Erweiterung und Integration des Open Source Workflow-Systems

am Beispiel eines Software-Wartungs-Prozesses

Georg-Simon-Ohm-Fachhochschule Nürnberg
Fachbereich Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Elektrotechnik
Studienschwerpunk Daten- und Informationstechnik

Diplomarbeit von

Thomas Schmidt

Betreuer:

Prof. Dr. Helmut Herold
Prof. Dr. Bruno Lurz
Dipl.-Ing. (FH) Klaas Freitag

Tag der Anmeldung: 03. Mai 2004
Tag der Abgabe: 01. Oktober 2004

Diplomarbeit Zusammenfassung

Studentin/Student (Name, Vorname):	Schmidt, Thomas
Studienschwerpunkt/Studienrichtung:	DT Sem.: 10
Aufgabensteller:	Prof. Dr. Helmut Herold
Durchgeführt in (Firma):	SUSE LINUX AG
Betreuer in Firma / Behörde:	Klaas Freitag
Ausgabedatum: 03.05.2004	Abgabedatum: 01.10.2004
	Semester der Abgabe: WS 04/05

Thema:

Erweiterung und Integration des Open Source Workflow-Systems SWAMP am Beispiel eines Software-Wartungs-Prozesses.

Zusammenfassung:

Diese Arbeit beschäftigt sich mit der Entwicklung und Einführung des Workflow-Management-Systems **SWAMP** (**S**uSE **W**orkflow **a**nd **M**anagement **P**latform) in der Entwicklungsabteilung der Firma SUSE LINUX AG. Insbesondere soll die Integration des Software-Wartungs-Prozesses (*Maintenance*) in dieses System beschrieben werden. Das System SWAMP dient dazu, die internen Arbeitsabläufe der Firma SUSE zentral zu verwalten und ein Benutzer-Interface zur Durchführung der Abläufe bereitzustellen. Der Hauptgedanke des Projektes ist es, die inhomogene, gewachsene Umgebung verwendeter Tools durch SWAMP abzulösen und eine einheitliche Plattform zu schaffen, die selbständig mit anderen Programmen und Datenbanken der Entwicklungsabteilung in Kontakt treten kann. Somit können die Prozesse teilweise automatisiert unterstützt werden. Dabei soll eine möglichst allgemeine Lösung erstellt werden, die verschiedene Prozess-Typen auf Basis von in XML kodierten Beschreibungen durchführen kann. Die technische Grundlage der Anwendung ist der in Java implementierte SWAMP-*Kern*, der die Verwaltung der Prozesse und die Funktionalität beinhaltet. Teil dieser Arbeit ist auch die Implementierung des Web-Interfaces (WebSWAMP) zur Bedienung der Prozesse. Hierzu wird das Web-Framework Turbine verwendet, das die Funktionalität des SWAMP-Kerns über eine Web-Oberfläche zur Verfügung stellt.

Schlüsselworte:

Workflow-Management, Java, Jakarta Turbine, Maintenance, SWAMP

Inhaltsverzeichnis

1. Einleitung

Um die internen Abläufe einer Firma reproduzierbar und transparent zu gestalten, ist ihre Ablauforganisation normalerweise in verschiedene arbeitsteilige Geschäftsprozesse und Vorgänge gegliedert. Im Gegensatz zu Beschreibungen auf einem hohen Abstraktionsniveau, wie z. B. die Zertifizierung nach ISO9000[1], werden hier Abläufe hohen Detaillierungsniveaus dargestellt. Ein solcher Prozess, der mit einem IT-System abgebildet und automatisiert werden kann, nennt sich ***Workflow***:
„Die (teilweise) Automatisierung eines Geschäftsprozesses zur Übergabe von Dokumenten und Informationen nach vorgegebenen Regeln von einem Teilnehmer zum anderen, um eine bestimmte Aufgabe zu erfüllen." [2]
Der Workflow definiert die Arbeitsabläufe einer Prozesskette und die Zuordnung von Personal- und IT-Ressourcen zu den verschiedenen Stationen des Prozesses. Ein Workflow-Management-System bietet die Laufzeitumgebung für verschiedene Workflows, es steuert und kontrolliert den Ablauf der in ihm gestarteten Workflow-Instanzen.

Diese Arbeit beschäftigt sich mit der Entwicklung und Einführung des Workflow-Management-Systems **SWAMP** (**S**uSE **W**orkflow **a**nd **M**anagement **P**latform) in der Entwicklungsabteilung der Firma SUSE LINUX und insbesondere mit der Integration des *Maintenance*[3]-Prozesses in dieses System.

1.1 Überblick

Das System SWAMP dient dazu, die internen Arbeitsabläufe der Firma SUSE zentral zu verwalten und ein Benutzer-Interface für ihre Durchführung bereitzustellen. Die Mitarbeiter sollen vom System bei ihren Aufgaben unterstützt werden und einen Nutzen aus der zentralen Datenhaltung ziehen. Die Verwendung einer zentralen Anwendung ist vorteilhaft, z. B. existiert auf der einen Seite eine einheitliche Oberfläche zur Bedienung der verschiedenen Workflows durch den Benutzer und auf der anderen Seite eine gebündelte Entwicklungsarbeit, da nur ein Tool entwickelt wird.

Bislang wurden verschiedene kleine firmeninterne Tools genutzt, um spezielle Prozesse der Entwicklung zu bedienen, z. B. der *CD-Tracker*, der den Arbeitsablauf bei der Herstellung einer Distributions[4]-CD steuert, der *Translationtracker*, der den Ablauf einer Übersetzung von Dokumentationen unterstützt, oder der *Laufzettel*, ein Prozess, der die Entwicklung von Patches[5] begleitet (siehe **??**).

Der Hauptgedanke des Projektes ist es nun, diese inhomogene, gewachsene Umgebung von Tools, die hauptsächlich mit „althergebrachten" Mitteln wie z. B. Mailinglisten arbeiten, durch SWAMP

[1] http://de.wikipedia.org/wiki/ISO_9000
[2] nach [Coal00]
[3] dt.: Software-Wartung
[4] Zusammenstellung von Softwarepaketen, die dann z. B. als SUSE LINUX 9.2 herausgegeben werden.
[5] Korrekturen für in Software enthaltene Fehler.

abzulösen und eine einheitliche Plattform zu schaffen, die selbständig mit anderen Programmen und Datenbanken der Entwicklungsabteilung in Kontakt treten und somit Workflows teilweise automatisiert unterstützen kann. Dabei soll eine möglichst generische (allgemeine) Lösung erstellt werden, die auf Basis von verschiedenen Workflow-Definitionen in der Lage ist, die ganze Bandbreite der benötigten Funktionalität zur Verfügung zu stellen. Es folgt eine Beschreibung der Tools, die in der Entwicklung zum Einsatz kommen und mit denen eine Kommunikation, z. B. ein gegenseitiges Benachrichtigen bei bestimmten Ereignissen oder das Abfragen von Daten, angestrebt wird:

- **Bugzilla**[6]
  Bugzilla ist ein Tool zur Verwaltung von gefundenen Fehlern (Bugs) und Änderungswünschen in Programmen. Ursprünglich wurde es im Rahmen des Mozilla-Projekts[7] entwickelt. Die Implementierung in Perl, einer plattformunabhängigen Skriptsprache, und seine gute Erweiterbarkeit haben inzwischen dazu geführt, daß Bugzilla in vielen Open Source Projekten zur Fehlerverfolgung eingesetzt wird. Beispiele hierfür sind die Bugtracking-Systeme von KDE[8], GNOME[9], und das hier beschriebene SUSE Bugzilla-System[10]. Benutzer können in Programmen gefundene Fehler mit Hilfe von Bugzilla erfassen und eine genaue Beschreibung inkl. des betroffenen Produktes, der Hardwareplattform und einer Priorität eingeben. Wenn der Bug erfasst ist, wird automatisch der in der Firma für das betroffenen Software-Paket Verantwortliche (der *Package Maintainer*) benachrichtigt. Dem Maintainer bietet das Bugzilla-System eine Übersicht über die Bugs, die ihm zugeordnet sind, und die Möglichkeit, falsche Zuordnungen zu ändern, nicht zutreffende oder bereits anderweitig gelöste Bugreports zu löschen und den Status von Bugs zu ändern, an denen er gerade arbeitet. Auf der für jeden sichtbaren Statusseite eines Bugs ist es möglich, Kommentare und zusätzliche Informationen, z. B. Screenshots und Logfiles, zu hinterlegen. Dies hilft zu klären, unter welchen Umständen der Fehler auftrat. Bugzilla ist somit das System, das die zentrale Koordinierung der Entwicklungsarbeit ermöglicht.

- **PDB**[11]**(Paketdatenbank)**
  Hier werden alle Meta-Informationen zu den Softwarepaketen der SUSE LINUX Distributionen gespeichert. Momentan beinhaltet die PDB die Paketdatenbasis der über 4000 Softwarepakete, aus denen sich dann die verschiedenen SUSE LINUX Produkte zusammensetzen. Typischerweise enthält ein Eintrag Informationen über den Autor der Software, den Maintainer innerhalb der Firma und produktspezifische Daten wie Lizenzen, unterstützte Hardware, die aktuelle Version und eine Beschreibung. Eine Kommunikation zwischen der PDB und SWAMP könnte das Benachrichtigen über Änderungen des Status oder der Attribute eines der Pakete in der Datenbank sein.

- **Das Autobuild-System**
  Da ein neues Release einer SUSE LINUX Distribution die zum Releasezeitpunkt aktuellsten Pakete enthalten soll, müssen die Pakete dieser Programme regelmäßig von den Entwicklern auf den neuesten Stand gebracht und mit Patches versehen werden. Der Vorgang, alle Pakete immer wieder zu kompilieren und dabei ihre Abhängigkeiten untereinander zu berücksichtigen, ist die Aufgabe des Autobuild-Systems. Um diesen Prozess automatisiert ablaufen zu lassen, sind auf vielen Arbeitsplatz-Rechnern und dedizierten Build-Servern Hintergrundprozesse installiert, die Kompilieraufträge des Autobuild-Systems entgegennehmen. Es entsteht also ein über die ganze Firma verteilter Park von Rechenleistung mit einem speziellen Rechner-Verbund im Zentrum, der vom Autobuild-Team benutzt wird, um den Prozess zu

[6] http://www.bugzilla.org
[7] http://www.mozilla.org
[8] http://bugs.kde.org/
[9] http://bugzilla.gnome.org/
[10] http://buzilla.suse.de (nur mit Zugangsberechtigung)
[11] Package Database

steuern und zu überwachen. Am Ende eines Autobuild-Durchgangs hat das System alle Dateien erstellt, die notwendig sind, um ein Produkt auf CD/DVD zu erstellen. Eine mögliche Nachricht des Autobuild-Systems an SWAMP wäre z. B. das Fehlschlagen der Kompilierung eines Pakets mit Informationen über das Paket, die Zielplattform, und den Package-Maintainer.

- **KoTrus**[12]
  Dieses Tool wird in der Abteilung *Research and Development* eingesetzt, um den Aufwand für verschiedene Projekte und Aufgaben festzuhalten, damit eine Transparenz gegenüber dem Controlling des Konzerns sichergestellt werden kann und die Personalressourcen optimal verteilt werden können. Jeder Mitarbeiter trägt am Ende einer Woche ein, für welches Projekt er wieviel Zeit aufgewendet hat.

Jedes dieser Systeme verarbeitet Informationen und stellt Informationen zur Verfügung. In einem typischen Arbeitsprozess müssen einige dieser Tools befragt werden und dann die richtige Aktion ausgeführt werden. Hierbei soll das Workflow-System anhand definierter Workflows helfen und dem Benutzer Arbeit abnehmen, indem es einen Teil der Aufgaben automatisch erledigt und logische Schlußfolgerungen selbständig im Hintergrund trifft. Durch Kombination mehrerer Informationen aus den beschriebenen Systemen kann sich eine viel wertvollere Information ergeben, da z. B. Informationen über dasselbe Paket sowohl in der PDB als auch in Bugzilla oder im Autobuild-System zu finden sein können. Ein weiteres Ziel von SWAMP ist es, den einzelnen Benutzern eine konfigurierbare Oberfläche zu bieten, um den enormen Nachrichtenausstoß der oben genannten Systeme zu filtern und nur die im jeweiligen Workflow-Kontext wichtigen Informationen darzustellen, d. h. zu verhindern, daß man von unwichtigen Nachrichten *überschwemmt* wird.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist die Erweiterung der SWAMP-Workflow-Engine um im täglichen Gebrauch benötigte Features und die Integration des Maintenance-Prozesses in das System. Dazu gehört auch die genaue Identifikation des Prozesses zusammen mit den verantwortlichen Mitarbeitern und das darauf folgende Erstellen eines Workflows. Es muß herausgefunden werden, inwiefern der Prozess oder das Workflow-Management-System verändert oder möglicherweise erweitert werden mus, um eine Integration zu ermöglichen. Ist es nicht möglich, den kompletten Prozess sofort abzubilden, dann ist es nötig, bestimmte Teilbereiche festzulegen, die schrittweise in SWAMP einfließen werden. Die SWAMP-Engine soll möglichst nicht auf spezielle Anwendungsfälle einzelner Workflows hin angepasst werden, sondern eine generische Umgebung zur Verfügung stellen, in die verschiedene Workflow-Definitionen eingelesen werden können. Die Bereitstellung der Workflow-Schemata erfolgt über XML-Dateien, die in einer definierten Syntax serverseitig vorliegen müssen. Am Ende der Integration des Maintenance-Workflows steht eine solche Workflow-Definition, die eine im Zuge dieser Arbeit eingebrachte Funktionalität benutzt. Das erstellte System soll vom Maintenance-Team produktiv im Tagesgeschäft eingesetzt werden.

[12] Abkürzung für „Kosten Tracking for us“

1.3 Vorgehensweise

Der Ablauf dieser Arbeit mit dem letztendlichen Ziel der Integration des Maintenance-Workflows in SWAMP gliedert sich folgendermaßen:

- Kennenlernen des Maintenance-Prozesses und der mit ihm verbundenen Systeme
- Erstellen einer Anforderungsspezifikation mit Hilfe des *Volere*-Templates
- Untersuchen der Grundlagen von Workflow-Management-Systemen und Übertragen dieser Grundlagen auf das Workflow-Management-System SWAMP
- Erstellen eines Überblicks über die technologischen Grundlagen und Möglichkeiten der verwendeten Infrastruktur
- Umsetzen der theoretischen Grundlagen in der Anwendung SWAMP und Implementierung der für den Maintenance-Workflow notwendigen Funktionalität
- Überführen des Maintenance-Prozesses in einen SWAMP-Workflow und seine Integration ins System

2. Analyse

Das Kapitel der Analyse des Maintenance-Prozesses der Firma SUSE ist nicht öffentlich. :-(

2.1 Anforderungsspezifikation nach „Volere“

Das Anforderungsmanagement spielt bei einem Software-Projekt eine große Rolle. Denn die Anforderungs- und Entwurfsfehler können leicht zu einem Mißerfolg führen können, da sie meist erst relativ spät in der Betriebsphase des Produkts bemerkt werden. Je später sie bemerkt werden, desto größer ist der Kosten- und Arbeitsaufwand, sie zu beheben. Die Ursache dieser Fehler liegt oft in unklaren Vorstellungen von der Funktionalität des Produkts, da die beteiligten Parteien oft nicht die gleiche Sprache sprechen und sehr unterschiedliche Vorstellungen haben. Aus diesem Grund ist es wichtig, die Anforderungen schriftlich zu definieren und zur Grundlage des Geschäftsverhältnisses zu machen. Ein *Requirement* ist eine einzelne, klar definierte Anforderung an das System, die keinen Interpretationsspielraum zuläßt. Um herauszufinden, welche Anforderungen überhaupt genau bestehen, muß eine Systemanalyse durchgeführt werden, die diese Anforderungen Schritt für Schritt ermittelt, das sog. *Requirements Engineering*. Das Ergebnis dieser Analyse ist das Pflichtenheft oder die Anforderungsspezifikation. Eine mögliche Art des Requirement Engineerings ist das von der Atlantic Systems Guild[1] eingeführte Requirements Specification Template *Volere* [J.Ro99]. Die folgende Anforderungsspezifikation wurde mit Hilfe (in verkürztem Umfang) der *Volere Templates*[2] erstellt, da sie eine sehr gute Grundlage zur Erfassung und Strukturierung der *Requirements* bilden.

2.1.1 Randbedingungen für das Produkt

Ausgangssituation

Das Wissen über den Ablauf des Maintenance-Prozesses ist in der SUSE Linux AG derzeit nicht strukturiert, zentral gebündelt oder abrufbar aufbereitet. Der Ablauf der Maintenance und des L3-Supports sind nicht eindeutig definiert, bzw. ist den Mitarbeitern nur ihr jeweiliger Teilprozess bekannt, weshalb es oft zu Problemen kommt. Es werden innerhalb eines Maintenance-Vorfalles unterschiedlichste Systeme (E-Mail, Web, Telefon) eingesetzt, was einen Systembruch darstellt. Dadurch entstehen Nachteile der Inkombatibilität, mehrfache Datenhaltung und ein insgesamt größerer Aufwand. Da der Maintenance-Prozess eine hohe Priorität hat und sein reibungsloser Ablauf sehr wichtig im Hinblick auf eine bestmögliche Kundenbetreuung ist, soll er künftig mit Hilfe der Anwendung SWAMP verwaltet werden und unter einer einheitlichen Oberfläche zur Verfügung stehen. Auf Basis des Wissens der Mitarbeiter wird der Prozessablauf in SWAMP abgebildet, um das Know-How über den Ablauf in ein konfigurierbares Workflow-Management-System zu verlagern. Dadurch entsteht eine größere Klarheit bei der Bearbeitung eines Maintenance-Falles, und es lassen sich weitere Funktionen des Systems nutzen. Beispiele hierfür sind Statistik- und Status-Darstellungsmöglichkeiten, die den gesamten Ablauf transparenter machen. Desweiteren soll es möglich sein, Aufgaben, die bislang manuell erledigt wurden, zu automatisieren.

Ziel des Projekts

Durch die Weiterentwicklung des Systems SWAMP und die Integration des bisherigen *Laufzettel*-Teilprozesses der Maintenance in dieses System soll der momentan unbefriedigende Zustand behoben und die Motivation der Mitarbeiter durch den Einsatz dieses neuen Tools gesteigert werden. Es soll, bei niedriger Redundanz, ein effizienter Umgang mit den in SWAMP vereinigten Daten verschiedener Systeme möglich sein. Folgende Hauptziele sollen durch die Integration erreicht werden:

- Systembrüche sollen soweit wie möglich vermieden werden. Dort wo dies nicht möglich ist, soll die Kommunikation mit anderen Systemen selbständig im Hintergrund abgewickelt werden. Der Benutzer sollte den Workflow vollständig mit SWAMP durchführen können.

[1] http://www.atlsysguild.com
[2] http://www.volere.co.uk/

- Bereitstellung einer detaillierten, konfigurierbaren Statusübersicht mit Statistikfunktionalität.
- Möglichst geringe Fehleranfälligkeit, durch Intelligenz des Systems zur Vermeidung von falschen/unvollständigen Benutzereingaben.
- Eine höhere Transparenz und Übersicht über den Prozess durch anwender- (rollen-)spezifische Ansichten.

Klienten, Kunden und weitere Beteiligte, Betroffene

Klienten

Das Programm SWAMP wird innerhalb der Abteilung *Internal Tools* für die Abteilung R&D (Research and Development) der Firma SUSE LINUX AG entwickelt.

Kunden

Das System wird momentan vorrangig für die Abteilungen Internal-Tools und Maintenance-Support entwickelt, da hier bereits ein großer Bedarf an einem System dieser Art besteht. Durch die Entwicklung auf einer generischen Basis ist es möglich, Geschäftsprozesse anderer Abteilungen abzubilden oder zukünftige Erweiterungen hinzuzufügen, falls der Bedarf besteht. Durch die Freigabe des Projekts unter der *GNU Public License*[3] ist es möglich, das Programm inklusive Sourcecode kostenfrei aus dem Internet herunterzuladen und eigene Erweiterungen einzubauen. Dadurch wird der Kundenkreis erweitert. Ziel dieser Arbeit ist jedoch hauptsächlich, die benötigte Funktionalität für den Level3-Support und das Maintenance-Team zur Verfügung zu stellen, da sie den primären Nutzerkreis darstellen. Die hier geschilderten Anforderungen wurden in zahlreichen Gesprächen mit Mitarbeitern dieser Abteilungen ermittelt und miteinander abgestimmt.

Weitere Beteiligte und Betroffene

Bei Entwicklung, Abnahme und Betrieb sind neben Prof. Dr. Helmut Herold (Georg-Simon-Ohm Fachhochschule Nürnberg), dem Betreuer der vorliegenden Arbeit, folgende Mitarbeiter der SUSE LINUX AG involviert:

- Der Projektleiter des Projekts SWAMP
- Mitglieder der Abteilung *Internal-Tools*
- Der Maintenance-Koordinator
- Der Teamleiter des SDI(SuSE Development Interface)-Supports

Nutzer des Produkts

Das System sieht mehrere Benutzergruppen vor:

- Workflow-Admins: Der Workflow-Admin ist der verantwortliche Ansprechpartner für alle Workflow-Instanzen eines bestimmten Typs, z. B. ist der Maintenance-Koordinator der Workflow-Admin für alle Maintenance-Workflows.
- Workflow-Owner: Der Workflow-Owner ist derjenige, der einen bestimmten Workflow gestartet hat. Er besitzt erweiterte Möglichkeiten, diesen Workflow zu beeinflussen, z. B. ihn zu beenden.

[3] http://www.gnu.org/copyleft/gpl.html, deutsche Version: http://www.gnu.de/gpl-ger.html

- Task-Owner: Der Task-Owner ist eine Person oder Personengruppe, der innerhalb eines Workflows eine bestimmte Aufgabe (Task) zugewiesen wurde, somit der normale Systembenutzer. Er muß sich ins System einloggen und bekommt die Informationen über seine Aufgaben auf der Oberfläche angezeigt. Das Ergebnis wird dann vom Task-Owner dort wieder eingetragen. Dem Task-Owner werden im Normalfall nur für ihn relevante Informationen auf der Oberfläche dargestellt. Er benötigt keine Kenntnisse über Interna des Workflows.

Beim Einsatz des Tools innerhalb der Firma SUSE kann bei allen Benutzern von einer sehr hohen technischen Affinität und einem umfangreichen Hintergrundwissen ausgegangen werden. Somit sind keine Probleme bei der Bedienung der Anwendung zu erwarten.

Im Spezialfall des Maintenance-Workflows konnten folgende Benutzergruppen identifiziert werden (siehe auch **??**):

- Projektmanager (Task-Owner)
- Security-Team (Task-Owner)
- Maintenance-Team (Workflow-Admin, Workflow-Owner)
- QA(Quality Assurance)-Team (Task-Owner)
- Support-Team (Task-Owner)
- Paket-Maintainer (Task-Owner)
- Autobuild-Team (Task-Owner)

Prioritäten der Benutzer

Das größte Nutzungsinteresse am System SWAMP haben die Mitglieder des Maintenance- und Level3-Support-Teams, da es sich hierbei um einen Workflow handelt, der ihre täglichen Arbeitsabläufe abbildet, und das bisherige Hilfsmittel *Laufzettel* ablösen soll. Diese Gruppe wird als Schlüsselbenutzer des Produkts betrachtet und erhält die höchste Priorität bei der Anforderungsdefinition. Die weiteren Benutzergruppen, die jeweils nur mit einem Teilaspekt des Maintenance-Workflows in Verbindung stehen, haben auch eine hohe Priorität, da der Erfolg des Systems nicht zuletzt von der Akzeptanz innerhalb dieser Gruppen abhängt. Da diese Gruppen aber durchaus sehr verschiedene und evtl. gegenteilige Anforderungen an das System haben, hat die Umsetzung ihrer Anforderungen eine geringerere Priorität als die der ersten Gruppe. Wünsche von Benutzern anderer (geplanter) Workflows des Systems SWAMP haben nur geringe Relevanz für diese Arbeit, da hier hauptsächlich auf einen funktionierenden Maintenance-Workflow hingearbeitet wird.

Beiträge der Benutzer

Die Erfassung der Anforderungen erfolgt durch Meetings mit den Vertretern der jeweiligen Benutzergruppen, insbesondere des Maintenance-Teams. Zusammen mit dem Maintenance-Team und dem L3-Support-Team wurden Grundlagen für die Erstellung eines Workflows und notwendige Erweiterungen am System SWAMP erörtert. Durch wöchentliche Präsentation der Erweiterungen und Prototyp-Workflows wird das Erfüllen der Anforderungen regelmäßig überprüft.

Randbedingungen für das Projekt

Produktvorschriften Der Prozess muß mit SWAMP abgebildet und bearbeitet werden können. Die dazu benötigten Erweiterungen sind im SWAMP-Code vorzunehmen, wobei die Workflow-Engine SWAMP unabhängig von den Eigenheiten einzelner Workflows gehalten werden muß, d. h. es dürfen keine zu speziellen Funktionen eingebaut werden. Die Workflow-Definition erfolgt

einzig über das XML-Template. Die verwendete Programmiersprache des SWAMP-Kerns ist JAVA (siehe 4.3.1) mit einem MySQL-Datenbank-Backend. Die primär zu erstellende Benutzeroberfläche ist per Browser erreichbar und sollte dem HTML 4.01 Standard entsprechen, um eine maximale Kompatibilität mit allen Browsern zu gewährleisten. Weiterhin muß die Möglichkeit bestehen bleiben, auch andere Zugriffsmöglichkeiten in Form von Clientprogrammen zu erstellen, die über dieselben Schnittstellen Zugriff auf den vollen Funktionsumfang ermöglichen.

Entwicklungsumgebung Das Zentrum des SWAMP-Systems bildet der SWAMP-Kern (siehe 5.1). Er wird mit dem momentan aktuellen Java Framework 1.4.2 entwickelt und ausgeführt. Für die Ausführung des WebSWAMP-Servers wird ein Servlet-Container der J2EE Servlet Spezifikation 2.4 benötigt. Als Datenbank wird momentan ein MySQL-Server eingesetzt, die Migration auf eine andere Datenbank gestaltet sich durch den Einsatz eines JDBC-Treibers aber unproblematisch. Dank der Platformunabhängigkeit der Programmiersprache Java wird an das Betriebssystem und die Hardwareplatform nur die Anforderung der Verfügbarkeit eines Java Runtime Environments und einer Datenbank mit JDBC-Treiber gestellt. Für die SUSE-interne Installation von SWAMP ist allerdings bereits klar, daß sie auf einem SUSE LINUX Enterprise Server mit MySQL-Datenbank betrieben werden wird.

Die Entwicklung der Anwendung erfolgt in der folgenden Arbeitsplatzumgebung:
Betriebssystem: SUSE LINUX 9.1,
Entwicklungsumgebung: Eclipse[4] 3.0M8, Datenbank: MySQL 4.0.18,
Java-Laufzeitumgebung: Jakarta Tomcat 5.0.19 mit einer Java VM 1.4.2

Partnerapplikationen Partnerapplikationen, mit denen ein Informationsaustausch im Rahmen des Maintenance-Workflows stattfindet, wurden unter 1.1 und **??** detailliert beschrieben, hauptsächlich sind dies:
- Autobuild-System
- Bugzilla

Die auf technischer Seite für das System notwendigen Partnerapplikationen werden im Kapitel 4 ausführlich beschrieben.

Verwendung kommerziell erwerblicher Softwarekomponenten Es kommen keine kommerziellen Komponenten zum Einsatz, da das Projekt der SUSE-Philosophie folgend vollständig auf freier Software basiert. Erklärungen zum Konzept freier Software finden sich im Abschnitt 4.4.

Erwartete Arbeitsplatzumgebung Der Zugriff auf die Anwendung wird in der ersten Phase ausschließlich über Web-Browser auf den Arbeitsplatz-Rechnern der Benutzer stattfinden. Die Wahl des Browsers sollte dem Benutzer freigestellt sein, bei den meisten Arbeitsplätzen dürfte als Betriebssystem SUSE LINUX ab Version 9.0 mit KDE oder GNOME als Windowmanager und Konqueror oder Mozilla als Browser verwendet werden. Durch die lokale Auswertung von HTTP-Anfragen mit Skripten kann es den Benutzern auch ermöglicht werden, sich an ihrem Arbeitsplatz eine lokale Oberfläche für die Anwendung zu erstellen.

Zeitrahmen Das System ist soweit wie möglich bis Ende August 2004 fertigzustellen, inklusive der funktionstüchtigen Integration des Maintenance-Workflows.

Budget Das Projekt dien dem Verfasser zur Erstellung einer Diplomarbeit an der Georg-Simon-Ohm Fachhochschule, daher können die Kosten vernachlässigt werden. Der personelle Aufwand beläuft sich auf sechs Personenmonate.

[4] http://www.eclipse.org

Namenskonventionen und Definitionen

In Tabelle 2.1 auf Seite 14 werden die in dieser Arbeit verwendeten Begriffe bzw. Abkürzungen erläutert, um Fehlinterpretationen zu vermeiden. Die Definitionen sind im Kontext dieser Arbeit zu sehen und stellen keine allgemeingültigen Definitionen dar.

Relevante Fakten

Das Projekt dient dem Verfasser dieser Arbeit dazu, die Diplomarbeit im Fachbereich Nachrichtentechnik der Georg- Simon-Ohm Fachhochschule Nürnberg zu erstellen. Bei den notwendigen Erweiterungen des Workflow-Management-Systems SWAMP ist darauf zu achten, daß diese nicht nur dem Anwendungsfall des Maintenance-Prozesses zugute kommen, sondern sich auch an den Anforderungen der anderen (geplanten) Workflows orientieren, da alle unterstützten Workflows innerhalb desselben Systems ablaufen werden.

Annahmen

Entscheidend für den Erfolg des Projekts ist seine gute Aufnahme bei den Benutzern. Das Tool und die integrierten Workflows müssen sich möglichst nahtlos und unproblematisch in den Arbeitsalltag der Benutzer integrieren lassen. Das Tool ist erst erfolgreich, wenn die Mitarbeiter durch seinen Einsatz bessere Information erhalten und durch die klare Strukturierung eines Workflows Zeit einsparen.

2.1.2 Funktionale Anforderungen

Abgrenzung des Systems

Im nachfolgenden Kontextdiagramm (siehe Abbildung 2.1) wird dargestellt, welche Akteure mit dem System in Interaktion stehen und auf welche externen Ereignisse das System reagieren kann.

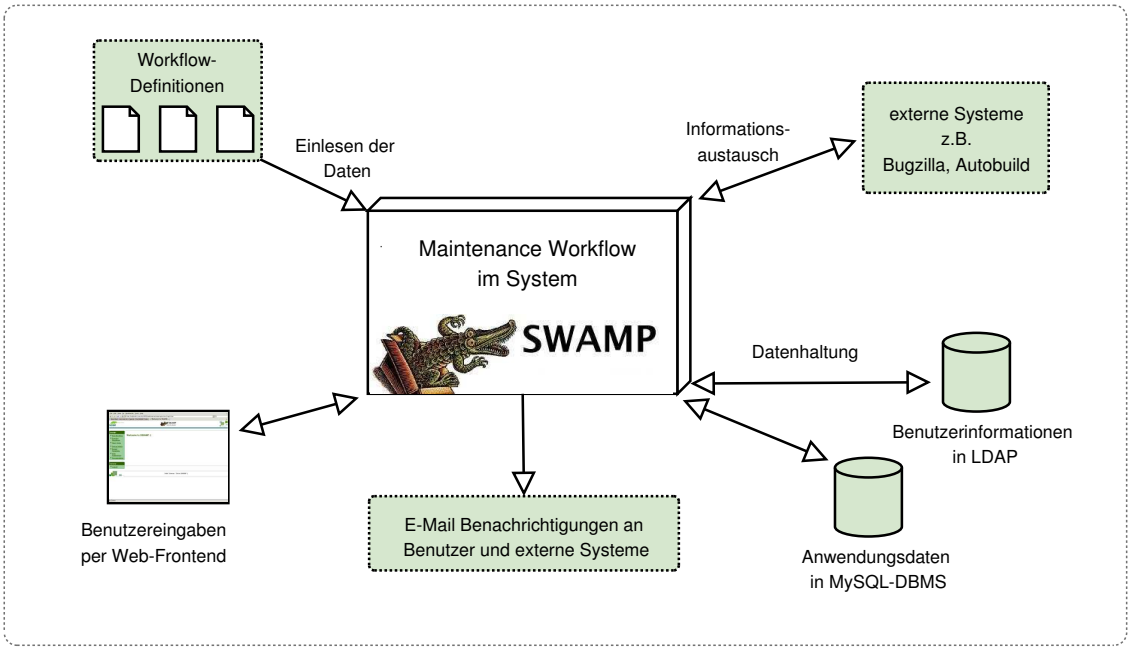


Abbildung 2.1: Der Kontext des Produkts

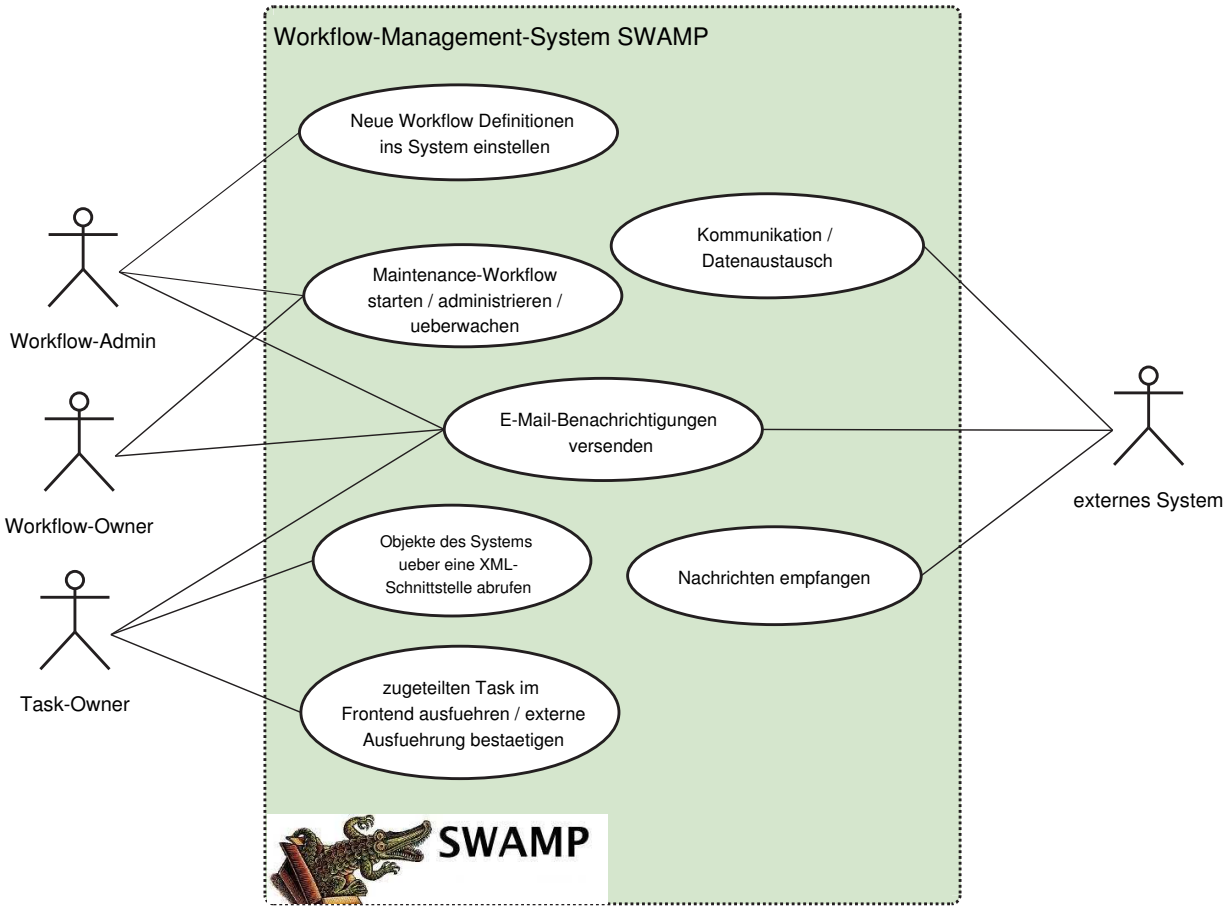


Abbildung 2.2: Anwendungsfalldiagramm (Use Cases)

Anforderungen an Funktionen und Daten des Systems

Das Anwendungsfalldiagramm in Abbildung 2.2 beschreibt die Zusammenhänge zwischen den (allgemeinen) Anwendungsfällen und den beteiligten Akteuren. Die zugehörigen Anwendungsfälle werden in Tabelle 2.2 näher erklärt. Die Funktionalität des Systems ergibt sich aus der Gesamtheit dieser Geschäftsprozesse.

2.1.3 Nicht funktionale Anforderungen

Oberflächenanforderungen

Die Standard-Oberfläche der Anwendung soll sich an die Corporate Identity der Firma SUSE anlehnen. Da die Möglichkeit spezieller Oberflächen für einzelne Workflows besteht, kann das Layout der einzelnen Workflows nach den Bedürfnissen der Abteilungen ausgerichtet werden. Die Oberfäche soll in Bezug auf die dargestellten Daten teilweise vom Benutzer selbst konfigurierbar sein, da jeder Benutzer andere Anforderungen an die Darstellung hat. Die zu erstellende Oberfläche erweitert die Standard-Oberfläche der Anwendung um spezielle Ansichten, die es ermöglichen, maintenance-workflow-spezifische Informationen einzublenden. Auf eine mehrsprachige Verfügbarkeit der Oberfläche wird vorerst kein Wert gelegt. Sie wird komplett in Englisch erstellt, da auch firmenintern viele nicht-deutschprachige Benutzer Zugriff haben.

Sicherheitsanforderungen

Der Zugriff auf Daten des Systems SWAMP soll nur mit einer gültigen SUSE Mitarbeiterkennung möglich sein, die beim Login mit Hilfe des zentralen LDAP-Servers überprüft wird. Durch den Login ist es möglich, jede Benutzeraktion innerhalb des Systems, z. B. das Ändern von Daten oder die

Ausführung einer Aufgabe, ihrem jeweiligen Verursacher zuzuordnen. In der zu erstellenden Version von SWAMP sind noch keine expliziten Berechtigungen für bestimmte Aktionen im System vorgesehen, d. h. jeder Benutzer kann theoretisch auch Aufgaben anderer Nutzer durchführen. Mit einer zunehmenden Vergrößerung des Nutzerkreises und einer größeren Anzahl an unterschiedlichen Workflows wird es jedoch notwendig, ein granulares Berechtigungskonzept einzuführen.

Zuverlässigkeit und Verfügbarkeit

Das System muß im 24h Betrieb verfügbar sein, was durch die Installation auf einem dedizierten Server im Rechnerpark der Entwicklungsabteilung sichergestellt werden kann. Da im Falle des Maintenance-Workflows die Arbeit des Maintenance-Teams und der im Workflow referenzierten Benutzergruppen direkt von der Erreichbarkeit und der ordnungsgemäßen Funktion des Systems abhängt, ist es sehr wichtig, diese dauerhaft anzubieten. Der in der Datenbank abgebildete Systemzustand muß regelmäßig gesichert werden, um im Falle eines Soft- oder Hardwaredefekts möglichst schnell den letzten gesicherten Zustand des Systems wieder herstellen zu können.

Kapazitätsanforderungen

Es wird davon ausgegangen, daß innerhalb der Enwicklungsabteilung eine Maximallast von 150 gleichzeitig im System angemeldeten Benutzern und bis zu 500 aktiven Workflow-Instanzen auftreten. Kapazitätsengpässe sind hier hauptsächlich in der (Hardware-)Performance und Kapazität, insbesondere Hauptspeicher, des Servers zu erwarten, auf dem die Anwendung und die Datenbank ausgeführt werden.

Anforderungen an die Integrität der Daten

Die Integrität der Daten von SWAMP ist essentiell, da im Fehlerfall Workflows nicht korrekt oder nicht ablaufen können. Da die Aufgaben der Datenhaltung im Hintergrund durch eine Datenbank übernommen werden, muß hier einerseits versucht werden, inkonsistente Daten der Anwendung abzuweisen und andererseits eine ständige Verfügbarkeit der Daten sicherzustellen. Dies kann durch Einsatz datenbankspezifischer Funktionalität (z. B. Foreign Keys, COMMIT/RESTORE Statements) und regelmäßige Backups der Daten geschehen.

Auf Anwendungsebene muß die Integrität der Daten ebenso sichergestellt werden, sowohl beim Lesen als auch beim Schreiben, und im Fehlerfall dem Benutzer und den SWAMP-Entwicklern kommuniziert werden.

Gesetzliche Anforderungen

Die vom System erfaßten und nutzerspezifisch auswertbaren Daten fallen unter den Schutz des Betriebsverfassungsgesetzes. So ist es z. B. nicht zulässig, genaue Übersichten darüber zu erstellen, wie lange die Mitarbeiter für die Bearbeitung einer Aufgabe benötigen, oder benutzerspezifische Profile zur statistischen Arbeitszeitanalyse aufzustellen.

2.1.4 Randbedingungen des Projekts

Inbetriebnahme und Migration

Das System SWAMP ist mit dem Workflow „*CD-Tracker*“ bereits zum 1.7.2004 im Testbetrieb online gegangen. Es befindet sich auf einem für alle Mitarbeiter per Intranet erreichbaren Server der Entwicklungsabteilung. In dieses System werden in regelmäßigen Abständen Updates des aktuellen Entwicklungsstands der im Zuge dieser Arbeit vorgenommenen Erweiterungen am System eingespeist. Prototypen des zu erstellenden Maintenance-Workflows werden ebenfalls regelmäßig eingespielt. Bei Fertigstellung dieser Arbeit gegen Ende August 2004 soll der erarbeitete Maintenance-Workflow in einer für alle Beteiligten zufriedenstellenden Version im System vorliegen.

Risiken

Das System SWAMP ist eine komplette Neuentwicklung und wurde noch mit keinem Workflow unter *realen*, d. h. Produktions-Bedingungen eingesetzt. Daher bestehen noch keine Erfahrungswerte, was die Akzeptanz des Tools bei seinen Benutzern angeht. Es gab bislang kaum Kritikpunkte von außerhalb des SWAMP-Entwicklungsteams, die in das Projekt miteingeflossen sind. Falls die Benutzer bei der Verwendung von SWAMP Funktionalität vermissen, die durch die Struktur der Anwendung nur sehr schwer eingearbeitet werden kann, so ist dies ein Risiko.

Ein weiteres Risiko, das durch die bislang nur begrenzt durchgeführten Tests einhergeht, ist die Unsicherheit, wie sich das System unter einer hohen Last verhält, d. h. bei vielen konkurrierenden Zugriffen, und bei einer großen Anzahl gleichzeitig laufender Workflows und damit sehr vielen Daten.

Diesen Risiken kann mit einer stufenweisen Einführung des Systems begegnet werden, wobei es gleichzeitig einen Rückfluß und eine Integration von Nutzer-Feedback und eine genaue Beobachtung des Systemverhaltens geben muß.

Benutzerdokumentation

Es wird eine Benutzerdokumentation im Docbook-Format[5] erstellt. Dieses XML-Format erlaubt den Export in unterschiedliche, für den Endanwender benutzbare Formate, wie z. B. PDF, HTML. Diese Dokumentation beinhaltet hauptsächlich allgemeine Themen zur Funktionsweise des SWAMP-Workflow-Management-Systems. Ebenso enthält es Hinweise zur Vorgehensweise bei der Erstellung von XML-Workflow-Definitionen. Die Dokumentation der Benutzeroberfläche WebSWAMP ist direkt in der Web-Oberfläche abrufbar. Hier werden dem Benutzer kontextabhängige Links zu den jeweiligen Hilfe-Seiten angeboten.

[5] http://www.docbook.org/

Bezeichnung	Abkürzung	Beschreibung
Access Control List	ACL	dt.: Zugangskontrolliste, erweitert die unter UNIX übliche Benutzer-Gruppe-Welt Aufteilung der Zugriffssteuerung um benutzerspezifische Rollen.
Distribution	Dist	Das Dist-Team innerhalb von SUSE ist für das Zusammenstellen der für eine spezielle Distribution notwendigen Pakete zuständig. Eine Distribution ist eine Zusammenstellung von Softwarepaketen, die dann z. B. als SUSE LINUX 9.2 herausgegeben wird.
	ISO 9000	Eine DIN-Norm, die einen Leitfaden für das Qualitätsmanagement innerhalb von Firmen, von der Entwicklung bis zum Kundendienst, darstellt.
Laufzettel		Die allgemeine Bezeichnung des Maintenance-Ablaufs, der bislang dadurch gekennzeichnet ist, daß die Beteiligten nacheinander Einträge in einem E-Mail Formular vornehmen.
Level3	L3	Die anspruchsvollste Art des Kunden-Supports, bei der evtl. eine Anpassung des Quelltexts notwendig werden kann.
Message Digest Algorithm 5	MD5	Ein 128 Bit langer, eindeutiger Hashwert einer Datei.
Patch		Korrektur für einen in einer Software enthaltenen Fehler.
Projektmanager	PM	Hat die organisatorische Leitung eines Projekts und ist für seine erfolgreiche Abwicklung zuständig.
Quality Assurance	QA	Die Qualitätssicherung stellt sicher, daß ein bestimmtes Produkt ein festgelegtes Qualitätsniveau erreicht.
SUSE Trouble Ticket System	STTS3	Ein E-Mail-System, in dem Anfragen automatisch klassifiziert und den jeweiligen Mitarbeitern zugestellt werden können.
SUSE LINUX	SUSE	Gesellschaft für Software- und System-Enwicklung.
SUSE Workflow and Management Plattform	SWAMP	Das Workflow-Management-System, innerhalb dessen der Maintenance-Workflow ablaufen soll.
Workflow-Management-System	WfMS	Eine (Software-)Anwendung, innerhalb derer Workflows durchgeführt und verwaltet werden.
Yet another Setup Tool	YaST	Tool zur Installation und Verwaltung von installierten Software-Paketen auf SUSE LINUX Distributionen.
YaST Online Update	YOU	Tool zum automatischen Update von Software über Netzwerke.

Tabelle 2.1: Begriffe & Abkürzungen

Ereignisname	Ereignis
Neue Workflow-Definitionen ins System einstellen	Die Workflow-Definitionen im XML-Format erzeugen oder verändern und im System verfügbar machen.
Maintenance-Workflow starten / administrieren / überwachen	Über die Oberfläche müssen Funktionen verfügbar sein, die auch außerplanmäßige Änderungen des Workflow-Ablaufs erlauben. Die Konfiguration vordefinierter und vom Nutzer selbst definierter Übersichtsseiten mit Hilfe von Filtern ist zu ermöglichen.
Objekte des Systems über eine XML-Schnittstelle abrufen	Die im System liegenden Workflows bzw. Tasks sollen in einem XML-Format abrufbar sein, um eine lokale Bearbeitung und Auswertung auf Client-Rechnern durch Skripte zu ermöglichen.
Zugeteilten Task im Frontend ausführen	Die Bedienung der Anwendung muß sich komplett über das Frontend ausführen lassen. Dieser Anwendungsfall stellt den Schwerpunkt der Benutzerschnittstelle dar, da ohne die reibungslose Bedienung der Anwendung durch die Nutzer kein flüssiger Ablauf des Workflows sichergestellt werden kann. Eine genauere Aufsplittung dieses Anwendungsfalles auf die einzelnen Benutzergruppen wurde bereits in der Analyse unter **??** erstellt.
Kommunikation / Datenaustausch	Es müssen Schnittstellen geschaffen werden, um mit externen Systemen zu kommunizieren und die gewonnenen Daten im richtigen Kontext zu präsentieren. Die Funktionen hierzu sollen möglichst allgemein gehalten werden, um nicht nur die im Maintenance-Prozess notwendigen, sondern auch weitere externe Systeme kontaktieren zu können.
Nachrichten empfangen	Ein Workflow kann an einer bestimmten Stelle auf eine bestimmte Nachricht eines externen Systems warten, um fortzufahren. Dies wird im ersten Schritt über einen Skript-Aufruf, längerfristig über eine E-Mail-Schnittstelle realisiert.
E-Mail-Benachrichtigungen versenden	Um die Benutzer und beliebige Systeme über bestimmte Systemzustände zu informieren, können an definierten Stellen eines Workflows Nachrichten mit Informationen zum aktuellen Zustand eines Workflows versendet werden.

Tabelle 2.2: Anwendungsfälle

3. Grundlagen Workflow-Management-Systeme

Im folgenden Kapitel soll erklärt werden, wozu Workflow-Management-Systeme (WfMS) im allgemeinen dienen, welche Schritte zur Erstellung eines Workflow-Modells notwendig sind und wie aus diesem Modell eine durch das WfMS zu verarbeitende Workflow-Definition erstellt wird. Die Erstellung eines solchen Modells erfolgt über Workflow-Patterns (siehe 3.2), die die interne Logik des Workflows repräsentieren. Im zweiten Teil dieses Kapitels werden die internen Mechanismen des Systems SWAMP geschildert, und anschließend wird überprüft, welche der Workflow-Patterns damit abgebildet werden können und inwiefern hiermit die Voraussetzungen für die Zielstellung (siehe **??**) dieser Arbeit gegeben sind.

Ein Workflow wird angestoßen, um einen bestimmten betriebsinternen Geschäftsvorfall, z. B. die Erzeugung eines Produktes oder einer Dienstleistung, abzuwickeln. Er wird durch ein bestimmtes Ereignis ausgelöst und ist durch einen definierten Startpunkt und einen oder mehrere definierte Endpunkte abgrenzbar. Dazwischen besteht er aus einer Reihe definierter Zustände, die jeweils mit bestimmten Aufgaben für mehrere Personen oder bestimmten Ereignissen verknüpft sind. Es können Bedingungen für die zeitliche Abfolge von Aufgaben bestehen (parallele/sequentielle Ausführung), und bestimmte Aufgaben können der Abstimmung bedürfen. Intern muß es den verschiedenen Stationen möglich sein, Informationen und Daten auszutauschen. Die Koordination dieser Aufgaben und die Darstellung des Fortschritts sowie der noch ausstehenden Aktivitäten sind die Hauptaufgaben des Workflow-Management-Systems.

Damit ein Geschäftsprozess in einem Workflow-Management-System möglichst gut abgebildet werden kann, muß er bestimmten Kriterien genügen oder auf das verwendete System hin angepaßt werden. Um einzuschätzen, ob ein Ablauf in ein Workflow-Management-System eingebunden werden kann und ob er auch sinnvoll in diesem System abbildbar ist, muß er nach [Koch96] auf folgende Kriterien hin analysiert werden:

- Formalisierbarkeit
- Auftretensart
- Auftretenshäufigkeit

Je nach Eigenschaft dieser Punkte ergeben sich andere Anforderungen an das Workflow-Management-System. Vollständig formalisierbare und gut strukturierte Vorgänge lassen sich viel besser durch Regeln beschreiben und automatisieren als nur kaum formalisierbare, unstrukturierte Vorgänge, die häufig eine starke Kommunikation verschiedener Beteiligter erfordern. Falls eine in dieser Richtung über die Leistungsfähigkeit des Workflow-Management-Systems hinausgehende Kooperation der mit Aufgaben belegten Personen erforderlich ist, kann dies durch den zusätzlichen Einsatz oder die Integration von Groupware-Systemen geschehen. Diese Systeme bieten Möglichkeiten der E-Mail-Kommunikation, Dokumentenmanagement, Gruppen-Terminkalender und weitere Unterstützung für die Zusammenarbeit von Teams. Während das Ziel der Groupware die Unterstützung von *Kooperation* zum Ziel hat, ist das Ziel eines Workflow-Management-Systems die *Koordination*.

Bei einem abgebildeten Workflow kann es sich lediglich um einen trivialen Ablauf handeln, es können aber auch komplexe und organisationsübergreifende Prozesse verarbeitet werden. Ein gutes Workflow-Management-Tool ermöglicht es, solche organisationsübergreifenden Workflows zu erstellen, indem es durch Schnittstellen Interoperabilität zu anderen Systemen bietet.

Der potentielle Nutzen bei der Verwendung eines solchen Systems ist eine Verbesserung der Effizienz der Geschäftsprozesse. Ein einfacheres Ändern und Anpassen der Strukturen auf aktuelle Veränderungen wird ermöglicht. Mit Hilfe der von allen Nutzern abrufbaren Informationen des Systems ist jeder in der Lage den aktuellen Status eines Ablaufs festzustellen. Der Informationsgrad der Benutzer wird somit verbessert. Wenn der Workflow strukturiert innerhalb des Systems vorliegt, ist es auch einfacher die Effizienz dieses Modells mit Hilfe von wissenschaftlichen Methoden zur Prozessoptimierung zu verbessern. Dadurch ergeben sich weitere Vorteile, z. B. die Zeiteinsparung bei der Bearbeitung und damit eine schnellere Reaktion bei kundenorientierten Workflows. Desweiteren erhöht sich die Qualität des Ergebnisses, da das System bestimmte Fehlerquellen, wie falsche Eingaben, selbständig erkennen kann.

Die Entwicklung von Workflow-Management-Systemen geht momentan hin zur Unterstützung von flexiblen Vorgängen mit der Möglichkeit, in bestimmten Situationen vom definierten Ablaufplan abzuweichen. Damit wird auch die Integration weniger strukturierter Workflows (sog. *Ad-hoc* Workflows) erlaubt.

Das Workflow-Management der meisten WfMS integriert die folgenden drei funktionalen Bestandteile:

- Unterstützung des Benutzers bei der Definition und Modellierung eines Workflow-Prozesses und seiner einzelnen durchzuführenden Aktionen.
- Die Kontrolle und Steuerung des Workflow-Flusses zu seiner Laufzeit, sowie die Steuerung und Delegierung seiner internen Aktionen.
- Die Interaktion mit den Workflownutzern und anderen den Workflow beeinflussenden Systemen über verschiedene Schnittstellen.

In der ersten Phase der Definition eines neuen Workflows wird ein Meta-Modell des Workflows erstellt, in dem die verschiedenen internen Aktivitäten und Funktionen Benutzer-Rollen zugeordnet werden. Beim Einspielen dieses Meta-Modells in ein WfMS generiert es hieraus ein konkretes Workflow-Modell, indem es den Rollen aufgrund interner Logik konkrete Personen zuordnet, die ihre Aufgaben dann über die Schnittstellen des Systems erledigen können. Bei den Rollen in der Definition muß es sich nicht zwingend um Personen handeln, WfMS können hier auch externe Systeme und Funktionen ansprechen. Ebenso kann der Auslöser einer Aktion ein systeminternes oder systemexternes Ereignis sein, auf das das WfMS mit einer Maßnahme reagiert.

3.1 Erstellen von Workflow-Modellen

Um einen bestehenden Workflow in ein WfMS zu integrieren, muß er sich in bestimmter Art und Weise formalisieren lassen, um ein ihn repräsentierendes Ablaufmodell zu erstellen. Wenn ein Ablaufplan inklusive der einzelnen Stationen, der Verknüpfung bestimmter Benutzergruppen mit ihren Aufgaben, und ein Regelwerk für den Ablauf des Workflows erstellt wurde, muß man dieses Modell in eine für das WfMS *verständliche* Form bringen.

Es ist hilfreich, bei der Bearbeitung und Definition von Workflows auf Standards zurückzugreifen, da das Workflow-Management kein in sich geschlossener Prozeß innerhalb einer Firma ist, sondern manche Workflows innerhalb ihres Lebenszyklus auch in Kontakt mit anderen Systemen oder Organisationen treten müssen. Dies wird durch die zugängliche Definition der Schnittstellen möglich. Weiterhin ist es sinnvoll, die verschiedenen Workflow-Patterns (siehe 3.2), d. h. die

Grundlagen eines Workflow-Modells, zu standardisieren, um eine Kompatibilität oder zumindest eine einfache Übertragbarkeit der Definitionen zwischen verschiedenen Modellen zu ermöglichen.

Die einzelnen Workflow-Definitionen liegen dem System in einer speziellen „*Process Description Language*“ vor, einer Sprache oder Grammatik, die vom Workflow-Management-System verarbeitet werden kann. Bei den meisten existierenden Workflow-Management-Systemen handelt es sich um XML- Formate (siehe 4.3.3), da dieses Format sehr unkompliziert in der Handhabung ist. Beispiele für auf XML aufbauende „*Process Description Languages*“ sind XPDL[1] (XML Processing Description Language), YAWL[2] (Yet Another Workflow Language), BPML[3] (Business Process Modeling Language), WS-BPEL (Business Process Execution Language for Web-Services) oder die von der WfMC[4] entwickelte Wf-XML[5].

Für einige dieser Beschreibungssprachen existieren grafische Editoren, so daß das Erstellen einer Workflow-Definition auch Benutzern ermöglicht wird, die nicht mit den Internas der Beschreibungssprache vertraut sind. Zusätzlich lassen sich mit solchen Tools Engpässe und mögliche Deadlocks in Workflow-Definitionen bereits zum Design-Zeitpunkt identifizieren und alternative Abläufe ausprobieren.

3.2 Workflow-Patterns

Workflow-Patterns (dt.: Workflow-Muster, -Schemata) können verwendet werden, um die Ausdrucksstärke eines Workflow-Management-Systems zu beschreiben, und sind die Grundlage, auf der eine Workflow-Definition aus einer gegebenen Anforderungsspezifikation erstellt werden kann. Standardisierte Design-Patterns bieten eine implementationsunabhängige Basis, sie sind *„die Abstraktion eines bestimmten Musters, das sich in speziellen, nicht beliebigen Zusammenhängen wiederholt.“*[6] Ein *Übersetzen* eines Geschäftsprozesses in seine verwendeten Workflow-Patterns ist also der erste Schritt seiner Integration in ein Workflow-Management-System. In dieser Phase können bereits Mängel an der Flexibilität des verwendeten Systems ausgemacht werden, die es notwendigkeit machen den Prozess umzugestalten. Im folgenden wird eine Übersicht über die wichtigsten von Van der Aalst, Hofstede und Kiepuszwewski entdeckten und ausformulierten[7] Workflow-Patterns erstellt. Anschließend wird versucht, den Funktionsumfang der Anwendung SWAMP anhand der in ihr umsetzbaren Workflow-Patterns einzuordnen.

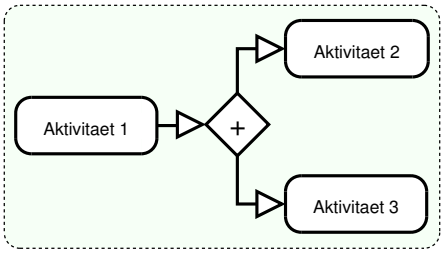
Diese Methode ist analog zur Vorgehensweise in Bartosz Kiepuszewskis Dissertation[8]. In ihr werden acht verschiedene kommerzielle Software-Tools auf ihre Mächtigkeit und Nützlichkeit hin überprüft, indem sie an der Modellierbarkeit von Workflow-Patterns gemessen werden. Dadurch läßt sich feststellen, welche Komplexität eines Geschäftsvorfalles mit dem System abgebildet werden kann, und wo eventuell Schwachpunkte liegen, die das System für spezielle Anforderungen disqualifizieren würden.

[1] http://www.simonstl.com/projects/xpdl/
[2] http://www.citi.qut.edu.au/yawl/index.jsp
[3] http://www.bpmi.org/specifications.esp
[4] Workflow Management Coalition, http://www.wfmc.org
[5] http://www.wfmc.org/standards/wfxml_demo.htm
[6] nach [Zül96]
[7] [vdAA00] und http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
[8] [Kiep02]

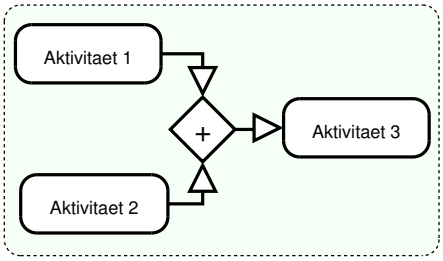
Einfache Patterns

- **Sequenz:**
  Die Aktionen eines Workflow-Abschnitts werden nacheinander durchgeführt, d. h. ein Zustand wird erst aktiv, wenn der vorhergehende beendet wurde. Die Sequenz stellt einen linearen Ablauf in einem Workflow dar.

- **Paralleler Split:** Der parallele Split teilt einen Pfad (im folgenden wird ein aktiver Pfad eines Workflow-Ablaufs auch Thread genannt) des Workflowablaufs in mehrere auf, die dann gleichzeitig nebeneinander ablaufen. Auf diese entstandenen, nebenläufigen Threads können z. B. Aktivitäten verteilt werden, die, unabhängig voneinander, durch verschiedene Personen gleichzeitig erledigt werden.



- **Synchronisieren:** Eine Synchronisierung führt mehrere Threads, die vorher z. B. durch einen parallelen Split erzeugt wurden, wieder zu einem einzigen zusammen. Damit der Workflow fortfahren kann, müssen alle ankommenden Threads ihre Aktivitäten beendet haben. Der Ablauf des Workflows wird somit in diesem Punkt sychronisiert. Dieser Übergang wird von der WfMC als *AND-Join* bezeichnet.



- **Exklusive Wahl:** Die exklusive Wahl definiert mehrere mögliche Pfade für den Workflow, es ist aber im Gegensatz zum parallelen Split nur ein Weg möglich, auf dem der Prozess fortfahren kann. Welcher Weg genommen wird, ergibt sich aus dem Resultat einer Aktivität (z. B. durch manuelle Entscheidung) oder wird auf andere Art und Weise automatisch vom Workflow-Modell bestimmt. Dieser Übergang wird von der WfMC als *XOR-Split* bezeichnet.

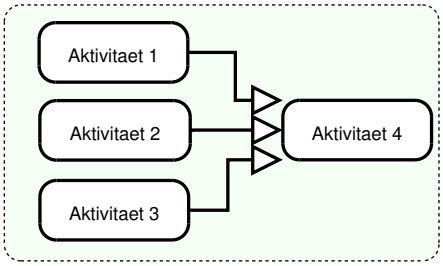
- **Einfache Vereinigung:** Die einfache Vereinigung führt mehrere Pfade zu einem einzigen zusammen. Die ankommenden Threads werden hier allerdings nicht synchronisiert, sondern es wird davon ausgegangen, daß nur auf einem der ankommenden Pfade ein Thread eintrifft (XOR-Join). Nachfolgend eintreffende Threads werden blockiert. Diese Situation entsteht z. B. nach der Trennung durch eine *exklusive Wahl*.

Erweiterte Patterns

- **Multiple Choice:** Das Multiple-Choice-Pattern erlaubt im Gegensatz zum *parallelen Split* und der *exklusiven Wahl* eine beliebige Anzahl zu aktivierender Pfade. Welche Pfade aktiviert werden, kann zur Laufzeit vom System aufgrund interner Logik oder durch eine von einem Benutzer manuell zu treffende Auswahl entschieden werden.

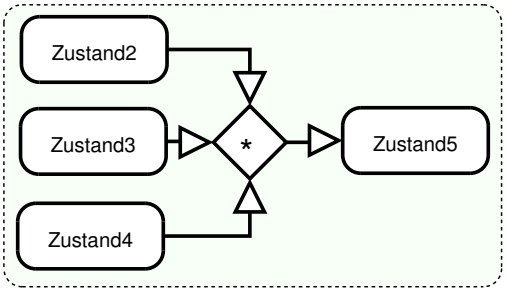
- **Synchronisierende Vereinigung:** Die synchronisierende Vereinigung ist eine Sonderform der bereits vorgestellten *Synchronisierung*. Sie wird nach der Aufsplittung des Workflow-Ablaufs durch das *Multiple-Choice-Pattern* eingesetzt, um die entstandenen Threads zu einem späteren Zeitpunkt wieder zu synchronisieren. Im Unterschied zur normalen *Synchronisierung* wird hier aber nicht auf die Beendigung aller ankommenden Pfade gewartet, sondern nur auf die, die bei dem vorangegangenen *Multiple-Choice* aktiviert wurden.

- **Mehrfache Vereinigung:** Bei der mehrfachen Vereinigung (Multi-Merge) wird eine beliebige Anzahl von Pfaden vereinigt. Dies geschieht hier jedoch ohne Synchronisierung und ohne Vorwissen über die Anzahl der zu erwartenden Threads. Der weiterführende Pfad kann beliebig oft aktiviert werden, es werden keine ankommenden Threads blockiert. Es besteht also im Gegensatz zur *einfachen Vereinigung* und der *Synchronisierung* die Möglichkeit einer Mehrfachausführung des weiterführenden Pfades und der in ihm enthaltenen Aktivitäten.



- **Diskriminator:** Das Diskriminator-Pattern ist fest mit einem vorher stattfindenden *Multiple-Choice-Pattern* verbunden. Es verhält sich im ersten Durchgang wie die *einfache Vereinigung*, wartet also auf den zuerst eintreffenden Thread und blockt nachfolgende Threads ab. Der Unterschied besteht darin, daß der Diskriminator merkt, wenn alle durch den *Multiple Choice* erzeugten Threads entweder durchgeleitet oder abgeblockt wurden, und daraufhin seinen Status zurücksetzt. Es ist also möglich, den zusammengefaßten *Multiple Choice-Diskriminator*-Block mehrfach während der Laufzeit des Workflows zu durchlaufen.

- **N aus M Join:** Der *N aus M Join* ist eine Mischung aus *Synchronisierung* und *Diskriminator*. Anstatt nur einen oder alle der parallel ankommenden Threads als Bedingung für das Weiterleiten zu haben, ist es hier möglich, zu definieren, auf wievielen (N) der M ankommenden Pfade aktive Threads ankommen müssen, um den weiterführenden Pfad zu aktivieren. Nachfolgend ankommende Threads werden, wie beim Diskriminator, gestoppt.



Strukturelle Patterns

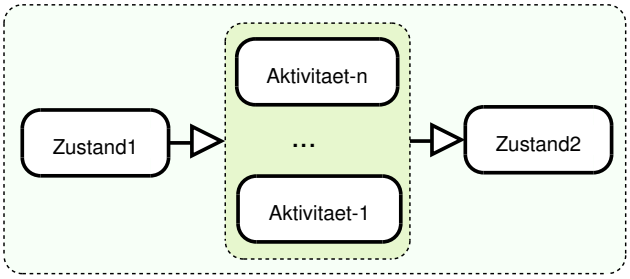
- **Arbiträre Kreise:** Ein arbiträrer Kreis erlaubt es einem Workflow-Prozeß, bestimmte Sektionen der Workflow-Definition mehrfach zu durchlaufen.

- **Implizite Terminierung:** Die Möglichkeit der Beendigung eines Pfades ohne das Beenden des gesamten Workflows nennt sich implizite Terminierung. Der Pfad wird beendet, wenn er keine auszuführenden Aktivitäten mehr enthält und in einer *Sackgasse* endet.

Patterns mit mehreren Instanzen einer Aktivität

- **Mehrfache Instanzen mit Festlegung zum Design-Zeitpunkt:** Dieses Pattern beschreibt, wie eine Aktivität eine bestimmte Anzahl an parallelen Instanziierungen erfahren kann. Die Anzahl der Instanziierungen ist zum Design-Zeitpunkt des Workflows bereits bekannt und wird in der Workflow-Definition hinterlegt. So kann z. B. festgelegt werden, daß aus einer Aktivität mehrere Kopien, die durch jeweils andere Benutzer ausgeführt werden können, erzeugt werden. Der Workflow-Thread fährt fort, sobald alle erzeugten Aktivitäten beendet wurden.

- **Mehrfache Instanzen mit Festlegung zur Laufzeit:** Dieses Pattern erzeugt mehrere Instanzen einer Aktivität, ihre Anzahl wird aber im Gegensatz zum vorherigen Pattern erst zur Laufzeit des Workflows durch das System bestimmt. Sie ist zum Design-Zeitpunkt noch unbekannt.



- **Mehrfache Instanzen ohne Festlegung zur Laufzeit:** Im Unterschied zum vorhergehenden Pattern *mit Festlegung zur Laufzeit* können hier noch neue Instanzen der Aktivität hinzugefügt werden, wenn bereits welche erzeugt wurden. Wieviele Instanzen erzeugt werden, wird durch die Logik des Workflows bestimmt und kann sich durch andere, parallel ablaufende Aktivitäten ändern. Es müssen alle erzeugten Aktivitäten beendet werden, bevor der Workflow fortgesetzt wird.
- **Mehrfache Instanzen mit/ohne Synchronisierung:** Die *Mehrfache-Instanzen*-Patterns können jeweils mit einer oder ohne eine Synchronisierung des Ausgangs ihrer Aktivitäten eingesetzt werden. Ohne eine Synchronisierung wäre es möglich, mehrfache Instanzen einer Aktivität zu erzeugen. und ohne, oder bei der ersten beendeten Instanz im Workflow fortzufahren. Mit Synchronisierung setzt der Workflow-Thread erst wieder ein, wenn alle erzeugten Instanzen beendet wurden.

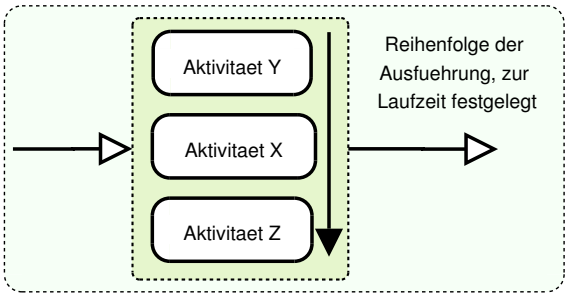
Temporäre Verbindungen

- **Überlappende Sequenz:** Es ist möglich, zwei Aktivitäten in einen zeitlichen Zusammenhang zu stellen, z. B. die Bedingung zu stellen, daß Aktivität 2 nach dem Start von Aktivität 1, aber vor der Beendigung von Aktivität 1 gestartet werden muß.

Zustandsbasierte Patterns

- **Verzögerte Auswahl:** Die *Verzögerte Auswahl* ist eine Variante des *Exklusive-Wahl*-Patterns, mit dem Unterschied, daß die Entscheidung über den Ziel-Pfad bei seinem Erreichen noch nicht getroffen ist. Das Pattern wartet auf das Eintreffen eines bestimmten Ereignisses innerhalb des Workflows, woraufhin der entsprechende Pfad aktiviert wird. Die anderen Pfade werden gesperrt.

- **Gruppiertes, paralleles Routing:** Hier werden mehrere Aktivitäten, die meistens auf dieselben Ressourcen zugreifen, gruppiert. Der Name parallel ist verwirrend, denn tatsächlich müssen die Aktivitäten sequentiell, aber in einer zur Laufzeit bestimmten Reihenfolge bearbeitet werden.



- **Meilenstein:** Die Aktivierung einer Aktivität kann von dem Erreichen eines sog. Meilensteins abhängen oder diesen erzeugen. Als Meilenstein wird das Erreichen eines wichtigen Punkts im Fortschritt des Workflows bezeichnet.

Cancellation Patterns

- **Abbruch einer Aktivität:** Es kann sinnvoll sein, einzelne Aktivitäten abzubrechen. Beispielsweise wenn ein paralleler Thread durchlaufen wird, der die Ausführung einer anderen Aktivität überflüssig macht. Der auf das Beenden der Aktivität wartende Thread stirbt mit

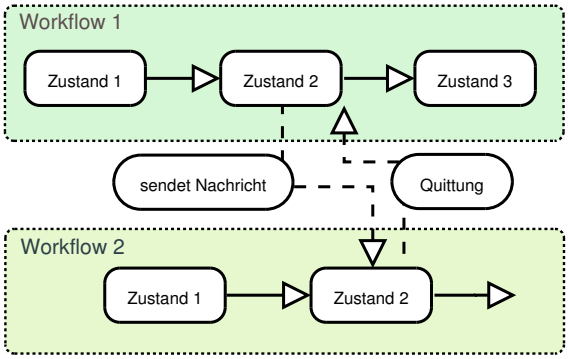
dem Abbruch der Aktivität. Es wäre auch möglich, dieses Pattern in einer Administrator-Funktionalität zur Verfügung zu stellen, um selektiv einzelne Aktivitäten und Threads des Workflows zu beenden.

- **Abbruch eines Workflows:** Es ist möglich, die komplette Instanz des laufenden Workflows und alle von ihr abhängigen Threads und Tasks zu beenden.

Synchronisierung zwischen Workflows

- **Kommunikation über Nachrichten:** Mehrere unabhängig voneinander laufende Workflow-Instanzen in einem Workflow-Management-System können über Nachrichten kommunizieren. Der Empfang der Nachricht kann bei einem wartenden Workflow an einer bestimmten Stelle als Bedingung für seinen Fortgang eingesetzt werden.

- **Koordinierte Kommunikation:** Durch eine koordinierte Kommunikation kann sichergestellt werden, daß der Empfänger eine an ihn gesendete Nachricht auch wirklich erhalten hat. Dies wird dadurch erreicht, indem der Empfänger dem Sender den Empfang der Nachricht durch eine *Quittung* bestätigt.



- **Multi-Cast-Nachrichten senden/empfangen:** Multi-Cast-Nachrichten haben keinen bestimmten Empfänger und können von einer beliebigen Anzahl aufgenommen werden. Dies ermöglicht das Modellieren von Geschäftsvorfällen, bei denen mehrere externe Empfänger auf das Eintreffen der gleichen Art von Nachricht warten.

3.3 Warum ein neues Workflow-Management-System?

Da die Verfügbarkeit und Möglichkeit der Erweiterung des Workflow-Management-Systems unter einer freien Lizenz (siehe auch 4.4) eine feste Voraussetzung zum Einsatz einer Software innerhalb der Firma SUSE ist, ist die Nutzung einer kommerziellen Workflow-Management-Lösung ausgeschlossen. Die Suche nach freien Implementierungen gestaltete sich allerdings schwierig, da keine Lösung gefunden wurde, die die bestehenden Anforderungen an Funktionsumfang und Erweiterbarkeit zufriedenstellend erfüllt. Die Erweiterbarkeit des Systems hat eine hohe Priorität, da die automatische Kommunikation mit SUSE-internen Tools zentraler Bestandteil des Systems sein soll und den Benutzern Vorteile im Gegensatz zur bisherigen Arbeitsweise mit einer Vielfalt von einzelnen Tools eröffnen soll. Da die Integration solcher neuer Informationseinheiten in bestehende Workflow-Modelle problematisch ist, wurde ein neues Projekt gestartet, mit dem Ziel, ein leicht erweiterbares Workflow-Management-System zu entwickeln, das den Einsatzzweck und die Anforderungen der Entwicklungsabteilung erfüllt - **SWAMP**. Da die für spezielle firmeninterne Anwendungsfälle implementierten Erweiterungen modulartig aus dem Programmkern ausgelagert wurden, entsteht eine offene Plattform, die auch von anderen Anwendern für ihre Anforderungen angepaßt und erweitert werden kann.

Beispiele für freie Implementierungen von Workflow-Management Systemen:
- Taverna (http://taverna.sourceforge.net)
- WfMOpen (http://wfmopen.sourceforge.net)
- Java Business Process Management (http://www.jbpm.org)
- Twister (http://www.smartcomps.org/twister)
- Jakarta Workflow (http://jakarta.apache.org/commons/sandbox/workflow)
- Openflow (http://www.openflow.it)

3.4 Das Workflow Modell von SWAMP

Im folgenden Abschnitt werde ich die Fähigkeiten des Systems SWAMP mit den eben beschriebenen Workflow-Patterns vergleichen, um zu einer Einschätzung zu kommen, ob damit genügend komplexe Workflows abgebildet werden können. Es soll herausgefunden werden, wo eventuelle Schwachpunkte im jetzigen Funktionsumfang sind. Diese Schwachpunkte sollen dann, wo nötig und möglich, zusammen mit der Integration des Maintenance-Workflows beseitigt werden. Das Design des entwickelten Workflow-Modells wurde unter der Vorgabe entworfen, daß ein typischer Prozess, der mit dem System abgebildet werden soll, eine relativ langsame Abarbeitung in vielen Einzelschritten darstellt und nicht sehr oft gestartet wird. Das System soll durch Abstraktion der abzubildenden Workflows eine generische Lösung anbieten, mit der beliebige Arbeitsabläufe dargestellt werden können und deren Erweiterung durch definierte Schnittstellen einfach möglich ist.

SWAMP integriert zum momentanen Zeitpunkt keine Tools zur Unterstützung des Benutzers bei der Modellierung eines Workflows und der Erstellung einer Workflow Definition. Die Erstellung einer grafischen Oberfläche, die hier Hilfestellung bietet, hat keine hohe Priorität im derzeitigen Projektstand, da die Workflows von den Entwicklern unter Mithilfe der betroffenen Mitarbeiter selbst erstellt werden. Wenn zu einem zukünftigen Zeitpunkt das Erstellen von Workflows auch nicht SWAMP-Entwicklern ermöglicht werden soll, erlangt die Verfügbarkeit einer grafischen Oberfläche hierfür höhere Priorität.

Das SWAMP-Workflow-Modell besteht aus einem Netz von Knoten, die durch Kanten verbunden werden können. Ein Knoten dient als Container für eine oder mehrere Aktionen, die bei ihrer Ausführung bestimmte Ereignisse aussenden. Auf welchem Pfad der Workflow nun durchlaufen wird, hängt von den ausgesendeten Ereignissen ab, die bestimmen welche Kante beschritten wird, und somit den nächsten Knoten aktivieren.

Durch die Mächtigkeit dieses Modells kann nicht ausgeschlossen werden, daß eine Workflow-Instanz durch schlechtes Design oder große Komplexität der Workflow-Definition in einer Deadlock-Situation hängen bleibt. Mechanismen, um solche Situationen zu lösen oder durch eine Analyse der Workflow-Definitionen nicht zuzulassen, könnten zu einem späteren Zeitpunkt implementiert werden.

Es folgen Erklärungen zu den Elementen des SWAMP-Workflow-Modells:

Knoten

Ein Knoten (Node) ist ein bestimmter Punkt im Ablauf eines Workflows, der auszuführende Aktionen, z. B. die Eingabe von Daten durch den Benutzer, beinhalten kann. Es gibt einen Startknoten, an dem der Prozess bei der Initialisierung des Workflows einsetzt, und einen oder mehrere Endknoten. Wird ein Endknoten erreicht, wird der Workflow als beendet angesehen. Noch aktive Aufgaben oder andere aktive Knoten werden verlassen und beendet. Leere Knoten und Knoten ohne weiterführende Kanten sind möglich, um bestimmte Workflow-Patterns abbilden zu können.

Aktion

Eine Aktion (Action), in der Literatur auch Aktivität genannt, befindet sich innerhalb eines Knotens. Sie definiert eine Aufgabe, die beim Betreten des Knotens ausgeführt werden soll. Ein Knoten kann eine beliebige Anzahl an Aktionen beinhalten. Mögliche Aktionen sind z. B. Dataedit (Dateneingabe), Decision (Entscheidung), Manualtask (Ausführen einer Aufgabe außerhalb des Systems und Bestätigung in der Oberfläche), Notification (verschiedene automatische Benachrichtigungen) oder der Aufruf von benutzerdefinierten Skripten und Java-Funktionen.

Aufgabe

Eine Aufgabe (Task) ist die konkrete Instanziierung einer Aktion eines aktiven Knotens. Möglich sind Aufgaben, die die Bearbeitung in der Oberfläche durch eine Person oder Personengruppe bedingen (Usertask), oder Aufgaben, die vom System selbständig ausgeführt werden (Systemtask).

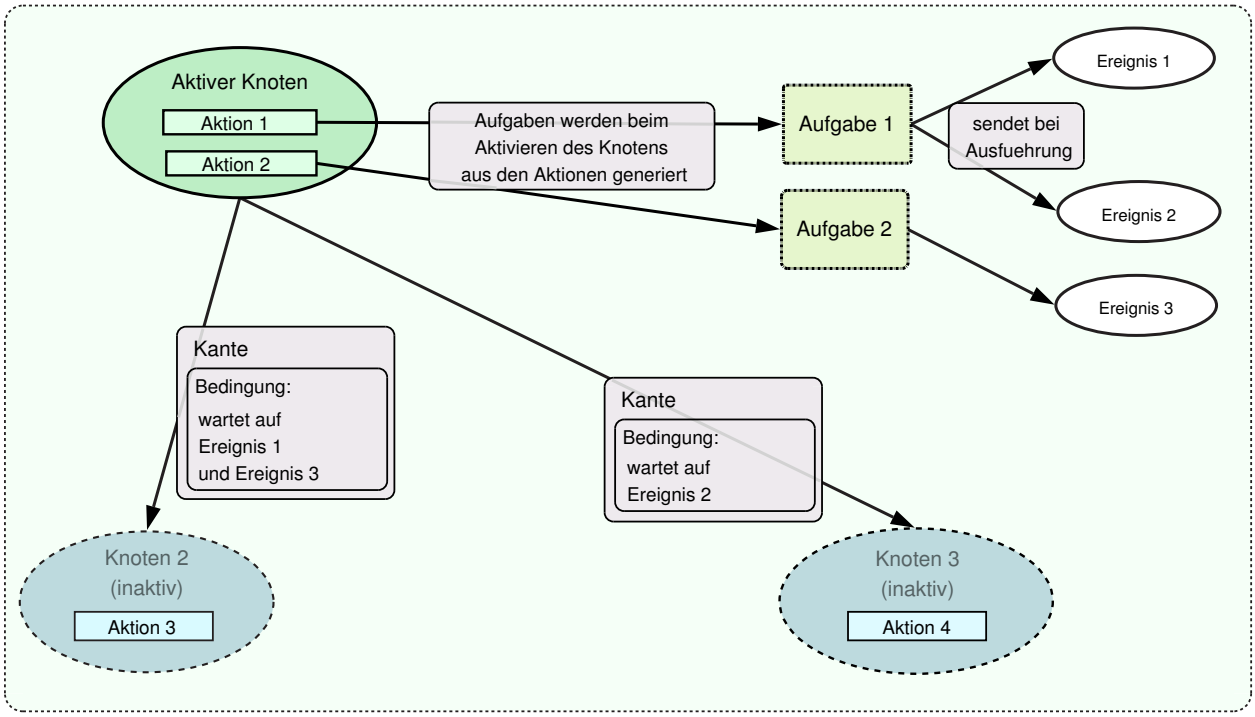


Abbildung 3.1: Elemente des SWAMP-Workflow-Modells

Kante

Eine Kante (Edge) bildet die Verbindung zwischen zwei Knoten, und beinhaltet eine Bedingung. Wenn die Bedingung erfüllt wurde, wird der Zielknoten der Kante aktiviert und der Startknoten deaktiviert. Eine Kante kann eine beliebige Anzahl unterschiedlich komplexer Bedingungen beinhalten, um spezielle Workflow-Patterns zu modellieren. Zusätzlich besteht die Möglichkeit, mehrere Kanten ohne Bedingungen einzusetzen, wodurch eine Aufteilung des Prozess-Flusses (Threads) ermöglicht wird.

Bedingung

Die Erfüllung einer Bedingung ist die Vorraussetzung für die Aktvierung des Zielknotens einer Kante. Die Bedingung wartet solange, bis sie erfüllt wurde oder der Startknoten ihrer Kante anderweitig verlassen wurde. Da eine Bedingung ein boolesches Ergebnis[9] liefert, wird durch die Verwendung von AND-, OR- und NOT-Verknüpfungen das Erstellen komplexer Bedingungen möglich, was die Flexibilität und Mächtigkeit des Workflow-Modells enorm erweitert. Wenn ein Teil einer zusammengesetzten Bedingung z. B. durch das Eintreffen eines Ereignisses erfüllt wird, dann wird, falls die Gesamtbedingung dadurch noch nicht erfüllt wurde, dieser Zustand gespeichert und beim nächsten Test der Bedingung wieder miteinbezogen. Die Ereignisse, auf die zusammengesetzte Bedingungen warten, müssen also nicht gleichzeitig eintreffen. Sobald die Gesamtbedingung erfüllt wurde, wird der Startknoten ihrer Kante deaktiviert und der Zielknoten aktiviert. aktiviert werden. Mögliche Bedingungen sind:

- Warten auf das Eintreffen eines Ereignisses
- Überwachen der Daten eines Workflows nach bestimmten Kriterien

[9] nach George Boole ein Element, das nur 2 Werte annehmen kann, hier Wahr/Falsch

3.5 Unterstützte Workflow-Patterns

Es soll untersucht werden, welche der vorgestellten Workflow-Patterns zum derzeitigen Projektstand vom System unterstützt werden und wie die Implementierung der jeweiligen Muster realisiert werden kann. Da der größte Teil aller Firmenprozesse auf diese Muster abgebildet werden kann, kann mit einer hinreichenden Unterstützung der Workflow-Patterns davon ausgegangen werden, daß das System flexibel genug ist, um die aktuellen und zukünftigen Erwartungen zu erfüllen. Auch wenn noch kein konkreter Anwendungsfall eines bestimmten Patterns aufgetreten ist und somit keine Implementierung einer Unterstützung für dieses Pattern vorliegt, ist es möglich, anhand der Struktur und der dahinter steckenden Konzepte von SWAMP abzuschätzen, ob die zukünftige Realisierung eines solchen Patterns möglich ist, ohne die Grundmechanismen der Anwendung zu ändern.

einfache Patterns (siehe Abschnitt 3.2, Seite 19): Eine *Sequenz* wird durch die lineare Verkettung von Knoten durch ihre Kanten hergestellt. Um eine *exklusive Wahl* darzustellen, werden die zur Wahl stehenden Kanten mit Bedingungen belegt, die jeweils auf ein anderes Ereignis warten. Um den *parallelen Split* und die *Synchronisierung* abzubilden, müssen leere Zwischenknoten modelliert werden, die das Aufteilen und das synchronisierte Zusammenführen übernehmen. Durch einen leeren Knoten (einen Knoten ohne Aktionen) und mehrere von ihm weiterführende Kanten kann der Prozeßfluß aufgeteilt werden(*paralleler Split*). Danach kann er durch einen leeren Knoten mit mehreren zu ihm führenden Kanten wieder vereinigt werden. Um diese Vereinigung zu synchronisieren, muß der Knoten eine weiterführende Kante haben, die erst bei der Erfüllung aller ankommenden Pfade weiterleitet. Die *einfache Vereinigung* kann direkt ohne Zwischenknoten abgebildet werden, da der Zielknoten im SWAMP-Modell wie verlangt auch mehrfach aktiviert werden kann, wenn ihn mehrere Threads erreichen.

erweiterte Patterns: Die Patterns *Multiple Choice* und *Synchronisierende Vereinigung* können, analog zu den einfachen Patterns *Exklusive Wahl* und *Synchronisierung*, mit leeren Zwischen-Knoten modelliert werden. Sie benötigen jedoch noch zusätzliche Logik, die durch Speicherung der Entscheidung des *Multiple Choice* im Datenbereich des Workflows oder durch geschicktes Auswählen der zu sendenden Ereignisse möglich ist. Die *Mehrfache Vereinigung* ist wie die *einfache Vereinigung* ohne zusätzliche Hilfsmittel modellierbar. Ein *Diskriminator* läßt sich durch die Blockierung der nachfolgenden Threads über Bedingungen realisieren, indem die ankommenden Kanten eines Knotens mit Bedingungen belegt werden, die nach dem ersten Ausführen der Aktion des Knotens immer negativ sind. Der *N aus M join* läßt sich nicht direkt realisieren. Es wäre aber möglich, eine ähnliche Funktionalität durch in den einzelnen Pfaden einzugebende Daten und eine Bedingung, die diese Daten überprüft, nachzubilden.

strukturelle Patterns (siehe Abschnitt: *Arbiträre Kreise* und *implizite Terminierung* lassen sich über Kanten, die zu einem früheren Punkt, einem bereits durchlaufenen Knoten, des Workflows zurückverzweigen und (End-)Knoten, aus denen keine weiteren Kanten hervorgehen, direkt realisieren.

Patterns, die mehrere Instanzen beeinflussen: Das dynamische Erzeugen mehrerer Aufgaben aus einer Aktion ist in SWAMP momentan nicht möglich. Da die Notwendigkeit, Aufgaben dynamisch zu erzeugen eher akademischer Natur erscheint, wurde auf eine Implementierung dieser Möglichkeit bislang verzichtet. Sie kann aber durch eine Erweiterung des Workflow-Modells um eine entsprechenden Aktion, die eine in der Workflowdefinition, oder während der Laufzeit bestimmte Anzahl von Aufgaben dynamisch generiert, implementiert werden.

temporäre Verbindungen: Um den Start- und Endzeitpunkt verschiedener Aufgaben in einen zeitlichen Zusammenhang zu bringen, ist es notwendig, die internen Zustände der Aufgaben für das System abfragbar zu machen. So können entsprechende Bedingungen z. B. auf den Startzeitpunkt einer Aufgabe zugegreifen, und somit die Funktionalität des Patterns im System abbilden.

zustandsbasierte Patterns: Die *verzögerte Auswahl* kann durch einen leeren Knoten, dessen fortführende Kanten jeweils auf ein anderes Ereignis warten, hergestellt werden. Ein *gruppiertes, paralleles Routing* ist in SWAMP bislang nicht vorgesehen. Die Reihenfolge muß in der Workflow-Definition festgelegt sein. Das Setzen eines *Meilensteins* kann in Form eines Dateneintrags im Workflow erfolgen.

cancellation Patterns: Der Abbruch einer einzelnen Aufgabe oder des kompletten Workflows ist möglich. Die Workflow- und Aufgaben-Objekte können jeweils einen Zustand aus einer vorgegebenen Liste möglicher Zustände einnehmen, darunter sind z. B. die Zustände abgebrochen, gelöscht und aktiv.

Synchronisierung zwischen Workflows: Der Austausch von Daten oder das Senden von Ereignissen an andere Workflow-Instanzen ist momentan noch nicht möglich, wird aber in einer zukünftigen Version von SWAMP integriert werden, da dies den Funktionsumfang um viele Möglichkeiten, z. B. das Starten von Sub-Workflows, erweitert.

Die vorangegangene Analyse der von SWAMP unterstützten Workflow-Patterns zeigt, daß der überwiegende Teil der unter 3.2 vorgestellten Patterns abgebildet werden kann und die zur Zeit noch nicht integrierten Patterns keine unlösbare Aufgabe darstellen, sondern ein Teil der zukünftigen Weiterentwicklung von SWAMP sind. Das Ziel von SWAMP ist auch keine lückenlose Unterstützung sämtlicher denkbarer Workflow-Patterns, sondern ein möglichst vollständiges Bereitstellen der von den im System laufenden Workflows benötigten Funktionen. Die Notwendigkeit der Integration an neuen Workflow-Patterns dürfte bei den Mustern *Workflow-Synchronisierung* und *temporäre Verbindungen* am naheliegendsten sein, da bereits absehbar ist, daß zukünftige Workflows der Firma SUSE in diese Richtung gehende Anforderungen haben werden.

4. Verwendete Technologien

4.1 Architektur von Web-Anwendungen

Die firmenweite Erreichbarkeit ist eine der grundlegenden Anforderungen an SWAMP. Deshalb wird die Anwendung nach dem Client-Server-Prinzip entworfen, das bedeutet eine verteilte Nutzung der Applikation durch auf den Rechnern der Anwender installierten Client-Programme, die über das Netz mit einem zentralen Server kommunizieren. Diese Architektur und der Einsatz eines Web-Browsers als Client ergeben günstige Voraussetzungen:

- Da die Kommunikation zwischen Client und Server über das Protokoll HTTP und die Beschreibungssprache HTML erfolgt, ist sichergestellt, daß jeder Nutzer über das Intranet mit einem Browser seiner Wahl auf den vollen Funktionsumfang der Anwendung zugreifen kann. Somit entfällt das Entwickeln und Installieren einer Client-Anwendung, es werden nur die Oberflächen erstellt, die dann im Browser angezeigt werden. Damit wird die Datenmenge auf die übertragenen HTML-Seiten beschränkt, und schnelle Zugriffszeiten sind sichergestellt.
- Die Bedienung von browserbasierten Anwendungen ist allen Mitarbeitern intuitiv geläufig, und es ist nicht nötig, neue Bedienelemente (sog. Widgets) einzuführen, da die Standardfunktionalität ausreicht, um SWAMP zu bedienen, z. B. Vor-Zurückbutton, Formularfelder, Hyperlink-Funktionalität, Scrolling.
- Durch die zentrale Datenhaltung auf dem Server können alle Nutzer jederzeit auf den aktuellen Status des Systems zugreifen. Inkonsistenzen werden von der Programmierung des Servers und der Datenbank verhindert. Interne Änderungen der Struktur des Servers werden von den Clients nicht wahrgenommen, falls sich die Schnittstelle nicht ändert. Dadurch wird eine größere Flexibilität erreicht, da z. B. mit wenig Aufwand eine neue Datenbank eingebunden, oder die Anwendung um Funktionalität erweitert werden kann.

Die gebräuchlichste Variante, bzw. Erweiterung der Client-Server-Anwendung, ist die Three-Tier (dt.: Drei Schichten)-Struktur (siehe Abb. 4.1), die durch den Einsatz von J2EE-Komponenten (siehe 4.3.1) unterstützt wird. Durch die Trennung der Komponenten auf verschiedene Ebenen mit genau definierten Schnittstellen wird sowohl eine leichtere Wart-, Austausch- und Erweiterbarkeit der einzelnen Komponenten, als auch eine erhöhte Sicherheit durch die gegenseitige Abschottung erreicht. Jede der drei Schichten kann auf einem separaten System ausgeführt werden. Der *Client-Tier* wird durch ein auf einem per Netzwerk mit dem Middle-Tier verbundenen Rechner laufendes Programm, meistens ein Web-Browser, dargestellt. Die Kommunikation zwischen ihnen erfolgt typischerweise über das Protokoll HTTP und die Darstellung über die Beschreibungssprachen HTML bzw. XML.

Im *Middle-Tier* wird die Funktionalität der J2EE-Komponenten durch sog. Container (z. B. Servlet-Container, EJB[1]-Container) bereitgestellt. Der Container stellt Schnittstellen zur Kommunikation

[1] Enterprise Java Beans, http://java.sun.com/products/ejb/

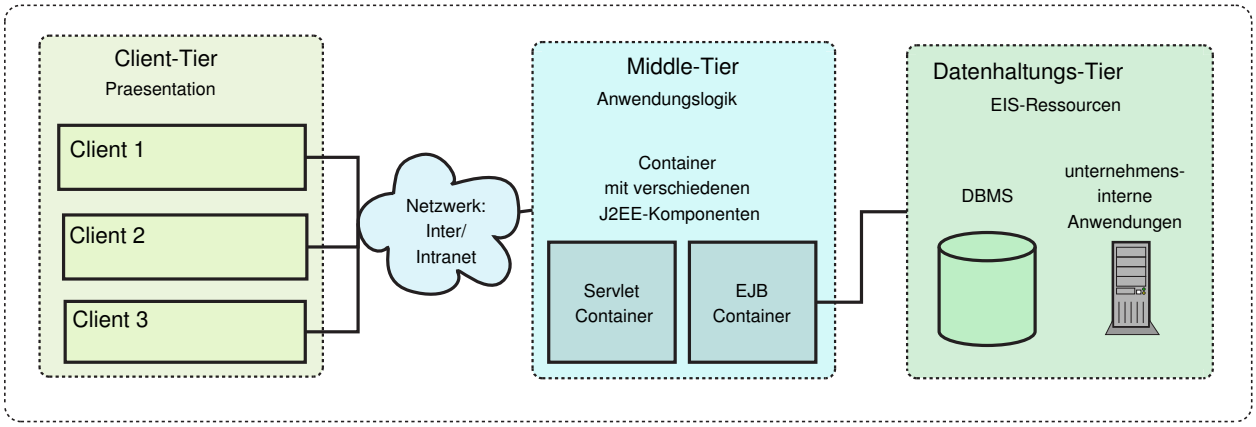


Abbildung 4.1: Three-Tier-Architektur

mit Diensten der Plattform und mit anderen Komponenten der Anwendung her und kann den in ihm laufenden Komponenten Ressourcen zuteilen. Die MVC-Architektur (siehe 4.2) kann als konsequente Weiterführung dieser Komponententrennung innerhalb des Containers angesehen werden, weshalb man dann insgesamt auch von einer n-Tier-Architektur spricht. Der Middle-Tier wird auch oft als Application-Server bezeichnet, so z. B. in den kommerziellen Produkten IBM Websphere[2] oder dem von SUN selbst entwickelten Java System Application Server[3].

Der *EIS(Enterprise Information Systems)-Tier* enthält die Daten, die der *Middle-Tier* zur Bearbeitung der Anfragen benötigt. Meistens kommt hier ein DBMS (Database Management System) zum Einsatz, z. B. Oracle oder MySQL (siehe 4.3.5), oder Verzeichnisdienste, wie z. B. LDAP, die die Konsistenz und Integrität von sehr großen Datenmengen gewährleisten. Auch nicht direkt in die Anwendung integrierte, unternehmensinterne Anwendungen zählen zu dieser Schicht.

4.2 Model-View-Controller(MVC)-Modell

Entwurfsmuster zur Trennung der eigentlichen Kernkomponenten werden in der modernen Software-Entwicklung schon seit einiger Zeit erfolgreich eingesetzt. Ein Modell zur Trennung von Daten, Layout und Logik bei Anwendungen mit grafischer Benutzeroberfläche ist die MVC(Model-View-Controller)-Architektur. Die damit erreichbare Entkoppelung der Komponenten schafft eine einfachere Wartung und eine höhere Wiederverwertbarkeit der Komponenten. Es folgt eine Erklärung der im MVC-Muster separierten Objekte und ihrer Funktion:

- Das Modell (Datenhaltung)
  enthält den Zustand einer Anwendung. Die dauerhafte Persistenz dieser Zustandsdaten wird über den Zugriff auf einen Speicher, z. B. eine Datenbank oder ein LDAP-System, sichergestellt.

- Der View (Darstellung)
  dient zur Darstellung des momentanen Zustands der Anwendung und nimmt Eingaben vom Benutzer entgegen, die er zum Controller weiterleitet. Ziel der MVC-Architektur ist die Eliminierung jeglicher Programmlogik aus der View-Komponente.

- Der Controller (Ablaufsteuerung)
  leitet Aktionen des Views auf entsprechende Funktionen des Modells weiter und definiert somit, welche Funktionalität dem View zur Verfügung steht. Er wählt den passenden View für die Darstellung des Modells aus und stellt ihm die benötigten Daten zur Verfügung.

[2] http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv
[3] http://www.sun.com/software/products/appsrvr/home_appsrvr.html

Durch die Trennung von Design und Logik wird die Zusammenarbeit von verschiedenen Entwicklern vereinfacht, da Änderungen in einer der drei Komponenten so durchgeführt werden können, daß die anderen davon nicht unmittelbar betroffen sind. Die Integrität der Daten des Modells läßt sich über die definierten Schnittstellen ihrer Bearbeitung über den Controller ohne direkten Zugriff des Views leichter gewährleisten. Im Kontext der vorgestellten Three-Tier-Architektur stellt das MVC-Modell eine Untergliederung des Middle-Tiers dar. Abbildung 4.3 zeigt die vom Turbine-Framework nochmals unterteilte Umsetzung der MVC-Architektur.

4.3 Technologien

4.3.1 Die Programmiersprache JAVA

Für die Entwicklung der Anwendung wird die Programmiersprache Java[4] der Firma SUN eingesetzt. Hierzu einige Erklärungen: Ursprünglich wurde Java dazu entwickelt, Programme direkt über das Internet laufen zu lassen, sog. Applets. Ein Applet ist ein Java-Programm, das direkt im Browser vom Server geladen und ausgeführt werden kann. Somit entsteht ein völlig neuer Typ von Anwendungen, bei denen das Netzwerk (bzw. das Internet) im Mittelpunkt steht. Java ist eine objektorientierte Sprache, bei deren Entwicklung viel Wert auf Sicherheit gelegt wurde. Die Programmierung stabiler und sicherer Anwendungen wird hierdurch erleichtert. Java entstand unter dem Einfluß der Programmiersprachen C und C++, was man der Syntax deutlich ansieht, weist aber einige Unterschiede zu ihnen auf und integriert auch bewährte Konzepte anderer Sprachen. Der Code wird nicht direkt in Maschinencode kompiliert, der nur auf bestimmten Rechnerarchitekturen ausführbar wäre, sondern es werden *.class* Dateien in einem sog. *Bytecode* erstellt. Zur Ausführung des Bytecodes ist eine *virtuelle Maschine* notwendig, die den Code interpretiert und seine Laufzeitumgebung darstellt. Der Bytecode ist also plattformunabhängig und kann unter allen Systemen ausgeführt werden, für die es eine Implementierung der Java Virtual Machine gibt[5]. Diese Architektur ermöglicht einen Gewinn an Sicherheit, da die Java-Programme innerhalb der Laufzeitumgebung in einer sog. *Sandbox*, mit einem eigenen Speicherbereich ablaufen und sich so vom Rest des Systems abschotten lassen. Die Methode des interpretierten Codes birgt allerdings auch einen Geschwindigkeitsnachteil, da direkt in maschinenlesbarer Form vorliegender Code schneller vom System bearbeitet werden kann. Java wird oft als *einfache* Sprache bezeichnet, da es keine Zeiger-Arithmetik gibt. Objekte werden als typsichere Referenzen verwaltet. Dies ist nicht als Nach- sondern eher als Vorteil zu sehen, da so robuste Programme in relativ kurzer Zeit entstehen können und der Programmierer zur *sauberen* Art und Weise der Strukturierung seines Code angeleitet wird. Weitere Merkmale der Sprache sind ihre Multithreading- und Ausnahmebehandlungs-Möglichkeiten sowie die Speicherverwaltung mit ihrer automatischen *Garbage Collection*. Ausführliche Beschreibungen dazu befinden sich hier[6].

Im Laufe der Zeit hat sich JAVA weiterentwickelt und wird von SUN in verschiedenen Versionen angeboten:

- Java2 Standard Edition (J2SE)
- Java2 Enterprise Edition (J2EE)
- Java2 Micro Edition (J2ME)

Während die Standard-Edition die Basisfunktionalität für Client-Anwendungen zur Verfügung stellt, definiert die Enterprise Edition[7] einen Standard zur Implementierung komponentenbasierter Server-Anwendungen (siehe auch Abbildung 4.1) mit Bibliotheken zur verteilten Entwicklung,

[4] http://java.sun.com
[5] Momentan sind dies u. a. : Windows, Linux, Solaris, MacOS
[6] http://www.galileocomputing.de/openbook/javainsel3/
[7] http://sun.java.com/j2ee

z. B. Datenbankzugriff per JDBC[8], Enterprise Java Beans, WebServices und Java Servlets/Java Server Pages. Die Micro-Edition stellt eine virtuelle Maschine für eingebettete Systeme bereit, z. B. für Mobiltelefone, die mit sehr begrenzten Ressourcen auskommen müssen.

Somit bietet Java hervorragende Voraussetzungen zur Programmierung von Netzwerk- bzw. Internet Anwendungen auf Basis der Client-Server-Architektur, die bei SWAMP zum Einsatz kommt. Die Voraussetzungen hierfür sind gegeben durch JAVA-Servlets und die sehr große Vielfalt der verfügbaren Bibliotheken, die benötigte Funktionalität in hoher Qualität und mit wenig Implementationsaufwand für den Entwickler einbringen.

4.3.2 Java Servlets

Java-Servlets sind in Java geschriebene Klassen, die dazu dienen, die an einen Servlet-Container gestellten Anfragen der Clients mit einer dynamisch generierten Antwort zu bedienen. Die Definition eines Servlets nach der J2EE-Spezifikation von Sun Microsystems lautet:
„A servlet is a Java technology based web component, managed by a container, that generates dynamic content."
Anders als beim Abrufen von statischen Dokumenten bei einem *einfachen* Webserver ohne Funktionen zur dynamischen Inhaltserzeugung, können so personalisierte Ansichten auf den aktuellen Zustands des Systems und den momentanen Inhalt der Datenbank erstellt werden. Eine andere Technologie, die das gleiche Ziel verfolgt, ist z. B. die im Apache integrierte CGI[9]-Schnittstelle, die die Ausführung von Skripten ermöglicht. Java bietet im Unterschied zu diesem Ansatz eine deutlich schnellere und leistungsstärkere Lösung, da ein Servlet der JVM[10] bereits im Bytecode vorliegt und somit nicht mehr interpretiert werden muß und nicht wie bei CGI-Skripten für jeden Aufruf ein neuer Prozess gestartet werden muß, sondern im Speicher gehalten wird. Ein Servlet muß das Interface-Servlet implementieren, um mit dem Web-Container über diese API zu kommunizieren. Der etwas umständlichen Erzeugung von Textausgaben in Java mit vielen `println()` Anweisungen kann mit dem Einsatz von Java-Server-Pages begegnet werden, das sind um Java-Anweisungen erweiterte HTML Seiten, die *on the fly* bei ihrem ersten Aufruf vom Container in ein entsprechendes Servlet übersetzt werden. Durch die Vermischung von Darstellung und Programmlogik ist es bei komplexeren Anwendungen notwendig, bestimmte Architekturmodelle (vgl. 4.2) anzuwenden, die eine saubere Trennung der Komponenten ermöglichen. Die konkrete Implementierung einer Servlet-Klasse[11] erfolgt über das vom Container bereitgestellte Objektmodell und über Schnittstellen zum Zugriff auf das Anfrage-Objekt (Request) und zur Erstellung eines Antwort-Objekts (Response). Jedes Servlet erbt direkt oder indirekt von der Klasse `javax.servlet.http.HTTPServlet` und implementiert u. a. die Funktionen `doGet` und `doPost`, die bei den entsprechenden HTTP Befehlen vom Servlet-Container aufgerufen werden. Mit der Klasse `javax.servlet.Servlet` ist es auch möglich, Servlets für andere Protokolle zu erstellen. Die Servlet-API stellt zusätzlich Features zur Behandlung von Cookies und zur Verwaltung von Benutzer-Sitzungen zur Verfügung. Da die Schnittstellen zwischen Servlet und Servlet-Container in der J2EE-Spezifikation festgelegt sind, ist es möglich, standardkonforme Servlets in jedem Container auszuführen, der den Standard unterstützt. Somit ist eine Portierbarkeit zwischen Produkten verschiedener Hersteller möglich, z. B. Apache Tomcat (siehe 4.3.4), Macromedia JRun[12] oder IBM Websphere[13].

4.3.3 XML

XML[14] ist ein flexibler Standard zur Definition von textbasierten Auszeichnungssprachen. Es handelt sich um eine sog. Metasprache, da nicht vorgeschrieben ist, welche Elemente ein Dokument

[8] Java Database Connectivity
[9] Common Gateway Interface
[10] Java virtual Machine, siehe 4.3.1
[11] Technische Spezifikation (Servlet API): http://java.sun.com/products/servlet/reference/api/index.html
[12] http://www.macromedia.com/software/jrun/?promoid=home_prod_jr_100803
[13] http://www-306.ibm.com/software/webservers/appserv/was/
[14] eXtensible Markup Language, Spezifikation unter http://www.w3.org/TR/2004/REC-xml11-20040204/

beinhalten muß, sondern auf welche Art und Weise diese Elemente definiert werden. Der Vorteil einer Speicherung von Daten in einem XML-Format liegt in der Trennung ihres Inhalts von der Darstellung. So können aus einer XML-Datenbasis verschiedene Ausgabeformate erstellt werden. Die Struktur eines XML-Dokuments besteht aus hierarchisch strukturierten *Tags* und zugehörigen *Attributen*, die beliebig tief ineinander verschachtelt sein können.

Listing 4.1: Beispiel XML-Datei

```
1 <tag1 attribut1="wert1">
2     <tag2>
3     Text
4     </tag2>
5 </tag1>
```

Wie in Listing 4.1 erkennbar, muß jedes geöffnete Start-Tag durch ein entsprechendes, mit „/“ eingeleitetes End-Tag abgeschlossen werden, und die Bereiche verschiedener Tags dürfen sich nicht überlappen. Erfüllt ein XML-Dokument diese Regeln, nennt man es *wohlgeformt* (*well-formed*).

Die Verwendung von XML-Formaten als Input für das System SWAMP gestaltet sich durch den Einsatz des XML-Parsers SAX[15] relativ unproblematisch, da mit ihm ein direkter Zugriff auf die Elemente der eingelesenen XML-Daten und eine einfache Überprüfung des Dokuments auf seine Wohlgeformtheit möglich ist.

Um eine funktionierende Kommunikation zwischen verschiedenen Systemen auf XML-Basis zu ermöglichen, muß die Reihenfolge der Tags und die Art des Inhalts ihrer Attribute definiert werden. Eine solche Struktur-Beschreibung eines XML-Dokuments nennt sich DTD[16].
Im Projekt SWAMP wird das Format XML unter anderem für den Input der Workflow-Definitionen und für die Kommunikation mit anderen Systemen genutzt (siehe hierzu auch die Kapitel 5.1.4 und 5.1.12).

4.3.4 Apache-Jakarta-Projekte

Die Apache Software-Foundation[17] ist 1999 hervorgegangen aus der Apache-Group, einer Gemeinschaft von Entwicklern, die sich ursprünglich zusammenfand, um einen leistungsfähigen Webserver auf Basis des CERN-Webservers von Tim Berners-Lee zu entwickeln. Durch die stetige Erweiterung des Servers entstand das Wortspiel *„a patchy Server“*[18], woraufhin sich der Name *Apache HTTP-Server* auch als offizieller Name für den Server durchsetzte. Der Apache hat seitdem eine erfolgreiche Entwicklung hinter sich und ist seit 1996 der meistgenutzte Webserver im Internet. Diese Position erreicht er durch seine Stabilität und hohen Verfügbarkeit und durch die Möglichkeit, seine Funktionalität durch Module (z. B. mod_perl, mod_php, mod_python, mod_ssl) zu erweitern. Maßgeblich zu seinem Erfolg beigetragen hat die Tatsache, daß er unter einer Open Source Lizenz, der sog. Apache-Lizenz, steht, und somit frei verfügbar ist. Sie ermöglicht Einblick und Änderung der Sourcen und erlaubt die Integration auch in kommerzielle Produkte. Mit der Gründung der Apache Software-Foundation erweiterte sich auch das Umfeld des Projektes, das seitdem nicht mehr nur den HTTP-Server entwickelt, sondern die Basis für weitere ehrgeizige und qualitativ hochwertige Softwareprojekte bietet. Eines dieser Projekte ist das Jakarta[19]- Projekt, das die Grundlage für weitere Sub-Projekte auf Java-Basis bietet. Ziel von Jakarta ist, serverseitige, kostenlose und hochqualitative Lösungen in Java zu erstellen, die unter der Apache-Lizenz veröffentlicht werden und frei zugänglich sind. Populäre, unter dem Jakarta Projekt entstandene Anwendungen sind z. B. die Servlet Engine Jakarta Tomcat, das Java Build Tool ANT und der XML Parser XERXES.

[15] Simple API for XML, http://www.saxproject.org/
[16] englische Abk. für *Document Type Definition*
[17] http://www.apache.org
[18] dt.: Ein (oft) ausgebesserter Server
[19] http://jakarta.apache.org, Der Name Jakarta entstand aufgrund des Namens des Konferenzraumes, in dem die Apache Group und SUN ihre Zusammenarbeit beschlossen. Zweifellos wurde die Namensfindung durch die Tatsache erleichtert, daß Jakarta die Hauptstadt Indonesiens ist, und auf einer Insel namens *Java* liegt. (Nach [Schm01])

In SWAMP wird reger Gebrauch dieser Projekte gemacht, denn es ist sehr hilfreich, Programme mit einer großen Entwickler-Gemeinschaft zu nutzen, da man hierzu viele Hilfestellungen in einschlägigen User-Foren und Mailinglisten finden kann. Zudem spart man in der Entwurfs- und Implementierungsphase Zeit, da der Einsatz bestehender und bereits bewährter Komponenten den Implementierungsaufwand verringert.

Der Servlet Container Tomcat

Der in der Anwendung SWAMP eingesetzte Servlet-Container (siehe auch 4.3.2) ist Jakarta-Tomcat[20] in der momentan aktuellen Version 5. Er stellt eine Referenzimplementierung für Servlets in der Version 2.4 und JSPs in der Version 2.0 nach der Spezifikation von SUN dar. Er implementiert zusätzlich einen eigenen HTTP-(Web-)Server, er kann ihn erreichende Anfragen also auch selbst verarbeiten und beantworten und statische Dateien, z. B. Dokumente oder Bilder ausliefern. Bei komplexeren Anforderungen an die Webserver-Funktionalität besteht die Möglichkeit, ihn über einen JTC-Connector[21] in den Apache Webserver zu integrieren und nur seine Container-Funktionalität (Servlet-Container *Catalina*, JSP-Container *Jasper*) zu nutzen. Die Aufgabe des Webservers ist es, anhand einer Benutzeranfrage zu erkennen, ob diese durch ein Servlet bedient werden soll, und falls ja, die entsprechenden Aufrufparameter an den Catalina Servlet-Container zu übergeben. Ob und welche Anfragen an den Container weitergereicht werden, ermittelt er mit Hilfe der ihm zur Verfügung gestellten Konfiguration, bei Tomcat die Datei `web.xml` (der sog. *Deployment Descriptor*), bei einem Apache Webserver mit JTC-Connector die Datei `workers2.properties`. Der Container kapselt die Anfrage und die Antwort in Objekte und übergibt sie über die in der Servlet API definierten Schnittstellen an das angeforderte Servlet. Nachdem das Servlet ausgeführt wurde, wird aus dem Antwort-Objekt eine Antwort für den Client erstellt und über das Netzwerk zurückgeschickt.

Dabei stellt der Container den in seiner VM laufenden Servlets Möglichkeiten zum Zugriff auf andere J2EE-Komponenten und -Ressourcen zur Verfügung. Innerhalb des Containers können auch Java-Anwendungen ohne direkte Client-Anbindung ablaufen, da der Container auch alle Funktionen einer *normalen* Java-Laufzeitumgebung bietet. Der Lebenszyklus eines Servlets im Container ist durch die J2EE-Spezifikation genau definiert und durchläuft verschiedene **Stationen**:

- Instanziierung und Initialisierung:
Ein Servlet kann von seinem Container bereits beim Start der Anwendung instanziiert werden, oder spätestens dann, wenn es konkret angefordert wird. Die Initialisierung erfolgt über die Methode `init`, die vom Programmierer für Aktionen genutzt werden kann, die nur einmal während des Lebenszyklus des Servlets notwendig sind. Beispiele hierfür sind das Öffnen einer Datenbank-Verbindung oder das Initialisieren von Variablen. Konnte die Initialisierung fehlerfrei beendet werden, ist das Servlet bereit, Benutzeranfragen zu bearbeiten.

- Beantwortung von Anfragen:
Die Bearbeitung von Benutzeranfragen erfolgt über die `service`-Methode. Bei HTTP-Servlets wird diese weitergeleitet an die je nach HTTP-Befehl zuständige Methode, z. B. `doGet` bei einem HTTP-GET-Befehl (dies ist der normale Befehl beim Aufruf einer URL über den Webbrowser). Es obliegt der Entscheidung des Containers, ob bei mehreren eintreffenden Anfragen an dasselbe Servlet mehrere Instanzen von ihm erzeugt werden oder seine `service`-Methode mehrmals aufgerufen wird. Beide Methoden sind einzeln oder in Kombination möglich, je nach Strategie des Containers. Daraus ergibt sich jedoch eine potentielle Gefahr für den Programmierer des Servlets, da es so passieren kann, daß mehrere Threads eines Servlets gleichzeitig laufen und auf dieselben Ressourcen zugreifen. Bei der Implementierung der von `service` aufgerufenen Funktionen ist also unbedingt auf Thread-Sicherheit zu achten.

- Freigabe des Servlets:
Der Zeitpunkt der Freigabe eines Servlets aus dem Container wird vom Container selbst bestimmt

[20] http://jakarta.apache.org/tomcat/index.html
[21] http://jakarta.apache.org/tomcat/tomcat-4.1-doc/config/connectors.html

und kann z. B. wegen Speicherbedarfs oder nach einer gewissen Zeit der Inaktivität des Servlets geschehen. Das Servlet kann frühestens dann freigegeben werden, wenn keine zugehörigen `service`-Threads mehr laufen. Zur Freigabe wird die Funktion `destroy()` des Servlets aufgerufen, in der z. B. Datenbank-Verbindungen geschlossen werden können.

Weitere Informationen zur Konfiguration und Interna von Tomcat findet man unter [Darw03].

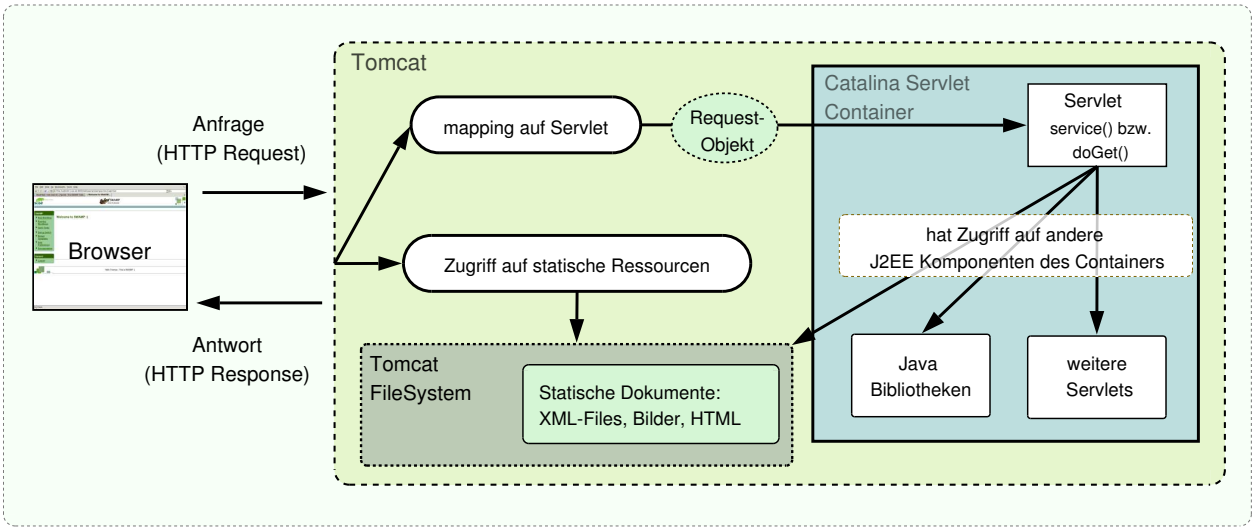


Abbildung 4.2: Tomcat-Struktur

Das Turbine-Framework

Der Begriff *Framework* steht für ein System kooperierender Java-Klassen (bei Turbine ca. 400), die einen Entwickler bei der Implementierung einer Anwendung für den (mehr oder weniger) speziellen Anwendungsfall des Frameworks unterstützen. Jakarta Turbine[22] ist ein servlet-basiertes Web-Application-Framework, das für den speziellen Anwendungsfall einer Web-Applikation mit Datenbank-Anbindung entworfen wurde. Somit erbt die Anwendung SWAMP die Architektur des Frameworks, das grundlegende Strukturen und Kommunikationsmechanismen vorgibt. Im Gegensatz zu anderen Produkten sind bei Turbine bereits viele Komponenten, die typischerweise innerhalb von Web-Applikationen benötigt werden, sog. Services, enthalten, was das Entwicklerteam mit der Aussage: „*A platform for building applications, not just running them*“ unterstreichen möchte. Auf Web-Application-Frameworks aufbauende Anwendungen werden oft mit dem Schlagwort *Portal* bezeichnet, womit die Integration von applikationsübergreifenden Diensten innerhalb des Frameworks und die Möglichkeit, diese über ein Web-Frontend zu bedienen, gemeint ist. Die Services Turbines, auf denen solche Dienste aufsetzen können, bringen u. a. Module für folgende Zwecke mit:

- Eine ACL[23]-basierte Verwaltung für Benutzer, Gruppen und Rollen
- Einen Scheduler zum zeitgesteuerten Ausführen von Hintergrundprozessen
- Ein XML-RPC-Interface zur Netzwerk-Kommunikation mit anderen Anwendungen
- Localization-Service zum Zugriff der Anwendung auf länderspezifische Ressourcen
- Services zur Integration der Template Engine Velocity, dem Datenbank-Mapping-Tool Torque und einen Logging-Service zur Integration von Log4J

[22] http://jakarta.apache.org/turbine
[23] siehe Tabelle 2.1

Details zu diesen Tools werden in den folgenden Abschnitten separat erklärt.

Die Existenz der Services bietet den Vorteil, daß wiederkehrende Funktionen in Komponenten gekapselt sind und somit einfach wiederverwendet werden können. Dadurch läßt sich der Entwicklungsprozess der Anwendung beschleunigen, und es wird eine höhere Sicherheit erreicht, da kritische Code-Stellen in einzelnen Komponenten liegen und nicht über die komplette Anwendung verteilt sind. Da das Turbine-Framework bereits seit 1999 entwickelt wird und inzwischen in Version 2.3 intensiv von verschiedenen Portalen im Internet verwendet wird, kann man davon ausgehen, daß es als Basis für SWAMP geeignet ist.

Die Turbine-Architektur basiert auf dem MVC-Paradigma (siehe auch 4.2), das eine klare Trennung von Präsentations- und Programm-Schicht ermöglicht. Turbine erweitert dieses Schema noch um einige Aspekte, um den Eigenheiten einer Web-Anwendung Rechnung zu tragen:
Bei klassischen Anwendungen wird der View vom Modell durch einen Event-Mechanismus über Änderungen der Daten informiert, damit er sich neu anzeigt. Dies ist bei einer Web-Oberfläche nicht möglich, da durch das Request-Response-Verfahren von HTTP keine automatische Änderung der Darstellung an den Client weitergegeben werden kann, sondern die Änderung erst für ihn sichtbar wird, wenn der View erneut von ihm angefordert wird. Das Modell wird in der Turbine-Architektur aufgeteilt in sog. *Screen*- und *Action*-Klassen. Die Screen-Klassen bilden den Anlaufpunkt aller eingehenden Requests und übergeben dem View die vom Modell bezogenen, benötigten Daten. Die Action-Klassen stellen wiederverwertbare Funktionalität zur aktiven Veränderung des Modells dar. Der View, bei Turbine erzeugt durch Velocity-Templates (siehe 4.3.4), ist die grafische Darstellung des Ergebnisses. Die Templates werden mit den entsprechenden Inhalten gefüllt, und der erzeugte View wird zurück an den Client gesendet. Diese Umsetzung wird, wegen der Aufteilung des Views in Screen-Klassen und Screen-Templates, von den Turbine Machern als *„MVC 2+1“*-Modell bezeichnet, um darzustellen, daß es sich um eine Erweiterung des *Modells 2* von SUN (eine Umsetzung des MVC-Prinzips mit Servlets, JSPs und EJBs) handelt.

Abbildung 4.3 zeigt die Umsetzung der MVC-Architektur innerhalb des Turbine-Frameworks. Durch Ableiten der erstellten Screen- und Action-Klassen von Turbines Secure-Klassen kann für jeden einzelnen View und für jede Aktion ein bestimmter Sicherheitskontext festgelegt werden, der durch die Benutzerverwaltung des Frameworks definiert werden kann.

Die Zuordnung eines Zugriffs zu einem bestimmten Benutzer ist im Fall einer Web-Anwendung etwas schwierig. Da HTTP ein zustandsloses Protokoll ist, ist es nicht ohne weiteres möglich, mehrere Anfragen in einen gemeinsamen Kontext zu stellen. Da es für das Sicherheitsmodell der Anwendung aber unbedingt notwendig ist, die Anfragen eines bestimmten Benutzers zu identifizieren, stellt Turbine einen Service zur Sitzungsverfolgung (Session Tracking) bereit. Er basiert auf einem nach der Authentifizierung gespeichertem Cookie[24], das eine eindeutige Session-ID zur Zuordnung der Anfrage an eine Session bereitstellt. Die Daten der Session werden auf dem Server verwaltet, und die Anwendung kann über Schnittstellen des Frameworks auf sie zugreifen.

Vergleich mit anderen Web-Application-Frameworks:

Ein Vorteil von Turbine gegenüber anderen Frameworks, wie z. B. Apache Struts[25], ist das bereits vorgesehene benutzer- und rollenbasierte Rechtemodell. Zur Verwaltung ist das einfache Verwaltungstool FLUX integriert, das die Benutzer und Rollen innerhalb einer Datenbank verwaltet. In SWAMP wird dies durch den Usermanager (siehe 5.1.1) erweitert, der zusätzliche Funktionalität bereitstellt und eine Verbindung zu bereits bestehenden Benutzerverwaltungen der SUSE auf LDAP-Basis herstellt. Ein weiterer Nachteil des Struts-Frameworks ist die Umsetzung in der „Modell 2“-Architektur, deren Möglichkeiten der Wiederverwendung von Code im Vergleich zum Modell Turbines begrenzt sind. Ein weiteres, sehr mächtiges Framework ist Apache Cocoon[26], das seine größte Leistungsfähigkeit allerdings in der Ausgabemöglichkeit von verschiedenen Formaten

[24] eine auf dem Client-Rechner gespeicherte Textdatei
[25] http://struts.apache.org
[26] http://xml.apache.org/cocoon

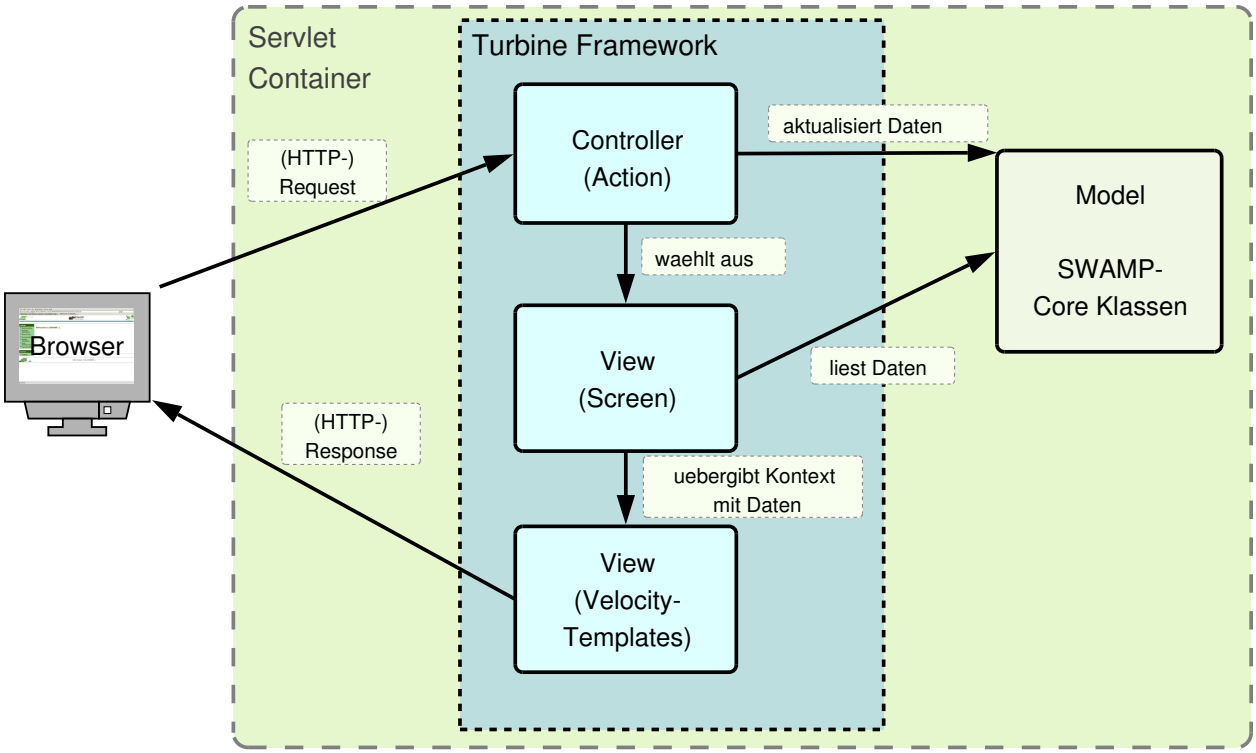


Abbildung 4.3: Von Turbine umgesetztes MVC-Modell, nach [Thei02a]

und dem anschließenden Versenden über verschiedene Protokolle hat. Dies wird im Hintergrund durch den Einsatz von XML- und XSLT-Technologien erreicht, die in einer sog. Pipeline verschiedene Arbeitsschritte durchlaufen. Somit hat Cocoon sein Haupteinsatzgebiet in Anwendungen, die verschiedene Ausgaben, z. B. PDF, HTML und XML, für verschiedene Zugriffsarten erstellen sollen, z. B. Zugriff über Internet-Browser, Handy-WAP-Browser, Drucken eines Kataloges usw.

Momentan bringt Cocoon (für SWAMP) zu viele nicht benötigte Teile mit und macht damit die Integration unnötig kompliziert. Turbine dagegen ist das geeignete Framework für SWAMP, da der Fokus der Entwicklung auf dem Zugriff per Web-Interface liegt und die integrierten Services viele im SWAMP-Konzept vorgesehene Funktionen bereitstellen.

Logging mit Log4J

Log4J[27] ist das Logging-Projekt des Jakarta-Teams. Es stellt eine Logging-API für Java-Programme bereit, über die sie Log-Meldungen zur Laufzeit des Programmes ausgeben können. Der Vorteil dieser Methode gegenüber eigener Konsolen-Ausgaben aus Klassen mit Hilfe des Befehls `System.out.println()` liegt in der einfachen und global für das ganze Projekt definierbaren Art und Weise der Konfiguration. Alle Log-Meldungen werden über eine zentrale Logger-Klasse geleitet, in deren Konfiguration für spezielle Namensräume des Projekts unterschiedliche Logger-Konfigurationen festgelegt werden können. Man spricht deshalb auch von einem hierarchischen Logger. Der Logger kann je nach Relevanz der Nachricht mit verschiedenen Log-Leveln (z. B. DEBUG, INFO, ERROR) angesprochen werden. Über sogenannte *Appender* in der Logger-Klasse kann das Format und der Zielort der Log-Nachrichten festgelegt werden, z. B. Ausgabe auf der Konsole, Textdatei, Syslog oder per E-Mail.

[27] http://logging.apache.org/log4j

Das Datenbanktool Torque

Torque[28] ist ein objektrelationales Mappingtool, das eine Zwischenschicht zwischen den Objekten einer Java-Anwendung und einer Datenbank bereitstellt und die Möglichkeit bietet, diese Objekte transparent in der Datenbank zu sichern (sog. *Persistence Layer*). Hierzu verwendet Torque eine datenbankunabhängige Beschreibung des Datenschemas bzw. Objektmodells im XML-Format[29] und generiert daraus zum einen Java-Klassen und zum anderen eine korrespondierende Tabellenstruktur in der Datenbank. Die erzeugten Klassen stellen eine Abstraktion des Datenbankschemas dar und können als Entitäten des Geschäftsmodells der Anwendung eingesetzt werden. Wenn Torque in einer Anwendung zum Einsatz kommen soll, muß das Objektmodell also bereits in der Entwurfsphase auf den Torque-Einsatz hin abgestimmt werden, da Torque nur die Persistenz für relativ simple Klassen automatisch erreichen kann. Bei Klassen mit Listen als Member-Variablen oder komplexen Objektmodellen ist zusätzliche Arbeit notwendig.

Um die Schnittstelle zum Datenbank-Interface relativ herstellerunabhängig zu halten, werden auf Anwendungsseite statt direkter SQL-Kommandos sog. Criteria-Objekte, auf das Objektmodell übertragene Kommandos, eingesetzt.

Innerhalb der Abstraktionsschicht werden weitere Mechanismen eingesetzt, die dazu dienen, der Anwendung deren Integration zu ersparen, z. B. ein Connection-Pool und Caching-Mechanismen für JDBC-Verbindungen und ein ID-Broker zur Generierung und Verwaltung von Primärschlüsseln für die Tabellen. Von Torque unterstützte Datenbanken sind zur Zeit: MySQL, DB2, Oracle, PostgreSQL, SAPDB und MsSQL.

Skriptsprache Velocity

Velocity[30] ist eine Java-basierte Template-Lösung zur Erstellung von textbasierten Formaten, hier in Form von Webseiten im HTML-Format. Mit Hilfe der einfachen Skriptsprache VTL (Velocity Template Language, ähnlich zu Webmacro[31]) ist es möglich, dynamische Seiten zu erstellen, und durch den geringen Funktionsumfang der VTL ist dies auch ohne Java-Kenntnisse möglich. Die Aufgabe Velocitys ist das Generieren eines angeforderten Dokuments aus dem korrespondierenden Template. Die Templates beinhalten sowohl HTML-Code als auch VTL-Skripte, die bei ihrem Aufruf von der Template-Engine interpretiert werden und so eine Seite mit dynamisch generierten Inhalten an den Benutzer zurücksenden.

Die VTL unterstützt syntaktisch einfache Kontrollstrukturen und den Zugriff auf Objekte, die dem Template innerhalb seines *Kontexts* übergeben wurden. Der Kontext ist ein der Template-Engine zum Zeitpunkt der Interpretation des Templates vorliegendes Hashtable-Objekt mit Objektreferenzen. Innerhalb einer Screen-Klasse fügt beispielsweise
`context.put("foo", new String("Test"))`
ein String-Objekt dem Kontext hinzu. Das aufgerufene Template könnte dann auf das Objekt zugreifen:

Listing 4.2: Velocity-Beispiel

```
#if($foo.equals("Test"))
    ## Kommentar
    <p>Dies ist ein $foo</p>
#end
```

Wie in Listing 4.2 zu erkennen, werden VTL-Kontrollstrukturen mit „#“, und VTL-Kommentare mit „##“ eingeleitet. Im Kontext bereitgestellte Objekte können direkt mit „`$Objektname`“ angesprochen werden. Eine vollständige Beschreibung der von der VTL unterstützten Syntax ist auf der Projekt-Homepage abrufbar.

[28] http://db.apache.org/torque/
[29] Hinweise zum im SWAMP verwendeten XML-Schema finden sich unter 5.1.1
[30] http://jakarta.apache.org/velocity/
[31] http://www.webmacro.org/

Die Bereitstellung von Objekten über den Kontext nennt sich *Push*-Modell. Es gibt einen weiteren Ansatz, das sog. *Pull*-Modell, bei dem das Template die benötigten Objekte selbst anfordern kann. Dieser Ansatz wird bei der Anwendung SWAMP nicht verfolgt, da mit dem *Push*-Modell sichergestellt werden kann, daß keine Daten von der View-Komponente der Anwendung verändert werden.

Velocity ist gut in das MVC-Modell des Turbine-Frameworks integriert und dient zur Darstellung und Aufbereitung der Daten (View-Komponente des Modells). Velocity bietet eine konsequentere Trennung der Geschäftslogik von ihrer Darstellung, als dies mit JSP möglich wäre, da nur der Zugriff auf dem Kontext explizit zur Verfügung gestellten Objekten möglich ist.

Das von Turbine vorgegebene Umfeld (siehe Abbildung 4.3) für die Velocity-Templates besteht aus:

Action-Klassen
Action-Klassen enthalten Funktionen, die eine wiederkehrende Aktion enthalten, die von verschiedenen Seiten aus aufgerufen werden kann. Normalerweise wird eine Action nach dem Absenden eines Formulars im Web-Browser aufgerufen und verändert daraufhin die Daten des Geschäftsmodells.

Screen-Klassen
Zu jedem Template besteht eine korrespondierende Screen-Klasse mit selber Bezeichnung in einem in der Turbine-Konfiguration festgelegten Pfad. Sie hat die Aufgabe, die Zugriffsberechtigung für das angeforderte Template zu überprüfen und ihm den Kontext mit benötigten Objekten zu übergeben.

Velocity-Templates
Das um VTL-Befehle erweiterte HTML-Template zur Darstellung der von der Screen-Klasse aufbereiteten Daten.

4.3.5 Anbindung an MySQL mit JDBC

Die JDBC[32] -API ermöglicht den einheitlichen Zugriff auf ein Datenbank-System aus Java-Klassen heraus über definierte Schnittstellen. Ähnlich dem ODBC-Konzept unter Windows und DBI unter Perl bildet die API eine Zwischenschicht zwischen der Applikation und dem eigentlichen datenbankspezifischen Treiber. Somit wird die Programmierung unabhängig von dem momentan verwendeten Treiber und Datenbank-System. Die Datenbank-Treiber können als Java-Binärdatei oder einem anderen nativen Format vorliegen. Es ist sogar möglich, eine vorhandene ODBC Schnittstelle mit Hilfe einer *JDBC-ODBC Bridge* als Treiber für JDBC zu verwenden. Der Ablauf eines Datenbankzugriffs (aus Anwendungssicht) gestaltet sich sehr einfach:

- Die Verbindung zu einer laufenden Datenbank über die JDBC-Schnittstelle aufbauen,
- einen SQL[33]-Befehl an die Datenbank senden und
- das zurückgelieferte Ergebnis weiterverarbeiten.

Das in dieser Arbeit verwendete DBMS[34] ist die relationale Datenbank MySQL der Firma MySQL AB[35], die in der Open Source Szene momentan populärste Datenbank. Sie wurde vor allem wegen ihres guten Zusammenspiels mit Java und Tomcat über den JDBC-Treiber ausgewählt. Der eingesetzten MySQL-Version 4 fehlen einige Features, die bei kommerziellen DBMS (z. B. Oracle, DB2) enthalten sind, z. B. *Stored Procedures* und *Subqueries*, die in kommenden Versionen ergänzt werden sollen. Für den momentanen Einsatzzweck ist der Funktionsumfang und die Leistungsfähigkeit der Datenbank allerdings ausreichend.

[32] Abk. für Java Database Connectivity (http://java.sun.com/products/jdbc/overview.html)
[33] Structured Query Language
[34] Datenbank Management System
[35] http://www.mysql.com/

4.4 Randbedingungen

Open Source & GPL

Da die Anwendung SWAMP regen Gebrauch von freien und Open Source Bibliotheken und Programmen macht, wird die Anwendung selbst, der SUSE Philosophie folgend, unter der freien Open Source Lizenz GPL[36] veröffentlicht, die hauptsächlich folgende Punkte umfaßt:

- Die GPL garantiert die dauerhafte Freiheit der unter ihr erstellten Software, da jede weitere abgeleitete oder erweiterte Version des Urprungsprogramms nur dann weiterverbreitet werden darf, wenn es ebenfalls unter der GPL steht, und der Quellcode der Öffentlichkeit verfügbar gemacht wird (sog. *Copyleft*-Mechanismus).
- Der Autor der Software übernimmt keinerlei Haftung für die Auswirkungen des Programms.
- Kopien oder veränderte Versionen der Software dürfen umsonst oder gegen Entgelt verteilt werden.

Die Verwendung von freien Hilfsmitteln aus der Open Source Welt verschafft dem Projekt einen Kostenvorteil, der durch die Freigabe des Projekts an die Community zurückgegeben werden kann. Durch die freie Verfügbarkeit wird es anderen Firmen und Benutzergruppen ermöglicht, das Tool selbst einzusetzen und eigene Erweiterungen mit einfließen zu lassen.

Die Begriffe *freie* Software und *Open Source* Software werden oft unterschiedlich interpretiert und mißverstanden. Der Begriff *frei* bezieht sich nicht auf die kostenlose Verfügbarkeit des Programmes (die trotzdem meistens gegeben ist), sondern ist im Sinne der *Freiheit* der Weiterentwicklung des Quelltextes zu sehen. Auf der anderen Seite wird der Begriff *Open Source* oft von Firmen für ihre Programme verwendet, von denen der Quelltext zwar einsehbar ist, aber die Veränderung und Verbreitung desselben durch seine Lizenz verboten sind.

[36] GNU General Public License, http://www.gnu.org/copyleft/gpl.html

5. Umsetzung der Theorie in der Anwendung SWAMP

Im folgenden Kapitel soll die Architektur der Anwendung SWAMP beschrieben werden. Sie wird gegliedert in den sog. *SWAMP-Core*, die Geschäftslogik und Datenhaltung der Anwendung, und den *SWAMP-Client*, der ein Interface für die Kommunikation mit dem Benutzer anbietet. In der vorliegenden Arbeit wird als Client die auf dem Turbine-Framework basierende Anwendung *WebSWAMP* vorgestellt. Es sollen die zentralen Klassen der Anwendung und ihre wichtigsten Funktionen beschrieben werden, um einen Überblick über die internen Abläufe sowie über das Zusammenspiel und die Schnittstellen zwischen dem Client und der zentralen Anwendung zu gewinnen.

Der gesamte Quellcode und die daraus generierte *JavaDoc*-Dokumentation sind auf der beiliegenden CD enthalten.

5.1 SWAMP-Core

Der SWAMP-Core stellt den *M*-Teil des MVC-Modells dar. Er beinhaltet die Geschäftslogik der Anwendung. Er steht auf der einen Seite in Kontakt mit der Datenhaltungsschicht (Datenbank, LDAP-Server) und bietet auf der anderen Seite Schnittstellen an, die dem Client Zugriff auf Funktionen des Systems ermöglichen. Der SWAMP-Core bildet die unter Abschnitt 3.4 beschriebenen Elemente auf konkrete Klassen und ihre Beziehungen ab und sorgt für eine konsistente Datenbasis. Des weiteren gibt es die als Singleton[1] realisierten Manager-Klassen, die die Verwaltung der Workflows und Tasks übernehmen. Abbildung 5.1 zeigt ein UML-Diagramm der Assoziationen der zum SWAMP-Core gehörenden Klassen. Da der SWAMP-Core nur die Umgebung für den Ablauf eines Workflow-Prozesses bereitstellt und keine Logik einzelner Workflows beinhaltet, gibt es zusätzlich das Paket `de.suse.swamp.suse.custom` und das Verzeichnis `conf/custom` für spezielle Klassen und Konfigurationsdateien, die jeweils nur von einzelnen Workflow-Typen verwendet werden.

5.1.1 Manager-Klassen

Da die Manager-Klassen als Singleton realisiert sind, liegt von ihnen nur jeweils eine Instanz im System vor, auf die aus anderen Klassen über den Aufruf der statischen Funktion `<Managername>.getInstance()` zugegriffen werden kann. Die Manager-Klassen sind im Paket `de.suse.swamp.core.container` untergebracht. Durch die Singleton-Realisierung besteht die Möglichkeit, daß die Klassen durch mehrere Threads gleichzeitig angesprochen werden. Deshalb ist innerhalb ihrer Funktionen auf Thread-Sicherheit zu achten, d. h. der gleichzeitige Zugriff auf gemeinsame Variablen und Ressourcen durch verschiedene Threads ist zu verhindern. Dies kann z. B. durch das Schlüsselwort `synchronized` in der Methodendefinition, das eine sequentielle Abarbeitung durch die einzelnen Threads erzwingt, sichergestellt werden.

[1] Ein Singleton (Unikat) ist ein Entwurfsmuster, das zur Anwendung kommt, wenn von einer Klasse nur genau ein Objekt existieren soll.

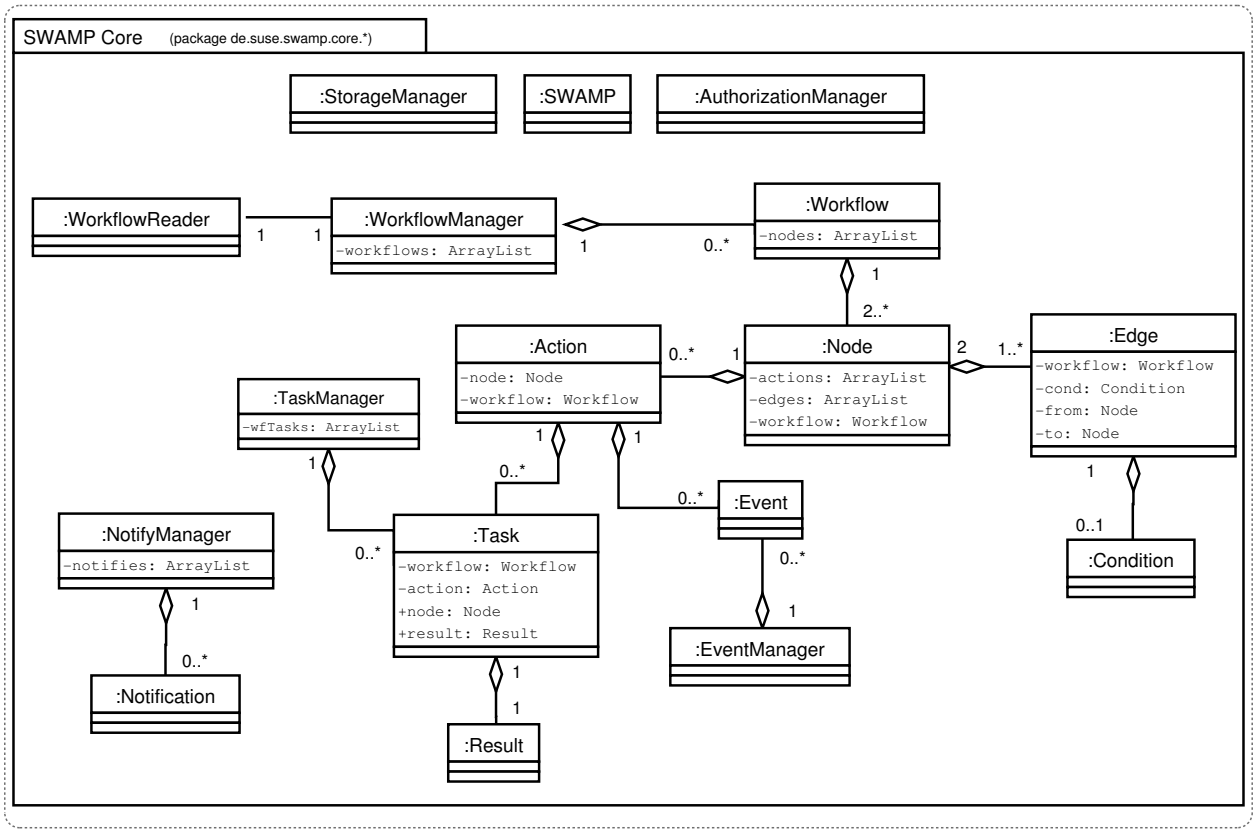


Abbildung 5.1: Übersicht SWAMP-Klassen

SWAMP

Die Klasse SWAMP ist das *Herz* des Core-Teils der Anwendung. Sie stößt beim Start die Initialisierung der Manager-Klassen an, die ihre gesicherten Daten aus der Datenbank wiederherstellen und somit den Status der Anwendung vor ihrer Beendigung rekonstruieren. Über die Klasse SWAMP laufen auch die Zugriffe der Objekte auf Konfigurationsdaten, die in verschiedenen Text-Dateien als Name-Wert-Paare hinterlegt sind (z. B. `conf/defaults`).

Authorizationmanager

SWAMP besitzt keine eigene Benutzerverwaltung, sondern nutzt den SUSE-internen LDAP-Server, auf dem jeder Benutzer mit bestimmten Gruppeninformationen hinterlegt ist. Die Authentifizierung beim Start einer Session in WebSWAMP erfolgt also mit der Benutzername/Passwort-Kombination, die der Benutzer auch im internen Netzwerk zur Anmeldung verwendet. Zusätzlich zu den Nutzerdaten im LDAP-System kann SWAMP spezifische Nutzerdaten und Einstellungen (siehe z. B. definierte Filter unter 5.2.4) in der eigenen Datenbank speichern und dem Nutzer bei einem späteren Login wieder zuordnen.

Jeder Task muß im Moment seiner Erzeugung zur Laufzeit in der Lage sein, sich einer Benutzergruppe oder einem bestimmten Benutzer zuzuordnen, der für seine Ausführung zuständig ist. Zu diesem Zweck ist das Rechtemanagement aufgeteilt in Benutzer, Gruppen und Rollen (sog. ACL[2] -basierte Zugriffsteuerung). Da Gruppen- und Rollen-Verwaltung im Authorizationmanager noch nicht fertiggestellt ist, kommt momentan eine Lösung auf anderer Ebene zum Einsatz:

Um eine dynamische Zuweisung von Benutzern zu Tasks auch während der Laufzeit eines Workflows zu gewährleisten, besitzt jeder Task die Eigenschaft „role“, die auf den Inhalt eines Databits des Workflows verweist. In diesem Databit kann eine mit Komma getrennte Liste von Benutzern

[2] siehe Tabelle 2.1

hinterlegt sein, denen der Task dann zugeteilt ist und die bei seiner Aktivierung durch den Notifymanager eine Nachricht erhalten.

Storagemanager

Der Storagemanager sorgt unter Verwendung des Datenbank-Mapping-Tools Torque (siehe 4.3.4) für die Persistenz der Daten des Systems. Alle Änderungen an Objekten und Daten innerhalb des Systems werden sofort an den Storagemanager weitergegeben und von ihm in der Datenbank gesichert. Das Speichern eines kompletten Workflows wird mit der folgenden Methode erreicht:

```
public int storeWorkflow(Workflow wf)
```

Die Methode speichert den Workflow und alle anhängenden Objekte, z. B. seine Nodes, Tasks usw. rekursiv ab. Der Storagemanager bemerkt selbständig anhand der Objekt-ID, welche der Objekte bereits in der Datenbank bestehen und somit per `update`-Kommando gespeichert werden müssen, und welche noch nicht in der Datenbank existieren und folglich per `insert`-Kommando gespeichert werden müssen. Der umgekehrte Weg funktioniert genauso: Bei der Initialisierung der Managerklassen laden diese mit Methoden der Form

```
public ArrayList load<Objektname>s()
```

die gesicherten Objekte der Datenbank ins System. Da die Datenbank jederzeit den aktuellen Status des Systems beinhaltet, kann ein Datenverlust bei einem plötzlichen Absturz der Anwendung verhindert werden. Intern bildet der Storagemanager die zu sichernden Objekte auf Torque-Objekte ab, die dann über eine Abstraktionsschicht mit der Datenbank kommunizieren. Durch den Einsatz der Torque-Abstraktionsschicht gewinnt SWAMP also Unabhängigkeit von den Eigenarten der eingesetzten Datenbank (ist auch teilweise durch den Einsatz von gegeben, siehe 4.3.5) und ist lediglich auf die von Torque unterstützten Datenbankanbieter festgelegt.

Die Definitionsdatei, aus der Torque die Datenbank und die davon abgeleiteten Objekte erzeugt, findet sich unter `conf/schema/swamp-schema.xml` für die Anwendung und unter `swamp-security-suse-schema.xml` für die Benutzer- und ACL-Daten des Authorizationmanagers. Das Datenbankmodell orientiert sich an der Architektur der Anwendung, muß aber dem Umstand Rechnung tragen, daß Vererbung und die Verwendung von Collection Datentypen in der Datenbank durch das Hinzufügen zusätzlicher Tabellen abgebildet wird. Die von Torque aus den XML-Schemata generierten SQL-Dateien liegen unter `torque/sql/` und legen bei ihrem Aufruf durch das DBMS die Datenstruktur an. Abbildung 5.2 zeigt einen Ausschnitt des Modells dieser Datenbankstruktur.

Die Kapselung der Schnittstelle zur Datenhaltung im Storagemanager schafft die Möglichkeit, die Art der Datenhaltung grundlegend zu ändern. So hätte z. B. der Wechsel zu einer Filesystem- oder XML-basierten Speichermethode oder zu einer anderen Art von Datenbank-Management-System nur Auswirkungen auf die Implementierung des Storagemanagers Andere Programmteile wären von dieser Umstellung nicht betroffen.

Notifymanager

Der Notifymanager verwaltet alle vom System nach außen zu sendenden Nachrichten. Er übernimmt z. B. die automatische Benachrichtigung von Personen, die für die Ausführung eines neu aktivierten Tasks zuständig sind, und weitere, in der Workflow-Definition individuell konfigurierbare Benachrichtigungen zu bestimmten Zeitpunkten (siehe Notifyactions, 5.1.4). Die versendeten Nachrichten basieren auf Text-Templates, die in der Workflow-Definition referenziert werden. Die Templates enthalten vorformulierte Texte, die den Empfänger auf das jeweilige Ereignis von SWAMP hinweisen. Um die Texte mit dynamischen Daten zu erweitern, ist es möglich, Platzhalter in der Form

```
$<Datasetname>.<Databitname>
```

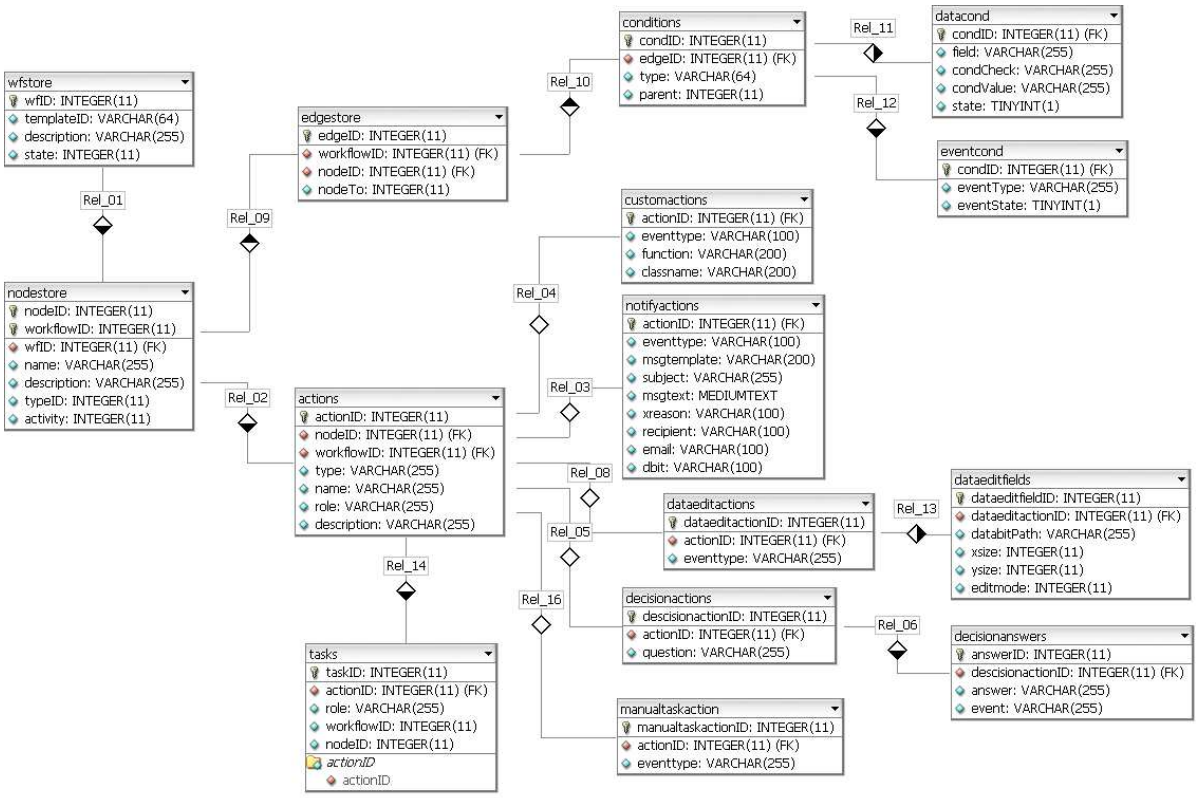


Abbildung 5.2: Datenbankstruktur der Anwendung SWAMP

einzufügen, die im Moment des Versendens der Nachricht aus dem Datapack (siehe 5.1.11), dem Datenbereich eines Workflows, ersetzt werden. Dies übernimmt die Funktion

```
public static String datapackReplace(String text, int workflowid)
```

der Klasse `NotificationTools`, die über einen „Suchen und Ersetzen"-Algorithmus den Text nach zu ersetzenden Werten durchsucht. Somit lassen sich die Benachrichtigungen um aktuelle Workflow-Daten bereichern. Die zu versendenden Nachrichten sind in einer Message-Queue organisiert, der sie durch den Aufruf der Funktion

```
addNotification(Notification notification)
```

hinzugefügt werden. Die Empfängerliste einer Nachricht kann als eine mit Komma getrennte Liste von Benutzernamen und E-Mails oder als Referenz auf ein bestimmtes Databit des Workflows, das eine solche Liste enthalten kann, angegeben werden. Um die Nachrichten für die einzelnen Empfänger zu erzeugen, ist es notwendig, die E-Mail-Adressen der Benutzer vom Authorizationmanager abzufragen. Dies geschieht in der Funktion

```
Notification.setUserInfo()
```

Hier ist es geplant, der Nachricht auch benutzerspezifische Vorgaben hinzuzufügen, die bislang jedoch noch nicht implementiert sind. Mögliche Optionen wären:

- Abruf der Nachrichten per E-Mail-Web-Frontend, über IRC...
- Häufigkeit der Benachrichtigung, z. B. einmal täglich einen Gesamtreport.

Nachdem die Daten der Notification vollständig sind, wird sie in eine Warteschlange eingereiht, die regelmäßig durch den Notifymanager durchgegangen wird, der zu sendende Notifications versendet und aus der Warteschleife entfernt. Im Moment werden alle Nachrichten sofort per E-Mail an den Empfänger versendet. Der Versand erfolgt mit der Bibliothek JavaMail[3]. Nach dem

[3]http://java.sun.com/products/javamail/

erfolgreichen Versand durch den Notifymanager wird die Nachricht mit ihrem Versanddatum gesichert.

Eventmanager

Der Eventmanager ist die Stelle des SWAMP-Cores, an die Events gesendet werden, um sie im System zu verteilen. Er empfängt die Events, führt der History relevante Informationen über die Verarbeitung der Events zu und leitet die Events an die relevanten Stellen im System weiter, wo evtl. auf sie gewartet wird. Über die Methode

```
public void handleWorkflowEvents(ArrayList events, int workflowID)
```

erhält der Eventmanager eine Liste mit Events, speichert ihre Bearbeitung in der History (siehe 5.1.10), und leitet die Events über den Workflowmanager an den Workflow mit der übergebenen Workflow-ID weiter. Events werden also immer an eine bestimmte Workflowinstanz und nicht an alle im System laufenden Workflows gesendet.

Workflowmanager

Methoden zur Manipulation und Erzeugung von Workflow-Instanzen stellt der Workflowmanager bereit. Er stellt bei seiner Initialisierung laufende Workflows mit Hilfe des Storagemanagers wieder her und lädt über den Aufruf

```
WorkflowReader.readWorkflowDef(workflowDefFiles[i]);
```

die verfügbaren XML-Workflow-Templates ins System. Intern führt der Workflowmanager Listen der laufenden Workflowinstanzen und der verfügbaren Workflowtemplates, die eine Client-Anwendung über den Aufruf der Methoden

```
public ArrayList getWorkflowTemplateNames()
public synchronized WorkflowTemplate getWorkflowTemplate(String name)
public synchronized ArrayList getWorkflows()
public synchronized Workflow getWorkflow(int id)
```

abrufen kann. Um einen Workflow zu starten oder zu beenden, stehen folgende Funktionen zur Verfügung:

```
public Workflow startWorkflow(String name) throws Exception
public Workflow prepareWorkflow(String name) throws Exception
public synchronized void finishWorkflow(Workflow wf)
```

Die Methode `prepareWorkflow` setzt den Status eines Workflows auf *prepared*, d. h. es wird bereits eine Workflowinstanz aus dem Template erzeugt, aber der Startknoten wird noch nicht aktiviert. Er ist also in einer Wartestellung. Diese Funktion ist primär dafür gedacht, bereits mit Zusatzdaten gefüllte Workflows anzulegen, um diese dann zu einem bestimmten Zeitpunkt starten zu können.

Workflowreader

Die Aufgabe des Workflowreaders ist es, aus den im Dateisystem vorliegenden XML-Workflowdefinitionen Workflowtemplate-Objekte zu erstellen und sie an den Workflowmanager zu übergeben, von wo aus sie dann gestartet werden können. Er implementiert die zwei Methoden

```
public void readWorkflowDef(File file) throws Exception
public WorkflowTemplate getWorkflowTempl(),
```

um die XML-Datei einzulesen und das generierte Template zu übergeben. Das XML-Dokument wird zuerst mit Hilfe der DTD (siehe `dtds/workflow.dtd`) auf seine Wohlgeformtheit validiert und dann geparst. Das Parsen der XML-Datei geschieht in der inneren Klasse

```
class WorkflowHandler extends DefaultHandler,
```

die mit Hilfe der Bibliothek SAX[4] das XML-Dokument einliest. SAX bildet nicht, wie bei der DOM[5]-Methode üblich, eine komplette Baumstruktur des XMLs im Speicher ab, sondern generiert Ereignisse, auf die man reagieren kann. So werden z. B. beim Öffnen und Schließen von Tags im XML die Funktionen

```
public void startElement(String uri, String localName, [...])
public void endElement(String uri, String localName, String qName)
```

des Workflowhandlers aufgerufen, der dann die Objekte aus ihrem Inhalt erzeugen kann. Vorteil dieser Methode ist der geringe Speicherbedarf und ihr hoher Durchsatz. Allerdings ist der Nachteil, daß in einem Dokument nicht mehr *zurückgeschaut* werden kann, sondern der Parser die Datei von oben nach unten durchläuft.

Template-Funktionalität

Aus den XML-Workflow-Definitionen werden bei ihrem Einlesen ins System SWAMP sog. Template[6]-Objekte erstellt, aus denen erst beim Starten eines Workflows eine neue Workflow-Instanz erstellt wird (siehe Abbildung 5.3). Ein Workflowtemplate besteht wie ein normaler Workflow auch aus Nodes, Edges, Conditions, Actions und einem Datapack, nur eben aus ihren korrespondierenden Template-Klassen. So erhält man die Möglichkeit, Änderungen an den Workflowdefinitionen vorzunehmen, z. B. zusätzliche Knoten einzufügen oder eine andere Umgestaltung des Workflowablaufs vorzunehmen, ohne dadurch bereits gestartete Workflowinstanzen zu beeinträchtigen. Bereits gestartete und aus der Datenbank wiederhergestellte Workflows laufen nach den Regeln weiter, die ihnen bei ihrer Erstellung aus der Workflowdefinition mitgegeben wurden. Die Templates der Workflowdefinitionen werden im Workflowmanager verwaltet und enthalten alle Informationen, die nötig sind, einen Workflow zu generieren. Die Workflow-Objekte und ihre Templates haben aber nach dem Start eines Workflows keinerlei Referenzen mehr aufeinander.

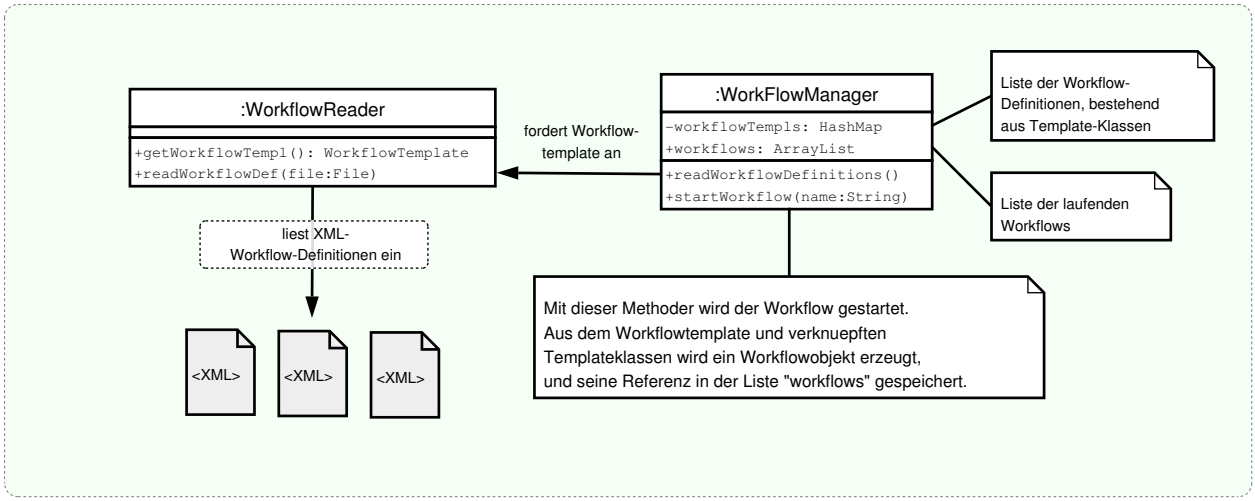


Abbildung 5.3: Workflow-Templates

Taskmanager

Der Taskmanager dient zur Verwaltung der im System auf ihre Bearbeitung wartenden und der bereits bearbeiteten Tasks (siehe 5.1.5). Bei Usertasks wartet er auf die Ausführung durch das Frontend, z. B. durch WebSWAMP. Bei Systemtasks wird die Bearbeitung durch das zuständige Subsystem erwartet. Er bietet Funktionen zum Abrufen einer Liste aller aktiven Tasks eines Workflows oder eines Knotens zur Anzeige und Bearbeitung im Client:

[4]Simple API for XML, http://www.saxproject.org
[5]Document Object Model, http://www.jdom.org/
[6]dt.: Vorlagen

```
public Task getTask(int taskId)
public ArrayList getAllUserTasks()
public ArrayList getAllWorkflowTasks()
public ArrayList getTasksForWorkflowNode(Node node)
public ArrayList getUserTasksForWorkflow(Workflow workflow)
```

Nach Abarbeitung eines Tasks beendet er diesen und übermittelt die vom Task generierten Events an den Eventmanager. Zusätzlich besitzt er Funktionen zum Abbruch eines einzelnen Tasks und aller zu einem bestimmten Knoten gehörigen Tasks.

```
public void finishTask(int id) throws Exception
public void cancelTasksForWorkflowNode(Workflow wf, Node node)
```

Die Methode

```
public synchronized void addTask(Task task) throws Exception
```

wird beim Aktivieren eines Knotens aufgerufen und der neu erzeugte Task der Taskliste hinzugefügt. Der Taskmanager stellt fest, ob es sich um einen Usertask handelt, und sendet über den Notifymanager eine Nachricht an die zugeordneten Benutzer, die sie über ihre neue Aufgabe informiert. Falls es sich um einen Systemtask handelt, der automatisch von SWAMP oder einem Subsystem ausgeführt werden kann, so wird dieser automatisch ausgeführt.

5.1.2 Workflow

Das Workflowobjekt repräsentiert eine Workflowinstanz im System. Es besitzt als Variablen hauptsächlich eine Liste seiner Knoten, eine Referenz auf sein Datapack, einen Status und eine Beschreibung. Das Objekt implementiert die Methoden

```
public void start() throws Exception
public void prepare() throws Exception
public void finish()
```

zum Starten und Beenden, und die Methode

```
public void handleEvent(Event e) throws Exception,
```

um Events zu empfangen und an seine aktiven Knoten weiterzugeben. Das folgende Listing zeigt den Ausschnitt der XML-Definition, der die Eigenschaften des Workflows beinhaltet:

```
<workflow name="CDTracker" datapack="cd">

    <metadata>
      <description>Workflow to track CD
        production of CD $CD-Name</description>
      <creator>freitag</creator>
      <history>
        <change who="freitag" when="long ago">Version 0.1</change>
      </history>
    </metadata>

    ...
    [Definitionen der Knoten]
    [Definition des Datapacks]
    ...

</workflow>
```

Mit der Zuweisung `datapack="cd"` wird auf das später in der XML-Datei zu definierende Datapack des Workflows verwiesen. Der `<metadata>`-Bereich enthält diverse Eigenschaften des Workflows und eine History, in der Änderungen des Templates festgehalten werden. Die kompletten Workflowtemplates liegen im Verzeichnis `conf/workflows/` der Anwendung.

5.1.3 Nodes

Ein Node-Objekt repräsentiert einen Knoten innerhalb des Workflowablaufs. Es hat die Eigenschaften *Name*, *Beschreibung* und *Typ*, wobei der Typ des Knotens einen der Werte NORMALNODE, STARTNODE, ENDNODE oder BRANCHENDNODE einnehmen mu. Jeder Knoten besitzt intern Listen der von ihm weiterführenden *Edges*, und die in ihm enthaltenen *Actions*. Die wichtigsten Funktionen eines Nodes sind `activate()` und `deactivate()` zum Betreten und Verlassen eines Knotens. Bei Aktivierung eines Knotens generieren seine enthaltenen Actions auszuführende Tasks. Über die Funktion

```
public ArrayList handleEvent(Event event)
```

werden eintreffende Events an die weiterführenden Edges übergeben, um zu überprüfen ob dadurch neue Knoten aktiviert werden. Das folgende Listing zeigt exemplarisch die Definition eines Knotens in der Workflowdefinition.

```
<startnode name="start">
  <description>Startnode of Workflow XYZ</description>

    ...
    [Definitionen von enthaltenenen Actions]
    [Definitionen von weiterfuehrenden Edges]
    ...

</startnode>
```

Der Typ des Knotens wird hier auf *STARTNODE* festgelegt. Er kann auf die gleiche Weise auf einen anderen Typ festgelegt werden.

5.1.4 Actions

Actions sind die Aktivitäten, die bei Erreichen eines bestimmten Knotens auszuführen sind. Sie implementieren das Interface Action, das ihnen u. a. bereits die Funktionen

```
public abstract ArrayList validate(Result r)
public abstract void act(Result r)
public abstract ArrayList getEvents(Result r)
```

zuweist, d. h. die abgeleiteten Action-Klassen müssen diese Funktionen implementieren.
`Validate(Result r)` überprüft anhand eines Result-Objekts, ob die Voraussetzungen für das erfolgreiche Beenden des zugehörigen Tasks gegeben sind. `act(Result r)` führt dann die in der Action-Klasse definierten Aktionen aus. Nach der Ausführung der Action werden die von ihr zu sendenden Events mit der Methode `getEvents(Result r)` abgerufen und über den Eventmanager versendet.

Abbildung 5.4 zeigt die Vererbungsstruktur der verschiedenen Action-Klassen. Durch die homogene Struktur der Action-Klassen mit gleichen Interfaces ist es ohne großen Aufwand möglich, neue Actions zu definieren und damit die Funktionalität des Workflow-Management-Systems zu erweitern. Die folgenden Erklärungen beziehen sich jeweils auf eine der bereits im Paket `de.suse.swamp.core.actions` implementierten Action-Klassen und zeigen auf, mit welchen Befehlen sie in die XML-Workflowdefinition eingebaut werden können.

Dataeditaction

Die Dataeditaction dient der Dateneingabe durch einen Benutzer. Die zu editierenden Daten liegen innerhalb des Datapacks des Workflows. Um die Databits anzuzeigen und zu bearbeiten, wird die Hilfsklasse *Field* verwendet, die die Daten des Databits und Einstellungen zu seiner Bearbeitung beinhaltet. So ist es z. B. möglich, das Ausfüllen eines Feldes verpflichtend (mandatory) zu

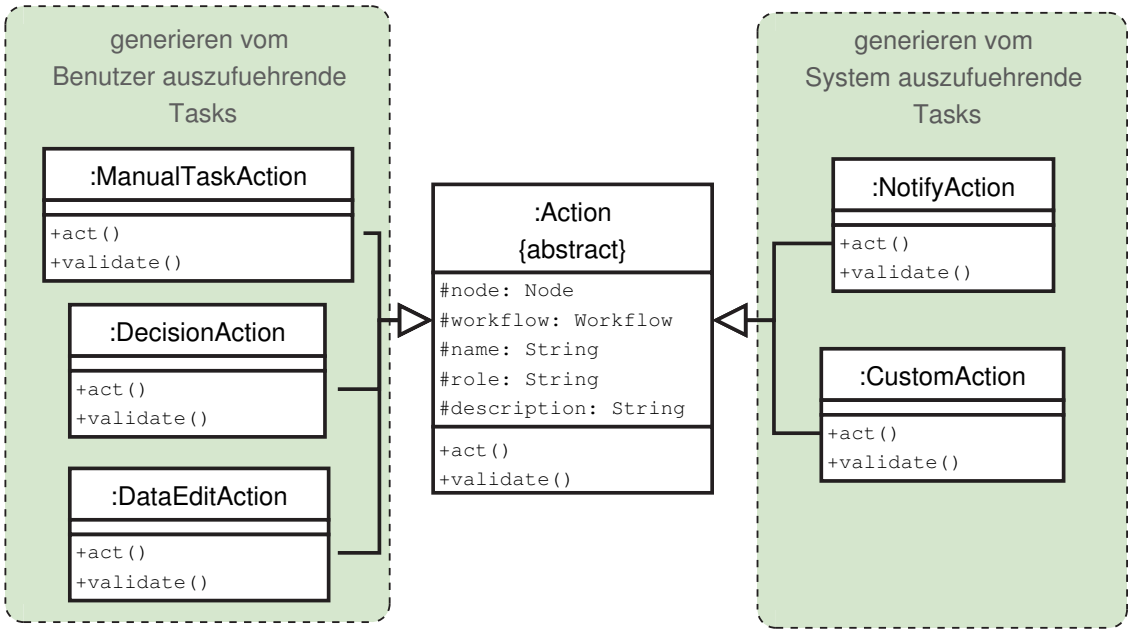


Abbildung 5.4: Vererbungsschema der Action-Klassen

machen und die Art der Anzeige eines Feldes in der Oberfläche (xsize und ysize) festzusetzen. Die Dataeditaction überprüft in der Methode `validate(Result result)`, ob alle mandatory-Felder ausgefüllt wurden und ob die Dateneingaben mit den festgelegten Datentypen der Databits übereinstimmen. Verläuft der Test positiv, wird die Funktion `act(Result result)` aufgerufen, die die im Result enthaltenen Werte in das Datapack des Workflows schreibt, und das in der Action definierte Event sendet. Zusätzlich sendet eine Dataeditaction immer das Event `DATA_CHANGED` bei einer Änderung der Daten, um evtl. wartende Dataconditions über Änderungen im Datapack zu informieren.

```
<dataedit name="initdata" role="Owners.startowners"
        eventtype="INITDATA_CHANGED">
    <description>Please fill in required data</description>
    <field path="Name" xsize="20" mandatory="yes" />
    <field path="Description" xsize="20" ysize="5" mandatory="no" />
</dataedit>
```

Das Listing zeigt die Definition einer Dataeditaction. Der Eintrag *role="Owners.startowners"* referenziert den Inhalt des Databits *startowners* im Dataset *Owners*, das eine Liste der zuständigen Personen für diesen Task beinhaltet. In der Oberfläche wären diese Benutzer dann dazu aufgefordert, die zwei Textfelder *Name* und *Description* auszufüllen, wobei das Feld *Name* ein Pflichtfeld ist. Wurden die Daten eingegeben, wird das Event `INITDATA_CHANGED` an den Eventmanager gesendet, und der Workflow kann entsprechend fortfahren.

Decisionaction

Die Decisionaction bietet dem Benutzer die Möglichkeit, in der Oberfläche eine von mehreren Möglichkeiten zu wählen und je nach der getroffenen Wahl ein anderes Event zu senden. Normalerweise wird eine Decisionaction in einem Knoten verwendet, um auszuwählen, welcher Pfad des Workflows weiter verfolgt wird. (Abbildung der Patterns *Multiple Choice* und *Exklusive Wahl*, siehe 3.2) Die weiterführenden Edges werden mit Conditions für die jeweiligen Events der Auswahlmöglichkeiten versehen. Das Listing zeigt die Definition einer Decisionaction mit der zugewiesenen Rolle (`role`), den Auswahlmöglichkeiten (`answer`) und den jeweils zu sendenden Events (`eventtype`).

```
<decision name="test_cd" role="qa">
    <question>Is Testing in QA department successful?</question>
    <answer eventtype="TEST_OK">CD passed all tests.</answer>
    <answer eventtype="TEST_FAIL">CD failed some tests</answer>
</decision>
```

Manualtaskaction

Die Manualtaskaction dient der Bestätigung eines außerhalb des Systems durchgeführten Vorgangs. Der zugewiesene Benutzer muß den Vorgang in der Oberfläche, z. B. durch einen *OK*-Button, bestätigen, und sendet dadurch den verknüpften Event.

```
<manualtask name="create_ptf" role="hwcert" eventtype="PTF_CREATED">
    <description>Click OK when you have created the PTF.</description>
</manualtask>
```

Notifyaction

Die Notifyaction dient dem Versenden von Nachrichten, wenn ein bestimmter Zustand des Systems erreicht wird. Das Listing zeigt ein Beispiel einer Notifyaction:

```
<notification msgtemplate="MailTemplate" x-reason="einfach so"
    subject="This is the Subject">
    <recipient name="tom" eventtype="TO_NAME_SENT" />
    <recipient email="tom@work.de" eventtype="TO_MAIL_SENT" />
    <recipient dbit="recpipient1" eventtype="RECP1_SENT" />
</notification>
```

Eine Notifyaction kann beliebig viele Empfänger (`<recipient>`-Elemente) haben, an die die Nachricht versendet wird. Der Nachrichtentext wird im Beispiel aus der Textdatei `conf/notifications/MailTemplate` übernommen. Es wäre auch möglich, den Nachrichtentext hier direkt mit dem Attribut `msgtext="Text"` anzugeben. Die Angabe der Empfänger kann über verschiedene Methoden erfolgen, so z. B. die Angabe eines SUSE-Benutzernamens (z. B. `name="tom"`), der dann vom Notifymanager aufgelöst wird, und der die Nachricht dann mit den persönlichen Einstellungen des Benutzers versendet. Mit dem Attribut `email="tom@work.de"` wird die Nachricht an eine E-Mail-Adresse verschickt. Hierdurch ist es möglich, auch automatisiert Nachrichten an externe Systeme zu senden und ihnen so Statusmeldungen oder Daten des Workflows zu übermitteln. Über `dbit="recpipient1"` wird der Notifyaction mitgeteilt, daß sich der Name oder die E-Mail-Adresse des Empfängers in einem Databit mit dem Namen recipient1 befindet. Das Attribut `x-reason` erstellt einen X-Header-Eintrag beim E-Mail-Versand, auf den bestimmte Filterregeln der Empfänger angewandt werden können, z. B. um die E-Mail automatisiert in eine spezielle Ablage zu leiten. Wurde die Nachricht erfolgreich an den Notifymanager übergeben und in seine Nachrichtenwarteschlange eingereiht, so wird der jeweils angegebene Event gesendet.

Customaction

Mit Hilfe der Customaction lassen sich Klassen ansprechen, die ansonsten nichts mit dem System SWAMP zu tun haben. Somit lassen sich Objekte erstellen, die nur sehr spezielle Funktionen einzelner Workflows ausführen und sich deshalb nicht im Core-Paket der Anwendung befinden sollen, da hierdurch das generische Konzept der Anwendung, keine workflowspezifischen Funktionen zu integrieren, unterlaufen würde.

```
<customtask name="ct-name" eventtype="CT_DONE"
    class="de.suse.swamp.suse.custom.MaintenanceAction"
    function="customaction2" description="Custom Task" />
```

Das Attribut `class="..."` bestimmt den Pfad der Klasse, die im Moment der Ausführung des Tasks instanziiert wird, und das Attribut `function="..."` den Namen der Methode, die aufgerufen wird. Diese Methode muß die Form

```
public Boolean customaction (String wfxml)
```

haben, damit die Customaction ihr das zugehörige Workflowobjekt in einer XML-Darstellung übergeben und durch den Boolean-Rückgabewert feststellen kann, ob die Methode erfolgreich ausgeführt werden konnte. Die Objekte der Customaction bekommen nur die XML-Daten und keine Objekt-Referenzen übergeben, damit hier keine externen Klassen Zugriff auf die internen Objekt-Strukturen der Anwendung SWAMP bekommen. Die Instanziierung der externen Klasse innerhalb der Customaction erfolgt über den sog. Java-Reflection-(auch Introspektion genannten)-Mechanismus, der es erlaubt, zur Laufzeit Informationen über die geladenen Klassen der Java-VM abzufragen und Klassen dynamisch zur Laufzeit zu laden.

5.1.5 Tasks

Ein Task ist eine konkrete, von einem Benutzer oder Subsystem zu erledigende Aufgabe. Im Moment der Aktivierung eines Knoten erzeugt dieser iterativ mit dem Aufruf `Action.getTask()` aus allen enthaltenen Actions Taskinstanzen und übergibt sie zur Verwaltung dem Taskmanager (siehe 5.1.1). Momentan wird unterschieden zwischen *Usertasks*, von einem bestimmten Benutzer oder einer Benutzergruppe durchzuführende Aufgaben, und *Systemtasks*, vom System selbständig auszuführende Aufgaben, z. B. automatische E-Mail Benachrichtigungen oder Datenverarbeitung im Hintergrund (siehe Notifyaction, Customaction). Bestandteil jedes Tasks sind Referenzen auf seine Action und auf ein Result-Objekt, mit dessen Hilfe der Task feststellen kann, ob seine Action erfolgreich abgeschlossen wurde oder ob Fehler bei der Bearbeitung auftraten.

Durch die Unterscheidung von Actions und ihre konkrete Repräsentation durch einer Benutzergruppe oder einem Subsystem zugeordnete Task-Klassen, gewinnt das System an Flexibilität, da es möglich ist, aus einer Action mehrere Tasks zu erzeugen. Dies kann z. B. notwendig sein, wenn ein Knoten durch einen Rücksprung im Workflowablauf mehrmals durchlaufen wird oder falls eine Action mehrere gleichartige Tasks für verschiedene Benutzergruppen erstellen soll. Diese Struktur ist auch die Grundlage für eine Integration der *Mehrfache-Instanzen-Patterns* (siehe 3.2). Abbildung 5.5 zeigt den Lebenslauf eines Tasks mit dem deutlichen Unterschied der Bearbeitung von Usertasks durch das Client-Programm und der Bearbeitung von Systemtasks durch das entsprechende Modul des Systems selbst.

5.1.6 Results

Zu jeder Action-Klasse existiert ein korrespondierendes Result-Objekt, anhand dessen der Task überprüfen kann, ob er ordnungsgemäß durchgeführt wurde. So hat z. B. die Dataeditaction ein Dataeditresult, in dem die eingegebenen Werte gespeichert werden, bevor sie mit `validate()` überprüft werden. Dadurch wird verhindert, daß eine Dataeditaction geänderte Werte sofort in die Databits schreibt und somit evtl. inkonsistente und falsche Daten in den Databits stehen.

5.1.7 Events

Ein Event besteht hauptsächlich aus einem Text-String, der seinen Typ beschreibt und mit der Methode `getType()` abgerufen werden kann. Der Event-Typ ist frei wählbar und wird in der XML-Workflow-Definition bei der Definition der Actions gesetzt. Ein feststehender Event-Typ ist `DATA_CHANGED`, der immer bei einer Änderung der Daten des Datapacks gesendet wird.

Abbildung 5.6 beginnt zeitlich dort, wo Abbildung 5.5 endet, und zeigt den Lebenslauf eines Events ab dem Aufruf der Funktion `handleWorkflowEvents()` des Eventmanagers. Falls eine Condition im letzten Schritt neu aktivierte Nodes zurückliefert, beginnt der Ablauf wieder mit der Methode `activate()` der Abbildung 5.5.

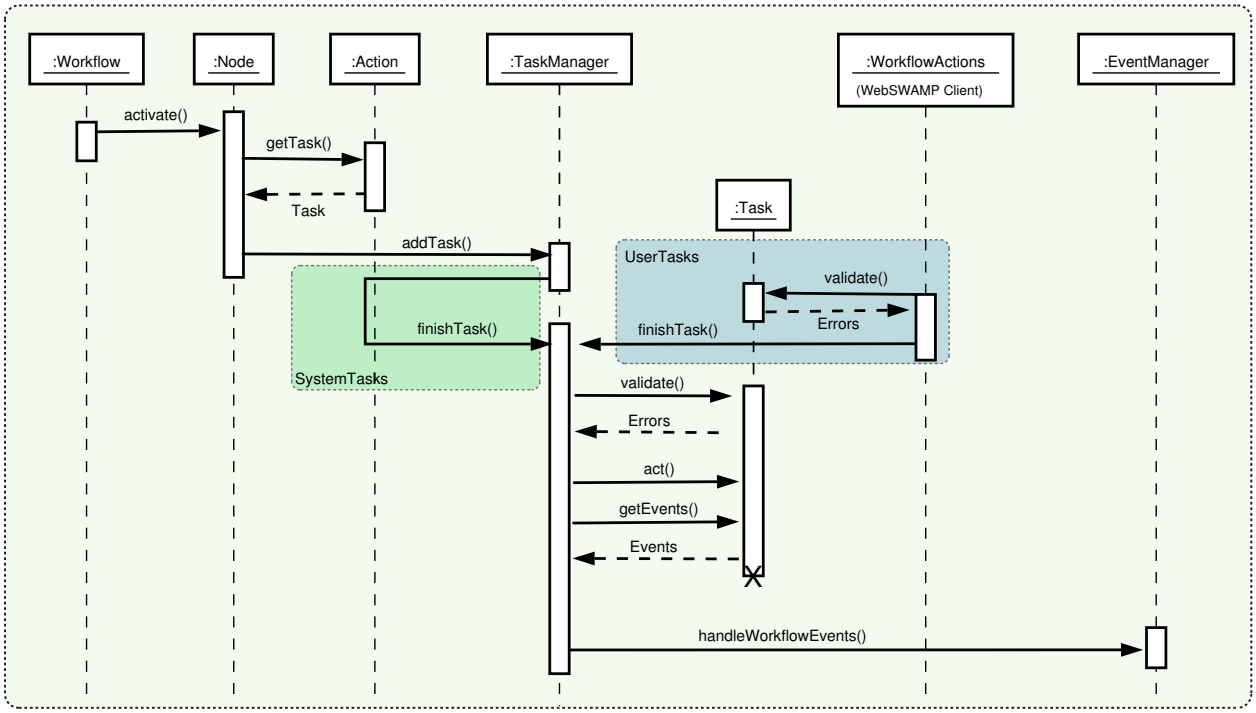


Abbildung 5.5: Lebenslauf eines Tasks

5.1.8 Edges

Eine Edge besteht aus Referenzen auf ihren Start- und Zielknoten und ihrer Condition. Die Definition einer Edge in der Workflowdefinition sieht folgendermaßen aus:

```
<edge to="qanotify">
    [Definition der verbundenen Condition]
</edge>
```

Falls lediglich eine Eventcondition mit ihr verbunden ist, was der Normalfall sein dürfte, ist eine Kurzform möglich:

```
<edge to="startjoin" event="INITDATA_CHANGED" />
```

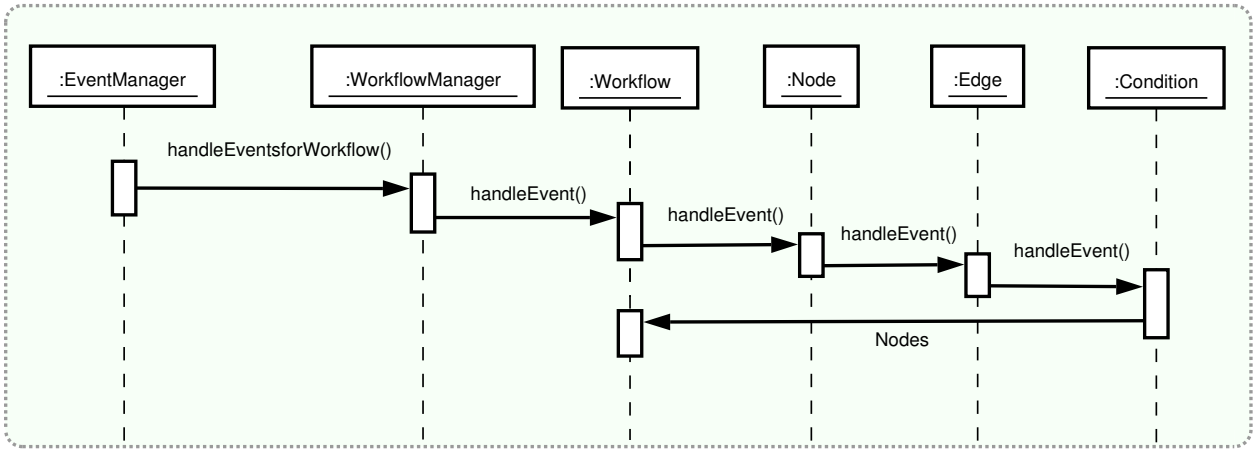


Abbildung 5.6: Lebenslauf eines Events

Durch diese Definition wird automatisch eine Eventcondition für den Event `INITDATA_CHANGED` angelegt.

5.1.9 Conditions

Eine Condition kann durch mehrere Teil-Conditions zusammengesetzt werden, um komplexe Zusammenhänge zu modellieren. Sie enthält die Funktionen

```
public abstract boolean evaluate()
public abstract void reset(),
```

um ihren Status herauszufinden bzw. zurückzusetzen, und die Methode

```
ArrayList getChildConditions(),
```

um auf evtl. mit ihr verknüpfte Unter-Conditions zuzugreifen. Falls eine Condition mehrere Unter-Conditions enthält, ist sie erst dann gelöst, wenn alle ihre Unter-Conditions auf den Aufruf von `evaluate()` mit `true` antworten. Es folgt eine Beschreibung der bereits implementierten Condition-Typen:

Event-Condition

Die Event-Condition wartet auf das Eintreffen eines bestimmten Event-Typs, und ist dann erfüllt, wenn dieser Event eintrifft. Eine Event-Condition wird wie folgt definiert:

```
<event type="BUILDCDDATA_CHANGED" />
```

Data-Condition

Die Data-Condition dient dazu, bestimmte Inhalte des Datapacks auf Änderungen hin zu beobachten und sich bei Änderungen, die ihrer Bedingung entsprechen, zu lösen. Die Condition wartet auf das Event `DATA_CHANGED`, das bei jeder Änderung der Daten vom System zu senden ist. Die Dataeditaction besitzt die Felder `field`, `check` und `value`, in denen bestimmt wird, welches Databit nach welcher Methode mit welchem Wert verglichen werden soll. Bislang ist nur die Methode „regexp“ implementiert, die den Inhalt des Databits mit dem in `value` angegebenen regulären Ausdruck vergleicht. Die Definition einer Data-Condition gestaltet sich einfach:

```
<data field="qarespuser" check="regexp" value=".+" />
```

Hier würde überprüft, ob das Databit „qarespuser“ mindestens ein Zeichen enthält.

AND-Condition

Die AND-Condition besitzt Referenzen auf zwei Teilbedingungen und wird gelöst, sobald diese erfüllt wurden.

```
<and>
    [Definition der 1. Condition]
    [Definition der 2. Condition]
</and>
```

NOT-Condition

Die NOT-Condition besitzt nur eine Untercondition und liefert genau das Gegenteil des Resultats ihrer Untercondition zurück.

OR-Condition

Die OR-Condition erstellt eine logische Oder-Verknüpfung zwischen zwei Unter-Conditions und wird äquivalent zur AND-Condition definiert.

5.1.10 History

Die History dient als Protokoll über sämtliche Vorgänge im System, und ihre Einträge werden automatisch in der Datenbank gesichert. Hier werden die von Benutzern oder vom System durchgeführte Aktionen inkl. ihrem Zeitpunkt und Verursacher protokolliert, um sie später, z. B. zur Fehlersuche, rekonstruieren zu können. History-Einträge sind über eine ID fest mit einem bestimmten Workflow verknüpft und ermöglichen so die Anzeige aller während der Laufzeit eines Workflows aufgetretenen Ereignisse. Das Hinzufügen eines History-Eintrags erfolgt über den Aufruf:

```
HistoryEntry.create("NODE_LEAVE", node.getId(), getId())
```

In die History können Einträge zu den folgenden Ereignissen gemacht werden:

```
WORKFLOW_START, WORKFLOW_END, NODE_ENTER, NODE_LEAVE, TASK_FINISHED
```

5.1.11 Datapack

Ein Datapack ist der *Datenspeicher* einer Workflow-Instanz. In ihm können Text-Informationen hinterlegt werden, die wichtig für die Bearbeitung des Workflows sind, z. B. Namen von zuständigen Personen oder Daten, die dann innerhalb einer Dataeditaction editiert werden können. Intern kann ein Datapack in mehrere Datasets gegliedert werden, die jeweils mehrere, in einem gemeinsamen Kontext stehende Databits enthalten können. Die Speicherung von Daten in einem Databit erfolgt über eine Name-Wert-Zuweisung mit den Methoden

```
public String getValue(String ifield)
public boolean setValue(String ifield, String value),
```

die als `ifield`-Argument den Pfad zum jeweiligen Databit in der Notation `<Datasetname>.<Databitname>` erwarten. Die Speicherung der Daten erfolgt databitintern als String. Es wird jedoch in der Workflow-Definition für jedes Databit ein Typ festgelegt, dessen Einhaltung bei jedem Ändern der Daten des jeweiligen Databits durch eine Methode überprüft wird. Definition eines Datapacks in der XML-Workflow-Definition:

```
<datapack name="l3pack">

    <databit name="MediaSets" description="Databit Bsp" type="number">
            <defaultvalue>20</defaultvalue>
    </databit>

    <dataset name="Bugzilla" description="All Bugzilla Infos in here">
        [Definitionen der Databits]
    </dataset>

</datapack>
```

Die Databits können mit einem Default-Wert vorbelegt werden, der dann beim Start des Workflows bereits vorhanden ist.

Datentypen der Databits

Ob der hinzuzufügende Wert mit ihrem Datentyp übereinstimmt, überprüfen die Databits mit Hilfe der Methode `checkDataType()`. Falls der Wert korrekt ist, wird er in das Databit geschrieben, und falls nicht, eine Exception weitergemeldet. Diese Exceptions können vom Client-Programm ausgewertet werden, um dem Nutzer darzustellen, was an seiner Eingabe inkorrekt war. Bereits implementierte Datentypen sind:

- **boolean** Ein Wahrheitswert mit der Wahlmöglichkeit true/false.
- **number** Eine Zahl.

- **string** Ein Text.
- **person** Der Typ Person enthält den Login-Namen eines SUSE-Mitarbeiters, oder eine E-Mail-Adresse. Die Gültigkeit des Usernamens wird mit Hilfe des Authorizationmanagers überprüft. Ebenso gültig ist hier der Eintrag einer E-Mail-Adresse oder eine mit Komma getrennten Liste mehrerer Usernamen und E-Mail-Adressen.
- **datetime** Eine Datumsangabe im Format JJJJ-MM-TT.
- **bugzilla_id** Eine gültige ID eines Eintrags im SUSE Bugzilla-System. Wird die ID eingegeben, nimmt SWAMP automatisch Kontakt mit Bugzilla auf und holt sich über die XML-Schnittstelle Bugzillas die Informationen des zugehörigen Bugs. Damit diese Informationen in SWAMP gespeichert werden können, muß ein Dataset mit dem Namen „Bugzilla“ bestehen. Innerhalb dieses Datasets können Databits angelegt werden, die die gleichen Namen haben wie die Elemente der XML-Ausgabe Bugzillas. Wird ein übereinstimmender Name gefunden, werden die entsprechenden Daten ins Databit übernommen.
- **enum** Der Datentyp enum dient dazu, Auswahllisten mit fest vorgegebenen Werten zu erstellen.

```
<databit name="Creator" description="The Creator" type="enum">
        <defaultvalue>unassigned</defaultvalue>
        <value type="string">unassigned</value>
        <value type="string">ro</value>
        <value type="string">mls</value>
</databit>
```

  Es können beliebig viele Werte zur Wahl gestellt und ein vorbelegter Default-Wert angegeben werden.

5.1.12 Erstellen der Workflow-Definitionen

Es wäre möglich, die Workflow-Definitionen mit Hilfe eines GUI[7]-Tools zu generieren. Die Erstellung eines solchen Tools hat momentan nur geringe Priorität, da aktuell mit dem System nur wenige Workflow-Typen bearbeitet und die Definitionen hierfür von den Entwicklern selbst erstellt werden. Sobald es außenstehenden Personen ohne die hierfür notwendige Sachkenntnis ermöglicht werden soll, eigene Workflow-Definitionen zu erstellen, ist eine grafische Oberfläche hierfür Pflicht. Da die einzelnen Elemente der XML-Workflow-Definition bereits in den Abschnitten der jeweiligen Objekte beschrieben wurden, ist eine weitere Erläuterung hierzu nicht notwendig. Beispiele für Workflow-Definitionen finden sich in den Projektdateien unter `conf/workflows`.

DTD der Workflow-Definition

Um zu überprüfen, ob eine einzulesende Workflow-Definition der erwarteten Struktur entspricht, überprüft der Workflow-Reader die Dateien mit Hilfe der SWAMP-Workflow-DTD. Ein Beispiel-Ausschnitt der unter `dtds/workflow.dtd` liegenden DTD:

```
[...]
<!ELEMENT workflow (metadata, startnode, node+, endnode+, datapack)>
<!ATTLIST workflow name         ID      #REQUIRED
                   datapack     CDATA   #REQUIRED>
[...]
```

Hier wird definiert, daß das Element `workflow` genau **einen** Startknoten hat, aber beliebig viele Elemente des Typs `node`. Die Attribute `name` und `datapack` müssen vorhanden sein, wobei der Name ein Schlüsselwort (`ID`) ist, das nur einmal in der Datei vorkommen darf.

[7] Graphical User Interface, dt.: grafische Benutzeroberfläche

5.2 Erzeugen des Web-Interfaces mit WebSWAMP

Die Bedienung und Administration der Workflows soll mit Hilfe von verschiedenen, unabhängigen Benutzeroberflächen möglich sein. Diese werden von serverseitigen SWAMP-Client-Modulen erzeugt, die über definierte Schnittstellen mit den Manager-Klassen des SWAMP-Cores kommunizieren. Durch die Kapselung der Geschäftslogik und Daten im SWAMP-Core ist es möglich, unterschiedliche Client-Anwendungen zu erstellen, die alle mit derselben zentralen Anwendung kommunizieren und somit auf derselben Datenbasis arbeiten. Momentan bereits existierende Client-Programme sind zum einen eine lokale Steuerung per Kommandozeile zum Testen bestimmter Funktionen des SWAMP-Cores und zum anderen die mit WebSWAMP generierte Oberfläche zur Bedienung der Anwendung über ein Web-Interface. Weitere Möglichkeiten des Zugriffs, z. B. per E-Mail, IRC oder Textbrowser, sind geplant. Die Priorität liegt aber auf der Implementierung aller Funktionen in WebSWAMP als Referenz-Client.

Das Web-Interface WebSWAMP basiert auf den unter 4.3.4 beschriebenen Anwendungen Jakarta Tomcat, Turbine und Velocity. Die Erstellung der Oberfläche erfolgt mit Velocity-Templates, die von korrespondierenden Screen-Klassen mit Daten versorgt werden, und Action-Klassen, die für die Interaktion mit den Manager-Klassen des SWAMP-Cores zuständig sind (siehe auch Grafik 4.3).

Die WebSWAMP-Oberfläche bietet die folgenden Möglichkeiten:

- Das Starten eines Workflows.
- Konfigurierbare Übersichtsseiten über im System bestehende Workflows und auf ihre Durchführung wartende Tasks.
- Detailansicht eines Workflows mit Möglichkeiten zur Änderung der Daten seines Datapacks, seines Status und seiner Visualisierung als gerichteter Graph.
- Durchführen eines Usertasks mit jeweils anderen Templates für die verschiedenen Action-Typen.

Turbine bietet mit Hilfe des *UI-Managers* Unterstützung für verschiedene Oberflächen, sog. *Skins*. So kann die Anwendung für Benutzer eines bestimmten Workflow-Typs ein vollkommen anderes Erscheinungsbild annehmen und zusätzliche Menüoptionen oder Ansichten auf Workflow-Listen anbieten. So wurde z. B. für die Oberfläche des Workflows „CDTracker“ eine zusätzliche Seite vor das Starten eines Workflows gesetzt, die workflow-spezifische Elemente enthält.

5.2.1 Screen-Klassen

Die Screen-Klassen dienen dem Abfragen von Daten aus dem SWAMP-Core und ihrer Aufbereitung und Übergabe an die Velocity-Templates zur Darstellung. Beispiel-Code einer Screen-Klasse:

```
public class Workflows extends SecureScreen
{
    public void doBuildTemplate(RunData data, Context context){
        WorkflowManager wfm = WorkflowManager.getInstance();
        ArrayList workflowList = wfm.getWorkflows();
        context.put("wflist", workflowList);
    }
}
```

Hier wird vom Workflow-Manager die Liste der aktiven Workflows abgerufen und mit `context.put()` im Kontext-Objekt gespeichert, aus dem sie das Velocity-Template anschließend abrufen kann. Durch das Erben der Klasse `SecureScreen` läßt sich ein Zugriffsschutz-Modell erstellen, da durch sie eine Überprüfung der Rechte des eingeloggten Benutzers durchgeführt wird. Bestehen keine ausreichenden Zugriffsrechte, wird automatisch auf die Login-Seite verwiesen.

5.2.2 Action-Klassen

Die Action-Klassen dienen der aktiven Veränderung der Daten des SWAMP-Modells. Hier finden sich Funktionen, die durch eine Benutzeraktion ausgelöst wurden, z. B. die Durchführung eines Tasks oder der Start eines Workflows, den das folgende Listing zeigt:

```
public class WorkflowActions extends SecureAction {

    public void doStartnewworkflow(RunData data, Context context) {
        [...]
        WorkflowManager wfm = WorkflowManager.getInstance();
        Workflow wf = wfm.startWorkflow(templateName);
        [...]
    }
}
```

Auch die Action-Klassen lassen sich durch Vererbung von der `SecureAction`-Klasse in ein Sicherheitskonzept einbinden, um den unbefugten Zugriff zu verhindern.

5.2.3 Screen-Templates

Das Erzeugen der Templates erfolgt mit Hilfe des unter 4.3.4 vorgestellten Tools Velocity. Die Inhalte der Seiten werden mit den von den Screen-Klassen im Kontext hinterlegten Objekten generiert. Die Erstellung der Workflow-Übersichtsseite erfolgt nach folgendem Muster:

```
[...]
#foreach ($workflow in $workflows)
    Workflow: ${workflow.getName()}
#end
[...]
```

Abbildung 5.7 zeigt die von Turbine generierte Übersichtsseite der laufenden Workflows.

5.2.4 Workflow-Filter

Um eine konfigurierbare Übersicht der laufenden Workflows zu erstellen, ist es sinnvoll, den Benutzern die Möglichkeit zu geben, die Listen der Workflows nach eigenen Kriterien zu filtern und darzustellen (siehe auch Abbildung 5.7). Da die Workflow-Liste ohne weiteres mehrere Hundert Einträge haben kann, sind personalisierte Ansichten unbedingt notwendig, um die Übersicht zu bewahren. Mögliche Kriterien, nach denen man die Workflow-Liste filtern kann, sind:

- der Inhalt eines bestimmten Databits paßt auf einen gegebenen regulären Ausdruck (Contentfilter)
- Anzeige aller Workflows eines bestimmten Typs oder eines bestimmten Status (Propertyfilter)
- Anzeige aller Workflows mit bestimmten Eigenschaften ihrer aktiven Tasks (Taskfilter)

Diese Filter können von den Benutzern selbst über die Oberfläche angelegt und gespeichert werden. Es besteht zusätzlich die Möglichkeit, Filter im Hintergrund durch das System anlegen zu lassen. So verbirgt sich hinter dem Menüpunkt „*Tasks assigned to me*“ eine Ansicht auf die Workflow-Liste, die durch einen automatisch im Hintergrund angelegten Taskfilter gefiltert wird und dem momentan eingeloggten Benutzer nur Workflows, die ihm zugeordnete Tasks besitzen, anzeigt.

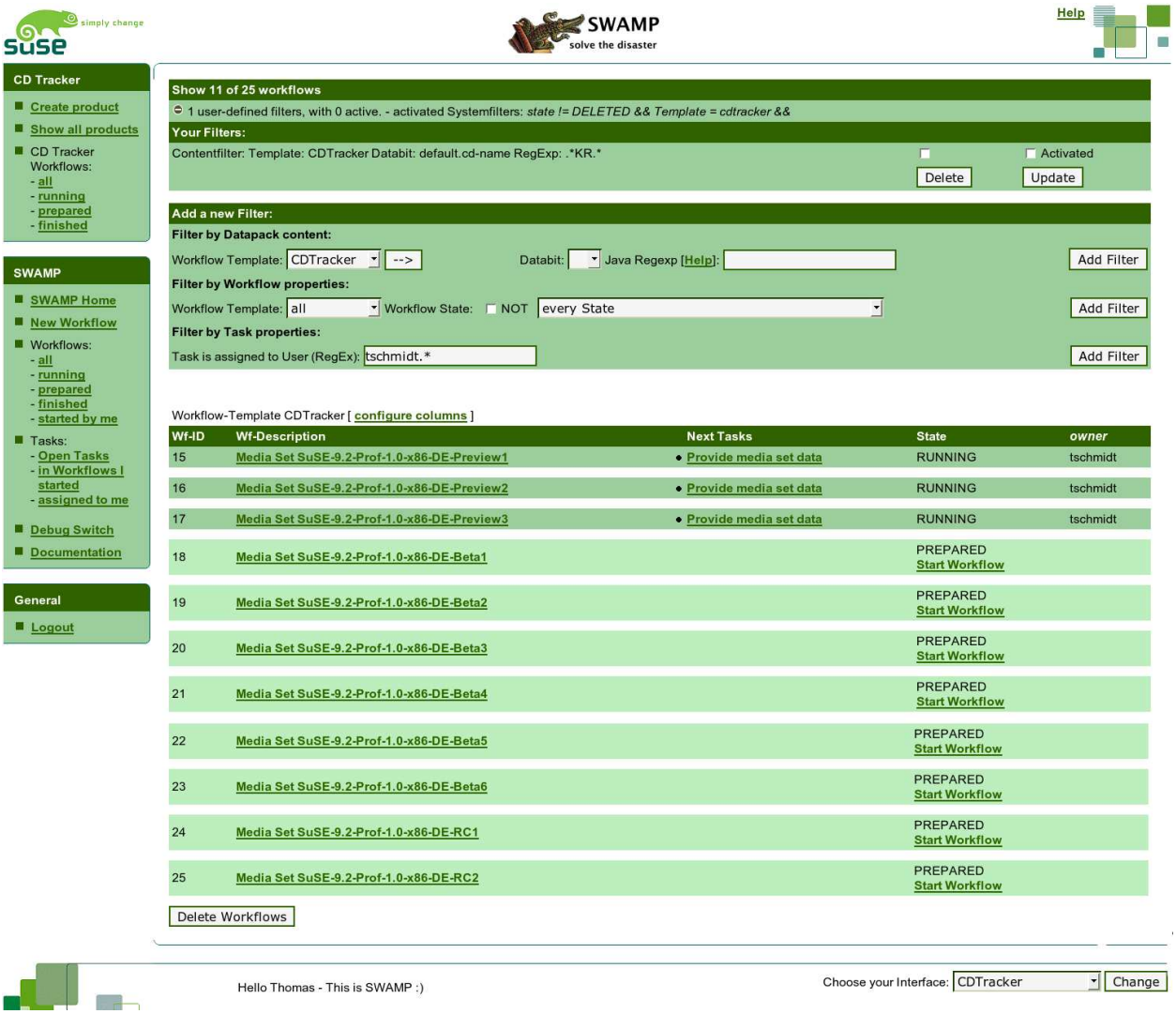


Abbildung 5.7: WebSWAMP-Workflow-Übersichtsseite

5.2.5 Visualisierung eines Workflow-Graphen

Innerhalb der Web-Oberfläche ist es möglich, eine visuelle Repräsentation des Workflows als gerichteten Graphen generieren zu lassen. Diese Aufgabe übernimmt die Klasse

`de.suse.swamp.tools.WorkflowDraw`,

die die Struktur des Workflows analysiert und eine `.dot` Datei erzeugt, die dann von dem Tool Graphviz[8] in eine PNG-Grafikdatei umgewandelt werden kann. Wie Abbildung 5.8 zeigt, können aktive Knoten farblich hervorgehoben und der Ablauf des Workflows dem Benutzer grafisch dargestellt werden. Die konkrete Darstellung erleichtert ihm so das Erfassen des Workflow-Ablaufs. Im dargestellten Beispiel sind die Knoten des Workflows mit ihrem Namen und die sie verbindenden Edges mit den Bezeichnungen der Events, auf die ihre Conditions warten, abgebildet. Die automatisierte Darstellung des Workflow-Graphen stößt allerdings bei komplexen Workflow-Konstruktionen an ihre Grenzen.

[8] http://www.research.att.com/sw/tools/graphviz/

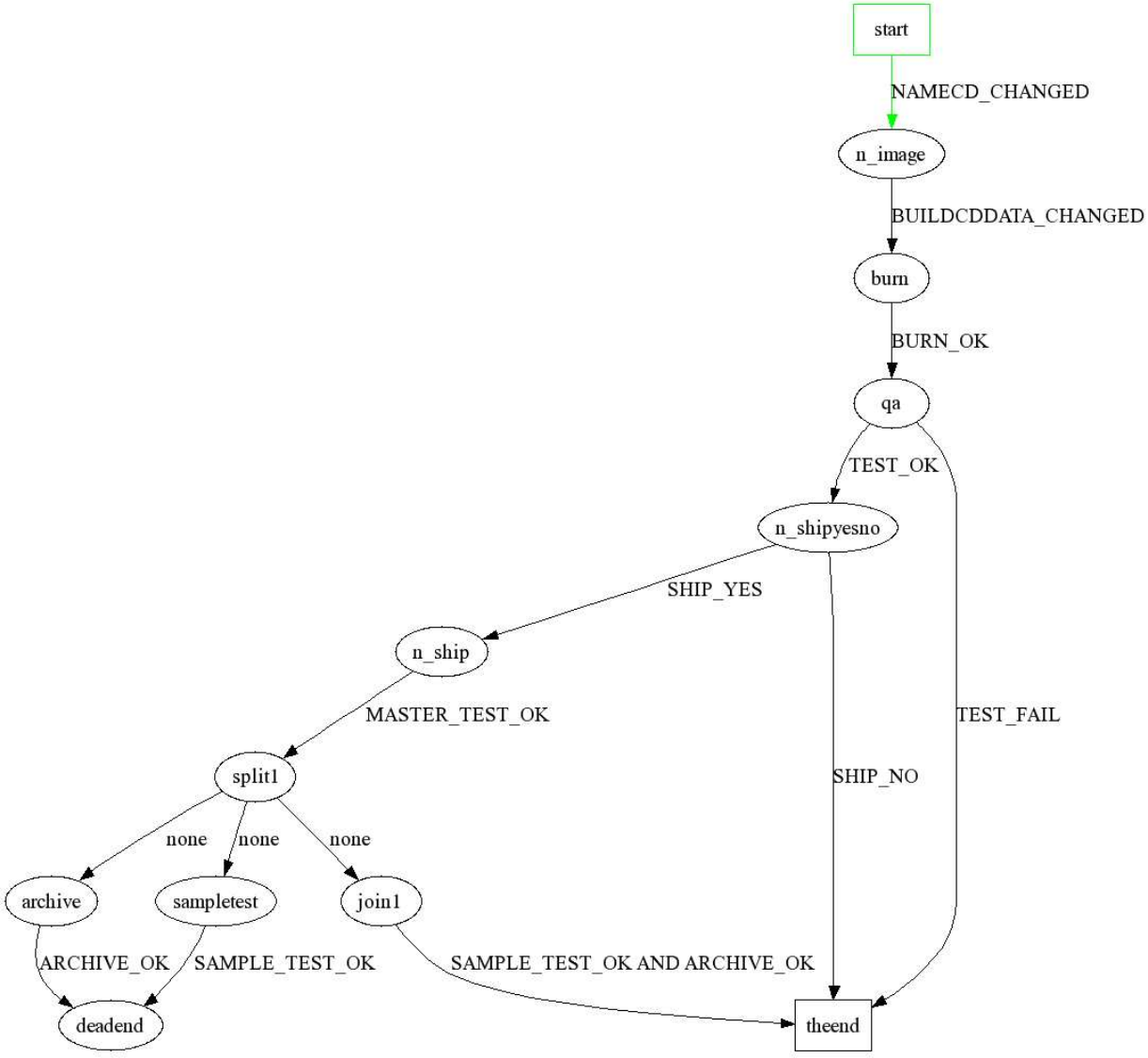


Abbildung 5.8: Visualisierung des Workflow-Graphen

6. Integration des Maintenance-Prozesses

Das Kapitel der Integration des Maintenance-Prozesses der Firma SUSE ist nicht öffentlich. :-(

7. Schlußbetrachtung

Der in dieser Arbeit implementierte Workflow zur Ablösung der Laufzettel-Mailingliste ist nur der erste Schritt einer Weiterentwicklung des Maintenance-Prozesses. In zukünftigen Versionen werden weitere Aspekte der Maintenance in die Entwicklung einfließen, um den Ablauf der Maintenance für alle Beteiligten mit Hilfe von SWAMP zu optimieren. Die Maintenance ist, wenn auch der wichtigste, so doch nicht der einzige Aspekt der momentanen und zukünftigen SWAMP-Entwicklung. Neben den bereits begonnenen Arbeiten an den Workflows *CD-Tracker* und *Level3-Support* kündigt sich an, daß innerhalb der Firma SUSE weitere Abteilungen Interesse an einer Nutzung des Workflow-Management-Systems SWAMP haben. Es wurden bereits erste Gespräche mit der Abteilung *Human Resources* geführt, die den Ablauf des Bewerbungs-Managements mit Hilfe des Systems abwickeln möchte, und mit der Abteilung *Internal-Tools* von Novell, die eine Integration des Workflows zur Koordinierung von Übersetzungsaufträgen (*Translation-Tracker*) plant.

Die Struktur von SWAMP ist so angelegt, daß sich Erweiterungen einfach bewerkstelligen lassen. Zukünftige Workflows werden, wie sich z. B. bei der Integration des Maintenance-Workflows gezeigt hat, zusätzliche Anforderungen an das System stellen, die noch nicht „*Out of the box*“, d. h. mit dem aktuellen Projektstand, abgedeckt werden können. Um diesen Herausforderungen zu begegnen, wurde bereits in der Entwurfsphase auf eine Erweiterbarkeit der einzelnen Elemente geachtet. Die derzeit implementierten Datentypen, Actions und die Erstellung der Benutzeroberfläche mit Hilfe des Turbine-Frameworks sind nicht das Ende der Entwicklungsmöglichkeiten. Wie die in Abschnitt 3.5 durchgeführte Untersuchung der Workflow-Patterns in SWAMP zeigt, können alle gebräuchlichen Patterns durch Erweiterungen im System SWAMP realisiert werden. Somit lohnt sich eine weitere Entwicklungsarbeit in das Workflow-Management-System, und der Integration weiterer Workflows kann optimistisch entgegengesehen werden.

Weitere Perspektiven für SWAMP ergeben sich durch die Freigabe als *Open Source* Projekt im Internet. Wenn es gelingt, weitere Anwender und Entwickler für das Programm zu begeistern, dann können die Weiterentwicklungen externer Benutzer in das Projekt miteinfließen und letztendlich allen Benutzern zugute kommen. Dieser Prozess verlagert das Zentrum des Projekts von der Firma SUSE hin zu einer öffentlichen SWAMP-Gemeinde, die das Projekt pflegt.

Abbildungsverzeichnis

Hinweis: Alle Grafiken dieser Arbeit wurden vom Autor mit Hilfe der Programme Dia (Version 0.92, http://www.gnome.org/projects/dia), KSnapshot (Version 0.7, http://www.kde.org), Gimp (Version 2.0.0, http://www.gimp.org), und DBDesigner 4 (http://www.fabforce.net/dbdesigner4) erstellt.

Literatur

[Balz01] Heide Balzert. *UML Kompakt*. Spektrum Akademischer Verlag. 2001.

[Böh02] Oliver Böhm. *Java Software Engineering*. SuSE Press. 2002.

[Coal00] Workflow Management Coalition. *Terminology & Glosssary*. Workflow Management Coalition. 2000.

[Corn03] Cay S. Horstmann & Gary Cornell. *Core Java2*. Sun Microsystems Press. 2003.

[Darw03] Jason Brittain & Ian F. Darwin. *Tomcat - The Definitive Guide*. O'Reilly. 2003.

[Jans04] Andreas Holubek & Rudolf Jansen. *Java Persistenz-Strategien*. Software und Support Verlag. 2004.

[J.Ro99] S. Robertson & J.Robertson. *Mastering the Requirements Process*. Addison-Wesley. 1999.

[Kiep02] Bartek Kiepuszewski. *Expressiveness and suitability of languages for control flow modelling in workflows*. Queensland University of Technology. 2002.

[Koch96] Olaf Koch. *Workflow Management*. Markt & Technik. 1996.

[Kull04] Daniel Kulla. *Der Phrasenprüfer*. Der grüne Zweig. 2004.

[Schm01] Volker Turau & Krister Saleck & Marc Schmidt. *Java Server Pages und J2EE*. dpunkt. Verlag. 2001.

[Simm04] Robert Simmons. *Hardcore Java*. O'Reilly. 2004.

[Thei02a] Fabian J. Theis. *Portale und Webapplikationen mit Apache Frameworks*. Software und Support Verlag. 2002.

[Thei02b] Fabian J. Theis. Power aus der Turbine. *Java Magazin* (06), 2002, S. 73–77.

[vdAA00] W. van der Aalst & A. ter Hofstede & B. Kiepuszewski. *Workflow Patterns*. Eindhoven University of Technology. 2000.

[Webe98] H. Weber. *Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Management-Systemen*. Physica Verlag. 1998.

[Zül96] D. Riehle & H. Züllighoven. *Understanding and using Patterns in Software Development*. Eindhoven University of Technology. 1996.

Index