden. Dieses Konzept muss an dieser Stelle nicht weiter beachtet werden, denn der Objektfluss wird mithilfe des Data Dictionary, welches durch die Klasse ProcessContext (wird auch als Prozesskontext bezeichnet) realisiert wird, gesteuert. Einmal im Prozesskontext abgelegte Informationen stehen zu jedem späteren Prozesszeitpunkt zur Verfügung, solange sie nicht gelöscht werden. Damit ist die Unterstützung eines separaten Objektflusses nicht notwendig.
Im Folgenden wird nun behandelt, wie das existierende Prozessmodell und die Prozessmaschine zu erweitern sind, damit die Anforderungen aus Tabelle 6.1 erfüllt werden. Dabei werden einige Anforderungen nur theoretisch betrachtet, d.h. sie werden nicht realisiert. Andere Anforderungen können im Rahmen dieser Arbeit nicht behandelt werden.

### 6.1.1 Erweiterung des Prozessmodells

Das konzeptuelle Klassendiagramm in Abbildung 6.1 zeigt das erweiterte Prozessmodell. Beim Vergleich mit der alten Version aus Abbildung 3.3 werden einige Änderungen deutlich, die im Folgenden besprochen und begründet werden sollen.

1. Die Klasse Endstate wird nun als Subklasse von Activity modelliert. Diese Änderung wurde eingeführt, um die Interpretation einer Prozessdefinition durch die Prozessmaschine einfacher zu gestalten. In der Prozessdefinition treten nun nur noch Aktivitäten und Transitionen auf. Damit stellt sich die konkrete Ausführung eines Prozesses als eine Folge von Aktivitäten dar, bzw. im Falle von Nebenläufigkeiten als mehrere Folgen von Aktivitäten, an deren Ende eine Endstate-Activity steht.

Dies führt zu einer „einheitlichen Modellierung“, indem eine Transition nun einen Übergang von einer Aktivität zur nächsten modelliert. Durch diese Änderung ist auch sichergestellt, dass wirklich nur Übergänge zwischen Activity-Objekten repräsentiert werden.

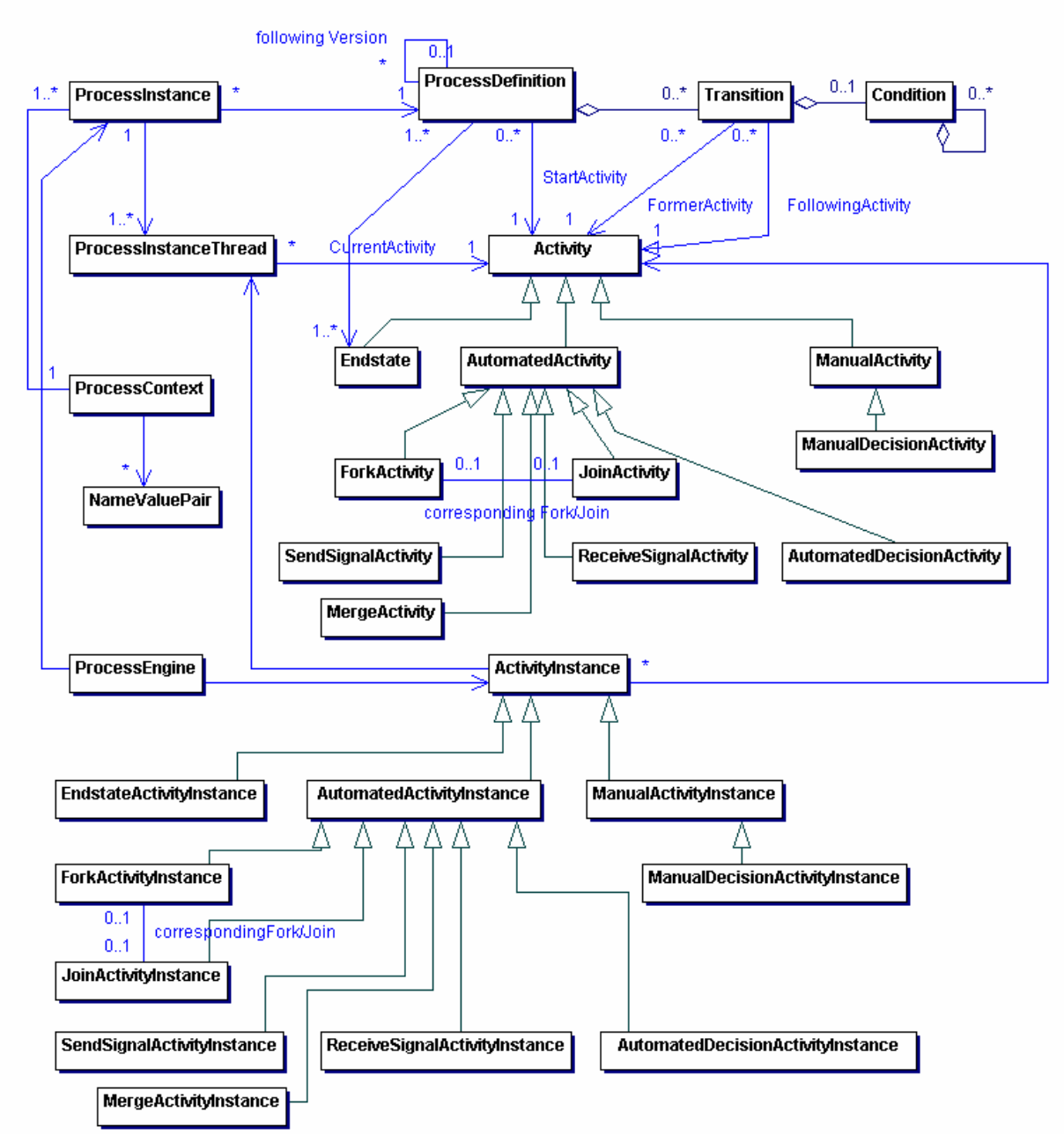


Abbildung 6.1 Konzeptuelles Klassendiagramm – Erweitertes Prozessmodell

2. Es wurden zwei neue Subklassen dem Modell hinzugefügt: AutomatedActivity und ManualActivity. Ein Objekt vom Typ AutomatedActivity repräsentiert eine automatische Aktivität und ein Objekt vom Typ ManualActivity eine manuelle Aktivität. Diese beiden unterschiedlichen Aktivitätstypen waren in dem alten Modell in der Klasse Activity zusammengefasst. Der Grund dafür liegt in der Tatsache, dass die infoAssetBroker-Plattform bei Asset-Klassen keine Mehrfachvererbung unterstützt. Was das bedeutet, wird im Abschnitt 6.2 näher erläutert.
   Allerdings ist es guter objektorientierter Entwurfs-Stil, diese beiden Typen in getrennten Klassen zu modellieren, die von einer Superklasse erben, so wie es nun auch modelliert ist.

3. Um Entscheidungen durch die Klassen des Modells darstellen zu können, wurden die Klassen AutomatedDecisionActivity und ManualDecisionActivity dem Modell hinzugefügt. Dabei repräsentiert ein Objekt vom Typ AutomatedDecisionActivity eine automatische Entscheidung, die vom System auszuwerten ist, weshalb die Klasse als Subklasse von AutomatedActivity hinzugefügt wurde. Ein Objekt vom Typ ManualDecisionActivity stellt dagegen eine manuelle Entscheidung dar, die vom Prozessteilnehmer auszuführen ist, und wird deshalb als Subklasse von ManualActivity modelliert. Allerdings hätte die Subklassenhierarchie auch über eine gemeinsame Superklasse Decision, die eine Entscheidung repräsentiert, gebildet werden können. Es sind zwei Verfeinerungen, die hier zur Subklassenbildung führen: die Repräsentation einer Entscheidung und die Unterscheidung zwischen manuell und automatisch. In dieser Arbeit wurde die Unterscheidung zwischen manuell und automatisch als Kriterium für die Subklassenhierarchie gewählt, da diese Verfeinerung für die Prozessmaschine von Bedeutung ist.
4. Die Klasse MergeActivity wurde hinzugefügt, um das Konzept „Merge“ zu unterstützen. Aus Systemsicht stellt das Zusammenführen von mehreren alternativen Kontrollzweigen eine automatische Aktivität dar, weshalb die Klasse AutomatedActivity als Superklasse gewählt wurde.
5. Um Nebenläufigkeiten zu unterstützen wurden die Klassen ForkActivity und JoinActivity dem Modell hinzugefügt. Dabei sollen die Objekte vom Typ ForkActivity das Konzept „Fork“ (oder Gabelung, siehe Kapitel 4) realisieren und die Objekte vom Typ JoinActivity das Konzept „Join“. Dabei sollen verschiedene Synchronisationsbedingungen unterstützt werden, wie z.B. AND-Synchronisation oder OR-Synchronisation (Kapitel 6.2). Die dabei auszuführenden Anweisungen betreffen nun die Prozessmaschine, so dass die beiden Klassen als Subklasse von AutomatedActivity modelliert werden. Zwischen den beiden Klassen wurde eine Referenz eingefügt, um auch eine strenge nebenläufige „Blockstruktur“ im Prozessmodell abbilden zu können. Ein solche „strukturierte Nebenläufigkeit“ ist beispielhaft in Abbildung 6.2 zu sehen. Dabei kann einer Fork-Struktur eindeutig eine passende Join-Struktur zugeordnet werden. Dieses Paar aus Fork und Join wird als korrespondierendes Fork/Join-Paar bezeichnet. Weil diese Zuordnung nicht immer möglich ist (nicht auf jede Fork-Struktur folgt eine Join-Struktur), wird in Abbildung 6.1 als Kardinalität 0..1 gewählt.

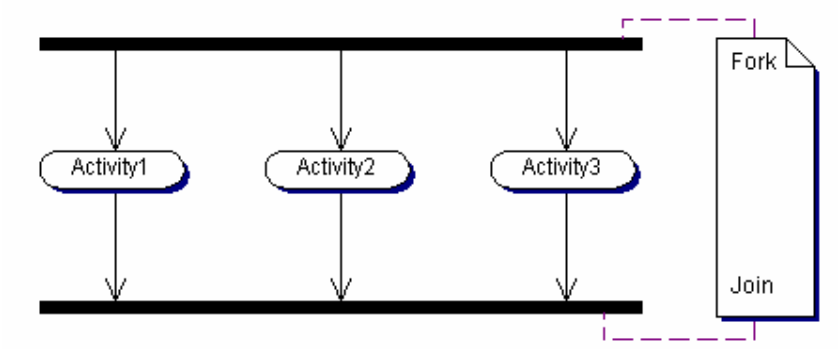


Abbildung 6.2 Korrespondierendes Fork/Join-Paar

6. Um die Ereignisbehandlung durch Prozesse zu unterstützen, wurden die Klassen SendSignalActivity und ReceiveSignalActivity dem Modell hinzugefügt. Beide sind als automatische Aktivitäten modelliert, wobei Objekte vom Typ SendSignalActivity das Auslösen eines in der Prozess-Definition angegebenen Ereignisses übernehmen und Objekte vom Typ ReceiveSignalActivity auf den Eintritt eines solchen Ereignisses warten.
7. Die gerichtete Referenz von AutomatedActivityInstance auf ProcessInstanceThread wurde gestrichen. Stattdessen zeigt nun eine Referenz von ActivityInstance auf ProcessInstanceThread.

Damit kennt jede Aktivität den Prozessthread, in dem sie ausgeführt wird. Diese Änderung wurde notwendig, damit in den Handlern, die eine manuelle Aktivität ausführen, über die ManualActivityInstance auf den Prozessthread und damit die Prozessinstanz zugegriffen werden kann.

8. Es wurde eine neue Klasse ActivityInstance eingefügt und die beiden existierenden Klassen AutomatedActivityInstance und ManualActivityInstance als Subklassen modelliert. Das Fehlen der gemeinsamen Superklasse im alten Modell ist wieder mit dem Defizit der infoAssetBroker-Plattform zu begründen: Es wird keine Subklassenbildung auf Assets unterstützt. Zu jeder der dem Modell hinzugefügten Subklassen von Activity wurde eine entsprechende Subklasse von ActivityInstance hinzugefügt, wobei die Vererbungsstruktur die gleiche ist wie bei den Activity-Klassen.
9. Um die Stabilität gegenüber Änderungen der Prozessdefinitionen zu sichern, wird der Ansatz verfolgt, Änderungen einer Prozessdefinition als neue Version im Prozessmodell darzustellen. So existiert die alte Version weiter und noch laufende Prozessausführungen können unbeeinträchtigt zu Ende geführt werden. Danach kann dann eine nicht mehr benötigte Prozessdefinition gelöscht werden. Andere Ansätze die z.B. in [Aa03] diskutiert werden, versuchen, laufende Prozesse an die geänderte Prozessdefinition anzupassen. In der hier gewählten Lösung wird eine geänderte Prozessdefinition als Nachfolger-Version der Ursprungs-Definition dargestellt. Dazu wird der Klasse ProzessDefinition eine Referenz hinzugefügt, die auf Nachfolger- bzw. Vorgänger-Versionen verweist. Eine Prozess-Definition kann danach nur maximal eine Vorgänger-Version, aber mehrere Nachfolger-Versionen haben. Es handelt sich also um eine baumstrukturierte Versionierung. Im Zusammenhang mit der Versionierung trat die Anforderung auf, Prozessdefinitionen in Gruppen einzuteilen, um den gewählten Versionierungszweig genauer zu beschreiben. Beispiel: Wir betrachten den Studienarbeitsprozess aus Kapitel 4.1. Dieser könnte variieren, je nachdem um welchen Studiengang es sich handelt. So kann durch Zuordnung zu Gruppen verdeutlicht werden, dass es zwei verschiedene Versionen desselben Prozesses sind.
10. Um die dynamische Zuweisung von Personen und Rollen zu Aktivitäten während der Laufzeit des Prozesses zu ermöglichen, wird die Klasse Activity so erweitert, dass ihre Instanzen mit einer Person oder Rolle im Prozesskontext assoziiert werden können. Diese Person oder Rolle kann zur Laufzeit des Prozesses durch den Prozess selbst bestimmt werden. Das Prozessunterstützungssystem prüft dann vor Ausführung der jeweiligen Aktivität, ob eine solche Person definiert ist und weist ihr dann die Aktivität zu.

Durch diese Modelländerungen wird die Unterstützung der in Tabelle 6.1 genannten, fehlenden Konzepte durch das Prozessmodell sichergestellt, Stabilität gegenüber Änderungen der Prozessdefinition gesichert und auch der schlechtere Entwurfsstil der Vorgängerversion korrigiert.

### 6.1.2 Unterstützung von Subprozessen

Die Klasse ProcessInstance unterstützt schon die Ausführung von Subprozessen, da ein Prozess eine Referenz auf übergeordnete Prozesse halten kann. Dadurch lassen sich Informationen darüber speichern, ob ein Prozess im Rahmen eines anderen Prozesses (des Superprozesses) ausgeführt wird. Es fehlt aber ein Mechanismus zum Starten von Subprozessen während der Prozessausführung. Dazu soll hier der Ansatz verfolgt werden, dass Subprozesse über Ereignisse gestartet werden. Es wird ein neues Ereignis definiert, dass den Start eines Prozesses auslöst. Dieses Ereignis kann in einer Prozess-Definition mithilfe von Objekten der Klasse SendSignalActivity dargestellt werden. Genauso wird ein Ereignis definiert, das das Ende einer Prozessausführung signalisiert. Eine ReceiveSignalActivity kann eingesetzt werden, um auf ein solches Ereignis zu warten.
Damit dieser ereignisbasierte Mechanismus unterstützt wird, müssen ein Ereignismodell und die entsprechenden Ereignisse definiert werden (siehe Kapitel 6.4) und die Prozessmaschine angepasst werden, damit sie die Ereignisse verarbeiten kann (siehe Abschnitt 6.1.6 und Kapitel 6.5). Wie solche Ereignisse grafisch in einer Prozess-

Definition dargestellt werden, wird im Kapitel 7 erläutert, welches die Integration eines Prozesseditors behandelt.

### 6.1.3 Zuweisung manueller Aktivitäten an andere Personen und Rollen

Um die fehlenden Anwendungsfälle zum Zuweisen einer persönlichen manuellen Aktivität an einen anderen Benutzer oder eine Rolle innerhalb des Portals zu realisieren, müssen die Klassen, die für das WorkList-Handling zuständig sind erweitert werden. Diese Klassen sind innerhalb des alten Modells als Handler-Klasse der infoAssetBroker-Plattform realisiert. Der visuelle Handler (siehe Kapitel 2 und 3) instanziiert ein Template, das eine Liste mit den persönlichen manuellen Aktivitäten und allen manuellen Aktivitäten, die einer der Rollen zugeordnet sind, deren Mitglied der aufrufende Benutzer ist, anzeigt. Das Template ist schon für diese Anwendungsfälle vorgesehen und besitzt zum Teil die notwendige Funktionalität. Allerdings sind die Handler-Klassen, die diese Funktionalität implementieren, nicht vollständig. Die Implementierung dieser Funktionalität wird im Kapitel 6.3 besprochen.

### 6.1.4 Verteilte Ausführung von Prozessen

Eine ebenfalls nicht erfüllte Anforderung ist die verteilte Ausführung von Prozessen. Diese Anforderung stammt aus der Fallstudie zur Schiffsbesichtigung aus Kapitel 4.1. Der Subprozess „Besichtigungsdurchführung“ soll auf einem entfernten Portal durchgeführt werden. Ein Lösungsansatz wird zu diesem Problem nur theoretisch behandelt. Eine Realisierung innerhalb des Prozessunterstützungssystems war aus Zeitgründen nicht möglich. Der Ansatz verwendet das in der Studienarbeit des Autors [Eel03] entwickelte Synchronisationsmodell zum Informationsaustausch zwischen zwei Portalen.
Es stellt sich nun folgende Ausgangslage dar: Prozess X auf Portal A möchte Prozess Y auf Portal B starten. Dazu werden die in 6.1.2 eingeführten Ereignisse verwendet. Da die Ereignis-Behandlung aber nur lokal auf Portal A geschieht, wird das Prozessunterstützungssystem auf Portal B darüber nicht informiert. Die Idee ist nun, die Informationen über das Ereignis „Start von Prozess Y“ mithilfe des in [Eel03] entwickelten Synchronisationsmodells auf Portal B zu übermitteln.
Da das Synchronisationsmodell aber nur Asset-Objekte synchronisiert und Ereignisse als Event-Objekte keine Assets sind (siehe Kapitel 6.4) wird ein neuer Asset-Typ eingeführt, nämlich ein NotifyEvent-Asset. Das NotifyEvent-Asset hat die Aufgabe die Informationen über ein Ereignis solange persistent zu speichern, bis sie mithilfe des Synchronisationsmodells auf ein entferntes Portal übermittelt worden sind. Anschließend kann das jeweilige NotifyEvent-Asset gelöscht werden. Dabei wird zu jedem Ereignis, das an ein entferntes Portal übermittelt werden muss, ein NotifyEvent-Asset generiert. Die Art und Weise, wie die Ereignis-Informationen in dem NotifyEvent-Asset gespeichert werden und wie aus einem NotifyEvent-Asset wieder ein Ereignis generiert wird, wird dabei durch den AssetContainer der NotifyEvent-Assets fest definiert.
Hier wird auch schon ein Nachteil dieser Methode deutlich. Es ist nicht möglich, zur Laufzeit des Portals dynamisch hinzugefügte Ereignistypen zu übermitteln. Dazu ist immer eine Änderung des AssetContainers notwendig, denn er definiert die Abbildung der Ereignisse auf NotifyEvent-Assets. Ein weiterer Nachteil ist, dass die NotifyEvent-Assets erst mit der Durchführung einer Synchronisation auf das entfernte Portal gelangen. Da eine Synchronisation in diesem Szenario üblicherweise durch den Schiffsbesichtiger gestartet wird (siehe [Eel03]), verstreicht also eine gewisse Zeitspanne zwischen dem Auslösen des Ereignisses auf Portal A und dem Erreichen der Informationen über dieses Ereignis auf Portal B. Das Verfahren eignet sich somit nicht zur Übermittlung zeitkritischer Ereignisse. Wenn allerdings dieses Verfahren nur für die beiden Ereignisse Prozessstart und Prozessende verwendet werden soll, ist es durchaus geeignet. Die beiden Ereignisse sind in dem Szenario des Besichtigungsprozesses nicht zeitkritisch. Außerdem kann in diesem Szenario nicht gewährleistet werden, dass das entfernte Portale ständig zu erreichen ist. Oft wird die Internetverbindung nur zur Synchronisation aufgebaut.

(siehe [Eel03]) Damit ist eine gesicherte zeitnahe Übermittlung zeitkritischer Ereignisse ausgeschlossen. Zur Übermittlung zeitkritischer Ereignisse müssten andere Übermittlungsverfahren eingesetzt werden. Vorraussetzung ist allerdings, dass das entfernte Portal auf einem Kommunikationskanal erreicht werden kann.
Die Abbildung 6.3 zeigt noch einmal den zeitlichen Verlauf dieser Ereignisübermittlung im Kollaborationsdiagramm.

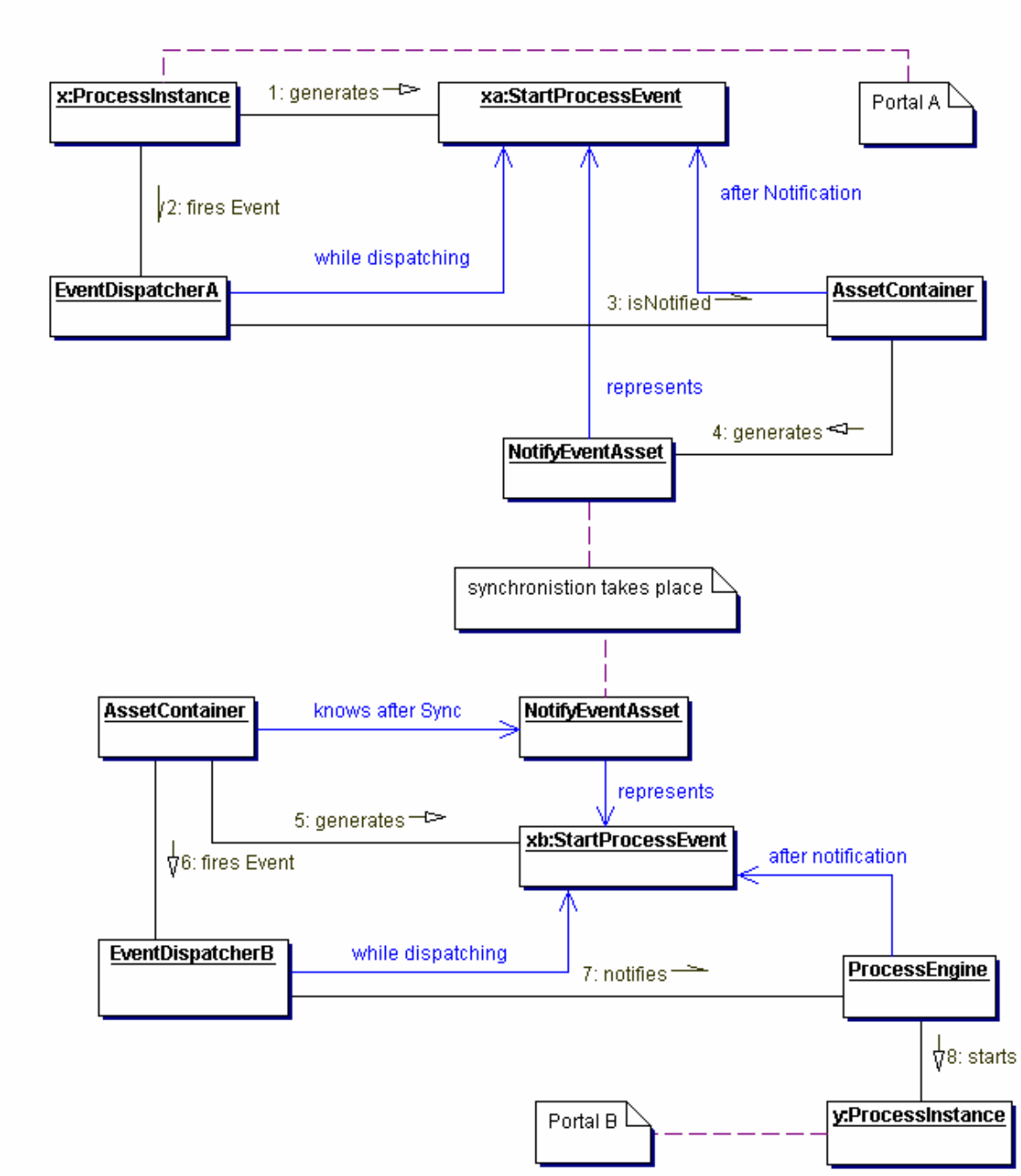


Abbildung 6.3 UML-Kollaborationsdiagramm: Synchronisation von Ereignissen mithilfe von NotifyEvent-Assets

Damit der hier präsentierte Ansatz in der Praxis funktioniert, müssen gewisse Voraussetzungen erfüllt sein:

1. Die Portale müssen auf der gleichen Portal-Plattform aufsetzen und beide das hier entwickelte Prozessunterstützungssystem betreiben. Nur so ist sichergestellt, dass das entfernte Portal auch das zu übermittelnde Ereignis versteht. Dies ist aber im WIPS-Szenario der Fall. Zentrales und entferntes Portal basieren auf der gleichen Plattform und verwenden das gleiche „Schema", d.h. die verwendeten Assettypen und die Semantik dieser Typen sind auf beiden Portalen identisch.
2. Die Prozess-Definition des zu startenden Prozesses muss auf dem entfernten Portal gespeichert sein und der Prozessstart muss dort erlaubt sein (siehe Kapitel 8).
3. Die Portale müssen eindeutig identifizierbar sein und eine Identifikation besitzen.
4. Das Ereignis Prozessstart muss Informationen enthalten, auf welchem Portal ein Prozess gestartet werden soll. Diese Information wird dann im NotifyEvent-Asset gespeichert.
5. Es muss eine (regelmäßige) Synchronisation zwischen den Portalen stattfinden und eine Synchronisationsstrategie existieren, die sicherstellt, dass die Synchronisation auch tatsächlich durchgeführt wird.

Ähnliche Lösungsvorschläge zur Kommunikation zwischen zwei Workflow-Maschinen in einem XML-basierten Format werden in [Kay01] besprochen.

### 6.1.5 Timeout-Strategie

Es wurde die Anforderung identifiziert, eine allgemeine Ausnahmebehandlung einzuführen, falls Aktivitäten nach einem gewissen Zeitraum nicht bearbeitet werden. Diese Anforderung stammt aus der Fallstudie zum Studienarbeiter-Prozess in Kapitel 4.2. Dort soll für den Fall, dass die Bewerbung eines Interessenten nicht bearbeitet wird, der zuständige Betreuer des Studienarbeitsthemas nach einem vordefinierten Zeitraum auf diese Tatsache hingewiesen werden. Es muss also möglich sein für Aktivitäten eine Timeout-Strategie zu definieren.
Eine Unterstützung dieser Anforderung wird im Rahmen dieser Arbeit ebenfalls nur theoretisch betrachtet. Dazu wird eine neue Eigenschaft der Klasse Activity definiert, die den Namen „timeoutDuration" trägt. Sie speichert den Zeitraum, der nach Start der Bearbeitung der Aktivität zur Verfügung steht, bevor ein Timeout eintritt. Dieser Zeitraum wird während der Prozessmodellierung mit dem Prozessdefinitions-Werkzeug durch den Modellierer angegeben. Das Prozessunterstützungssystem berechnet dann nach dem Start der Aktivitätsausführung den Zeitpunkt, zu dem die Bearbeitung abgeschlossen sein muss und speichert diesen in dem ActivityInstance-Objekt, das die Ausführung der Aktivität repräsentiert.
Diese Information kann dann zum Beispiel verwendet werden, um manuelle Aktivitäten, deren Ausführungszeit überschritten ist, in der WorkList eines Portalnutzers farblich hervorzuheben. Dies ist noch keine geeignete Ausnahmebehandlung, sondern nur ein Hinweis. Aber die Ausnahmebehandlung kann relativ leicht implementiert werden. Dazu wird ein neuer Prozess definiert, der regelmäßig nach Aktivitäten sucht, deren Ausführungszeit überschritten ist. Findet er eine solche Aktivität, so kann eine vordefinierte Ausnahmebehandlungen vorgenommen werden, deren Wahl vom Typ der Aktivität abhängen kann. Beispiele für solche Ausnahmebehandlungen sind: Zurückgeben einer persönlichen Aktivität an die Standard-Rolle, Weiterleiten einer persönlichen Aktivität an den Vertreter des aktuellen Bearbeiters und Informieren einer Person über die Überschreitung der Bearbeitungszeit. Die Konfiguration dieses Prozesses kann über eine eigens dafür entwickelte Benutzerschnittstelle realisiert werden.

### 6.1.6 Erweiterung der Prozessmaschine

Die oben präsentierten Erweiterungen des Prozessmodells machen eine Erweiterung der Prozessmaschine notwendig. Die Prozessmaschine muss so erweitert werden, dass sie die dem Prozessmodell neu hinzugefügten Konzepte interpretieren kann. Des Weiteren wird eine Überarbeitung (Refactoring) des Aufbaus der Prozessma-

schine sowie der Ausführungsstruktur notwendig, damit die Klasse übersichtlicher und besser wartbar wird. Ebenfalls überarbeitet werden muss der Mechanismus zum Starten von Prozessen. Die Prozessmaschine muss die Ereignisse zum Prozessstart verarbeiten und selber Ereignisse generieren können. Einzelheiten zum Thema Ereignisse werden im Abschnitt 6.4 präsentiert. Die Implementierung der geänderten Prozessmaschine wird im Kapitel 6.5 behandelt.

### 6.1.7 Nicht behandelte Anforderungen

Ein Ansatz zur Unterstützung des Anwendungsfalls „Data Dictionary bearbeiten“ durch das Prozessunterstützungssystem wird im Rahmen dieser Arbeit nicht präsentiert. Der Grund liegt in der Komplexität der Implementation. Das Data Dictionary wird im Prozessunterstützungssystem durch die Klasse ProcessContext realisiert. Es können Objekte verschiedener Typen gespeichert werden: Einzelne Werte wie String- und Integer-Variablen, aber auch Objekte und speziell Assets. Eine Benutzerschnittstelle, die eine Bearbeitung des Prozesskontextes realisiert, müsste viele Fallunterscheidungen machen, um zwischen den einzelnen Typen zu unterscheiden (Input Validation). Eine Realisierung einer solchen Schnittstelle geht damit über das Ziel dieser Arbeit hinaus und muss gesondert untersucht werden. Dennoch wird im Kapitel 11.2 eine Idee präsentiert, wie eine solche Schnittstelle ansatzweise realisiert werden könnte.

## 6.2 Realisierung der Erweiterungen für das Prozessmodell

In diesem Abschnitt soll die Realisierung der Erweiterungen detailliert behandelt werden. Es wird erläutert, wie die im Abschnitt 6.1 besprochenen Erweiterungen für das Prozessmodell konkret realisiert werden. Die wesentlichen Erweiterungen betreffen die Klassen Activity und ActivityInstance, die in den Abschnitten 6.2.1 und 6.2.2 beschrieben werden. Abschnitt 6.2.3 behandelt die Erweiterungen der Klasse ProcessDefinition und Abschnitt 6.2.4 behandelt die Änderung der Zustandsabfrage einer Prozessinstanz.

### 6.2.1 Erweiterungen der Klasse Activity

Instanzen der Klasse Activity stellen atomare Arbeitsschritte innerhalb eines Prozesses dar, die durch das Prozessunterstützungssystem auszuführen sind. Anders als im Entwurf in Abbildung 6.1 dargestellt, gibt es in der Implementierung nur eine Klasse Activity in der Realisierung, die das Verhalten und die Eigenschaften aller ihrer in Abbildung 6.1 modellierten Subklassen implementiert. Der Grund dafür liegt in der Beschränkung der zugrunde liegenden infoAssetBroker-Plattform. Die Klasse Activity ist als Asset-Klasse im bisherigen Prozessmodell realisiert, wodurch sie alle Eigenschaften von Assets (siehe Kapitel 2) besitzt. Da die infoAssetBroker-Plattform keine Vererbung von Asset-Klassen unterstützt, ist also eine Übernahme des Entwurfs aus Abbildung 6.1, wonach Activity mehrere Subklassen besitzt, nicht ohne weiteres möglich. Es wird darauf verzichtet, jede Klasse einzeln zu implementieren. Stattdessen werden alle Eigenschaften der Subklassen in der Klasse Activity implementiert. Ferner wird ein Attribut zum Feststellen des Typs der Klasse hinzugefügt. Diese Vorgehensweise wird im Folgenden als „Klassen-Überladung“ bezeichnet. Die Semantik einer Instanz einer „überladenen“ Klasse hängt danach vom Wert des Attributs ab, das den Typ angibt.
Alle den Lebenszyklus von Assets bestimmenden Eigenschaften und Methoden werden in einer AssetContainer-Klasse implementiert. Für die Asset-Klasse Activity ist dies die Klasse Activities. Die Abbildung 6.4 zeigt alle Eigenschaften und Methoden, die der Klasse Activity zur Realisierung der Konzepte aus Abschnitt 6.1 hinzugefügt wurden, sowie alle Methoden die der zugehörigen AssetContainer-Klasse hinzugefügt wurden.

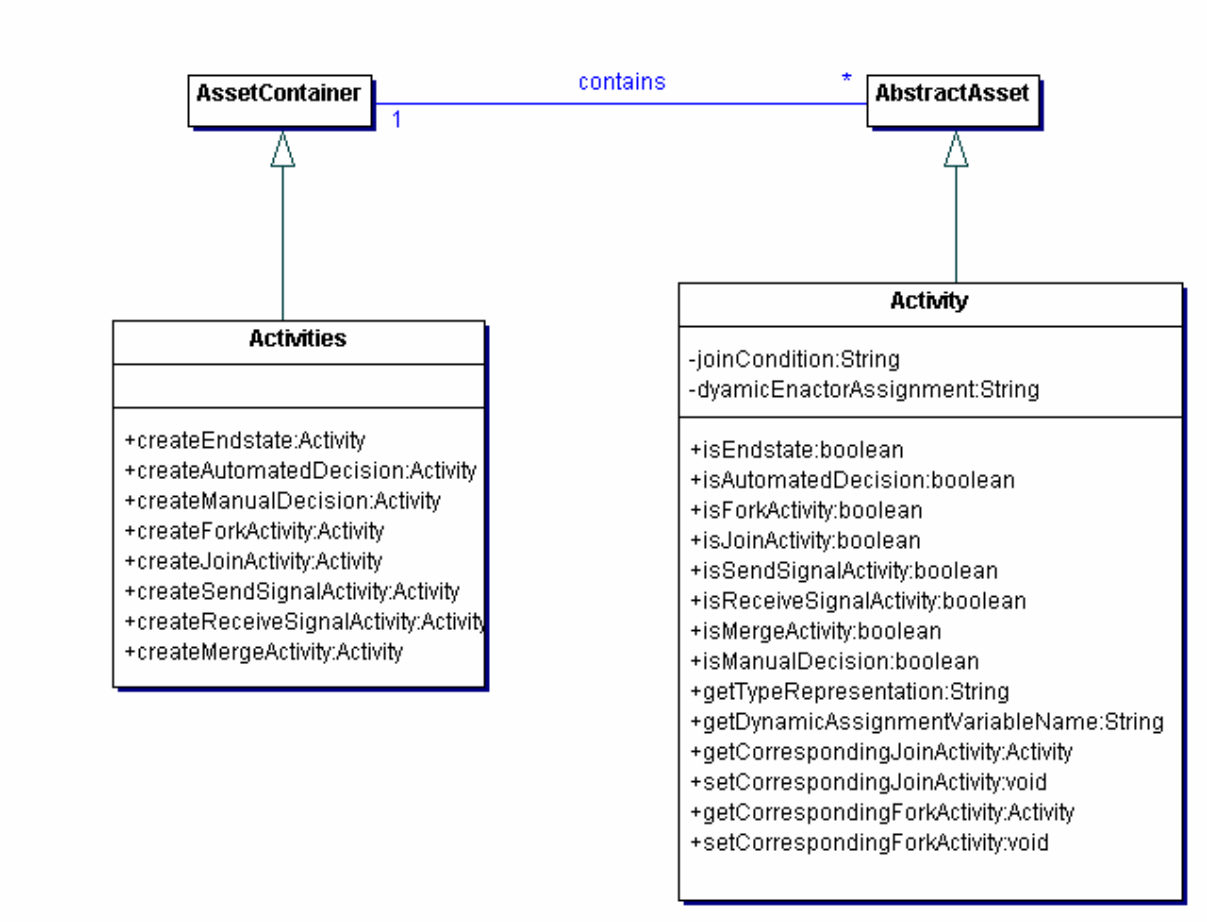


Abbildung 6.4 Abbildung mehrerer Entwurfsklassen auf eine Realisierungsklasse Activity

**Überladen der Asset-Klasse Activity**

Der Klasse Activity wurden Methoden hinzugefügt, um den Typ der Aktivität festzustellen. Eine Methode vom Typ isXXX() gibt einen Wahrheitswert zurück. Für XXX sind die im Abschnitt 6.1 neu hinzugefügten Konzepte (dort als Subklassen von Activity modelliert) einzusetzen. Es wurden also die Methoden isEndstate(), isAutomatedDecision(), isManualDecision(), isMergeActivity(), isForkActivity(), isJoinActivity(), isSendSignalActivity() und isReceiveSignalActivity() implementiert, die jeweils dann true als Rückgabewert liefern, wenn eine Instanz der Klasse Activity das jeweilige Konzept repräsentiert.

Bei der Implementierung wurde darauf geachtet, dass die modellierte Subklassen-Hierarchie aus Abbildung 6.1 eingehalten wird. Das bedeutet, dass im Falle, dass eine Instanz eine Subklasse von AutomatedActivity (aus Abbildung 6.1) repräsentiert, sowohl isAutomated(), als auch die den jeweiligen Typ abfragende isXXX()-Methode true zurückgeben. Die Methode getTypeRepresentation() liefert eine String-Repräsentation des Typs der jeweiligen Instanz der Klasse Activity.

Beispiel: Bei einer Instanz der Klasse Activity vom Typ „ForkActivity“ liefern sowohl die Methode isAutomated() als auch die Methode isForkActivity() true als Rückgabewert. Die Methode getTypeRepresentation() würde „Fork“ als Rückgabe liefern.

Um eine Instanz der Klasse Activity vom gewünschten Typ zu erstellen, wurde die AssetContainer-Klasse `Activities` um entsprechende Methoden erweitert. Diese Methoden sind von der Form createXXX(), wobei XXX wie bei den isXXX()-Methoden entsprechend zu ersetzen ist. Sie liefern als Rückgabewert eine Instanz der Klasse Activity, die den gewünschten Aktivitätstyp repräsentiert.

**Hinzufügen weiterer Eigenschaften**

Außerdem wurden der Klasse Activity noch vier weitere Eigenschaften hinzugefügt, sowie Methoden, um diese Eigenschaften zu lesen und zu ändern. Das Attribut dynamicEnactorAssignment speichert in einem String den Namen einer Variablen aus dem Prozesskontext. Diese Variable wird von der Prozessmaschine vor Ausführung der Aktivität überprüft. Verweist sie auf eine gültigen Benutzergruppe oder einen gültigen Benutzer des Portals, so wird die Aktivität dieser Gruppe oder Person zugewiesen. Damit kann zur Laufzeit des Prozesses definiert werden, welche Gruppe oder Person eine Aktivität ausführen soll. Der Wert dieses Attributs wird als Parameter an eine create-Methode übergeben und kann mit der Methode getDynamicAssignmentVariableName() abgefragt werden. Da es diese Eigenschaft nur für die Ausführung von Aktivitäten durch Benutzer oder Benutzergruppen gibt, macht ihre Definition auch nur bei manuellen Aktivitäten Sinn.

Zur Unterstützung verschiedener Join-Bedingungen (siehe Abschnitt 6.1) wurde der Klasse Activity ein String-Attribut joinCondition hinzugefügt. Dieses Attribut soll die Informationen darüber speichern, welche Synchronisationsmethode die Prozessmaschine für den Fall verwenden soll, dass eine Instanz der Klasse Activity vom Typ „JoinActivity“ ist. Es wird über einen Parameter der createJoinCondition()-Methode definiert und kann über die Methode getJoinCondition() abgefragt werden. Wird als Wert „JOIN_AND“ beim Erstellen mit der create-Methode angeben, so entspricht dies der AND-Synchronisation, wonach alle parallelen Ausführungszweige, die durch diese Instanz vom Typ „JoinActivity“ synchronisiert werden sollen, beendet sein müssen, bevor die Synchronisation abgeschlossen werden kann. Der Wert „JOIN_OR“ entspricht dagegen der OR-Synchronisation, wonach nur einer der parallelen Ausführungszweige beendet sein muss, um die Synchronisation abzuschließen.

Weiterhin wurden der Klasse noch zwei Eigenschaften hinzugefügt, um bei korrespondierenden Fork/Join-Paaren (siehe Abschnitt 6.1.1) Referenzen auf die jeweiligen korrespondierenden Join- oder Fork- Aktivitäten zu speichern. Dazu wurden die Methoden setForkActivity(Activity a), getForkActivity(), setJoinActivity(Activity a) und getJoinActivity() implementiert. Die set-Methoden dienen zum Setzen der Referenzen und die get-Methoden zum Lesen der Referenzen.

### 6.2.2 Die Klasse ActivityInstance

Eine Instanz der Klasse ActivityInstance repräsentiert die Ausführung einer Aktivität durch das Prozessunterstützungssystem. In Abbildung 3.3 ist zu sehen, dass in dem alten Prozessmodell aus [Ahm03] die beiden Klassen AutomatedActivityInstance und ManualActivityInstance diese Aufgabe übernommen haben, wobei zwischen manuellen und automatischen Aktivitäten unterschieden wurde. Wie im Abschnitt 6.1 bereits erwähnt wurde, wird die Klasse ActivityInstance als gemeinsame Superklasse eingeführt. Da auch hier bei der Implementation der Subklassen das Problem der Vererbung aus dem vorigen Abschnitt auftaucht, wird auch diese Klasse nach dem Prinzip der „Klassen-Überladung“ aus dem vorigen Abschnitt realisiert. Abbildung 6.5 zeigt alle neu hinzugefügten Methoden.

Aufgabe der zugehörigen AssetContainer- Klasse AcivityInstances ist die Verwaltung der ActivityInstance-Assets. Zum Erstellen einer Instanz der Klasse ActivityInstance wird die Methode createOrGetActivityInstance(Activity a, ProcessInstanceThread t) verwendet. Die Methode liefert als Rückgabewert eine Instanz von ActivityInstance zurück, die einer Ausführung des jeweiligen Typs der als Parameter übergebenen Instanz a von Activity durch den ProcessInstanceThread t entspricht. Dabei wird nur dann eine neue Instanz erstellt, wenn keine Instanz für die übergebene Aktivität und den ProcessInstanceThread existiert, die den Zustand „ausführbar“ (engl. executable) besitzt. Die möglichen Ausführungszustände einer ActivityInstance sind in Abbildung 3.5 dargestellt.

Die anderen Methoden die mit „get“ beginnen, liefern als Rückgabe ein oder mehrere bereits existierende Instanzen von ActivityInstance, die im Container vorhanden sind:

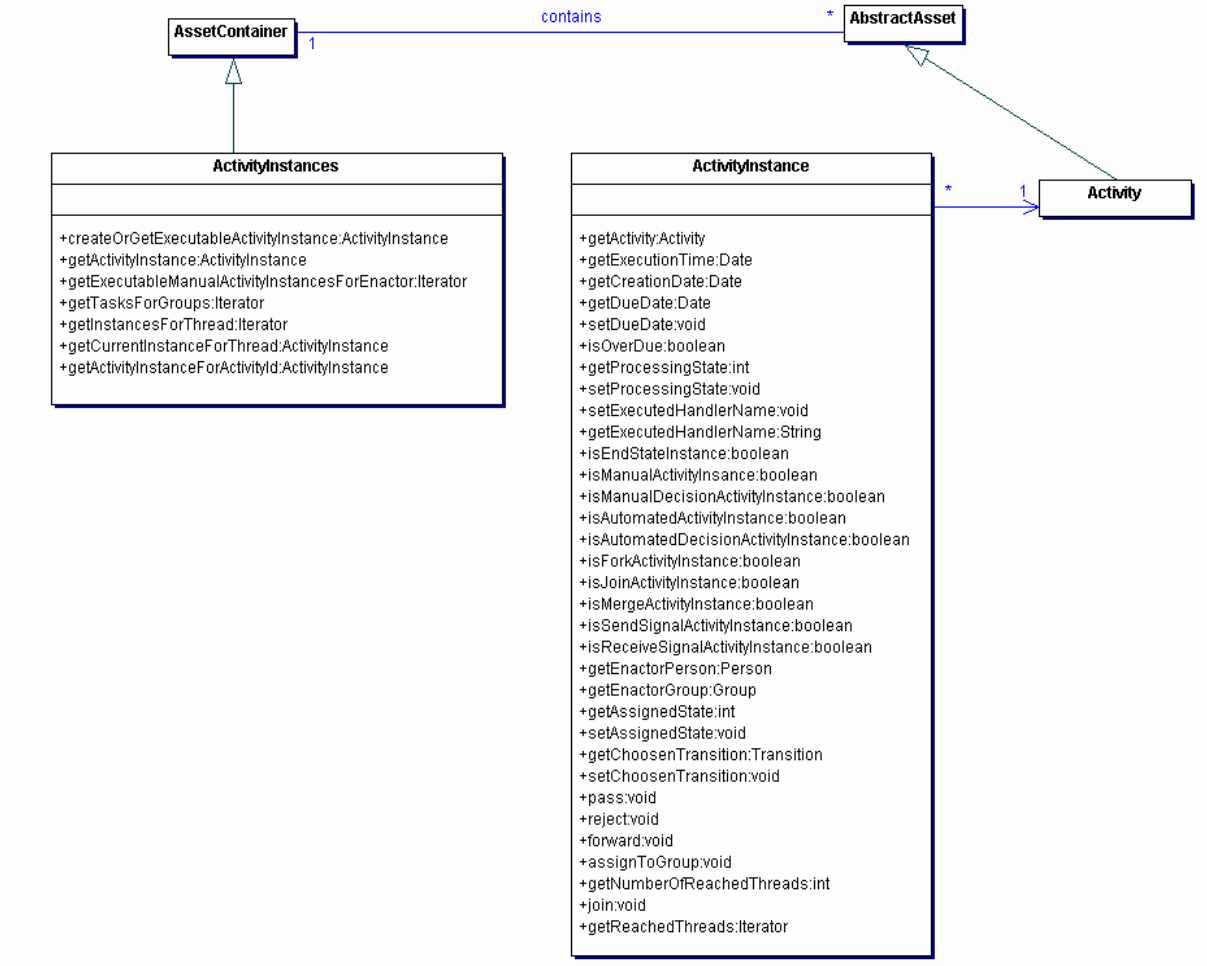


Abbildung 6.5 Klassendiagramm der erweiterten Klassen ActivityInstance und ActivityInstances

- getActivityInstance(String id): Liefert als Rückgabe das durch die als String übergebene AssetId identifizierte ActivityInstance-Asset innerhalb des Asset-Containers, falls vorhanden. Das Konzept der AssetId ist im Kapitel 2 bei der Vorstellung der infoAssetBroker-Plattform beschrieben.
- getExecutableManualActivityInstancesForEnactor(Person enactor): Diese Methode liefert als Rückgabe ein Iterator-Objekt, das über die Menge der ActivityInstance-Assets iteriert, die den Zustand „ausführbar“ besitzen und zur Ausführung dem als Parameter übergebenen Person-Objekt zugeordnet sind.
- getTasksForGroups(Iterator groups): erfüllt die gleiche Aufgabe wie getExecutableManualActivityInstancesForEnactor() für eine Menge von Group-Objekten. Ein Group-Objekt identifiziert eine Rolle innerhalb des Portals. (z.B. die Rolle Administratoren)
- getInstancesForThread(ProcessInstanceThread t): Diese Methode liefert die Menge der existierenden ActivityInstance-Assets als Iterator zurück, die dem als Parameter übergebenen ProcessInstanceThread zugeordnet sind. Der Thread hat diese Instanzen schon ausgeführt oder führt sie gerade aus.
- getCurrentInstanceForThread(ProcessInstanceThread t): Diese Methode liefert als Rückgabe die die aktuelle Aktivitätsausführung des als Parameter übergebenen Threads t repräsentierende Instanz der Klasse ActivityInstance. Falls der Thread keine Aktivität ausführt liefert die Methode null zurück.

- getActivityInstanceForActivityId(String id, ProcessInstance pi): Diese Methode interpretiert die als Sting übergebene id als AssetId eines Activity-Assets und sucht nach einer Instanz von ActivityInstance, die der durch die AssetId referenzierten Aktivität zugeordnet ist und innerhalb der Prozessinstance pi ausgeführt wird oder wurde. Existiert eine solche Instanz, so wird diese als Rückgabe geliefert, andernfalls wird null zurückgegeben.

Nach der Erläuterung der implementierten Methoden der AssetContainer-Klasse ActivityInstances sollen nun die implementierten Eigenschaften der Asset-Klasse ActivityInstance besprochen werden.
Über die Methode getActivity() lässt sich die Instanz der Klasse Activity feststellen, deren Ausführung eine Instanz von ActivityInstance repräsentiert. Die Methode getProcessInstanceThread() liefert als Rückgabe eine Instanz der Klasse ProcessInstanceThread, die die Aktivitätsausführung kontrolliert. Um zu überprüfen, welchen Typ die Aktivität besitzt, deren Ausführung durch eine Instanz der Klasse ActivityInstance repräsentiert wird, sind Methoden der Form isXXXActivityInstance() implementiert.
Die Methode getCreationDate() liefert das Erstellungsdatum einer Instanz als Rückgabewert, das den Beginn einer Aktivitätsausführung darstellt. Dieses Datum wird beim Anlegen einer Instanz festgelegt. Die Methode setDueDate(Date dueDate) dient zum Festlegen eines Fälligkeitsdatums. Mit der Methode getDueDate() lässt sich dieses Datum an einem ActivityInstance-Objekt abfragen. Ist kein Fälligkeitsdatum gesetzt, so liefert die Methode null zurück. Mit der Methode isOverDue() lässt sich einfach überprüfen, ob eine Ausführung einer Aktivität bereits überfällig ist. Mit der Methode getExecutionTime() lässt sich der Abschlusszeitpunkt der Aktivitätsausführung feststellen. Sie liefert für den Fall, dass die Aktivitätsausführung des ActivityInstance-Objektes, an dem sie aufgerufen wird, beendet ist, den Zeitpunkt als Datum zurück, zu dem der Ausführungszustand auf „successful“ gesetzt wurde. Der Ausführungszustand lässt sich mit der Methode setProcessingState(int state) setzen. Eine Übersicht über alle möglichen Ausführungszustände einer Aktivitätsausführung zeigt Abbildung 3.5. Ihre Bedeutung wird in [Ahm03] erläutert. Die verschiedenen Ausführungszustände sind in der Klasse ProcessingState definiert. Mit der Methode getProcessingState() lässt sich der Ausführungszustand eines ActivityInstance-Objektes bestimmen.
Um den Namen des Handlers, der die Ausführung der zugehörigen Aktivität übernommen hat, der ActivityInstance bereitzustellen, wird die Methode setExecutedHandlerName(String name) verwendet. Die Prozessmaschine ruft diese Methode nach erfolgter Aktivitätsausführung auf und speichert so den Namen des ausgeführten Handlers in der die Ausführung repräsentierenden Instanz der Klasse ActivityInstance. Die Methode getExecutedHandlerName() liefert den gespeicherten Namen als Rückgabewert, oder null, falls die Aktivitätsausführung noch nicht beendet ist.
Die nun im Folgenden beschriebenen Eigenschaften sind jeweils nur für die Ausführung eines konkreten Aktivitätstyps von Bedeutung.

**Ausführung von manuellen Aktivitäten (auch manuelle Entscheidungen):**
Mit der Methode setAssignmentState(int state) lässt sich der Zuweisungszustand der Aktivitätsausführung setzen. Es wird zwischen zwei Zuständen unterschieden: „Person zugewiesen“ und „Rolle zugewiesen“. Wird der Zustand „Person zugewiesen“ gesetzt, so bedeutet das, dass die zur Aktivitätsausführung gehörende Aktivität einer Person zur persönlichen Ausführung zugewiesen ist. Der Zustand „Rolle zugewiesen“ dagegen markiert diese Ausführung als verfügbar für alle Mitglieder einer Rolle. Die Methode getAssignmentState() liefert den Ausführungszustand.
Die Methoden pass(Person p), reject(), forward(Person p) und assignToGroup(Group g) dienen der Zuweisung von Personen und Rollen, die für die Ausführung der manuellen Aktivität, deren Ausführung durch das ActivityInstance-Objekt repräsentiert werden, verantwortlich sind. Dabei wird die Methode assignToGroup(..) verwendet, um eine Aktivität einer Rolle zur Ausführung zuzuweisen. Voraussetzung ist, dass

die Aktivität ausführbar ist. Wird null als Parameter übergeben, so wird die Aktivität der Standard-Rolle, die durch das die Aktivität repräsentierende Activity-Objekt definiert ist, zugewiesen. Die Methode pass(..) weist die Aktivität dem als Parameter übergebenen Person-Objekt zur Ausführung zu, wenn die Aktivität ausführbar ist, die übergebene Objekt-Referenz von null verschieden ist und vorher einer Rolle zugewiesen war. Mit der Methode forward(..) wird eine Aktivität von einer Person an eine andere Person zur Ausführung weitergeleitet. Die Aktivität muss einer Person zugewiesen sein, ausführbar sein und die übergebene Objekt-Referenz auf ein Person-Objekt darf nicht null sein. Die Methode reject(..) gibt eine Aktivität von der persönlichen Ausführung zurück an die zuletzt zugewiesen Rolle oder die Standard-Rolle. Voraussetzung ist, dass die Aktivitätsausführung noch nicht abgeschlossen ist. Um die der Aktivitätsausführung zugewiesen Group- oder Person-Objekte abzufragen, werden die Methoden getEnactorPerson() und getEnactorGroup() verwendet. Sie liefern die Referenzen der zugewiesenen Objekte oder null, falls keine Rolle oder Gruppe zugewiesen ist.

**Ausführung von Entscheidungen (sowohl manuell als auch automatisch):**
Als Besonderheit bei manuellen Entscheidungen ist die Tatsache zu sehen, dass dem Prozessunterstützungssystem mitgeteilt werden muss, welche Transition als nächstes zu aktivieren ist. Dazu wird die Methode setChosenTransition(Transition t) verwendet. Sie speichert eine Referenz auf das die gewählte Transition in dem die Ausführung repräsentierenden ActivityInstance-Objekt. Das Prozessunterstützungssystem fragt diese Transition nach Abschluss der Entscheidung mit der Methode getChoosenTransition() ab und bestimmt daraus die folgende Aktivität.
Bei automatischen Entscheidungen wertet das Prozessunterstützungssystem Kontrollfluss-Bedingungen der von einer automatischen Entscheidung ausgehenden Transitionen aus und bestimmt so die Transition, deren Bedingung erfüllt ist. Über diese Transition wird die folgende Aktivität ermittelt.

**Ausführung von Join-Aktivitäten:**
Eine weitere Besonderheit zeigt sich bei der Ausführung von Join-Aktivitäten. Da die Ausführung einer Join-Aktivität der Synchronisation mehrerer paralleler Ausführungszweige entspricht und jeder dieser Ausführungszweige durch einen anderen Prozessinstanz-Thread ausgeführt wird, muss gespeichert werden, welche Threads zu einem Zeitpunkt bereits synchronisiert wurden. Dazu werden die Methoden join(ProcessInstanceThread t), getNumberOfReachedThreads() und getReachedThreads() verwendet. Die Methode join(..) wird durch einen Prozessinstanz-Thread aufgerufen, wenn er in seiner Ausführung auf eine Join-Aktivität trifft. Er übergibt dabei eine Referenz auf sich selbst, beendet seine Ausführung und gilt damit als synchronisiert. Diese Referenz wird in dem die Ausführung der Join-Aktivität repräsentierenden ActivityInstance-Objekt gespeichert. Die Methode getNumberofReachedThreads() gibt die Anzahl der Threads zurück, die bereits synchronisiert wurden. Beim Aufruf der Methode getReachedThreads() wird die Menge der Referenzen der synchronisierten Prozessinstanz-Threads zurückgeliefert. Die konkrete Implementierung des Thread-Synchronisationsmechanismus wird in der Prozessmaschine vorgenommen und später im Abschnitt 6.5 erläutert.

### 6.2.3 Erweiterungen der Klasse ProcessDefinition

Instanzen der Klasse ProcessDefinition repräsentieren eine konkrete Prozessdefinition innerhalb des Prozessunterstützungssystems. Im Abschnitt 6.1 wurde zur Sicherung der Stabilität bei Änderungen der Prozessdefinition eine baumstrukturierte Versionierung für Prozessdefinitionen eingeführt. An dieser Stelle soll nun beschrieben werden, wie diese Versionierung implementiert ist. Der Klasse wurden zwei Referenz-Attribute hinzugefügt, die die Referenz auf die Vorgängerversion und die Menge der Referenzen auf die Nachfolger-Version persistent speichert. Es hätte auch ein Referenz-Attribut zur Realisierung der baumstrukturierten Versionierung genügt,

aber zur schnelleren Navigation innerhalb der Versionshistorie wurden zwei gewählt (Der Versionsbaum muss dann nicht immer neu berechnet werden).
Nach de, Entwurf in Kapitel 6.1 soll es möglich sein, Prozessdefinitionen nach bestimmten Bezeichnungen zu gruppieren, um so eine bessere Verwaltung der Prozessdefinitionen zu ermöglichen. Die Klasse ProcessDefinition wurde um das Attribut group, sowie die Methode getGroup() zum Lesen des Attributs erweitert. Das Attribut dient zur Aufnahme einer Gruppenbezeichnung für eine Prozessdefinition. Die Gruppe wird bei der Erzeugung einer neuen Instanz der Klasse ProcessDefinition mithilfe der dafür erweiterten Methode createProcessDefinition( Person author, String title, String description, String group ) der AssetContainer-Klasse ProcessDefinitions als neuer Parameter angegeben.
Die Referenz auf die Vorgängerversion einer Prozessdefinition wird als AssetId im Referenz-Attribut formerVersionId gespeichert. Ist keine Vorgängerversion vorhanden, so ist der Wert des Attributs null. Die Methode getFormerVersion() liefert die Objekt-Referenz auf die Vorgängerversion als Rückgabe. Die Nachfolgerversionen einer Prozessdefinition werden in einem speziellen Asset, einer Asset-Liste verwaltet. Dazu wird die Klasse AssetList, die Teil der infoAssetBroker-Plattform und selbst als Asset implementiert ist, verwendet. Ihre Instanzen können mehrere andere Assets persistent in einer Liste verwalten. In diesem Fall wird eine Instanz von AssetList dazu verwendet, die Nachfolgerversionen einer Prozessdefinition zu verwalten. Mit der dafür implementierten Methode getFollowingVersions() wird ein Iterator-Objekt zurückgeliefert, das über die Menge der Nachfolgerversionen iteriert. Die Referenz auf die Vorgängerversion und das Eintragen in die Liste der Nachfolgerversionen der entsprechenden Instanz wird von der Methode createProcessDefinitionAsNewVersion( ProcessDefinition formerDefinition, Person author, String title, String description ) der AssetContainer-Klasse übernommen. Diese Methode legt eine neue Instanz der ProcessDefinition als Nachfolgerversion der als Objekt-Referenz übergebenen Prozessdefinition formerDefinition an. Dabei wird die neu angelegte Instanz derselben Gruppe zugeordnet wie die Vorgängerversion.

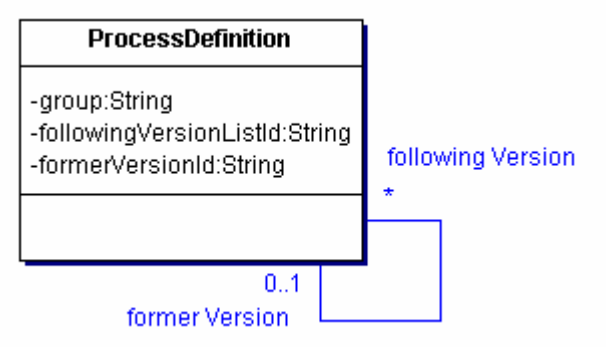


Abbildung 6.6 Versionierung von Prozessdefinitionen

### 6.2.4 Erweiterungen der Klasse ProcessInstance

Einzig die Methode getState(), mit der sich der Zustand einer Prozessinstanz feststellen lässt, wurde an das neue Prozessmodell angepasst. Dadurch, dass nun auch nebenläufige Prozessausführungen möglich sind, wird eine Prozessinstanz möglicherweise in mehreren Prozessinstanz-Threads ausgeführt. Der Zustand der Prozessinstanz errechnet sich dann aus der Menge der Zustände der einzelnen Threads. Dabei wurden folgende Definitionen aus [Ahm03] übernommen und entsprechend implementiert:

- Eine Prozessinstanz befindet sich im Zustand „INACTIVE“, wenn sich alle ihr zugeordneten Threads im Zustand „INACTIVE“ befinden. Ein Thread ist genau dann inaktiv, wenn er erstellt wurde, die Ausführung aber noch nicht gestartet wurde.

- Eine Prozessinstanz befindet sich im Zustand „HOLD“, wenn sich alle ihr zugeordneten Threads im Zustand „HOLD“ befinden. Das ist genau dann der Fall, wenn die Ausführung aller Threads gestoppt wurde.
- Eine Prozessinstanz befindet sich im Zustand „ABORTED“, wenn sich alle ihr zugeordneten Threads im Zustand „ABORTED“ befinden. Ein Thread befindet sich genau dann im Zustand „ABORTED“, wenn seine Ausführung durch einen Administrator oder einen Fehler während der Ausführung abgebrochen wurde.
- Eine Prozessinstanz befindet sich im Zustand „FINISHED“, wenn sich alle ihr zugeordneten Threads im Zustand „FINISHED“ befinden. Ein Thread befindet sich im Zustand „FINISHED“, wenn seine Ausführung erfolgreich abgeschlossen wurde.
- Sollte ein oder mehr Threads den Zustand „ABORTED“ haben und alle anderen Threads sich im Zustand „FINISHED“ befinden, so befindet sich die Prozessinstanz im Zustand „ERROR“. Dieser Zustand ist gegenüber der Definition in [Ahm03] neu. Er soll signalisieren, dass ein Fehler während der Ausführung der Prozessinstanz aufgetreten ist und die Prozessinstanz nicht erfolgreich beendet werden konnte.
- In allen anderen Fällen befindet sich die Prozessinstanz im Zustand „ACTIVE“.

Die eben beschrieben Logik wurde in der Methode `getState()` implementiert. Informationen über die Zustandswechsel eines Prozessinstanz-Threads werden im Abschnitt 6.5, der die Prozessmaschine behandelt, näher beschrieben. Die möglichen Zustände eine Prozessinstanz-Threads sind auch aus dem Zustandsdiagramm in Abbildung 3.4 ersichtlich.

## 6.2 Erweiterung der Schnittstelle zur Ausführung manueller Aktivitäten

Ein wichtiger Teil des Prozessunterstützungssystems ist eine Schnittstelle zur Ausführung manueller Aktivitäten (darunter fallen nach Abbildung 6.1 auch manuelle Entscheidungen) durch Benutzer des Portals, die sich aus zwei Teilen zusammensetzt. Der erste Teil ist eine Webseite, die im Web-Browser des Benutzers angezeigt wird und die Worklist des Benutzers anzeigt. Eine WorkList ist eine tabellarische Übersicht der Aktivitäten, die der Benutzer ausführen kann. Über einen Link in der WorkList kann das zur manuellen Aktivität gehörige Template angezeigt und ausgeführt werden. Die Worklist-Seite wird durch einen sichtbaren Handler und ein zugeordnetes Template realisiert. Im Rahmen dieser Arbeit wurde das Template erweitert. Im Abschnitt 6.3.1 werden die Änderungen an der WorkList-Seite erläutert.
Im Abschnitt 6.3.2 wird die Erweiterung des zweiten Teils der Schnittstelle betrachtet. Es handelt sich dabei um Systemkomponenten, die die zur Ausführung einer manuellen Aktivität notwendigen sichtbaren Handler aufrufen, wodurch dem Prozessteilnehmer eine Webseite im Web-Browser angezeigt wird, die den Beginn einer manuellen Aktivität repräsentiert.

### 6.3.1 Erweiterung der WorkList-Seite

In Abbildung 6.7 ist ein Screenshot der WorkList-Seite zu sehen. Die rot umrandeten Bereiche wurden im Rahmen dieser Arbeit hinzugefügt. Die WorkList ist in zwei Bereiche geteilt. Der erste Bereich ist die Tabelle „*Personal Task List*“. Hier werden alle manuellen Aktivitäten aufgelistet, die dem Benutzer zur persönlichen Ausführung zugeordnet sind. Den zweiten Bereich bildet die Tabelle „*Task Lists for the Group(s)*“. Hier werden alle manuellen Aktivitäten aufgelistet, die einer der Rollen, deren Mitglied der Benutzer ist, zur Ausführung zugeordnet sind. Die beiden Tabellen unterscheiden sich nur in der letzten Spalte. Als neue Spalte wurde in Rahmen dieser Arbeit die Spalte mit der Bezeichnung „*Process Instance*“ eingefügt. Sie zeigt den Namen der Prozessin-

stanz an, der die Aktivität zugeordnet ist. Diese Änderung war notwendig, damit identische Aktivitäten unterscheidbar sind. Sie können nun nach ihrer Zugehörigkeit zu einer Prozessinstanz unterschieden werden.
Eine weitere Änderung hat sich in der letzten Spalte ergeben. Hier wurden Auswahllisten hinzugefügt, die es dem Benutzer ermöglichen, persönliche Aktivitäten an andere Benutzer des Portals und rollenzugeordnete Aktivitäten an andere Rollen des Portals zu delegieren. Somit wird die im Abschnitt 6.1.3 entworfene Erweiterung realisiert. Durch die Änderung des Templates war auch die Änderung des sichtbaren Handlers, der die Platzhalter des Templates durch Inhalt ersetzt, notwendig. Die Klasse TasksListHandler implementiert den sichtbaren Handler. Die Funktionalität für die Weiterleitung wurde in der Klasse ForwardHandler implementiert.


Abb. 6.7 Screenshot der WorkList-Seite

## 6.3.2 Erweiterung der Schnittstelle zur Ausführung von manuellen Aktivitäten

Im Rahmen dieser Arbeit wurde auch die technische Schnittstelle zur Ausführung manueller Aktivitäten erweitert. Die Ausführung einer manuellen Aktivität führt dazu, dass dem ausführenden Prozessteilnehmer eine oder mehrere Seiten, die die Aufgabe der manuellen Aktivität beschreiben oder beinhalten, angezeigt werden. Für jede dieser Seiten existiert eine Template-Vorlage, die durch einen sichtbaren Handler mit Informationen gefüllt wird. Allen Handler-Objekten gemein ist, dass sie eine Methode handleRequest() definieren. Diese Methode wird durch das Session-Objekt, das dem aufrufenden Benutzer zugeordnet ist, aufgerufen. In dieser Methode werden durch das Handler-Objekt Templates bearbeitet oder andere Instruktionen ausgeführt.
Wenn ein Prozessteilnehmer eine manuelle Aktivität ausführen will, so klickt er in der WorkList-Ansicht (siehe Abbildung 6.7) auf den Link „*>Do Task*“. Dadurch wird das Session-Objekt veranlasst, die Methode handleRequest() des Handlers ExecuteManualActivityHandler auszuführen. Dieser nicht sichtbare Handler hat die Aufgabe, die Ausführung der manuellen Aktivität vorzubereiten. Dazu bestimmt das Handler-Objekt die Aktivität, welche ausgeführt werden soll. Die AssetId der zugehörigen ActivityInstance wird als Request-Parameter des Session-Objektes mit übergeben. Die Instanz liefert per Methodenaufruf den Handlernamen zurück, der der Ausführung der manuellen Aktivität entspricht. Anschließend wird die Session zur Ausführung der manuellen Aktivität vorbereitet. Dazu wird die AssetId des ActivityInstance Assets als Request-Parameter festgelegt und der Session der Prozesskontext des zugehörigen Prozesses zugewiesen. Im Anschluss wird die Session angewiesen, die Request-Bearbeitung an den Handler, der der manuellen Aktivität zugeordnet ist, weiterzu-

leiten. Dieser Handler übernimmt nun die weitere Bearbeitung. Idealerweise sollte dieser Handler oder ein Handler, an den der Request durch diesen Handler weitergeleitet wurde, dem Benutzer eine Webseite anzeigen, die der Benutzer dann bearbeiten kann.
Dieser Mechanismus, die Ausführung einer manuellen Aktivität einzuleiten, ist aus der Arbeit [Ahm03] übernommen worden. Es wurde allerdings eine Erweiterung vorgenommen, die die Nachbereitung einer manuellen Aktivitätsausführung betrifft. Die Nachbereitung war in [Ahm03] nicht vorgesehen. Dennoch macht eine Nachbereitung aus folgenden Gründen Sinn:

- Folgt auf eine manuelle Aktivität eine weitere manuelle Aktivität, so kann gleich zu dieser weitergeleitet werden.
- Der Handler, der einer manuellen Aktivität zugeordnet ist, muss nicht länger für den Abschluss einer manuellen Aktivität sorgen. Der Entwickler eines solchen Handlers muss dann keine Detailkenntnisse über die Arbeitsweise des Prozessunterstützungssystems besitzen. Vor allem muss er nicht berücksichtigen, ob der Handler direkt oder als Aktivität aufgerufen wurde.
- Bei manuellen Entscheidungen muss die durch den Benutzer gewählte Transition in der Instanz der Klasse ActivityInstance, die die Ausführung der Aktivität repräsentiert, gespeichert werden.

Es wurde im Rahmen dieser Arbeit die Klasse PostManualActivityExecutionHandler realisiert, die genau die eben angesprochenen Aufgaben erledigt. Zur Beendigung einer manuellen Aktivität muss eine der Seiten, die dem Benutzer während der manuellen Aktivitätsausführung angezeigt werden, einen Link beinhalten, der das Session-Objekt anweist, die handleRequest() Methode des PostManualActivityExecutionHandlers aufzurufen. Wird in der Session die AssetId des die Ausführung repräsentierenden ActivityInstance-Assets und der Request-Parameter „*State*“ mit dem Wert „DONE“ übergeben, so markiert die Methode die zugehörige Aktivität als ausgeführt. Folgt auf diese manuelle Aktivität eine weitere, so wird das Session-Objekt angewiesen, die Ausführung der nächsten manuellen Aktivität zu starten. Anderenfalls wird zur Anzeige der WorkList weitergeleitet.
Soll eine manuelle Entscheidung ausgeführt werden, so muss zusätzlich zum Request-Parameter „*State*“ noch die AssetId der gewählten Transition im Request-Parameter „*transitionId*“ übergeben werden. Fehlt diese, so kann die Entscheidung nicht abgeschlossen werden.
Wird im Request-Parameter „State“ ein anderer Wert als „DONE“ übergeben, so wird die Ausführung als gescheitert betrachtet und der Zustand entsprechend gesetzt.
Das Sequenzdiagramm in Abbildung 6.8 veranschaulicht noch einmal den Ablauf einer Aktivitätsausführung. Dabei startet der Prozessteilnehmer im mit „1.“ markierten Schritt durch Benutzung des Links „>Do Task“ die Ausführung und beendet sie im mit „2.“ markierten Schritt durch einen geeigneten Mechanismus, der vom Designer der Prozess-Seiten vorgesehen werden muss.

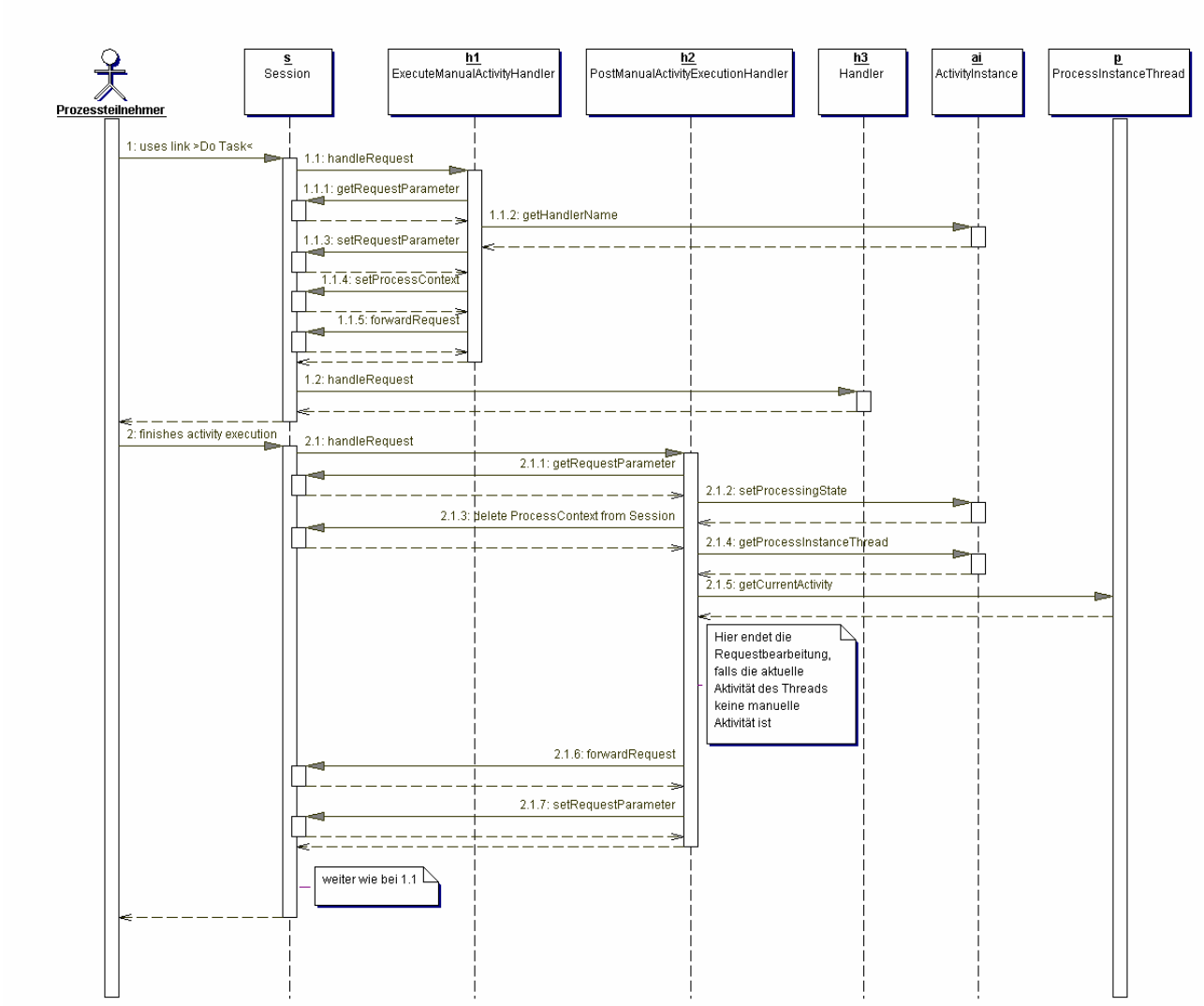


Abb. 6.8 Ausführung einer manuellen Aktivität

## 6.4 Entwurf und Implementierung des Ereignismodells

Aus den beiden Beispielprozessen in Kapitel 4 wurde die Anforderung identifiziert, dass Prozesse Ereignisse auslösen oder auf den Eintritt eines bestimmten Ereignisses warten können müssen. Im Abschnitt 6.1.2 wurde beschrieben, dass mithilfe von Ereignissen Subprozesse gestartet werden sollen. Da innerhalb der infoAssetBroker-Plattform kein hierfür geeignetes Ereignismodell existiert, wurde ein Modell entworfen. Abschnitt 6.4.1 beschreibt das entworfene Ereignismodell. In Abschnitt 6.4.2 wird auf Implementierungsdetails eingegangen und im Abschnitt 6.4.3 werden einige Ereignisklassen beschrieben, die mithilfe des Ereignismodells für das Prozessunterstützungssystem realisiert worden sind.

### 6.4.1 Beschreibung des Ereignismodells

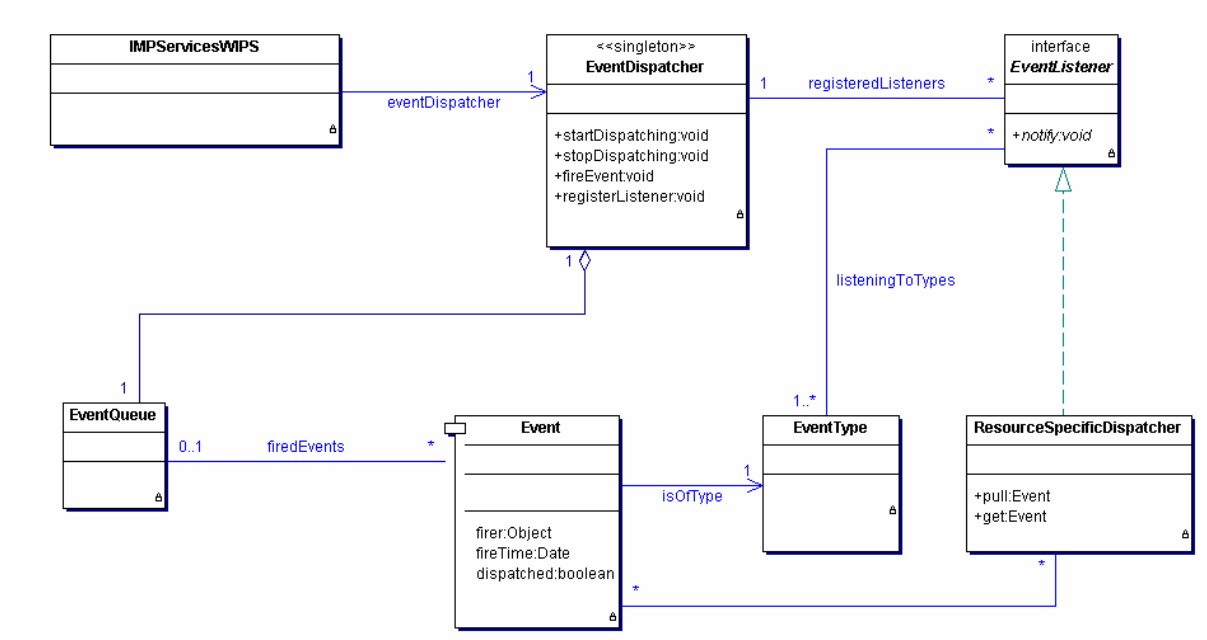


Abbildung 6.9 Klassendiagramm Ereignismodell

In Abbildung 6.9 wird das entworfene Ereignismodell gezeigt. Im Folgenden werden die gezeigten Klassen kurz beschrieben.

**EventDispatcher**
Ein Objekt vom Typ EventDispatcher hat die Aufgabe, ausgelöste Ereignisse in eine Warteschlange zu stellen und nach Eingang an registrierte *EventListener*-Objekte zu verteilen. Die Klasse EventDispatcher ist nach dem Singleton-Pattern [GHJV95] entworfen, d.h. es existiert maximal eine Instanz zur Laufzeit. Klassen, die das EventListener-Interface implementieren, können sich beim EventDispatcher für beliebig viele *EventTypes* registrieren und werden beim Auftritt eines Ereignisses dieser Typen benachrichtigt.

**EventListener**
Das Interface EventListener muss von einer Klasse implementiert werden, damit sich deren Instanzen beim EventDispatcher-Objekt für die Benachrichtigung bei neuen Ereignissen eines bestimmten Ereignistyps registrieren können.

**ResourceSpecificDispatcher**
Instanzen der Klasse RessourceSpecificDispatcher ermöglichen es, aufgetretene Ereignisse solange zu speichern, bis ein Empfängerobjekt diese Ereignisse abfragt. Die Klasse wurde entworfen, damit Ereignisse für deren Bearbeitung im Moment kein Empfängerobjekt verfügbar ist, später auf Anfrage eines Empfängers noch bearbeitet werden können.

**EventType**
Jedes Ereignis besitzt einen konkreten Ereignistyp. Eine Instanz der Klasse EventType repräsentiert einen Ereignistyp.
**Event**
Die Klasse Event modelliert ein generisches Ereignis. Jede Instanz stellt ein Ereignis dar. Die Klasse ist als abstrakte Klasse entworfen, so dass Ereignisklassen nur als Spezialisierung dieser Klasse realisiert werden können.

Sie ist nicht als Asset-Klasse entworfen, weshalb Ereignisse nicht persistent im Portalsystem gespeichert werden, sondern nur zur Laufzeit existieren. Ereignisse sind flüchtige Objekte. Zum Persistieren von Ereignissen, wie bei portalübergreifenden Ereignissen, muss ein spezieller Mechanismus realisiert werden. Dies geschieht jedoch nicht im Rahmen dieser Arbeit.

**EventQueue**
Die Klasse EventQueue implementiert eine Warteschlange für Event-Objekte. Sie werden beim Auslösen des Ereignisses durch den EventDispatcher in die Warteschlange eingefügt und werden in dieser Reihenfolge bearbeitet.

**IMPServices**
Diese Klasse existiert schon als Teil der infoAssetBroker-Plattform. Es existiert nur eine Instanz zur Laufzeit, die anderen Systemkomponenten die Dienste des Portals zur Verfügung stellt. Sie wird nun so erweitert, dass sie auch den Dienst des EventDispatchers zur Verfügung stellt.

### 6.4.2 Realisierung des Ereignismodells

In diesem Abschnitt sollen Implementierungsdetails der vier wichtigsten Klassen des Ereignismodells erläutert werden. Dabei handelt es sich um die Klassen EventDispatcher, EventType, Event und RessourceSpecificDispatcher.
Als erstes wird die Klasse Event beschrieben. Die Klasse ist, wie schon erwähnt, abstrakt. Konkrete Ereignis-Typen müssen deshalb als Spezialisierung dieser Klasse implementiert werden. Diese Klasse stellt einige grundlegende Eigenschaften der Ereignisse zur Verfügung. Die Tabelle 6.2 gibt eine Übersicht über die Attribute, die die Ereignisse realisieren.

| **Attribut** | **Beschreibung** |
|---|---|
| source | Objekt, das dieses Ereignis ausgelöst hat. |
| fireTime | Zeit, zu der das Ereignis ausgelöst wurde. |
| dispatched | Wahrheitswert, der angibt, ob die Instanz schon durch den EventDispatcher an registrierte EventListener-Objekte weitergeleitet wurde. |
| type | Ereignistyp, zu dem dieses Ereignis gehört. |

Tabelle 6.2 Attribute der Klasse Event

Als nächstes wird nun die Klasse EventType besprochen. Die Instanzen dieser Klasse repräsentieren unterschiedliche Ereignistypen. Die Klasse realisiert einen statischen Namensdienst, der es ermöglicht, zur Laufzeit über die Angabe der Bezeichnung eines Ereignistyps die zugehörige Instanz der Klasse zu ermitteln. Die Methoden des Namensdienstes sind als statische Methoden der Klasse realisiert. Ereignistypen werden extern in einer eigenen Datei mit Namen „*event.txt*“ definiert.
Die infoAssetBroker-Plattform wurde dazu so angepasst, dass sie während des Systemstarts diese Datei einliest und für jeden definierten Ereignistyp eine Instanz der Klasse EventType anlegt.
Die Syntax für die Definition eines Ereignistypen in der Datei lautet: <*Bezeichnung Ereignistyp*> = <*Ereignisklasse*>. Für <Bezeichnung Ereignistyp> muss die Bezeichnung eines Ereignistyps eingesetzt werden und für <Ereignisklasse> der qualifizierte Name der zugehörigen Ereignisklasse. In der Tabelle 6.3 werden die Attribute der Klasse EventType beschrieben, in der Tabelle 6.4 ihre wichtigsten Methoden.

| Attribut | Beschreibung |
|---|---|
| (static) createdEventTypes | Statisches Attribut. Referenz auf eine Instanz der Java-Standardklasse HashMap, die zur Verwaltung der angelegten Instanzen der Klasse EventType dient. |
| name | Bezeichnung des Ereignistyps |
| description | Beschreibung des Ereignistyps |
| eventClass | Ereignisklasse, die zu diesem Ereignistyp gehört. |

Tabelle 6.3 Attribute der Klasse EventType

| Methode | Beschreibung |
|---|---|
| createEvent() | Legt ein Ereignis an, in dem eine Instanz der zugehörigen Ereignisklasse erzeugt wird. |
| (static) newInstance(String name, Class eventClass) | Statische Methode. Legt für die als Parameter übergebene Bezeichnung *name* eine neue Instanz an und ordnet dieser die übergebene Ereignisklasse zu. Dabei wird überprüft, ob die übergebene Klasse eine Subklasse von Event ist und ob die Bezeichnung nicht schon verwendet wird. Bezeichnungen müssen eindeutig sein, d.h. es darf zu jeder Bezeichnung nur eine Instanz der Klasse EventType geben. Die erzeugte Instanz wird in der HashMap *createdEventTypes* gespeichert. |
| (static) getInstance(String name) | Statische Methode. Diese Methode liefert die Instanz zurück, die der als Parameter übergebenen Bezeichnung in der HashMap *createdEventTypes* zugeordnet ist, oder null, falls keine solche existiert. |
| (static) loadEventTypes(String path) | Statische Methode. Versucht aus der durch *path* definierten Datei Definitionen von Ereignistypen zu laden und für jeden gefundenen Ereignistyp eine Instanz anzulegen. (siehe auch Beschreibung oben) |

Tabelle 6.4 Wichtige Methoden der Klasse EventType

Nun wird die Klasse EventDispatcher beschrieben. Wie schon erwähnt, ist die Klasse als Singleton [GHJV95] realisiert, d.h. es existiert maximal eine Instanz dieser Klasse zur Laufzeit. Des Weiteren erbt die Klasse von der Java-Standardklasse Thread. Die Klasse realisiert einen eigenen Thread, dessen Aufgabe es ist, registrierte EventListener-Objekte über in die Warteschlange eingestellte Ereignisse zu benachrichtigen und diese dann aus der Warteschlange zu entfernen. Dieser Thread wird als Dispatching-Thread bezeichnet. Dabei ist die Warteschlange als FIFO-Puffer (FIFO: First In First Out) realisiert, d.h. die Ereignisse werden in der Reihenfolge ihres Einstellens in die Warteschlange durch den Dispatching-Thread bearbeitet. Die Tabelle 6.5 beschreibt die Attribute der Klasse EventDispatcher, während in Tabelle 6.6 die wichtigsten öffentlichen Methoden erläutert werden.

| Attribut | Beschreibung |
|---|---|
| stopRequested | Flag, das der Dispatching-Thread überprüft. Sollte es den Wert „true“ annehmen, so beendet sich der Dispatching-Thread. Dieses Flag wurde zum Beenden des Threads realisiert. |
| eventQueue | Die Warteschlange |
| registeredTypes | Eine Instanz der Java-Standardklasse HashMap, in der registrierte EventListener gespeichert sind. |
| lockedHashMap | Flag, das zum Sperren der HashMap *registeredTypes* dient. Alle Methoden dieser Klasse, die die HashMap bearbeiten oder auslesen, müssen diese vorher sperren, um gleichzeitige Änderungen an der HashMap zu vermeiden. Mithilfe dieses Flags wird ein einfaches Semaphor realisiert. |

Tabelle 6.5 Attribute der Klasse EventDispatcher

| Methode | Beschreibung |
|---|---|
| startDispatching() | Startet den Dispatching-Thread |
| stopDispatching() | Stoppt den Dispatching-Thread |
| fireEvent(Event event, Object source) | Diese Methode wird verwendet, um Ereignisse auszulösen. Das als Parameter übergebene Ereignis *event* wird in die Warteschlange eingefügt. Außerdem werden die Eigenschaften des Ereignisses *fireTime* und *source* gesetzt. Das als Parameter source übergebene Objekt wird dabei als Quelle des Ereignisses betrachtet. |
| registerListener(EventListener listener, EventType type) | Mit dieser Methode wird das als Parameter übergebene EventListener-Objekt *listener* beim EventDispatcher registriert und wird von diesem benachrichtigt, falls ein Ereignis vom Type *type* auftritt. |
| unregisterListener(EventListener listener, EventType type) | Diese Methode wird verwendet, um dem EventDispatcher mitzuteilen, dass das als Parameter übergebene EventListener-Objekt *listener* nicht länger bei Auftreten eines Ereignisses vom Typ *type* zu informieren ist. |

Tabelle 6.6 Wichtige öffentliche Methoden der Klasse EventDispatcher.

Als letzte Klasse soll nun die Klasse RessourceSpecificListener besprochen werden. Sie wurde implementiert, um Ereignisempfängern die verzögerte Bearbeitung eines Ereignisses zu ermöglichen. Dies war notwendig, nachdem bei der Realisierung des Konvertierungsprozesses in Kapitel 7 die Anforderung entstanden ist, dass ein Empfängerobjekt zwar Interesse an aufgetretenen Ereignissen hat, aber diese erst bearbeiten will, wenn die Kapazitäten dafür zur Verfügung stehen und nicht zum Benachrichtigungszeitpunkt durch den EventDispatcher. Für diesen Fall müssen die Ereignisse zwischengespeichert werden, denn wenn zum Bearbeitungszeitpunkt eines Ereignisses im Dispatching-Thread des Event-Dispatchers kein EventListener-Objekt für den zugehörigen Ereignistyp registriert ist, dann werden die Ereignisse normalerweise gelöscht.
Die Klasse implementiert das EventListener-Interface. Ihre Instanzen können so als EventListener-Objekt beim EventDispatcher registriert werden. Um die Ereignisse vorübergehend zu speichern, implementiert der RessourceSpecificDispatcher zu diesem Zweck einen eigenen FIFO-Puffer für jeden Ereignistyp. Allerdings erfolgt das Dispatching in diesem Falle nicht aktiv, sondern erst auf Wunsch eines anderen Objektes wird ein Ereignis aus

der Warteschlange bearbeitet. Die Tabelle 6.7 beschreibt die wichtigsten öffentlichen Methoden der Klasse ResourceSpecificDispatcher.

| Methode | Beschreibung |
|---|---|
| (static) getSpecificDispatcher(Class resourceClass) | Statische Methode. Diese Methode liefert eine Instanz zurück, die der als Parameter übergebenen Klasse *resourceClass* zugeordnet ist. So kann für jeden Klassentyp eine eigene Instanz angelegt werden. Die Instanzen werden intern in einer Liste verwaltet. |
| register(EventType et) | Registriert die Instanz, an der die Methode aufgerufen wird beim EventDispatcher für den Ereignistyp *et.* |
| unregister(EventType et) | Deregistriert die Instanz beim EventDispatcher für den Ereignistyp *et*. |
| get(EventTyp et) | Liefert eine Ereignis des Typs *et* aus der internen Warteschlange zurück, falls ein solches vorhanden ist, ansonsten null. |
| pull(EventTyp et) | Wie get, nur dass der Aufruf blockierend ist, d.h. die Methode kehrt erst zurück, wenn ein solches Ereignis in der Warteschlange gefunden wurde. |
| notifyListener(Event e) | Diese Methode stammt aus dem EventListener-Interface und wird durch den Dispatching-Thread des EventDispatchers aufgerufen, wenn die Instanz als EventListener-Objekt für den zum Ereignis *e* zugehörigen Ereignistyp registriert ist. Das Ereignis wird dann in die interne Warteschlange des ResourceSpecifcDispatchers eingefügt. |

Tabelle 6.7 Methoden der Klasse ResourceSpecificDispatcher

### 6.4.3 Realisierte Ereignisklassen

In diesem Abschnitt wird beschrieben, welche konkreten Ereignisklassen im Rahmen dieser Arbeit für das Prozessunterstützungssystem realisiert wurden und wo sie zum Einsatz kommen. Es handelt sich dabei um die vier Ereignisklassen ProcessStartEvent, ProcessStartedEvent, CouldNotStartProcessEvent und ProcessFinishedEvent, die alle von der abstrakten Klasse Event erben.

**ProcessStartEvent**
Die Instanzen dieser Klasse werden im Prozessunterstützungssystem verwendet, um einen Prozess zu starten. Die Bezeichnung des zugehörigen Ereignistyps lautet „*StartProcess*“. Die Prozessmaschine wird beim Auftritt eines Ereignisses vom Typ StartProcess benachrichtigt und startet daraufhin den gewünschten Prozess, falls die Startkriterien erfüllt sind (siehe auch Abschnitt 6.5). Dazu wurden der Klasse Attribute hinzugefügt, die Informationen darüber enthalten, welcher Prozess zu starten ist. Des Weiteren lässt sich mit dem Ereignis auch ein zusätzlicher Parameter übermitteln, der nach Prozessstart in den Prozesskontext geschrieben wird. Dadurch ist es möglich. dem Prozess zu Beginn Informationen zur Verfügung zu stellen. Die Tabelle 6.8 zeigt die Attribute der Klasse ProcessStartEvent.

| Attribut | Beschreibung |
|---|---|
| definition | Die Prozessdefinition, des Prozesses, der gestartet werden soll. |
| instance | Ist dieses Feld von null verschieden, so soll der Prozess als Subprozess der in diesem Feld angegebenen Prozessinstanz gestartet werden. |
| enactor | Portalbenutzer, der den Prozess gestartet hat. |
| additionalId | „*nonce*“ (number used once), die den Startvorgang eindeutig identifiziert. Die Prozessmaschine gibt diese Zahl mit der Antwort auf dieses Ereignis zurück. |
| variableName | Name, unter dem der optionale Parameter im Prozesskontext abgelegt werden soll. |
| variableValue | Wert des optionalen Parameters. Wird bei erfolgreichem Prozessstart im Prozesskontext unter dem Namen, der durch *variableValue* definiert wird, abgelegt. |

Tabelle 6.8 Attribute der Klasse ProcessStartEvent

**ProcessStartedEvent**

Die Prozessmaschine löst bei erfolgreichem Start eines Prozesses ein Ereignis dieses Typs aus. Dies kann als Antwort auf das vorher ausgelöste und durch die Prozessmaschine verarbeitete Ereignis vom Typ „*StartProcess*“ verstanden werden. Um eine Zuordnung zu einem Startvorgang zu ermöglichen wird die mit dem Startereignis übermittelte *additionalId* als Attribut einer Instanz der Klasse ProcessStartedEvent zurückgeliefert. Die Bezeichnung des zugehörigen Ereignistyps lautet: „*ProcessStarted*“. Die Tabelle 6.9 beschreibt alle Attribute der Klasse ProcessStartedEvent.

| Attribut | Method |
|---|---|
| instance | neu gestartete Prozessinstanz. |
| intiator | Objekt, das das vorausgehende Ereignis vom Typ „*StartProcess*“ ausgelöst hat. |
| additionalId | Der Wert dieses Attributs wird aus dem Startereignis übernommen. Somit kann ein Objekt eine Instanz dieser Klasse eindeutig einem von ihm ausgelösten Startereignis zuordnen. |

Tabelle 6.9 Attribute der Klasse ProcessStartedEvent

**CouldNotStartProcessEvent**

Die Prozessmaschine löst bei fehlgeschlagenem Start eines Prozesses ein Ereignis dieses Typs aus. Dies kann als Antwort auf das vorher ausgelöste und durch die Prozessmaschine verarbeitete Ereignis vom Typ „*StartProcess*“ verstanden werden. Um eine Zuordnung zu einem Startvorgang zu ermöglichen, wird die mit dem Startereignis übermittelte *additionalId* als Attribut einer Instanz der Klasse CouldNotStartProcessEvent zurückgeliefert. Ferner wird eine Begründung mit dem Ereignis übermittelt, warum der Prozess nicht gestartet werden konnte. Die Bezeichnung des zugehörigen Ereignistyps lautet: „*StartFailed*“. Die Tabelle 6.10 beschreibt alle Attribute der Klasse CouldNotStartProcessEvent.

| Attribut | Beschreibung |
|---|---|
| definition | Prozessdefinition, die gestartet werden sollte. |
| initiator | Referenz auf das Objekt, das das vorausgehende Ereignis vom Typ „*StartProcess*“ ausgelöst hat. |

| | |
|---|---|
| additionalId | Der Wert dieses Attributs wird aus dem Startereignis übernommen. Somit kann ein Objekt eine Instanz dieser Klasse eindeutig einem von ihm ausgelösten Startereignis zuordnen. |
| reason | Grund, warum der Prozess nicht gestartet werden konnte. |

Tabelle 6.10 Attribute der Klasse CouldNotStartProcessEvent

**ProcessFinishedEvent**
Die Prozessmaschine löst Ereignisse dieses Typs aus, wenn ein Prozess erfolgreich beendet wurde. Die Klasse enthält nur ein einziges Attribut mit Namen *instance,* das eine Referenz auf die Prozessinstanz enthält, die erfolgreich beendet wurde. Prozesse oder andere Systemkomponenten können so über das Ende einer Prozessausführung benachrichtigt werden. Die Bezeichnung des zugehörigen Ereignistyps lautet: „*ProcessFinished*".

## 6.5 Realisierung der Erweiterungen für die Prozessmaschine

In Abschnitt 6.1.6 wurde bereits erwähnt, dass eine Erweiterung der Prozessmaschine notwendig ist, um die Konzepte, um die das Prozessmodell erweitert wurde, interpretieren zu können. Diese Erweiterung wird im Abschnitt 6.5.1 beschrieben. Im Abschnitt 6.5.2 wird erläutert, wie die Prozessmaschine an den ereignisgesteuerten Startvorgang angepasst wurde.

### 6.5.1 Unterstützung der neuen Konzepte

Dieser Abschnitt behandelt die Erweiterungen für die Prozessmaschine, die aus den neu zum Prozessmodell hinzugefügten Konzepten resultiert. Bisher unterstützte die Prozessmaschine keine Nebenläufigkeiten sowie Entscheidungen. Hierzu wird zunächst der Aufbau der Prozessmaschine beschrieben. Die allgemeine Funktionsweise wurde schon im Abschnitt 3.2.3 erläutert.
Die Prozessmaschine ist nach dem Singleton-Pattern [GHJV95] realisiert und führt Prozesse in eigenen Threads aus. Diese Threads sind nicht identisch mit den Instanzen der Klasse ProcessInstanceThread. Die Klasse ProcessEngine, die die Prozessmaschine implementiert, besitzt eine interne Klasse ProcessEngineThread. Diese kapselt die ProcessInstanceThreads für die Prozessmaschine. Wenn ein Prozess in der Prozessmaschine ausgeführt wird, dann wird zu jeder Instanz der Klasse ProcessInstanceThread eine Instanz der Klasse ProcessEngineThread erzeugt. Der ProcessEngineThread ruft Methoden der Klasse ProcessEngine auf, um Aktivitäten auszuführen. Die Instanz der Klasse ProcessInstanceThread ist ein Asset und repräsentiert die persistenten Informationen der Ausführung einer Instanz der Klasse ProcessEngineThread in der Prozessmaschine, die auch bei einem Systemabsturz oder Runterfahren des Portals erhalten bleiben sollen. Dazu zählen Informationen wie Zustand des Threads und aktuell ausgeführte Aktivität. Eine genauere Beschreibung der Klasse ProcessInstanceThread findet sich in [Ahm03].
Eine Instanz der Klasse ProcessEngineThread, die einen eigenen Thread darstellt, erhält eine Referenz auf ihr persistentes Gegenstück, die Instanz von ProcessInstanceThread (im Folgenden als Prozessthread bezeichnet). Mit dieser Referenz wird die Methode executeProcessInstanceThreadInProcessEngineThread(..) der Prozessmaschine aufgerufen. Im Aufruf dieser Methode besteht die Hauptaufgabe des ProcessEngineThreads. Die Methode realisiert eine Schleife (im folgenden Hauptschleife genannt), die solange läuft, wie der Zustand des Prozessthreads „ACTIVE" ist. (Siehe Abbildung 3.4 für die möglichen Zustände eines Prozessthreads). Innerhalb dieser Schleife wird gerade bearbeitete Aktivität des Prozessthreads ermittelt, ihr Typ festgestellt und abhängig davon andere Methoden aufgerufen. Es existiert für jeden Aktivitätstyp eine eigene Methode, die Aktivitäten dieses Typs bearbeitet. Im Anschluss an die Aktivitätsbearbeitung wird die zu aktivierende Transition bestimmt, und daraus die nächste Aktivität ermittelt. Damit endet die Schleife. Die Ausführung von manuellen

Aktivitäten und automatischen Entscheidungen wurde schon in [Ahm03] besprochen und hat sich nur leicht verändert. Bei der Ausführung von manuellen Aktivitäten wird zunächst durch die Prozessmaschine geprüft, ob eventuelle eine dynamische Zuweisung an eine Rolle oder Person vorgenommen werden kann, bevor die manuelle Aktivität zur Ausführung der Standard-Rolle oder –Person, die durch die Prozessdefinition vorgesehen ist, zugewiesen wird. Das Konzept der dynamischen Zuweisung wurde weiter oben in diesem Kapitel bereits besprochen. Neu hinzugekommen sind Methoden, die Aktivitäten der Typen „*Endstate*“, „*ManualDecision*“, „*AutomatedDecision*“, „*Fork*“, „*Join*“, „*Merge*“, „*SendSignal*“ und „*ReceiveSignal*“ ausführen. Diese Methoden werden nun erläutert.

**executeThreadEndstate()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*Endstate*“ besitzt. Diese Methode setzt den Zustand zur Ausführung gehörenden ProzessThreads auf „FINISH“, was dazu führt, dass die Hauptschleife verlassen wird und der ProcessEngineThread seine Ausführung beendet. Wird die Hauptschleife auf diese Weise verlassen, so wird durch die Prozessmaschine ein Ereignis vom Typ „*ProcessFinished*“ ausgelöst.

**executeAsAutDecision()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*AutomatedDecision*“ besitzt. In dieser Methode wird eine automatische Entscheidung getroffen, welche der von einer Aktivität ausgehenden Transitionen als nächstes zu aktivieren ist. Dazu werden die Kontrollflussbedingungen der ausgehenden Transitionen ausgewertet. Die Transition, deren Bedingung erfüllt ist, wird aktiviert. Ist mehr als eine Bedingung erfüllt, so wird unter den in Frage kommenden Transitionen zufällig eine ausgewählt (Auswahl 1 aus n). Der Prozessmaschine wird dann als Rückgabewert mitgeteilt, welche Transition zu aktivieren ist.
Wird keine Transition gefunden, deren Kontrollflussbedingung erfüllt ist, so wird die Ausführung des ProcessEngineThreads abgebrochen. Damit wird nicht darauf gewartet, dass eine Kontrollflussbedingung erfüllt ist.

**executeAsManDecision()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*ManualDecision*“ besitzt. Dieser Methode gleicht in ihrer Funktion fast der, die manuelle Aktivitäten ausführt. Allerdings wird nach der Ausführung der Entscheidung durch einen Prozessteilnehmer der Prozessmaschine die gewählte Transition mitgeteilt. Die weitere Ausführung entspricht der Ausführung manueller Aktivitäten.

**executeAsMergeActivity()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*Merge*“ besitzt. Da das Konzept des „Merge“ eigentlich nur verwendet wird, um mehrere alternative Ausführungszweige eines Prozesses wieder zu vereinigen, wird lediglich der Zustand der zur Aktivitätsausführung gehörenden ActivityInstance auf „SUCESSFUL“ gesetzt und die nächste zu aktivierende Transition berechnet.

**executeAsForkActivity()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*Fork*“ besitzt. Eine Aktivität dieses Typs erzeugt nebenläufige Ausführungen von Aktivitäten. Dazu werden durch die Methode zu den N ausgehenden Transitionen der bearbeiteten Fork-Activity je N-1 ProcessEngineThreads und N-1 ProcessThreads erzeugt, die die Ausführung der auf die Transitionen folgenden Aktivitäten übernehmen. Dies führt zu einer parallelen Ausführung in der Prozessmaschine. Die letzte der N Transitionen wird durch den Thread selbst bearbeitet, der diese Methode aufgerufen hat.

**executeAsJoinActivity()**
Diese Methode wird durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*Join*“ besitzt. Eine Join-Aktivität wird eingesetzt, um parallele Ausführungen von ProcessEngineThreads zu synchronisieren. Im Prozessunterstützungssystem sind dazu zwei Synchronisationsarten vorgesehen: JOIN_OR oder JOIN_AND. Die Synchronisationsart wird über die Prozessdefinition für jede Join-Aktivität vorgegeben. Bei JOIN_OR wird, sobald eine Thread die Join-Aktivität erreicht, mit der Ausführung des Prozesses fortgefahren. Bei JOIN_AND wartet die Methode, bis alle Threads diese Join-Aktivität erreicht haben. Dazu wird die Zahl der zu synchronisierenden Threads mit der Zahl der tatsächlich synchronisierten Threads verglichen, bis diese übereinstimmen.

**executeAsSendSignalActivity()** und **executeAsReceiveSignalActivity()**
Diese Methoden werden durch die Prozessmaschine ausgeführt, wenn eine Aktivität den Typ „*SendSignal*“ bzw. „*ReceiveSignal*“ besitzt. Von diesen Methoden sind nur die Rümpfe vorhanden. Es existiert keine Implementation, da sich diese Aktivitäten im Prozesseditor noch nicht modellieren lassen (vgl. Kapitel 7). Eine konkrete Implementierung soll ermöglichen, dass eine Aktivität vom Typ „*SendSignal*“ ein durch die Prozessdefinition spezifiziertes Ereignis auslöst, bzw. dass eine Aktivität vom Typ „*ReceiveSignal*“ auf ein solches Ereignis wartet.

Mit den hinzugefügten Methoden werden alle dem Prozessmodell hinzugefügten Konzepte auch durch die Prozessmaschine unterstützt, mit der Ausnahme, das die Aktivitäten vom Typ *„SendSignal“* und *„ReceiveSignal“* noch nicht ausgeführt werden können. Dies ist aber auch nicht notwendig, da sie sich auch nicht mit dem Prozesseditor modellieren lassen (vgl. Kapitel 7).
Um den Entwicklern der Handler-Klassen, deren Instanzen Aktivitäten ausführen den Zugriff auf den zu einem ProcessEngineThread gehörenden Prozessthread zu ermöglichen, wurde eine Klasse ProcessInstanceThreadMapper implementiert, die ein Mapping zwischen ProcessEngineThreads und ProzessThreads realisiert. Dabei ist der ProcessEngineThread dafür verantwortlich, sich in die Mapping-Tabelle zu Beginn seiner Ausführung einzutragen und nach der Ausführung wieder auszutragen. Mithilfe dieser Klasse ist es dann möglich, zur Laufzeit aus dem Thread, der gerade die Methode handleRequest() eines Handlers ausführt, den zugeordneten Prozessthread und damit alle prozessrelevanten Informationen zu ermitteln.

### 6.5.2 Unterstützung des ereignisgesteuerten Startvorgangs

Im Abschnitt 6.4 wurden einige Ereignisklassen beschrieben, deren Instanzen Ereignisse darstellen, die im Prozessunterstützungssystem zum Starten von Prozessen verwendet werden. Die Prozessmaschine soll bei Auftritt eines Ereignisses vom Typ „ProcessStart“ einen neuen Prozess starten. Damit die Prozessmaschine beim Auftritt dieser Ereignisse vom EventDispatcher benachrichtigt wird, implementiert sie das EventListener-Interface. Die Methode notifyListener(Event e) wird vom EventDispatcher aufgerufen, um die Prozessmaschine zu benachrichtigen. Sie übergibt als Parameter ein Ereignis, das von einem der Typen ist, die bei Registrierung der Prozessmaschine als EventListener-Objekt beim EventDispatcher angegeben wurden. Die Prozessmaschine wird beim Systemstart als EventListener-Objekt für Ereignisse des Typs „ProcessStart“ registriert, so dass sie auch nur über den Eintritt dieser Ereignisse informiert wird.
In der Benachrichtigungsmethode wird ein eigener Thread gestartet, der in der dafür implementierten inneren Klasse ProcessEngineStartThread definiert ist. Dieser übernimmt den weiteren Startvorgang und entkoppelt den Dispatching-Thread, der die Benachrichtigungsmethode aufruft, von der eigentlichen Durchführung des Startvorganges.
In den Startvorgang wurde eine Überprüfung von Startkriterien eingefügt, die sich aus den Anforderungen im Kapitel 5.4 ergeben und in Kapitel 8 genau beschrieben werden. Sind nicht alle Kriterien erfüllt, so kann der Prozess nicht gestartet werden. Die Prozessmaschine löst daraufhin ein Ereignis vom Typ „*StartFailed*“ aus. Sind alle Kriterien erfüllt, wird eine Instanz vom Typ ProcessInstance angelegt. Wird durch das Startereignis

eine Referenz auf eine übergeordnete Prozessinstanz übermittelt, so wird die anzulegende Prozessinstanz als Subprozess gestartet. Anschließend wird noch der Startparameter, der in einem Startereignis definiert werden kann, im Prozesskontext der neu angelegten Prozessinstanz gespeichert. Danach wird die Prozessausführung gestartet, indem eine Startaktivität ermittelt wird und eine Instanz der Klasse ProcessEngineThread erzeugt wird. Damit ist der Startvorgang erfolgreich abgeschlossen. Es wird abschließend ein Ereignis vom Typ „*ProcessStarted*“ erzeugt. Die Ausführung der Instanz der Klasse ProcessEngineThread endet an diesem Punkt.

# Kapitel 7

# Prozessdefinition

Dieses Kapitel behandelt die Auswahl eines Werkzeugs zur Definition von Prozessen und die Integration in die infoAssetBroker-Plattform. Die mit diesem Werkzeug definierten Prozesse sollen durch das Prozessunterstützungssystem ausgeführt werden, weshalb ein geeigneter Konvertierungsmechanismus gefunden werden muss, um aus den vorliegenden Prozessdefinitionen entsprechende Instanzen der Klassen des Prozessmodells zu erzeugen. In der Arbeit [Ahm03] wurde bereits ein einfacher Prozesseditor zur Definition von Prozessen entworfen. Im Abschnitt 7.1 wird erläutert, welche Defizite dieser Prozesseditor hat und wie diese Defizite im Rahmen dieser Arbeit beseitigt werden können. Der Abschnitt 7.2 beschäftigt sich mit der Untersuchung verschiedener UML-Modellierungswerkzeuge zur Eignung als Prozesseditor. Im Abschnitt 7.3 wird dann schließlich eines dieser Werkzeuge als Prozesseditor ausgewählt und in die Portalplattform integriert. Abschnitt 7.4 behandelt die Anpassung der UML-Aktivitätsdiagramme. Im Abschnitt 7.5 wird die Realisierung des schon oben angesprochenen Konvertierungsmechanismus erläutert.

## 7.1 Defizite des existierenden Prozesseditors

Im Abschnitt 5.3 wurden die Anforderungen an einen Prozesseditor zusammengetragen. Der existierende Prozesseditor wurde in Abschnitt 3.2.4 vorgestellt. In diesem Abschnitt werden die Defizite des existierenden Prozesseditors vorgestellt, die sich ergeben, weil einige der Anforderungen aus Kapitel 5.3 nicht erfüllt werden. Die Tabelle 7.1 zeigt alle Anforderungen, die der existierende Prozesseditor nicht erfüllt.
Mit dem existierenden Prozesseditor lassen sich Diagramme erstellen, die UML-Aktivitätsdiagammen [JBR99] nachempfunden sind. Neben den in Tabelle 7.1 präsentierten Defiziten gibt es noch einen weiteren Nachteil, der gegen die Verwendung des existierenden Prozesseditors spricht: Der existierende Prozesseditor wurde als Prototyp entworfen und besitzt daher eine unausgereifte und ungetestete Benutzerschnittstelle. Funktionalitäten wie Drucken eines Diagramms oder Exportieren des Diagramms in ein Grafikformat wie beispielsweise das JPEG-Format sind nicht vorhanden oder unausgereift. Ebenso unausgereift ist die Menuführung des Prozesseditors.

| Defizite des existierenden Prozesseditors |
|---|
| |
| Nebenläufige Prozessschritte sind nicht modellierbar. |
| Modellierung von Ereignissen ist nicht möglich. Hierzu gehören Auslösen von Ereignissen und Warten auf ein bestimmtes Ereignis. |
| Benennung von Rollen für die Ausführung von Aktivitäten ist nicht möglich. |
| Modellierung von Entscheidungen wird nicht unterstützt. |
| Definition von Kontrollflussbedingungen für Transitionen wird nicht unterstützt. |
| Definition von Meta-Informationen wird nicht unterstützt. (z.B. Beschreibung von Aktivitäten) |

Tabelle 7.1 Defizite des existierenden Prozesseditors

Die hier beschriebenen Defizite sollen im Rahmen dieser Arbeit beseitigt werden, so dass ein Prozesseditor zur Definition von Prozessen verwendet werden kann, der alle Anforderungen aus dem Abschnitt 5.3 erfüllt.

## 7.2 Untersuchung von UML-Modellierungswerkzeugen

Grundsätzlich kommen drei Alternativen zum Beseitigen der im Abschnitt 7.1 genannten Defizite des existierenden Prozesseditors in Betracht:

- Erweiterung des existierenden Prozesseditors,
- Neuentwicklung,
- Verwendung eines Fremdwerkzeugs zur Definition von Prozessen.

In diesem Abschnitt sollen dazu mehrere Fremdwerkzeuge daraufhin untersucht werden, inwieweit sie als Prozesseditor verwendet werden können. Es handelt sich dabei um verschiedene UML-Modellierungswerkzeuge (UML: Unified Modeling Language). Mit ihnen lassen sich unter anderem UML-Diagramme [JBR99] verschiedener Typen erstellen, darunter auch UML-Aktivitätsdiagramme.

### 7.2.1 UML-Aktivitätsdiagramme und UML-Profile

UML-Aktivitätsdiagramme [JBR99] werden allgemein zur Definition von Prozessen verwendet, weshalb dieser Aktivitätstyp auch in dieser Arbeit zur Definition von Prozessen verwendet werden soll. Da UML-Modellierungswerkzeuge sehr verbreitet sind und häufig bei der Softwareentwicklung zum Einsatz kommen, ist davon auszugehen, dass die verwendeten Diagrammtypen und die Werkzeuge selber gut verstanden und ausgereift sind.
Aktivitätsdiagramme bieten viele Modellierungselemente, mit denen sich die benötigten Konzepte modellieren lassen: Es lassen sich Anfangszustände, Endzustände, Aktivitäten und Transitionen modellieren. Zur Definition von Nebenläufigkeiten können Fork- und Join-Elemente verwendet werden. Entscheidungen lassen sich über Choice-Elemente modellieren, Ereignisse über Signals und Verantwortungsbereiche über Swimlanes. Alle verwendeten Modellierungselemente von Aktivitätsdiagrammen werden im Abschnitt 7.4 genauer beschrieben.
Zur Anpassung der UML-Aktivitätsdiagramme an die Anforderungen der Definition von Prozessen für das in dieser Arbeit entwickelte Prozessunterstützungssystem, sollen UML-Profile verwendet werden. Eine Anpassung ist notwendig, da zur Definition von Prozessen für das hier entwickelte Prozessunterstützungssystem spezielle Angaben notwendig sind, die UML-Aktivitätsdiagramme standardmäßig nicht unterstützen. Dazu zählen die Angabe eines Handlernamens zu jeder Aktivität, sowie die Unterscheidung zwischen automatischen und manuellen Entscheidungen. Alle Anpassungen werden im Abschnitt 7.3 beschrieben.

UML-Profile bieten eine Möglichkeit das UML-Metamodell an konkrete Anwendungsdomänen anzupassen, ohne das Modell selber zu verändern. Das UML-Metamodell wird durch die *Object Management Group* [OMG04] definiert und ist Teil des UML-Standards. Es bildet die Grundlage für die Modellierung von UML-Diagrammen. Eine konkrete Instanz dieses Modells wird auch als UML-Modell bezeichnet. UML-Profile sind seit der Version UML 1.4 Teil des UML-Standards. Zur Erweiterung des UML-Metamodells werden dazu die folgenden UML-Modellierungselemente verwendet: Stereotypes, Eigenschaftswerte (auch TaggedValues genannt) und in der Object Constraint Language (OCL) formulierte Einschränkungen. Die folgende Definition für Stereotyp stammt aus [Ess01]:

> „Ein Stereotyp ist ein in «..» eingerahmter Begriff, der nicht direkt zum Sprachumfang der UML gehört, aber für die Anwendung, das Modell oder die Programmiersprache notwendig ist."
> „Durch Stereotype wird die UML anpassbar. Stereotype erweitern oder variieren vorhandene Modell-Elemente."

TaggedValues sind Name/Wert-Paare, die an jedes UML-Element angehängt werden können, um zusätzliche Informationen zu modellieren. Die Object Constraint Language (OCL) wird zur Einschränkung der Regeln des UML-Metamodells verwendet. Sie bietet Sprachkonstrukte, die die Menge der zulässigen Instanzen eines UML-Modells einschränken.
Als UML-Profil wird eine konkrete Sammlung dieser Anpassungskonstrukte bezeichnet. Mit dem kurz vor der Verabschiedung stehenden Standard UML 2.0 werden Profile in das UML-Metamodell als eigene Elemente aufgenommen, wodurch Beziehungen (z.B. Vererbung) zwischen den Profilen oder die Wiederverwendung von Profilen möglich sind. Nähere Informationen zu UML, OCL, UML-Diagrammen und UML-Profilen findet man in [OMG04] und [Jck04], sowie in [JBR99].

### 7.2.2 Untersuchungskriterien für UML-Modellierungswerkzeuge

Im Folgenden werden nun die Kriterien vorgestellt, nach denen die UML-Modellierungswerkzeuge in dieser Arbeit untersucht und bewertet wurden. Sie ergeben sich aus den Anforderungen an einen Prozesseditor, aber auch aus allgemeinen Anforderungen an ein Softwarewerkzeug mit Benutzerschnittstelle. Die Untersuchung wurde im Rahmen dieser Arbeit durchgeführt. Da die Untersuchung aber nicht der Hauptschwerpunkt dieser Arbeit war, wurde auf eine ausführliche Bewertung der Werkzeuge nach vorher definierten Kriterien verzichtet. Es sollen vielmehr die Möglichkeiten der Werkzeuge und der erste Eindruck, der bei der Benutzung der Werkzeuge entsteht, untersucht werden. Die folgende Auflistung zeigt die untersuchten Eigenschaften der Werkzeuge und beschreibt sie kurz.

- Allgemein: In diesem Abschnitt soll das zu untersuchenden Werkzeuge kurz beschrieben werden. Des Weiteren soll hier präsentiert werden, wie der Hersteller selbst sein Werkzeug beschreibt.
- Bedienung: Der Bereich Bedienung gliedert sich in dieser Arbeit in drei Unterbereiche :
    - Benutzerfreundlichkeit
    - Installation
    - Umfang an UML-Modellierungsmöglichkeiten

  Im Bereich *Benutzerfreundlichkeit* soll untersucht werden, ob das Modellierungswerkzeug leicht zu bedienen ist, ob die Bedienung intuitiv ist und ob die Menüführung übersichtlich und einfach zu verstehen ist. Der Bereich soll eine allgemeine Einschätzung der Handhabung des Systems durch einen Benutzer liefern. Dabei wird dabei davon ausgegangen, dass es sich um einen im Umgang mit UML-Modellierungswerkzeugen erfahrenen Benutzer handelt, denn ein unerfahrener Benutzer wird sich in

der Eingewöhnungsphase sowieso erst mit der Handhabung eines Werkzeugs und der eingesetzten Modellierungssprache UML vertraut machen müssen. Im Rahmen der Untersuchung dieses Bereiches wurde das gleiche UML-Aktivitätsdiagramm mit allen untersuchten Softwarewerkzeugen einmal erstellt. Dieses „Test-Diagramm“ ist in Abbildung 7.1 dargestellt.

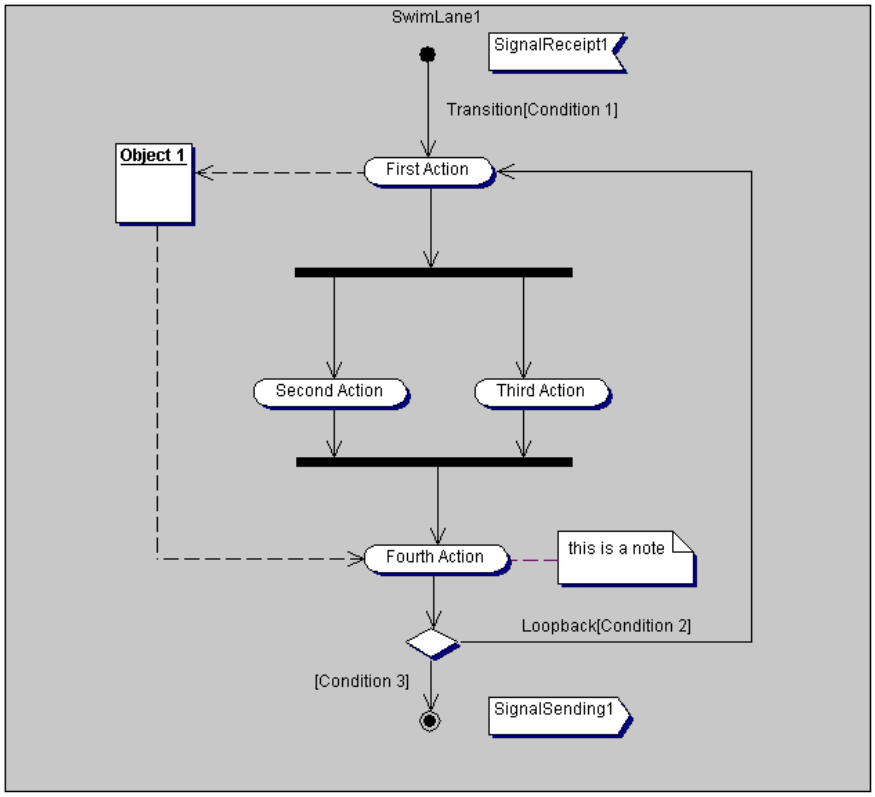


Abbildung 7.1 „Test-Diagramm“
(mit Together Control Center 6.0 erstellt und als GIF-Grafik exportiert)

Im Bereich *Installation* soll nur untersucht werden, ob sich die untersuchte Software ohne Probleme und spezielle Voraussetzungen installieren lässt und ob der Installationsprozess einfach gestaltet ist. Dazu wurden alle untersuchten Werkzeuge einmal installiert.
Unter „*Umfang an UML-Modellierungsmöglichkeiten*“ soll untersucht werden, welche UML-Diagramme mit dem untersuchten Werkzeug erstellt werden können und wie weit UML-Profile unterstützt werden.

- Technische Kriterien: In diesem Bereich sollen die technischen Kriterien untersucht werden. Dazu sollen in einzelnen die folgenden Bereiche untersucht werden :
    - Systemanforderungen des zu untersuchenden Werkzeugs
    - Erweiterbarkeit: Unter diesen Bereich fällt die Klärung der Frage, ob das zu untersuchende Werkzeug erweiterbar ist und wenn ja mit welcher Technik es erweitert werden kann. Außerdem soll eine Abschätzung des Komplexitätsgrades des Erweiterungsmechanismus gegeben werden. Erweiterungen sind deshalb interessant, da das zu untersuchende Werkzeug Prozesseditor unter Umständen auch zur Prozessverfolgung eingesetzt werden kann, wenn es so erweitert werden kann, dass es durch die Prozessmaschine fernsteuerbar ist.
    - Installationsumfang (Speicherplatzbedarf einer Standardinstallation)
    - Java-Web Start: In diesem Bereich wird geklärt, ob das zu untersuchende Werkzeug mit der Java Web Start-Technologie (siehe Kapitel 7.3) beim Benutzer gestartet werden kann. Wenn

dies möglich ist, ist eine einfache Integration in die infoAssetBroker-Plattform möglich, da diese schon die notwendigen technischen Voraussetzungen besitzt, um integrierte Anwendungen per Java Web Start zu starten.

- Export/Import der UML-Diagramme: In diesen Bereich fällt die Untersuchung der Möglichkeiten, die mit dem Werkzeug erstellten Diagramme zu exportieren, bzw. mit anderen Werkzeugen erstellte Programme zu importieren. Dabei soll geklärt werden, welche Grafikformate unterstützt werden, und ob andere Darstellungen der Diagramme (z.B. in XML-Formaten) unterstützt werden.
- Lizenzen: Unter diesen Bereich fällt die Untersuchung der zur Verfügung stehenden Lizenzen und Preise für das untersuchte Werkzeug, soweit diese online zu ermitteln waren. Eine Rückfrage beim Hersteller oder einem Handler ist dazu nicht erfolgt, da dieser Bereich eigentlich nur informativ und nicht als hartes Kriterium dieser Arbeit untersucht wurde, um dem Nutzer eine gewisse Preisvorstellung zu geben.

In den folgenden Abschnitten werden nun die untersuchten Werkzeuge einzeln beschrieben. Dabei sind die Ergebnisse nach den eben präsentierten Untersuchungsbereichen gegliedert.

### 7.2.3 Together Control Center 6.0.1

Allgemein: Together Control Center 6.0.1 ist viel mehr als nur ein reines UML-Tool zum Zeichnen von UML-Diagrammen. Es ist eine so genannte Model-Build-Deploy Plattform, d. h., dass das Werkzeug den Benutzer in den verschiedenen Phasen der Softwareentwicklung unterstützt. In der Modellierungs- und Entwurfsphase wird der Benutzer beim Zeichnen von UML-Diagrammen unterstützt. Die erstellten Modelle werden in der Implementationsphase durch das Tool in Java-Codegerüste umgesetzt, welche durch den Programmierer zu vervollständigen sind. Ferner unterstützt Together Control Center den Benutzer beim Erstellen einer grafischen Benutzerschnittstelle, sowie der Dokumentation des entworfenen Codes. Als Laufzeitumgebung hilft es beim Starten und Warten von EJB-Anwendungen, sowie Webservices (genaueres unter [Bor04]).

Bedienung :

- Benutzerfreundlichkeit: Die Bedienung ist leicht und intuitiv. Einziger Nachteil bei Together ist der deutliche Bedarf an Arbeitsspeicher und Rechenleistung, der sich bei langsameren Rechnern während der Arbeit mit dem Together Control Center bemerkbar macht, da häufiger einen Augenblick gewartet werden muss. Ein Pluspunkt von Together ist, dass sich verschiedene Benutzerrollen einstellen lassen, die sich durch die Art der Fensteraufteilung und den Umfang der Menüs unterscheiden (Beispiel: Business Modeller, Designer, Programmer, Developer) .
- Installation: Die Installation erfolgt ohne Probleme. Allerdings lässt sich das Programm ohne gültiges Lizenzfile nicht starten. Das Lizenzfile wird nur separat durch den Borland-Vertrieb ausgeliefert.
- Umfang an UML-Modellierungsmöglichkeiten: Es wird die vollständige Palette an UML-Modellierungsmöglichkeiten unterstützt, die UML 1.4 vorsieht:
  - Klassendiagramme
  - Sequenzdiagramme
  - Anwendungsfalldiagramme
  - Zustandsdiagramme
  - Kollaborationsdiagramm
  - Komponentendiagramm
  - Aktivitätsdiagramme
    - Aktivitäten

        - Transitionen: Es werden sowohl Objektfluss- als auch Kontrollfluss-Transitionen unterstützt.
        - Objekte
        - Zustände (auch History-Zustände)
        - Start/End/Synchronisationszustände
        - Branch
        - Fork/Join
        - Kommentare
        - ***Bedingungen lassen sich nur an Transitionen über die Guard-Eigenschaft angeben.***
        - Signale
        - Swimlanes
    - UML-Profile: Alle oben angesprochenen Anpassungskonstrukte sind vorhanden, jedoch wird das Konzept des UML-Profils selbst nicht unterstützt.

Technische Kriterien :

- Technische Vorsaussetzungen
    - 384 MB RAM ( empfohlen werden **1 GB** !!!)
    - 340 MB Festplattenspeicher
    - Sun Java SDK 1.3.1_06 oder besser
    - Intel Pentium III / 500 MHZ oder schneller
    - Windows 2000/XP Professional, oder NT (Service Pack 6a) oder Linux (RedHat 7.3 oder 8.0 / SUSE 8.0 oder 8.1)
    - oder alternativ auch Solaris bzw. Mac
- Erweiterbarkeit: Das Werkzeug ist erweiterbar. Es sind grundsätzlich zwei Arten der Erweiterungen möglich. Änderungen an Config-Files: Sie beeinflussen das Aussehen der Diagramme und Eigenschaftsdialoge sowie die zugrunde liegenden Meta-Modelle. Die Config-Files können über einen einfachen Text-Editor geändert werden. Um Funktionalität zu Together hinzuzufügen, müssen neue Module geschrieben werden. Dies geschieht in der Programmiersprache Java durch einbinden der OpenAPI von Together, mit deren Hilfe auf zentrale Funktionen von Together Control Center zugegriffen werden kann. Dadurch ist das Werkzeug auch fernsteuerbar. Es besitzt eine Schnittstelle zum Lesen/Schreiben der zentralen Laufzeitobjekte dieses Werkzeugs. Die Komplexität der Erweiterung wird durch den Autor als hoch eingeschätzt. Obwohl die Dokumentation auf der Webseite von Borland gut ist, gibt es doch viele verschiedene Möglichkeiten dieses Werkzeug zu erweitern, deren Fähigkeiten vor einer Erweiterung zunächst alle evaluiert werden müssen, um die richtige Methode auszuwählen(Interfaces, OPENAPI, config-Files).
- Installationsumfang: Eine Standardinstallation verbraucht 128 MB Festplattenspeicher.
- Java Web Start: Die Java Web Start-Technologie wird nicht unterstützt.
- Export/Import der UML-Diagramme: Die Diagramme lassen sich einzeln oder als komplettes Modell in XMI 1.1 nach Oma-Standard (ein spezielles XML-Format, siehe Abschnitt 7.3) exportieren und importieren. Zum Export in ein Grafikformat werden GIF, SVG, WMF und Bitmap unterstützt.
- Lizenzen: Ein Lizenzfile muss vorhanden sein, sonst läuft die Anwendung nicht. Über Lizenzpreise war auf der Webseite des Herstellers nichts zu erfahren.

### 7.2.4 Rational Rose Enterprise 2003

Allgemein: Rose Enterprise 2003 ist ähnlich wie Together Control Center eine komplette Entwicklungsplattform, die allen am Software-Entwicklungsprozess beteiligten Personen eines Unternehmens eine einheitliche Grundlage zur visuellen Modellierung bieten soll. Die Möglichkeit, UML-Diagramme zu zeichnen, ist nur eine der vielen Möglichkeiten dieser Software. Sie bietet umfangreiche Funktionen zur Entwicklung von Java, C++, C# und anderen Anwendungen. Dabei steht die Unterstützung von mehreren Benutzern, die gleichzeitig an einem Projekt arbeiten genauso im Vordergrund, wie die Präsentation der modellierten Daten (vgl. [Ibm04]).

Bedienung :

- Benutzerfreundlichkeit: Die Verwendung von Rational Rose ist ein wenig gewöhnungsbedürftig und nicht so intuitiv wie bei den anderen untersuchten Modellierungswerkzeugen. Das Erstellen eines UML Diagramms ist komplizierter als bei den anderen Werkzeugen, was aber daran liegt, dass Rational Rose die Diagramme bestimmten Views zuordnet. Ein View stellt bei RationalRose eine bestimmte Sicht auf ein Softwareprojekt dar. Die Sichten unterscheiden sich durch die Aspekte, die sie betonen sollen (z.B. LogicView für Programmlogik, der UseCaseView für Anwendungsfälle). Diese Information ist aber für den in dieser Arbeit geforderten Anwendungsfall, das Erstellen von Aktivitätsdiagrammen zum Modellieren von Geschäftsprozessen, unwichtig. Besser ist allerdings die Angabe von Eigenschaften zu Diagrammelementen gelungen. Sie erfolgt, für Windows-Anwendungen typisch, über Rechte-Maustaste und Dialog mit Registerkarten.
- Installation: Die Installation lief auf einem Windows 2000 Rechner nicht ohne Fehlermeldung ab. Der erforderliche Neustart lieferte ebenfalls eine Fehlermeldung. Die Software lässt sich nicht ohne den Einsatz des mitgelieferten „License Server“ verwenden, der dann allerdings ständig im Hintergrund läuft. Zum Starten der Software ist eine Lizenz nötig, die durch den „License Server“ verwaltet wird.
- Umfang an UML-Modellierungsmöglichkeiten: Es wird die volle Palette an UML-Modellierungsmöglichkeiten unterstützt, die UML 1.4 vorsieht:
    - Klassendiagramme
    - Sequenzdiagramme
    - Anwendungsfalldiagramme
    - Zustandsdiagramme
    - Kollaborationsdiagramm
    - Komponentendiagramm
    - Aktivitätsdiagramme – mögliche Modellierungselemente:
        - Aktivitäten
        - Transitionen (nur Kontrollfluss-Transitionen)
        - Keine Objekte
        - Zustände
        - Start/End/Synchronisationszustände
        - Branch
        - Fork/Join
        - Kommentare
        - Bedingungen lassen sich nur an Transitionen über die Guard-Eigenschaft angeben.
        - ***Keine Signale !!***
        - Swimlanes
    - UML-Profile: Es sind alle oben beschriebenen Anpassungskonstrukte vorhanden. UML Profile lassen sich auch gezielt als Aggregation dieser Anpassungskonstrukte definieren, soweit es der UML1.4-Standard vorsieht.

Technische Kriterien :

- Technische Vorsaussetzungen:
  - Windows NT4.0 (SP6a) , Windows 200 (SP2 od.3) , Windows XP
  - 600 MHz Intel Pentium III
  - 512 MB Ram
  - 400 MB Festplattenspeicher
- Erweiterbarkeit: Das Werkzeug ist erweiterbar. Die Erweiterung wird über einen Plug-In-Mechanismus vorgenommen. Allerdings konnte der Mechanismus und die Programmierung eines Plug-Ins nicht untersucht werden, da diese Informationen geschützt und nur für Rational Kunden zugänglich sind.
- Installationsumfang: Eine Standardinstallation verbraucht 335 MB Plattenspeicher.
- Java Web Start: Die Java Web Start-Technologie wird nicht unterstützt.
- Export/Import der UML-Diagramme: Diagramme lassen sich durch das Werkzeug nur in einem speziellen Rose-Petal-Format exportieren. Andere Formate können über Plug-Ins unterstützt werden, die jedoch standardmäßig nicht mit ausgeliefert werden.
- Lizenzen: Getestet wurde mit einer 15-tägigen Testversion. Über Lizenzpreise war auf der Webseite des Herstellers nichts zu erfahren.

### 7.2.5 Poseidon Community Edition 2.0

Allgemein: Poseidon ist ein reines UML-Modellierungstool, was bedeutet, dass es einzig zu dem Zweck eingesetzt wird, UML-Diagramme zu zeichnen. Es ist aus dem OpenSource-Projekt ArgoUML hervorgegangen. (siehe [Tig04]) und erfreut sich zunehmend wachsender Beliebtheit. Es wird in verschiedenen Versionen ausgeliefert, wobei die Community Edition kostenlos erhältlich ist. Die anderen Versionen bieten zusätzliche Möglichkeiten, wie z.B. automatische Codeerzeugung (vgl. [Gen04]).

Bedienung :

- Benutzerfreundlichkeit: Die Menüführung ist übersichtlich und einfach zu bedienen. Die gewünschten Menüpunkte zur Bearbeitung der UML-Diagramme sind einfach zu finden und die wichtigsten Funktionen sind in der Toolbar zusammengefasst. Die einzelnen Übersichtsfenster lassen sich alle individuell anpassen. Bei größeren Diagrammen lässt sich der gewünschte Diagramm-Ausschnitt mithilfe eines stufenlosen Zoom-Interfaces bestimmen. Allerdings gibt es für die Community Edition (siehe Lizenzen) kein eingebautes Hilfesystem. Der Benutzer wird auf eine Webseite bei Gentleware weitergeleitet.
- Installation: Die Installation verlief ohne Probleme. Sowohl als Java Web-Start-Version, wie auch als Installationsversion unter Windows XP waren keine Probleme zu beobachten.
- Umfang an UML-Modellierungsmöglichkeiten: Der Hersteller gibt an, dass sein Werkzeug schon heute den kommenden Standard UML 2.0 unterstützt und damit auch alle dort vorkommenden Modellierungselemente. Wie aber unten zu sehen ist, ist dies nicht der Fall.
  - Klassendiagramme
  - Sequenzdiagramme
  - Anwendungsfalldiagramme
  - Zustandsdiagramme
  - Kollaborationsdiagramm
  - Komponentendiagramm
  - Aktivitätsdiagramme
    - Aktivitäten

        - Transitionen: Es werden sowohl Objektfluss- als auch Kontrollfluss-Transitionen unterstützt.
        - Objektflusszustände
        - Start/End/Synchronisationszustände
        - Branch
        - Fork/Join
        - Kommentare
        - Benutzerdefinierte Figuren (Ellipsen, Rechtecke, Polygone..)
        - ***Bedingungen lassen sich nur an Transitionen über die Guard-Eigenschaft angeben.***
        - ***Signale sind nicht definierbar. ( sind aber in UML 2.0 enthalten )***
        - ***Ebenso werden keine Swimlanes unterstützt. ( sind ebenfalls in UML 2.0 enthalten )***

    - UML-Profile: Alle oben angesprochenen Anpassungskonstrukte sind vorhanden, jedoch wird das Konzept des UML-Profils als solches nicht unterstützt. Mit der Version 2.2 lassen sich einem Stereotyp ein oder mehrere Eigenschaftswerte zuordnen, die dann jedes Element übernimmt, das mit dem Stereotyp markiert wird.

Technische Kriterien :

- Technische Vorsaussetzungen:
    - Java JDK 1.4
    - Microsoft Windows Betriebssystem
- Erweiterbarkeit: Das Werkzeug ist erweiterbar durch einen Plug-In-Mechanismus. Dazu kann in Java unter Verwendung der durch Gentleware zur Verfügung gestellten PoseidonAPI ein eigenes Plug-In implementiert werden. Da Poseidon ein relativ neues Produkt ist, findet man erst wenige Plug-Ins. Ein Beispiel-Plug-In, welches allerdings nur ab der Standardversion aufwärts läuft, ist aber auf der Website des Herstellers verfügbar. Zum Kompilieren werden Java und Ant 1.5 benötigt. Dokumentation zum Erweitern und Schreiben von eigenen Plug-Ins fehlte zum Zeitpunkt der Untersuchung noch, ist aber inzwischen veröffentlicht worden. Da Java zum Realisieren der Plug-Ins verwendet wird, ist Poseidon im Rahmen der Möglichkeiten von Java fernsteuerbar. Über die Komplexität kann nichts gesagt werden, da zum Zeitpunkt der Untersuchung keine Dokumentation zum Erweitern verfügbar war.
- Installationsumfang: Die Version, die mit einem Windows-Installationsprogramm ausgeliefert wird, verbraucht 18 MB Festplattenspeicher. In der Java Web Start-Variante werden 15 MB benötigt.
- Java Web Start: Die Java Web Start-Technologie wird unterstützt.
- Export/Import der UML-Diagramme: Poseidon speichert Diagramme als gezipptes XML-File ab. Dabei wird XMI[UML+DI] (siehe Kapitel 7.3) als Format gewählt, welches sich in der Abschlussphase der Standardisierung bei der Object Management Group [OMG04] befindet und demnächst als Standard zum Diagramm Interchange von UML-Diagrammen erhoben werden soll. Auch ein Export in ein Poseidon-eigenes XML-Format ist möglich. Natürlich können auch beide Formate zum Import genutzt werden. Allerdings können die im XMI-Format importierten Diagramme nur angezeigt werden, wenn der XMI-Standard zum „Diagramm Interchange“ verwendet wird. Die Diagramme lassen sich problemlos in gängige Grafikformate exportieren.
- Lizenzen: Es gibt eine kostenfreie Community Edition, die allerdings in ihrer Funktion eingeschränkt ist. Die in ein Grafikformat exportierten Diagramme besitzen alle einen Hinweis auf die Community Edition. Ferner bietet sie keine Möglichkeiten zum Drucken aus Poseidon heraus und kann auch keine Plug-Ins verwenden.

Die Standard Edition, die dies kann, kostet als einfache Lizenz: 200 €. Für Universitäten gibt es 50%-90% Rabatt abhängig von der Anzahl der gekauften Lizenzen.

### 7.2.6 Microsoft Visio 2002 Professional

Allgemein: „Visio Professional 2002, die erste von Microsoft entwickelte Visio-Version, stellt Diagrammlösungen zur Verfügung, die Experten im Unternehmen und im Technikbereich beim Dokumentieren und Kommunizieren einer Vielzahl von Ideen, Informationen und Systemen unterstützen. In Visio Professional erstellte Diagramme ermöglichen wertvolle Einblicke in bestehende technische Systeme und unterstützen sowohl Einzelpersonen als auch Teams, auf effektivere Weise neue Systeme zu entwickeln. Visio-Diagramme können zur Verbesserung von Text und Zahlen verwendet werden. Mitteilungen werden dadurch prägnanter, Benutzer können wichtige Punkte leichter in Erinnerung behalten, und kulturelle und technische Grenzen werden überschritten." (von der Microsoft-Homepage, inzwischen nicht mehr verfügbar ).
Visio stellt nicht nur UML Diagramme sondern, jegliche Art von Diagrammen zur Verfügung, sowohl aus dem geschäftlichen, wie auch aus dem technischen Bereich. Es benötigt keine Java-Laufzeitumgebung zur Ausführung.

Bedienung:

- Benutzerfreundlichkeit: Visio unterscheidet sich von Poseidon und Together dadurch, dass die gewünschten Diagrammelemente per Drag&Drop-Verfahren aus dem Auswahlbereich in das Diagramm transportiert werden und nicht mit einem Klick an der gewünschten Stelle erscheinen. Daran gewöhnt sich der Benutzer schnell. Die Benutzeroberfläche ist an die Office-Produkte angelehnt und so findet sich der Windows-Benutzer schnell zurecht. Auch das von Microsoft gewohnte Hilfesystem ist eingebaut und umfangreich.
- Installation: Die Installation verlief ohne Probleme.
- Umfang an UML-Modellierungsmöglichkeiten: Es wird die volle Palette an UML-Modellierungsmöglichkeiten unterstützt, die **UML 1.2** vorsieht. :
    - Klassendiagramme
    - Sequenzdiagramme
    - Anwendungsfalldiagramme
    - Zustandsdiagramme
    - Kollaborationsdiagramm
    - Komponentendiagramm
    - Aktivitätsdiagramme
        - Aktivitäten
        - Transitionen: Es werden sowohl Objektfluss- als auch Kontrollfluss-Transitionen unterstützt.
        - Objekte
        - Zustände
        - Start/End/Synchronisationszustände
        - Branch
        - Fork/Join
        - Kommentare
        - ***Bedingungen lassen sich nur an Transitionen über die Guard-Eigenschaft angeben.***
        - Constraints
        - Signale

        - Swimlanes
    - UML-Profile: Es sind alle Anpassungskonstrukte vorhanden. Benutzerdefinierte Eigenschaften lassen sich über TaggedValues oder Constraints angeben. Die Eigenschaften sind grundsätzlich umfangreicher als bei Together oder Poseidon, allerdings wird nur UML 1.2 unterstützt, das noch keine UML-Profile vorsieht.

Technische Kriterien :

- Technische Vorsaussetzungen:
    - Intel - Pentium 200 MHz oder besser
    - 64 MB Hauptspeicher
    - 210MB Festplatte
    - Windows 95,98,NT, 2000 oder XP
    - CD-Rom-Laufwerk
- Erweiterbarkeit: Das Werkzeug kann mit einem COM-Addin erweitert werden. Fernsteuerbar wird es über die OLE-Technik auf einem Windows-Rechner. Im Rahmen der COM-Addins kann es dann aber auch von anderen Rechnern gesteuert werden. Den Komplexitätsgrad der Erweiterung schätzt der Autor als hoch ein. Es müssen Kenntnisse in Windows Programmierung, Visio SDK, C++, COM und OLE vorhanden sein. Allerdings ist die Erweiterbarkeit gut getestet und hat sich vielfach schon bewährt (COM-Modell).
- Installationsumfang: Die Standardinstallation verbraucht 178 MB bis 210MB Festplattenspeicher.
- Java Web Start: Die Java Web Start-Technologie wird nicht unterstützt.
- Export/Import der UML-Diagramme: Es gibt ein Add-On, um Visio so zu erweitern, dass ein UML–Diagramm als XMI-File exportiert. werden kann. Allerdings muss dieses Add-on noch mithilfe des Visual.Net Studios kompiliert werden, bevor es eingesetzt werden kann (Dies wurde im Rahmen dieser Arbeit nicht durchgeführt). Der Import als XMI-File ist allerdings nicht möglich. Ansonsten werden zur Darstellung nur reine Grafikformate unterstützt, davon allerdings alle gängigen.
- Lizenzen: Eine Lizenz kostet ca. 570 €. 10 Lizenzen gibt es für ca. 1500 €. Für akademische Zwecke kann im Rahmen der MSDNAA ( Microsoft Developer Network Academic Alliance) Visio kostenfrei benutzt werden.

### 7.2.7 Visual UML 3.21

Allgemein: Visual UML ist ein reines UML-Modellierungstool zum Erstellen von UML-Modellen. Es besitzt zudem eine Funktion zur Codeerzeugung aus den erstellten UML-Modellen in verschiedene Programmiersprachen ([Vis04]).

Bedienung :

- Benutzerfreundlichkeit: Die Benutzeroberfläche ist an Visual Studio .NET angelehnt. Zum Zurechtfinden in großen Diagrammen ist ein Übersichtsfenster mit stufenlosem Zoom-Interface enthalten. Die Menüführung ist aus den Officeprogrammen gewohnt einfach, das Zeichnen von Diagrammen fällt von Anfang an leicht. Eigenschaften von Diagrammelementen lassen sich wie bei Rose sehr einfach über Registerkarten angeben.
- Installation: Die Installation verlief ohne Probleme. Ein Lizenzfile wurde nicht benötigt.
- Umfang an UML-Modellierungsmöglichkeiten: Es wird die volle Palette an UML-Modellierungsmöglichkeiten unterstützt, die UML 1.4 vorsieht:
    - Klassendiagramme
    - Sequenzdiagramme

- Anwendungsfalldiagramme
- Zustandsdiagramme
- Kollaborationsdiagramm
- Komponentendiagramm
- Aktivitätsdiagramme
  - Aktivitäten
  - Transitionen: Es werden sowohl Objektfluss- als auch Kontrollfluss-Transitionen unterstützt.
  - Objekte
  - Zustände
  - Start/End/Synchronisationszustände
  - Branch
  - Fork/Join
  - Kommentare
  - ***Bedingungen lassen sich nur an Transitionen über die Guard-Eigenschaft angeben.***
  - Signale
  - Swimlanes
- UML-Profile: Alle Anpassungskonstrukte werden unterstützt. UML-Profile als solches werden jedoch nicht unterstützt.

Technische Kriterien :

- Technische Vorsaussetzungen:
  - Alle Windows-Betriebssysteme werden unterstützt.
  - 64 MB Ram
  - 50-70MB Festplattenspeicher

- Erweiterbarkeit: Das Werkzeug ist laut Informationen des Herstellers erweiterbar. Eine genaue Dokumentation ist aber nicht frei verfügbar. Zum Erweitern ist die Standard Plus Version erforderlich (795 US $). Sie enthält Visual Basic for Applications 6.3 (VBA 6.3). Zum Erweitern werden VBA 6.3 und OLE Automation zusammen mit der ActiveX-Schnittstelle verwendet, wodurch das Werkzeug fernsteuerbar wird. Die Erweiterung ist nach Meinung des Autors mittelmäßig schwer. Die Hilfe scheint gut dokumentiert zu sein, ist allerdings erst mit dem Erwerb der Standard Edition verfügbar.
- Installationsumfang: Die Installation verbraucht 40 MB Festplattenspeicher.
- Java Web Start: Die Java Web Start-Technologie wird nicht unterstützt.
- Export/Import der UML-Diagramme: Das Werkzeug bietet die Möglichkeiten, ein komplettes UML-Modell in XMI zu exportieren und zu importieren. Einzelne Diagramme können nur in ein spezielles durch Visual UML 3.21 definiertes XML-Format exportiert werden. Es werden zahlreiche Grafikformate zum Export der Diagramme unterstützt.
- Lizenzen: Getestet wurde eine 30-tägige Testversion, die voll funktionsfähig war. Eine Standardlizenz kostet: 495 US $. Die Developer Edition ist für 795 US $ erhältlich.

## 7.3 Auswahl und Integration eines UML-Modellierungswerkzeugs

Im vorigen Abschnitt wurden fünf UML-Modellierungswerkzeuge untersucht. In diesem Abschnitt wird nun eines dieser Werkzeuge ausgewählt und beschrieben, wie dieses Werkzeug in die infoAssetBroker-Plattform integriert werden kann, um als Prozesseditor eingesetzt werden zu können.

### 7.3.1 Auswahl eines UML-Modellierungswerkzeugs

Der Grund, ein vorhandenes Werkzeug zu verwenden und nicht den existierenden Prozesseditor zu verwenden oder neu zu entwickeln, ist, dass die Integration eines Fremdwerkzeuges mit der Java Web Start-Technologie schnell umzusetzen ist, während eine Erweiterung des existierenden Prozesseditor eine nicht unerhebliche Entwicklungsdauer beanspruchen würde, die im Rahmen dieser Arbeit nicht durchzuführen wäre. Ein weiterer Grund ist, dass natürlich vorhandene Software wieder verwendet werden sollte, wenn das Anwendungsszenario es zulässt. In diesem Fall müssen aber auch Einschränkungen der wieder verwendeten Software in Kauf genommen werden, wenn sich die Software nicht anpassen lässt.
In dieser Arbeit soll das Werkzeug Poseidon der Firma Gentleware ([Gen04]) in der Community Edition als Prozesseditor verwendet werden. Die Gründe dafür sind:

- Die Community Edition ist frei verfügbar.
- Das Werkzeug unterstützt die weiter unten vorgestellte Java Web Start-Technologie und bietet damit auch gute Offline-Unterstützung, d.h. der Benutzer kann das Werkzeug auch ohne Verbindung mit dem Informationsportal benutzen.
- Das Werkzeug ist einfach bedienbar und besitzt eine ausgereifte Benutzerschnittstelle.
- Das Werkzeug bietet die Möglichkeit, UML-Diagramme im XMI-Format zu exportieren, wobei schon der kommende Standard UML 2.0 unterstützt werden soll.
- In der käuflich zu erwerbenden Standard Edition kann das Werkzeug mithilfe von in Java programmierten Plug-Ins angepasst werden.

### 7.3.2 XML Metadata Interchange (XMI)

Nach [Jck04] ist XMI ein durch die Object Management Group [OMG04] standardisierter Formalismus zum Generieren von XML-Vokabularen. Im XMI-Standard enthalten ist bereits ein Vokabular zur textuellen Repräsentation von graphisch dargestellten UML-Modellen. Aktuelle Versionen sind XMI 1.2 und 2.0 (1.2 erzeugt DTDs, 2.0 erzeugt XML Schemata, [Jck04]) Mit dem kommenden Standard UML 2.0 sollen dann auch Informationen zum „Diagramm Interchange“ in XMI mit aufgenommen werden. Dies sind Informationen über Aussehen und Layout der Diagramme, so dass andere UML-Werkzeuge die Diagramme importieren und genauso anzeigen können, wie sie erstellt wurden. Bisher wurden nur Informationen über die verwendeten UML-Elemente generiert.
In dieser Arbeit werden Geschäftsprozesse durch UML-Aktivitätsdiagramme definiert. Diese werden dann in das XMI-Format exportiert und in dieser Darstellung im Informationsportal gespeichert. Die Verwendung von XMI stellt sicher, dass das Werkzeug Poseidon jederzeit durch ein anderes ersetzt werden kann, das Aktivitätsdiagramme im XMI-Format exportieren und importieren kann.

### 7.3.3 Nachteile von Poseidon

Die Verwendung von Poseidon als UML-Modellierungswerkzeug bringt einige Nachteile mit sich: Poseidon unterstützt in der aktuellen Version 2.3 keine Signale und keine Swimlanes, die der UML 2.0 Standard vorsieht

und die für den Einsatz als Prozesseditor von Bedeutung sind. Swimlanes definieren Verantwortungsbereiche für Aktivitäten und könnten deshalb dazu verwendet werden, Rollen und Personen die Ausführung von Aktivitäten zuzuweisen, wenn diese in ihrem Verantwortungsbereich liegen. Signale modellieren Ereignisse (siehe Abschnitt 7.5). Nach Auskunft des Herstellers sollte die Unterstützung dieser beiden Modellierungselement aber bis zur aktuellen Version realisiert sein. Weil dies aber nicht der Fall ist, wird die Ereignisunterstützung in dieser Arbeit nicht praktisch umgesetzt. Die Verantwortungsbereiche für Aktivitäten werden über einen Eigenschaftswert realisiert (siehe Abschnitt 7.5).

### 7.3.4 Java Web Start - Technologie

Wie schon weiter oben erwähnt, soll das Werkzeug Poseidon mithilfe der Java Web Start - Technologie in die infoAssetBroker-Plattform integriert werden. Die Java Web Start- Technologie wird ausführlicher in [Ahm03, Sun04c] erklärt, so dass hier nur eine kurze Beschreibung präsentiert wird.
Die Java Web Start-Technologie erlaubt es, eine aus einem oder mehreren JAR-Archiven bestehende Java-Anwendung über eine bestehende Internetverbindung von einem Server zu laden und lokal auf einem PC zu installieren. Voraussetzung ist dafür, dass auf dem lokalen Rechner eine aktuelle Version des Java-Runtime-Environment (JRE) installiert ist, die den Java Web Start-Client enthält. Klickt ein Benutzer des lokalen Rechners auf einer Internetseite auf einen HTML-Link, der auf eine JNLP-Datei (JNLP: Java Network Launching Protocol [Sun04c]) verweist, so wird diese JNLP-Datei von dem lokalen Java Web Start-Client interpretiert. Eine JNLP-Datei enthält folgende Informationen über die zu installierende Anwendung, wobei einige optional sind: Name, Beschreibung, Hersteller, Namen der benötigten JAR-Archive, Hostname auf dem die benötigten Archive verfügbar sind, angeforderter Grad der Sicherheitsbeschränkung, Name der Klasse, die die zu startende Main-Methode enthält, sowie Startparameter.
Der Java Web Start-Client prüft zunächst, ob die durch die JNLP-Datei beschriebene Anwendung bereits auf dem lokalen Rechner installiert ist und ob die Version aktuell ist. Ist dies nicht der Fall, werden die benötigten Archive von der angegebenen Quelle angefordert und im Cache abgelegt. Stimmt der Benutzer dem Grad der Sicherheitsbeschränkung zu, wird danach die Anwendung mit dem Aufruf der angegebenen Main-Methode gestartet. Über den Grad der Sicherheitsbeschränkung wird definiert, welche Zugriffe auf dem lokalen Rechner durch die installierte Anwendung erlaubt sind. Dadurch, dass zunächst der Cache abgefragt wird, ist es möglich, dass die Anwendung beim zweiten Start ohne Internetverbindung ausgeführt werden kann, da alle benötigten Ressourcen lokal vorhanden sind.
Die infoAssetBroker-Plattform ist schon für die Integration von Java-Anwendungen per Java Web Start-Technologie vorbereitet. Zu diesem Zweck gibt es dort eine spezielle Schnittstelle, das „AppInterface“. Die Schnittstelle wird über eine XML-Datei, die Datei *AppInterfaceConfig.xm,l* konfiguriert. Hier werden die Einträge gemacht, die auch später in der JNLP-Datei stehen sollen. Die benötigten JAR-Archive müssen im Verzeichnis *AppInterface/distribution* abgelegt werden. Ein Administrator kann zur Laufzeit das Portal anweisen, nach neu installierten Anwendungen zu suchen. Dabei wird dann die neue Anwendung konfiguriert und ein Link auf einer der Portalseiten generiert. Wird dieser Link durch einen Besucher der Seite verwendet, dann wird vom Portal aus der *AppInterfaceConfig.xml* eine JNLP-Datei mit allen relevanten Informationen generiert und an den Webbrowser des Besucher gesendet. Dort startet dann der oben beschriebene Vorgang. Auf diese Weise wurde auch das Werkzeug Poseidon in die Portalplattform integriert.

### 7.3.5 Verwendung der WebDAV-Schnittstelle zum Einstellen von Prozessdefinitionen

Auf einem lokalen Rechner werden Geschäftsprozesse mithilfe von Poseidon als UML-Aktivitätsdiagramme definiert. Diese Diagramme werden dann in das XMI-Format exportiert und als XMI-Datei gespeichert. Damit diese XMI-Dateien aber durch das Prozessunterstützungsystem verarbeitet werden können, müssen sie von dem

lokalen Rechner auf das Portal publiziert werden. Dazu soll in dieser Arbeit eine WebDAV-Schnittstelle benutzt werden. WebDav steht für „Web-based Distributed Authoring and Versioning“. Mithilfe dieser Technologie können Internetbenutzer Dateien auf einem entfernten HTTP-Server kollaborativ verwalten und editieren. WebDav realisiert dies durch eine Erweiterung des HTTP-Protokolls [WDAV04].
Für die infoAssetBroker-Plattform wurde in [Ge03] die WebDav-Schnittstelle realisiert und integriert. Damit ist es möglich, dass Portalbenutzer über WebDav auf Dateien auf dem Portal zugreifen können, diese auf den lokalen Rechner laden können, oder lokale Dateien auf dem entfernten Portalserver publizieren können, wenn sie die Berechtigung dafür haben. Jede Datei, die durch das Portal zur Verfügung gestellt werden soll, wird durch ein Asset vom Typ Document repräsentiert. Ein Document-Asset speichert verschiedene Informationen über eine Datei: Autor, Beschreibung, Datei-Format, Inhalt und Publikationsdatum. Werden Dateien über die WebDAV-Schnittstelle auf dem Portal publiziert, die dort noch nicht vorhanden sind, so erkennt die WebDav-Schnittstelle dies und legt automatisch ein neues Document-Asset an. Als Autor wird der Benutzer gespeichert, der die Datei im Portal publiziert hat.
Da die WebDav-Schnittstelle auch mit einem einfachen Dateibrowser benutzt werden kann, wird sie zum Übertragen der XMI-Dateien verwendet. Der Prozessmodellierer (siehe Kapitel 5) kann die XMI-Dateien so von seinem lokalen Rechner auf das Portal übertragen, bzw. Prozessdefinitionen von dem Portal auf den lokalen Rechner laden.

## 7.4 Anpassungen der UML-Aktivitätsdiagramme

Dieser Abschnitt beschreibt, wie mithilfe der Anpassungskonstrukte von UML-Profilen die UML-Aktivitätsdiagramme so angepasst werden, dass sie zur Definition von Prozessen für das in dieser Arbeit zu entwickelnde Prozessunterstützungssystem eingesetzt werden können. Zwar lassen sich mit UML-Aktivitätsdiagrammen Aktivitäten modellieren, doch sind zusätzliche Informationen notwendig, damit diese Aktivitäten auf Instanzen der Klasse Activity abgebildet werden können. Diese zusätzlichen Informationen sind: Handlername, ausführende Standard-Rolle, ausführende Standard-Person, Variablenname für dynamische Zuweisung von Aktivitäten. Der Handlername gibt an, welcher Handler durch das Prozessunterstützungssystem aufgerufen werden soll, um die Aktivität auszuführen. Die ausführende Standard-Person bzw. Rolle soll angeben welche Rolle, oder Person die Aktivität normalerweise ausführen soll. Der Variablenname für dynamische Zuweisung von Aktivitäten weist auf eine Variable im Prozesskontext hin, die von der Prozessmaschine vor der Prozessausführung geprüft wird und auf eine Person oder Rolle verweisen kann, die die Aktivität ausführen soll, wenn nicht die Standard-Person oder -Rolle verwendet werden soll (siehe Kapitel 6). Für jede dieser vier Eigenschaften wurde ein Eigenschaftswert definiert, der dem Stereotyp <<Activity>> zugeordnet wird. Damit übernimmt jede Aktivität, die mit diesem Stereotyp markiert wird, diese Eigenschaften. Im Eigenschaftsfenster können diesen Eigenschaften Werte zugewiesen werden. Abbildung 7.2 zeigt das Eigenschaftsfenster des Werkzeugs Poseidon beispielhaft mit den vier Eigenschaften und zugeordneten Werten.

| Inhabender Stereotyp | Eigenschaft | Wert |
|---|---|---|
| << Activity >> | Role | Besichtiger |
| << Activity >> | Person | |
| << Activity >> | DynamicAssignment | assignedToSurveyor |
| << Activity >> | Template | processsupportwips/acceptOrder.htm |

Abbildung 7.2 Stereotyp <<Activity>> und zugeordnete Eigenschaftswerte

In der Abbildung 7.2 wird die Standard-Rolle durch die Eigenschaft *Role* angegeben. In diesem Falle wäre dies die Rolle „Besichtiger“. Die Eigenschaft Standard-Person ist nicht definiert, denn es ist kein Wert zu der Eigenschaft *Person* angegeben. Der Variablenname wird bei der Eigenschaft *DynamicAssignment* angegeben. In der Abbildung wäre dies der Name „assignedToSurveyor“. Die Eigenschaft *Template* dient der Angabe des Templa-

tenamens, der in der Abbildung 7.2 den Wert „processsupport/acceptOrder.htm“ hat. Wie weiter in der Abbildung zu sehen ist, sind die Eigenschaften alle dem Stereotyp <<Activity>> zugeordnet. Sie tauchen damit in jedem Eigenschaftsdialog eines Elementes auf, dass mit dem Stereotyp markiert wird. Nur Aktivitäten, die mit dem Stereotyp <<Activity>> markiert sind, sind Aktivitäten im Sinne der Prozessdefinition. Der Mechanismus, der dieses sicherstellt, wird im nächsten Abschnitt vorgestellt.
Eine weitere Anpassung der UML-Aktivitätsdiagramme ist für Entscheidungen notwendig. UML-Aktivitätsdiagramme bieten standardmäßig keine Unterscheidungsmöglichkeit für manuelle oder automatische Aktivitäten und Entscheidungen. Sie bieten eine Unterstützung für Entscheidungen, welche über ein Choice-Element modelliert werden können. Zur Unterscheidung zwischen manuell und automatisch wird ein weiterer Stereotyp eingeführt, der Stereotyp <<automatic>>. Um eine automatische Aktivität zu modellieren, muss eine Aktivität mit dem Stereotyp markiert werden. Andernfalls gilt die Aktivität als manuelle Aktivität. Gleiches gilt für Entscheidungen. Die Abbildung 7.3 zeigt eine automatische Entscheidung (*Abbruchprüfung*) gefolgt von einer automatischen Aktivität (*Auftraggeber per Email informieren*) in einem Ausschnitt eines UML-Aktivitätsdiagramms, das mit Poseidon modelliert wurde.

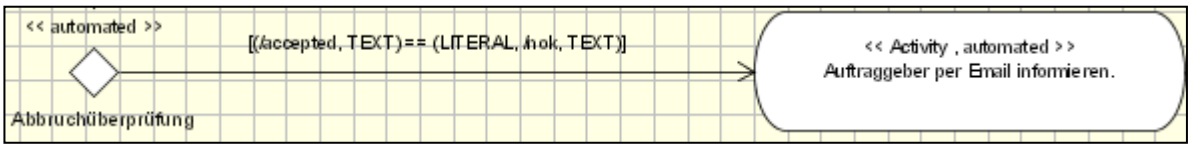


Abbildung 7.3 Verwendung des Stereotyps <<automated>>

In der Abbildung 7.3 zeigt sich noch eine weitere Anpassung. Die Bedingungen, die für die automatische Entscheidung ausgewertet werden sollen, werden als Kontrollflussbedingungen der ausgehenden Transitionen einer automatischen Entscheidung modelliert. Sie werden über die Wächter-Eigenschaft (engl. Guard), eine Eigenschaft, die alle Transitionen in UML standardmäßig besitzen, definiert. In der Abbildung 7.3 stellt die Zeichenkette „(/accepted, TEXT) == (LITERAL, /nok, TEXT)“ eine solche Bedingung dar. Um diese Bedingungen anzugeben wurde in dieser Arbeit eine eigene Grammatik definiert, die im Abschnitt 7.5 präsentiert wird. Der Dialog in Poseidon zur Angabe der Wächter-Eigenschaft wird in Abbildung 7.4 gezeigt. Ein Guard hat neben dem Ausdruck noch weitere Eigenschaften, wie z.B. den Namen, die aber nicht verwendet werden. Sie sind für die Prozessdefinition nicht relevant.

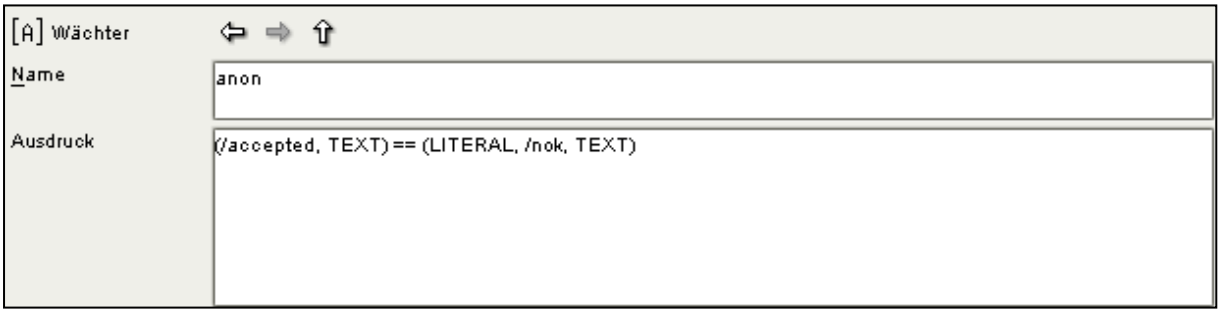


Abbildung 7.4 Dialog zur Angabe der Wächter-Eigenschaft in Poseidon

Die in diesem Abschnitt präsentierten Anpassungen des UML-Aktivitätsdiagramms mithilfe der Anpassungskonstrukte, die Poseidon bietet, müssen auch benutzt werden, wenn Geschäftsprozesse mithilfe eines UML-Aktivitätsdiagramms definiert werden sollen. Im Abschnitt 7.5 wird vorgestellt, wie das Prozessunterstützungssystem sicherstellt, dass UML-Aktivitätsdiagramme gültig sind, d.h. den präsentierten Anpassungen folgen.

## 7.5 Konvertieren von Prozessdefinitionen

In diesem Abschnitt wird der Konvertierungsmechanismus besprochen, der in dieser Arbeit realisiert wurde, um Prozessdefinitionen in XMI im Portal auf ProcessDefinitions abzubilden. Zu diesem Zweck wurde ein eigener Prozess definiert und realisiert. Es handelt sich dabei um den Konvertierungsprozess, der nun zunächst beschrieben werden soll.

### 7.5.1 Der Konvertierungsprozess

Der Konvertierungsprozess hat die Aufgabe, zu einer als XMI-Datei im Portal gespeicherten Prozessdefinition, die entsprechenden Instanzen der Klassen des Prozessmodells zu erzeugen, so dass der durch die XMI-Datei definierte Prozess durch das Prozessunterstützungssystem ausgeführt werden kann. Damit der Konvertierungsprozess diese Aufgabe durchführen kann, muss er durch das System benachrichtigt werden, wenn eine neue Prozessdefinition vorliegt. Zu diesem Zweck wurde die Klasse WIPSNewFileEvent implementiert. Sie erbt von der abstrakten Klasse Event und stellt damit ein Ereignis dar, dessen Typ auf „WIPSNewFileEvent“ festgelegt wurde (siehe Kapitel 6.4). Die WebDav-Schnittstelle wurde so erweitert, dass sie dieses Ereignis auslöst, wenn eine XMI-Datei, die eine Prozessdefinition darstellt, im Dateisystem des Portals abgelegt wird. Des Weiteren sollte die WebDav-Schnittstelle so konfiguriert werden, dass sie ein Überschreiben von existierenden Prozessdefinitionen verhindert. Änderungen sollen explizit als neue Version modelliert werden (siehe auch Abschnitt 7.4.2 und 6.2).
Um eine Datei als XMI-Datei für Prozessdefinitionen zu klassifizieren, wird ein Klassifizierer eingesetzt. Für diesen Zweck wurde die Klasse ProcessDefinitionClassifier implementiert. Eine Instanz dieser Klasse durchsucht XML-Dateien (XMI erzeugt eine XML-basierte Sprache) nach bestimmten Kriterien. Eine XML-Datei gilt dann als Prozessdefinition, wenn sie einen Knoten <XMI> enthält, der ein Attribut xmi.version besitzt, das den Wert „1.2“ besitzt und wenn es die folgende Knotenhierarchie in der XML-Datei gibt:

```
<XMI>
    <XMI.content>
        <ActivityGraph>
                        …
```

Die erste Bedingung stellt sicher, dass nur XMI-Dateien, die den aktuellen XMI-Standard in der Version 1.2 benutzen, als Prozessdefinition gelten können. Die zweite Bedingung stellt sicher, dass die XMI-Datei mindestens ein Aktivitätsdiagramm beschreibt, denn Aktivtätsdiagramme sollen ja zum Definieren eines Geschäftsprozesses in dieser Arbeit verwendet werden.
Die WebDav-Schnittstelle klassifiziert also mit einer Instanz der Klasse ProcessDefinitionClassifier neu im Portal gespeicherte Dateien. Sollte dabei eine XMI-Datei, die eine Prozessdefinition enthält, entdeckt werden, so wird ein Ereignis vom Typ „WIPSNewFileEvent“ ausgelöst, das in einem Attribut die Referenz auf das Document-Asset speichert, das die XMI-Datei beschreibt (siehe Abschnitt 7.3.6). Dieses Ereignis wird vom Konvertierungsprozess verarbeitet.
Der Konvertierungsprozess selbst wird einmalig durch einen Handler initialisiert, bevor er XMI-Dateien konvertieren kann. Dabei wird der Konvertierungsprozess so konfiguriert, dass maximal eine Instanz des Prozesses zur Laufzeit existiert. Nach der einmaligen Initialisierung wird der Prozess jedes Mal beim Starten des Portals initiiert, solange der Administrator dies nicht anders konfiguriert.
Die Abbildung 7.5 zeigt das Aktivitätsdiagramm des Konvertierungsprozesses. Da es sich nicht um die Definition eines Prozesses handelt, der konvertiert werden soll, sondern um den Konvertierungsprozess selber, wurden die Stereotypen <<Activity>> und <<automated>> weggelassen. Zu jeder der dort gezeigten Aktivitäten wurde eine Handlerklasse implementiert, deren Instanzen die jeweilige Aktivität ausführen sollen (vgl. auch Kapitel 6.2 und 6.5). Die Aktivitäten werden im Folgenden kurz vorgestellt.

**WIPSNewFileEvent**

Dieses Ereignis wird im Diagramm als empfangenes Signal modelliert. Da diese Funktionalität aber nicht im Prozessunterstützungssystem umgesetzt werden konnte, wird diese Aufgabe durch eine automatische Aktivität übernommen. Die Handler-Klasse für diese Aktivität ist die Klasse WaitForChangedHandler. Im Handler wird eine Instanz der Klasse RessourceSpecificListener erzeugt und als EventListener für diese Handler-Klasse und den Ereignistyp „WIPSNewFileEvent" registriert (vgl. Kapitel 6.4).
Im Anschluss wird die erzeugte Instanz solange abgefragt, bis ein Ereignis vom Typ „WIPSNewFileEvent" empfangen wird. Dann wird die automatische Aktivität beendet.

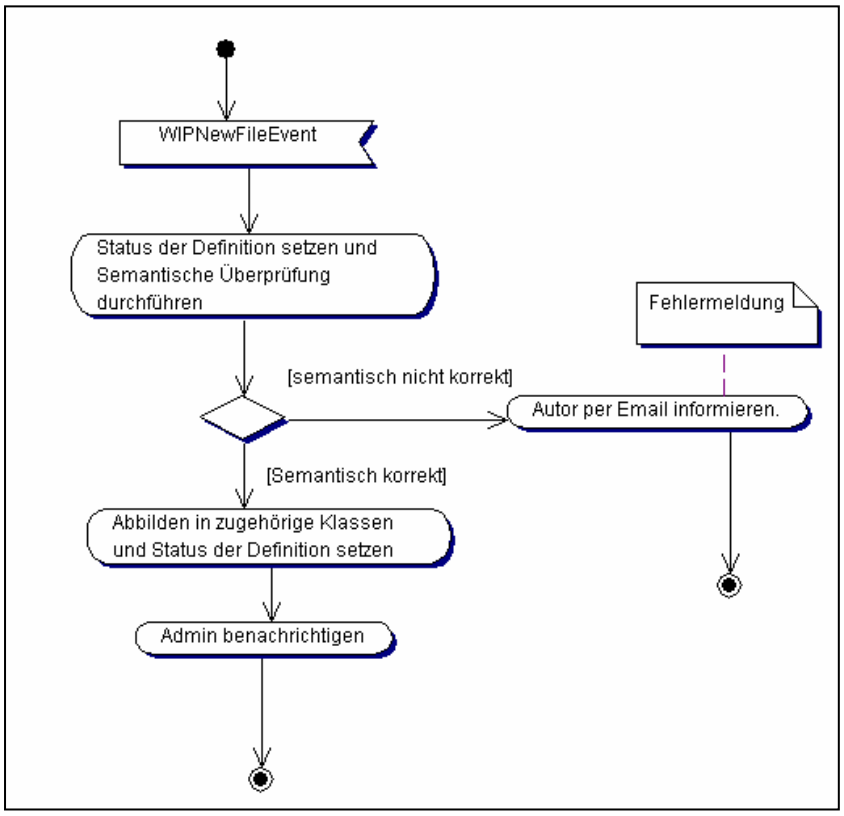


Abbildung 7.5 UML-Aktivitätsdiagramm des Konvertierungsprozesses

**Status der Definition setzen und Semantische Überprüfung durchführen**

Die Handler-Klasse, die diese Aktivität ausführen soll, ist die Klasse VerifyProcessDefinitionHandler. Im Handler wird die durch die XMI-Datei beschriebene Prozessdefinition auf Korrektheit geprüft. Dies ist notwendig, damit Modellierungsfehler entdeckt werden und damit die eigentliche Konvertierung ohne Fehler durchgeführt werden kann. Dazu wird die im nächsten Abschnitt 7.2 beschriebene Klasse XMIConverter verwendet. Ist die Prozessdefinition korrekt, wird die Variable mit dem Namen „verified" auf den Wert true gesetzt, anderenfalls auf false. Für den Fall, dass die Prozessdefinition nicht korrekt ist, liefert die Klasse XMIConverter eine Fehlermeldung, die dann durch den Handler im Prozesskontext abgelegt wird. Die Variable „verified" wird von der Prozessmaschine bei der auf diese Aktivität folgenden automatischen Entscheidung ausgewertet. Zum Abschluss wird der entsprechende Zustand dieser Prozessdefinition gesetzt. Dieser Zustand wird in einem speziellen ProcessDefinitionProject-Asset gespeichert, das hier nicht näher erläutert werden soll. Es sind fünf Zustände definiert: Neu (initialer Zustand), Konvertiert, Verifikationsfehler, Konvertierungsfehler und gelöscht.

**Autor per Email informieren**

Diese Aktivität wird nur ausgeführt, wenn ein Verifikationsfehler aufgetreten ist. Es wird durch das Prozessunterstützungssystem eine Email an den Autor der Prozessdefinition gesendet, welche die Fehlermeldung aus dem Prozesskontext enthält. Voraussetzung ist, dass der Autor definiert ist und eine gültige Email-Adresse in seinem Benutzerprofil für das Portal angegeben hat. Die Handler-Klasse, die dies realisiert, ist die Klasse InformAuthorHandler.

**Abbilden in zughörige Klassen und Status der Definition setzen**
Diese Aktivität wird nur ausgeführt, wenn kein Verifikationsfehler aufgetreten ist. Es werden zu der XMI-Datei Instanzen der Klassen des Prozessmodells erzeugt, die den Prozess im Portal definieren. Dazu wird die Klasse XMIConverter (siehe 7.4.2) eingesetzt. Dieser Vorgang wird in dieser Arbeit als Konvertierungsvorgang bezeichnet. Verläuft er fehlerfrei, so wird der Zustand der Prozessdefinition auf „Konvertiert", anderenfalls auf „Konvertierungsfehler" gesetzt. Es wird im Fehlerfall eine Fehlernachricht im Prozesskontext gespeichert, die den aufgetretenen Fehler näher beschreibt. Die Handler-Klasse, die dies realisiert, ist die Klasse ConvertProcessDefinitionHandler.

**Admin benachrichtigen**
Diese Aktivität sendet eine Nachricht an die Administratoren des Prozessunterstützungssystems und informiert sie über den Zustand der Prozessdefinition. Damit Prozesse einer erfolgreich konvertierten Prozessdefinition gestartet werden können, muss dies durch die Administratoren noch erlaubt werden (siehe Kapitel 5.4). Sollte die Konvertierung fehlerhaft sein, so muss ein Administrator eingreifen und den Fehler analysieren. Die Handler-Klasse, die diese Emails verschickt, ist die Klasse InformAdminHandler.

### 7.5.2 Die Klasse XMIConverter

Diese Klasse wurde implementiert, um zwei Funktionen zu realisieren: Die semantische Prüfung von als XMI-Datei vorliegenden Prozessdefinitionen und die Durchführung des im vorigen Abschnitt erwähnten Konvertierungsvorgangs. Dazu besitzt die Klasse zwei öffentliche, statische Methoden verify() und convert(). Die Methode verify() führt für ein als Parameter übergebenes Dokument die semantische Überprüfung durch und löst einen Fehler aus, wenn das Dokument keine korrekte Prozessdefinition beschreibt. Die Methode convert() führt den Konvertierungsvorgang durch und löst eine Fehlermeldung aus, falls ein Fehler während des Konvertierens auftritt.

**Die semantische Überprüfung**
Um eine semantische Überprüfung durchführen zu können, muss zunächst definiert werden, wann eine als XMI-Datei vorliegende Prozessdefinition korrekt ist. Dazu wird in dieser Arbeit Folgendes definiert:

Eine als XMI-Datei vorliegende Prozessdefinition gilt als semantisch korrekt, wenn die folgenden Kriterien erfüllt sind:

- Es wird der XMI-Standard in der Version 1.2 verwendet.
- Es wird ein Stereotype <<Activity>> verwendet, dem die oben beschriebenen Eigenschaften zugeordnet sind.
- Es ist ein Stereotyp <<automated>> definiert.
- Es wird nur ein UML-Aktivitätsdiagramm pro XMI-Datei beschrieben (Hierzu ist zu ergänzen, dass der XMI-Standard auch mehrere zulässt).
- Es ist ein Anfangszustand definiert.
- Es ist mindestens ein Endzustand definiert (Jeder Prozess sollte sich per Definition beenden lassen, auch wenn sich Schleifen im Prozessablauf befinden.).

- Es ist mindestens eine Aktivität modelliert.
- Alle modellierten Aktivitäten sind mit dem Stereotyp <<Activity>> markiert.
- Es sind mindestens zwei Transitionen modelliert.
- Wenn Entscheidungen modelliert sind, dann sind den ausgehenden Transitionen Bedingungen zugeordnet.
- Alle Bedingungen entsprechen der im nächsten Abschnitt erläuterten Syntax.
- Die Transitionen verbinden nur Elemente, die auf Instanzen des Prozessmodells abgebildet werden können, d.h. Anfangszustände, Aktivitäten, Endzustände, Entscheidungen, Join, Fork, Merge und Endzustände.

Diese Kriterien werden in der Methode verify() überprüft. Dazu wird überprüft, ob die entsprechenden Knoten in der XMI-Datei vorhanden sind, und ob ihre Anzahl mit der in Kriterien angegebenen übereinstimmt. In [Uml14Di] ist die DTD definiert, auf deren Grundlage Poseidon UML-Aktivitätsdiagramme in XMI-Dateien exportiert. Dort wird beschrieben, welche Struktur ein XMI-Dokument hat.
Zum Überprüfen der Bedingungen wird ein Syntax-Parser eingesetzt, der im nächsten Abschnitt beschrieben wird.
Ist eine Prozessdefinition nicht korrekt, so wird ein Fehler ausgelöst. Dabei wurde zu jedem Kriterium eine eigene Fehlerklasse implementiert, so dass sich die Fehler unterscheiden lassen. Die Fehlerklassen erben alle von der gemeinsamen Fehlerklasse XMIVerifyException, die von der Java-Standardklasse Exception erbt.

**Speichern von Versionsinformationen**
Der XMIConverter hat auch die Aufgabe Versionsinformationen in einer Prozessdefinition zu speichern. Zu diesem Zweck sollten alle Aktivitätsdiagramme das in Abbildung 7.6 gezeigte Kommentarfeld enthalten. Der Inhalt dieses Kommentarfeldes wird von der Methode convert() während der Konvertierung überprüft und entsprechend angepasst. Es enthält einen Hinweis auf die Vorgängerversion und die Gruppe der Prozessdefinition (siehe Kapitel 6.1 und 6.2). Bei der Konvertierung wird dann als Vorgängerversion der Dateiname der gerade konvertierten Version eingetragen. So kann bei Änderungen die Vorgängerversion bestimmt werden und eine Instanz in der Versionshistorie entsprechend erzeugt werden. Deshalb darf der Inhalt dieses Kommentars nicht geändert werden.

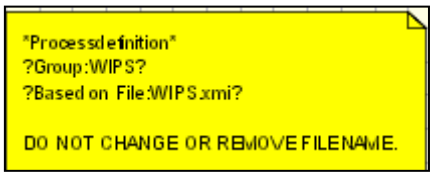


Abbildung 7.6 UML-Kommentarfeld mit Angabe zur Vorgängerversion

**Der Konvertierungsvorgang**
Wie schon erwähnt wird dieser Vorgang in der Methode convert() durchgeführt. Es werden entsprechend der Elemente des UML-Aktivitätsdiagramms, die durch Knoten in der XMI-Datei repräsentiert werden, Instanzen von Klassen des Prozessmodells erzeugt. Es wird nun beschrieben, welche Instanzen zu welchen Elementen erzeugt werden:

1.) UML-Aktivitätsdiagramm
Da nur ein Aktivitätsdiagramm durch eine XMI-Datei beschrieben werden darf, wird während des Konvertierungsvorganges eine Instanz der Klasse ProcessDefinition erzeugt. Als Name wird dabei der Dateiname der

XMI-Datei verwendet. Die Beschreibung ist in der XMI-Datei in einem Knoten gespeichert. Für jedes UML-Element und Diagramm lässt sich in Poseidon solch eine Beschreibung angeben. Des Weiteren wird die Instanz als Nachfolger-Instanz einer existierenden Prozessdefinition angelegt, wenn das XMI-Dokument einen Hinweis auf die Vorgänger-Version enthält (s. o.).

2.) Aktivitäten

Aktivitäten sind mit dem Stereotyp <<Activity>> markiert. Jede Aktivität wird auf eine Instanz der Klasse Activity abgebildet. Sollte der Name der Aktivität, der in der Mitte der Aktivität angegeben wird, bereits vorhanden sein und auch der Typ der Aktivität übereinstimmen, dann wird keine neue Instanz erzeugt, sondern die bereits vorhandene genutzt. Ist die Aktivität auch mit dem Stereotyp <<automated>> markiert, so wird eine automatische Aktivität erzeugt, anderenfalls eine manuelle. Wenn eine Instanz erzeugt wird, werden die Beschreibung sowie die Werte der vier Eigenschaften dieser Aktivität, die dem Stereotyp <<Activity>> zugeordnet sind, als Eigenschaften der erzeugten Instanz gesetzt. Sie sind ebenfalls in der XMI-Datei gespeichert. Abbildung 7.7 zeigt eine manuelle Aktivität, wie sie in Poseidon in UML-Aktivitätsdiagrammen modelliert wird.


Abbildung 7.7 Aktivität im UML-Aktivitätsdiagramm

3.) Endzustände

Abbildung 7.8 zeigt einen Endzustand, wie er in UML-Aktivitätsdiagrammen modelliert wird. Für jeden Endzustand in einem Aktivitätsdiagramm wird während der Konvertierung eine Instanz der Klasse Activity vom Typ „Endstate“ erzeugt. Name und Beschreibung werden so gesetzt, wie sie in der XMI-Datei definiert sind und im Diagramm angegeben wurden.

Abbildung 7.8 Endzustand im UML-Aktivitätsdiagramm

4.) Anfangszustand

Abbildung 7.9 zeigt die Grafik, die einen Anfangszustand darstellt. Für Anfangszustände werden keine neuen Instanzen erzeugt. Vielmehr wird in der Instanz der Klasse ProcessDefinition, die zu dem gerade konvertierten Aktivitätsdiagramm erzeugt wurde, die Eigenschaft „StartActivity“ entsprechend gesetzt. Dazu wird die ausgehende Transition des Anfangszustandes bestimmt, und deren Ziel als Start-Aktivität festgelegt.

Abbildung 7.9 Anfangszustand im UML-Aktivitätsdiagramm

5.) Transitionen

In der Abbildung 7.10 wird eine Transition mit zugeordneter Bedingung gezeigt. Während der Konvertierung wird zu jeder Transition eine neue Instanz der Klasse Transition erzeugt. Das Transitions-Ziel und der Ursprung der Transition werden als Eigenschaften der neu erzeugten Instanz zugewiesen. Ist der Transition eine Bedingung zugewiesen, so wird der Bedingsausdruck durch den im nächsten Abschnitt beschriebenen Syntax-Parser in

eine Instanz der Klasse Condition verwandelt. Eine Referenz auf diese Bedingung wird der Transitions-Instanz übergeben. Referenzen auf alle neu erzeugten Transitionen werden in der ProcessDefinition gespeichert.

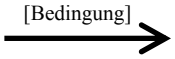


Abbildung 7.10 Transition im UML-Aktivitätsdiagramm

6.) Fork-Elemente

Die Abbildung 7.11 zeigt ein Fork-Element. Zu jedem Fork-Element in einem Aktivitätsdiagramm wird während der Konvertierung eine Instanz der Klasse Activity vom Typ „Fork“ erzeugt. Name und Beschreibung werden so gesetzt, wie sie in der XMI-Datei definiert sind und im Diagramm angegeben wurden.

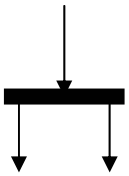

Abbildung 7.11 Fork-Element im UML-Aktivitätsdiagramm

7.) Join-Elemente

In der Abbildung 7.12 wird ein Join-Element gezeigt. Zu jedem Join-Element in einem Aktivitätsdiagramm wird während der Konvertierung eine Instanz der Klasse Activity vom Typ „Join“ erzeugt. Name und Beschreibung werden so gesetzt, wie sie in der XMI-Datei definiert sind und im Diagramm angegeben wurden. In der aktuellen Version wird als Synchronisationsbedingung als Standard „JOIN_AND“ gesetzt. Andere Werte („JOIN_OR“) werden zurzeit nicht unterstützt.

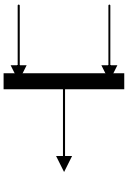

Abbildung 7.12 Join-Element im UML-Aktivitätsdiagramm

8.) Manuelle Entscheidungen

Manuelle Entscheidungen unterscheiden sich in der Modellierung von automatischen. Die Abbildung 7.13 zeigt eine manuelle Entscheidung. Zu jeder manuellen Entscheidung wird eine Instanz der Klasse Activity vom Typ „ManualDecision“ erzeugt. Da manuelle Entscheidungen in der Ausführung mit manuellen Aktivitäten zu vergleichen sind, werden auch die gleichen Eigenschaften aus der XMI-Datei ausgelesen und der Instanz übergeben. Der Bedingungsausdruck der ausgehenden Transitionen (in der Abbildung „InfoText1“ und „InfoText2“) müssen keiner bestimmten Syntax genügen. Sie sollen dem Benutzer bei der Ausführung informativ angezeigt werden und die Alternativen beschreiben. Zu diesem Zweck wird für jeden Bedingungsausdruck eine Instanz der Klasse Condition erzeugt und der Text in einer Eigenschaft dieser Instanz gespeichert. Diese Art der Bedingungserzeugung unterscheidet sich von der bei Transitionen eingesetzten, weshalb die ausgehenden Transitionen und ihre zugeordneten Bedingungen vor allen anderen Transitionen erzeugt werden. Es werden grundsätzlich auch mehr als die in der Abbildung gezeigten zwei Transitionen unterstützt.

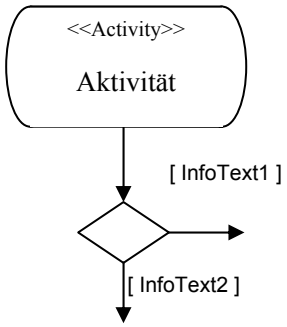


Abbildung 7.13 Manuelle Entscheidung im UML-Aktivitätsdiagramm

9.) Automatische Entscheidung

In der Abbildung 7.14 wird eine automatische Entscheidung gezeigt. Zu jeder automatischen Entscheidung in einem Aktivitätsdiagramm wird während der Konvertierung eine Instanz der Klasse Activity vom Typ „automatedDecision“ erzeugt. Name und Beschreibung werden so gesetzt, wie sie in der XMI-Datei definiert sind und im Diagramm angegeben wurden.

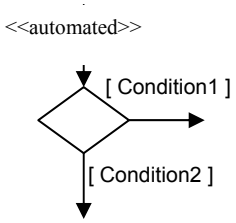


Abbildung 7.14 Automatische Entscheidung im UML-Aktivitätsdiagramm

10.) Merge-Elemente

Die Abbildung 7.15 zeigt ein Merge-Element. Zu jedem Merge-Element in einem Aktivitätsdiagramm wird während der Konvertierung eine Instanz der Klasse Activity vom Typ „Merge“ erzeugt. Name und Beschreibung werden so gesetzt, wie sie in der XMI-Datei definiert sind und im Diagramm angegeben wurden. Es sind grundsätzlich auch mehr als die in der Abbildung gezeigten zwei eingehenden Transitionen möglich.

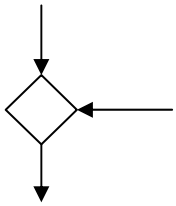

Abbildung 7.15 Merge-Element im UML-Aktivitätsdiagramm

11.) Nicht realisierte Ereignisunterstützung

In den Abbildungen 7.16 und 7.17 werden ein eingehendes und ein ausgehendes Signal gezeigt. Für diese Elemente sollten Instanzen der Klasse Activity vom Typ „ReceiveSignal“ für eingehende Signale, bzw. „SendSignal“ für ausgehende Signale erzeugt werden, um eine Ereignisunterstützung zu realisieren. Da sich diese Elemente mit Poseidon bis zum Ende dieser Arbeit nicht mit Poseidon darstellen ließen und folglich auch nicht in eine XMI-Datei exportiert werden konnten, wurde die Konvertierung für diese Element nicht implementiert.


Abbildung 7.16 Ausgehendes Signal im UML-Aktivitätsdiagramm


Abbildung 7.17 Eingehendes Signal im UML-Aktivitätsdiagramm

Die Methode convert() gibt als Rückgabe eine Referenz auf die neu erstellte Instanz der Klasse ProcessDefinition. Sollte die Konvertierung fehlschlagen, so bricht die Methode mit einer entsprechenden Ausnahme ab.

### 7.5.3 Prüfen und Parsen von Bedingungen

Die Bedingungen werden über die Guard-Eigenschaft einer Transition in Poseidon definiert. Auf der Ebene des Prozessmodells sollen sie auf Instanzen der Klasse Condition abgebildet werden. Diese Klasse ist Teil eines Bedingungsmodells, welches in [Gic03] für die infoAssetBroker-Plattform entworfen und realisiert wurde. Dieses Bedingungsmodell ist in Abbildung 7.18 gezeigt. Die abstrakte Klasse Condition repräsentiert eine Bedingung. Bedingungen können mit der Methode evaluate() unter Übergabe einer Instanz der Klasse EvaluationContext ausgewertet werden. Die Methode liefert einen Wahrheitswert zurück, der true ist, wenn die Bedingung erfüllt ist, anderenfalls false. Die Klasse EvaluationContext repräsentiert eine Menge von Variablenbindungen (Name-Wert-Paaren), auf der die Bedingung ausgewertet wird.

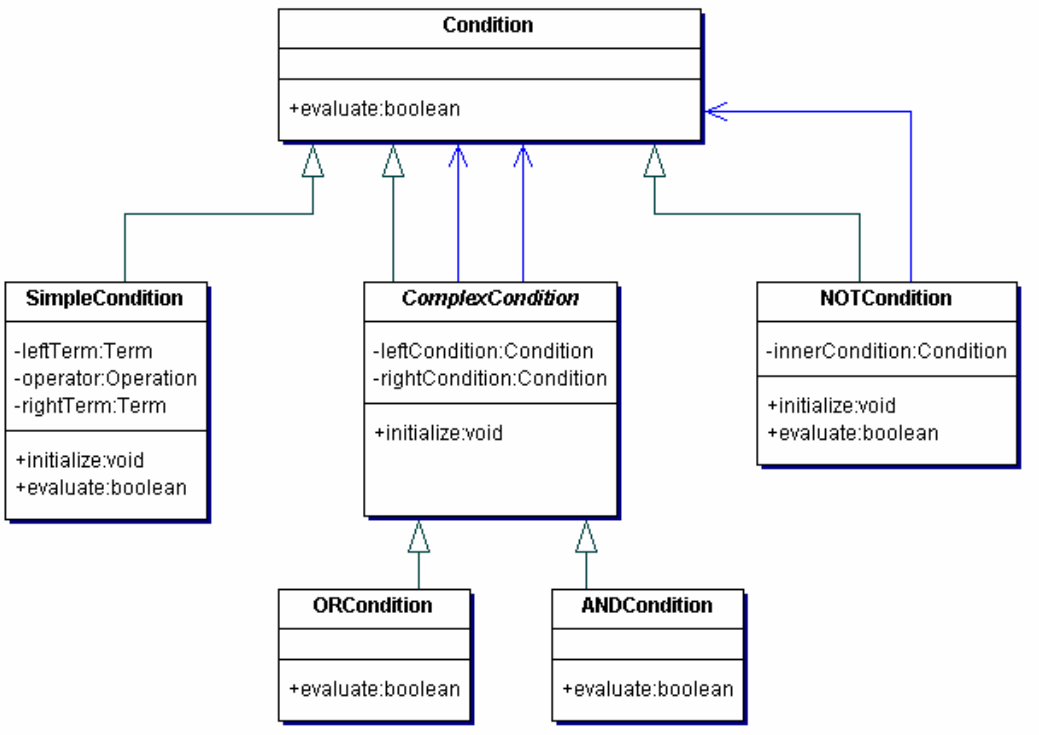


Abbildung 7.18 Bedingungsmodell aus [Gic03]

Eine konkrete einfache Bedingung wird durch die Klasse SimpleCondition implementiert. Die Klasse definiert zwei Attribute vom Typ Term. Die Klasse Term repräsentiert eine Variable im EvaluationContext oder einen festen Wert (Literal). Das Attribut *compOperator* vom Typ Operation stellt einen Vergleichsoperator dar, der zwischen den beiden Werten der Variablen oder Literale, die durch *leftTerm* und *rightTerm* identifiziert werden, eingesetzt wird. Das Ergebnis der Vergleichsoperation muss wahr sein, damit die Bedingung erfüllt ist. Als Vergleichsoperatoren kommen die Folgenden in Frage: < , >, ==, != . Alle unterstützten Operatoren sind in [Gic03] aufgeführt.
Die abstrakte Klasse ComplexCondition stellt eine aus zwei Bedingungen zusammengesetzte Bedingung dar. Konkrete Subklassen sind ORCondition (realisiert eine Logisch-Oder-Verknüpfung) oder ANDConditon (realisiert eine Logisch-Und-Verknüpfung). Die Klasse NOTCondition realisiert eine logische Verneinung einer Bedingung. Eine genauere Beschreibung des Bedingungsmodells ist in [Gic03] zu finden.
Um eine Bedingung textuell in Aktivitätsdiagrammen zu definieren, ist eine Sprache notwendig. Die Grammatik der Sprache wird im Folgenden definiert. Die Grammatik orientiert sich am Bedingungsmodell aus Abbildung 7.18 und wird im Folgenden in der Erweiterten Backus-Naur-Form dargestellt:

```
CONDITION ::= not CONDITION | SIMPLECOND | COMPLEXCOND
COMPLEXCOND ::= opbr CONDITION logicop CONDITION clbr
SIMPLECOND ::= TERM compop TERM
TERM ::= lbracket (literal comma)? expression comma datatype rbracket

logicop ::= "&&" | "||"
not ::= "!!"
compop ::= "==" | "<" | ">" | "<=" | ">=" | "!=" | "in"
literal ::= "LITERAL"
lbracket ::= "("
rbracket ::= ")"
opbr ::= "["
clbr ::= "]"
comma ::= ","
expression ::= ("/" ("a".."z"|"A".."Z"|"0".."9")+)+
datatype ::= "DATE" | "DOUBLE" | "INTEGER" | "TEXT" | "OBJECT" | "BOOLEAN"
```

Das Symbol `CONDITION` ist das Startsymbol der Grammatik. Die Ähnlichkeit mit den Klassennamen ist nicht zufällig. Wenn eine der Produktionsregeln für die Nicht-Terminal-Symbole (`CONDITION`, `COMPLEXCOND`, `SIMPLECOND`, `TERM`) angewandt wird, ist dies gleichbedeutend mit dem Erzeugen einer Instanz dieser Klasse, um die Bedingungen zu modellieren. Die Produktionsregeln für die Terminal-Symbole definieren, welche Zeichenfolge welches Terminal-Symbol darstellt. Besonders erwähnt werden soll das Terminal-Symbol `expression`. Es definiert einen Pfadausdruck zu einer Variablen im Prozesskontext. Jeder Pfad beginnt mit einem „/“. Jeder weitere Schrägstrich bedeutet, dass nun in die Eigenschaften eines Objektes im EvaluationContext verzweigt wird. Beispiel: „/person/name“. Damit wird die Eigenschaft „name“ der Variablen „person“ im EvaluationContext adressiert. Weitere Informationen werden in [Gic03] präsentiert. Mit dem Terminal-Symbol `datatype` wird der Datentyp der Variable, die durch `expression` adressiert wird, angegeben.
Mit dieser Grammatik ist es nun möglich, Bedingungen textuell zu definieren, die auf die Bedingungskonzepte im Portal abgebildet werden können. Allerdings fehlt noch eine Software-Komponente, die den Bedingungsausdruck auf korrekte Syntax während der semantischen Überprüfung testen kann und in der Konvertierungsphase Instanzen des Bedingungsmodells erzeugt. Für diese Aufgabe wurde ein Syntax-Parser realisiert.

Syntax-Parser lesen einen Ausdruck ein und prüfen ihn gegen eine gegebene Grammatik. Ein Syntax-Parser, der einen gegebenen Ausdruck von links nach rechts einliest und nach links ableitet (d.h. er versucht von links möglichst weit nach rechts zu gelangen, ohne dass die Grammatik verletzt wird) heißt LL-Parser. (engl., **L**eft to right, **L**eftmost derivation)
Zur Erstellung des Syntax-Parsers wurde das Framework ANTLR verwendet [Pa04]. ANTLR steht für Another Tool for Language Recognition. Dieses Framework erleichtert die Erstellung eines Parsers sehr. Der Entwickler muss nur eine Grammatik definieren und daraus generiert ANTLR den Sourcecode für die gewünschte Programmiersprache. ANTLR unterstützt die Programmiersprachen Java, C# und C++.
Mit ANTLR erfolgt der Bau eines Parsers in drei Phasen:

Phase 1 : Definition einer Grammatik-Datei. Die Grammatik-Datei enthält Informationen über die Produktionsregeln für den Parser und die Klassennamen für Lexer- und Parserklassen. Es lässt sich auch Sourcecode der gewählten Programmiersprache angeben, der immer dann ausgeführt wird, wenn eine Regel ausgelöst wird.

Phase 2: ANTLR generiert aus der Grammatik-Datei eine Lexer-Klasse, eine Parser-Klasse und optional eine Tree-Parser-Klasse.

Phase 3: Anpassen des generierten Sourcecodes an die jeweiligen Anforderungen der Anwendung und kompilieren.

Ein Lexer hat die Aufgabe, eine Folge von Eingabezeichen nach definierten Regeln in sog. Token-Objekte umzuwandeln. Token-Objekte sind Instanzen der Klasse Token des ANTLR-Frameworks und repräsentieren eine Folge von Zeichen, die eine syntaktische Einheit repräsentieren. Der Parser sucht in einer Folge von Token-Objekten nach Mustern, die durch Parser-Regeln definiert sind. Die Parser-Regeln entsprechen den Produktionsregeln einer Grammatik. Der Tree-Parser hat dieselbe Aufgabe wie ein Parser. Allerdings durchsucht er einen Baum, den sog. Syntaxbaum, von Token-Objekten, der durch den Parser aufgebaut werden kann, wenn dies in der Grammatik-Datei konfiguriert ist. Ein Tree-Parser wird benötigt, wenn Berechnungen ausgeführt werden müssen, die ein Einlesen des kompletten Ausdrucks erfordern. In diesem Fall dient der Syntaxbaum als Zwischenform des durch den Parser interpretierten Ausdrucks (vgl. auch Dokumentation auf [Pa04]).
Für die Generierung des Sourcecodes wurde die Grammatik-Datei so definiert, dass die oben beschriebene Grammatik durch den Parser erkannt werden kann. Des Weiteren wurde ANTLR so konfiguriert, dass der Parser einen Syntaxbaum erstellt. Es wurde eine Tree-Parser-Klasse definiert. Die Grammatik-Datei ist im Anhang C aufgeführt. Sie definiert die Klassen ConditionLexer, ConditionParser und ConditionTreeParser. Die Klasse ConditionTreeParser wurde in der Phase 3 so angepasst, dass sie aus dem Syntaxbaum, der durch den ConditionParser erzeugt wird, Instanzen des Bedingungsmodells erzeugt und eine Referenz auf diese Bedingung als Rückgabewert einer ihrer Methoden zurückgibt.
Instanzen der erzeugten Klassen werden während der semantischen Überprüfung und während der Konvertierungsphase einer Prozessdefinition erzeugt: Eine Instanz der Lexer-Klasse ConditionLexer liest die Bedingungsausdrücke ein und generiert Token-Objekte, die an eine Instanz der Parser-Klasse ConditionParser weitergeleitet werden. Diese Parser-Klasse überprüft, ob diese Folge von Token-Objekten eine korrekt definierte Bedingung entsprechend der oben definierten Grammatik darstellt und stellt einen Syntaxbaum auf. In der semantischen Überprüfung wird die TreeParser-Klasse ConditionTreeParser nicht benötigt, da nur die Syntax des Bedingungsausdrucks überprüft wird. Die Instanz der Klasse ConditionParser liefert eine Fehlermeldung, wenn dies nicht der Fall ist. In diesem Falle ist die Bedingung semantisch nicht korrekt.
In der Konvertierungsphase wird der generierte Syntaxbaum einer Instanz der TreeParser-Klasse übergeben, die dann die entsprechenden Instanzen des Bedingungsmodells erzeugt und eine Referenz auf die generierte Bedingung zurückliefert.

Der in dieser Arbeit entwickelte Parser hat also die Aufgabe, Bedingungsausdrücke gegen die oben präsentierte Grammatik zu prüfen und aus korrekten Bedingungsausdrücken eine Bedingung im Sinne des Bedingungsmodells aus [Gic03] zu erzeugen, die der textuellen Repräsentation entspricht und bei der Ausführung von Prozessinstanzen evaluiert werden kann.

# Kapitel 8

# Prozessverwaltung

Die Anforderungen an die Prozessverwaltung wurden bereits im Kapitel 5.4 zusammengetragen. In diesem Kapitel soll die Realisierung dieser Anforderungen für das Prozessunterstützungssystem erläutert werden. Im Abschnitt 8.1 wird erläutert, wie das Prozessmodell erweitert und eine Benutzerschnittstelle realisiert wird, damit ein Administrator laufende Prozesse mit den in Kapitel 5.4 präsentierten Anwendungsfällen administrieren und Prozessdefinitionen konfigurieren kann. Der Abschnitt 8.2 erläutert, wie ein einfacher Monitoring-Dienst für das Prozessunterstützungssystem realisiert wird. Der Monitoring-Dienst informiert einen Prozessteilnehmer über den aktuellen Prozessfortschritt. Die Anwendungsfälle für den Monitoring-Dienst sind ebenfalls in Kapitel 5.4 beschrieben.

## 8.1 Administration von Prozessen und Prozessdefinitionen

In diesem Abschnitt wird die Realisierung einer Benutzerschnittstelle besprochen, die von Administratoren benutzt werden kann, um laufende Prozesse zu verwalten und Prozessdefinitionen zu konfigurieren.

### 8.1.1 Administration von Prozessen

In diesem Unterabschnitt wird die Realisierung des Teils der Benutzerschnittstelle erläutert, der es einem Administrator des Informationsportals erlaubt, von der Prozessmaschine ausgeführte Prozesse zu administrieren. Dies umfasst die folgenden schon im Abschnitt 5.4 präsentierten Anwendungsfälle:

- Prozess anhalten,
- Prozess fortsetzen,
- Prozess abbrechen,
- Prozess neu starten,
- Prozessfortschritt einsehen,
- Aktuell ausgeführte Aktivität beenden (und zur nächsten springen),
- Manuelle Aktivitäten anderen Personen oder Rollen zuweisen und

- Prozesskontext bearbeiten.

Um die Benutzerschnittstelle im Portal zur realisieren, werden Handler und Templates eingesetzt, die die Interaktion mit einem Benutzer (Administrator) übernehmen (siehe hierzu auch Kapitel 2). Für die Benutzerschnittstelle wurden zwei Templates und mehrere Handler entworfen, die die oben aufgelisteten Anwendungsfälle realisieren. Die Idee war dabei, diese genannten Anwendungsfälle sowohl für den gesamten Prozess als auch für die einzelnen Prozessthreads zu realisieren. Auf diese Weise kann dem Administrator eine feinere Granularität bei der Verwaltung der Prozessinstanzen zur Verfügung gestellt werden. Bei Prozessen, die in mehreren Prozessthreads ablaufen, muss nicht immer der gesamte Prozess bei Problemen abgebrochen werden, sondern es kann der betreffende Prozessthread abgebrochen und neu gestartet werden. Die Handler, die die Funktionalität für diese Anwendungsfälle realisieren, müssen auf Methoden des Prozessmodells zugreifen können. Um diese Verwaltungsfunktionalität im Prozessmodell zu unterstützen, wurden Klassen des Prozessmodells um zusätzliche Methoden erweitert. Die zusätzlichen Methoden sind in Abbildung 8.1 gezeigt. Diese neuen Methoden werden durch Handler, die für die Administrator-Benutzerschnittstelle realisiert wurden, aufgerufen.

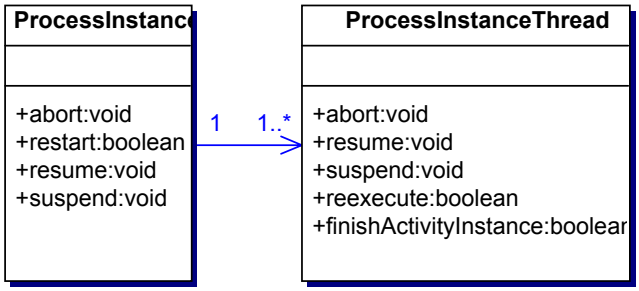


Abbildung 8.1 Realisierte Methoden zur Verwaltung von Prozessen

Die hinzugefügten Methoden werden nun kurz beschrieben.

**ProcessInstance**

- abort(): Diese Methode bricht die Ausführung der Prozessinstanz ab, an der sie aufgerufen wird. Dazu wird an allen Prozessthreads, die diese Prozessinstanz ausführen, die Methode abort() aufgerufen.
- restart(): Der Aufruf dieser Methode startet die Ausführung einer Prozessinstanz neu. Dazu wird an allen Prozessthreads, die diese Prozessinstanz ausführen, die Methode abort() aufgerufen. Anschließend werden die abgebrochenen Prozessthreads gelöscht und die Ausführung startet mit der durch die Prozessdefinition festgelegten Startaktivität neu. Wenn der Prozess in der Prozessmaschine neu gestartet werden konnte, gibt die Methode true als Rückgabewert, anderenfalls false.
- resume(): Der Aufruf dieser Methode setzt die Ausführung einer angehaltenen Prozessinstanz fort. Dazu wird an allen angehaltenen Prozessthreads, die diese Prozessinstanz ausführen, die Methode resume() aufgerufen.
- suspend(): Diese Methode hält die Ausführung einer Prozessinstanz an. Dazu wird an allen aktiven Prozessthreads, die diese Prozessinstanz ausführen, die Methode suspend() aufgerufen.

Alle Methoden werfen eine Ausnahme (StateChangeNotPossibleException), wenn die Zustandsänderung nicht durchgeführt werden kann. Es sind nur solche Zustandsänderungen erlaubt, die durch das Zustandsdiagramm in Abbildung 3.4 beschrieben sind.

**ProcessInstanceThread**

- abort(): Der Aufruf dieser Methode bricht die Ausführung des Prozessthreads ab, an der sie aufgerufen wird. Dazu wird der Zustand des Threads auf „ABORT“ gesetzt. Der Zustand wird bei jedem Schleifendurchlauf der Prozessmaschine überprüft (vgl. auch Kapitel 6.5). Die Prozessmaschine erkennt den neuen Zustand und bricht den zugehörigen ProcessEngineThread ab. Threads, die sich in einem Schlaf-Zustand befinden, werden mit der interrupt()-Methode geweckt. Damit alle Threads auf diese Weise beendet werden können, muss in den handleRequest()-Methoden der Handler, die automatische Aktivitäten ausführen, der Thread-Zustand abgefragt werden (siehe hierzu auch Kapitel 9).
- suspend(): Diese Methode funktioniert nach demselben Prinzip wie die abort()-Methode. Der Unterschied liegt darin, dass der Zustand des Threads auf „HOLD“ gesetzt wird und die Information über die gerade ausgeführte Aktivität nicht gelöscht wird.
- resume(): Diese Methode setzt einen mit der suspend()-Methode angehaltenen Prozessthread fort. Dazu wird die Ausführung mit der Aktivität, die zum Zeitpunkt des Anhaltens ausgeführt wurde, durch die Prozessmaschine weitergeführt. Mit der Ausführung der Aktivität wird dabei von vorne begonnen.
- reexecute(): Diese Methode wird verwendet, um eine Aktivität die durch einen Prozessthread ausgeführt wurde, nochmals auszuführen. Voraussetzung ist, dass der Thread während der Ausführung durch den Administrator abgebrochen wurde oder durch einen Fehler in der Aktivitätsausführung automatisch abgebrochen wurde. Der Thread wird in der Prozessmaschine reaktiviert und die Aktivität neu gestartet. Wenn die Voraussetzungen nicht erfüllt sind, gibt die Methode false als Rückgabewert zurück, wenn die Aktivität neu gestartet werden konnte, dann gibt sie true zurück.
- finishActivityInstance(): Wird diese Methode an einem Prozessthread aufgerufen, dessen Ausführung angehalten wurde, so wird die Aktivität, die zuletzt bearbeitet wurde, für beendet erklärt und die nächste Aktivität bestimmt, falls diese existiert. Anschließend wird die Ausführung mit dieser nächsten Aktivität fortgesetzt. Kann keine nächste Aktivität bestimmt werden, oder wurde der Thread vor dem Aufruf der Methode nicht angehalten, so gibt die Methode false als Rückgabewert. Konnte die nächste Aktivität bestimmt und die Ausführung wieder gestartet werden, so gibt die Methode true als Rückgabewert.

Die Methoden abort(), suspend() und resume() liefern die Ausnahme StateChangeNotPossibleException, falls eine Zustandsänderung nicht erlaubt ist, wobei wieder das Zustandsdiagramm aus Abbildung 3.4 zugrunde gelegt wird.
Die folgenden Abbildungen zeigen die beiden entworfenen Templates, die ein Administrator des Portals nutzen kann, um die Prozesse zu administrieren. Die zusammen mit diesen Templates entworfenen Handler nutzen die oben vorgestellten Methoden und realisieren so die Anwendungsfälle.
Die Abbildung 8.2 zeigt die Seite zum Administrieren von Prozessinstanzen. Im oberen Teil werden dem Administrator Informationen zum aktuellen Prozess und dessen Zustand angezeigt. Dann folgt eine Liste der an diesem Prozess beteiligten Prozessthreads. Auf dem Screenshot wird der Konvertierungsprozess aus Kapitel 7 gezeigt, der aus nur einem Prozessthread besteht. In der Liste werden weitere Informationen zum jeweiligen Thread präsentiert. Über die drei Links, die links unten gezeigt sind, kann der Prozess angehalten, abgebrochen, oder neu gestartet werden. Der vierte Link zum Fortsetzen eines angehaltenen Prozesses wird hier verborgen, da es sich um einen aktiven Prozess handelt. Über den Link „[>bearbeiten]“ gelangt der Administrator zur Administration eines einzelnen Prozessthreads. Die Seite, die dann präsentiert wird, ist Abbildung 8.3 zu sehen.

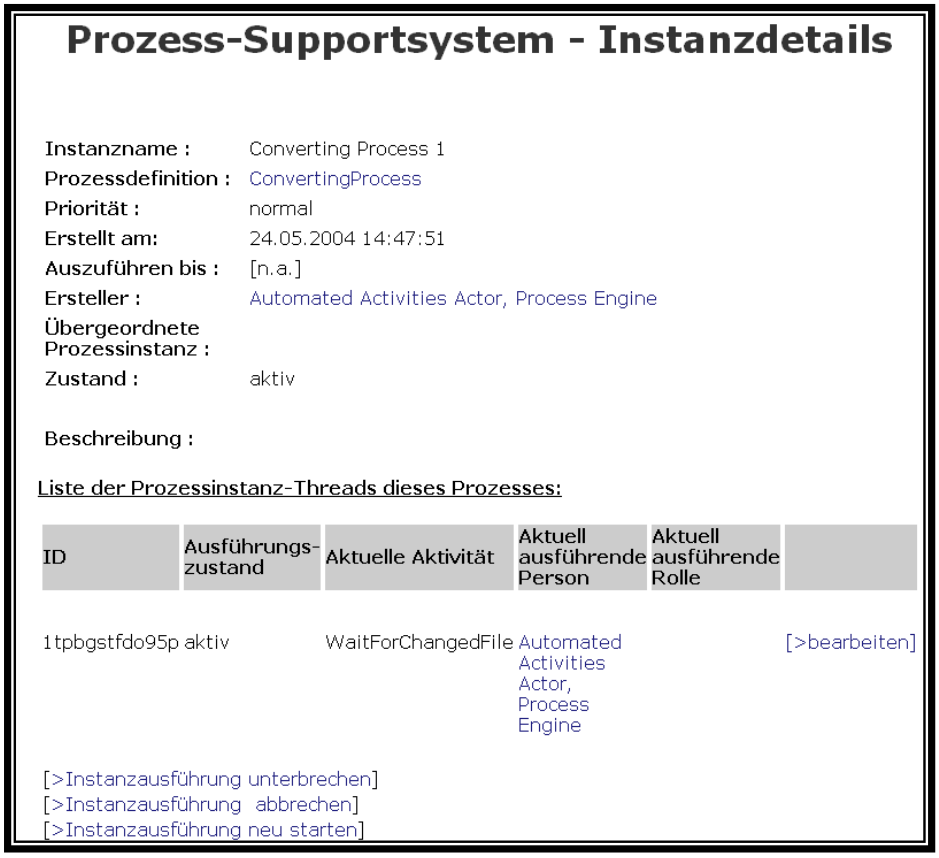


Abbildung 8.2 Screenshot - Prozessinstanzdetails

Die Seite zur Administration eines einzelnen Prozessthreads (Abbildung 8.3) zeigt in der oberen Hälfte Informationen zum Prozessthread und Zustand. Anschließend folgen die gerade ausgeführte Aktivität und der Zustand der zugeordneten Aktivitäts-Instanz. Darunter ist die Liste der bereits durch den Thread ausgeführten Aktivitäten zu sehen, wobei in Abbildung 8.3 der Thread noch keine Aktivitäten ausgeführt hat. Unter der Liste werden die Links zum Administrieren des Prozessthreads angezeigt. Da der Thread in Abbildung 8.3 angehalten ist, ist hier der Link zum Fortsetzen des Threads zu sehen. Auf der rechten Seite befindet sich der Link, um die aktuell ausgeführte Aktivität zu beenden („[>Instanz beenden]“). Eine Besonderheit stellt der Link „[>Thread unsicher beenden]“ dar. Dieser Link weist das Prozessunterstützungssystem an, den ProcessEngineThread, der diesem Prozessthread zugeordnet ist, direkt zu beenden. Da diese Vorgehensweise in Java für Threads jedoch nicht empfohlen wird, sollte dieser Link nur benutzt werden, wenn der Thread sich nicht mit dem Link „[>Thread beenden]“ beenden lässt. Dieser Link führt dazu, dass ein Flag gesetzt wird, wodurch sich der Thread selbst beenden sollte (siehe auch Kapitel 9).
Ein Anwendungsfall konnte in diesem Zusammenhang nicht berücksichtigt werden: Die Bearbeitung des Prozesskontextes. Dieser Anwendungsfall wurde aus denselben Gründen, die schon in Kapitel 6 angeführt wurden, nicht realisiert. Hierzu wären zunächst eine nähere Analyse des Prozesskontextes und der Entwurf einer Strategie zum Bearbeiten des Prozesskontextes notwendig, die im Rahmen dieser Arbeit nicht geleistet werden konnten.


Abbildung 8.3 Screenshot – Administration eines Prozessthreads

## 8.1.2 Konfiguration von Prozessdefinitionen

In diesem Abschnitt soll beschrieben werden, wie die Anwendungsfälle zur Konfiguration der Prozessdefinition realisiert wurden. Unter Konfiguration der Prozessdefinition fallen die Anwendungsfälle zur Konfiguration des Prozessstarts. Dazu zählen die folgenden Anwendungsfälle (vgl. Kapitel 5.4):

- Prozessstart freigeben
- Prozessstart verbieten
- Anzahl parallel laufender Prozesse einer Prozessdefinition wählen
- Startart wählen

**Benutzerschnittstelle**

Damit der Administrator diese Funktionen nutzen kann, wurde ein Dialog implementiert. Der Dialog wird wieder mit dem Template/Handler-Konzept realisiert, wie im Abschnitt 8.2.1. Die Abbildung 8.4 zeigt die Seite zur Konfiguration der Prozessdefinition, welche zu diesem Zweck realisiert wurde.


Abbildung 8.5 Screenshot – Konfiguration der Prozessdefinition

Im oberen Bereich werden dem Administrator Informationen zur Prozessdefinition angezeigt. Im Einzelnen sind das der Name und die Gruppe der Prozessdefinition, der Autor (im Beispiel ist dieser nicht bekannt), der zugeordnete Administrator und eine Beschreibung der Prozessdefinition. Danach folgen die Angabe der XMI-Datei, aus der die Prozessdefinition konvertiert wurde, sowie die Startaktivität und die Anzahl der momentan laufenden Instanzen dieser Prozessdefinition.

An den Informationsbereich schließt sich der Konfigurationsbereich an. Hier kann der Administrator den Prozessstart konfigurieren sowie den zuständigen Administrator definieren. Die Konfiguration des Prozessstarts setzt sich aus einem allgemeinen Teil und einem Teil zur Konfiguration des Prozessstarters zusammen. Im allgemeinen Teil kann der Prozessstart erlaubt oder verboten werden. Dabei wird zwischen Start als Subprozess oder als eigenständiger Prozess unterschieden. Der Prozessstart kann daher so konfiguriert werden, dass ein Start einer Instanz des Prozesses verboten ist, dass der Start sowohl als eigenständiger als auch Subprozess erlaubt ist,

oder jeweils nur als eigenständiger oder als Subprozess erlaubt ist. Des Weiteren kann im allgemeinen Teil die Anzahl der maximal gleichzeitig laufenden Instanzen eines Prozesses festgelegt werden. Versucht ein Benutzer oder eine Systemkomponente eine Prozessinstanz einer Prozessdefinition zu starten, während die maximale Zahl an Instanzen erreicht ist, so verweigert die Prozessmaschine den Start (vgl. auch Kapitel 6.5).
An den allgemeinen Teil schließt sich die Konfiguration des Prozessstarters an. Der Prozessstarter ist eine Softwarekomponente, die die Aufgabe hat, den Start eines Prozesses zu steuern. Der Prozessstarter wurde zu diesem Zweck in dieser Arbeit realisiert. Der Prozessstarter wird weiter unten noch näher erläutert. Im Wesentlichen besteht die Konfiguration des Prozessstarters aus der Angabe des Starttyps und eventuell der Rolle oder des Ereignistyps.
Es werden zwei Starttypen unterstützt:

1. automatisch: Der Prozessstarter startet solange Prozessinstanzen einer Prozessdefinition, bis die maximale Anzahl gleichzeitig laufender Prozesse erreicht ist. Ein Beispiel für einen solchen Prozess ist der Konvertierungsprozess. Die maximale Anzahl gleichzeitig ausgeführter Instanzen für diesen Prozess ist eins. Wenn also eine Datei konvertiert wurde und der Prozess danach beendet wurde, startet der Prozessstarter für diesen Prozess eine neue Instanz, so dass die nächste Datei konvertiert werden kann.
2. manuell/ereignisgesteuert: Bei diesem Starttyp kann der Prozess sowohl durch ein Ereignis, dessen Ereignistyp bei der Konfiguration des Prozessstarters angegeben wird als auch manuell durch einen Benutzer gestartet werden, wobei die Rolle der Benutzer, die zum Starten des Prozesses berechtigt ist, ebenfalls angegeben werden muss. Der Prozessstarter startet einen Prozess, wenn ein Ereignis des angegebenen Typs auftritt. Beim manuellen Prozessstart wird die Rolle des Benutzers mit der in der Konfiguration angegebenen Rolle verglichen. Stimmen die Rollen überein, startet der Prozessstarter den Prozess. Ein Beispiel für diesen Starttyp ist der WIPS-Prozess (vgl. Kapitel 9.2), dessen Konfigurationsseite in Abbildung 8.4 zu sehen ist. Als Ereignistyp ist dabei „WIPSRequestGeneratedEvent“ angegeben.

Über den Link „<Änderungen übernehmen>“ kann der Administrator die Einstellungen speichern. Dieser Link kann nur durch einen Administrator des Portals aufgerufen werden. Die Einstellungen sind aber für alle Benutzer sichtbar.
Am Ende dieser Seite befinden sich noch die Versionsinformationen. Hier werden die Namen aller Vorgänger und Nachfolgerversionen angezeigt. Der in Abbildung 8.4 gezeigte Prozess hat allerdings keine Vorgänger- und Nachfolgerversionen.

**Änderungen der Klasse ProcessDefinition**

Um die Einstellungen, die zum Prozessstart gemacht werden können, persistent im Portal speichern zu können, wurde die Klasse ProcessDefinition um Attribute und Methoden zum Zugriff auf diese Attribute erweitert. Instanzen dieser Klasse repräsentieren eine Prozessdefinition. Die Handler, die im Rahmen der Benutzerschnittstelle realisiert worden sind, rufen diese Methoden zum Auslesen und Speichern der Konfigurationseinstellungen auf. Das folgende Klassendiagramm in Abbildung 8.5 zeigt die neu implementierten Methoden, sowie die Klasse ProcessStarter, die weiter unten noch erläutert wird.
Zu jedem Attribut gibt es eine get/set-Methode, wobei hier nur die set-Methoden erläutert werden sollen:

- setAdministrator(Person p): Setzt die Person p als Administrator der Prozessdefinition fest, falls p ein Administrator ist.
- setStartingType(int startingType): Definiert den Starttyp der Prozessdefinition, der als Integer kodiert übergeben wird (Kodierung : 1 = automatischer Start, 0 = manueller/ereignisgesteuerter Start).
- setStartingEvent(String eventTypeName): Setzt den Namen des Ereignistypen zum ereignisgesteuerten Start, der als eventTypeName übergeben wird, fest.
- setStartingRole(Group g): Setzt die als g übergebene Rolle (Rollen werden in der infoAssetBroker-Plattform durch die Klasse Group realisiert) als Rolle für den manuellen Start fest.

- setMaxConcurrentInstances(int max): Setzt die Anzahl der maximal gleichzeitig laufenden Prozessinstanzen für diese Prozessdefinition.
- updateProcessStarter(): Ruft die Methode update() der Instanz der Klasse ProcessStarter auf, die dieser Prozessdefinition zugeordnet ist. Diese Methode wird aufgerufen, wenn Einstellungen zum Prozessstart mit der oben beschriebenen Benutzerschnittstelle für eine Prozessdefinition geändert wurden. Der zugeordnete Prozessstarter übernimmt daraufhin die geänderten Werte.

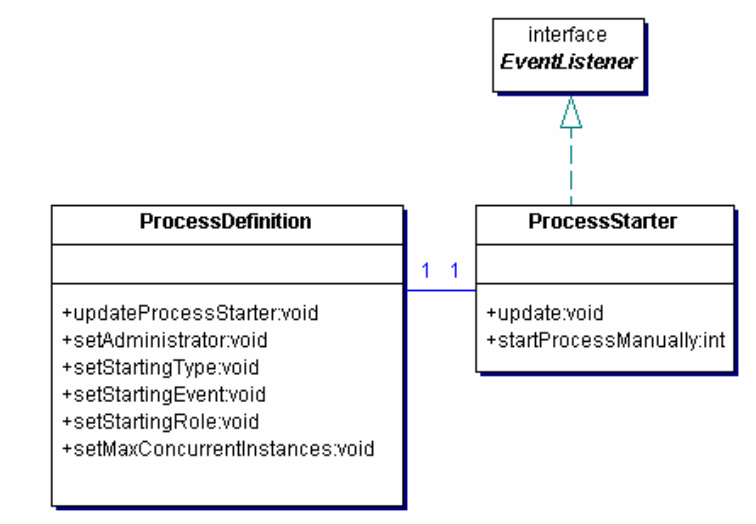


Abbildung 8.5 Änderungen der Klasse ProcessDefinition

**Die Klasse ProcessStarter**

Die Ausführung des Startvorganges wird durch die dafür bestimmte Klasse ProcessStarter kontrolliert. Jeder Prozessdefinition ist eine Instanz dieser Klasse zugeordnet. Diese Instanz, die als Prozessstarter bezeichnet wird, wird durch die Methode update() zur Aktualisierung der Startparameter aufgefordert. Der Prozessstarter liest dann die Startparameter der Prozessdefinition ein und konfiguriert sich daraufhin selbst. Sind die Voraussetzungen für einen Prozessstart erfüllt, so generiert der Prozessstarter ein Ereignis vom Typ „*StartProcess*“ für die zugeordnete Prozessdefinition und löst es aus. Die Prozessmaschine reagiert auf dieses Ereignis und startet den Prozess, falls alle Voraussetzungen erfüllt sind (siehe Kapitel 6.4 und 6.5). Die Prozessmaschine wurde zu diesem Zweck so erweitert, dass sie prüft, ob das Startereignis vom richtigen Prozessstarter ausgelöst wurde, nämlich dem Prozessstarter, der der Prozessdefinition des zu startenden Prozesses zugeordnet ist. Nur wenn dies der Fall ist, kann der Prozess gestartet werden, anderenfalls nicht. Der Prozessstarter steuert den Start eines Prozesses entweder aktiv oder passiv:

- Aktiv: Eine aktive Steuerung findet statt, wenn als Starttyp der zugeordneten Prozessdefinition der automatische Start festgelegt wurde. In diesem Falle startet der Prozessstarter einen Thread, der in regelmäßigen Abständen die Anzahl der laufenden Prozessinstanzen der zugeordneten Prozessdefinition ermittelt und mit der Anzahl der maximal gleichzeitig ausgeführten Instanzen vergleicht. Kann noch eine Instanz gestartet werden, ohne dass die maximale Anzahl überschritten wird, so wird der Prozessstart durch das Auslösen des Ereignisses vom Type „*StartProcess*“ veranlasst. Der Thread ist in der inneren Klasse ProcessStarterThread der Klasse ProcessStarter implementiert.
- Passiv: Die passive Steuerung verwendet der Prozessstarter, wenn als Starttyp der manuelle oder ereignisgesteuerte Start gewählt wurde. Im Falle des manuellen Starts werden nur die Rollen, deren Mitglied der Benutzer ist, mit der Rolle, die in der Prozessdefinition gespeichert ist, verglichen. Wird für eine Rolle eine Übereinstimmung gefunden, so wird der Prozessstart veranlasst. Ein Prozess kann manuell gestartet werden, indem die Methode startManually() der zu startenden Prozessdefinition aufgerufen

wird. Diese Methode ruft dann die Methode startProcessManually() am Prozessstarter auf. Ist dagegen die Prozessdefinition auf ereignisgesteuerten Start konfiguriert, so registriert sich der Prozessstarter als EventListener-Objekt beim EventDispatcher für den in der Prozessdefinition gespeicherten Ereignistypen (In Abbildung 8.4 ist dies der Ereignistyp „*WIPSRequestGeneratedEvent*“). Der Prozessstarter wird dann vom EventDispatcher beim Eintritt des Ereignisses benachrichtigt, woraufhin der Prozessstart veranlasst wird.

In den beiden Abschnitten 8.1.1 und 8.1.2 wurde beschrieben, wie die Realisierung der Verwaltung von laufenden Prozessen und die Konfiguration des Prozessstarts mit einer Benutzerschnittstelle durch einen Administrator umgesetzt wurde und welche Erweiterungen dazu an Klassen des Prozessmodells notwendig waren.

## 8.2 Der Monitoring-Dienst

Dieser Abschnitt beschreibt die Realisierung des Monitoring-Dienstes, für den im Kapitel 5.4 die Anforderungen definiert wurden. Generell soll es der Monitoring-Dienst einem Prozessteilnehmer erlauben, den Fortschritt eines Prozesses zu verfolgen. Die Anwendungsfälle für die Prozessverfolgung sind in Kapitel 5.4 vorgestellt worden.
Als Präsentationsart wird die grafische Präsentation gewählt, da im Allgemeinen grafische Präsentationen einfacher für Benutzer zu verstehen sind. Dabei wird in dieser Arbeit eine Idee aus [ZHMS01] verfolgt, nach der Pläne, die U-Bahn-Netzplänen ähneln, zur grafischen Präsentation des Kontrollflusses genutzt werden. Ein Haltestellen-Symbol in einem solchen Plan repräsentiert danach eine Aktivität. Die Vebindungslinien zwischen den Haltestellen repräsentieren die verschiedenen Transitionen.
Nachdem nun Präsentationsart und –metaphern gewählt sind, stellt sich die Frage, wie die Präsentation technisch realisiert werden soll. Da der Prozessteilnehmer, während er manuelle Aktivitäten ausführt, mit Webseiten interagiert, bietet sich als Realisierungstechnik ein Applet an, das in die Webseiten integriert ist. Diese Technik wird auch in dieser Arbeit zur Realisierung des Monitoring-Dienstes verwendet.

**Applet**
Ein Applet ist ein Java-Programm, das über spezielle HTML-Tags in eine Webseite eingebunden werden kann. Ein Webbrowser lädt die Klassendefinition zur Ausführung des Applets vom Webserver herunter, wenn er eine HTML-Seite mit einem eingebundenen Applet anzeigen soll. Anschließend wird der Bytecode in der Virtuellen Maschine des Browser-Herstellers, oder eines Plug-Ins zur Ausführung gebracht. Die Benutzeroberfläche des Applets wird im Browser-Fenster innerhalb der Webseite angezeigt. Java Applets sind Sicherheitsbeschränkungen unterworfen, z.B. beim Zugriff auf lokale Ressourcen. Diese betreffen aber nicht die grafische Benutzeroberfläche, weshalb Applets gut zur grafischen Präsentation eingesetzt werden können. Das in dieser Arbeit entworfene Applet zur grafischen Präsentation des Prozessfortschritts ist als MonitorApplet implementiert worden. Dem Prozessteilnehmer soll das Applet während der Ausführung von manuellen Aktivitäten den aktuellen Prozessfortschritt anzeigen. Um den benötigten Platz bei der Anzeige des Applets klein zu halten, werden einem Prozessteilnehmer nur die gerade von ihm bearbeitete Aktivität, die unmittelbar vorangegangenen Aktivitäten, sowie die unmittelbar folgenden Aktivitäten angezeigt. Da dieses Applet für alle im Portal ausgeführten Prozesse eingesetzt werden soll, muss zur Laufzeit dem Applet mitgeteilt werden, welche Aktivitäten anzuzeigen sind. Zu diesem Zweck wird JavaScript eingesetzt. Mit JavaScript ist es möglich, öffentliche Methoden eines Applets einer Webseite aufzurufen. Der JavaScript-Code wird beim Laden der Webseite durch einen Web-Browser interpretiert und die entsprechenden Anweisungen, wie z.B. der Aufruf öffentlicher Methoden eines Applets, ausgeführt. Die öffentliche Methode setActualActivityInstanceId(String id) des Applets ist für diesen Zweck definiert. Sie übergibt eine AssetId an das Applet. Diese AsssetId sollte die AssetId der ActivityInstance sein, die die Ausführung der aktuellen Aktivität repräsentiert.

Damit die AssetId mit JavaScript an das Applet übergeben werden kann, muss die Webseite, während sie im Browser geladen wird, eine JavaScript-Funktion ausführen, die die AssetId an das Applet übergibt. Zuvor wird ein Platzhalter im Template der Webseite von einem Handler durch die AssetId ersetzt. Welche JavaScript-Funktionen aufgerufen werden, hängt von der Einbindung des Applets in die Portalseiten ab. Im folgenden Beispiel-Code wird eine konkrete Einbindung mit zwei JavaScript-Funktionen präsentiert:

```
function updateMonitorApplet() {
  parent.iABnavigatorFrame.setActualActivityInstanceId("$ainstanceId$");
}

function invalidateMonitorApplet() {
  parent.iABnavigatorFrame.setActualActivityInstanceId("");
}
```

Dieser JavaScript-Code wird dem Template, welches das MonitorApplet aktualisieren will, hinzugefügt. Die Funktion updateMonitorApplet() ruft eine Methode des Frames auf, in den das Applet eingebettet ist. Dieser Methoden-Aufruf bewirkt, dass der Platzhalter *$ainstanceId$* als AssetId an das Applet übergeben wird. Dieser Platzhalter muss durch den dem Template zugeordneten Handler mit einer gültigen AssetId substituiert werden, bevor das Template im Webbrowser angezeigt wird. Die Methode invalidateMonitorApplet() bewirkt ein Überschreiben der AssetId im Applet mit dem leeren String, wodurch die grafische Präsentation gestoppt wird. Zum Aktualisieren wird in das Template eine Anweisung eingefügt, die beim Laden der Seite dazu führt, dass der Webbrowser die entsprechende Methode updateMonitorApplet() des Templates zum Aktualisieren aufruft. Beim Verlassen der Webseite durch den Prozessteilnehmer wird die Methode invalidateMonitorApplet() aufgerufen.
Die AssetId der aktuellen Aktivität reicht nicht aus, um den Prozessfortschritt grafisch dar zustellen. Deshalb muss das Applet nach dem Übergeben der AssetId die benötigten Informationen beim Portal abfragen. Die Informationen werden als serialisierter Bytestrom an das Applet übermittelt. Zu diesem Zweck wurde eine Serialisierungsstruktur entworfen, die in Abbildung 8.6 gezeigt wird. Die dort gezeigten Klassen implementieren das Serializable-Interface. Ihre Instanzen können deshalb zur Laufzeit von der Java Virtual Machine in einen Bytestrom serialisiert werden und vom Portal an das Applet übertragen werden. Die Klassen besitzen keine Methoden, da ihre Instanzen nur zur Übermittlung der in den Attributen gespeicherten Informationen verwendet werden.
Im Folgenden wird kurz erläutert, welche Klassen und Attribute welche Informationen repräsentieren.

ProcessInformation: Instanzen dieser Klasse werden zur Übermittlung von Informationen über einen ausgeführten Prozess verwendet. Die Attribute werden folgendermaßen zur Speicherung von Informationen verwendet:

- processDefinition: Name der Prozessdefinition des ausgeführten Prozesses
- processInstance: Name der Prozessinstanz
- processInstanceId: AssetId der Prozessinstanz
- state: String-Repräsentation des Zustandes der ausgeführten Instanz.

ActivityInstanceInformation: Instanzen dieser Klasse werden zur Übermittlung von Informationen über eine Aktivität verwendet. Die aktuell vom Prozessteilnehmer ausgeführte Aktivität wird dabei über die Referenz „*actual activity*“ von der ProcessInformation-Instanz identifiziert. Die unmittelbar davor ausgeführten Aktivitäten oder die umittelbar folgenden Aktivitäten werden über „*future activities*“ bzw. „*recent activities*“ referenziert .Die Attribute werden folgendermaßen zur Speicherung von Informationen verwendet:

- name: Name der Aktivität
- type: Typ der Aktivität als String
- processingState: Ausführungszustand der Aktivität als String
- executionDate: Ausführungszeitpunkt der Aktivität
- dueDate: Zeitpunkt, bis zu dem die Aktivität ausgeführt sein muss
- assignedState: Zuweisungszustand der Aktivität (Rolle oder Person zugewiesen)
- enactor: Person, die die Aktivität ausführt, oder ausgeführt hat.
- Description: Beschreibung der Aktivität
- joinCondition: Synchronisationsbedingung bei Join-Aktivitäten
- numberOfReachedThreads: Anzahl der synchronisierten Prozessthreads bei Join-Aktivitäten

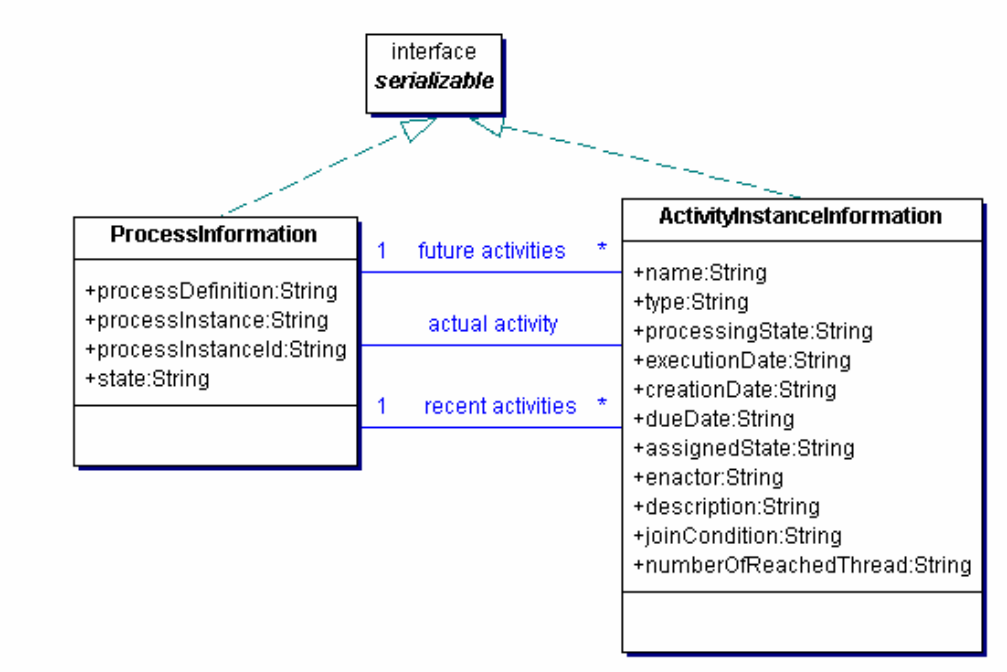


Abbildung 8.6 Serialisierungsstruktur

Zur Übermittlung des serialisierten Bytestroms zwischen dem Portalserver und dem Applet wird das HTTP-Protokoll verwendet. Das Applet fragt beim Portalserver nach einer virtuellen Ressource, indem es eine URL an den Portalserver schickt, die dem folgenden Muster entspricht:

```
http://[host]:[port]/[path]?[param]=[assetId]
```

Die dabei verwendeten Platzhalter (dargestellt in eckigen Klammern) haben folgende Bedeutung:

- host: Hostname oder IP des Portalservers. Das Applet kann diesen Parameter aus dem eigenen Kontext ermitteln.
- port: Port an den die URL gesendet werden soll. Muss als Applet-Parameter in der HTML-Seite, die das Applet einbindet, definiert werden.
- path: Anfragepfad. Muss als Applet-Parameter in der HTML-Seite, die das Applet einbindet, definiert werden.
- param: Name des Parameters der mit der Anfrage übermittelt werden soll. Muss als Applet-Parameter in der HTML-Seite, die das Applet einbindet, definiert werden.
- assetId: AssetId der aktuell ausgeführten Aktivität. Wird per JavaScript-Funktionsaufruf dem Applet beim Laden der Webseite im Browser mitgeteilt.

Applet-Parameter werden notiert als spezielle HTML-Tags, die bei der Einbindung eines Applets in eine HTML-Seite verwendet werden, um Parameter zu definieren, die dem Applet übergeben werden, wenn es im Browser-Fenster angezeigt wird. Durch die Verwendung von Applet-Parametern wird das Applet in diesem Fall dynamisch konfiguriert. So lässt sich zum Beispiel der Anfrageport einfach durch eine Änderung des HTML-Quelltextes der Webseite, die das Applet einbindet, wechseln, ohne dass das Applet neu kompiliert werden muss. Die Anfrage des Applets wird auf dem Portalserver durch eine Instanz der Klasse MonitorServlet bearbeitet.

**Die Klasse MonitorServlet**

Die Klasse MonitorServlet ist eine Subklasse von Servlets [Sun04b]. Servlets sind protokoll- und plattformunabhängige Java-Programme, die die Funktionalität eines Web-Servers erweitern. Sie bearbeiten Anfragen an den Web-Server und erzeugen eine Antwort. Servlets werden dynamisch durch eine ServletEngine geladen. Eine ServletEngine ist eine Serveranwendung, die Servlets ausführt, Client-Anfragen an ein angefragtes Servlet weiterleitet und die Antwort vom Servlet zurück an den Client sendet [Serv04].
In der infoAssetBroker-Plattform wird ein Tomcat-Webserver [Jak04] eingesetzt, um Servlets zu unterstützen. Ein Tomcat-Webserver ist eine OpenSource-ServletEngine besitzt. Er kann so konfiguriert werden, dass er abhängig vom Anfragepfad (s. o.) verschiedene Servlets aufruft.
Die Klasse MonitorServlet wurde realisiert, um die Anfragen des Applets zu beantworten. Dazu muss der Tomcat-Webserver so konfiguriert werden, dass der dem Applet als Parameter mitgeteilte Anfragepfad mit dem für Servlets der Klasse MonitorServlet konfigurierten Anfragepfad im Tomcat-Webserver übereinstimmt.

**Kommunikation zwischen Applet und MonitorServlet**

Ein Servlet der Klasse MonitorServlet bearbeitet eine Anfrage des Applets nach der virtuellen Ressource, indem es aus der Anfrage-URL, die dem oben präsentierten Muster entspricht, die AssetId ermittelt. Anschließend wird überprüft, ob es sich um eine gültige AssetId einer Aktivität des Prozessunterstützungssystems handelt. Ist das der Fall, so werden der Prozess, in dem die Aktivität ausgeführt wird, und die unmittelbar zuvor ausgeführten Aktivitäten sowie die nachfolgenden Aktivitäten bestimmt. Aus diesen Informationen werden Instanzen der oben gezeigten Serialisierungsstruktur erzeugt, die als Antwort auf die Anfrage des Applets serialisiert und in einem Bytestrom an das Applet zurückgesendet werden. Stellt die AssetId keine gültige AssetId einer Aktivität dar, so wird eine Instanz der Klasse ProcessInformation erzeugt und das Attribut *state* dieser Instanz auf den Wert „NO_VALID_ID“ gesetzt. Diese Instanz wird dann serialisiert und an das Applet zurückgesendet. Dadurch wird das Applet informiert, dass die übermittelte AssetId in der Anfrage keine gültige AssetId darstellt. Der übermittelte Bytestrom wird durch das Applet deserialisiert, so dass die zuvor serialisierten Instanzen im Applet als Objekt zur Verfügung stehen und die in den Attributen gespeicherten Informationen ausgelesen werden können. Diese Informationen werden durch das Applet zur grafischen Präsentation des Prozessfortschritts genutzt. Abbildung 8.7 zeigt einen Screenshot einer solchen Präsentation.


Abbildung 8.7 Screenshot – MonitorApplet

Die Darstellung stammt aus dem WIPS-Prozess, dessen Realisierung ein Teil dieser Arbeit war und im Kapitel 9.2 behandelt wird. Dem Prozessteilnehmer werden über der grafischen Präsentation die Informationen zum Prozess angezeigt. Dieses sind Zustand, der Name der Prozessdefinition und der Name der Prozessinstanz. Der

Name der Prozessinstanz ist anklickbar und bringt die Webseite mit den Prozessinstanzdetails zur Anzeige im Browser des Prozessteilnehmers (siehe Abbildung 8.2). Die Links zur Administration des Prozesses sind allerdings nur für Mitglieder der Administratoren-Gruppe sichtbar. Von hier kann der Prozessteilnehmer weiter zur Konfigurationsseite für die Prozessdefinition gelangen, um den Administrator der Prozessdefinition festzustellen. Dieses war eine der Anforderungen aus Kapitel 5.4.
Die grafische Präsentation ist so aufgebaut, dass in der Mitte die aktuelle Aktivität angezeigt wird. Mit dem Mauszeiger kann über die verschiedenen Aktivitäten gefahren werden, wobei die gerade ausgewählte Aktivität durch einen speziellen Kasten markiert wird. Befindet sich der Mauszeiger über einer anderen Aktivität als der gerade ausgewählten, so wird diese Aktivität mit einem Kasten markiert und gilt dann als ausgewählt. Am unteren Rand wird der Name der gerade ausgewählten Aktivität angezeigt.
Bei den verwendeten Grafiken handelt es sich um Linien, sowie Haltestellen- bzw. Aktivitäts-Symbole. Nicht alle Aktivitäten werden als Haltestellen-Symbol, welches in Abbildung 8.7 durch den Kasten markiert ist, präsentiert. Fork- und Join-Aktivitäten werden durch ein Verzweigungssymbol dargestellt. In Abbildung 8.7 ist die linke Aktivität eine Fork-Aktivität. Das Symbol für die Join-Aktivität ist um 180° vertikal gedreht. Das Fragezeichen-Symbol (in Abbildung 8.7 rechts zu sehen) zeigt an, dass es sich bei der aktuellen Aktivität um eine Entscheidung handelt und erst nach Ende der Entscheidung feststeht, welche Aktivität auf die aktuelle folgt. Wird auf eine Aktivität im Applet mit der Maus geklickt, so öffnet sich ein Applet-Fenster, das weitere Informationen zur gewählten Aktivität anzeigt. Ein solches Applet-Fenster mit Informationen zu der markierten Aktivität in Abbildung 8.7 wird in Abbildung 8.8 gezeigt. Beim Anklicken des Fragezeichen-Symbols wird zunächst eine Liste der möglichen Nachfolger-Aktivitäten eingeblendet. Erst nach Auswahl einer dieser Aktivitäten in der Liste wird das Applet-Fenster angezeigt.

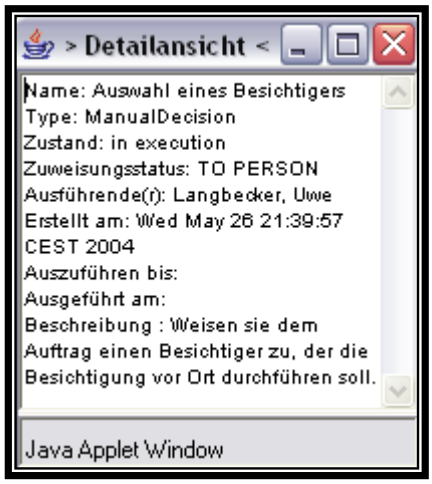


Abbildung 8.8 Applet-Fenster mit weiteren Informationen zu einer Aktivität

Die Informationen zu einem Prozess oder zu einer Aktivität können über das Applet abgefragt werden. Bei Bedarf können auch die detailierteren Seiten mit Informationen zur Prozessinstanz bzw. Prozessdefinition über das Applet aufgerufen werden. Damit sind alle Anforderungen aus Kapitel 5.4 an den Monitoring-Dienst realisiert worden.

# Kapitel 9

# Realisierung von Prozessen mit dem entwickelten Prozessunterstützungssystem

In den vorangegangenen Kapiteln wurde die Realisierung eines Prozessunterstützungssystems für die infoAssetBroker-Plattform beschrieben. In diesem Kapitel soll nun beschrieben werden, wie mit dem realisierten Prozessunterstützungssystem Prozesse definiert und umgesetzt werden können. Dies wird im Abschnitt 9.1 erläutert. Der Abschnitt 9.2 behandelt die Realisierung eines konkreten Prozesses mit dem Prozessunterstützungssystem. Dabei handelt es sich um den Schiffsbesichtigungs-Prozess, der in Kapitel 4.1 vorgestellt wurde.

## 9.1 Vorgehensweise zur Umsetzung eines Prozesses

Um mit dem realisierten Prozessunterstützungssystem Prozesse umzusetzen sind mehrere Schritte notwendig, die nun der Reihe nach vorgestellt werden. Dabei wird vorausgesetzt, dass das Prozessunterstützungssystem Teil eines mit der infoAssetBroker-Plattform realisierten Informationsportals ist.

1. Definieren des Prozesses: In der vorliegenden Arbeit wurde dazu das UML-Modellierungswerkzeug Poseidon der Firma Gentleware [Gen04] in die infoAssetBroker-Plattform integriert (vgl. Kapitel 7). Der Prozess wird mit Poseidon in einem UML-Aktivitätsdiagramm modelliert. Welche Modellierungselemente dabei eingesetzt werden können, ist in Kapitel 7 erläutert. Wichtig ist, dass bei allen manuellen Aktivitäten (dazu zählen auch manuelle Entscheidungen) eine Rolle oder Person definiert wird, der die Aktivität zur Ausführung zugewiesen wird. Wird keine Rolle oder Person definiert, so ist die manuelle Aktivität durch jeden Benutzer eines konkreten Portals ausführbar. Des Weiteren muss der Handlername für eine automatische bzw. manuelle Aktivität angegeben werden, damit das Prozessunterstützungssystem den Handler, der zur Ausführung einer Aktivität vorgesehen ist, bestimmen kann. Wird kein Handler angegeben, kann die Aktivität nicht ausgeführt werden. Die Angabe des Handlernamens ist für Join-, Fork- und Merge-Aktivitäten sowie für automatische Entscheidungen nicht erforderlich, da diese Aktivitäten direkt in der Prozessmaschine ausgeführt werden.

2. Entwicklung benötigter Handler und Templates: Wenn in dem Prozess Handler-Klassen oder Templates verwendet werden sollen, die noch nicht im Portal vorhanden sind, so müssen diese Handler-Klassen und Templates realisiert werden. Für automatische Aktivitäten werden Handler benötigt, die keine Ausgaben haben („*invisible*“, vgl. Kapitel 2.2) Für manuelle Aktivitäten werden Handler benutzt, die eine Webseite erzeugen. Zu diesem Zweck ist dort die Definition eines Templates notwendig. Wenn der Prozess das im letzten Kapitel beschriebene Applet nutzen soll, um den Prozessfortschritt anzuzeigen, dann müssen die Templates JavaScript-Funktionen enthalten, die das Applet steuern (vgl. Kapitel 8).

3. Die neu realisierten Handler-Klassen und Templates müssen in das Portal integriert werden. Zu diesem Zweck ist in der augenblicklichen Version der Portalplattform ein Neustart des Portalssystems erforderlich. Es ist aber problemlos möglich, die Portalplattform so zu erweitern, dass die Handler-Klassen und Templates dynamisch zur Laufzeit durch einen Administrator nachgeladen werden können. Dies ist jedoch nicht Teil dieser Arbeit. Genauere Erläuterungen zur Integration von Templates und Handler-Klassen sind in der Literatur zur infoAssetBroker-Portalplattform zu finden [Wegn00].

4. Der als UML-Aktivitätsdiagramm modellierte Prozess wird als XMI-Datei aus Poseidon exportiert und über die WebDav-Schnittstelle in das Portalsystem importiert und dort abgelegt. Die WebDav-Schnittstelle ist ein Teil der infoAssetBroker-Plattform, und deshalb können mit der Plattform realisierte Portale diese Schnittstelle anbieten.

5. Der Konvertierungsprozess wird über ein Ereignis von der neu im Portal abgelegten Prozessdefinition informiert und konvertiert diese dann automatisch. Falls die semantische Prüfung der Prozessdefinition fehlschlägt, wird der Autor der Prozessdefinition mit einer Email, die den Grund des Fehlers enthält, darüber informiert (vgl. Kapitel 7). Als Autor der Prozessdefinition wird dabei der Benutzer festgelegt, der die Prozessdefinition im Portal gespeichert hat.

6. Im Falle der erfolgreichen Konvertierung wird der Administrator des Systems mit einer automatischen Email darüber informiert, dass eine neu konvertierte Prozessdefinition im Prozessunterstützungssystem vorliegt. Dann ist es die Aufgabe des Administrators, den Prozessstart freizugeben und die Startkriterien für diesen Prozess festzulegen (vgl. Kapitel 8).

7. Der Prozess liegt nun im Portal vor und der Prozessstart ist freigegeben. Daher können nun Prozessinstanzen gestartet und ausgeführt werden.

## 9.2 Realisierung des Schiffsbesichtigungs-Prozesses

Als ein weiterer Teil dieser Arbeit soll mit dem entwickelten Prozessunterstützungssystem ein Prozess realisiert und zur Ausführung gebracht werden, um die Funktion des Prozessunterstützungssystems zu zeigen. Der Fokus liegt dabei nicht auf der genauen Modellierung eines Geschäftsprozesses, sondern auf der Realisierung und Ausführung eines Prozesses. Dazu wurde der Schiffsbesichtigungs-Prozess gewählt, der in Kapitel 4.1 vorgestellt wurde. Der Prozess soll im WIPS-Portal, welches mit der infoAssetBroker-Plattform realisiert wurde, definiert und ausgeführt werden. Dieses Kapitel beschäftigt sich alleine mit der Definition und Umsetzung des Prozesses. Dabei wurde die oben vorgestellte Vorgehensweise verwendet. Im Folgenden sollen nun einzelne Schritte genauer erläutert werden. Anschließend werden noch einige Screenshots präsentiert, die bei der Ausführung des Prozesses erstellt wurden.

### 9.2.1 Definition und Realisierung

Zur Definition des Prozesses wurde das Modellierungswerkzeug Poseidon verwendet. Das Aktivitätsdiagramm des Schiffsbesichtigungs-Prozesses ist in Abbildung D.2 gezeigt. Ferner wird im Kapitel 4.1 erläutert, dass der Prozess auch einen Subprozess besitzt (Besichtigungsdurchführung), der auf einem persönlichen Informationsportal eines Schiffsbesichtigers ausgeführt wird. Da einige der Anforderungen aus Kapitel 5, die aus dem Schiffsbesichtigungsprozess resultieren, im Prozessunterstützungssystem nicht realisiert werden konnten, wurde das Aktivitätsdiagramm angepasst. Es wurden die Aktivitäten, die nicht durch das Prozessunterstützungssystem ausgeführt werden sollen, aber zum besseren Verständnis des Diagramms modelliert sind („NotInSystem"-Modellierung), nicht berücksichtigt. Die mit Poseidon erstellten Diagramme des Schiffsbesichtigungsprozesses werden in den Abbildungen 9.1 und 9.2 gezeigt.
In der Abbildung 9.1 ist das Aktivitätsdiagramm des Hauptprozesses zur Schiffsbesichtigung gezeigt. In Abbildung D.2 sind zum Start des Subprozesses Signale modelliert worden. An ihre Stelle tritt die automatische Aktivität *Warten bis Besichtigung durchgeführt.* Aufgabe des zugeordneten Handlers ist es, den Subprozess zu starten und sein Ende abzuwarten. Außerdem wurde der Prozess verkürzt, um den Realisierungsaufwand geringer zu machen. Die fehlenden Aktivitäten am Ende sind reine Verwaltungsaktivitäten, die nicht unbedingt benötigt werden um die Funktion des Prozessunterstützungssystems zu zeigen.
Die Abbildung 9.2 zeigt das Aktivitätsdiagramm des Subprozesses zur Besichtigungsdurchführung. Der Vergleich mit dem Aktivitätsdiagramm aus Abbildung D.1 zeigt, dass auch hier einige Änderungen in der Modellierung vorgenommen wurden. Zum einen wurden die Aktivitäten zur Synchronisation mit dem zentralen Portal entfernt, da die Prozess-Synchronisation zwischen Portalen nicht in dieser Arbeit realisiert wird. Der Prozess wird also auf demselben Portalsystem ausgeführt wie auch der Hauptprozess. Zum anderen wurde der erste Teil aus Abbildung D.1 weggelassen, der es dem Schiffsbesichtiger ermöglicht, die Liste der Besichtigungspunkte zu kürzen. Die Aktivitäten im mobilen Ausführungszweig wurden zusammengefasst. Auch hier liegt der Grund in der Verringerung des Realisierungsaufwandes.
Für beide modellierten Prozesse wurden in Poseidon für die Aktivitäten Handlernamen und Rollen mit dem im Kapitel 7 vorgestellten Mechanismus definiert. Des Weiteren wurde zu allen modellierten Aktivitäten eine Beschreibung in Poseidon angegeben.
Für beide Prozesse wurden 30 Handler-Klassen und einige Templates realisiert und in das Portal integriert. Zum Teil wurden auch bereits existierende Templates wieder verwendet. Alle neu implementierten Handler-Klassen sind im Package de.tuhh.sts.processsupport.handler.WipsProcess enthalten. Die Übersicht über alle Klassen ist in der Anlage B zu finden. Eine genauere Realisierung des Prozesses, die auch in einem Unternehmen eingesetzt werden könnte, würde aber deutlich mehr Templates und Handler benötigen, da Benutzerinteraktion in einem größeren Umfang realisiert werden muss.
Alle an beiden Prozessen beteiligten Handler und Templates wurden so erweitert, dass sie das Applet des Monitoring-Dienstes aus Kapitel 8 unterstützen. Das Applet wird, wie in Kapitel 8 besprochen, über JavaScript angesprochen.
Die beiden mit Poseidon modellierten Prozesse wurden über die WebDav-Schnittstelle im Portalsystem abgelegt und durch den Konvertierungsprozess automatisch konvertiert. In der Rolle des Systemadministrators werden dann die beiden folgenden Prozesse wie folgt konfiguriert:

<u>Hauptprozess Schiffsbesichtigung (Name: WIPS.xmi)</u>

- Prozessstart : erlaubt
- Start als Subprozess : verboten
- Maximale Anzahl gleichzeitig ausgeführter Prozesse : beliebig
- Starttyp: ereignisgesteuert
- Typ des Startereignisses : WIPSRequestGeneratedEvent

Subprozess zur Besichtigungsdurchführung (Name: Inspection.xmi)

- Prozessstart : erlaubt
- Start als Subprozess : erlaubt
- Maximale Anzahl gleichzeitig ausgeführter Prozesse : beliebig
- Starttyp: ereignisgesteuert
- Typ des Startereignisses : WIPSNewInspectionEvent

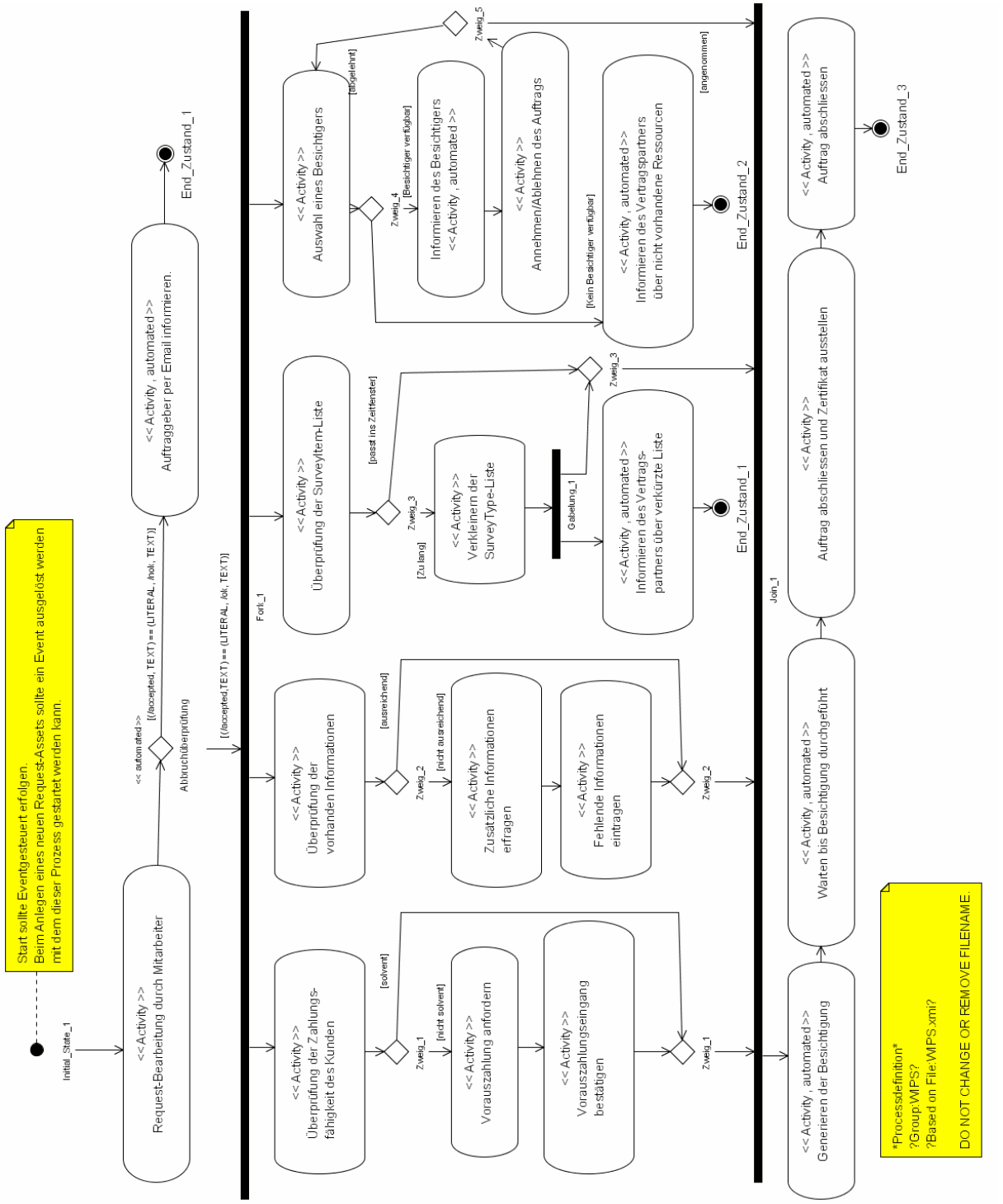


Abbildung 9.1 UML-Aktivitätsdiagramm des realisierten Schiffsbesichtigungs-Prozesses

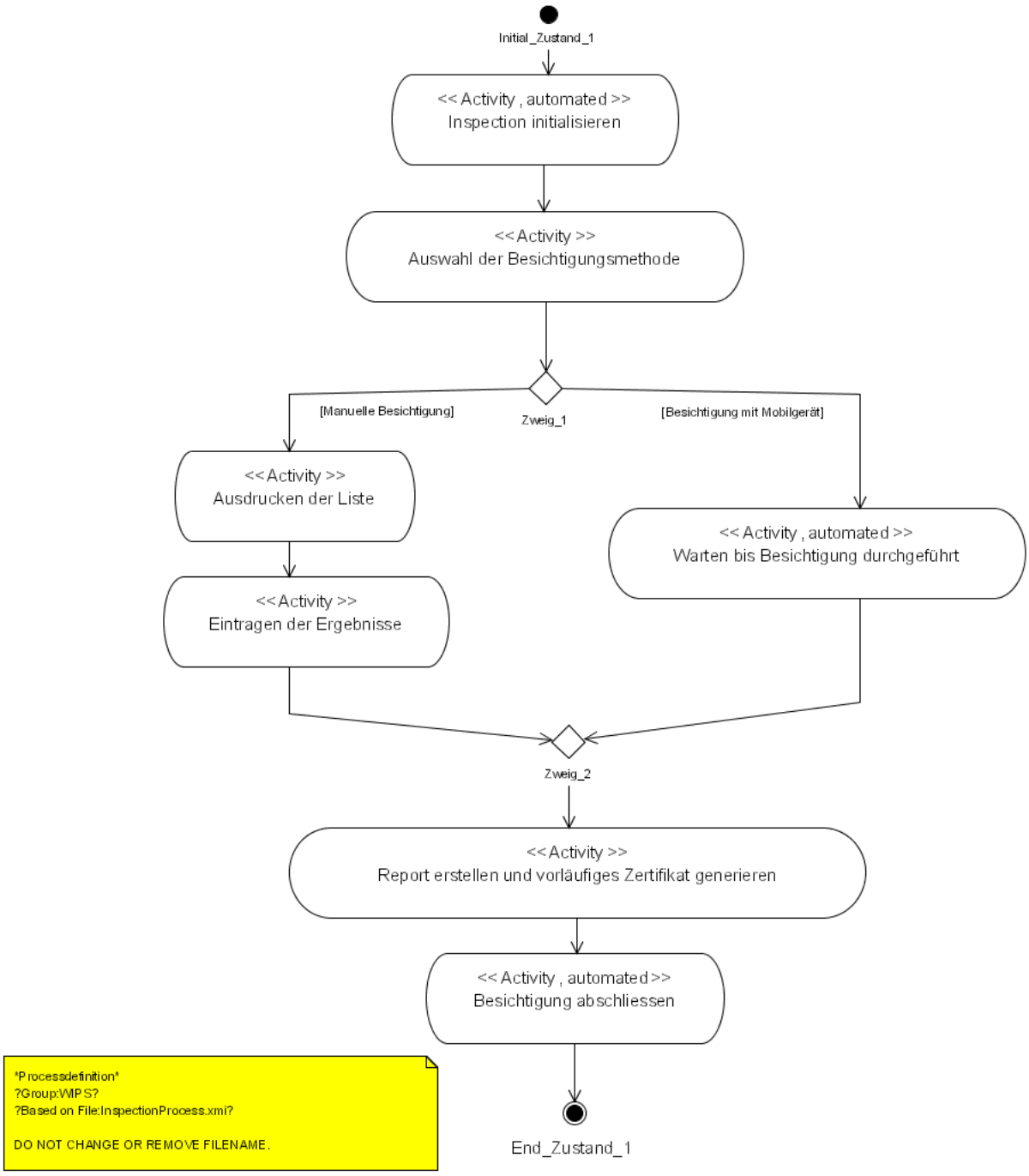


Abbildung 9.2 UML-Aktivitätsdiagramm des Subprozesses Besichtigungsdurchführung

Zum Start des Hauptprozesses war eine Änderung des Portalsystems notwendig. Die Container-Klasse Requests verwaltet Assets vom Typ Request. Ein solches Asset repräsentiert eine Kundenanfrage nach einer Schiffsbesichtigung. Die Container-Klasse wurde so erweitert, dass sie ein Ereignis des Typs *WIPSRequestGeneratedEvent* auslöst, falls ein neuer Kundenauftrag angelegt wird. Das für den Start des Subprozesses notwendige Ereignis vom Typ *WIPSNewInspectionEvent* wird durch einen Handler des Hauptprozesses ausgelöst.

Nachdem nun die Realisierung des Prozesses vorgestellt wurde, werden in den beiden folgenden Abschnitten ausgewählte Screenshots vorgestellt, die der Ausführung der beiden Prozesse entnommen sind.

## 9.2.2 Screenshots aus dem Hauptprozess

In Abbildung 9.3 wird der Screenshot der ersten manuellen Aktivität des Hauptprozesses gezeigt. Ein Mitarbeiter des Schiffsklassifizierungs-Unternehmens hat hier die Möglichkeit den Auftrag abzulehnen oder anzunehmen. Erst wenn er die Entscheidung getroffen hat, erscheint am unteren Bildrand der grüne Button zum Fortsetzen des Prozesses.

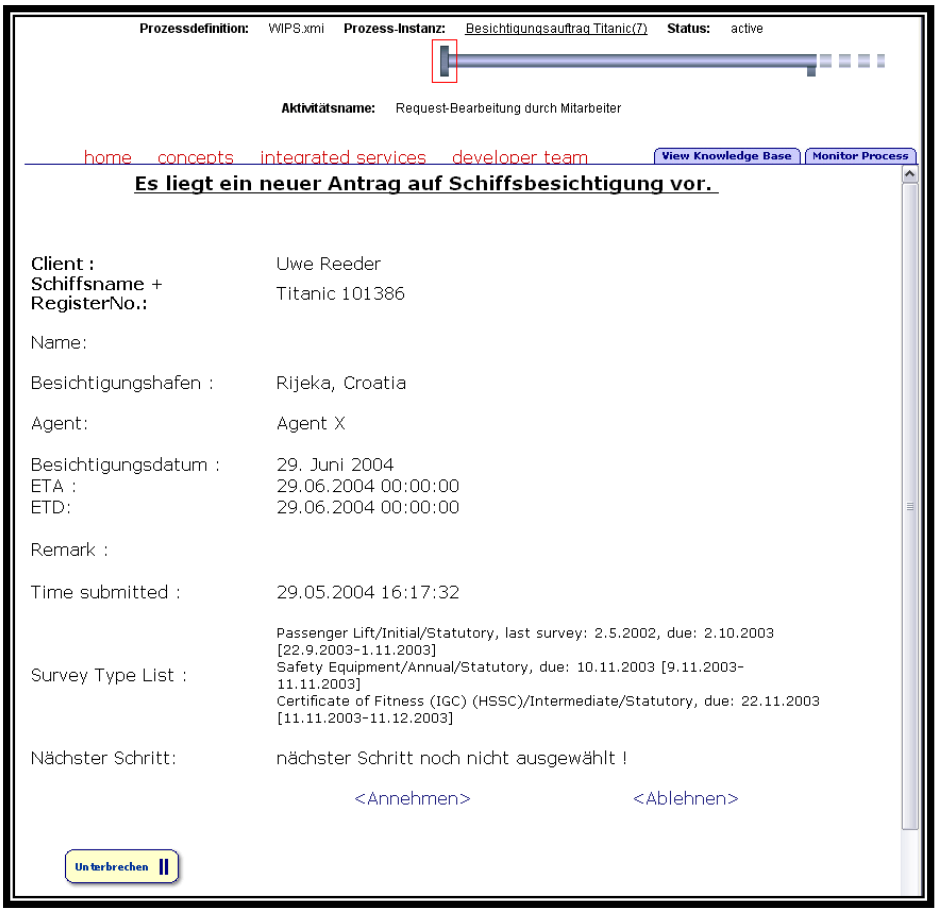


Abbildung 9.3 Screenshot zur manuellen Aktivität *Request-Bearbeitung durch Mitarbeiter*

In Abbildung 9.4 wird eine manuelle Entscheidung vom Benutzer gefordert. Der Prozessteilnehmer muss hier entscheiden, wie der Prozess fortgeführt werden soll. Es wird eine Liste von Besichtigungstypen und –punkten präsentiert. (engl. Survey types und items) Wenn die Liste zu lang ist und nicht in der für die Besichtigung zur Verfügung stehenden Zeit abgearbeitet werden kann, kann der Prozessteilnehmer diese Liste kürzen. Die Auswahl über alle Alternativen ist in den grünen Button zur Prozessfortsetzung als Auswahlliste eingebaut.

In der Abbildung 9.5 wird eine weitere manuelle Entscheidung gezeigt. Der Prozessteilnehmer muss aus einer Liste von möglichen Besichtigern einen Besichtiger auswählen, der die Schiffsbesichtigung durchführen soll. Man beachte hier, dass, solange kein Besichtiger ausgewählt ist, nur die Alternative „Kein Besichtiger verfügbar“ zur Prozessfortführung verfügbar ist. Erst nach Auswahl eines Besichtigers steht eine weitere Alternative zur Verfügung.

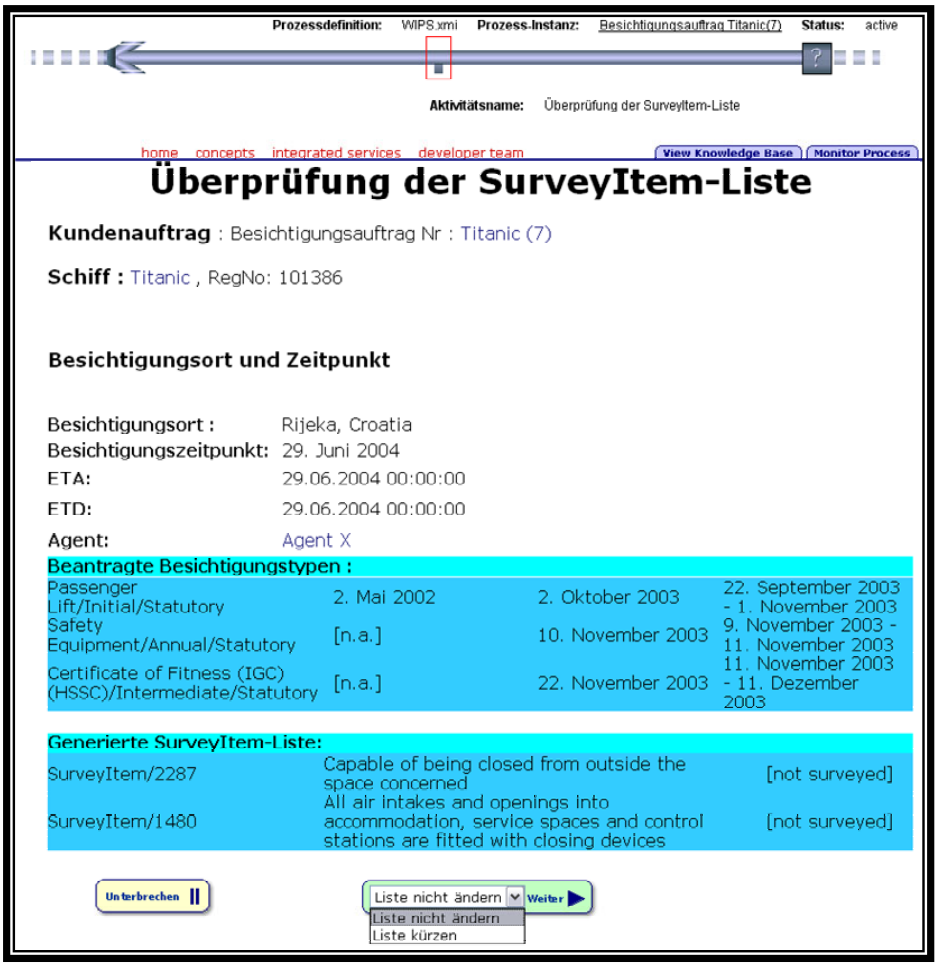


Abbildung 9.4 Screenshot zur manuellen Entscheidung *Überprüfung der SurveyItem – Liste*


Abbildung 9.5 Screenshot zur manuellen Entscheidung *Auswahl eines Besichtigers*

### 9.2.3 Screenshots aus dem Subprozess

Abbildung 9.6 zeigt eine weitere manuelle Aktivität, die während der Ausführung des Subprozesses zur Besichtigungsdurchführung durch einen Besichtiger aus-/durchgeführt wird: Der Besichtiger muss entscheiden, ob er die Besichtigung mit Unterstützung durch ein mobiles Gerät durchführen möchte oder ob er die manuelle Variante wählt. Weitere Informationen im Zusammenhang mit der mobilen Schiffsbesichtigung können [Aziz03] entnommen werden.

In Abbildung 9.7 wird die Liste der Besichtigungspunkte gezeigt, die sich der Besichtiger über die Druck-Funktion des Webbrowsers ausdrucken kann, wenn er die manuelle Besichtigungsmethode gewählt hat.

Abbildung 9.8 zeigt den Bildschirm, den der Besichtiger zur Eingabe der Besichtigungsergebnisse verwendet, wenn er die manuelle Besichtigungsmethode gewählt hat. Der Bildschirm ist an die Gestaltung der ausgedruckten Liste angepasst, damit der Besichtiger schneller die Ergebnisse eingeben kann.

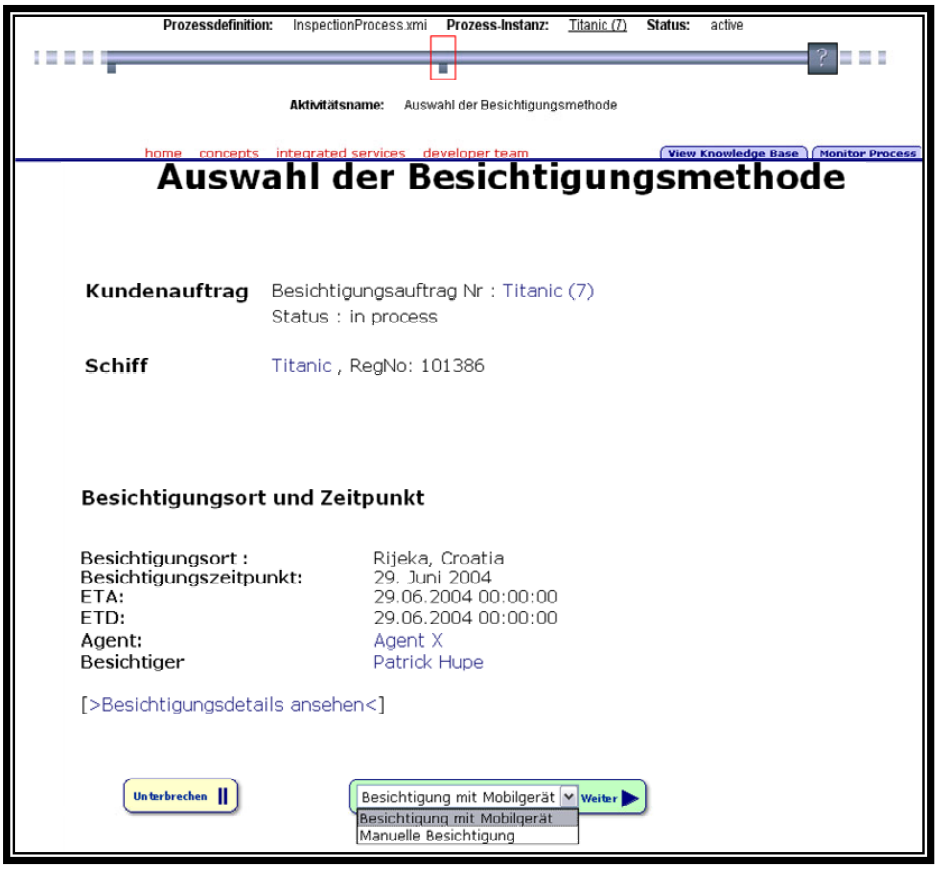


Abbildung 9.6 Screenshot zur manuellen Entscheidung *Auswahl der Besichtigungsmethode*

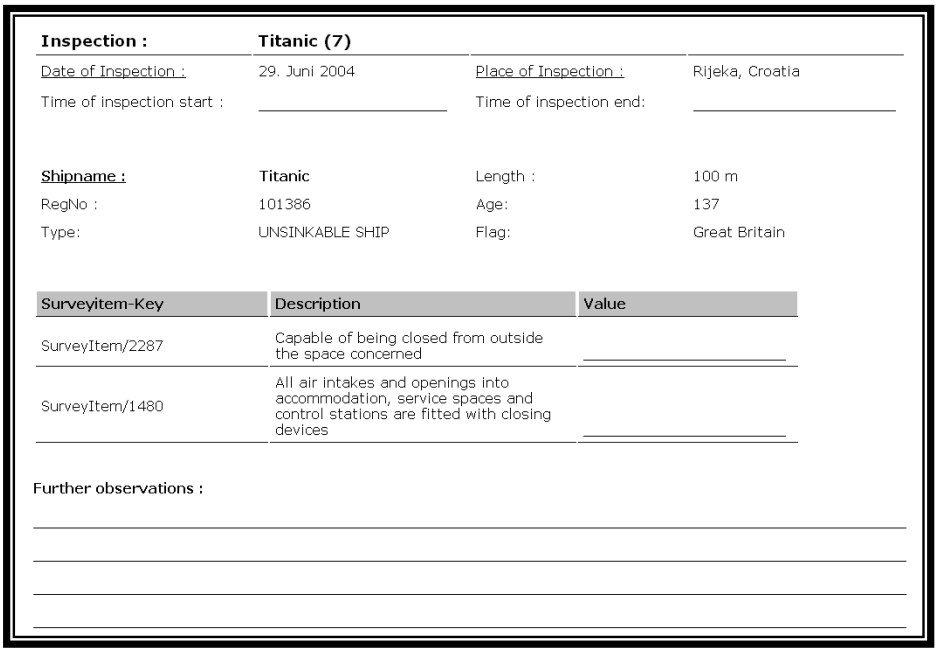


Abbildung 9.7 Druckansicht der Liste der Besichtigungspunkte

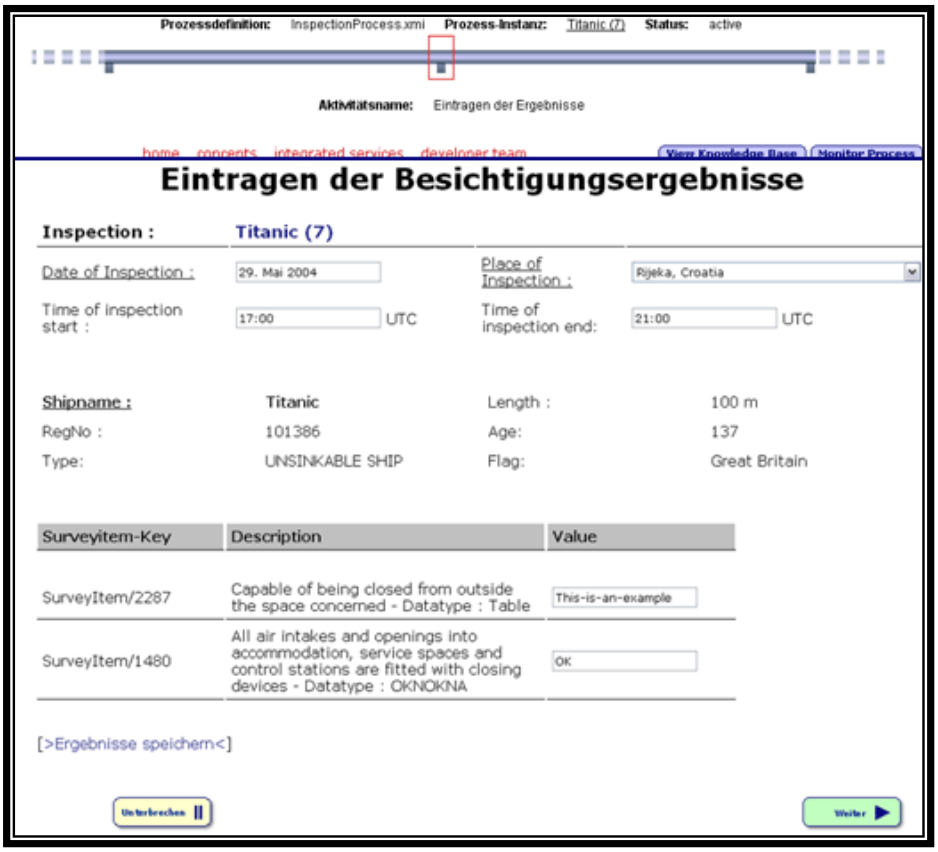


Abbildung 9.8 Screenshot zur manuellen Aktivität *Eintragen der Ergebnisse*

In der letzten Abbildung 9.9 wird der Abschlussbildschirm des Subprozesses zur Besichtigungsdurchführung gezeigt. Hier kann der Besichtiger über einen Link das *Survey Statement* vom System generieren lassen. Das Survey Statement ist der offizielle Besichtigungsbericht. Zur Generierung werden die Ergebnisse der Besichtigung, die mit dem in Abbildung 9.8 gezeigten Bildschirm eingegeben wurden, verwendet. Zusätzliche Bemerkungen zum Bereicht können im Textfeld *Report* angegeben werden. Des Weiteren werden hier alle Fotos gezeigt, die der Besichtiger während der Besichtigung mit einem mobilen Gerät aufgenommen hat (vgl. [Aziz03]). Dazu muss allerdings vorher die mobile Besichtigungsmethode gewählt worden sein.

Hiermit wurde präsentiert, wie der Schiffsbesichtigungs-Prozess mit dem in dieser Arbeit entwickelten Prozessunterstützungssystem definiert und realisiert wurde.

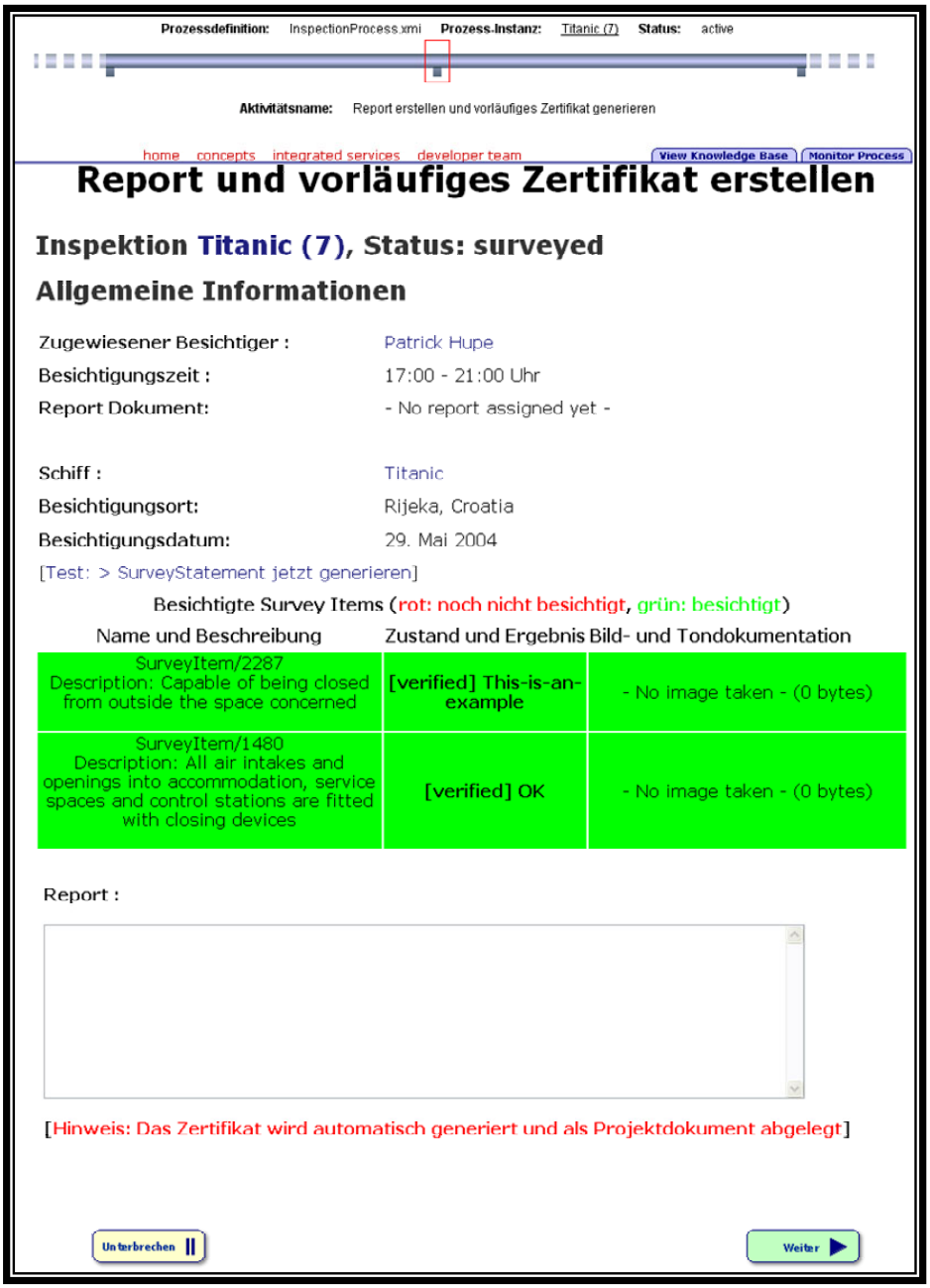


Abb 9.9 Screenshot zur manuellen Aktivität *Report erstellen und vorläufiges Zertifikat generieren*

# Kapitel 10

# Zusammenfassende Bewertung und Ausblick

Im Abschnitt 10.1 wird die Diplomarbeit zusammengefasst und evaluiert. Im Abschnitt 10.2 folgt dann abschließend ein Ausblick über denkbare Erweiterungs- und Verbesserungsmöglichkeiten des entwickelten Prozessunterstützungssystems.

## 10.1 Zusammenfassende Bewertung

In diesem Abschnitt wird diese Arbeit zusammengefasst und dabei die Evaluation des entworfenen Prozessunterstützungssystems aus Sicht des Autors geliefert. Es wird erklärt, welche der Anforderungen das entwickelte Prozessunterstützungssystem erfüllt und zu welchem Grade sie erfüllt werden. Des Weiteren wird dargestellt, ob jetzt andere Entwurfsentscheidungen getroffen würden. Die Evaluation erfolgt für die drei in Kapitel 5.1 definierten Komponenten des Prozessunterstützungssystems getrennt.
In Rahmen dieser Diplomarbeit wurde ein Prozessunterstützungssystem zur Unterstützung von Prozessen in Informationsportale entworfen und realisiert. Dazu wurde zunächst der Begriff der Portale eingeführt und eine Einführung in die verwendete Portalplattform geliefert. Anschließend wurde die Motivation zur Integration von Prozessunterstützungssystemen in Informationsportale aufgezeigt.
Zur Ermittlung der fachlichen Anforderungen wurden zwei Beispielprozesse analysiert. Ziel der Analyse war die Identifikation von Konzepten, die ein Prozessunterstützungssystem realisieren muss, damit die beiden Prozesse mit diesem System umgesetzt werden könnten. Bei den beiden Beispielprozessen handelte es sich um den Schiffsbesichtigungs-Prozess sowie um den Studienarbeiter-Prozess (vgl. Kapitel 4).
Im Anschluss an die Analyse der Beispielprozesse wurde der generelle Aufbau des Prozessunterstützungssystems festgelegt. Außerdem wurden allgemeine Anwendungsfälle, die bei der Verwendung eines Prozessunterstützungssystems auftreten, identifiziert. Diese Anwendungsfälle und die zuvor identifizierten Konzepte wurden als Anforderungen an das Prozessunterstützungssystem festgehalten.

**Prozessmaschine und Prozessmodell**
Als Prozessmodell und Prozessmaschine wurden dann Teile des Prototyps der Arbeit [Ahm03] wieder verwendet. Durch die Wiederverwendung des Prozessmodells und der Prozessmaschine aus [Ahm03] waren wesentli-

che Konzepte bereits auf Modellklassen abgebildet. Zur Anpassung an die Anforderungen, die vom existierenden Prozessmodell und der Prozessmaschine nicht erfüllt werden konnten, wurden diese beiden Komponenten erweitert.

Es wurden folgende Funktionalitäten hinzugefügt:

- Unterstützung von nebenläufigen Prozessen (Ablaufkonzepte Fork und Join)
- Unterstützung von manuellen und automatischen Entscheidungen (Ablaufkonzepte Choice und Merge)
- Unterstützung von Subprozessen (Definition, Start und Endnotifikation)
- Weiterleitung von Aktivitäten an andere Personen oder Rollen durch einen Benutzer
- Dynamische Zuweisung von Aktivitäten zur Ausführungszeit an Personen oder Rollen
- Versionierung von Prozessdefinitionen zur Sicherung der Stabilität gegenüber Änderungen.
- Ein einfaches Ereignis-Modell zur Ereignis-Behandlung.

Die folgende Übersicht zeigt die Anforderungen aus Tabelle 6.1, die das existierende Prozessmodell und die Prozessmaschine aus [Ahm03] nicht erfüllen und gibt an, ob diese Anforderungen durch die Erweiterungen realisiert werden konnten:

| **Anforderungen an Prozessmaschine und Prozessmodell** | **Konnte diese Anforderung realisiert werden?** |
|---|---|
| Unterstützung von Entscheidungen (Konzepte Choice und Merge) | Ja |
| Unterstützung nebenläufiger Aktivitäten (Konzepte Fork und Join ) | Ja |
| Unterstützung einer Ereignis-Behandlung (Auslösen von / Warten auf Ereignisse) | Teilweise |
| Unterstützung der verteilten Ausführung von Prozessen | Ja, aber nicht implementiert |
| Unterstützung von Subprozessen | Ja |
| Timeout-Strategie für Aktivitäten | Nein |
| Unterstützung des Anwendungsfalls „DataDictionary bearbeiten“ | Nein |
| Unterstützung der Anwendungsfälle zum Weiterleiten an einen anderen Benutzer oder an eine andere Rolle | Ja |
| Stabilität gegenüber Änderungen der Prozess-Definition | Ja |
| Dynamisches Zuweisen von Aktivitäten an Personen und Rollen zur Laufzeit | Ja |

Tabelle 10.1 Übersicht über erfüllte und nicht erfüllte Anforderungen für Prozessmodell und Prozessmaschine

Nur teilweise realisiert werden konnte die Unterstützung zur Ereignis-Behandlung. Der Grund liegt in der Tatsache, dass zur Darstellung von Ereignissen in Prozessdefinitionen die UML-Modellierungselemente „Signale“ verwendet werden sollten, die standardmäßig in UML-Aktivitätsdiagrammen vorhanden sind. Leider unterstützte das gewählte Modellierungswerkzeug *Poseidon* in der zur Verfügung stehenden Version entgegen den Aussagen des Herstellers keine Signale, so dass diese Anforderung nicht realisiert werden konnte. Auch in der aktuellen Version 2.4 werden keine Signale in UML-Aktivitätsdiagrammen unterstützt. Eine Unterstützung soll aber in einer späteren Version realisiert werden.
Das Prozessunterstützungssystem kann Ereignisse verarbeiten, die aber intern sind und nicht in den Diagrammen modelliert werden können. Sollen Ereignisse trotzdem modelliert werden, so können automatische Aktivitäten verwendet werden. Der zugeordnete Handler hat dann die Aufgabe, die Ereignisbehandlung zu übernehmen.

Die Ursache für die fehlende Berücksichtigung der anderen Anforderungen nach Tabelle 10.1 liegt in der Tatsache, dass der Umfang dieser Arbeit dann zu groß geworden wäre und die Realisierung nicht in vertretbarer Zeit durchzuführen gewesen wäre. Teilweise sind im Kapitel 6 für diese Aspekte theoretische Realisierungsmöglichkeiten präsentiert worden.
Ein entscheidender Nachteil ist, dass es keine Timeout-Strategie für Aktivitäten gibt. Dies würde es erlauben, explizite Ausnahmebehandlungen für Aktivitäten zu definieren, wenn diese nicht in einem vorgesehenen Zeitrahmen ausgeführt werden können. Auch die Möglichkeit zur generischen Bearbeitung des „DataDictionary" wäre wünschenswert gewesen. Dadurch könnten zum Beispiel Dokumente einem Prozesskontext hinzugefügt werden, die dann von folgenden Prozessteilnehmern eingesehen werden können. Sicher benötigen Prozesse auch speziellere Dienste zum Bearbeiten des DataDictionary. Diese könnten aber auf einem generischen Basisdienst aufbauen.
Die Unterstützung der verteilten Ausführung von Prozessen war eine spezielle Anforderung aus dem Schiffsbesichtigungs-Prozess, die dann bei einer konkreten Umsetzung des Prozesses mit diesem System gemäß dem Entwurf in Kapitel 6.1.4 implementiert werden müsste. Alle anderen in Tabelle 10. präsentierten Anforderungen, sowie die dort nicht aufgeführten Anwendungsfälle aus Kapitel 5.2 präsentierten Anforderungen sind realisiert worden.

Bewertung:
Die Entscheidung, das Prozessmodell und die Prozessmaschine aus der Arbeit [Ahm03] wieder zu verwenden und um die benötigten Anforderungen zu erweitern, war richtig. Dadurch konnte viel Zeit für die Realisierung andere Funktionen eingesetzt werden. Ein völlig neuer Entwurf eines Prozessmodells und einer Prozessmaschine, sowie die zugehörigen Realisierung wäre neben den anderen Implementierungen im Rahmen dieser Arbeit nicht durchführbar gewesen. Trotzdem konnten einige Anforderungen nicht realisiert werden. Der größte Nachteil in der Verwendung des existierenden Prozessunterstützungssystems besteht wohl in der fehlenden Unterstützung einer Timeout-Strategie. Dies ist eine Eigenschaft, die bei der Modellierung von Prozessen sehr sinnvoll ist.

**Definition und Verwaltung von Prozessen**
Zur Definition von Prozessen wurden in dieser Arbeit UML-Aktivitätsdiagramme gewählt. Es wurden mehrere UML-Modellierungswerkzeuge hinsichtlich ihrer Eignung als Prozesseditor untersucht. Es wurde dann die Entwurfsentscheidung getroffen, dass Modellierungswerkzeug *Poseidon* der Firma *Gentleware* [Gen04] als Prozesseditor einzusetzen und nicht den existierenden Prozesseditor aus [Ahm03] zu erweitern. Das Modellierungswerkzeug wurde in die Portalplattform integriert und kann über die Java Web Start-Technologie gestartet werden. Es wurden Konstrukte aus dem Bereich der UML-Profile verwendet, um die Aktivitätsdiagramme an den Anwendungskontext anzupassen. Die modellierten Prozesse werden in das XMI-Format [Xmi] konvertiert und über die WebDAV-Schnittstelle in das Portal importiert. Es wurde eine Softwarekomponente im Portal realisiert, die die in XMI vorliegenden Prozessdefinitionen auf Konzepte des Prozessmodells abbildet. Diese Softwarekomponente wird in einem eigens dafür realisierten Konvertierungsprozess verwendet.
Des Weiteren wurde eine Sprache zur Definition von Bedingungen in den Prozessdefinitionen entworfen und hierfür ein Syntaxparser realisiert, der diese Sprache erkennt und auf Bedingungskonstrukte des Portalsystems abbildet.

Die folgende Tabelle zeigt wie in Tabelle 10.2, welche Anforderungen zu realisieren waren, und ob dies gelungen ist:

| Anforderungen an die Definition von Prozessen | Konnte diese Anforderung realisiert werden? |
|---|---|
| | |
| Modellierung von nebenläufigen Prozessschritten | Ja |
| Modellierung von Ereignissen | Nein |
| Benennung von Rollen für die Ausführung von Aktivitäten | Ja |
| Modellierung von Entscheidungen | Ja |
| Definition von Kontrollflussbedingungen für Transitionen | Ja |
| Definition von Meta-Informationen (z.B. Beschreibung von Aktivitäten) | Ja |

Tabelle 10.2 Übersicht über erfüllte und nicht erfüllte Anforderungen für die Definition und Verwaltung von Prozessen

Nicht realisiert werden konnte die Modellierung von Ereignissen. Der Grund liegt wie oben schon erwähnt in der fehlenden Funktionalität des gewählten Modellierungswerkzeugs. Ansonsten konnten alle Anforderungen durch die Wahl von UML-Aktivitätsdiagrammen zur Definition von Prozessen erfüllt werden.

Bewertung:
Die Wahl von UML-Aktivitätsdiagrammen zur Definition von Prozessen war richtig, denn die UML-Aktivitätsdiagramme sehen alle benötigten Modellierungselemente vor. Auch nach dem derzeitigen Forschungsstand werden UML-Aktivitätsdiagramme zu diesem Zweck eingesetzt und sind daher weit verbreitet und gut bekannt. Die Entscheidung das XMI-Format zum Export der modellierten Diagramme zu nutzen, war richtig. Genau zu diesem Zweck wurde der XMI-Standard entwickelt. Allerdings gibt es noch erhebliche Unterschiede in der Unterstützung durch UML-Modellierungswerkzeuge. So gibt es zum Teil Unterschiede in der Repräsentation der Aktivitätsdiagrammelemente in XMI bei den verschiedenen Werkzeug-Herstellern, so dass sich mit einem Werkzeug als XMI-Darstellung exportierte Diagramme nicht ohne weiteres von dem Werkzeug eines anderen Herstellers importieren lassen. Hier muss abgewartet werden, ob der kommende Standard UML 2.0 [Jck04], der eine Erweiterung des XMI-Standards vorsieht, Akzeptanz findet. Des Weiteren könnte aber auch die XML Process Definition Language (XPDL), die von der WfMC zur Beschreibung und zum Austausch von Prozessdefinitionen zwischen verschiedenen Workflow-Anwendungen vorgeschlagen wird, bei einem Neuentwurf des Prozessunterstützungssystems eingesetzt werden. Hier sind in den letzten Jahren Fortschritte gemacht worden. Mit *JaWe* [Obj04] steht nun auch ein viel versprechender OpenSource-Editor zum Definieren von Workflows zur Verfügung, der in Java geschrieben und WebStart-fähig ist. Er unterstützt XPDL und erfüllt alle WfMC-Spezifikationen für Workflow-Editoren. Hier sollte genau abgewogen werden, welcher der beiden Standards in Zukunft wohl besser unterstützt wird. Die Verwendung von OpenSource-Produkten hat auch einen weiteren Vorteil: Der Sourcecode des Softwareproduktes steht zur Verfügung und das Produkt kann den Anforderungen entsprechend angepasst werden. So könnte der Editor zum Beispiel auch zur Überwachung von Prozessen eingesetzt werden. Bei Verwendung von kommerziellen Produkten ist man oft an die Erweiterungsmöglichkeiten, die der Hersteller vorsieht, gebunden.
Ein weiterer Kritikpunkt in diesem Zusammenhang ist, dass bei der Wahl von Poseidon als Modellierungswerkzeug zu sehr auf Aussagen des Herstellers vertraut wurde. Danach sollte eine Unterstützung der fehlenden UML-Modellierungselemente „Signale“ und „Swimlanes“ (siehe Kapitel 7) noch während der Anfertigung dieser Arbeit realisiert werden. Dies war aber nicht der Fall.

**Prozessverwaltung**

Im Rahmen der Realisierung der Anforderungen zur Prozessverwaltung in Kapitel 8 wurde das Prozessunterstützungssystem um Funktionalitäten erweitert, die es Benutzern erlaubt, die Anwendungsfälle aus Kapitel 5.4 zu nutzen. Zu diesem Zweck sind überwiegend Webseiten realisiert worden, die die entsprechenden Funktionalitäten über HTML-Links einem Administrator zur Verfügung stellen. Zur Überwachung und Verfolgung des Prozessfortschritts wurde die Java Applet-Technologie [Sun04d] verwendet. Es konnte nur ein Anwendungsfall nicht berücksichtigt werden: Die Bearbeitung des Prozesskontextes durch einen Administrator. Hierzu wären eine nähere Analyse des Prozesskontextes und der Entwurf einer Strategie zum Bearbeiten des Prozesskontextes notwendig, die im Umfang dieser Arbeit nicht geleistet werden konnten.

Bewertung:

Die Realisierung der Prozessverwaltung ist gut gelungen und erfüllt ihren Zweck. Auch die Verwendung der Java Applet-Technologie zur Prozessüberwachung macht in diesem Anwendungszusammenhang Sinn, denn die Benutzerinteraktion mit einem Portal erfolgt überwiegend über Webseiten und HTML-Formulare. Applets lassen sich auf einfache Weise in eine Web-Seite integrieren. Das Applet gibt einen graphisch, einfach zu erfassenden Überblick über die augenblickliche Aktivität, sowie über die nächsten bzw. vorangegangenen Aktivitäten, so dass auch dies als gelungen angesehen werden kann.

**Gesamtsystem**

In Abbildung 10.1 werden die realisierten und erweiterten Komponenten in einer Architekturübersicht gezeigt. Blau dargestellte Komponenten wurden in dieser Arbeit realisiert, während rot dargestellte Komponenten erweitert wurden. Als Portalplattform wurde die infoAssetBroker-Plattform [IA04] verwendet.

Zur Demonstration der Funktion des Prozessunterstützungssystems wurde der als Beispielprozess analysierte Schiffsbesichtigungs-Prozess mit dem System beispielhaft und in vereinfachter Form umgesetzt.

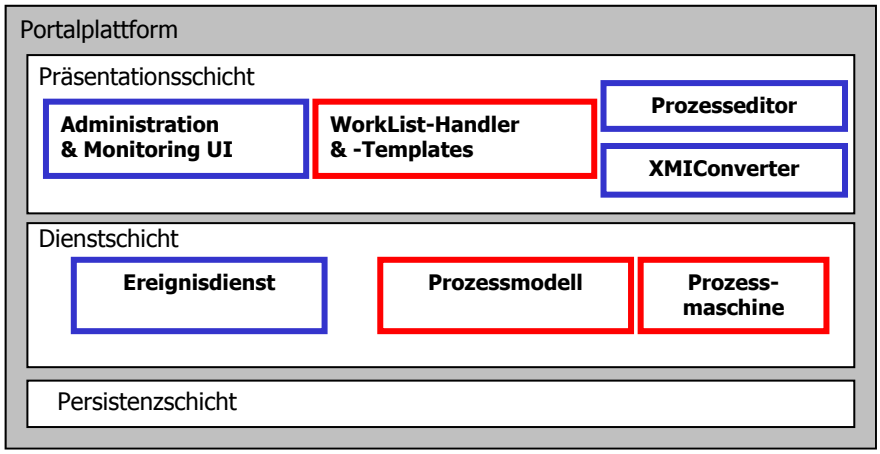


Abbildung 11.1 Realisierte und erweiterte Softwarekomponenten des Prozessunterstützungssystems
(Rot: erweitert | Blau : neu)

Bewertung des Gesamtsystems:

Insgesamt kann man sagen, dass die Realisierung des Prozessunterstützungssystems als gelungen bezeichnet werden kann. Das System kann Prozesse ausführen und übernimmt die Steuerung des Kontrollflusses zwischen den Aktivitäten. Den Benutzern des Portals werden als potentielle Prozessteilnehmer die ihnen oder den Rollen, deren Mitglied sie sind, zugeordneten Aktivitäten über eine Liste präsentiert, von der aus sie die Aktivitätsbearbeitung starten können. Etwas gewöhnungsbedürftig dürfte die Definition von Prozessen sein, denn nur UML-Aktivitätsdiagramme, die mit den notwendigen Anpassungskonstrukten erstellt worden sind, können als Prozessdefinition verwendet werden. So muss zum Beispiel zu jeder Aktivität ein ausführender Handler definiert

und implementiert werden. Zudem ist die Realisierung von Benutzerinteraktionen mit den Konzepten der Handler und Templates nicht gerade trivial. Dies setzt ein hohes Wissen des Modellierers über Funktion und Aufbau der zugrunde liegenden Portalplattform voraus. Aber auf der anderen Seite muss der Modellierer eines Prozesses immer ein gewisses Grundwissen des Systems haben, für das ein Prozess modelliert wird.

## 10.2 Ausblick

Dieser Abschnitt gibt einen Ausblick über die Verbesserungs- und Erweiterungsmöglichkeiten des realisierten Prozessunterstützungssystems.
Wenn das Modellierungswerkzeug *Poseidon* zu einem zukünftigen Zeitpunkt „Swimlanes“ und „Signale“ in UML-Aktivitätsdiagrammen unterstützen sollte, müssen der Konvertierungsmechanismus und die Prozessmaschine angepasst werden. Dann werden auch diese Modellierungselemente korrekt abgebildet und interpretiert. Die Alternative zur Zeit wäre, dass Signale über Aktivitäten mit speziellen Stereotypen modelliert werden, z.B. könnte eine Aktivität im UML-Aktivitätsdiagramm mit dem Stereotypen <<SendSignal>> gekennzeichnet werden, so dass dies gleichbedeutend mit einem ausgehenden Signal wäre (Siehe Kapitel 7.4).

Des Weiteren kann das Prozessunterstützungssystem um eine generische Benutzerschnittstelle zum Bearbeiten des Prozesskontextes erweitert werden. Dieser Punkt wurde in dieser Arbeit nicht berücksichtigt, weil die Realisierung einer solchen Schnittstelle vermutlich viel Zeit gekostet hätte, und so nicht im Umfang dieser Arbeit umzusetzen gewesen wäre. Die Schnittstelle könnte sowohl für Administratoren zu Verwaltungszwecken, als auch durch Prozessteilnehmer zum Hinzufügen von Informationen (z.B. Dokumenten) zum Prozess genutzt werden. Im jetzigen Zustand muss diese Funktionalität für Prozessteilnehmer aufwändig für jeden Prozess und jede manuelle Aktivität über Benutzerinteraktion mit HTML-Formularen und entsprechenden Handlern realisiert werden. Eine generische Schnittstelle könnte für alle mit dem Prozessunterstützungssystem ausgeführten Prozesse genutzt werden. Ein Entwurfsvorschlag wäre, diese Schnittstelle ebenfalls über ein Java Applet zu realisieren, welches in die Web-Seiten eines Prozesses eingebunden werden kann. Das Applet würde den konkreten Prozesskontext ermitteln und enthaltene Assets sowie ihre Attribute darstellen. Dazu sollte allerdings ein Sicherheitskonzept entworfen werden, das beschreibt, welcher Benutzer welche Informationen des Prozesskontextes lesen oder schreiben darf.

Als zusätzliche Erweiterung könnte der Modellierer beim Zusammensetzen eines Prozesses aus existierenden Aktivitäten unterstützt werden. Diese Unterstützung könnte durch einen Aktivitätenpool, der alle im Prozessunterstützungssystem bereits definierten Aktivitäten auflistet, realisiert werden. Damit wird dem Modellierer geholfen. Er könnte aus dem Aktivitätenpool bereits existierende Aktivitäten auswählen und in einem neu modellierten Prozess verwenden. Dies vermeidet eine doppelte Definition und Realisierung einer Aktivität, die schon im System vorhanden ist. Natürlich lassen sich nicht alle Aktivitäten ohne weiteres in Prozessen wieder verwenden, da die Realisierung der zugeordneten Handler häufig speziell auf einen Prozess ausgerichtet ist. Im Einzelfall kann aber eine Wiederverwendbarkeit von Standardaktivitäten gegeben sein (z.B. das Erstellen eines Berichts durch einen Prozessteilnehmer).

Eine weitere Verbesserung wäre die Unterstützung der Ausnahmebehandlung für Aktivitäten im Prozessunterstützungssystem. Diese Ausnahmebehandlung könnte dann eingreifen, wenn Fehler während der Ausführung von Aktivitäten auftreten oder wenn die Ausführungsdauer für eine Aktivität überschritten wird. Im letzteren Fall könnte die Aktivität zum Beispiel an einen Vertreter weitergeleitet werden, oder an die Standard-Rolle zurückgegeben werden. Die Ausnahmebehandlung müsste so realisiert sein, dass die konkrete Ausnahmeaktion zu jeder Aktivität schon bei der Modellierung des Prozesses im UML-Aktivitätsdiagramm definiert werden kann.

Ebenfalls eine Verbesserung würde die Integration der Prozessfortführung (Beenden der aktuellen Aktivität, Wahl der nächsten Aktivität bei Entscheidungen) bei manuellen Aktivitäten in das in dieser Arbeit realisierte Java Applet darstellen. Dadurch kann das Applet nicht nur dazu genutzt werden, den Prozessfortschritt zu verfolgen, sondern ihn auch zu steuern. Der Vorteil wäre, dass diese Funktionalität für alle Prozesse zentral zur Verfügung stünde. Bisher muss die Funktionalität zur Fortführung des Prozesses in jeder Seite über HTML-Links realisiert werden.

Die letzte Verbesserungsmöglichkeit betrifft die verwendete Technologie bei der Realisierung der nebenläufigen Ausführungen von verschiedenen Prozesszweigen. Mit Erscheinen des Java Development Kits in der Version 1.5 wird ein Standard-Paket zur Behandlung von Nebenläufigkeit hinzugefügt: java.util.concurrent. Es bietet Unterstützung für nebenläufige Ausführungen von Aufgaben und definiert auch eine Klasse CyclicBarrier, die zum Synchronisieren von mehreren Threads, welche alle aufeinander warten, verwendet werden kann. Dies trifft im Prozessunterstützungssystem bei der Ausführung einer Join-Aktivität mit Join-Bedingung „AND-JOIN“ zu. Derartige Klassen des neuen Paketes sollten auch im Prozessunterstützungssystem verwendet werden, da sie extra für nebenläufige Probleme entwickelt und getestet werden.

# Anhang A – Klassendiagramm des Job-Dienstes

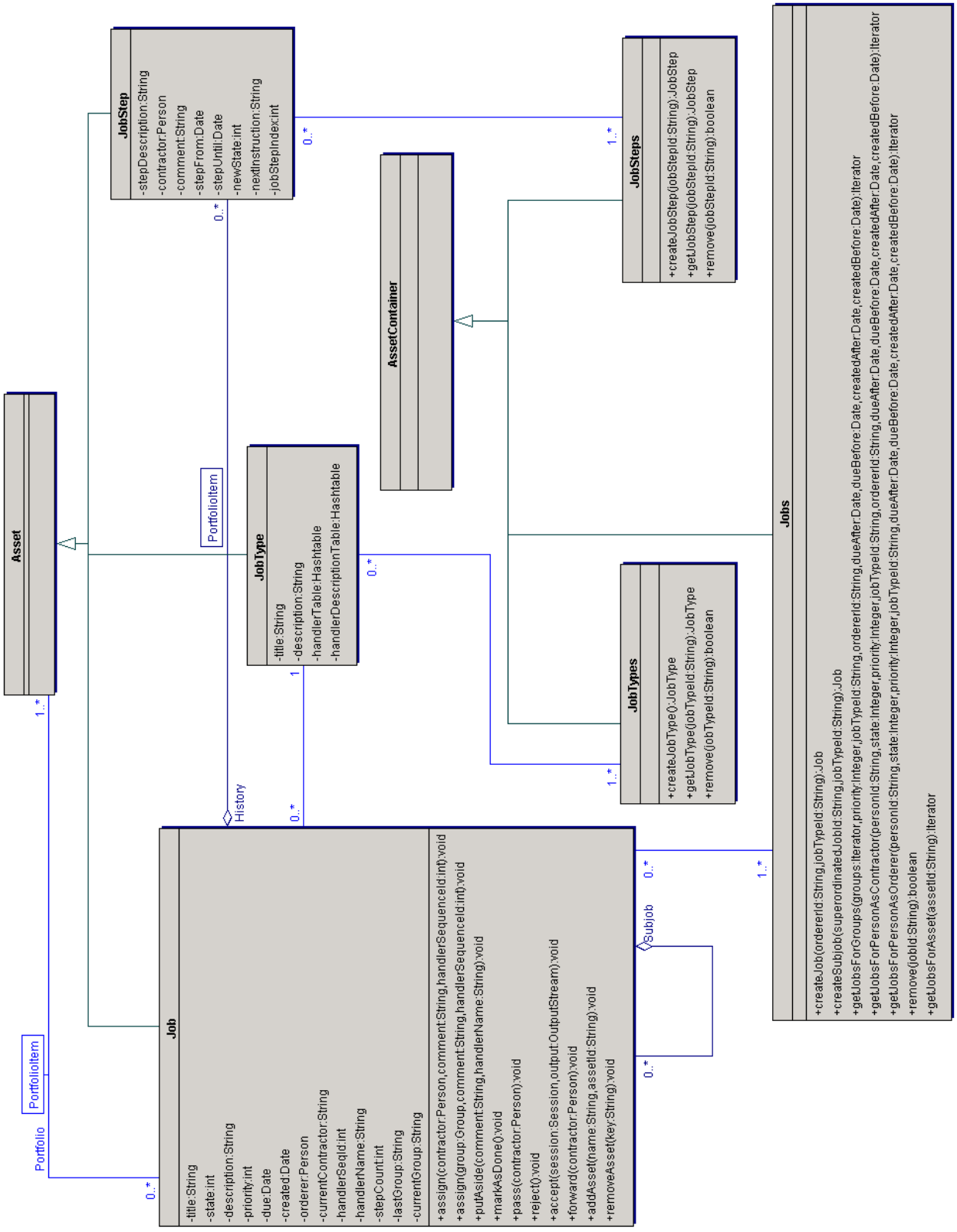


Abbildung A.1 Klassendiagramm des Job-Dienstes aus [Leh01]

# Anhang B – Packages des Prozessunterstützungssystems

| Package | Enthaltene Klassen |
|---|---|
| de.tuhh.sts.processsupport.handler.convertingProcess | ConvertProcessDefinitionHandler<br>CreateConvertProcessHandler<br>InformAdminHandler<br>InformAuthorHandler<br>StartProcessHandler<br>VerifyProcessDefinitionHandler<br>WaitForChangedFileHandler |
| de.tuhh.sts.processsupport.handler.execution | AssignHandler<br>ExecuteManualActivityHandler<br>ExecuteManualDecisionHandler<br>ForwardedHandler<br>ForwardHandler<br>PostManualActivityExecutionHandler<br>RejectHandler<br>TasksListHandler |
| de.tuhh.sts.processsupport.handler.management | BreakInstanceHandler<br>BreakInstanceThreadHandler<br>BreakInstanceThreadUnsafeHandler<br>DelegateActivityHandler<br>FinishActivityInstanceHandler<br>ReexecuteActivityInstanceHandler<br>RemoveDefinitionsHandler<br>RemoveInstancesHandler<br>ResetInstanceHandler<br>ResumeInstanceHandler<br>ResumeInstanceThreadHandler<br>SaveDefinitionChangesHandler<br>StoppInstanceHandler<br>StoppInstancesHandler<br>ViewProcessDefinitionHandler<br>ViewProcessInstanceDetailsHandler<br>ViewProcessInstancesHandler<br>ViewProcessInstanceThreadHandler |
| de.tuhh.sts.processsupport.handler.wipsProcess | AcceptOrderHandler<br>AddInfoHandler<br>AdvancePaymentHandler<br>AdvancePaymentReceiveHandler<br>AssignInspectorHandler<br>EnterResultsHandler<br>FinishHandler |

| Package | Enthaltene Klassen |
|---|---|
| de.tuhh.sts.processsupport.handler.wipsProcess | FinishInspectionHandler<br>FinishSurveyHandler<br>GenerateInspectionHandler<br>GenerateReportAndCertificateHandler<br>GetInfoHandler<br>InformInspectorHandler<br>InformRequestRejectedHandler<br>InformShipOwnerNoInspectorHandler<br>InformShipOwnerRemoveSurveyTypesHandler<br>InitInspectionHandler<br>InitProcessHandler<br>PrintListHandler<br>RemoveSurveyTypeHandler<br>RequestFormHandler<br>RequestOverviewHandler<br>SelectInspectiontypeHandler<br>SolvencyHandler<br>SurveyTypeItemOverviewHandler<br>ViewInspectionDetailsHandler<br>ViewOrderHandler<br>ViewShipHandler<br>ViewSurveyItemListHandler<br>WaitForFinishingHandler<br>WaitForMobileInspectionFinishingHandler |
| de.tuhh.sts.processsupport.handler | OverviewHandler |
| de.tuhh.sts.processsupport.interfaces | Activities<br>Activity<br>ActivityInstance<br>ActivityInstances<br>ProcessContext<br>ProcessContexts<br>ProcessDefinition<br>ProcessDefinitionProject<br>ProcessDefinitionProjects<br>ProcessDefinitions<br>ProcessInstance<br>ProcessInstances<br>ProcessInstanceThread<br>ProcessInstanceThreads<br>Transition<br>Transitions |
| de.tuhh.sts.processsupport.services | IMPActivities<br>IMPActivity |

| Package | Enthaltene Klassen |
|---|---|
| de.tuhh.sts.processsupport.services | IMPActivityInstance<br>IMPActivityInstances<br>IMPProcessContext<br>IMPProcessContexts<br>IMPProcessDefinition<br>IMPProcessDefinitionProject<br>IMPProcessDefinitionProjects<br>IMPProcessDefinitions<br>IMPProcessInstance<br>IMPProcessInstances<br>IMPProcessInstanceThread<br>IMPProcessInstanceThreads<br>IMPTransition<br>IMPTransitions<br>StateChangeNotPossibleException<br>TypeNotSupportedException |
| de.tuhh.sts.processsupport.services.Engine | AssignmentState<br>CommonProcessEvents<br>CouldNotStartProcessEvent<br>ExecutionImpossibleException<br>ProcessEngine<br>ProcessFinishedEvent<br>ProcessingState<br>ProcessInstanceThreadMapper<br>ProcessStartedEvent<br>ProcessStarter<br>ProcessStartEvent<br>StateChangeException |
| de.tuhh.sts.processsupport.monitoring | ActivityInstanceInformation<br>Box<br>HelperThread<br>MonitorApplet<br>MonitorServlet<br>ProcessInformation<br>UnderlinedLabel |
| de.tuhh.sts.processsupport.util | ActivityGraphException<br>ConditionException<br>ConditionLexer<br>ConditionParser<br>ConditionTreeParser<br>CreateConvertProcess<br>ElementNotSupportedException<br>InitialStateException<br>NoActivityException<br>NoEndStateException |

| Package | Enthaltene Klassen |
|---|---|
| de.tuhh.sts.processsupport.util | NoGuardException<br>NotAXmiDocumentException<br>NotEnoughTransitionsException<br>XMIConverter<br>XmiVerifyException |
| de.tuhh.sts.classifier | ProcessDefinitionClassifier |

Tabelle B.1 Packages des Prozessunterstützungssystems

# Anhang C – ANTLR-Grammatik-Datei für den Syntaxparser

```
class ConditionParser extends Parser;
options { buildAST=true; }

condition:   (simpleCondition
             | complexCondition
             | NOT^ condition );

protected complexCondition : (OPBR! condition LOGICOP^ condition CLBR!);

protected simpleCondition
       : ( term COMPOP^ term
       { System.out.println("Found new Condition");}
       );

protected term
    : ( LBRACKET! (LITERAL COMMA!)? EXPRESSION COMMA! DATATYPE RBRACKET!)
    { System.out.println("Found new Term");}
    ;

class ConditionTreeParser extends TreeParser;

condition returns [Condition c]
{ c = null;
  Condition a,b;
  Term g,h;
}       : #(con:LOGICOP a=condition b=condition)
        { if (con.getText().equals("&&")) {
                    //AND Condition
                    c = services.getConditions().createANDCondition(a,b);
                    System.out.println("Building new AND Condition : Left
"+a.printExpression() + " - Right : "+b.printExpression());
        } else {
                    //OR Condition
                    c = services.getConditions().createORCondition(a,b);
                    System.out.println("Building new OR Condition : Left
"+a.printExpression() + " - Right : "+b.printExpression());
                } }
        | #(op:COMPOP g=term h=term)
        { Operator o = ser-
vices.getOperators().getOperatorByName(op.getText());
                c = services.getConditions().createExpression(g,o,h);
                                System.out.println("Building new Condition :
"+c.printExpression()); }
        | #(NOT a=condition)
        {   c = services.getConditions().createNOTCondition(a);
                        System.out.println("Building new NOT Condition :
"+c.printExpression());}
        ;

protected term returns [Term t]
{ t = null;} :  (ex:EXPRESSION d:DATATYPE)
         { DataType type = servi-
ces.getDataTypes().getDataTypeByName(d.getText());
           t =
services.getTerms().createVariableTerm("",ex.getText(),type);
           System.out.println("Building new Term : "+
t.getDataType().getName()+" - "+t.getPrintExpression() );}
```

```
        | (LITERAL exp:EXPRESSION dt:DATATYPE)
        { DataType type = ser-
vices.getDataTypes().getDataTypeByName(dt.getText());
                t = ser-
vices.getTerms().createLiteralTermByInput(type,exp.getText().substring(1));
                                System.out.println("Building new LITERAL Term
: "+ t.getDataType().getName()+" - "+t.getPrintExpression() );};


class ConditionLexer extends Lexer;
options { k = 2;}

LOGICOP: "&&"
         | "||"
         ;

NOT: "!!"
   ;

COMPOP:   ("=="
          |'<'
          |'>'
          |"<="
          |">="
          |"!="
          |"in"
          );

LITERAL : "LITERAL";

LBRACKET : ('(') ;

RBRACKET : (')');

OPBR : ('[');

CLBR : (']');

COMMA : (',');

EXPRESSION: ('/'('a'..'z'|'A'..'Z'|'0'..'9')+)+
    ;

DATATYPE: ("DATE"
          |"DOUBLE"
          |"INTEGER"
          |"TEXT"
          |"OBJECT"
          |"BOOLEAN");
IGNORE
    :(   '\r'   // DOS
    |   '\n'    // UNIX
    |   ' '    )// WhiteSpaces
    {$setType(Token.SKIP);};
```

# Anhang D – UML-Aktivitätsdiagramme

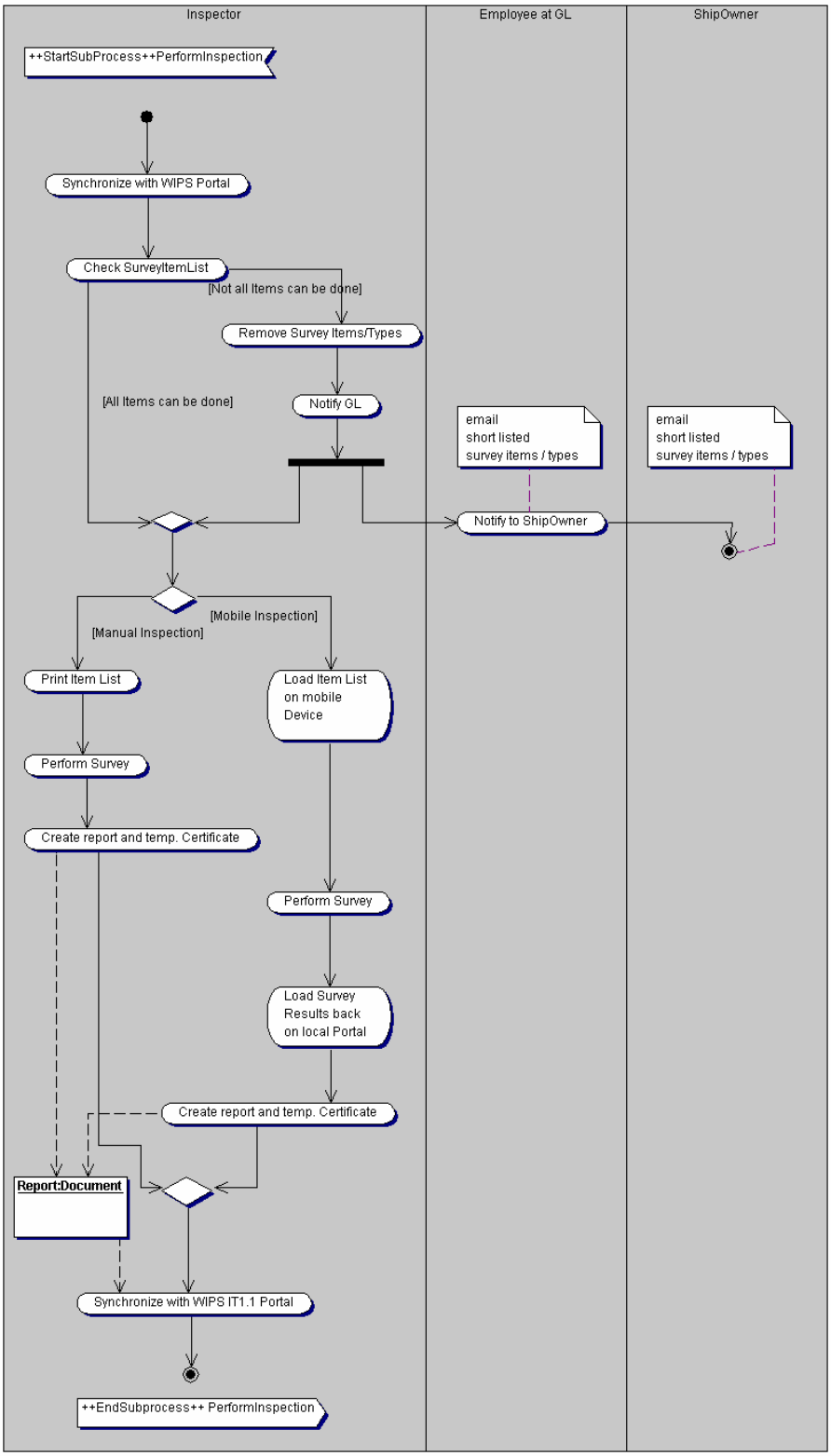


Abbildung D.1 Aktivitätsdiagramm des Subprozesses Besichtigungsdurchführung

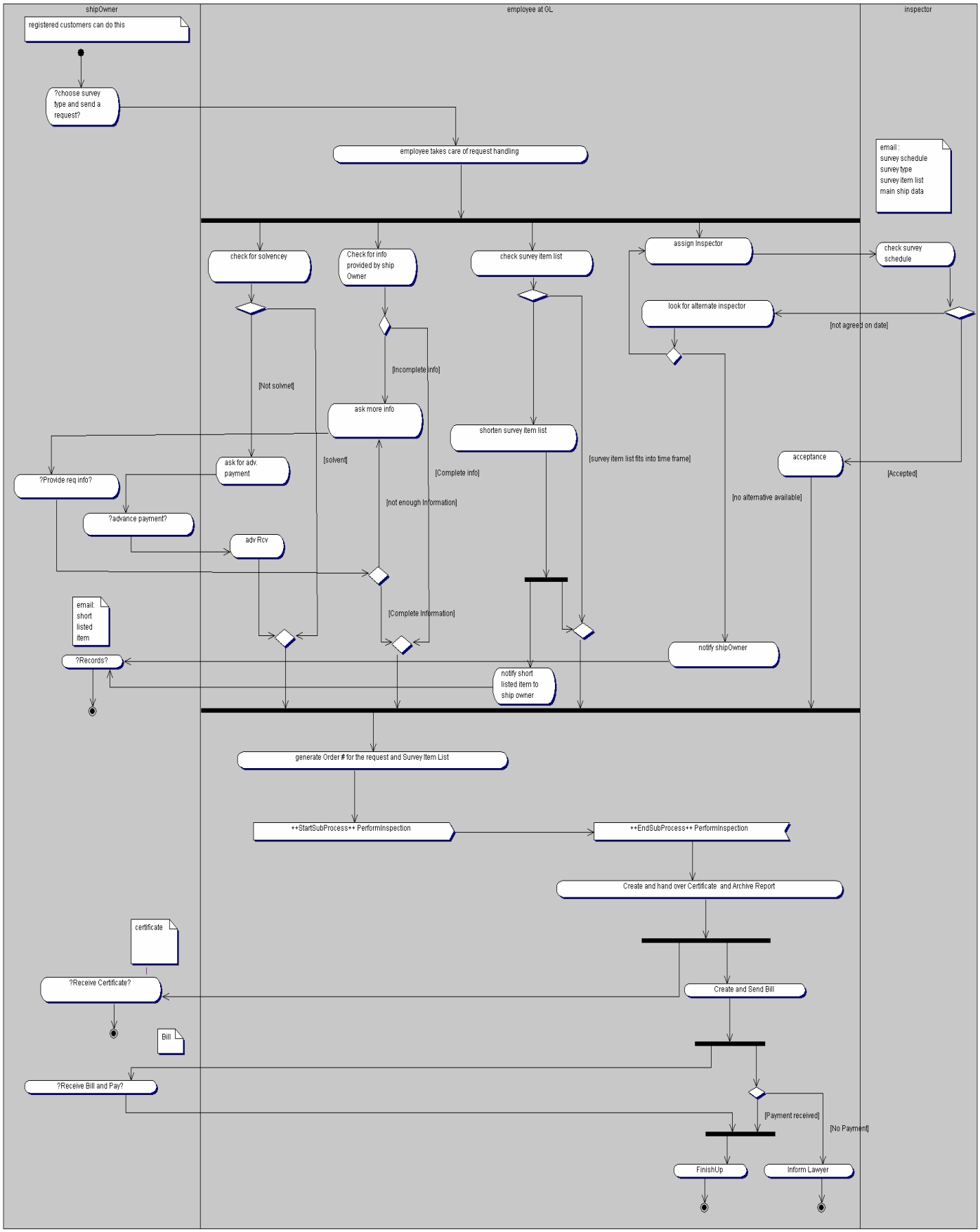


Abbildung D.2 Aktivitätsdiagramm Schiffsbesichtigungs-Prozess

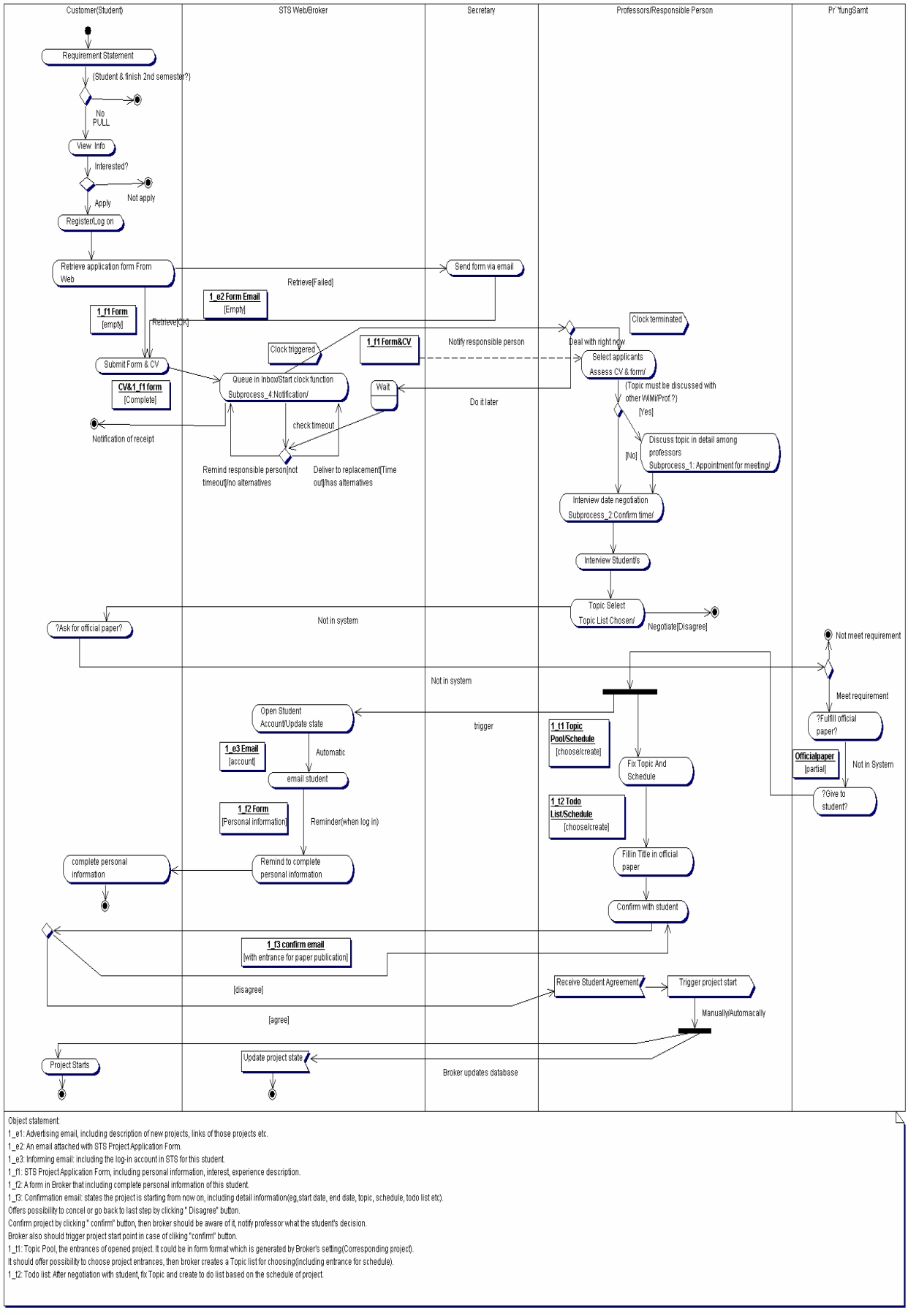


Abbildung D.3 Aktivitätsdiagramm Studienarbeitsprozess

# Literaturverzeichnis


[Aa03] VAN DER AALST, Wil – *Inheritance of Business Processes: A Journey visiting Four notorious Problems*. Erschienen in: EHRIG, H.; REISIG, W.; WEBER, H. – *Petri Net Technology for Communication Based Systems*, volume 2472 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin 2003

[AH02] VAN DER AALST, W.; VAN HEE, K.M. – *Workflow Management –Models, Methods and Systems*, MIT Press 2004

[Ahm02] AHMED, Ayaz - *Analysis and Design of Process Concepts for a Cooperative Information System*, Student Project, Technische Universität Hamburg-Harburg, Februar 2002

[Ahm03] AHMED, Ayaz - *Prototype of Software Components for Process Support in an Enterprise Information Portal*, Master Thesis, Technische Universität Hamburg-Harburg, Februar 2003

[Amz04] AMAZON.DE, vertikales Portal, Website, Juni 2004
http://www.amazon.de

[AOD00] VAN DER AALST, W.; OBERWEIS, A.; DESEL, J. – *Lecture Notes in Computer Science, Volume 1806, Business Process Mangement*, Springer-Verlag, Berlin Heidelberg 2000

[Aol04] AOL, horizontales Portal, Website, Juni 2004
http://www.aol.com

[Aziz03] AZIZ, Mubashir – *A model for mobility-aware information system services*, Master Thesis, Technische Universität Hamburg-Harburg, März 2003

[BaHa00] BHATTI, N.; HASSAN, W. Object Serialization and Deserialization Using Xml. 2000.

[Bor04] BORLAND – TOGETHER CONTROL CENTER, Website, Juni 2004
http://www.borland.de/together/controlcenter/index.html

[CDK01] COLOURIS, G.; DOLLIMORE, J.; KINDBERG, T. – Distributed Systems, Addison-Wesley, Harlow 2001

[Des00] DESFRAY, Philippe – *UML Profiles versus Metamodel extensions: An ongoing debate*, Vortagsfolien, Firma SOFTEAM – Think Object, 2000
http://www.softeam.fr (Juni 2004)

[Eel03] EELBO, Jan – *Synchronisation von Besichtigungsinformationen, Studienarbeit*, Technische Universität Hamburg-Harburg, Juli 2003

[Ess01] ESSER, Friedrich – *Java 2, Designmuster und Zertifizierungswissen*, Galileo Press, Bonn 2001

[FrMü03] FRANSSON, J.; MÜLLER, S. – *UML-Diagramme – Austausch und Interaktion*, Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Verteilte Systeme und Informationssysteme, Juli 2003

[Ge03] GE, Jing – *Einbindung des WebDav-Dienstes in ein Informationsportal*, Studienarbeit, Technische Universität Hamburg-Harburg,, Mai 2003

[Gen04] GENTLEWARE – POSEIDON COMMUNITY EDITION, Website, Juni 2004
http://www.gentleware.com/products/descriptions/ce.php4

[Ger03] GERKENS, Christoph - *Entwicklung einer generischen, metadaten-orientierten Business-Schicht eines Portalssytems*, Studienarbeit, Technische Universität Hamburg-Harburg, Septemper 2003

[GHJV95] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. – *Design patterns, elements of reusable object-oriented software*, Addison-Wesley, 1995

[Gic03] GICHAHI, Henry Kamau – *Rule-based Process Support for an Enterprise Information Portal*, Master Thesis, Technische Universität Hamburg-Harburg, Juni 2004

[Gl04] GERMANISCHER LLOYD AG , Website, Juli 2004.
http://www.gl-group.com

[HLAS04] HUPE,P,; LANGBECKER, U.; AZIZ, M.; SCHMIDT, J.W. – *Mobile and Web-based Services for Ship Operation and Survey: A Portal-centric Approach*, Konferenz COMPIT'04 Siguenza, Spanien 2004.

[HMS04] HUPE, P.; MARRONE, R.; SCHMIDT, J.W. – *WIPS : Wettbewerbsvorteile durch Informationstechnisch unterstützte Produktsimulation im Schiffbau*, Schlussbericht Teilprojekt WIPS IT 1.1, Arbeitsbereich Softwaresysteme, Technische Universität Hamburg-Harburg, März 2004

[IA04] INFOASSET AG, München, Website, Juni 2004.
http://www.infoasset.de

[IAWP03] INFOASSET AG, *InfoAsset Broker Technical Whitepaper*, Hamburg, April 2001

[Ibm04] IBM – RATIONAL SOFTWARE, Website, Juni 2004 (Hinweis: Produktfamilie Rational Rose Enterprise ist nicht länger verfügbar. Als Modellierungswerkzeug wird nun Rational Rose XDE Modeller angeboten.)
http://www.rational.com

[Info01] INFOASSET AG. *Pflichtenheft-Contentmap*, Glossar, Hamburg, 2001

[JaBu96] JABLOSNKI, S.; BUSSLER, C. – *Workflow Management, Modeling Concepts, Architecture and Implementation*, International Thomson Computer Press, 1996

[Jak04] THE APACHE JAKARTA PROJECT, *Apache Tomcat*, SevletEngine, Website, Juni 2004
http://jakarta.apache.org/tomcat/

[JBR99] JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James. *The Unified Software Development Process*, Addison-Wesley, 1999.

[Jck04] JECKLE.DE, Website, Juni 2004
http://www.jeckle.de

[JH03] GAMMA, E.; EGGENSCHWILER,T. - *The JHotDraw framework*, Website, Juni 2004
http://www.jhotdraw.org

[JTB00] JANSEN, C., THIESSE, F., BACH, V. *Wissensportale aus Systemsicht in: Business Knowledge Management in der Praxis: Prozessorientierte Lösungen zwischen Knowledge Portal und Kompetenzmanagement*, Springer Verlag, Berlin 2000

[Kaya01] KAYA, Atila – *Workflow Interoperability: The WfMC Reference Model and an Implementation*, Master Thesis, Technische Universität Hamburg-Harburg, April 2001

[KLT00] KOENEMANN, J.; LINDNER, H.-G.; THOMAS, C. - *Unternehmensportale: Von Suchmaschinen zum Wissensmanagement*. In: nfd Information –Wissenschaft und Praxis 51 (2000), September, Nr. 6

[Leh01] LEHEL, Vanda - *Entwurf und Realisierung eines Basisdienstes zur Unterstützung arbeitsteiliger Geschäftsvorgänge in einem Unternehmensportal*, Technische Universität Hamburg-Harburg, Hamburg, Juli 2001

[Leh02] LEHEL, Vanda, Synchronisation von Informationen zwischen Portalen, Diplomarbeit, Hamburg im September 2002

[Lyc04] LYCOS, horizontales Portal, Website, Juni 2004
http://www.lycos.de

[MH02] MÜLLER, R.; HUPE, P. – *Referenzfolien über die infoAssetBroker-Plattform*, zur Verfügung gestellt durch den Arbeitsbereich Softwaresysteme der Technische Universität Hamburg-Harburg, 2002

[Msn04] MSN, horizontales Portal, Website, Juni 2004
http://www.msn.de

[Obj04] OBJECTWEB – *The JaWe project*, Website, Juni 2004
http://jawe.objectweb.org

[Ock01] OCKE, Stefan – *XMI & Java*, Folien zum Vortrag an der TU-Dresden, Fakültät für Informatik, 05.07.2001

[OMG04] OBJECT MANAGEMENT GROUP, Website, Juni 2004
http://www.omg.org

[Pa04] PARR, Terence – *ANTLR*, Website, Juni 2004
http://www.antlr.org

[RFKR02] ROSE, T.; FÜNFFINGER, M.; KNUBLAUCH, H.;RUPPRECHT, C. – *Prozessorientiertes Wissensmanagement, Forschungsinstitut für anwendungsorientierte Wissensverarbeitung*, Ulm, Januar 2002

[RM03] RAHM, Prof. Dr. E.; MÜLLER, R. – *Workflow-Management und E-Services*, Online-Skript zur Vorlesung im Sommersemester 2003, Kapitel 1, Website, Juni 2004
http://dbs.uni-leipzig.de/de/skripte/WMES/HTML/kap1-1.html

[Serv04] *Einführung in die Servlet-Technologie*, Website, Juni 2004
http://pubwww.fhzh.ch/~ainci/public/verteiltesysteme/servlet.html

[Sts04] ARBEITSBEREICH SOFTWARESYSTEME, Technische Universität Hamburg-Harburg, Website, Juni 2004.
http://www.sts.tu-harburg.de

[Sun04a] FOX, Joshua – Article: *When is a Singleton not a Singleton* ?
http://developer.java.sun.com/developer/technicalArticles/Programming/singletons (Januar 2001).

[Sun04b] J2EE – *Java Servlet Technology*, Website , Juni 2004
http://java.sun.com/products/servlet

[Sun04c] *Java Web Start Technology* Version 1.0.1, Website, Juni 2004
http://java.sun.com/products/javawebstart/index.html

[Sun04d] JAVA APPLET TECHNOLOGY, Website, Juni 2004
http://java.sun.com/applets

[Tig04]] TIGRIS.ORG – Open Source Software Engineering, *Project argouml*, Website, Juni 2004
http://argouml.tigris.org

[Ton04] T-ONLINE, horizontales Portal, Website, Juni 2004
http://www.t-online.de

[Tuhh04] TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG, Website, Juni 2004.
http://www.tuhh.de

[Uml14Di] DOCUMENT TYPE DEFINITION – DTD for UML 1.4 plus Diagram Interchange, Dateilink von der Gentlerware-Homepage, Juni 2004
http://www.gentleware.com/support/dev/uml14di_dtd.zip

[Vis04] VISUAL UML, Website, Juni 2004
http://www.visualobjectmodelers.com

[WC04] *The Simple Object Acces Protocol (SOAP) 1.1 und 1.2*, W3C recommendation, Website, Juni 2004
http://www.w3.org/TR/SOAP

[WDAV04] WEB DAV RESSOURCES, Website, Juni 2004
http://www.webdav.org

[Wegn00] WEGNER, Holm. *Der infoAsset Broker: Architektur, Anpassung & Erweiterung*, infoAsset AG, Hamburg, 2000.

[Wegn02] WEGNER, Holm. *Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement*, Technische Universität Hamburg-Harburg, Dissertation, 2002.

[WfMC] WORKFLOW MANAGEMENT COALITION, Website, Juni 2004.
http://www.wfmc.org

[WfMC95] HOLLINGSWORTH, David – The Workflow Reference Model, WfMC – Specification, Winchester Hamshire, United Kingdom, Janaur 1995

[WfMC99] WORKFLOW MANAGEMENT COALITION – Terminology & Glossary, WfMC – Specification, Winchester Hamshire, United Kingdom, Februar 1999

[Wips04] FORSCHUNGSPROJEKT WIPS, Website, Juli 2004.
http://www.wipsnet.de

[WVJS02] WIKLUND, A.; VAGSNES, K.; JOHANSEN, Y.; SOLHEIM, B. – *Generating Code from XMI*, November 2002

[Xmi] OBJECT MANAGEMENT GROUP – *OMG XML Metadata Interchange (XMI) Specification*, Version 1.2, formal/02-01-01 (Veröffentlicht in Januar 2002)

[Xpdl] WORKFLOW MANAGEMENT COALITION -*Workflow Process Definition Interface -- XML Process Definition Language*. Document Number WFMC-TC-1025, Document Status – 1.0 Final Draft, (Veröffentlicht am 25. Oktober 2002)

[Ya02] YANG, Shirley – *Reengineering Business Processes at STS Research Group*, Student Project, Technische Universität Hamburg-Harburg, Mai 2002

[Yah04] YAHOO, horizontales Portal, Website, Juni 2004
http://www.yahoo.com

[ZHMS01] ZIEMER, S.; HUPE, P.; MATTHES, F.; SCHMIDT, J.W. – *Enterprise Service Maps: Bringing the Public Transport Map Metaphor to Work*, Hamburg, November 2001

# Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt zu haben.

Hamburg, den 24.06.2004

(Jan Eelbo, Matr.Nr. 12160, Informatik-Ingenieurwesen, TUHH)