Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Entwurfsvergleich einer SOA-Anwendung auf SOA-Me- und BPEL- Basis

Bachelorarbeit

im Studiengang Informatik

von

Christoph König

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Udo Lipeck
Betreuer: Dipl.-Wirt.-Inform. Daniel Lübke

Hannover, 15. August 2007

Zusammenfassung

Geschäftsprozesse werden in der heutigen Zeit häufig nach dem Konzept der serviceorientierten Architektur als Software realisiert. Dies geschieht meistens mit Hilfe von Webservices und der Business Process Execution Language (BPEL). Wenngleich sich diese Sprache sehr für automatisch ablaufende Geschäftsprozesse eignet, unterstützt sie keine Benutzerinteraktionen, obwohl viele reale Geschäftsprozesse menschliches Eingreifen erfordern.

Aus diesem Grund wurde die SOA-Me-Plattform entwickelt, die, im Gegensatz zu BPEL, Benutzerinteraktionen gezielt unterstützt.

Diese Arbeit vergleicht die beiden Systeme anhand eines konkreten Geschäftsprozesses mit Hilfe von speziell dafür ausgewählten Metriken und zieht aufgrund dieses Vergleiches Rückschlüsse darauf, welches der beiden Systeme für diese Anwendung besser geeignet ist.

Abstract

The service orientated architecture paradigm (SOA) is more and more important for the realization of business processes today. Mostly SOA is implemented by web services, composed by the Business Process Execution Language (BPEL). While this language is high suitable for completely automated business processes, it does not support user interactions. Although most real-live business processes require human intervention.

For this reason the SOA-Me platform was developed. Contrary to BPEL, this system supports user interactions by design.

This thesis compares the two systems, implementing a concrete business process, with the help of well chosen metrics. Based on the results of this comparison, it will show which of the both systems suits better for the given process.

Danksagung

Für die vorbildliche und
hervorragende Betreuung dieser Arbeit
möchte ich Herrn Daniel Lübke an dieser
Stelle herzlich danken.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Service-orientierte Architekturen sind in der heutigen Zeit ein häufig genutztes Konzept um Geschäftsprozesse zu automatisieren. Bei diesem Vorhaben wird meist auf Webservices gesetzt, die einen kleinen Teil der Funktionalität des Prozesses kapseln und diese über ein Netzwerk zur Ausführung anbieten.

Um größere Geschäftsprozesse unterstützen zu können müssen diese Webservices zu größeren Kompositionen zusammengefasst werden. Um dieses zu ermöglichen existieren verschiedene Kompositionssprachen die komplexe Arbeitsabläufe (Workflows) unterstützen.

Eine davon ist die 2002 von einigen großen Firmen vorgestellte Business Process Execution Language (BPEL). Sie hat gute Chancen die Standardsprache für die Komposition von Webservices zu werden. Dabei liegt der Schwerpunkt dieser Sprache auf komplett automatisch ablaufenden Geschäftsprozessen und nicht auf Prozessen, die das Eingreifen des Menschen erfordern. Diese kommen aber in vielen realen Geschäftsprozessen vor.

Aus diesem Grund wurde 2006 am Fachgebiet Software Engineering der Universität Hannover die SOA-Me-Plattform entwickelt. Dieses System wurde mit dem Ziel geschaffen, Benutzerinteraktionen und Webservicekompositionen gleichermaßen zu unterstützen. Dabei setzt es vor allem auf zur Laufzeit generierte Benutzeroberflächen, um die Realisierung von Benutzerinteraktionen so einfach wie möglich zu machen. SOA-Me verwendet zur Prozesskomposition ereignisgesteuerte Prozessketten (EPKs), da davon ausgegangen wird, dass diese einfacher zu Beherrschen sind als die BPEL-Notation.

1.2 Ziel dieser Arbeit

Ziel dieser Arbeit ist es herauszufinden, ob sich BPEL oder SOA-Me eher für einen bestimmten Geschäftsprozess eignet. Bei diesem Geschäftsprozess handelt es sich um ein Verwaltungssystem für studentische Abschlussarbeiten am Fachgebiet Software Engineering.

Dieser Prozess wurde im Wintersemester 2006/2007 im Rahmen der Projektarbeit "Entwicklung einer Webservice-basierten Anwendung" bereits mit Hilfe von BPEL realisiert. Im Rahmen dieser Arbeit soll er auf die SOA-Me-Plattform portiert werden. Im Zuge dieser Portierung soll auch die Architektur der Realisierung auf BPEL-Basis analysiert und bewertet werden.

Nach der Portierung sollen die beiden Anwendungen verglichen werden. Zu diesem Zweck müssen geeignete Metriken erhoben, angewendet und die Ergebnisse interpretiert werden. Dabei interessiert besonders welcher der beiden Ansätze aufwendiger zu realisieren und zu warten ist. Des Weiteren stellt sich die Frage, ob die automatisch generierten Benutzeroberflächen von SOA-Me gegenüber der Benutzeroberfläche der BPEL-Anwendung eine ähnliche Usability aufweisen und ob EPKs wirklich intuitiver zu verstehen sind als die BPEL-Notation.

1.3 Struktur dieser Arbeit

Zunächst sollen in dieser Arbeit einige Grundlagen vermittelt werden. Dabei wird kurz das Konzept der serviceorientierten Architektur erläutert. Anschließend wird ein Überblick über die beiden SOA-Plattformen BPEL und SOA-Me gegeben.

Im darauf folgenden Kapitel werden die Anforderungen an die Anwendung vorgestellt, die für beide Plattformen implementiert wurden. Außerdem werden die beiden verschiedenen Entwürfe mit ihren unterschiedlichen aber auch gemeinsamen Entwurfentscheidungen erläutert.

Im vierten Kapitel dieser Arbeit wird die Portierung der SOA-Anwendung von BPEL auf die SOA-Me-Plattform vorgestellt und die Rolle und Einbindung der verschiedenen Frameworks erläutert. Auch werden die bei der Implementierung aufgetretenen Probleme thematisiert.

Kapitel fünf beschäftigt sich mit dem Vergleich der beiden SOA-Anwendungen. Es gibt einen Überblick über die Herangehensweise und angewandten Methoden des Vergleichs und stellt schließlich die Stärken und Schwächen beider Lösungen dar.

Im letzten Kapitel wird ein Fazit gezogen und die Bedeutung dieser Arbeit für folgende Arbeiten herausgestellt. Besonders wird hier auf die Konsequenzen für die SOA-Me-Plattform eingegangen, die sich aus dem Vergleich mit BPEL herausgestellt haben. Dabei werden einige Empfehlungen für die weitere Entwicklung der SOA-Me-Plattform gegeben.

2 Grundlagen

2.1 Serviceorientierte Architektur

Die Serviceorientierte Architektur (SOA) ist ein Konzept, das besonders zur Unterstützung von Geschäftsprozessen geeignet ist. Im Gegensatz zu traditionellen Software-Designs versucht SOA prozessorientierte Softwareanwendung besser zu unterstützen, indem einzelne Softwarekomponenten (Services) an dem Geschäftsprozess ausgerichtet werden. Diese Services decken meist eine Aktivität im Geschäftsprozess ab (z.B. „Warenbestellung abschicken“) und sollen weitgehend unabhängig voneinander sein, so dass bei sich ändernden Geschäftsprozessen eine möglichst hohe Flexibilität der Infrastruktur gewährleistet ist. Häufig sind Services nicht nur lokal sondern auch über Netzwerke erreichbar.

Die heute meist anzutreffende Art Services zu implementieren ist die Implementierung durch Webservices nach den Standards SOAP (Protokoll für das Aufrufen entfernter Prozeduren), WSDL (Webservice Description Language; Schnittstellenbeschreibung von Webservices) und UDDI (Universal Description, Discovery and Integration; Webservice-Verzeichnisdienst), auch wenn prinzipiell andere Technologien wie z.B. die „Common Object Request Broker Architecture“ (CORBA) benutzt werden können.

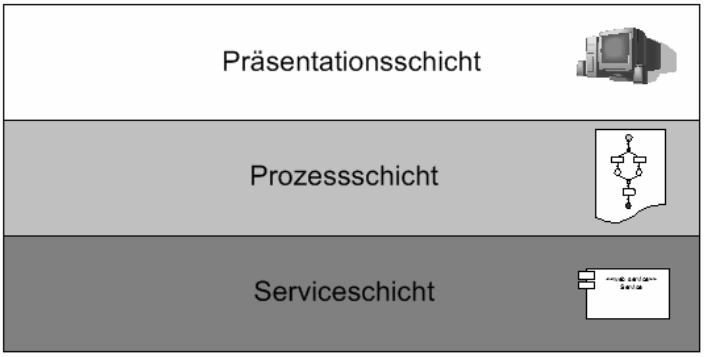

Abbildung 1: Schichten einer SOA-Anwendung nach [Lue05]

SOA-Anwendungen können (nach [Lüe05]) in drei Schichten unterteilt werden (siehe Abbildung 1): die Serviceschicht, die Prozessschicht und die Präsentationsschicht. Die Service Schicht bildet die Grundlage der SOA-Anwendung. Auf dieser Schicht werden alle Services z.B. als Webservices realisiert. Darauf baut die Prozess-Schicht auf, die die Aktivitäten des Geschäftsprozesses auf die Services der Service-Schicht abbildet und mit der Präsentations-Schicht kommuniziert. Diese beinhaltet die Benutzerschnittstelle zur Steuerung und Interaktion mit dem Prozess.

SOA-Anwendungen müssen nicht notwendigerweise alle diese Schichten umfassen. Vollautomatische Prozesse können zum Beispiel auf eine Präsentationsschicht verzichten, einfache benutzerorientierte Prozesse im Einzelfall auf die Serviceschicht.

2.2 Business Process Execution Language

Die Business Process Execution Language (BPEL) ist eine auf XML basierende Sprache, die Geschäftsprozesse beschreibt, deren einzelne Aktivitäten durch Webservices implementiert werden. Sie wurde im Jahr 2002 von IBM, BEA und Microsoft eingeführt, die Version 2.0 wurde von OASIS standardisiert.

BPEL wurde dazu entwickelt Webservices zu orchestrisieren, das heißt mehrere Webservices zu einem Webservice zusammenzustellen. Diese Orchestrisierung kann wieder als Webservice angesprochen werden, damit ist BPEL hauptsächlich auf der Service-Ebene des Drei-Schichten-Modells aus 2.1 angesiedelt. Es ist aber auch möglich einen kompletten Geschäftsprozess in BPEL zu beschreiben, falls dieser vollständig automatisiert ablaufen kann. Somit deckt BPEL die Prozess- und Serviceschicht ab.

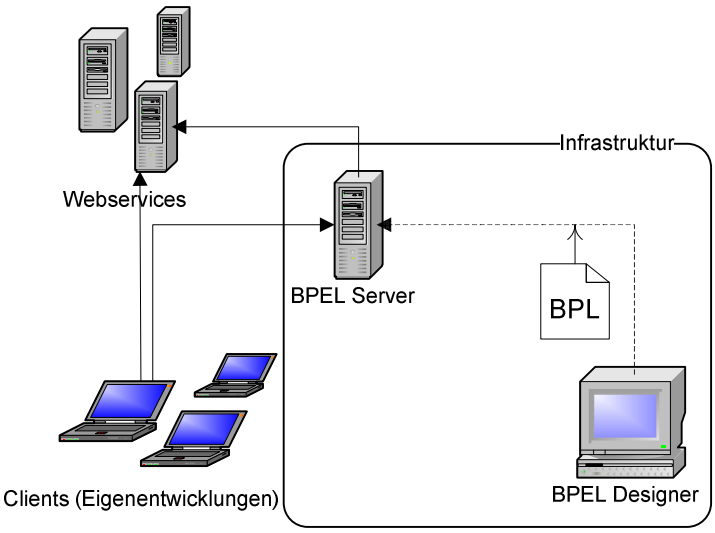

Abbildung 2: Übersicht über eine BPEL-Anwendung

Eine BPEL-Anwendung (siehe Abbildung 2), wie z.B. die freie Implementierung ActiveBPEL, funktioniert folgendermaßen: Der BPEL-Prozess wird auf dem BPEL-Designer modelliert. Dabei werden die benötigten Datenstrukturen mit XML Schema und die benötigten Webservices mit ihren WSDLs beschrieben. Der Prozess wird dann in Dateiform auf den BPEL-Server geladen (Deployment). Dieser ist fähig einzelne Instanzen des Prozesses auszuführen und dazu Webservices aufzurufen. Da sich ein BPEL-Prozess nach außen wie ein Webservice verhält, wird er gestartet, indem er von außen (z.B. durch einen Client, anderen Webservice oder einem beliebigen anderen Programm, dass Webservices nutzen kann) aufgerufen wird.

2.2.1 Mächtigkeit

Nach [ACD+03] setzt sich ein BPEL-Prozess aus so genannten Aktivitäten zusammen. Dabei werden einfache und strukturierende Aktivitäten unterschieden.

Einfache Aktivitäten (Basic Activities) sind grundlegende Aktivitäten, die nicht aus anderen Aktivitäten aufgebaut sind. Die wichtigsten einfachen Aktivitäten sind:

assign: Verändert den Inhalt einer Variablen.

invoke: Aufruf eines Webservices. Dieser kann asynchron (ohne warten auf eine Antwort) oder synchron (mit warten auf eine Antwort) erfolgen.

receive: Öffnet einen Webservice-Port und blockiert den Prozess, bis über diesen Port ein Aufruf erfolgt ist.

reply: Sendet eine Antwort auf einen per receive erhaltenden Aufruf.

throw: Signalisiert einen Fehler (ähnlich wie bei Java), der, falls er nicht durch die Fehlerbehandlung abgefangen werden kann, zur Terminierung des Prozesses führt.

wait: Verzögert den Prozess für eine gewisse Zeitspanne oder bis zu einem bestimmten Zeitpunkt.

empty: Leere Aktivität, z.B. zum Unterdrücken von Fehlern verwendbar.

terminate: Beendet den Prozess.

Strukturierende Aktivitäten enthalten andere (sowohl einfache als auch strukturierende) Aktivitäten. Mit Ihrer Hilfe können komplexe Prozesse aufgebaut werden. Zu den strukturierenden Aktivitäten gehören:

sequence: Enthaltende Aktivitäten werden sequentiell abgearbeitet.

while: Die enthaltene Aktivität wird solange ausgeführt, wie eine zugeordnete Bedingung erfüllt ist.

pick: Wartet auf ein bestimmtes Ereignis (Nachricht, Timeout), je nach Ereignis wird eine andere Aktivität ausgeführt.

switch: Führt, abhängig von einer zugeordneten Bedingung, eine Aktivität aus.

flow: Führt mehrere Aktivitäten gleichzeitig (oder in beliebiger Reihenfolge) aus. Synchronisation der Aktivitäten ist durch so genannte Links möglich.

scopes: Scopes erlauben es, eine neue Umgebung für die enthaltene Aktivität zu öffnen. Innerhalb eines Scopes angelegte Variablen gelten dann nur für diesen Scope und wirken sich nicht global aus. Auch kann jeder Scope über eine eigene Fehlerbehandlung (Fault-Handler genannt) und einen eigenen Compensation-Handler verfügen. Letzterer dient dazu die von Aktivitäten innerhalb des Scopes gemachten Änderungen rückgängig zu machen. Dadurch können mit Hilfe von Scopes Transaktionen realisiert werden.

2.2.2 Schwächen von BPEL

BPEL sieht keine Benutzerinteraktionen und damit, wie schon auf Abbildung 2 zu sehen, keinen Client vor. Es ist nicht direkt möglich innerhalb des Prozesses Aufgaben durch Benutzer sondern nur durch Webservices durchführen zu lassen. Zwar kann für eine Benutzerinteraktion die Webservice-Schnittstelle des Prozesses verwandt werden, dabei ruft der Client aber immer den Prozess auf, der Prozess kann nicht auf eigene Initiative mit dem Client kommunizieren. Dies ist nur indirekt über einen Webservice der zwischen Prozess und Client vermittelt oder durch Polling durch den Client möglich.

Ein Versuch diese Schwäche von BPEL zu beheben ist das von IBM in Zusammenarbeit mit SAP entwickelte BPEL4People [KKL+05], welches BPEL um so genannte „people activities“ erweitert. Diese regeln, was ein Benutzer tun muss (Aufgabenbeschreibung), welche Gruppe von Benutzern diese Aufgabe erledigen darf und welche Person aus dieser Gruppe zur Laufzeit letztendlich die Aufgabe erledigt (hat). BPEL4People lässt dabei aber offen, über welche Protokolle dies geschehen soll und wie letztendlich die Aufgaben dargestellt werden. Dazu müssen Individual-Lösungen entwickelt werden.

2.3 SOA-Me

Die SOA-Me Plattform ist ein Prototyp einer Entwicklungs- und Ausführungsumgebung für auf Webservices basierende Geschäftsprozesse. Sie wurde im Wintersemester 2006/2007 im Rahmen des Software Projekts am Fachgebiet Software Engineering der Universität Hannover mit dem Ziel entwickelt sowohl automatisierbare als auch nichtautomatisierbare Geschäftsprozesse zu unterstützen.

Dabei wurde sich konzeptionell an den Schwächen von BPEL orientiert (siehe 2.2.2), so unterstützt die SOA-Me Plattform nicht nur explizit von Benutzern zu bearbeitende Aktivitäten sondern stellt auch einen einheitlichen Client zur Verfügung, der dynamisch zur Laufzeit zu dem Geschäftsprozess passende Benutzeroberflächen generieren kann. Allerdings existiert keine Webserviceschnittstelle für die einzelnen Prozesse mehr. SOA-Me Prozesse können also, im Gegensatz zu BPEL, nicht als Webservices angesprochen werden. Damit deckt SOA-Me die oberen beiden Schichten, die Prozess- und die Darstellungsschicht, des Schichtmodells aus 2.1 ab.

Die Plattform basiert auf ereignisgesteuerten Prozessketten (EPKs) die auf der Grundlage von [Lue05] und [Int06] um Benutzerinteraktionen und Webservice-Aufrufe erweitert wurden. Sie (genauer der SOA-Me Integrationsserver) ist in der Lage beliebige wohlstrukturierte EPKs auszuführen, welche nach [GL05] alle in der Praxis relevanten EPKs abdecken.

Die zur Laufzeit benötigten Daten, die zum Aufrufen eines Webservices oder einer Benutzerinteraktion nötig sind, werden mit Hilfe von Extensible Stylesheet Language Transformation (XSLT) aus dem so genannten Prozessdokument gewonnen. Dieses Dokument ist eine einfache XML-Datei deren Struktur zum Großteil vom Entwickler des Prozesses vorgegeben werden kann. Diese Datei wächst mit der Ausführungszeit des Prozesses, da die Ergebnisse von Webservice-Aufrufen und Benutzerinteraktionen nach einer weiteren XSL-Transformation an das Prozessdokument angehängt werden.

2.3.1 Aufbau

Die SOA-Me-Plattform (Abbildung 3) besteht aus drei Komponenten: Dem SOA-Me Integrationserver, der die Geschäftsprozesse ausführt, dem SOA-Me Kompositionseditor, mit dessen Hilfe Geschäftsprozesse modelliert werden können, und dem SOA-Me Client, für die Präsentation des Geschäftsprozesses.

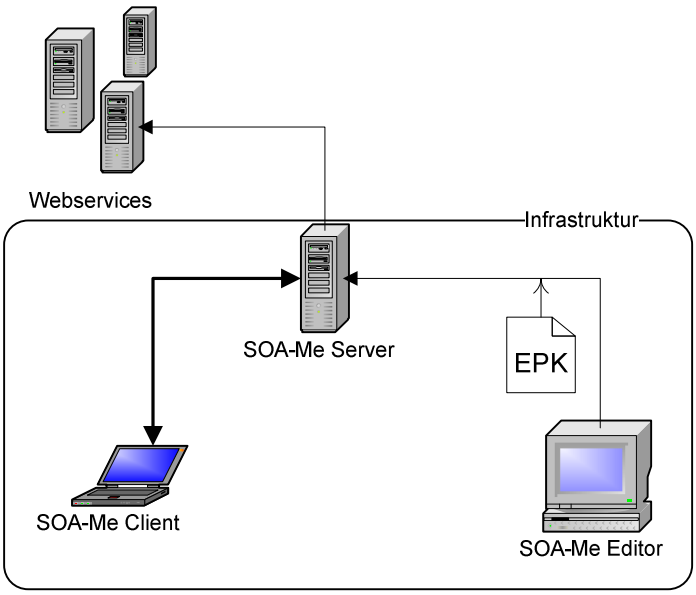

Abbildung 3: Übersicht über die SOA-Me Plattform

Die Plattform funktioniert wie folgt: Der auf dem SOA-Me Editor entwickelte Prozess in Form einer ereignisgesteuerten Prozesskette wird zusammen mit den WSDL-Dateien der benötigten Webservices sowie der XSLT-Dateien zur Transformation der Prozessdaten entweder in Dateiform oder über eine Netzwerkverbindung auf den SOA-Me Server geladen (deployed).

Dieser hält alle ihm bekannten Prozesse in einer Datenbank und bietet dazu berechtigten Clients den Start dieser Prozesse an. Wird eine Prozessinstanz gestartet, wird diese vom Server entsprechend der EPK ausgeführt, wobei Webservices aufgerufen und im Rahmen von Benutzerinteraktionen Clients über anstehende Aufgaben benachrichtigt werden. Für letzteres werden die zur Ausführung der Aufgabe berechtigten Personen ermittelt. Sofern sie zurzeit an dem Server angemeldet sind, werden den Clients, die diese Personen gerade benutzen, Informationen über diese Aufgabe gesendet.

Der SOA-Me Client fasst diese Informationen in einer ToDo-Liste für den Benutzer zusammen. Dieser kann nun eine Aufgabe aus dieser Liste auswählen, woraufhin der Client aufgrund der Informationen über die Aufgabe eine zur Aufgabe passende graphische Benutzeroberfläche generiert. Diese kann aus Eingabefeldern (Edit), Auswahllisten (Choice, nur ein Element selektierbar), Auswahlbuttons (Select, beliebig viele Elemente selektierbar) und Textinformationen (Display) bestehen (siehe auch 2.3.2.2). Einen Überblick über dieses Konzept verschafft [LLSG06]. Die Informationen zur Generierung der Oberfläche werden bereits im SOA-Me Editor erstellt. Hat der Benutzer die Aufgabe bearbeitet (indem er z.B. eine Eingabe getätigt, oder einfach nur den angezeigten Text gelesen hat), werden die Ergebnisse an den Server geschickt, der den Prozess darauf fortsetzt.

Der SOA-Me Client besitzt ferner noch ein Experience Forum. Mit diesem Feature ist es möglich die Erfahrungen der Benutzer zu einer Aufgabe zu sammeln und später durch das Entwicklungsteam auszuwerten. Dieses Konzept wurde in [LS06] vorgestellt und in [Gri07] in die SOA-Me Plattform integriert.

2.3.2 Mächtigkeit

Die Mächtigkeit der SOA-Me Plattform wird von zwei Teil-Modellen bestimmt. Einmal dem Modell der ereignisgesteuerten Prozesskette, welches zur Komposition der Prozesse dient, und einmal dem Task-Modell, welches zum Beschreiben der Aufgaben, die vom Benutzer ausgeführt werden, dient.

2.3.2.1 EPK

EPKs sind die Grundlage der SOA-Me Prozesse und bestehen im Wesentlichen aus drei Klassen von Aktivitäten: Ereignisse, Funktionen und Konnektoren.

Ereignisse stellen eingetretene Zustandsänderung dar. Sie können an Bedingungen geknüpft sein, so dass sie im Zusammenhang mit entsprechenden Konnektoren den Prozessablauf beeinflussen können. Ansonsten haben sie nur strukturierende Funktion (passives Element). Eine EPK beginnt immer mit einem Ereignis, dann folgen alternierend Funktionen und weitere Ereignisse.

Funktionen stellen die eigentlichen Aktivitäten der Prozesse da (aktives Element). Sie werden durch Ereignisse ausgelöst und resultieren wieder in einem Ereignis. Die SOA-Me Plattform sieht zwei Arten von Funktionen vor: Webservice-Aufrufe und Benutzer-Interaktionen.

Konnektoren können die EPK verzweigen (Split) und wieder zusammenführen (Join). Sie werden entweder zwischen Funktion und Ereignis oder zwischen Ereignis und Funktion eingesetzt. Es gibt drei Arten von Konnektoren: OR-, XOR- und AND- Konnektoren.

OR-Konnektoren beschreiben Alternativen. Auf eine OR-Verzweigung müssen bedingte Ereignisse folgen. Alle Ereignisse, deren Bedingung wahr ist, werden gleichzeitig aktiviert. Bei der Zusammenführung durch ein OR wird solange gewartet, bis alle eingehenden Zweige Ihre Aktivität beendet haben.

XOR-Konnektoren verhalten sich wie OR-Konnektoren, nur dass genau ein Ereignis aktiviert wird. Aus diesem Grund braucht bei der Zusammenführung durch ein XOR auch nicht gewartet zu werden.

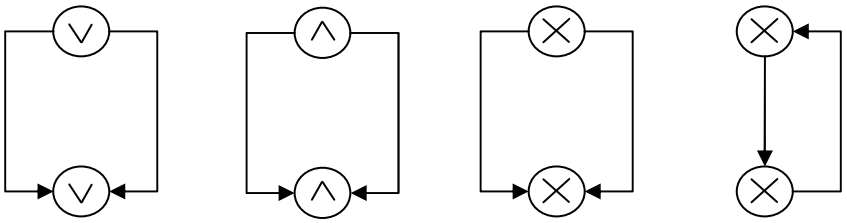

Abbildung 4: Typische Konstrukte mit Konnektoren

AND-Konnektoren können nur nach Ereignissen eingesetzt werden. Sie führen zu einer unbedingten gleichzeitigen Aktivierung aller nachfolgenden Funktionen. AND-Zusammenführungen warten ebenfalls wie OR-Zusammenführungen, bis alle aktiven Zweige ihre Arbeit beendet haben.

2.3.2.2 Task-Modell

Aufgaben der UI-Funktionen werden mit dem Task-Modell beschrieben. Dieses sieht eine Sequenz von UI-Elementen vor, die dem Benutzer angezeigt werden. Jedem dieser Elemente wird ein so genanntes „Information Object“ zugeordnet, welches die darzustellenden oder zu modifizierenden Daten enthält.

Folgende UI-Elemente werden unterstützt:

Display: Das Display Element zeigt die in dem Information Object enthaltenden Daten an. Die Daten können in HTML (Hypertext Markup Language) formatiert sein.

Selection: Dieses Element lässt den Benutzer Daten auswählen. Bei diesen Daten kann es sich um Daten aus einem Information Object oder aber auch um ganze „Information Objects“ handeln. Es ist möglich die Auswahl von mehreren Daten aber auch die Abwahl aller Daten zuzulassen.

Edit: Mit Hilfe dieses Elements kann der Benutzer die Daten des zugehörigen Information Objects verändern.

Control: Ein Control fordert den User zu einer Entscheidung auf. Diesem Element ist kein Information Object zugeordnet, sondern zwei oder mehr Aktionen (Actions) aus denen der User eine auswählen muss.

2.3.2.3 Benutzeroberfläche

Das Task-Modell des SOA-Me-Systems sieht einen in vier Bereiche eingeteilten Client vor (siehe Abbildung 5). In der Prozessliste (A) werden dem Benutzer Prozesse angeboten, die dieser starten darf. Die ToDo-Liste (B) enthält alle Aufgaben aus laufenden Prozessen, die der Benutzer bearbeiten muss bzw. darf. In der Aufgabenanzeige (C) können die zuvor ausgewählten Aufgaben bearbeitet werden. Dieser Bereich wird während des Prozessablaufs automatisch aus den zu den Benutzerinteraktionen gehörigen Elementen generiert. Im Experience Forum (D) kann der Benutzer Kommentare zu den einzelnen Aufgaben hinterlassen und sich die Erfahrungen anderer Benutzer ansehen.

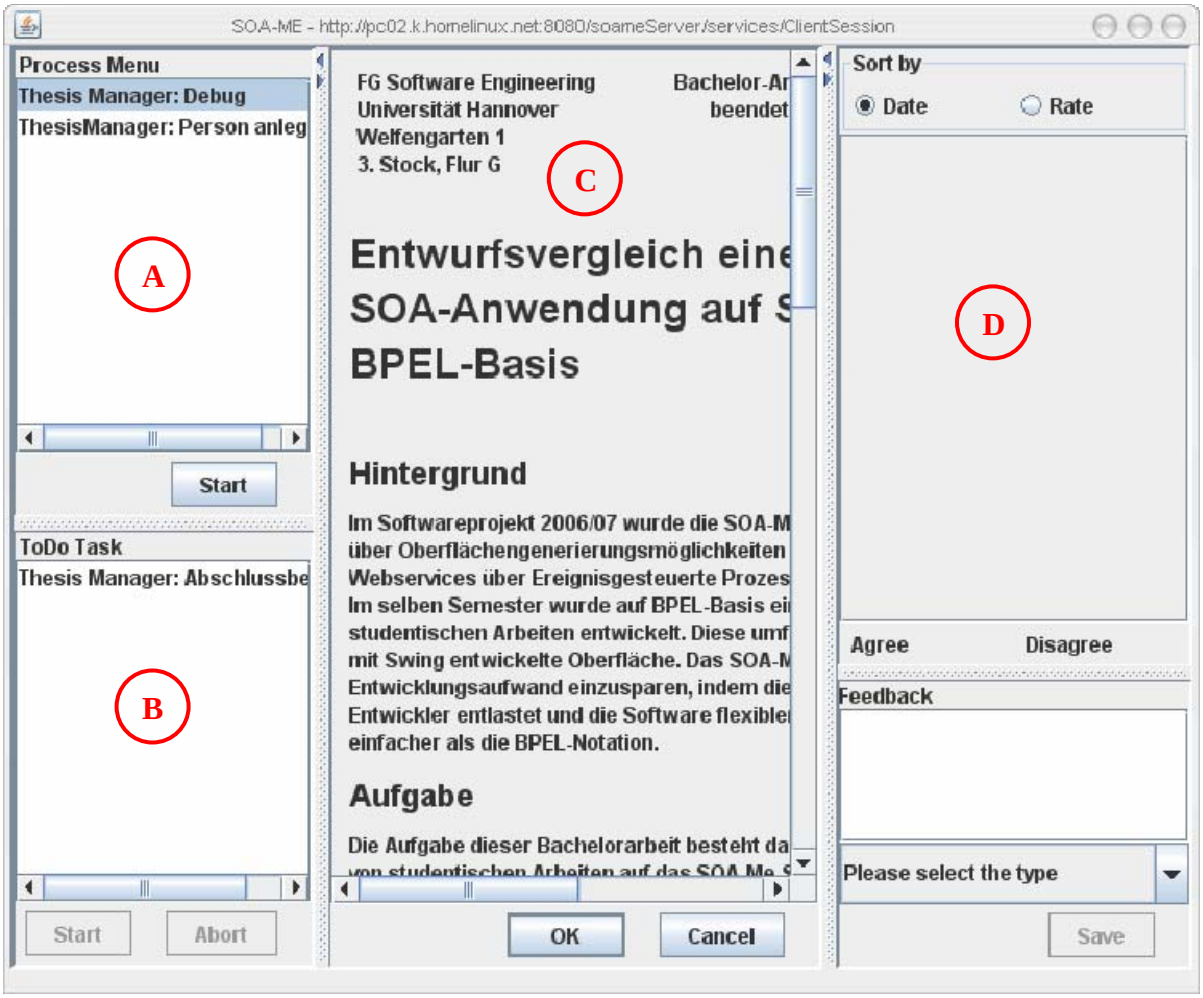

Abbildung 5: Benutzeroberfläche des SOA-Me Client

2.4 Gegenüberstellung

In [Sch07] wurde auf Basis von [ADH03] die Mächtigkeiten von EPKs und BPEL verglichen. Dabei ging es darum, welche Muster von Arbeitsabläufen (Workflow Patterns) die beiden Kompositionssprachen beherrschen. Wie in Tabelle 1 zu sehen deckt dabei BPEL sechs Muster mehr ab als EPK. Dabei ist BPEL aber keine Obermenge von EPK denn diese unterstützen ineinander verschränkte Schleifen, welche mit dem blockorientierten BPEL nicht möglich sind.

Ansonsten gibt es noch System-inhärente Unterschiede zwischen BPEL und SOA-Me. Der fehlende generische Client und Unterstützung von Benutzerinteraktionen seitens BPEL wurde davon schon ausreichend behandelt. Da SOA-Me ein Prototyp ist, fehlen diesem System einige Standardfunktionen für Workflow-Engines wie Persistenz und Transaktionen. Diese Funktionen müssen aufwendig und explizit in den zu realisierenden Prozessen eingebaut werden.

Basic Control Patterns	**EPK**	**BPEL**
WP1 - Sequence	+	+
WP2 - Parallel Split	+	+
WP3 - Synchronization	+	+
WP4 - Exclusive Choice	+	+
WP5 - Simple Merge	+	+
Advanced Branching and Sync. Patterns		
WP6 - Multi Choice	(+)	+
WP7 - Synchronizing Merge	(+)	+
WP8 - Multi-Merge	-	-
WP9 - Discriminator	-	-
Structural Patterns		
WP10 - Arbitrary Cycles	+	-
WP11 - Implicit Termination	+	+
Involving Multiple Instances (MI)		
WP12 - MI Without Synchronisation	-	+
WP13 - MI With a Priori Design Time	-	+
WP14 - MI With a Priori Runtime	-	-
WP15 - MI Without a Priori Runtime	-	-
State-based Patterns		
WP16 - Deferred Choice	-	+
WP17 - Interleaved Parallel Routing	-	(+)
WP18 - Milestone	-	-
Cancellation Patterns		
WP19 - Cancel Activity	-	+
WP20 - Cancel Case	-	+

Tabelle 1: Workflow Patterns in untersuchten Sprachen, nach [ADH03]

3 Anforderungen und Entwurf

In diesem Kapitel sollen die Anforderungen an das Verwaltungssystem für Abschlussarbeiten sowie der Entwurf auf BPEL und SOA-Me erläutert werden.

3.1 Gemeinsame Anforderungen und Entwurfsentscheidungen

Da die BPEL-Anwendung schon existierte und auf SOA-Me portiert werden musste, konnte vieles aus dem BPEL-Entwurf in den SOA-Me-Entwurf übernommen werden. Dieses Kapitel behandelt neben den gegebenen Anforderungen diese gemeinsamen Entwurfsentscheidungen.

3.1.1 Vision

Es soll ein Computersystem entwickelt werden, dass die Abschlussarbeiten am Fachgebiet Software Engineering verwaltet. Es soll dabei sicherstellen, dass alle nötigen verwaltungstechnischen Formalitäten (z.B. Anmeldung beim Prüfungsamt, Erfassen der Abgabe) durchgeführt werden und dass die wesentlichen Teile der Betreuung (z.B. das Planen von Vorträgen) eingehalten werden. Außerdem soll es den Prozess der Bearbeitung von Abschlussarbeiten, soweit möglich, automatisieren indem es beispielsweise einen geplanten Vortrag per E-Mail ankündigt.

Um das System flexibel und erweiterbar zu halten und auch eventuell anderen Systemen Zugriff auf einzelne Funktionalitäten zu geben, soll es auf Basis von Webservices entwickelt werden.

3.1.2 Stakeholder

Stakeholder eines Projektes sind, nach ISO 10006, alle Personen, die ein Interesse am Projekt haben oder vom Projekt in irgendeiner Weise betroffen sind.

Folgende Personen sind an dem Prozess zur Verwaltung von Abschlussarbeiten beteiligt:

Betreuer

Aufgaben: Erstellt Themen, betreut Abschlussarbeiten (meldet an, plant Vorträge, erfasst Abgabe)
Erwartungen: Möchte schnell und einfach Abschlussarbeiten verwalten

Erstgutachter

Aufgaben: Genehmigt Themen, erstellt Erstgutachten für Abschlussarbeiten
Erwartungen: Möchte ausreichend über Themen informiert werden und sein Gutachten möglichst einfach eingeben können

Zweitgutachter

Aufgabe: Erstellt Zweitgutachten für Abschlussarbeiten
Erwartung: Möchte sein Gutachten wie üblich (d.h. nicht mit dem System, sondern über das Sekretariat) erstellen

Student

Aufgabe: Bearbeitet Themen im Rahmen von Abschlussarbeiten
Erwartungen: Reibungslose Verwaltung und faire Benotung seiner Abschlussarbeit

Sekretariat

Aufgabe: Erfasst Zweitgutachten
Erwartung: Unkomplizierte Eingabe des Zweitgutachtens in das System

Diese Gruppen sind nicht notwendigerweise disjunkt. So kann zum Beispiel ein Erstprüfer die Arbeit gleichzeitig betreuen.

3.1.3 Ablauf des Prozesses

Der Prozess (Abbildung 6) beginnt mit dem Erstellen des Themas der Abschlussarbeit durch den Betreuer. In diesem Schritt wird auch der Erstprüfer der Arbeit festgelegt.

Im nächsten Schritt wird das Thema dem Erstprüfer zur Genehmigung vorgelegt. Dieser hat die Möglichkeit das Thema abzulehnen oder zu genehmigen und diese Entscheidung zu kommentieren.

Wird das Thema abgelehnt, wird der Betreuer benachrichtigt und erhält die Möglichkeit das Thema zu überarbeiten. Im Anschluss wird es dem Erstprüfer erneut vorgelegt.

Wird das Thema akzeptiert wird es automatisch auf der Webseite des Fachgebiets Software Engineering veröffentlicht und ausgedruckt.

Nun wird der Betreuer aufgefordert sowohl einen Studenten für die Bearbeitung der Arbeit als auch einen Zweitgutachter zu benennen. Erst wenn beide Personen eingetragen sind, kann die Arbeit angemeldet werden.

Die Anmeldung der Arbeit soll nicht aktiv unterstützt werden. Der Betreuer bekommt lediglich eine Checkliste über die notwendigen Schritte.

Falls es sich bei der Arbeit um eine externe Abschlussarbeit handelt, wird der Betreuer nun aufgefordert einen Termin für den Anfangsvortrag festzulegen, der Termin wird automatisch dem Erstgutachter und dem WAT-System des Fachgebiet Software Engineering mitgeteilt.

Anschließend werden in jedem Fall Zwischen- und Abschlussvortrag, analog zum Anfangsvortrag geplant. Bei der Planung des Abschlussvortrages muss das System sicherstellen, dass der Termin nicht vor der geplanten Abgabe der Abschlussarbeit liegt.

Im nächsten Schritt erfasst und archiviert der Betreuer die Abgabe der Arbeit. Damit wird die Bewertungsphase eingeleitet.

In dieser können Betreuer, Erst- und Zweitgutachter die Arbeit bewerten. Dabei können nur Erst- und Zweitgutachter Noten vergeben, der Betreuer hat nur die Möglichkeit einen Kommentar zu verfassen. Der Zweitgutachter agiert nicht direkt mit dem System, sein Gutachten wird vom Sekretariat erfasst und in das System eingegeben.

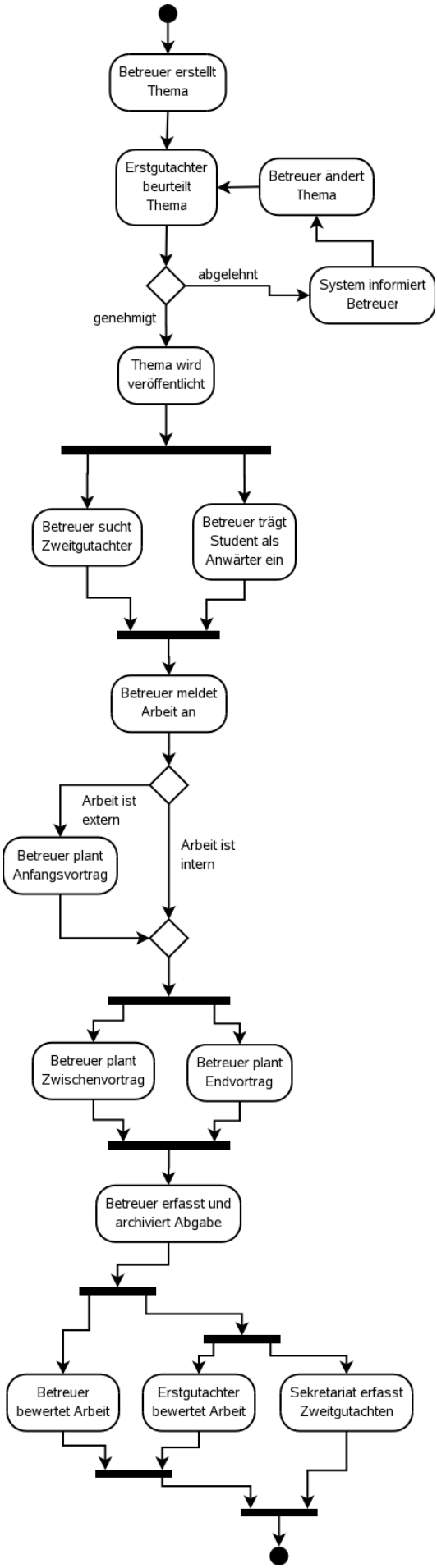

Abbildung 6: Prozessablauf

3.1.4 Anbindung an externe Systeme

Eine wichtige Anforderung an das System zur Verwaltung von Abschlussarbeiten ist die Anbindung an schon bestehende Systeme um Standardaufgaben automatisieren zu können.

Bei diesen Aufgaben handelt es sich um das Verschicken von E-Mails, einmal über das Simple Mail Transfer Protokoll (SMTP) und einmal über das WAT-Anmeldesystem des Fachgebiet Software Engineering, das Drucken von Aushängen und das Laden von Aushängen auf den WebServer des Fachgebiet Software Engineering.

Für das WAT-Anmeldesystem existiert bereits ein Webservice, die restlichen müssen noch erstellt werden. SOA-Me und BPEL sehen einen Webservice namens PrintService für das Drucken und einen Webservice namens MailService für das Versenden von Mails via SMTP vor. BPEL benötigt darüber hinaus einen Proxy-Webservice namens WAT-Manager für das aufrufen des WAT-Anmeldesystems. Der Service zum Upload wurde von keinem Entwurf realisiert.

3.1.5 Datenbankschema

Sowohl bei der BPEL als auch bei der SOA-Me Lösung kommt eine Datenbank zum Einsatz, in der u.a. alle zur Abschlussarbeitverwaltung nötigen Daten gespeichert werden. Diese ist nötig, da zum einen das SOA-Me System nicht persistent ist, die Daten also bei Abbruch des Prozesses verloren wären, und zum anderen benötigt der BPEL-Entwurf eine Möglichkeit Daten aus dem Prozess dem Client zur Verfügung zu stellen.

Die SOA-Me und die BPEL-Lösung benutzen dabei ein einheitliches Basisschema (siehe Abbildung 7), das in der Projektarbeit "Entwicklung einer Webservice-basierten Anwendung" entwickelt wurde. Da beide Lösungen dieses Schema nur minimal erweitern, sind die Datenbestände untereinander kompatibel.

Das Basisschema enthält folgende Entitäten:

Personen (Person) sind alle natürlichen Personen, die am Prozess beteiligt seien können, also Erst- und Zweitgutachter, Betreuer und Student sowie alle, die Zugang zum System haben (Mitarbeiter, Sekretariat). Zu jeder Person wird Anrede (salutation), Titel, Vor- und Nachname (firstName, lastName) sowie (falls vorhanden) Telefonnummer, E-Mail-Adresse, Matrikelnummer (studentId) und Zimmernummer gespeichert.

Eine Person kann beliebig viele Themen (Subject) betreuen, prüfen (als Erst- und Zweitgutachter) und bearbeiten. Da nicht explizit zwischen Mitarbeitern, Mitgliedern der Universität und Studenten unterschieden wird, sind noch zusätzliche Anforderungen an diese Beziehungen nötig: Eine Person darf niemals bei einem Thema sowohl als Erst- als auch als Zweitgutachter auftreten. Tritt sie als Erstgutachter oder Betreuer auf, muss ihr ein Benutzer (User) zugeordnet sein, da Betreuer und Erstgutachter mit dem System direkt interagieren müssen. Ferner darf eine Person nur ein Thema zurzeit bearbeiten und dieses Thema dann weder betreuen noch prüfen.

Benutzer (User) sind Personen, die zum Nutzen des Systems berechtigt sind und ordnen ihnen einen Usernamen und ein Passwort zum Login in das System zu. Es ist möglich, dass einer Person mehrere Benutzer zugeordnet werden, so kann sich eine Person unter mehreren Pseudonymen im System einloggen. Einem Benutzer können darüber hinaus Daten zum Login in das WAT-System des Fachgebiet Software Engineering zugeordnet werden. Über dieses System werden unter anderem die Massenbenachrichtigungen über die Vorträge zu Abschlussarbeiten verschickt.

Themen (Subject) stellen vom Betreuer vorgeschlagene Themen für Abschlussarbeiten dar. Einige der Attribute dieser Entität lehnen sich an die vom Fachgebiet Software Engineering

verwendeten Aushänge über Abschlussarbeiten an. Dazu gehören der Titel, der voraussichtliche Typ der Arbeit (titleType), die Aufgabe, der Hintergrund, der frühest mögliche Beginn (appliableDate) und die Randbedingungen, sowie die internen Zusatzangaben für das Fachgebiet Software Engineering über Bewertungskriterien (criteria), „Benutzerprofil und Aufgabenkontext“ (profile) und den Grund der Ausgabe (reason).

Darüber hinaus beinhaltet ein Thema die Daten der Vorträge, das Datum des offiziellen Beginns der Abschlussarbeit sowie das geplante Abgabedatum. Falls der Erstgutachter das Thema nicht genehmigt hat, wird seine Begründung für diese Entscheidung ebenfalls beim Thema abgelegt (rejectReason). Jedem Thema sind vier Personen zugeordnet: Ein Student, der das Thema bearbeitet, ein Betreuer und Erst- und Zweitgutachter. Außerdem kann ein Thema zugeordnet sein.

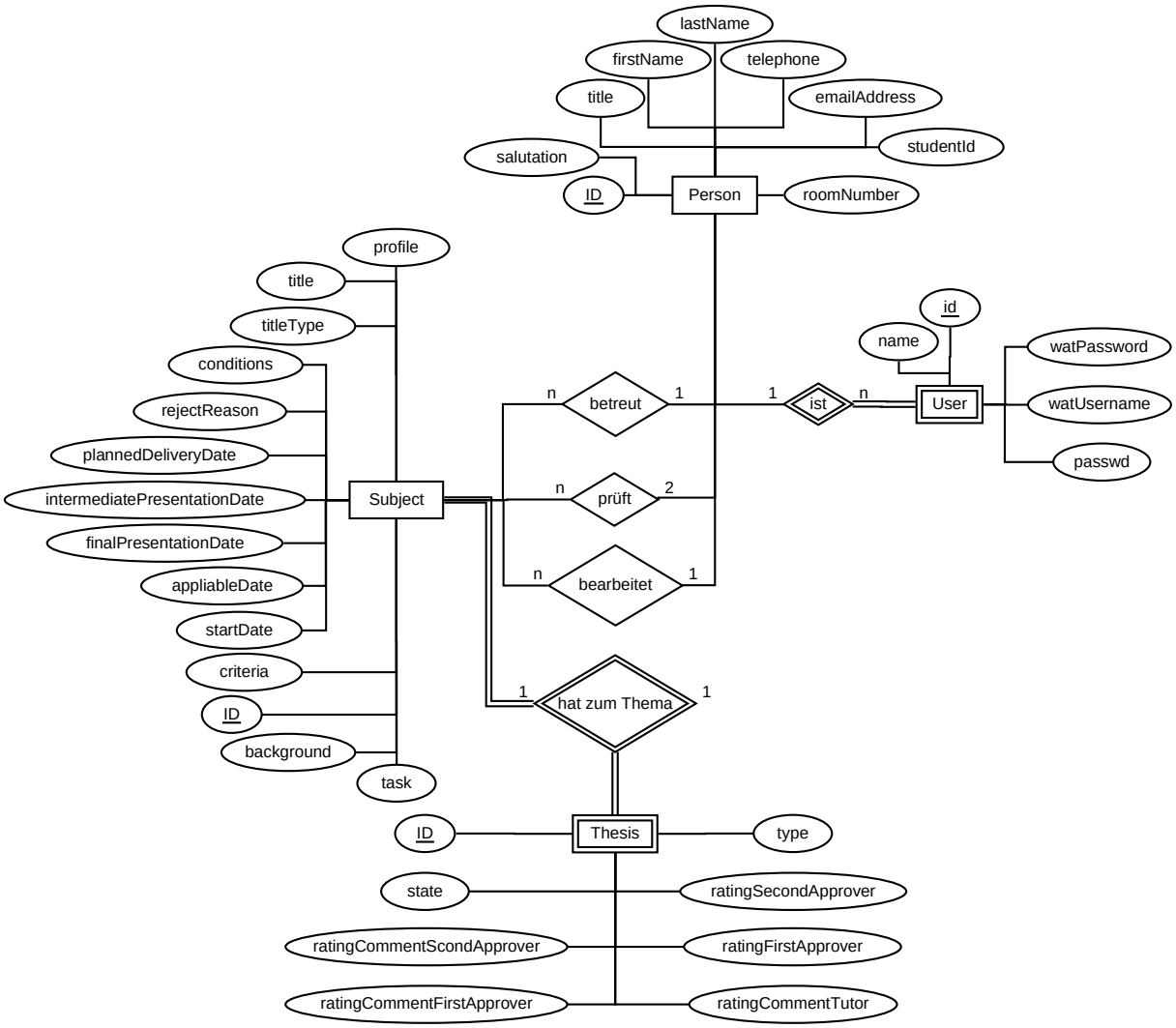

Abbildung 7: grundlegendes Datenbankschema

Abschlussarbeiten (Thesis) beinhalten den aktuellen Status der Arbeit (noch nicht genehmigt, zurückgewiesen, zu vergeben, vergeben, abgegeben, beendet oder abgebrochen), den letztendlich angemeldeten Typ der Arbeit (Bachelor-, Master-, Studien- oder Diplomarbeit) sowie die Bewertung von Erst- und Zweitgutachter als Kommentar und Note und die Bewertung vom Betreuer als Kommentar. Eine Abschlussarbeit ist einem Thema fest zugeordnet.

3.1.5.1 Schwächen des Datenbankschemas

Die größte Schwäche des Entwurfs ist die Aufteilung der Attribute zwischen Abschlussarbeit und Thema. Ein Thema ohne zugeordnete Arbeit macht praktisch keinen Sinn, da der Student, der die Arbeit bearbeitet dem Thema zugeordnet wird und nicht der Arbeit. Umfangreiche Themen, die von zwei Studenten bearbeitet werden, sind so nicht darstellbar. Auch wird der Status der Arbeit nicht am Thema gespeichert, sondern bei der Abschlussarbeit, was dazu führt, dass am Thema alleine nicht feststellbar ist, ob es schon vom Erstprüfer genehmigt wurde oder nicht, da diese Information im Status-Attribut abgelegt wird.

Ein weiteres Problem ist, dass in dem Schema keine Rollen oder Gruppen von Personen modelliert werden können. So können Studenten, Mitarbeiter, Prüfer und Sekretariat nicht unterschieden werden. Am deutlichsten wirkt sich das beim Sekretariat aus, da es direkt mit dem System interagieren muss (Erfassen des Zweitgutachtens), aus dem Schema aber nicht hervorgeht, welche Personen zum Sekretariat gehören.

3.2 BPEL

Der BPEL-Entwurf ist sehr umfangreich, da die BPEL-Plattform im Vergleich zur SOA-Me Plattform viele der geforderten Funktionen nicht bietet. Im Folgenden wird, nach der Vorstellung der allgemeinen Prozess-Architektur, auf die Entwürfe dieser Funktionen eingegangen.

3.2.1 Prozess-Architektur

Der BPEL-Entwurf unterteilt den Prozess aus 3.1.3 hierarchisch in einen Hauptprozess und 12 Unterprozesse die vom Hauptprozess aufgerufen werden (siehe Abbildung 8). Jeder dieser Unterprozesse erledigt nur eine atomar ablaufende Funktion wie das Erstellen eines Themas oder Planen eines Vortrags.

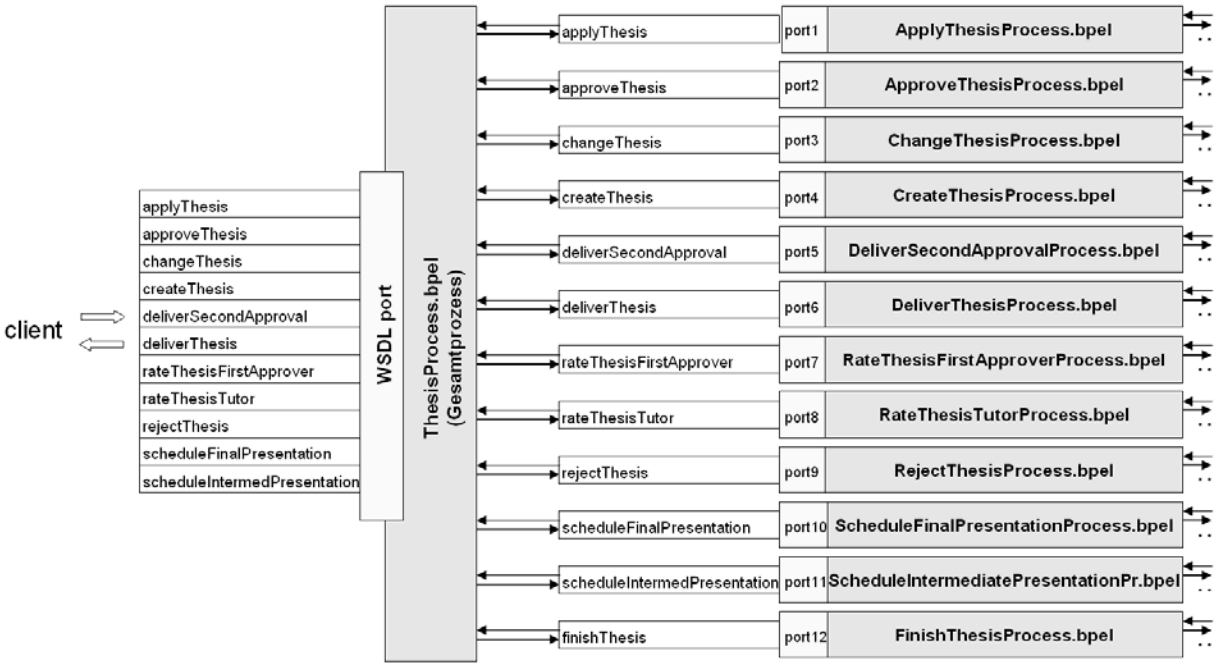

Abbildung 8: Prozess-Architektur des BPEL-Entwurfes [Sal07]

Diese Architektur ähnelt sehr dem Entwurfsmuster Fassade (Facade-Pattern) aus [GHJV95], welches die Komplexität eines Teilsystems (hier die Unterprozesse) hinter einem einzelnen Modul (hier der Hauptprozess) versteckt. Im Gegensatz zu diesem Entwurfsmuster beschränkt sich der Hauptprozess nicht nur darauf, die von den Teilprozessen angebotenen Operationen

an den Client weiterzureichen. Jeder gestartete Hauptprozess verwaltet genau eine Abschlussarbeit. Er sorgt dafür, dass nur solche Operationen nach außen verfügbar sind, die auf die Abschlussarbeit in ihrem gegenwärtigen Status zugelassen sind. Der Hauptprozess bildet also den in 3.1.3 vorgestellten Ablauf ab. So ist sichergestellt, das der Client zum Beispiel nach dem Anmelden einer Arbeit dessen Thema nicht mehr zurückweisen kann.

Die Unterprozesse sind alle sehr ähnlich aufgebaut. Zunächst überprüfen sie, ob der Benutzer, der für den Aufruf verantwortlich ist, zu diesem Aufruf autorisiert ist (näheres dazu in Kapitel 3.2.2). Danach laden sie alle für die Operation nötigen Daten aus der Datenbank nach und führen anschließend die für die Operationen benötigten Aktivitäten und Webservice aus.

3.2.2 Sicherheitssystem

Der BPEL-Entwurf sieht ein kompliziertes Sicherheitssystem vor. Jeder Webservice-Aufruf eines Prozesses oder Webservices muss von einem zentralen Authentifizierungs-Webservice autorisiert werden. Dieser Webservice ist mit dem Benutzernamen und dem Passwort eines Superusers geschützt. Diese Daten werden in einem OASIS-Web-Services-Security-Header übertragen.

3.2.2.1 Ablauf

Den Ablauf der Authentifizierung zeigt Abbildung 9. Zu Beginn loggt sich der Benutzer beim Client ein. Dieser ruft den Authentifizierungs-Webservice mit dem vom Benutzer eingegebenen Benutzernamen (u) und Passwort (pw als MD5-Hash) sowie einem Security-Header auf, der den Benutzernamen (su) und das Passwort (spw) des Superusers enthält. Der Superuser ist dem Benutzer nicht bekannt. Sind Benutzername und Passwort des Benutzers korrekt antwortet der Authentifizierungs-Webservice positiv.

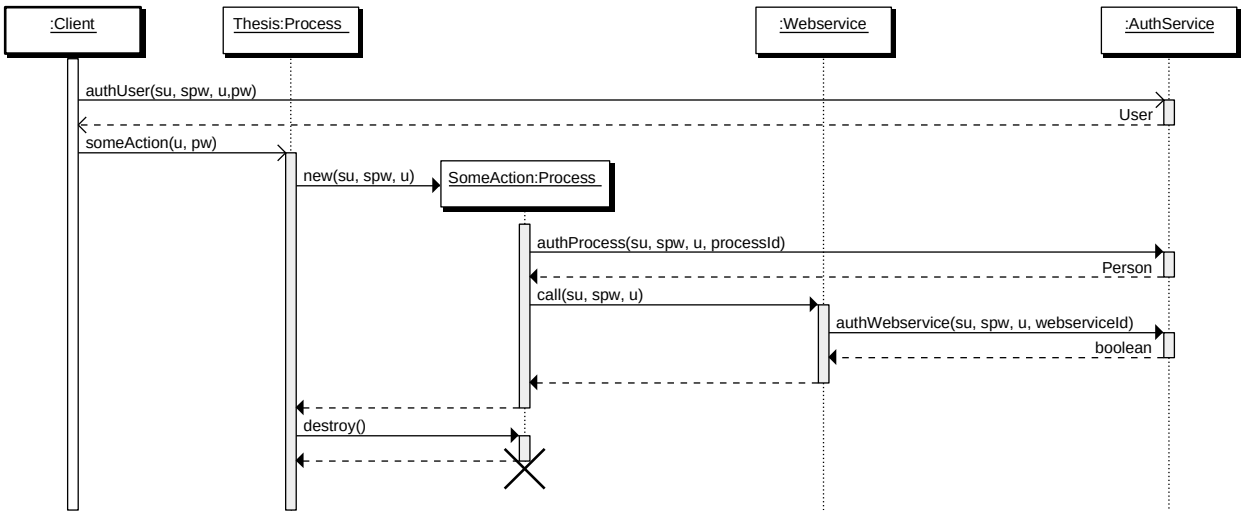

Abbildung 9: Ablauf der Authentifizierung durch das Sicherheitssystem des BPEL-Entwurfs

Nun kann der Client den Hauptprozess (siehe 3.2.1) aufrufen. Dieser Aufruf enthält nun Benutzernamen und Passwort des Benutzers im Security-Header. Der Hauptprozess extrahiert den Benutzernamen aus dem Security-Header und ruft mit diesem und den Benutzerdaten des Superusers, den für die vom Client angeforderte Funktion nötigen Unterprozess auf. Dieser prüft erst einmal durch eine Anfrage an den Authentifizierungs-Webservice, die Benutzername, Prozessidentifikation und im Security-Header die Benutzerdaten des Superusers enthält, ob der Benutzer zur Ausführung des Prozesses berechtigt ist. Ist er es, wird daraufhin der nötige Webservice aufgerufen, wieder mit den Daten des Superusers und dem Benutzernamen. Dieser prüft die Berechtigung analog zum Unterprozess und fährt dann normal fort.

Nach vollendetem Webserviceaufruf terminiert der Unterprozess nach einer Meldung an den Hauptprozess.

Dieses Vorgehen ist unnötig kompliziert. Es würde vollkommen ausreichen, wenn der Hauptprozess die Legitimation des Benutzers überprüfen würde.

3.2.2.2 Datenbankerweiterungen

Client, Unterprozess und Webservice prüfen also bei jedem Aufruf die Berechtigungen des Benutzers. Die Rechte eines Benutzers werden dabei in der Datenbank gespeichert. Dazu muss das Schema aus 3.1.5 um zwei Entitäten (siehe Abbildung 10) erweitert werden: Dem Process und dem Webservice.

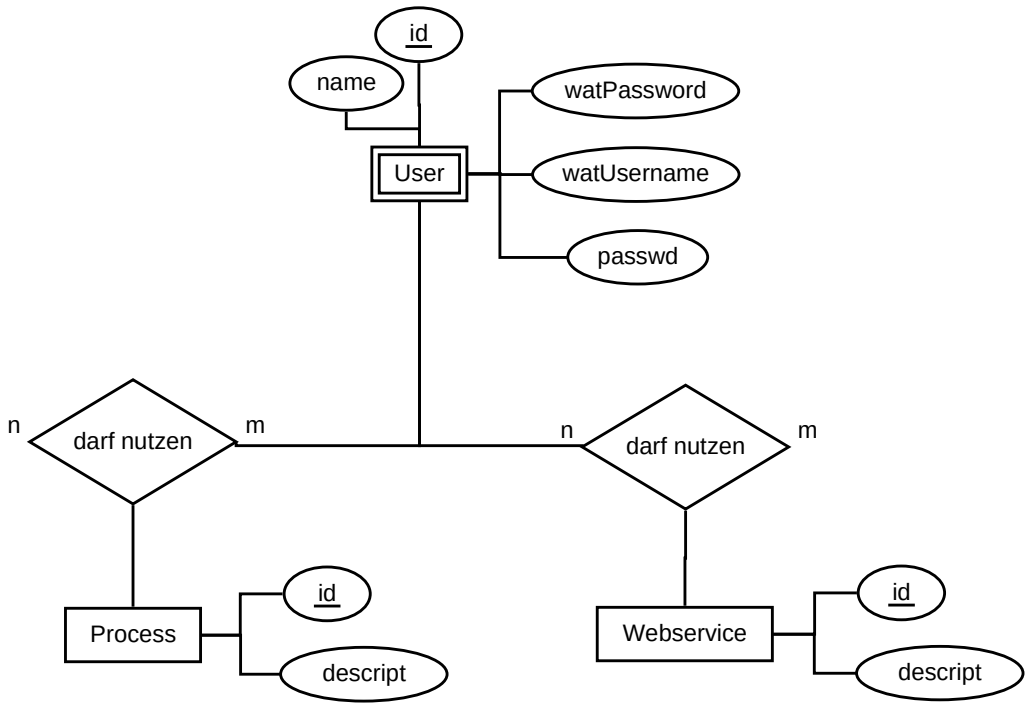

Abbildung 10: Änderungen des Grundschemas für BPEL

Beide Entitäten sind sehr ähnlich, sie besitzen jeweils das Attribut id und descript und identifizieren jeweils einen Unterprozess bzw. Webservice, wobei descript eine Beschreibung darstellt. Sie stehen auch in derselben Beziehung zu Benutzern (User), diese dürfen beliebig viele Webservices bzw. Prozesse nutzen. Umgekehrt dürfen Webservices und Prozesse auch von beliebig vielen Benutzern benutzt werden. Durch diese Gemeinsamkeiten fällt auf, dass diese Erweiterung etwas vereinfacht werden könnte, indem man Webservices und Prozesse zu einer Entität zusammenfasst.

3.2.2.3 Schwachstellen

Obwohl das Sicherheitssystem auf den ersten Blick sehr durchdacht scheint, weißt es doch einige Sicherheitsmängel auf:

- Die Datenübertragung erfolgt unverschlüsselt
- Die Hashfunktion wird auf eine relativ unsichere Weise verwandt

Da die Übertragung der Anfragen an den Authentifizierungs-Webservice unverschlüsselt erfolgt, kann ein Angreifer die Kommunikation mit relativ einfachen Mitteln im Klartext mitlesen. Dabei wird bei der Anfrage vom Client an den Authentifizierungs-Webservice zwecks

Authentifizierung des Benutzers das Passwort nicht im Security-Header sondern als direkten MD5-Hash (das heißt, dass der Wert direkt durch Anwenden der Hashfunktion auf das Passwort erfolgt) übergeben.

In [Oec03] wurde eine Datenstruktur, die so genannte Rainbow-Table, vorgestellt, die eine schnelle, probabilistische Entschlüsselung von Hash-Werten ermöglicht. Somit ist das entschlüsseln der gehashten Passwörter unter bestimmten Vorraussetzung (Vorhandensein einer geeigneten Rainbow-Table) effizient möglich. Einem Internetdienst, der auf diesem Konzept basieret, war es innerhalb von zwei Stunden möglich das Passwort des Superusers (alphanumerisches Passwort aus 6 Zeichen) zu entschlüsseln.

3.2.3 Gruppenverwaltung

Der BPEL-Entwurf sieht keine explizite Gruppenverwaltung vor. Es wird aufgrund von schon vorhandenen Daten auf die Zugehörigkeiten der einzelnen Personen geschlossen. Trat eine Person z.B. schon einmal als Bearbeiter eines Themas auf, so wird sie vom PersonManager Service (einer der Webservices, der die Datenbank abstrahiert) als Student geführt. Personen die Arbeiten betreuen oder prüfen werden als Mitarbeiter geführt. Dabei geht der Entwurf von der Annahme aus, dass jede Person, die in die Datenbank eingetragen wird, automatisch zu einer Gruppe gehört. Es wird also davon ausgegangen, dass eine Person unmittelbar nachdem sie in das System eingetragen wurde, in irgendeiner Weise (z.B. als Prüfer, Betreuer oder Student) einer Abschlussarbeit zugeordnet wird. Auch ist nicht vorgesehen, dass Personen in mehreren Gruppen Mitglied sind, oder diese wechseln. Der Datensatz für einen neuen Mitarbeiter, der schon dem System als Student bekannt war, muss erneut angelegt werden, anstatt die Daten aus dem bestehenden Datensatz weiter zu verwenden, denn der vorhandene Datensatz gilt implizit als Student, der z.B. nicht berechtigt ist Abschlussarbeiten zu betreuen.

Das größte Problem dieses Ansatzes ist jedoch, dass Personen, die in den vom Prozess benötigten Daten nicht vorkommen (in diesem Fall die Mitarbeiter des Sekretariats) keiner Gruppe zugeordnet werden können. In der Implementierung des BPEL-Entwurfs führt das dazu, dass die Namen der Sekretariats-Mitarbeiter als magische Konstanten im Quellcode der Webservices auftauchen.

3.2.4 Aufgabenverwaltung

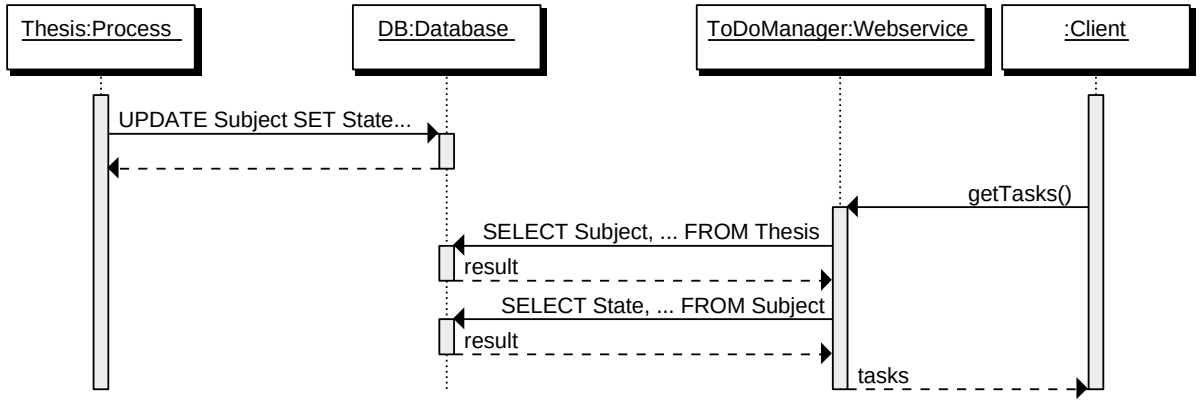

Abbildung 11: Ablauf der Aufgabenverwaltung

Der BPEL-Entwurf sieht einen extra Webservice für das Verwalten von Aufgaben vor: Den ToDo-Manager. Da der BPEL-Prozess dem Client nicht mitteilen kann, dass eine Aufgabe ansteht, muss dies der Client selbst tun, indem er periodisch den ToDo-Manager aufruft (siehe Abbildung 11). Der ToDo-Manager fragt daraufhin die Datenbank ab und leitet aus dem State-Attribute der Thesis-Entität (siehe Abbildung 7) sowie den Attributen der Subject-Entität die aktuelle Position im Thesis-Process ab. Damit stehen die zu erledigenden Aufgaben fest

und werden, sofern sie den am Client eingeloggtem Benutzer betreffen, an den Client zurückgegeben.

Dieses Konzept hat einen Nachteil: Der Fortschritt eines Prozesses wird redundant an zwei Stellen gleichzeitig gespeichert. Einmal explizit über die Datenbank und einmal implizit über die aktuelle Position im Prozess, die von der Persistenzschicht des BPEL-Servers gespeichert wird. Für die korrekte Funktion des Systems sind beide Speicherstellen nötig, denn folgende zwei Fehlerszenarien sind denkbar:

Falls der Eintrag über die Abschlussarbeit, die der Prozess verwaltet, in der Datenbank verloren geht (Fehler im Datenbank-Management-System), kann der Client nicht mehr feststellen, welche Aufgaben für diesen Prozess noch ausgeführt werden müssen. Da der Prozess, wie in 3.2.1 vorgestellt, notfalls unendlich lange auf einen Aufruf eines geöffneten Webservice-Ports wartet, kann er nicht mehr terminiert werden. Die Arbeit ist verloren, obwohl der Prozess in seinen Variablen noch alle Daten bereithält.

Falls der Prozess, während er auf Aktionen des Clients wartet, unerwartet terminiert wird (Fehler im BPEL-Server speziell in dessen Persistenzschicht), sind alle Daten über der zu ihm gehörigen Arbeit zwar noch in der Datenbank vorhanden und der ToDo-Manager kann die noch zu erledigenden Aufgaben korrekt bestimmen, aber der Aufruf des Prozesses durch den Client misslingt, da der zugehörige Webservice-Port geschlossen wurde. Damit ist die Arbeit ebenfalls verloren.

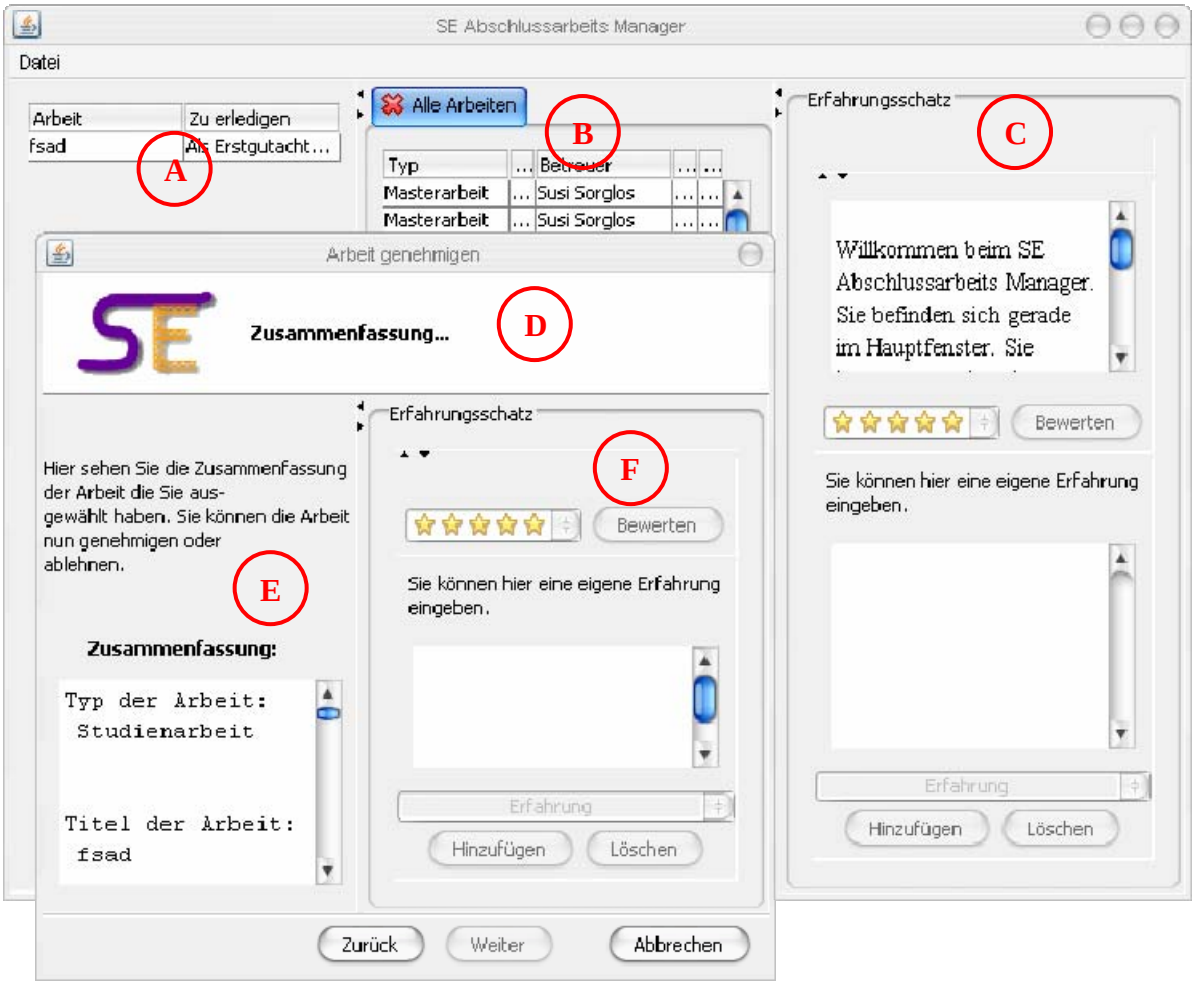

Abbildung 12: Design des Clients des BPEL-Entwurfs

3.2.5 Benutzeroberfläche

Die Benutzeroberfläche des Clients des BPEL-Entwurfs ist wie in Abbildung 12 gezeigt aufgebaut.

Die ersten Anlaufstellen für den Benutzer ist das Menü (Datei), mit Hilfe dessen er neue Themen anlegen und sich laufende Arbeiten anzeigen lassen kann, und die ToDo-Liste (A) in der die noch ausstehenden Aufgaben angezeigt werden. Das Anzeigen der Arbeiten zu Informationszwecken geschieht im Anzeigebereich (B) tabellarisch in mehreren Registerkarten. Aktive Aufgaben für den Benutzer (neues Thema erstellen; Aufgabe aus ToDo-Liste berabeiten) werden in einem neuen Fenster (D) angezeigt, welches in einen Arbeitsbereich (E) und das Experience Forum (F) unterteilt ist. Alle Aktionen des Benutzers, die mit der Aufgabe direkt zu tun haben geschehen im Aufgabenbereich. Im Experience Forum hat der Benutzer die Möglichkeit seine Erfahrungen zu der aktuellen Aufgabe anzugeben und Seiten des Handbuchs aufzurufen. Das Experience Forum ist noch mal im Hauptfenster (C) vorhanden.

3.3 SOA-Me

Der SOA-Me Entwurf ist nicht so umfangreich wie der BPEL-Entwurf. Dies liegt daran, dass die SOA-Me Plattform viele der in 3.1 genannten Anforderungen schon erfüllt. Funktionen wie das Experience Forum oder eine Aufgaben- und Gruppenverwaltung existieren bereits und müssen nicht wie bei BPEL zusätzlich entworfen werden.

Auch ein Sicherheitssystem ist bereits vorhanden, wenngleich dieses keine Autorisierungsmechanismen für das Ausführen von Webservices beinhaltet. Das Problem mit diesem System ist nur, dass es die Benutzerdaten nicht in einer relationalen Datenbank sondern in einer XML-basierten Konfigurationsdatei ablegt. Damit müssen die Benutzerdaten einmal in dieser Konfigurationsdatei und (zumindest der Benutzername) einmal in der Datenbank abgelegt werden, damit der Prozess des SOA-Me-Entwurfes den in der Datenbank gespeicherten Personen Benutzernamen zuordnen kann. Es ist also im Betrieb darauf zu achten, dass Datenbank und Konfigurationsdateien synchron sind. Wünschenswert wäre eine erweiterte Schnittstelle für den SOA-Me Server, so dass dieser die Benutzerdaten aus einer Datenbank verwenden kann. Diese Schnittstelle wurde in dieser Arbeit aus Zeitgründen weder entworfen noch implementiert. Am Rande sei noch erwähnt, dass das Sicherheitssystem der SOA-Me Pattform wie das Sicherheitssystem des BPEL-Entwurfs unverschlüsselte Verbindungen und die einfache Anwendung einer Hashfunktion nutzen, so dass ein Angriff wie er in 3.2.2.3 vorgestellt wurde auch bei SOA-Me funktionieren würde.

Auch der SOA-Me Entwurf sieht eine Erweiterung des Datenbankschemas (siehe Abbildung 7) vor: Um auch externe Arbeiten unterstützen zu können, die nach 3.1.3 vorgesehen sind aber vom BPEL-Entwurf nicht berücksichtigt werden, werden die Entitäten Thesis und Subject je um ein Attribut erweitert. Subject erhält zusätzlich das Attribut startPresentationDate, dass das Datum des Startvortrags für externe Arbeiten enthält; Thesis erhält das zusätzliche Attribut internally, das als boolscher Wert anzeigt, ob die Abschlussarbeit als externe oder interne Arbeit läuft.

3.3.1 Prozess-Aufbau

Der SOA-Me-Entwurf sieht drei Prozesse vor. Ein Prozess zum Anlegen und Verwalten einer Abschlussarbeit (Hauptprozess), ein Prozess zum Fortsetzen einer durch einen Serverfehler unterbrochenen Abschlussarbeitsverwaltung (Persistenzprozess) und ein Hilfsprozess zum Anlegen neuer Personen. Die ersten beiden Prozesse verwenden dabei große Prozessteile gemeinsam.

Im Gegensatz zum BPEL-Entwurf wird der Prozess aus 3.1.3 nicht in mehrere Teilprozesse aufgeteilt sondern als ganzes modelliert (Hauptteil). Dies hat mehrere Gründe: Zum einen

erlaubt es die SOA-Me Plattform schlicht nicht aus einem Prozess einen anderen Prozess aufzurufen (die entsprechenden Elemente, Prozesswegweiser genannt, sind in der EPK-Notation zwar vorhanden, werden aber von der SOA-Me Plattform nicht unterstützt), zum anderen wäre damit auch die Implementierung der expliziten Persistenz (siehe 3.3.2) und der einheitlichen Fehlerbehandlung erschwert worden.

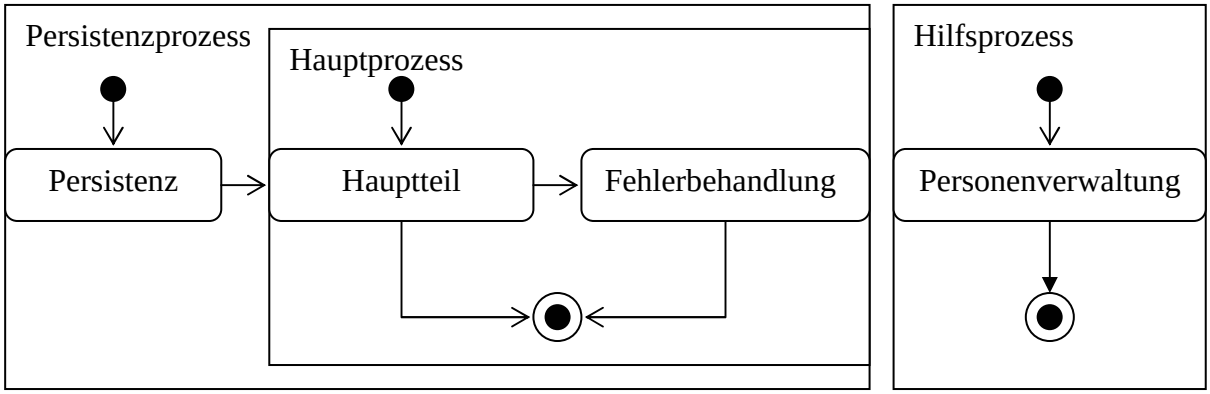

Abbildung 13: konzeptioneller Aufbau der verschiedenen Prozesse

Wie in Abbildung 13 zu sehen, besteht der Hauptprozess aus zwei Teilen. Der Hauptteil des Prozesses implementiert alle funktionalen Anforderungen an die Abschlussarbeitsverwaltung. Im normalen fehler- und abbruchsfreien Prozessablauf wird dieser Teil nie verlassen.

Ein weiterer Teil ist für die Fehlerbehandlung zuständig. Kommt es bei einem Webserviceaufruf zu einem Fehler sorgt dieser Teil dafür, dass der Benutzer über den Fehler informiert und der Prozess kontrolliert beendet werden kann.

Der Persistenzprozess umfasst alle Teile des Hauptprozesses (bis auf den Startpunkt) und erweitert ihn um einen für die Persistenz zuständigen Teil. Hier wird wie in 3.3.2 beschrieben beim Fortsetzen des Prozesses an die entsprechende Stelle gesprungen.

Trotz des hohen Überdeckungsgrades des Haupt- und Persistenzprozesses können beide nicht in einem Prozess zusammengeführt werden, da das SOA-Me-System nur einen Startereignis pro Prozess zulässt.

Der Hilfsprozess ist sehr einfach aufgebaut. Er besteht nur aus einer Komponente, die sich um das Anlegen der Personen kümmert.

3.3.2 Persistenz

Ein großes Problem beim SOA-Me System ist die fehlende Persistenz: Das System ist nicht in der Lage einen laufenden Prozess zu suspendieren, seinen Status zu speichern und später fortzusetzen. Dies bedeutet konkret für das Abschlussarbeit-Verwaltungssystem, dass der SOA-Me Server nicht deaktiviert werden oder ausfallen kann, ohne dass alle Daten zu den laufenden Abschlussarbeiten verloren gehen.

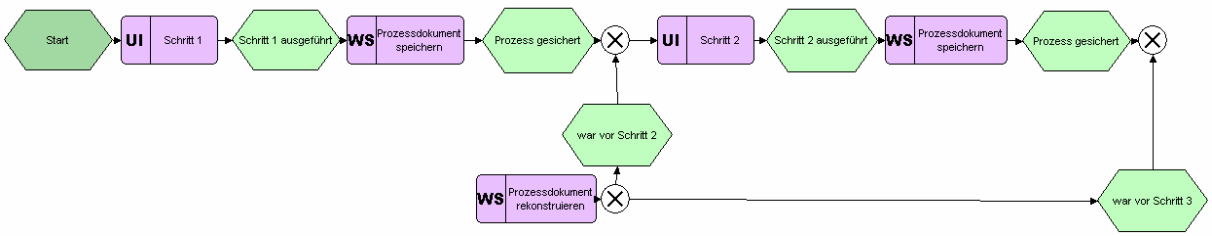

Abbildung 14: Simpler, um Persistenz erweiterter, Prozess

Eine Lösung wäre eine Persistenzschicht in den SOA-Me Server einzubauen, diese könnte jeden vom Server gemachten Schritt protokollieren sowie die zum Prozess gehörenden Daten (Prozessdokument) speichern. Mit diesen Daten wäre ein Fortsetzen des Prozesses beim Serverstart möglich. Diese Persistenzschicht wäre aber zu komplex, um sie im Umfang dieser Arbeit zu entwickeln.

Daher wurde eine andere Lösung verfolgt, die keine Änderung an dem SOA-Me Server erfordert: Der Prozess kümmert sich selber um das Speichern seiner Daten. Dazu sind prinzipiell zwei Webservices nötig (vgl. Abbildung 14): Ein Webservice speichert alle relevanten Daten aus dem Prozessdokument in eine Datenbank, der zweite Webservice liest diese Daten aus der Datenbank aus und rekonstruiert daraus das Prozessdokument.

Das Speichern des Prozessdokumentes ist dabei relativ unproblematisch: In den bestehenden Prozess muss nur an geeigneten Stellen der entsprechende Webservice gestartet werden.

Das Wiederherstellen des Prozesses erfordert aber nicht nur die Rekonstruktion des Prozessdokumentes, sondern auch die Bestimmung des vom Server zuletzt ausgeführten Schrittes, der Position im Prozess. Dieses kann aber im Fall des Abschlussarbeits-Verwaltungssystems durch eine Analyse des Prozessdokuments geschehen, da der aktuelle Status („zu genehmigen“, „zu vergeben“, „abgegeben“ usw.) dort vermerkt ist.

Steht die Position fest, kann an die entsprechende Stelle in den Prozess „eingesprungen werden“, und der Prozess wird von da an fortgesetzt, als wäre er nie unterbrochen worden.

Nachteilig an dieser Lösung ist allerdings, dass der Prozess durch die zahlreichen Einsprünge schnell relativ unübersichtlich wird; besonders dann, wenn eine parallele Prozessabarbeitung erfolgt, da für jeden Prozesszweig festgestellt werden muss, in welchem Zustand er sich befand. Auch muss das Fortsetzen eines Prozesses manuell und für jeden Prozess einzeln geschehen und kann nicht automatisch beim Start des Servers erfolgen.

Außerdem werden die Informationen über die Abschlussarbeit redundant im laufenden Prozess (genauer im Prozessdokument) und in der Datenbank gespeichert. In den beiden Fehlerszenarien aus 3.2.4 (Fehler in der Datenbank, Fehler im Server) ist das unproblematisch. Im Gegenteil: Dadurch, dass entweder der Prozess oder die Datenbank alle Informationen besitzt, kann der Prozess normal weiterlaufen bzw. manuell wieder zum Laufen gebracht werden. Allerdings kann nicht ausgeschlossen werden, dass ein korrekt laufender Prozess manuell erneut „fortgesetzt“ wird. Dies führt dazu, dass zwei Prozesse um dieselbe Abschlussarbeit konkurrieren, was zu unvorhersehbaren Verhalten der beiden Prozesse führt. Insbesondere ist nicht klar, welcher Prozess welche Daten in die Datenbank geschrieben hat.

4 Implementierung und Entwicklung

Die Implementierung war in drei Inkremente aufgeteilt, wobei jedes Inkrement ca. zwei Wochen Entwicklungszeit in Anspruch nahm. Im ersten Inkrement wurden die Datenbankwebservices (4.2.1), der Hilfsprozess und erste Teile des Hauptprozesses, darunter die wesentlichen Teile für die Persistenz, entwickelt. Die Entwicklung des Prozesses wurde dann im zweiten Inkrement beendet. Das dritte Inkrement umfasste die Entwicklung sowie das Einbinden der Webservices für die externen Systeme Drucken, E-Mail und WAT-Service.

Bei der Implementierung musste darauf geachtet werden, dass der entwickelte Quellcode zu dem Quellcode, der aus dem BPEL-Entwurf hervorgegangen war, vergleichbar bleibt. So wurde schon in der Entwurfsphase möglichst viel vom BPEL-Entwurf übernommen (wie z.B. das Datenbankschema, siehe 3.1.5). Für die Implementierung bedeutete das möglichst dieselben Frameworks zu verwenden.

4.1 Prozesse

Wie schon in 3.3.1 erläutert gilt es drei Prozesse zu implementieren von denen zwei, der Haupt- und Persistenzprozess, sehr große Teile gemeinsam haben. Um den Aufwand gering zu halten wurden diese beiden Prozesse zunächst in einem Prozess zusammengefasst. Dabei wurden die beiden verschiedenen Startereignisse durch eine Fallunterscheidung miteinander verbunden, so dass beim Start des Prozesses zunächst eine Abfrage erschien, welcher der beiden Teil-Prozesse gestartet werden sollte. Nach Vollendung des Prozesses wurde dieser zweimal gespeichert und in jeder Kopie die Abfrage und alle von dem jeweiligen Teil nicht benötigten Elemente entfernt.

Der dritte Prozess (Hilfsprozess) dient zum Anlegen von neuen Personen in der Datenbank und ist sehr einfach aufgebaut. Nach dem Start wird der Benutzer nach den Personendaten und danach gefragt, ob diese Person Zugang zum System haben soll. Falls ja, wird zusätzlich noch nach den Benutzerdaten gefragt, danach werden die Daten in der Datenbank gespeichert.

Im Folgenden wird ausschließlich der Prozess der aus dem Zusammenschluss von Haupt- und Persistenzprozess hervorgegangen ist behandelt.

4.1.1 Aufbau des Prozessdokuments

Das Prozessdokument ist das Herzstück des Prozesses. In diesem XML-Dokument werden alle für den Prozess relevanten Daten abgelegt.

Die XML-Elemente lehnen sich sehr stark an das Datenbankschema an, mussten aber für einige Funktionen aufgeteilt werden, da während des Prozessablaufs Daten aus dem Prozessdokument nur gelesen oder angehängt aber nicht verändert werden können. Dieses Anhängen erfolgt bei jedem UI-Element einer Benutzerinteraktion. So ist jedem UI-Element mindestens ein XML-Element zugeordnet.

Ein Beispiel für die Gestalt des Prozessdokumentes ist im Anhang auf Seite 53 dargestellt.

4.1.2 Hauptteil

Die Implementierung des Hauptteils, also des Geschäftsprozesses, konnte relativ einfach aus den Anforderungen und der Darstellung in Abbildung 6 abgeleitet werden. Dieses Vorgehen soll anhand der Aktion „Betreuer plant Endvortrag“ erläutert werden.

Laut Anforderungen soll in dieser Aktivität der Betreuer dazu aufgefordert werden einen Termin (Datum und Uhrzeit) für den Endvortrag zu planen. Dieser Termin darf nicht nach dem Abgabetermin der Abschlussarbeit liegen. Nachdem der Termin geplant ist, soll dieser per Mail und per WAT-System bekannt gegeben werden.

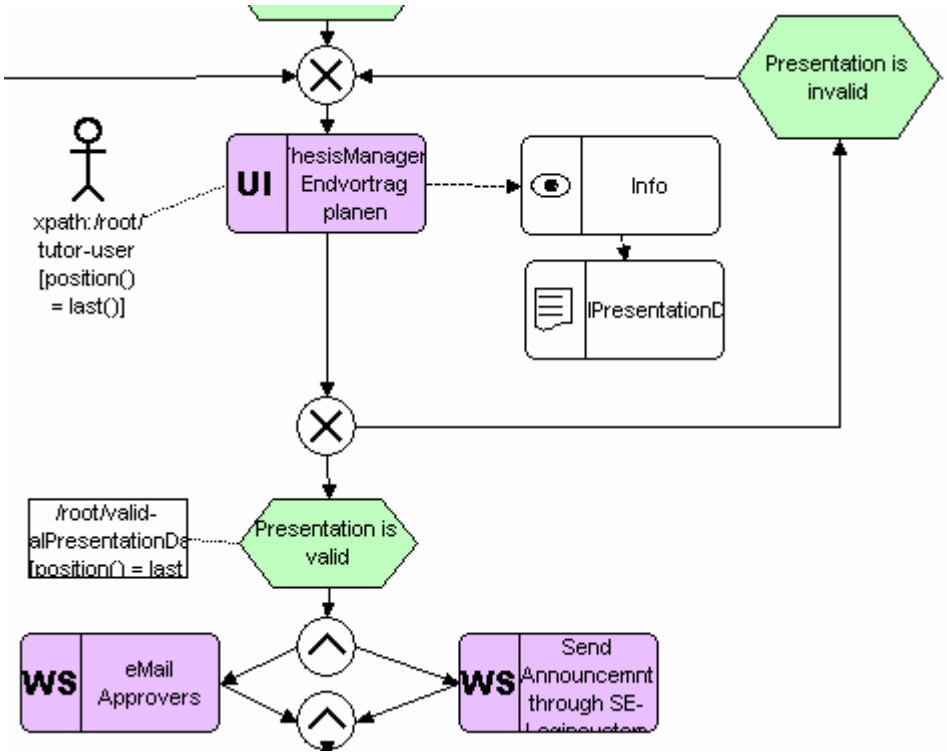

Abbildung 15: Planen des Endvortrags

Die Gestalt der EPK (siehe Abbildung 15) folgt direkt aus den Anforderungen. Eine Benutzerinteraktion informiert über ein Display-Element den Benutzer und gibt ihm mit dem Edit-Element eine Möglichkeit einen Termin einzugeben. Die dem Edit-Element zugeordneten Stylesheets (nicht im Bild) sorgen dafür, dass die vom Benutzer angegebenen Daten in das Format des Prozessdokuments konvertiert werden. Anschließend folgt durch ein XOR-Split und den Bedingungen an den Events die Abfrage, ob der Termin gültig ist, sich also nach dem Abgabetermin befindet.

Falls der Termin nicht gültig ist, wird nach dem XOR-Join fortgefahren und damit die Benutzerinteraktion erneut aufgerufen. Die Stylesheets, die zu dem Display-Element gehören, sorgen in diesem Fall dafür, dass der Benutzer eine entsprechende Nachricht erhält.

Falls der Termin gültig ist, werden die beiden Webservices zum Senden von Nachrichten via E-Mails und WAT-System parallel gestartet. Die Stylesheets, die zu diesen beiden Webserviceaufrufen gehören, sorgen dafür, dass die für den Versand nötigen Daten über die Abschlussarbeit und die Empfänger aus dem Prozessdokument extrahiert und in das vom Webservice erwartete Format konvertiert werden.

An diesem Beispiel lassen sich auch drei Probleme zeigen, die bei der Implementierung auftraten. So unterstützte der SOA-Me Client zunächst das XML-Standardformat xsd:datetime nicht, das Datum und Uhrzeit enthält. Diese Formatierung musste erst noch in dem SOA-Me Client eingebaut werden.

Das zweite Problem war, dass der SOA-Me Kompositionseditor keine „<“ und „>“ Zeichen in den Bedingungen an den Events unterstützte, die aber für die Abbfrage, ob der Termin für den Endvortrag gültig ist, benötigt wurde. So musste die Berechnung der Bedingung im Stylesheet

des Edit-Elements erfolgen, so dass an den Events nur abgefragt werden musste, ob die zuvor berechnete Bedingung wahr oder falsch war.

Das dritte Problem trat bei dem Aufruf des Webservices für das WAT-System auf. Dieser Webservice verwendete eine über SSL gesicherte Verbindung, die der SOA-Me Server nicht unterstützte. Das Problem konnte durch eine Anpassung des SOA-Me Servers gelöst werden.

4.1.3 Persistenz

Die Persistenz wurde nach dem Konzept in 3.3.2 eingebaut. Dabei wurden die Webserviceaufrufe zum Speichern des aktuellen Prozessdokumentes immer vor die nächste Benutzerinteraktion gesetzt. Diesem Aufruf folgt, nach dem obligatorischen Event, immer ein XOR-Join als Einsprungpunkt für das Fortsetzen des Prozesses. Beim Beispiel des Endvortrages erfolgte das Speichern vor dem XOR-Join, der gleichzeitig als Einsprungpunkt dient, und nach dem AND-Join.

Zum Laden des Prozessdokumentes waren mehrere Webserviceaufrufe nötig. Ein Webserviceaufruf für das Laden der Informationen über die Abschlussarbeit und je ein Aufruf für das Nachladen der Benutzerdaten zu dem in der Abschlussarbeit genannten Betreuer und Erstprüfer.

Bei dem Einspringen in den Prozess gab es allerdings ein Problem mit der EPK-Notation: Diese erlaubt keine aufeinander folgenden Events und damit keine verfeinernden Bedingungen. Dies war aber beim Einspringen in den parallelen Teil am Ende des Prozesses, wo Betreuer und Prüfer gleichzeitig ihre Bewertungen abgeben können, nötig. An dieser Stelle musste erst abgefragt werden, ob sich der Prozess zum Zeitpunkt des Abbruchs in dieser parallelen Ausführung befand und falls dies der Fall war für jeden parallelen Prozesszweig ob dieser schon ausgeführt wurde (also ob derjenige Betreuer oder Prüfer schon bewertet hatte) oder nicht. Das Problem musste durch das Deaktivieren der Überprüfungsroutine des Editors für diese EPK-Eigenschaft gelöst werden. Besser wäre eine leere Funktion als neues EPK-Element einzuführen um die aufeinander folgenden Events zu trennen.

4.1.4 Fehlerbehandlung

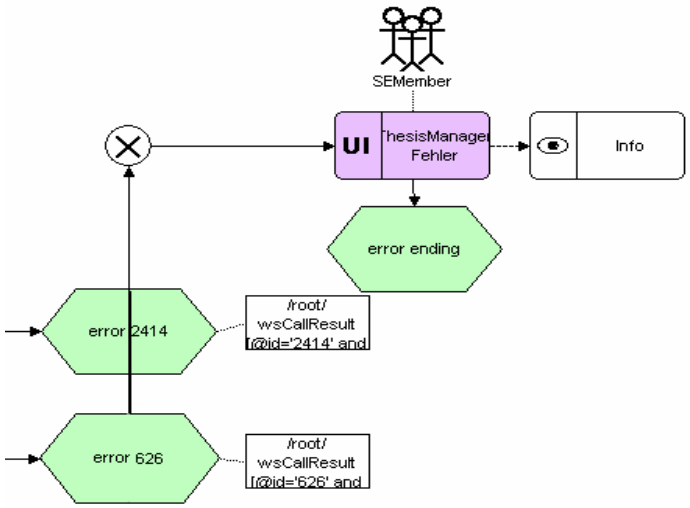

Abbildung 16: zentrale Fehlerbehandlung

Die Fehlerbehandlung dient dazu fehlgeschlagene Webserviceaufrufe zu erkennen und zu behandeln. Diese Art von Fehler wird vom SOA-Me Server toleriert und nur durch einen kurzen Eintrag ins Prozessdokument angezeigt. Bei allen anderen Arten von Fehlern wird der Prozess umgehend vom SOA-Me Server beendet.

Nach jedem Webservice-Aufruf muss also eine Entscheidung getroffen werden, ob dieser Aufruf erfolg hatte oder nicht. Dies wird durch ein einfaches XOR-Split realisiert, dessen zugeordnete Bedingungen grade auf die vom Server getätigten Eintragungen reagieren. Tritt ein Fehler auf, so wird dadurch der normale Prozessablauf unterbrochen und dem Benutzer eine Aufgabe, die eine Fehlermeldung enthält, gezeigt.

Um nicht für jeden Webservice eine eigene Benutzerinteraktion für die Fehlermeldung erstellen zu müssen werden alle Fehlerabzweigungen durch ein XOR-Join auf eine zentrale Benutzerinteraktion geführt, wie in Abbildung 16 zu sehen. Deren Stylesheet kann durch Auswerten des Prozessdokuments die exakte Fehlerstelle (ID des Webserviceaufrufes) angeben. Der Benutzer wird aufgefordert diese Information an den Administrator des Systems weiterzuleiten. Bis das Problem behoben ist wird der Prozess unterbrochen, kann dann aber über den Persistenzprozess kurz vor Eintreten des Fehlers fortgesetzt werden.

4.1.5 Benutzeroberfläche

Zu der Benutzeroberfläche ist nicht viel zu sagen. Die meisten Design-Elemente wie ToDo-Liste, Experienceforum usw. sind fest im SOA-Me Client eingebaut. Beim Design der Display-Elemente für die Anzeige der Informationen wurde sich an den Aushängen des Fachgebiets Software Engineering orientiert, da diese den Benutzern bekannt seien und so wenig Umstellung bedeuten sollte. Im Kopf dieser Aushänge wird immer der aktuelle Status (zu vergeben, zu genehmigen usw.) und der Titel der Arbeit angezeigt, so dass dem Benutzer auf einen Blick klar wird, was zu tun ist und um welche Abschlussarbeit es sich genau handelt.

4.2 Webservices

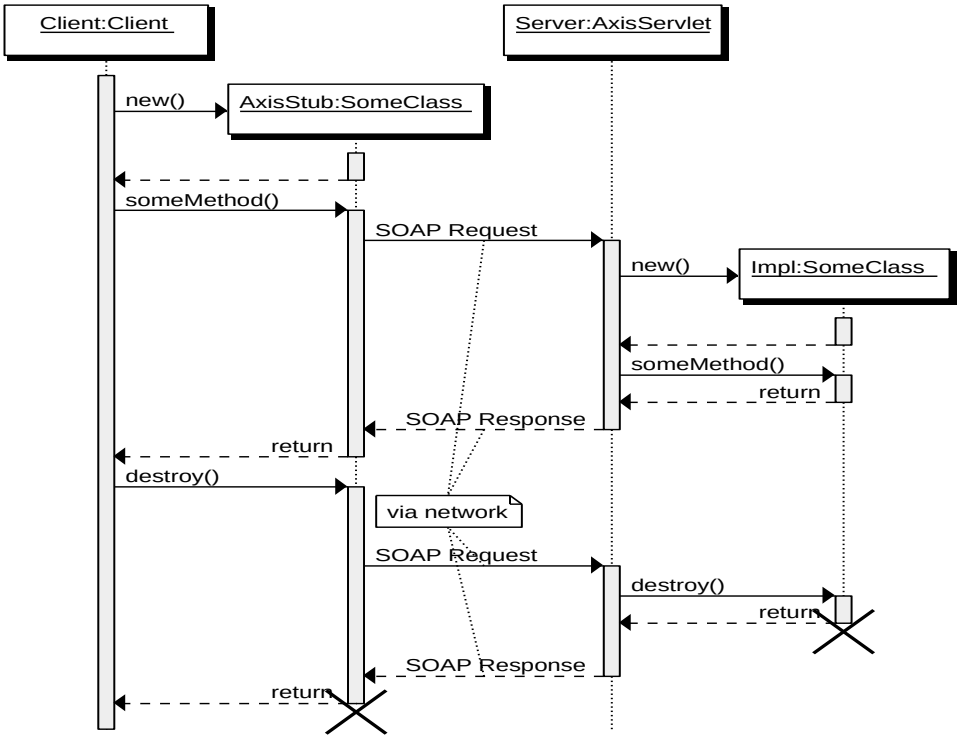

Abbildung 17: Funktionsweise des AXIS-Frameworks

Alle Webservices benutzen das Java-Framework AXIS (Apache eXtensible Interaction System) [Axi]. Es erleichtert das Erstellen und Aufrufen von SOAP-basierten Werbservices. Hier ist vor allem die Möglichkeit des Erstellens von Webservices gefragt, aufgerufen werden diese vom SOA-Me Server und damit aus dem Prozess. Axis erlaubt in Verbindung mit einem Servlet-Container wie Apache Tomcat eine nahezu transparente Kommunikation auf Java-

Ebene zwischen zwei über das Internet verbundene Systeme. Der Client kann den vom Server angebotenen Dienst wie ein lokales Objekt benutzen. Dabei werden die Methodenaufrufe von AXIS auf Clientseite in SOAP-Messages verpackt, verschickt und auf Serverseite an die implementierende Klasse weitergereicht.

Ein mächtiger Bestandteil des AXIS-Frameworks ist das Tool WSDL2Java. Es erstellt zu einem gegebenen WSDL-Dokument die Javaquellen für die Client und die Serverseite. Somit erleichtert sich das Erstellen eines Webservices enorm, es muss nur die WSDL-Datei geschrieben und die von AXIS daraus erstellte Skelettklasse mit Funktionalität gefüllt werden.

Die Stubs, die AXIS für die Clientseite generiert, können für Testfälle genutzt werden.

Alle Webservices verwenden einen gemeinsamen Stamm von Klassen, die der Datenhaltung dienen. Diese Klassen wurden durch das AXIS-Framework aus einem XML Schema erzeugt, das Bestandteil aller WSDL-Dateien ist. So wird redundanter Code vermieden, da beispielsweise die Klasse für die Abschlussarbeiten (Thesis) sowohl von den Datenbankservices als auch von dem Druckwebservice benötigt wird.

Die Struktur des Projektes, das die Webservices beinhaltet, ist so gewählt, dass alle Webservices gemeinsam in einem Webarchiv ausgeliefert werden können.

4.2.1 Datenbankwebservices

Es wurden insgesamt die drei Webservices UserManager, ThesisManager und PersonManager für die Datenbankanbindung implementiert, je einen für eine der drei Entitäten Thesis, Person und User aus dem Datenbankschema (siehe 3.1.5). Dabei werden die Datensätze aus der Datenbank in Objekte überführt (Object-Relational-Mapping). Die Schnittstellen der Webservices sind einander recht ähnlich. So existiert eine Operation zum Ändern eines bestehenden Objekts (ist dieses nicht vorhanden, so wird es erstellt) und zum Auflisten aller verfügbaren Objekte. Zum Zugriff auf ein bestimmtes Objektes bieten Thesis- und PersonManager eine Operation an, die das Objekt aufgrund seiner ID zurückgibt.

Der UserManager verhält sich etwas anders. Da der Benutzername eindeutig sein muss wird dieser als identifizierendes Attribut für den Zugriff verwand. Außerdem existiert eine Operation, die den User zu einer gegebenen Person (über dessen ID) findet. Diese Operation musste eingeführt werden, da dem Prozess durch das Laden einer Abschlussarbeit (Thesis-Objekt) zunächst nur die Personen bekannt sind. Im Fall des Betreuers und Erstprüfers ist aber auch der Username von Bedeutung. Da diese Operation nur ein User-Objekt zurückgibt, wird dadurch leider das Konzept der Aliase untergraben. Damit ist es einer Person nicht mehr möglich sich über verschiedene Benutzernamen am System anzumelden.

Wie auch bei der BPEL-Implementation verwenden die Datenbankwebservices das Hibernate-Framework [Hib] für das Object-Relational-Mapping. Wie oben kurz angerissen sorgt dieses Framework für die Übersetzung von Objekten in Datensätze einer relationalen Datenbank. Diese Übersetzung wird über Mapping-Dateien realisiert, in denen zum Beispiel festgelegt wird, in welcher Tabelle Objekte einer Klasse abgelegt werden, welche Tabellenspalten den einzelnen Attributen entsprechen und mit welchen anderen Objekten sie in Beziehung stehen. Diese Mapping-Dateien ließen sich mit einem Tool des Hibernate-Frameworks aus dem Datenbankschema erzeugen und auf die durch das AXIS-Framework erzeugten Klassen zur Datenhaltung anwenden.

Zum Management der Datenbankverbindung wurde eine Hilfsklasse geschrieben, denn da das Hibernate-Framework nicht vollständig threadsicher ist, kann es beim gleichzeitigen Verwenden derselben Datenbankverbindung durch mehrere Threads, wie es vorkommt, wenn zwei Prozesse zufälligerweise gleichzeitig auf die Datenbank zugreifen möchten, zu unerwünschten Nebeneffekten und Deadlocks kommen. Die Hilfsklasse verwaltet für jeden Thread eine

eigene Verbindung zur Datenbank und sorgt dafür, dass diese nach einiger Zeit der Inaktivität automatisch geschlossen wird.

4.2.2 Druckservice

Der Druckservice ist zum Drucken der Aushänge, die Studenten über die zu vergebenen/laufenden Abschlussarbeiten informieren sollen, vorgesehen.

Drucken ist eine sehr plattformabhängige Aufgabe und daher mit dem plattformunabhängigen Java sehr schwer zu realisieren. Einige Versuche mit der Java-Printing-API führten zu kaum brauchbaren Resultaten. Aus diesem Grund wurde der Ansatz direkt aus Java drucken zu wollen verworfen und derselbe Ansatz wie bei der BPEL-Implementation verfolgt.

Die Idee dabei ist nicht Java sondern ein plattformspezifisches Programm zum Ducken zu verwenden. Um dort auf Standard-Software zurückgreifen zu können, konvertiert der Druckservice die zu druckende Arbeit in ein PDF-Dokument. Dies wird mit dem iText-Framework [iTe] realisiert. Nachdem das Dokument erstellt und in einer temporären Datei gespeichert wurde, wird das in einer Konfigurationsdatei bezeichnete Programm (z.B. ghostscript oder der Acrobat Reader) zum Drucken aufgerufen.

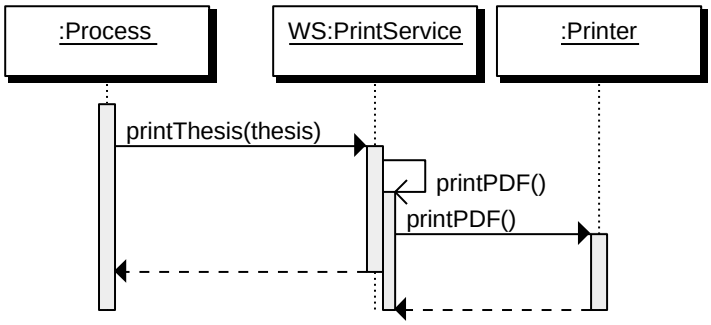

Abbildung 18: Nichtblockierendes drucken

Für diesen Webservice konnte fast der komplette Quellcode von der BPEL-Implementation genutzt werden. Es waren nur einige kleine stilistische Änderungen nötig. Außerdem wurde der Programmaufruf in einen eigenen Thread ausgelagert (siehe Abbildung 18), so dass durch das mehrere Sekunden dauernde Ausdrucken der Prozessablauf nicht verzögert wird.

Der Webservice verfügt außerdem noch über eine Operation zum reinen Generieren eines PDF-Dokuments ohne es auszudrucken. Diese Operation war in Zusammenhang mit einem Service zum Bereitstellen des Aushangs auf einem Webserver gedacht, dieser wurde aber nicht implementiert.

4.2.3 E-Mail-Service

Der E-Mail-Service ist für das Versenden von E-Mails vorgesehen. Er verwendet das Java-Mail Framework um die E-Mails an einen SMTP-Server, der in einer Konfigurationsdatei eingetragen ist, zu senden. Bei dem SMTP-Server kann es sich um einen so genannten Open-Relay-Server, der E-Mails an beliebige E-Mail-Adressen sendet (heutzutage aufgrund der hohen Spam-Belastung kaum noch zu finden), oder einen Closed-Relay-Server, der E-Mails erst nach gelungener Authentifizierung weiterleitet, handeln. Zudem kann noch die Absenderadresse der E-Mails verändert werden, falls der SMTP-Server nur E-Mails mit einer bestimmten Absenderadresse akzeptiert. In dem Fall wird der wahre Absender als Antwortadresse angegeben und auch im Textteil der E-Mail verwandt. Anhänge und E-Mails, die nicht im Standard-Textformat geschrieben sind, werden nicht unterstützt.

5 Vergleich

5.1 GQM

Die Metriken für den Vergleich wurden nach dem GQM-(Goal Question Metric [BCR94]) Ansatz hergeleitet. Dabei werden von einem zentralen Ziel (hier: „Vergleich zwischen dem BPEL und dem SOA-Me System“) mehrere Fragestellungen abgeleitet, die dann wiederum von Messungen mit den verschiedenen Metriken beantwortet werden sollen. Dieser Vorgang lässt sich als Baum darstellen, indem die Wurzel das Ziel (Goal), die internen Knoten die Fragen (Question) und die Blätter die Metriken darstellen.

Abbildung 19 stellt den GQM-Baum für den Vergleich der beiden SOA-Anwendungen dar. Dabei führt das Ziel „Vergleich zwischen dem BPEL- und dem SOA-Me-System“ zu folgenden Fragen:

1. Welches System lässt sich besser warten?
2. Auf welchem System ist das Entwickeln aufwendiger?
3. Welches System wird besser von Tools unterstützt?
4. Welches System ist flexibler?
5. Welches System lässt sich vom Endanwender besser bedienen?

Die Frage nach der Wartbarkeit (nach [IEE90] als Leichtigkeit definiert, mit der ein System geändert werden kann, um Fehler zu beheben, seine Fähigkeiten zu erhöhen oder es an eine veränderte Umgebung anzupassen) lässt sich durch die intuitive Lesbarkeit (Readability) der Kompositionen (in BPEL-Notation bzw. in Form einer EPK) und durch die Größe der Codebasis beantworten. Erstere soll dabei durch einen Versuch bestimmt werden, indem mehrere Testpersonen nacheinander beide Beschreibungen vorgelegt bekommen und nur anhand dieser einen Fragenkatalog zu den Funktionalitäten des Prozesses beantworten sollen. Die Codebasisgröße wird mit dem Eclipse-Plugin Metrics [Met] in Lines of Code (LOC) und Methoden pro Klasse und ggf. Service gemessen.

Die Frage nach dem Aufwand wird ebenfalls durch die Codebasisgröße und die Anzahl der benötigten Personenstunden beantwortet. Allerdings muss bei der Codebasisgröße berücksichtigt werden, dass nicht alle Teile gleichschwer zu implementieren sind. Teile die von Tools generiert wurden, erfordern weniger Aufwand als Teile, die handgeschrieben wurden.

Wie gut das Entwickeln durch Tools unterstützt wurde, wird durch zwei Aspekte beantwortet. Erstens durch die Auswertung meiner Erfahrungen beim Erstellen der SOA-Me-basierten Anwendung sowie der Erfahrungen der Entwickler des BPEL-Entwurfs und zweitens durch den Anteil der Codezeilen an der gesamten Codebasis, die durch Tools erzeugt wurden.

Die Flexibilität des Systems ist nach [IEE90] als „Leichtigkeit, mit der ein System abgeändert werden kann, um es in Anwendungen oder Umgebungen zu benutzen, für die es nicht entwickelt worden ist“ definiert und hängt damit auch von der Größe der Komposition des Prozesses (der Anzahl der Aktivitäten bzw. Funktionen) ab, die hier gemessen werden soll.

Usability ist in ISO 9241, Teil 11 wie folgt definiert:

„Der Grad, zu dem ein Produkt oder System durch definierte Benutzer verwendet werden kann, um spezielle Ziele, nämlich:

- Effektivität (Genauigkeit und Vollständigkeit, mit der die Benutzer bestimmte Ziele erreichen)

- Effizienz (von Benutzern aufgewendete Mittel im Verhältnis zur Zielerreichung)
- Zufriedenheit (Freiheit von Unbehagen und Beschwerden, sowie die positive Einstellung zur Nutzung des Produkts oder Systems)

in einem bestimmten Nutzungskontext zu erreichen.“

Hier soll, zum Einen durch die Zeit, die ein Benutzer durchschnittlich benötigt um den Prozess einmal zu durchlaufen (auch hier ist wieder einen Versuch mit Testpersonen vorgesehen) und zum Anderen über die Fähigkeit des SOA-Me Systems, die von dem Client des BPEL-Entwurfs benutzten Elemente der Benutzeroberfläche nachzubilden, auf die Usability geschlossen werden.

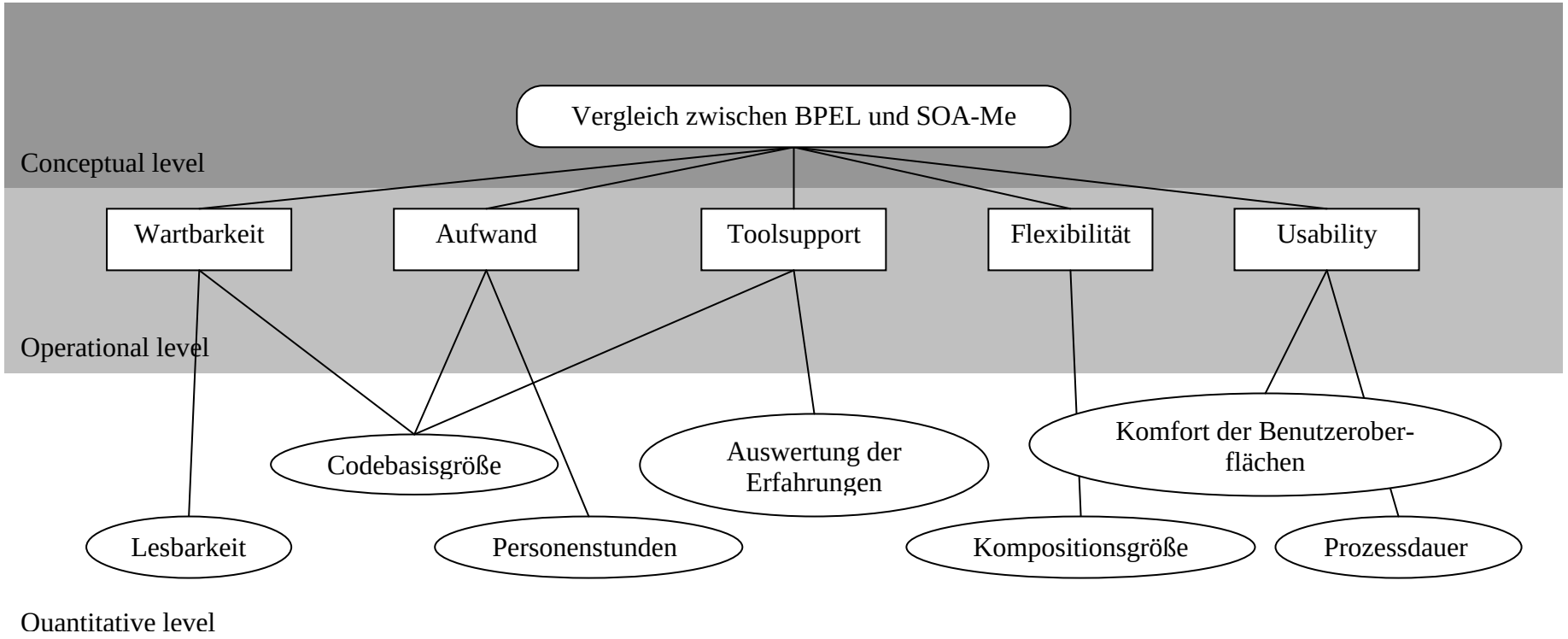

Abbildung 19: GQM-Baum für den Vergleich des SOA-Me- und BPEL-Entwurfs

5.2 Lesbarkeit

Um die Lesbarkeit der beiden Notationen zur Prozesskomposition zu vergleichen, wurden die Prozesse des BPEL-Entwurfs und die EPK des Hauptprozesses des SOA-Me-Entwurfs einigen Probanden vorgelegt. Obwohl die Anzahl der Probanden sehr klein war, konnten sie doch in zwei Gruppen unterschieden werden. Zwei der Probanden (A und B) sind Informatik-Studenten, die das SOA-Me System mit entwickelt hatten, jedoch nicht mit BPEL vertraut waren. Der dritte Proband (C) ist Mathematik-Student und besaß nur rudimentäre XML-Kenntnisse. Es war also zu erwarten, dass Proband A und B mit dem SOA-Me Editor weniger Probleme haben würden als Proband C.

Den Probanden wurde zunächst ein grober Überblick über den Allgemeinen Prozessablauf und den daran beteiligten Personen im Umfang der Abschnitte 3.1.2 und 3.1.3 gegeben. Anschließend wurde ihnen ein Fragenkatalog bestehend aus 5 Fragen ausgehändigt. Zu jeder Frage wurde eine Auswahl von möglichen Antworten gegeben, so dass die Probanden nur die richtige auswählen mussten. Die Fragen sollten sie jeweils für den Prozess in BPEL-Notation und den Prozess in EPK-Form beantworten und anschließend bewerten, wie schwer ihnen diese Antwort auf einer Skala von 1 (sehr leicht) bis 5 (sehr schwer) gefallen ist.

Die Fragen wurden dabei so gestellt, dass sie sich auf bestimmte Modellierungsmöglichkeiten bezogen. Frage 1 und 2 betrafen die Ausführungsreihenfolge von bestimmten Aktivitäten. Bei Frage 1 mussten die Probanden erkennen, dass bestimmte Prozessteile gleichzeitig ablaufen können, bei Frage 2 ging es darum die Abhängigkeiten zwischen zwei Aktivitäten zu erfassen. Die beiden ersten Fragen konnten noch gelöst werden, indem sich die Probanden nur die grafische Notation ansahen, für die folgenden Fragen war eine Analyse von verschiedenen Ausdrücken nötig.

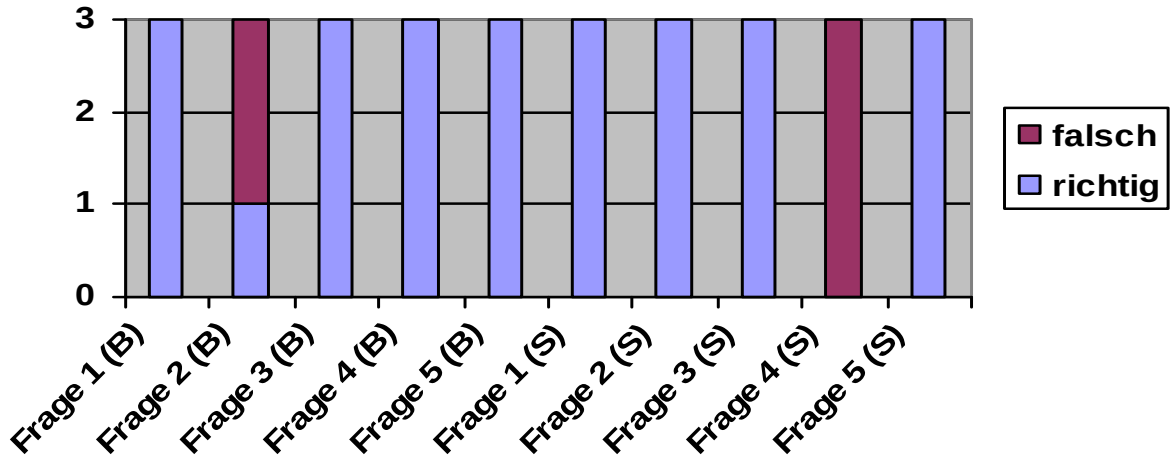

Abbildung 20: Korrektheit der Antworten (B = BPEL, S= SOA-Me)

Frage 3 und 4 handelten von Fallunterscheidungen. Um Frage 3 zu beantworten musste in der BPEL-Notation die entsprechende Fallunterscheidung gefunden und eine einfache boolesche Bedingung, die eine Variable mit einer Konstanten verglich, verstanden werden. Bei SOA-Me gab es eine entsprechende Fallunterscheidung nicht, was die Pobanden erkennen mussten. Frage 4 drehte sich um die Plausibilitätsprüfung von Benutzerdaten. Die entsprechende Fallunterscheidung verglich dazu zwei Variable. Diese Fallunterscheidung war im SOA-Me EPK zweigeteilt. Sie wurde in einem Stylesheet berechnet und führte erst in einem späteren XOR-Konnektor zu einer Verzweigung.

Die 5. Frage handelte von einer Implementierungsentscheidung. Die Probanden sollten herausfinden, welchen Wert eine Konstante mit einer gegebenen Bedeutung hatte. Dafür musste in BPEL die entsprechende Assign-Anweisung und in SOA-Me der entsprechende Stylesheet gefunden werden. Dies war besonders bei BPEL schwierig, da die entsprechende Stelle nicht kommentiert war.

Für die Bearbeitung des Fragebogens benötigten die Probanden je Notation ca. 30 min.

	Richtige Antworten		**Schwere (1-5)**	
	BPEL	*SOA-Me*	*BPEL*	*SOA-Me*
Probanden A und B	90,00%	80,00%	4,10	3,60
Proband C	80,00%	80,00%	4,40	3,60
Durchschnitt	86,67%	80,00%	4,20	3,60

Tabelle 2: Ergebnisse der Lesbarkeitsanalyse (Schwere: 1=sehr leicht, 5=sehr schwer)

Nach der Auswertung der Fragebögen zeigte sich erst einmal eine Überraschung: Obwohl die Probanden A und B Erfahrungen mit der SOA-Me Plattform hatten, machten sie, wie in Tabelle 2 zu sehen, genauso viele Fehler bei der Beantwortung der Fragen zur SOA-Me-Plattform und bewerteten die Schwere der Fragen im Schnitt genauso wie Proband C. Mit der BPEL-Notation kamen A und B besser zu Recht als C und bewerteten diese auch leicht besser, diese Abweichung ist aber sehr knapp, da nur eine richtige Antwort und 0,3 Einheiten „auf der Schwere-Skala“ zwischen beiden Gruppen liegt.

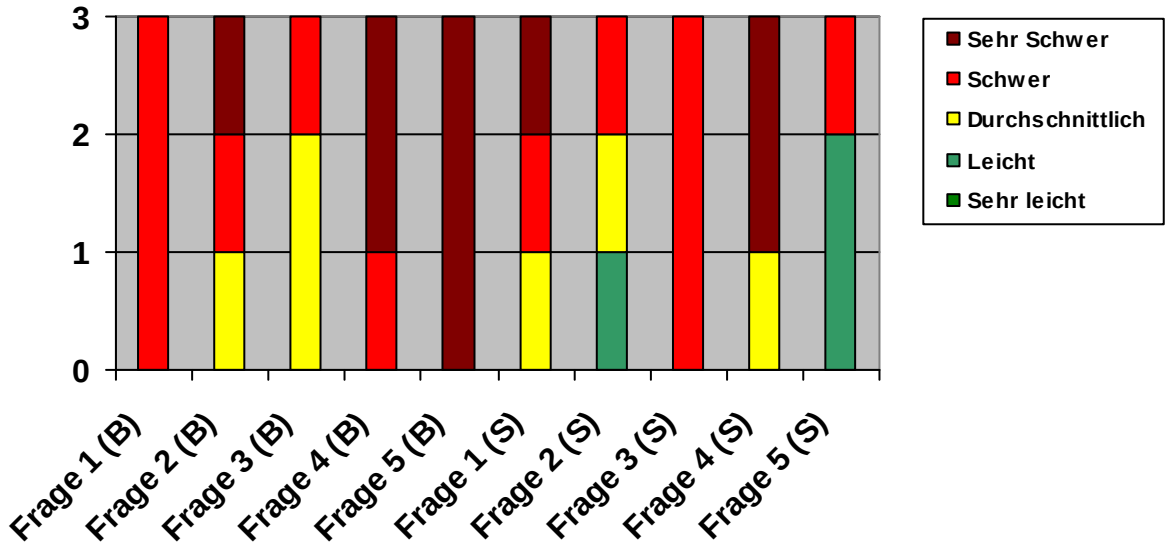

Abbildung 21: Bewertung der Schwierigkeit der Fragen (B = BPEL, S= SOA-Me)

Insgesamt ergibt sich, dass die SOA-Me-EPK etwas leichter empfunden wird als die BPEL-Notation, aber bei SOA-Me auch ein Fehler mehr gemacht wurde als bei BPEL. Wie in Abbildung 20 zu sehen wurde, bezogen auf das SOA-Me System, nur Frage 4 falsch beantwortet. Die Aufteilung der Bedingung auf ein Stylesheet und einen XOR-Konnektor war wohl für die Probanden nicht zu erkennen. Das zwei von drei Probanden bei BPEL die Frage 2 falsch beantwortet haben lies sich nicht nachvollziehen. Die entsprechende Stelle in der BPEL-Notation war übersichtlicher als bei Frage 3. Auffällig ist hier auch, dass Frage 5 und nicht Frage 2 als schwierigste Frage empfunden wurde.

Sowohl bei SOA-Me als auch bei BPEL wurden die falsch beantworteten Fragen von den Probanden als schwer bis sehr schwer bewertet (siehe Abbildung 21). Bei SOA-Me wurde sie sogar als schwerste Frage klassifiziert, bei BPEL nimmt diesen Platz die Frage 5 ein. Bei dieser Frage war die entsprechende Stelle in der BPEL-Notation sehr schwer zu bestimmen. Ein Proband fand sie nur dadurch, indem er von der grafischen Visualisierung auf die Text-Darstellung umschaltete und in dieser suchte.

Im anschließenden Gespräch mit den Probanden waren diese der Meinung, die SOA-Me EPK sei übersichtlicher gewesen als die BPEL-Notation, wenngleich die explizite Persistenz und die Fehlerbehandlung zu Verwirrung geführt hätte. Die BPEL-Notation sei sehr aufgebläht, durch die großflächigen Scopes hätte man sehr viel vertikal scrollen müssen, was es schwer gemacht hätte sich einen Überblick über den Prozess zu verschaffen.

5.3 Analyse der Codebasis

Die Analyse der Codebasis sollte mehrere Fragen beantworten: Zuerst die Frage nach dem Aufwand, nach der Wartbarkeit und nach dem Toolsupport. Dabei werden zwei verschiedene Codequellen betrachtet: Einmal die Javaquellen der Webservices und des BPEL-Clients und einmal die XSLT-Stylesheets, die die Aufgaben und die Persistenz des SOA-Me Entwurfes abdecken.

Zur Messung der Größe der Javaquellen (Webservices und BPEL-Client) wird die Total Lines Of Code Metrik (LOC) benutzt. Nach der Definition von [Met] zählt diese alle Zeilen, die nicht leer sind und nicht nur Kommentare enthalten. Diese Anzahl wird noch manuell auf zwei Aspekte aufgeteilt: Code, der von Hand geschrieben werden musste (manuelle LOC) und Code, der von einem Hilfsprogramm erstellt wurde (generierte LOC). Dies ist für die Frage nach dem Aufwand und nach dem Toolsupport nötig. Des Weiteren werden bei den Javaquellen noch die Anzahl der Klassen und die Anzahl der (statischen und nicht statischen) Methoden gezählt. Gemessen wurde mit dem Metrics Plugin für Eclipse, das ebenfalls auf [Met] angeboten wird.

Zur Messung der Größe der Stylesheets wurde das GNU Tool wc benutzt, dass einfach nur die Anzahl der Zeilen zählt. Wie bei Java gibt es eine Unterteilung in manuelle und generierte Zeilen, allerdings entstanden die generierten Zeilen nicht durch ein Hilfsprogramm, sondern durch das Kopieren bereits vorhandener Teile aus andern Stylesheets. Damit kann dieser Wert nicht zum Toolsupport beitragen. Statt der Klassen wird bei den Stylesheets die Anzahl der Stylesheets angegeben. Etwas Vergleichbares zu Methoden gibt es bei Stylesheets nicht, so dass in dieser Kategorie nichts gemessen werden konnte.

Komponente	**LOC**				**Klassen**	**Methoden**	
	generiert	**manuell**	**total**		**total**	**total**	**pro Klasse**
Datenbank WS	1221	394	1615	75,60%	7	147	21,00
PrintService	0	314	314	0,00%	3	7	2,33
MailService	0	309	309	0,00%	4	38	9,50
WATManager	667	66	733	91,00%	9	76	8,44
TodoManager	187	266	453	41,28%	3	24	8,00
ExperienceForum	170	251	421	40,38%	6	46	7,67
AuthService	0	434	434	0,00%	8	15	1,88
Client	5544	11496	17040	32,54%	222	1592	28,785315

Tabelle 3: Codebasisanalyse der BPEL-Komponenten

Komponente	TLOC				Klassen	Methoden	
	generiert	manuell	total		total	total	pro Klasse
Datenbank WS	1172	293	1465	80,00%	8	132	16,50
PrintService	5	335	340	1,47%	3	7	2,33
MailService	159	75	234	67,95%	2	18	9,00
WATManager	0	4	4	0,00%	0	0	-
Client	205	579	784	26,15%	18	Stylesheets	
Persistenz	1408	600	2008	70,12%	19		

Tabelle 4: Codebasisanalyse der SOA-Me-Komponenten

Tabelle 3 und Tabelle 4 zeigen die Ergebnisse der Messungen im Einzelnen.

Insgesamt ergibt sich, dass die SOA-Me-Codebasis mit 4.835 Codezeilen in 13 Klassen bzw. 37 Stylesheets wesentlich kleiner ist als die BPEL-Codebasis mit 21.319 Codezeilen in 262 Klassen. Die größte Komponente ist bei BPEL der Client mit allein 17.040 Codezeilen in 222

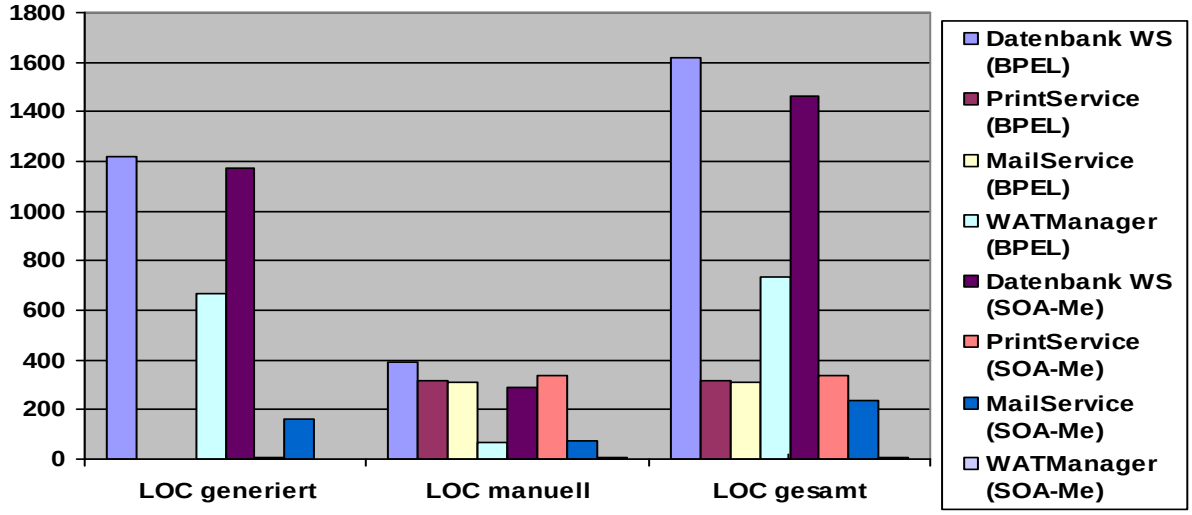

Abbildung 22 LOC Webservices

Klassen. Damit ist der Client fast viermal so groß wie der Rest der BPEL-Codebasis. Die größte Komponente des SOA-Me-Entwurfs stellt die Persistenzverwaltung mit 2.008 Codezeilen in 19 Stylesheets da.

Gut gegenüberstellen lassen sich die Webservices beider Entwürfe (siehe Abbildung 23). Es fällt auf, dass die Größe der einzelnen Webservices ungefähr gleich ist. In den meisten Fällen sind die Webservices des SOA-Me-Entwurfs etwas schlanker.

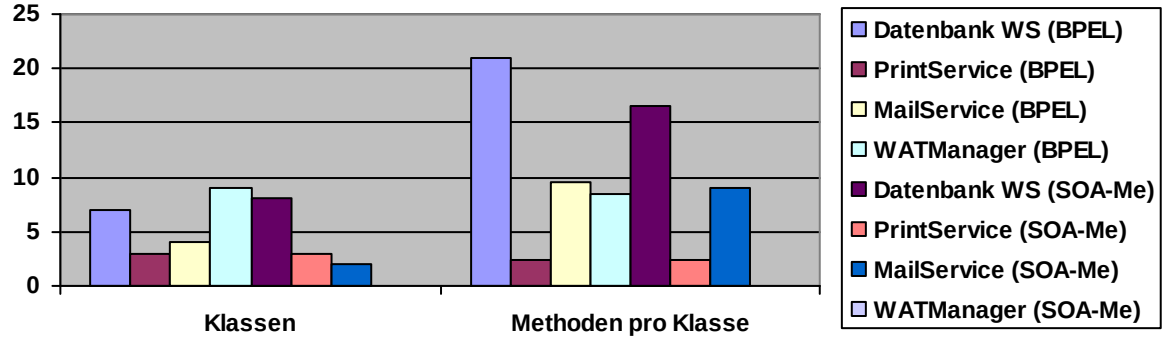

Abbildung 23: Methoden und Klassen der Webservices

Der höchste Größenunterschied tritt beim Webservice für das WAT-System des Fachgebiet Software Engineering auf (733 zu 4 Zeilen). Der BPEL-Entwurf sieht hier einen Proxy-Service vor, der die eingehenden Aufrufe einfach an den schon bestehenden Webservice für das WAT-System weitereicht. Das Front- und das Backend dieses Proxys wurden dabei komplett mit dem Axis-Framework generiert, so dass nur 9% des Codes selbst geschrieben werden musste. Bei SOA-Me hingegen war so ein Proxy nicht nötig. Der SOA-Me Server war nach dem Hinzufügen von vier Zeilen Javacode in der Lage den gegebenen Service direkt aufzurufen.

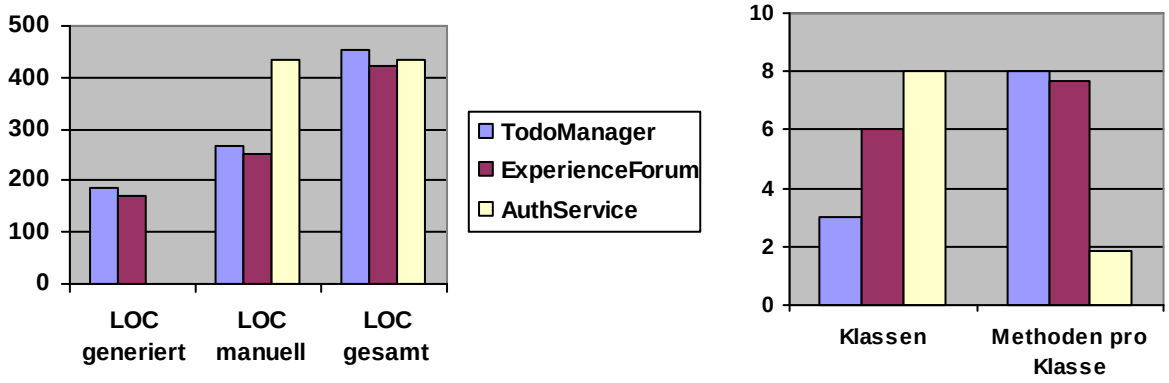

Abbildung 24: BPEL eigene Webservices

Auch bei den Webservices, die als Datenbankinterface dienen, ist das Verhältnis von generiertem zu manuell geschriebenem Code sehr hoch. Dies liegt vor allem an den großen Klassen, die zum Halten der Daten für die Entitäten der Datenbank dienen. Diese Klassen können ebenfalls vom Axis-Framework generiert werden. Allerdings wirkt sich dieses Feature des Axis-Frameworks augenscheinlich auf die Wartbarkeit aus. Die Anzahl der Methoden pro Klasse sind bei diesen Webservices sehr hoch.

Bei den Webservices zum Drucken und Verschicken von E-Mails zeigten sich die geringsten Unterschiede zwischen den beiden Entwürfen. Vor allem beim Drucken überrascht das nicht, konnte doch bei der Implementierung der SOA-Me-Anwendung viel von BPEL übernommen werden. Bei der E-Mail-Service Implementierung des BPEL-Entwurfes überrascht, dass die Klasse, die die Daten der E-Mail hält, nicht von Axis generiert, sondern augenscheinlich handgeschrieben wurde.

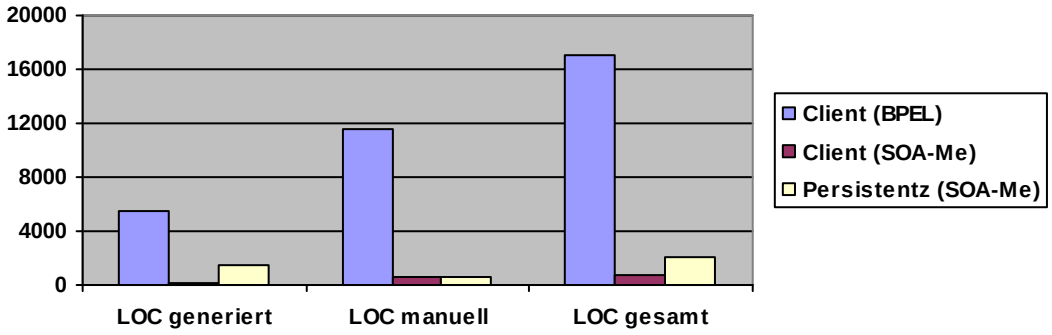

Abbildung 25: LOC Client und Persistenz

Bei den Webservices, die Funktionalitäten anbieten, die in der SOA-Me-Plattform schon implementiert sind und daher nur beim BPEL-Entwurf zu finden sind, fällt auf, dass der Anteil von generiertem Code relativ klein ist (siehe Abbildung 24). Der Webservice zur Authentifi-

zierung ist sogar komplett handgeschrieben. Auffällig ist auch, dass alle diese Webservices trotz ihrer verschiedenen Funktionen fast gleich groß sind, aber große Unterschiede, was die Anzahl der Klassen angeht, aufweisen.

Der größte Unterschied beim Vergleich der Codebasen der beiden Entwürfe fällt beim Client auf. Wie in Abbildung 25 zu sehen, stehen die Anzahl der Codezeilen, die zum Erzeugen der Benutzeroberfläche dienen, in keinem gesunden Verhältnis zueinander. Dadurch dass für BPEL eine komplette grafische Anwendung geschrieben werden muss, ist die Codebasis 22-mal größer als die von SOA-Me. Da fällt auch nicht ins Gewicht, dass es sich bei SOA-Me nicht um Java-Code sondern um XSL-Stylesheets handelt. Das Konzept mit den zur Laufzeit generierten Benutzeroberflächen scheint an dieser Stelle seinen größten Vorteil zu haben. Selbst der Mehraufwand für die explizite Persistenz wird mehr als ausgeglichen. Auffällig ist jedoch dass der Anteil von generiertem Code mit 33% beim BPEL-Client deutlich höher liegt als der Anteil von wieder verwendetem Code bei den entsprechenden SOA-Me Stylesheets. Aber ähnlich wie bei den Webservices führt dies auch zu einer extrem hohen Anzahl von Methoden pro Klasse. Diese liegt beim Client über 28.

5.4 Kompositionsgröße

Um Rückschlüsse auf die Flexibilität der beiden Entwürfe ziehen zu können, soll in diesem Abschnitt die Größe der beiden Kompositionen ermittelt werden. Dazu werden im Fall von BPEL die Anzahl der verwendeten strukturierenden und Basis-Aktivitäten, im Fall von SOA-Me die Anzahl der Ereignisse, Funktionen und Konnektoren bestimmt. Der direkte Vergleich dieser Zahlen wird jedoch nicht sehr aussagekräftig sein, da beide Kompositionsnotationen doch sehr unterschiedlich sind, doch beginnen wir zunächst mit der Erhebung der Daten.

Dies wurde für den BPEL-Entwurf im Rahmen der Masterarbeit von Alex Salnikow [Sal07] schon getan. Dabei ergab sich eine Gesamtzahl von 1258 Aktivitäten: 750 Basisaktivitäten und 508 strukturierende Aktivitäten, die sich wie folgt verteilen:

Strukturierende Aktivitäten		**Basisaktivitäten**	
Aktivität	**Anzahl**	**Aktivität**	**Anzahl**
flow	237	assign	387
pick	1	empty	109
switch	71	recive	21
while	19	terminate	14
sequnece	0	wait	10
scope	180	invoke	56
		reply	55
		throw	98

Tabelle 5: Aktivitäten des BPEL-Prozesses

Auffällig ist, dass der BPEL-Entwurf keine Sequenzen verwendet. Stattdessen werden 596 Synchronisationslinks innerhalb von Flow-Aktivitäten verwendet. Dies kommt aber der Vergleichbarkeit mit der SOA-Me EPK zugute, da hier die Sequenz die Standardverknüpfung zweier Elemente ist.

Die EPKs des Haupt- und Hilfsprozesses aus dem SOA-Me Entwurfs wurde mit EPCMetrics, eine Erweiterung der EPCTools die in [GLM06] vorgestellt wurde, analysiert. Dabei wurden insgesamt 236 Elemente gezählt. Davon sind 55 Funktionen, 110 Events, 46 Verzweigungen (Splits) und 25 Zusammenführungen (Joins).

Im Detail:

Konnektoren		Funktionen	
Element	**Anzahl**	**Element**	**Anzahl**
And-Split	5 (1/0)	Webservice-Aufruf	34 (18/0)
And-Join	4 (0/0)	Benutzerinteraktion	21 (1/1)
Xor-Split	39 (28/24)	**Ereignisse**	
Xor-Join	21 (14/1)	Startereignis	2 (0/0)
Or-Split	2 (0/2)	Zwischenereignis	102 (33/28)
Or-Join	0 (0/0)	Endereignis	6 (0/1)

Tabelle 6: Elemente der EPK des SOA-Me-Entwurfs
Durch Persistenz/Fehlerbehandlung bedingte Anzahlen in Klammern

Außerdem enthalten die EPKs 292 Kanten. Die erhöhte Zahl der XOR-Konnektoren resultiert aus der expliziten Persistenz, die für jeden Prozessabschnitt einen Einsprungspunkt (XOR-Join) benötigt, und aus der Fehlerbehandlung bei Webserviceaufrufen, bei der nach jedem Webserviceaufruf entscheiden werden muss, ob dieser geglückt ist oder nicht. (XOR-Splits, die nach einem der Persistenz dienenden Webserviceaufruf zur Fehlerbehandlung benötigt werden, zählen sowohl zur Fehlerbehandlung als auch zur Persistenz)

Aus der Gesamtzahl der Elemente/Aktivitäten scheint zu folgen, dass die SOA-Me Komposition nur etwa 19% der BPEL-Komposition ausmachen. Es zeigt sich aber, dass nicht jede BPEL-Aktivität ihre Entsprechung im SOA-Me EPK findet. So sind pick, scope, empty und wait Aktivitäten, die in SOA-Me nicht vorgesehen sind. Auch etwas wie throw gibt es nur implizit, etwa bei einem Webserviceaufruf. Start- und Zwischenereignisse sind BPEL unbekannt. Aus diesem Grund werden die genannten Aktivitäten und Elemente im Folgenden nicht berücksichtigt.

Die restlichen Aktivitäten lassen sich aber gegenüberstellen. AND-Konnektoren entsprechen der Flow-Aktivität, OR- und XOR-Konnektoren entsprechen zusammengenommen den switch und while Aktivitäten. Webservice-Aufrufe entsprechen exakt invokes und da die receive-Aktivitäten im BPEL-Entwurf hauptsächlich zur Kommunikation mit dem Client benutzt werden, können diese der Benutzerinteraktion gegenübergestellt werden. Terminate-Aktivitäten entsprechen den Endereignissen. Assigns nehmen eine Sonderposition ein. Sie entsprechen nicht einem Element in der EPK, sondern den XSL-Stylesheets die jeder Funktion zugeordnet sind. Aus diesem Grund werden die Anzahl der Assigns mit der Anzahl der in der EPK eingebundenen Stylesheets verglichen.

Dadurch lässt sich folgende Gegenüberstellung ableiten:

BPEL-Aktivitäten	**Anzahl**	**EPK Element**	**Anzahl**
flow + symclinks (parallel)	8	AND	9
switch + while	90	XOR+OR	62
invoke	56	Webserviceaufruf	34
recive + reply	76	Benutzerinteraktionen	21
terminate	14	Endereignisse	6
assign	387	Stylesheets	162
flow + synclinks (Sequenz)	821	Kanten	292

Tabelle 7: Gegenüberstellung der BPEL und SOA-Me Kompositions-Elemente

Diese Gegenüberstellung zeigt die Verhältnisse sehr genau. SOA-Me kommt mit 69% der bedingten Verzweigungen, 61% der Webserviceaufrufe, 28% der Benutzeraktionen und 42% der Zuweisungen aus und besitzt nur 43% der Prozessenden. Nur das Verhältnis von parallelen Abläufen fällt mit 112% zu ungunsten von SOA-Me aus. Allerdings wird in der SOA-Me Implementierung reger gebrauch von parallelen Abläufen gemacht, wenn es darum geht zwei

unabhängige Webservices auszuführen. Bei den Sequenzen benötigt SOA-Me 36% der BPEL-Elemente. Hier hätte das Verwenden der sequence-Aktivität zu einer deutlich kompakteren Komposition führen können. Damit ist die SOA-Me-Komposition (trotz expliziter Persistenz) wesentlich kompakter als die BPEL-Komposition. Aus Tabelle 7 folgt, das bei ausschließlicher Betrachtung der Teile der Kompositionen, die sich funktionell vergleichen lassen, die Größe der SOA-Me-Komposition nur 45% der BPEL-Komposition beträgt.

5.5 Benutzeroberflächen

Um zu bestimmen, inwieweit die Clients der beiden Plattformen den Benutzer bei seinen Aufgaben unterstützen, soll in diesem Abschnitt ermittelt werden, welcher der beiden Clients die besseren Konzepte zu diesem Zweck anbietet. Da der BPEL-Standard keinen interaktiven Client vorsieht bzw. das Entwickeln eines Clients nicht unterstützt, ist der Komplexität der Benutzeroberfläche für BPEL-Prozesse, im Gegensatz zur SOA-Me Plattform, keine Grenze gesetzt. Die SOA-Me Plattform hingegen benutzt ein festes Task-Model, das die zur Verfügung stehenden UI-Elemente begrenzt.

Aus diesem Grund ist der SOA-Me Client, was die Menge der UI-Konzepte angeht, im Allgemeinen einem Client für einen BPEL-Prozess unterlegen. Aus diesem Grund wird hier der universelle SOA-Me Client mit dem speziellen BPEL-Client für den Abschlussarbeits-Prozess verglichen. Der BPEL-Client wird als Referenz für die Frage herangezogen, ob der SOA-Me Client alle relevanten UI-Konzepte abdeckt.

Der Vergleich wurde wie folgt durchgeführt: Zunächst wurde analysiert, welche Konzepte im BPEL-Client Anwendung finden, danach die entsprechenden Elemente des SOA-Me Task-Modells (sofern vorhanden) gegenübergestellt.

Der Client aus dem BPEL-Entwurf besteht aus zwei Teilen, einem Informationsteil, der Informationen zu den laufenden und abgeschlossenen Arbeiten sowie den anstehenden Aufgaben anzeigt, und einen auf Assistenten gestützten Teil, der durch den Prozess führt.

Im Informationsteil werden folgende Konzepte benutzt:

Registerkarten dienen dazu, Informationen und Eingabefelder eines Programmfensters auf mehreren, hintereinander liegenden Ebenen (Karten) anzuordnen. Dabei befindet sich immer eine Karte im Vordergrund, von den dahinter liegenden Karten ist nur der Reiter zu sehen (siehe Abbildung 26). Durch Anklicken des entsprechenden Reiters kann man eine andere Registerkarte in den Vordergrund holen, wobei die Informationen und ggf. getätigten Einstellungen auf den vorher genutzten Registerkarten bestehen bleiben. Der BPEL-Client verwendet dieses Konzept nur für die Präsentation von Informationen über laufende Arbeiten und Themen.

Tabellen werden vom BPEL-Client zum Anzeigen aller aktuellen Arbeiten und Themen verwendet (siehe Abbildung 26).

Für die Abarbeitung des Prozesses werden folgende Konzepte benutzt:

Assistent (Wizard; 11x vorhanden): Der BPEL-Client stellt für fast alle vom Benutzer auszuführenden Aufgaben einen Assistenten bereit. Dabei werden dem Benutzer die zu der Aufgabe gehörenden UI-Elemente Schritt für Schritt, nach entsprechenden Gesichtspunkten gegliedert, präsentiert. Damit kann für jeden Teilschritt individuelle Hilfestellungen gegeben werden. Die im Folgenden beschriebenen Elemente und Konzepte finden alle innerhalb dieser Assistenten statt.

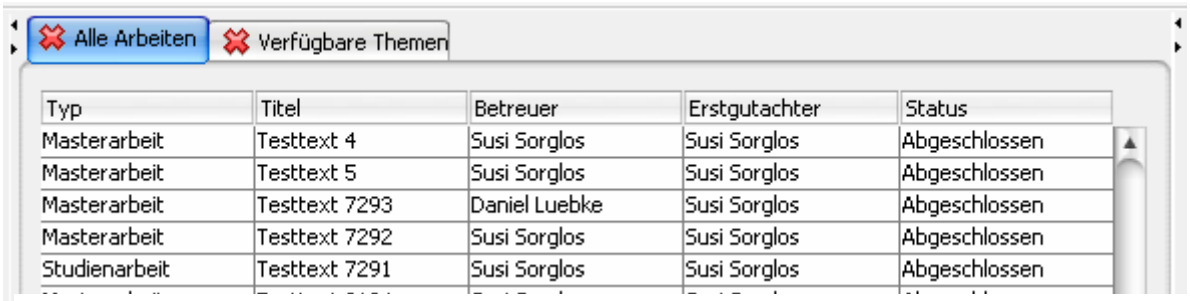

Abbildung 26: Informationsteil des BPEL-Clients

Radiobuttons (4x) stellen ein Standardelement moderner Benutzeroberflächen dar. Ein Radiobutton kann zwei Zustände annehmen: selektiert und unselektiert. Radiobuttons werden normalerweise (und auch im BPEL-Client) für die Auswahl einer Option eingesetzt. Zu diesem Zweck werden sie zu Gruppen zusammengefasst, aus der immer nur ein Radiobutton selektiert werden kann.

Checkboxen (12x) stellen ebenfalls ein Standardelement moderner Benutzeroberflächen dar und ähneln auch sonst den Radiobuttons. Genau wie diese können sie selektiert oder unselektiert sein. Allerdings arbeiten Checkboxen normalerweise unabhängig voneinander und werden daher für ja/nein Optionen eingesetzt.

Drop-Down Listbox (8x): Eine einfache Auswahlliste, die zunächst einzeilig dargestellt wird und erst bei bedarf ausgeklappt wird.

Button (4x): Einfache Schaltfläche. Hier sind aber nicht die standardmäßigen OK/Abbrechen Buttons gemeint, wie sie zum Bestätigen einzelner Aufgabenschritte benutzt werden, sondern explizit gesetzte Buttons, die eine spezielle Bedeutung für die Aufgabe haben. Im BPEL-Client wird so ein spezieller Button für die Genehmigung von Themen durch den Erstprüfer verwandet. Das Thema ist genehmigt, wenn der Button betätigt wurde. Außerdem werden Buttons zum Ein- und Ausblenden weiterer Optionen verwendet.

Textfelder (36x) sind einfache Textfelder, in die der Benutzer Daten eingeben kann.

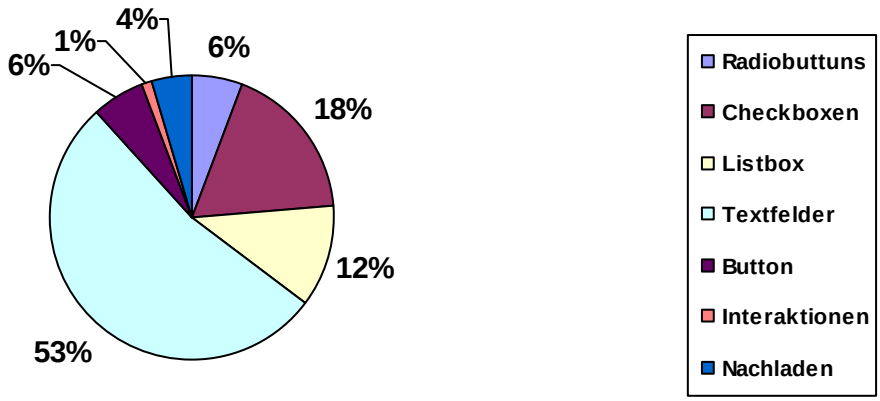

Abbildung 27: Prozentualer Anteil der einzelnen UI-Elemente am Client des BPEL-Entwurfs

Interagierende Elemente (1x): Im BPEL-Client können UI-Elemente miteinander interagieren. So ist es möglich, dass eine falsche Eingabe in einem Textfeld dazu führt, dass der

aktuelle Schritt nicht beendet werden kann (der entsprechende Button wird ausgegraut). Auch können so Elemente intelligent verbunden werden, so dass ein Textfeld erst dann Eingaben akzeptiert, wenn z.B. ein entsprechender Radio-Button selektiert wurde.

Interaktives Nachladen von Daten (3x): Der BPEL Client ist in der Lage einzelne UI-Elemente mit Werten, die z.B. aus einer Datenbank stammen, zu füllen. Dies geschieht interaktiv zur Laufzeit des Clients, so dass sich Änderungen an dem Datenstamm sofort im Client auswirken. Dieses Konzept wird konkret bei der Angabe von Personendaten im Prozess benutzt

Das SOA-Me Task-Modell deckt die wichtigsten dieser Konzepte ab. Radiobuttons und Checkboxen können durch eine single bzw. multiple Selection, Textfelder durch ein Edit und Drop-Down-Listboxes durch ein Control modelliert werden. Damit werden schon 89% der im BPEL-Client vorkommenden UI-Elemente abgedeckt (siehe Abbildung 27).

Tabellen sind im Task-Modell nicht explizit vorgesehen, können aber durch ein Display dargestellt werden, da Display-Elemente HTML (Hypertext Markup Language) enthalten können. Über den Umweg HTML sind also auch Tabellen in SOA-Me möglich. Allerdings sehen diese eben wie typische HTML-Tabellen aus und fügen sich damit nicht so gut in das Look and Feel des SOA-Me Clients ein.

Assistenten sind dem SOA-Me Client überhaupt nicht bekannt. Eine Aufgabe kann nur als Ganzes abgearbeitet und nicht in einzelne Schritte zerteilt werden. Diese könnte aber nützlich sein, wenn eine Aufgabe aus vielen einzelnen UI-Elementen besteht, die nicht alle auf den Bildschirm passen.

Buttons für spezielle Aktionen kennt der SOA-Me Client ebenfalls nicht. Allerdings darf die Sinnhaftigkeit dieses Konzeptes angezweifelt werden. Die entsprechende Verwendung im BPEL-Client (Genehmigen eines Themas) hätte man ebenso mit einer Checkbox darstellen können. Zudem scheint die Verwendung eines Buttons in diesem Bereich fehleranfällig zu sein: Während den Untersuchungen zur Prozessdauer fiel den Probanden auf, dass ein Ablehnen des Themas unmöglich ist. Der entsprechende Assistent ließ sich nur fortsetzen, wenn man das Thema genehmigte.

Das Task-Modell sieht außerdem keine miteinander interagierenden UI-Elemente vor. Die einzige Funktionalität, die dem nahe kommt, ist die Möglichkeit leere Textfelder zu verbieten. In dem Fall lässt der Client eine Bestätigung der aktuellen Aufgabe nicht zu. Zumindest die genauere Überprüfung des Inhaltes von Textfeldern (z.B. durch reguläre Ausdrücke) ist aber eine wichtige Funktionalität. So ist es in der SOA-Me-Anwendung durch eine falsch formatierte Eingabe möglich den Prozess, ohne Meldung an den Client, fehlschlagen zu lassen.

Das interaktive Nachladen von Daten durch den Client ist eine sehr interessante und ebenfalls im SOA-Me Client nicht vorhandene Funktionalität. Zwar ist es auch mit SOA-Me möglich UI-Elemente mit Daten aus einer dynamischen Datenbasis (z.B. einer Datenbank) zu belegen, dies kann allerdings nur der SOA-Me Server tun. Dies führt dazu, dass die Daten unter Umständen nicht mehr aktuell sind, wenn der Client sie anzeigt. Denn die Aufgabenbeschreibung, in der diese Daten enthalten sind, kann sich, nachdem sie vom Server publiziert wurde, nicht mehr ändern.

5.6 Prozessdauer

Um die Prozessdauer zu bestimmen wurde eine kleine Gruppe von Probanden gebeten, einmal am SOA-Me Client und einmal am BPEL-Client eine typische Abschlussarbeit zu verwalten. Den Probanden wurde zu diesem Zweck eine genaue Schritt-für-Schritt-Anleitung gegeben (siehe Anhang Seite 57), die genau festschrieb, welche Daten einzugeben waren. Über die genaueren Funktionen der beiden Clients wurden die Probanden nicht informiert.

Die Daten der Abschlussarbeit waren so gewählt, dass möglichst alle Aktivitäten, die in 3.1.3 spezifiziert sind, vorkamen. Es handelte sich um eine interne Bachelorarbeit (da der BPEL-Entwurf keine externeren Arbeiten unterstützt) deren Erstprüfer gleichzeitig Betreuer war, die zunächst vom Erstprüfer abgelehnt werden sollte und deren Abschlusspräsentation zunächst vor dem Abgabetermin stehen sollte. Die Probanden sollten dabei nicht nur als Betreuer sondern auch als Erstprüfer und Sekretär auftreten, was ein doppeltes Einloggen (einmal als Betreuer/Erstprüfer und einmal als Sekretär) nötig machte.

Es wurde die Zeit vom Anlegen der Arbeit bis zur Anzeige der Zusammenfassung der Bewertungen gestoppt. Die Personen A-C arbeiteten erst mit dem SOA-Me System und danach mit der BPEL-basierten Anwendung, E-F umgekehrt. Folgende Zeiten wurden gemessen:

	BPEL	**SOA-Me**	**SOA-Me/BPEL**
Person A	00:15:00*	00:12:12	81,33%
Person B	00:14:00*	00:14:03	100,36%
Person C	00:22:20	00:13:50	61,94%
Person D	00:21:18	00:17:01	79,89%
Person E	00:25:31	00:21:22	83,74%
Person F	00:29:44	00:24:29	82,34%
Durchschnitt	00:21:19	00:17:09	80,50%

Tabelle 8: Prozessdauer

Mit * markierte Zeiten wurden nur minutengenau erhoben. Es zeigte sich, dass der SOA-Me-Prozess im Durchschnitt 4 Minuten und 10 Sekunden schneller zu durchlaufen war als der BPEL-Prozess, damit ließ sich die Aufgabe mit SOA-Me in nur 81% der Zeit erfüllen. Ein überraschendes Ergebnis, wo doch BPEL aufgrund der breiteren Unterstützung von UI-Konzepten auch effizienter zu bedienen sein sollte.

Diese Konzepte werden aber von dem BPEL-Client anscheinend nicht optimal eingesetzt. So erfolgt z.B. die Prüfung auf richtigen Inhalt der Textfelder ohne entsprechenden Hinweis an den Benutzer. Der entsprechende Assistent verweigert bei falschen Eingaben einfach das Fortsetzen, ohne dass der Benutzer Anhaltspunkte dafür erhält, was er falsch gemacht hat. Die Assistenten hemmen zudem das Vorankommen, wenn der Benutzer mit dem Prozess vertraut ist, was hier durch das vorgelegte Szenario simuliert wurde. Denn „das ständige ‚weiter'-Klicken" in den Assistenten, so ein Proband, koste mehr Zeit, als wenn die entsprechenden Eingabefelder, wie bei SOA-Me untereinander angeordnet, durch Scrollen zu erreichen wären.

5.7 Fazit

Im Folgenden sollen die oben vorgestellten Messungen zusammengefasst und auf die nach GQM gebildeten Fragestellungen bezogen und bewertet werden.

5.7.1 Aufwand

Schon durch Betrachtung der nötigen Personenstunden für Entwurf und Entwicklung der beiden SOA-Anwendungen lässt sich eine Tendenz erkennen. An dem mit sechs ECTS-Punkten vergüteten Projekt zur Entwicklung der Anwendung auf BPEL-Basis nahmen 10 Personen teil. Ein ECTS-Punkt entspricht einem Aufwand von zweidrittel Semesterwochenstunden. Da das Projekt 16 Wochen dauerte ergibt sich so ein Wert von 640 Personenstunden. Allerdings gaben die Entwickler in ihrem Erfahrungsbereicht an, länger als 4 Stunden die Woche mit dem Projekt beschäftigt gewesen zu sein.

Im Fall von SOA-Me konnten die Entwicklungszeit exakt gemessen werden. So war ich mit dem grundlegenden Entwurf der EPKs 2 Tage, mit der Konzeption des Prozessdokumentes 1,5 Tage, mit den Datenbankservices 3 Tage und mit dem Entwickeln der EPKs 10 Tage beschäftigt. Die Zusatzservices erforderten 3 Tage. Bei einem Arbeitstag von ca. 8 Stunden ergibt sich ein Aufwand von 156 Personenstunden.

Allerdings muss bei der Betrachtung der Personenstunden auch berücksichtigt werden, dass bei dem Entwurf der SOA-Me-Anwendung viel vom Entwurf der BPEL-Anwendung übernommen werden konnte.

Die Auswertung der Codebasis bestätigt die Tendenz, dass die Entwicklung auf SOA-Me weniger aufwendig war als bei BPEL: Für SOA-Me mussten 1.886 Codezeilen handgeschrieben werden, für BPEL 13.530. Wobei sich hier der Client am stärksten bemerkbar machte, die Entwicklung der Webservices war für beide Systeme in etwa gleichaufwendig.

Auch bei den Kompositionen musste bei BPEL mehr entwickelt werden als bei SOA-Me. Letztere enthielt nur 236 Elemente während es bei BPEL um 1258 Aktivitäten waren.

Damit ist das Entwickeln auf der SOA-Me Plattform klar einfacher.

5.7.2 Toolsupport

Der Toolsupport ist eng mit dem Aufwand verbunden. Denn je besser die Entwicklung durch Tools unterstützt wird, umso weniger aufwändig sollte sie sein.

Die Unterstützung durch Tools, die Quellcode generieren, lies sich mit Hilfe der Analyse der Codebasis messen.

Viele Aspekte der Entwicklung von Webservice-basierten SOA-Anwendungen werden sehr gut vom AXIS-Framework und den zugehörigen Tools unterstützt. Genau in diesen Bereichen wurde auch ein sehr hoher Anteil von generiertem Code gefunden. Dabei nutzt die SOA-Me-Anwendung diese Unterstützung etwas besser aus. Im Bereich der Webservices hat sie mit 65% den etwas höheren Anteil an generiertem Code als BPEL mit 53%.

Insgesamt ergibt sich aber ein anders Bild, da auch der BPEL-Client von AXIS generierten Code enthält und die Aufwandsreduktion bei den SOA-Me Stylesheets nicht durch den Einsatz von Tools, sondern durch geschicktes Kopieren schon vorhandener Teile entstand; die Anzahl der generierten Codezeilen bei Stylesheets also nicht zum Toolsupport zählen.

Somit konnte bei BPEL 37% des Codes durch Tools erzeugt werden, bei SOA-Me nur 28%, also gut 9 Prozentpunkte weniger.

Die Unterstützung durch Tools, die keinen Programmcode erzeugen, war bei beiden Anwendungen ungefähr gleich gut. Beide Systeme verfügten mit dem ActiveBPEL Designer bzw. dem SOA-Me Kompositionseditor über einen leistungsfähigen Editor. Im Fall von SOA-Me wurde der Entwurf der Datenbankservices sehr durch die Tools zum Hibernate-Framework erleichtert. Ob dieses Tool auch bei der Entwicklung der BPEL-Anwendung eingesetzt wurde, war weder dem Erfahrungsbericht der Entwickler noch den entsprechenden Konfigurationsdateien zu entnehmen.

5.7.3 Usability

Der hohe Aufwand, der in den Client für die BPEL-basierte Anwendung investiert wurde, macht sich bei der Usability bemerkbar. Der BPEL-Client bietet mehr den Benutzer unterstützende UI-Konzepte und ist, nach Meinung der Probanden in den entsprechenden Versuchen, dadurch auch etwas besser zu bedienen als der SOA-Me Client.

Allerdings macht sich dieser Vorteil nicht in der Zeitspanne bemerkbar, die zum kompletten Durchlaufen des Prozesses nötig ist. Mit dem BPEL-Client wurde deutlich mehr Zeit benötigt.

Damit schneidet der SOA-Me-Client in dem Teilaspekt Zufriedenheit der Usability zwar schlechter ab als der BPEL-Client, ist aber trotzdem effizienter.

5.7.4 Wartbarkeit

Bei der Wartbarkeit verhält es sich, was die Codebasisgröße angeht, ähnlich wie beim Aufwand. Allerdings muss bei der Wartbarkeit die gesamte Codebasis (also die generierten und handgeschriebenen Teile) zusammengenommen gewertet werden. Denn die aus der Definition der Wartbarkeit folgenden Ziele (Fehlerbehebung, Erweiterung) benötigen ein Verständnis der gesamten Codebasis. Dieses Verständnis sollte umso leichter fallen, je weniger komplex z.B. eine einzelne Klasse ist. Damit spielen auch die Methoden pro Klasse für die Wartbarkeit eine Rolle.

Die gesamte Codebasisgröße beträgt bei BPEL, wieder hauptsächlich durch den Client bedingt, 21.319 Zeilen, bei SOA-Me 4.835. Die durchschnittliche Anzahl Methoden pro Klasse beträgt bei SOA-Me 3,14 und bei BPEL 7,42. Damit sollte die SOA-Me Codebasis leichter zu warten sein, wobei hier natürlich vernachlässigt wurde, wie gut der jeweilige Code dokumentiert ist.

Bei der Lesbarkeit der Kompositionen schnitt SOA-Me überraschend schlecht ab. Zwar empfanden die Probanden die EPK-Semantik tatsächlich intuitiver als die BPEL-Notation, SOA-Me schnitt aber in der Bewertung nur unwesentlich besser ab als BPEL, obwohl die BPEL-Komposition größer ist als die SOA-Me-Komposition.

So lässt sich in dieser Frage kein eindeutiger Sieger bestimmen.

5.7.5 Flexibilität

Für die Flexibilität war die Größe der Komposition als Metrik vorgesehen. Wie schon erwähnt ist diese bei SOA-Me geringer als bei BPEL. Somit scheint es einfacher die SOA-Me-Anwendung für nicht vorhergesehene Aufgaben einzusetzen als die BPEL-Anwendung, da bei SOA-Me tendenziell weniger Elemente verändert werden müssen.

6 Zusammenfassung und Ausblick

Wie sich bei dem Vergleich zwischen SOA-Me- und BPEL-Entwurf gezeigt hat, kann sich die experimentelle SOA-Me-Plattform bei der gegebenen Anwendung sehr gut gegenüber der BPEL-Plattform behaupten. Dabei ist sie, was den für die Entwicklung erforderlichen Aufwand angeht, BPEL sogar weit überlegen. Das Konzept der zur Laufzeit generierten Oberflächen aus [LLSG06] zahlt sich bei diesem Aspekt aus.

Unter diesem Konzept leidet aber leicht die Usability. Der individuelle Client des BPEL-Entwurfs kann sich besser an die speziellen Bedürfnisse der Anwendung zur Verwaltung von Abschlussarbeiten anpassen als der auf eine Handvoll UI-Elemente festgelegte SOA-Me Client. An dieser Stelle sollte das verantwortliche Task-Modell um die entsprechenden UI-Elemente erweitert werden. Auch sollte ein neues Task-Modell möglichst alle Standarddatentypen aus der XML Schema Definition des W3C unterstützen, um Probleme wie in 4.1.2 zu vermeiden. Mit diesem Thema beschäftigt sich eine zurzeit am Fachgebiet Software Engineering laufende Bachelorarbeit von Quang Lam Nguyen.

Auch gilt es weitere Funktionen der SOA-Me Plattform zu verbessern bzw. hinzuzufügen:

Ein Grund für das unerwartet schlechte Abschneiden des SOA-Me Systems bei der Lesbarkeit der zur Komposition nötigen EPKs lag in der expliziten Persistenz, die die betroffene EPK stark aufblähte und unübersichtlich werden ließ. Würde diese Persistenz implizit durch den SOA-Me Server erfolgen, wäre dieses Problem behoben und von einer Steigerung der Lesbarkeit ist auszugehen.

Auch die Möglichkeit innerhalb eines Prozesses einen Unterprozess aufzurufen (Implementierung des in 3.3.1 kurz angerissenen Prozesswegweisers) könnte zu einer besseren Lesbarkeit beitragen, da durch sie eine bessere Strukturierung der SOA-Me Prozesse möglich wäre.

Ebenfalls könnte es sich lohnen über die Integration einer verbesserten Fehlerbehandlung, die auf dem auch von BPEL verwandten Konzept von Ausnahmen beruht, die in entsprechenden für eine Fehlerbehandlung vorgesehenen Prozessteilen bearbeitet werden, nachzudenken. Diese könnten ebenfalls die Struktur der EPKs und damit die Lesbarkeit verbessern. Außerdem würde es ein solches System auch erlauben Fehler, die bei der Ausführung eines Stylesheets entstehen können, abzufangen.

Was den Toolsupport angeht, so macht sich bei SOA-Me nur die Entscheidung für XSL-Stylesheets negativ bemerkbar. Es gibt einfach keine Tools, die bei der Erstellung dieser Stylesheets, so wie sie für das SOA-Me System benötigt werden, entscheidend assistieren. Dies wäre eine lohnende zusätzliche Funktion für den SOA-Me-Kompositioseditor.

Ein Aspekt, der bei dieser Arbeit nur am Rande angesprochen wurde, ist die Sicherheit der Datenübertragung. Hier benötigt das SOA-Me-System ein besseres Sicherheitskonzept wenn es jemals in Produktivumgebungen eingesetzt werden soll. Dabei wären zwei Alternativen, die in [Gro06] vorgestellt werden, denkbar: Einmal die Verschlüsselung der gesamten Verbindung mit Techniken wie TLS und SSL und einmal die gezielte Verschlüsselung einzelner Nachrichtenteile (z.B. mit WS-Security).

Zusammenfassend kann gesagt werden, dass mit BPEL höchstens ein leicht besseres Endergebnis möglich war, dies jedoch einen erheblichen Aufwand gekostet hat. Dadurch, dass bei der SOA-Me-Plattform Funktionalitäten wie der Client aber auch Experience-Forum, Gruppen- und Aufgabenverwaltung sowie ein Sicherheitssystem schon vorhanden sind und nicht aufwändig entworfen und implementiert werden müssen, ist dieses System für interaktionsreiche Geschäftsprozesse wie das Verwaltungssystem für Abschlussarbeiten die bessere Wahl. Für hochautomatisierte Prozesse, die ohne Benutzerinteraktionen auskommen, bietet sich

wohl eher BPEL an, wobei eventuell auch hier ein Vergleich der beiden Plattformen in der Art dieser Arbeit interessant wäre.

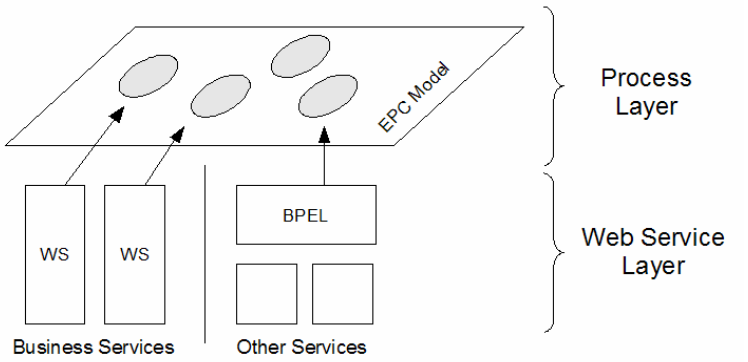

Abbildung 28: Konzept für die Verbindung von SOA-Me und BPEL [LLSG06]

Abschließend soll noch ein Konzept für gemischte Prozesse, die sowohl Teile mit hoher Benutzerinteraktion als auch hoch automatisierte Teile enthalten, vorgestellt werden: Die Kombination von SOA-Me und BPEL (siehe Abbildung 28) wie sie in [LLSG06] vorgestellt wurde. Da BPEL-Prozesse über ein Webservice-Interface verfügen, können diese auch durch den SOA-Me Server aufgerufen werden. Somit ist es möglich den groben Ablauf und die Benutzerinteraktionen mit SOA-Me auf der Prozessebene zu realisieren und BPEL für die Komposition der Webservices zu verwenden.

Literaturverzeichnis

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana: „Business Process Execution Language for Web Services Version 1.1“ In: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf vom 31.07.2007

[ADH03] Wil M. van der Aalst, Marlon Dumas, Arthur H. ter Hofstede: „Web Service Composition Languages: Old Wine in New Bottles?” In: euromicro, p. 298, 29th Euromicro Conference (EUROMICRO'03), 2003.

[Axi] Apache Software Foundation : „WebServices – Axis” In: http://ws.apache.org/axis/ vom 7.8.2007

[BCR94] Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach, “The Goal Question Metric Approach”, 1994

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: „Design Patterns“ Addison-Wesley Publishing Company, 1995

[GL05] Volker Gruhn, Ralf Laue: „Einfache EPK-Semantik durch praxixtaugliche Stilregeln“ In: Markus Nüttgens, Frank J. Rump (Ed.): EPK2005 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Hamburg, Deutschland, 08.12.2005 - 09.12.2005, 176-189. CEUR-WS

[GLM06] Volker Gruhn, Ralf Laue, Frank Meyer: „Berechnung von Komplexitätsmetriken für ereignisgesteuerte Prozessketten” In: Markus Nüttgens, Frank J. Rump, Jan Mendling (Ed.): “EPK 2006 Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten”, Wien, Österreich, 30.11 - 01.12 2006 189-202

[Gri07] Sabina Grimm: „Integration eines Experience Forums in das SOA-Me-System“ Studienarbeit, Universität Hannover, 2007.

[Gro06] Rix Groenboom: „Securing Web Services“ In: ASQF e.V. (ed.): Software Quality in Service-Oriented Architectures, CONQUEST 2006, pp. 215-229

[Hib] Red Hat Middleware, LLC: “hibernate.org – Hibernate” In: http://www.hibernate.org/ vom 7.8.2007

[IEE90] IEEE Standards Board: „IEEE Standard Glossary of Software Engineering Terminology“—IEEE Std 610.12-1990. 1990

[Int06] Sebastian Intas; „Konzept für die Einbindung von Webservices in Ereignisgesteuerte Prozessketten“. Masterarbeit, Universität Hannover, 2006.

[iTe] Bruno Lowagie: „iText, a Free Java-PDF Library“ In: http://www.lowagie.com/iText/ vom 7.8.2007

[KKL+05] Matthias Kloppmann, Dieter Koenig, Frank Leymann, Gerhard Pfau, Alan Rickayzen, Claus von Riegen, Patrick Schmidt, and Ivana Trickovic: „WS-BPEL Extension for People BPEL4People A Joint White Paper by IBM and SAP” IBM/SAP, August 2005

[LLSG06] Daniel Lübke, Tim Lüecke, Kurt Schneider, Jorge Marx Gómez: „Using Event-Driven Process Chains for Model-Driven Development of Business Applications“ In: 3rd GI Workshop XML4BPM – XML Interchange Formats for Business Process Management, February 2006.

[LS06] Daniel Lübke, Kurt Schneider: „Leveraging Feedback on Processes in SOA Projects“ In: Ita Richardson, Per Runeson, Richard Messnarz (Eds.): Software Process Improvement, 13th European Conference, EuroSPI 2006, pp 194-205, 2006

[Lue05] Tim Lüecke. „Development of a concept for creating and managing user-interfaces bound to business processes“ Masterarbeit, Universität Hannover, 2005

[Met] Sourceforge.net: „Metrics 1.3.6 - Getting started“ In: http://metrics.sourceforge.net/ vom 26.07.2007

[Oec03] Philippe Oechslin: „Making a Faster Cryptanalytic Time-Memory Trade-Off“ In: Lecture Notes in Computer Science (Proceedings of Crypto'03) , 2003

[Sal07] Alex Salnikow: „Ermittlung von Testabdeckungsmetriken in BPEL-Kompositionen“, Masterarbeit, Universität Hannover, 2007

[Sch07] Oleg Schmelzle: „Transformation von annotierten Geschäftsprozessen nach BPEL“ Masterarbeit, Universität Hannover, 2007

Anhang

Beispiel für ein Prozessdokument

```
<root><!-- Wurzelelement aller SOA-Me Prozesse-->
   <starter-user>
      <!-- Benutzer der den Prozess gestartet hat -->
   </starter-user>
   <thesisOverview>
      <!-- Zusammenfassung der dem System bekannten Arbeiten -->
      <thesis>
         <!--
          Kurzinformation über eine Arbeit, vgl. Datenbankschema
         -->
         <ID/>
         <title/>
         <tutor-info>
            <!-- Kurzinformationen über Betreuer -->
         </tutor-info>
      </thesis>
   </thesisOverview>
   <wsCallResult id="626">
      <!-- Status eines Webserviceaufrufs -->
   </wsCallResult>
   <thesis-id>
      <!-- ID einer Arbeit -->
   </thesis-id>
   <subject-info>
      <!-- Ausführliche Informationen über ein Thema (vgl. Schema) -->
      <reason/>
      <criteria/>
      <conditions/>
      <appliableDate/>
      <task/>
      <titleType/>
      <title/>
      <background/>
      <profile/>
   </subject-info>
   <tutor>
      <!--  Details über den Betreuer, vgl. Datenbankschema (Person) -->
      <ID/>
      <salutation/>
      <title />
      <firstName />
      <lastName/>
      <emailAddress/>
      <Telefone/>
      <roomNumber/>
      <studentId/>
   </tutor>
   <firstApprover>
      <!-- Details über den Erstprüfer, wie beim Tutor-Element-->
      …
   </firstApprover>
   <subject-id><!-- ID eines Themas --></subject-id>
   <student>
      <!-- Details über den Studenten, wie beim Tutor-Element-->
      …
   </student>
   <secondApprover>
      <!-- Details über den Zweitprüfer, wie beim Tutor-Element-->
```

```
      …
   </secondApprover>
   <thesis-state><!-- Status der Arbeit --></thesis-state>
   <approved><!-- gibt an, ob das Thema genehmigt wurde</approved>
   <rejectReason>
      <!-- Grund der Ablehnung, falls nicht genehmigt -->
   </rejectReason>
   <thesis-info>
      <!-- Details über die Arbeit (vgl. Datenbankschema) -->
      <startDate/>
      <plannedDeliveryDate/>
      <type/>
      <intern/>
   </thesis-info>
   <intermediatePresentationDate><!-- Datum des Zwischenvortrags -->
   </intermediatePresentationDate>
   <finalPresentationDate><!-- Datum des Endvortrags -->
   </finalPresentationDate>
   <tutor-user><!-- Benutzername des Betreuers --></tutor-user>
   <tutor-wat><!-- Zugangsdaten des Betreuers zum WAT-Service -->
      <username/>
      <password/>
   </tutor-wat>
   <firstApprover-user><!-- Benutzername des Erstprüfers -->
   </firstApprover-user>
   <ratingSecondApprover>
      <!-- Bewertung des Zweitprüfers als Note und Kommentar -->
      <rating/>
      <comment/>
   </ratingSecondApprover>
   <ratingFirstApprover>
      <!-- Bewertung des Erstprüfers wie beim Zweitprüfer -->
      …
   </ratingFirstApprover>
   <ratingTutor><!-- Bewertung des Betreuers als Kommentar -->
      <comment/>
   </ratingTutor>
</root>
```

Fragebogen zur Lesbarkeit der BPEL- und SOA-Me-Notation

Bitte beantworten Sie folgende Fragen durch Auswerten der BPEL- bzw. SOA-Me Prozessbeschreibungen des Verwaltungstools für Abschlussarbeiten.

1 Ist es möglich gleichzeitig die Bewertung von Erst- und Zweitprüfer einzugeben?

Relevante BPEL Datei: ThesisProcess.bpel

Ja	Nein

Wie schwer war es dies herauszufinden?

Sehr leicht	Leicht	Durchschnittlich	Schwer	Sehr Schwer

2 Ist es möglich erst die Abgabe der Arbeit zu erfassen und erst danach den Abschlussvortrag zu planen?

Relevante BPEL Datei: ThesisProcess.bpel

Ja	Nein

Wie schwer war es dies herauszufinden?

Sehr leicht	Leicht	Durchschnittlich	Schwer	Sehr Schwer

3 Gibt es bei Studeinarbeiten (ThesisTyp 30) einen Zwischenvortrag?

Relevante BPEL Datei: ThesisProcess.bpel

Ja	Nein

Wie schwer war es dies herauszufinden?

Sehr leicht	Leicht	Durchschnittlich	Schwer	Sehr Schwer

4 Ist es möglich, dass das Datum des Abschlussvortrages vor dem Datum der Abgabe liegt?

Relevante BPEL Datei: ScheduleFinalPresentationProcess.bpel

Ja	Nein

Wie schwer war es dies herauszufinden?

Sehr leicht	Leicht	Durchschnittlich	Schwer	Sehr Schwer

5 Welcher ThesisState entspricht einer beendeten Arbeit?

Relevante BPEL Datei: FinishThesisProcess.bpel

40	60	70

Wie schwer war es dies herauszufinden?

Sehr leicht	Leicht	Durchschnittlich	Schwer	Sehr Schwer

Antworten (BPEL/SOA-Me): 2 ja/ja, 3 ja/nein, 4 nein/ja, 5 nein/nein, 6 70/70

Szenario zum Messen der Prozessdauer

Bitte führen Sie folgende Schritte mit dem System aus:

1. Melden Sie sich mit Ihrem Nachnamen als User und Ihrem Vornamen als Passwort am System an.
2. Legen Sie ein neues Thema mit folgenden Daten an:
   - Betreuer: *sie selbst*
   - Erstprüfer: *sie selbst*
   - Titel: *Ihr Name:* Entwurfsvergleich einer SOA-Anwendung auf SOA-Me und BPEL-Basis
   - Typ: Bachelor-Arbeit
   - Hintergrund: Im Softwareprojekt 2006/07 wurde die SOA-Me-Plattform entwickelt.
   - Aufgabe: Die Aufgabe dieser Bachelorarbeit besteht darin, die Anwendung zur Verwaltung von studentischen Arbeiten auf das SOA-Me-System zu portieren.
   - Beginn: ab sofort
   - Randbedingungen: Erfahrungen mit Webservices und XML sind wünschenswert
   - Grund der Ausgabe: Das beschriebene Problem wurde auf einer Mailingliste geäußert und schreit danach, einmal für alle gelöst zu werden.
   - Benutzerprofil: Benutzer sind fiktive Projekt-Teams (insb. XP-Teams), die eine große Anzahl von Unit-Tests häufig ausführen wollen.
   - Bewertungskriterien: Programmierung der LösungLehnen Sie das Thema mit der Begründung „Kriterien nicht ausreichend“ ab.
4. Ändern Sie die Kriterien zu
   „Programmierung der Lösung
   Dokumentation der Performance, insb. ab wann lohnt sich das Verteilen?
   Eigenständigkeit und das Einbringen von eigenen Ideen (Sicherheit!)„
   und legen Sie das Thema erneut vor
5. Genehmigen Sie das Thema. Hinweise zum drucken können Sie ignorieren
6. Tragen Sie als Studenten/Anwärter „Christoph König“ oder „Leif Singer“ ein
7. Wählen Sie „Kurt Schneider“ als Zweitprüfer
8. Melden Sie die Arbeit als interne Bachelorarbeit an. Wählen Sie als Ausgabedatum den „1.1.2007“ und (nur SOA-Me) für das Enddatum den 30.4.2007
9. Planen Sie den Zwischenvortrag am 1.3.2007 14h
10. Planen Sie den Endvortrag am 20.4.2007 10h
11. Korrigieren Sie den Endvortrag auf 15.5.2007 11h
12. Erfassen Sie die Abgabe
13. Kommentieren Sie die Arbeit als Betreuer mit „gute Arbeit“
14. Kommentieren Sie die Arbeit als Erstprüfer mit „gute Arbeit“ und Bewerten Sie sie mit 2.0
15. Melden sie sich mit dem Usernamen „dreschel“ und dem Passwort „ulrike“ erneut beim System an (Client erneut starten)
16. Kommentieren Sie die Arbeit als Zweitprüfer mit „durchschnittliche Arbeit“ und Bewerten Sie sie mit 2.7
17. Überprüfen Sie die Zusammenfassung (nicht als „dreschel“)

Inhalt der CD-ROM

Die beiliegende CD-ROM hat folgenden Inhalt:

- Diese Ausarbeitung als
  - Word 2002 Dokument
  - Adobe PDF Dokument
- Die behandelten Programme
  - Auf BPEL-Basis
    - Als Quellcode in Form eines Eclipse Workspaces
    - Den Client als lauffähige Java-Anwendung
  - Auf SOA-Me-Basis
    - Als Quellcode in Form eines Eclipse Workspaces
    - Den Client als lauffähige Java-Anwendung
    - Den Integrationsserver als deploy-fähiges web application archive
    - Die SOA-Me Prozesse
    - Die von den Prozessen benötigten Webservices als deploy-fähiges web application archive

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit und die dazugehörige Implementierung selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, 15. August 2007 _____

Christoph König