Fachhochschule Köln

Cologne University of Applied Sciences

Fakultät für Informatik und Ingenieurswissenschaften

Masterthesis

zur Erlangung

des Grades

Master of Science

im Verbundstudiengang Wirtschaftsinformatik

„Konzeption und Implementierung einer generischen Schnittstelle für menschliche Interaktion in Service-orientierten Architekturen“

Erstprüferin: Prof. Dr. Heide Faeskorn-Woyke

Zweitprüferin: Prof. Dr. Birgit Bertelsmeier

vorgelegt am: 17.04.2012

von: Sven Bernhardt

An der Ronne 9

50859 Köln

Matrikel-Nr.: 11073595

Telefon-Nr.: 0177 / 4023115

Email: svenbernhardt@googlemail.com

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

ACM	Adaptive Case Management
ADF	Application Development Frameworks
API	Application Programming Interface
BAS	Business Activity Service
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modelling Notation
BPS	Business Process Service
CIO	Chief Information Officer
CMMN	Case Management Modeling and Notation
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
CRUD	Create Read Update Delete
CS	Connectivity Service
DS	Decision Service
DTO	Data Transfer Object
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planing
ES	Entity Service
GWT	Google Web Toolkit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Text Protocol
JAAS	Java Authentication and Authorization Services
JAR	Java Archive
JDK	Java Development Kit
JEE	Java Enterprise Edition
JSE	Java Standard Edition
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model View Controller
OASIS	Organization for the Advancement of Structured Information Standards

OPSS	Oracle Platform Securtiy Service
PCI DSS	Payment Card Industry Data Security Standard
PoC	Proof of Concept
POM	Project Object Model
PS	Presentation Service
PVM	Process Virtual Machine
REST	Representational State Transfer
RIA	Rich Internet Application
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SOA	Service orientierte Architektur
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language
URI	Unified Resource Identifier
WAR	Web Archive
WSDL	Web Service Description Language
WS-HT	WS Human Task

1 Einleitung

1.1 Motivation

Die Integration von menschlichen Akteuren als Entscheidungs- und Wissensträger in Service-orientierte Architekturen ist eine Notwendigkeit, um komplexe fachliche Entscheidungen, bei denen viele verschiedene Parameter berücksichtigt werden müssen, IT-technisch effektiv umsetzen zu können. Die Plattform- beziehungsweise Toolhersteller haben diese Notwendigkeit bereits frühzeitig erkannt und in ihre Produkte entsprechende Lösungen integriert, welche die Interaktion zwischen Menschen und IT-Komponenten ermöglichen.

In diesem Zusammenhang ist von Human Workflows und Human Workflow-Engines die Rede, wobei es sich in der Regel um herstellerspezifische Implementierungen handelt. Die Interaktion zwischen Menschen und den Human Workflow-Engines erfolgt mittels so genannter „Tasks“. Aus diesem Grund werden die Human Workflow-Engines auch als Task-Engines bezeichnet. Für die Bearbeitung von Tasks werden Client-Applikationen benötigt, die über ein Engine-spezifisches API (Application Programming Interface) mit der Task-Engine kommunizieren.

Mit der WS-Human Task Spezifikation (WS-HT) gibt es in diesem Bereich Bestrebungen die Schnittstellen der proprietären Engine-Implementierungen durch einen standardbasierten Ansatz abzulösen. Allerdings befindet sich diese Spezifikation seit nunmehr fünf Jahren bei der OASIS (Organization for the Advancement of Structured Information Standards) im Standardisierungsprozess. Wie lange dieser Prozess noch andauert ist nicht abzusehen. Die Akzeptanz der Adaption durch Industrieunternehmen leidet unter diesem ungewissen Status.

Mit dem Opensource Projekt Apache HISE[1], das vor gut zwei Jahren gestartet wurde, ist der Versuch unternommen worden, eine Task-Engine basierend auf den Vorgaben der WS-HT Spezifikation zu implementieren. Mittlerweile sind die Entwicklungsarbeiten an dem Projekt aus nicht genannten Gründen eingestellt worden. Ob und inwiefern sich aktuell vorhandene Human Workflow-

[1] Vergleiche [HISE]

Implementierungen bereits heute an den Vorgaben der WS-HT Spezifikation orientieren, wurde im Rahmen einer dieser Abschlussarbeit vorgelagerten, Projektarbeit[2] evaluiert. Ergebnis hiervon ist, dass die untersuchten Engines von Oracle und Activiti zwar in Teilbereichen funktionale Aspekte, die in der genannten Spezifikation definiert werden, berücksichtigen. Formale oder syntaktische Vorgaben werden dabei aber fast gänzlich ignoriert.

1.2 Problemstellung und Zielsetzung

Die Ideen und Konzepte, welche in der WS-HT Spezifikation definiert werden, sind gut durchdacht und schlüssig. Ein Problem der Spezifikation ist, dass die proprietären Lösungen von namhaften Herstellern bereits verbreitet und etabliert sind. Der Aufwand für die Hersteller, die aktuellen Lösungen an einen potentiellen Standard anzupassen, wäre immens. Da sich WS-HT aber, wie bereits angesprochen, noch immer im Standardisierungsprozess befindet und noch kein verbindlicher Standard ist, besteht für Hersteller kein Handlungsbedarf. Zudem stehen aktuell andere Themen beziehungsweise Standards, wie beispielsweise der BPMN 2.0-Standard (Business Process Modelling and Notation) oder ein Standard für die Definition adaptiver Prozesse (CMMN, Case Management Modeling and Notation), mehr im Fokus der Produkthersteller.

Die Probleme, die mit der Verwendung herstellerspezifischer Schnittstellen einhergehen, sind jedoch offensichtlich: Die Interoperabilität und Wiederverwendbarkeit einer proprietären Human Workflow-Engine ist stark eingeschränkt. Ändern sich die Schnittstellen einer Task-Engine, müssen alle konsumierenden Applikationen angepasst werden. Anwenderunternehmen machen sich überdies abhängig von einem Hersteller. Ein Wechsel auf die Plattform eines anderen Herstellers ist infolgedessen aufwändig und kostenintensiv. Eine Möglichkeit um solchen Probleme zu begegnen ist die proprietären Schnittstellen mit Hilfe einer Fassade weg zu kapseln. Da eine eigene Fassade wiederrum eine proprietäre Schnittstelle darstellt, entstehen auf diese Weise allerdings Insellösungen, die unter Umständen auch nicht kompatibel zu den Task-Engines von Drittherstellern sind.

[2] Vergleiche [BERS11]

Ausgehend von den skizzierten Problemen, wird in der vorliegenden Abschlussarbeit eine allgemeingültige Abstraktionsschicht in Form eines kleinen Frameworks des Generic Human Interaction Adapter-Frameworks – kurz GHIA –, konzipiert und, zum Zweck der Verifizierung der Konzeption, prototypisch implementiert. Mit Hilfe dieses Adapterframeworks sollen herstellerspezifische Funktionalitäten gekapselt und in standardisierter Form zur Verfügung gestellt werden. Als Grundlage für Konzeption und Implementierung des Frameworks dient die WS-HT Spezifikation beziehungsweise die darin definierten Artefakte für eine SOAP-Schnittstelle. Ausgehend davon könnten, wie bereits in der Projektarbeit angesprochen, Rückschlüsse in Richtung der Praxistauglichkeit und Vollständigkeit der Spezifikation gezogen werden. Vorteile der Verwendung eines solchen Frameworks sind die Unabhängigkeit von einer spezifischen Implementierung und im Idealfall von einer speziellen Laufzeitumgebung sowie die Robustheit gegenüber Änderungen an den proprietären Schnittstellen und Datenstrukturen.

Das **Hauptziel** dieser Arbeit besteht in der Erstellung des GHIA-Frameworks. Dies lässt sich in die beiden Teilziele:

- **Teilziel 1**: Konzeption auf Basis der WS-HT Spezifikation
- **Teilziel 2**: Verifizierung der Konzeption mit Hilfe einer prototypischen Implementierung

Wei das GHIA-Framework auf der WS-HT Spezifikation fußt, ergibt sich unmittelbar ein weiteres Teilziel:

- **Teilziel 3**: Bewertung der Praxistauglichkeit der WS-HT Spezifikation als generische Fassade für Task-Engines

1.3 Gliederung

Im Anschluss an dieses Einleitungskapitel werden in Kapitel 2 grundlegende Begrifflichkeiten und Zusammenhänge aus dem Bereich SOA erläutert. In Kapitel 3 werden die Anforderungen an das GHIA-Framework formuliert und diese mit Abnahmekriterien sowie einem „Proof-of-Concept“-Szenario (PoC) hinter-

legt. In Kapitel 4 werden verschiedener Aspekte der verwendeten Task-Engines von Oracle und Activiti, die im Rahmen dieser Abschlussarbeit betrachtet werden, analysiert. Die hierbei gewonnenen Ergebnisse bilden gemeinsam mit den Anforderungen die Grundlage für die in den Kapiteln 5 und 6 folgende Konzeption und Implementierung des GHIA-Frameworks. In Kapitel 7 erfolgt anschließend die Validierung der Konzeption und Implementierung gegen die definierten Anforderungen. Grundlage hierfür ist eine prototypische Umsetzung des GHIA-Frameworks sowie des definierten PoC-Szenarios. Das letzte Kapitel enthält eine abschließende Zusammenfassung sowie ein Fazit. Zudem wird ein Ausblick gegeben, welcher mögliche Einsatzszenarien des Frameworks sowie Erweiterungen skizziert.

1.4 Konventionen

Eine wichtige Grundlage für das Verständnis der vorliegenden Arbeit ist die im Vorfeld angefertigte Projektarbeit zum Thema „Integration von Menschen in automatisiert ablaufende Geschäftsprozesse“. Relevante Ergebnisse dieser Projektarbeit werden referenziert und nicht im Detail wiedergegeben. Detaillierte Informationen zu den entsprechenden Sachverhalten können an den entsprechenden Stellen, auf die explizit verwiesen wird, in der Projektarbeit eingeholt werden.

In dieser Ausarbeitung werden Fachbegriffe in der Regel im Original belassen und ohne Übersetzung verwendet, da im Deutschen häufig allgemein akzeptierte Übersetzungen fehlen. Entsprechende Fachbegriffe werden im Glossar vermerkt und erklärt. Zudem werden verwendete Synonyme aufgeführt.

2 Menschliche Interaktion in Serviceorientierten Architekturen

In den nachfolgenden Abschnitten wird die Rolle menschlicher Interaktion im Kontext Service-orientierter Architekturen (SOA) erörtert. Dabei werden zunächst die Grundkonzepte erläutert und ein Architekturvorschlag, wie eine Anwendungslandschaft, den Konzepten der Serviceorientierung folgend, aufgebaut werden kann, skizziert. Auch die Besonderheiten hinsichtlich der Interaktion zwischen menschlichen Akteuren und den entsprechenden, zum Teil langlaufenden Services, sind ein Thema. Mit dem, von Thomas Erl[3] definierten, User Interface Mediator-Pattern wird ein Designkonzept vorgestellt, wie die Interaktion zwischen grafischen Benutzeroberflächen (GUI, „Graphical User Interface“) definiert und implementiert werden kann.

2.1 Service-orientierte Architekturen (SOA)

Wird der Begriff Service-orientierte Architektur, beziehungsweise das korrespondierende Akronym SOA, bei einer bekannten Internet Suchmaschine eingegeben, so werden 119.000.000 Treffer angezeigt.

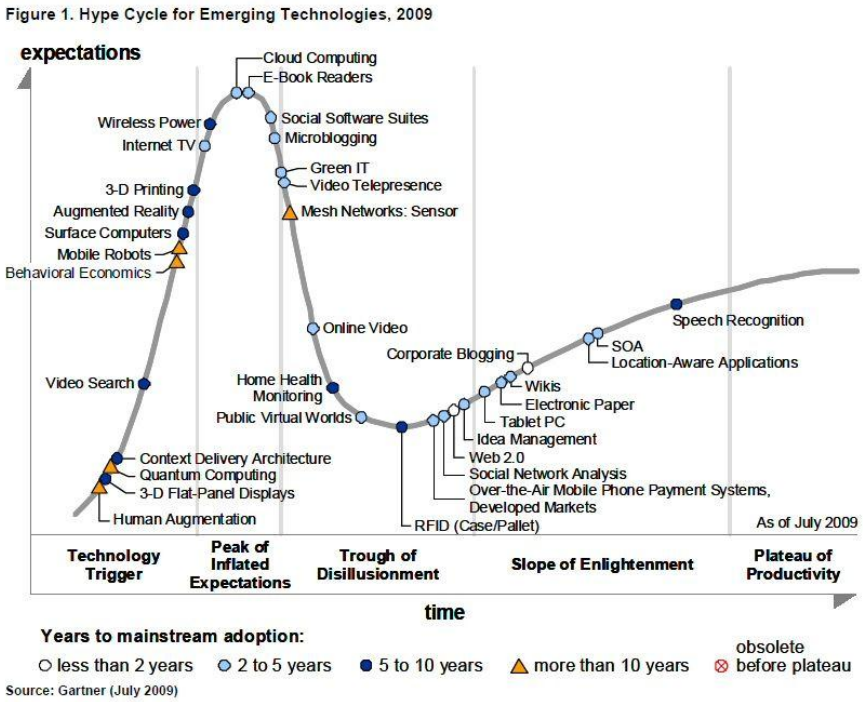

Abbildung 1: Gartner Hype-Cycle 2009[4]

[3] Vergleiche auch [ERLT091]], S. 366-371
[4] Aus [PÜCH09]

Der SOA-Begriff scheint offenbar noch immer aktuell zu sein. Auch, wenn der Hype rund um das Thema vorbei ist. Dies zeigte bereits der Gartner Hype-Cycle aus dem Jahr 2009, der in Abbildung 1 dargestellt ist. SOA war bereits zu diesem Zeitpunkt über die Phasen Hype sowie Desillusionierung hinweg und nach der Abklärungsphase auf dem besten Weg in Richtung Mainstream. Dies ist in der Abbildung anhand der farblichen Markierung des Punktes SOA erkennbar.

Seit der Erstellung des dargestellten Hype-Cycle sind fast drei Jahre vergangen. SOA müsste, wenn man dem nach dem Trend geht, heute fast Mainstream sein. In der Realität ist dem aber nicht so. Es existieren noch immer unterschiedliche und vor allem, falsche Meinungen hinsichtlich der Frage was SOA ist. Um dies zu verdeutliche werden im Folgenden zwei Beispiele aufgeführt.

Eine Aussage ist, dass SOA gleichbedeutend mit der Webservice-Technologie ist. Überspitzt gesagt, gehen Verfechter dieser Sichtweise davon aus, dass sobald Funktionalitäten mit Hilfe der Webservice-Technologie umgesetzt werden, auch im Sinne einer Service-orientierten Architektur gehandelt wird. Andere wiederrum denken, dass SOA das neue Enterprise Application Integration (EAI) beziehungsweise eine Weiterentwicklung davon ist und die vorhandenen Probleme allein durch den Einsatz einer Middleware-Lösung adressiert werden können. Die beiden dargestellten Meinungsbilder zum Thema SOA betrachten maximal Teilaspekte dessen, was sich tatsächlich hinter dem Begriff verbirgt.

Ursächlich für solche falschen Vorstellungen von der SOA-Thematik ist die Tatsache, dass SOA ein umfassendes Konzept darstellt. Dabei wurde das Rad nicht neu erfunden, sondern bestehende und auch bewährte Methoden sowie Konzepte aufgegriffen und verwendet. Hieraus entstehen Berührungspunkte zu anderen bestehenden Themen, was die Verwirrung rund um das Thema SOA noch zusätzlich beeinflusst.

Um solchen Missverständnissen vorzubeugen, soll an dieser Stelle eine, für diese Arbeit gültige, Definition einer Service-orientierten Architektur erarbeitet werden. Die im Folgenden aufgeführte SOA-Definition soll die Grundlage für die weiteren Ausführungen dieses Abschnitts sein. Die Definition stammt von der OASIS (Organization for the Advancement of Structured Information Standards)

und wurde im Rahmen ihres SOA-Referenzmodells publiziert. Die zentralen Begriffe der Definition sind fett hervorgehoben.

*„Service Oriented Architecture (SOA) is a **paradigm** for organizing and utilizing **distributed capabilities** that may be under the control of **different ownership domains**."*[5]

Den Begriff des Paradigmas, also des Musters oder Vorbildes, soll an dieser Stelle als Architekturmuster konkretisiert werden. Demnach ist SOA also ein Architekturmuster mit dessen Hilfe verteilte Ressourcen, die aus unterschiedlichen Verantwortungsbereichen stammen, organisiert und gemeinsam genutzt werden können. Bei den Ressourcen handelt es sich um Services, die aus IT-Sicht eine Softwarekomponente darstellen. Diese sind im Idealfall durch fachliche Anforderung motiviert und bilden ein in sich abgeschlossenes Stück Geschäftslogik ab.

Technisch können Services beispielsweise als SOAP Webservices umgesetzt werden, welche in irgendeiner Form verteilt sind. Die Verteilung meint hierbei nicht allein die physikalische Verteilung der technischen Implementierungen auf unterschiedlicher Hardware, sondern vielmehr die organisatorische Verteilung von Services über Abteilungs- und Unternehmensgrenzen hinweg. Dadurch entsteht aus technischer Sicht die Notwendigkeit sicherzustellen, dass klare Kommunikationsvereinbarungen in Form robuster Schnittstellen bestehen und bewährte Protokolle für die Kommunikation verwendet werden. In diesem Zusammenhang ist auch von Interoperabilität die Rede. Diese ist durch die Verwendung von Industriestandards sicherzustellen. So können zum Beispiel Schnittstellenbeschreibungen mit Hilfe der Webservice Description Language (WSDL) abgebildet und als Kommunikationsprotokoll das Hypertext Transfer Protocol (HTTP) verwendet werden.

Um über Abteilungs- und Unternehmensgrenzen hinweg verteilte Services organisieren, wiederfinden und letztendlich auch benutzen zu können, müssen die Services beziehungsweise deren Beschreibungen an zentraler Stelle hinterlegt werden. In Service-orientierten Architekturen wird zu diesem Zweck idealerweise eine so genannte „Service Registry" eingesetzt. Vom Prinzip her ist eine

[5] Aus [SOARM], S. 8

Service Registry vergleichbar mit einem öffentlichen Telefonbuch: Alle Services sind hier namentlich, mit einer Beschreibung und der entsprechenden Endpunkt-URL, über die der Service erreichbar ist, aufgeführt. Im Zusammenhang mit der Servicebeschreibung ist auch von einem Servicekontrakt[6] die Rede. Dieser besteht aus einer technischen Servicebeschreibung, bei Webservices beispielsweise in Form einer WSDL-Datei, und weiteren fachlichen Informationen zu Servicefunktionalität und –qualität. Hierzu zählen zum Beispiel Service Level Agreements (SLA) die Angaben hinsichtlich Verfügbarkeiten oder Antwortzeiten machen. Der Servicekontrakt ist dabei unabhängig von der eigentlichen Serviceimplementierung. Dies hat den Vorteil, dass die Serviceimplementierung geändert werden kann, ohne Anpassungen am Servicekontrakt oder auf Seiten des Service-Konsumenten vornehmen zu müssen. In diesem Zusammenhang ist auch von loser Kopplung die Rede.

Die Service-Registry ist die erste Anlaufstelle, um nach einer benötigten Funktionalität zu suchen. So ist es möglich redundante Implementierungen zu vermeiden. Idealerweise existiert bereits ein Service der die gesuchte Funktion bietet und der verwendet werden kann. Ein Beispiel hierfür ist ein Service für die Abwicklung von Kreditkartenzahlungen, der heute von fast jedem Online-Shop benötigt wird. Da für den Umgang mit sensiblen Kreditkarteninformationen besondere Restriktionen bestehen, die im Payment Card Industry Data Security Standard (PCI DSS) festgelegt werden, ist es für die Shop-Provider in der Regel günstiger einen Service von einem Drittanbieter zu benutzen, als eine Eigenimplementierung vorzunehmen.

Bei einer Service-orientierten Architektur handelt es sich also zusammenfassend betrachtet um ein Architekturmuster für verteilte Systeme. Dabei ist eine SOA an sich technologieneutral. Wichtig ist, dass verbindliche und robuste Serviceschnittstellen für Services definiert werden. Die erfolgreiche Umsetzung einer SOA erfordert, wie jede andere Architektur – und das gilt nicht nur für IT-Architekturen – auch, die Beachtung einiger Grundprinzipien und Designrichtlinien. Diese wurden in diesem Abschnitt bereits indirekt angesprochen und sollen an dieser Stelle noch einmal explizit aufgeführt werden:

[6] Siehe hierzu auch [ERLT092]

- Verwendung von Standards wodurch die Interoperabilität sichergestellt wird,
- Lose Kopplung (unter anderem durch die Trennung von Servicekontrakt und Serviceimplementierung),
- Wiederverwendung bestehender Funktionalitäten.

Die konsequente und konsistente Anwendung dieser Prinzipien und Richtlinien führt auf IT-Ebene zu einer maximal flexiblen und leicht anpassbaren IT-Systemlandschaft. Neue oder geänderte Anforderungen seitens des Fachbereichs können so schnell implementiert werden. Unternehmen können so sehr agil auf veränderte externe Einflussgrößen reagieren und sich dadurch Wettbewerbsvorteile gegenüber der Konkurrenz verschaffen.

Anhand dieser Zusammenfassung wird klar: SOA ist kein revolutionäres und gänzlich neues Konzept. Vielmehr ist SOA eine Sammlung bereits vorhandener und bewährter Konzepte, wie zum Beispiel das der Wiederverwendbarkeit von Komponenten, gepaart mit neuen Einflüssen, wo beispielsweise ein Service der elementarste Architekturbaustein ist.

2.2 SOA Referenzarchitektur

Nachdem im vorherigen Abschnitt definiert wurde, wie der Begriff SOA zu charakterisieren ist, wird es im Folgenden darum gehen, wie eine IT-Landschaft im Sinne der Serviceorientierung aufgebaut werden kann. In diesem Zusammenhang wird zunächst darauf eingegangen, wie Anwendungslandschaften aussehen die zumeist applikationszentriert aufgebaut sind, wie in solchen Landschaften die Integration verschiedener Applikationen realisiert wird und wo dabei die Probleme solcher Architekturen liegen.

2.2.1 Klassischer Ansatz: „Point-to-Point“

Die Applikationen sind in vielen IT-Organisationen heute immer noch der kleinste Baustein innerhalb einer bestehenden Systemlandschaft. Um in solchen Szenarien Applikationen zu verbinden, kann die Integration der Systeme individuell und unter Verwendung der jeweiligen, zumeist proprietären, APIs implementiert werden. Hierbei handelt es sich um eine Point-to-Point Integration. Was passiert, wenn sich ein solches Integrationsmuster durch eine komplexe

Applikationslandschaft zieht in der es mehrere Client- und Backendapplikationen gibt die miteinander integriert sind, zeigt Abbildung 2.

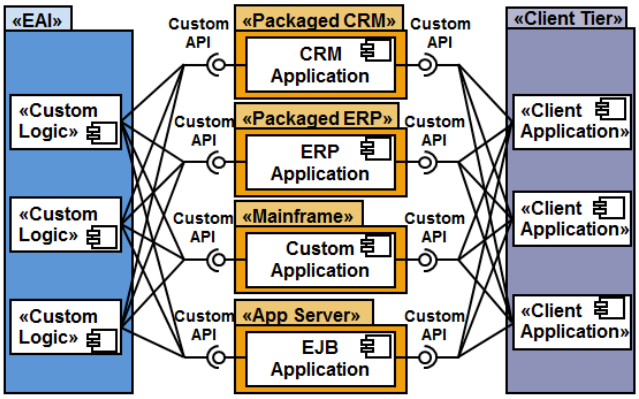

Abbildung 2: Point-to-Point Integration mehrerer Systeme[7]

In solchen Szenarien sind Anpassungen oder Erweiterungen zeitintensiv und somit auch teuer. Da die Applikationen sehr eng miteinander gekoppelt sind, bedingen Änderungen an einer der Applikationen in der Regel auch Änderungen an den angebundenen Applikationen. Vor jeder Änderung ist es also nötig die Auswirkungen auf andere Systeme zu analysieren. Auch das Einbinden neuer Applikationen in eine solche Systemlandschaft ist ein langwieriger Prozess, da hier zunächst die Integrationslogik implementiert werden muss. Eine schnelle Reaktion auf veränderte oder neue fachliche Anforderungen ist somit kaum möglich. Ein weiteres Problem stellen die proprietären Systemschnittstellen dar. Hier ist die Gefahr groß, dass das Know-how, zum Beispiel durch Mitarbeiterfluktuation, verlorengeht oder der Support seitens des Herstellers eingestellt wird. In beiden Fällen würden Anpassungen an den betroffenen Systemen ungleich schwieriger werden.

Die Backend-Applikationen betreffend existiert zudem das Problem, dass redundante Funktionalitäten vorhanden sind, welche im Zuge geänderter fachlicher Anforderungen ebenfalls angepasst werden müssen. In diesem Zusammenhang ist auch von Applikationssilos die Rede, da jede Applikation weitestgehend isoliert von den anderen Applikationen arbeitet. Ein Beispiel für eine solche redundante Funktionalität wären ein ERP- (Enterprise Resource Planning) und ein CRM-System (Customer Relationship Management), die beide eine Funktionalität für das Management von Kundendaten haben. Hinzu kommt

[7] Aus [OSOA01], S. 26

noch, dass im beschriebenen Beispiel die Datenstrukturen für einen Kunden unterschiedlich aussehen und nicht harmonisiert sind.

2.2.2 Klassischer Ansatz: Enterprise Application Integration (EAI)

Eine Möglichkeit den Problemen und Gefahren der Point-to-Point Integrationsarchitektur zu begegnen ist der Einsatz einer EAI-Middleware. Diese Komponente stellt ein Bussystem dar, an das alle Backend-Applikationen angeschlossen werden.

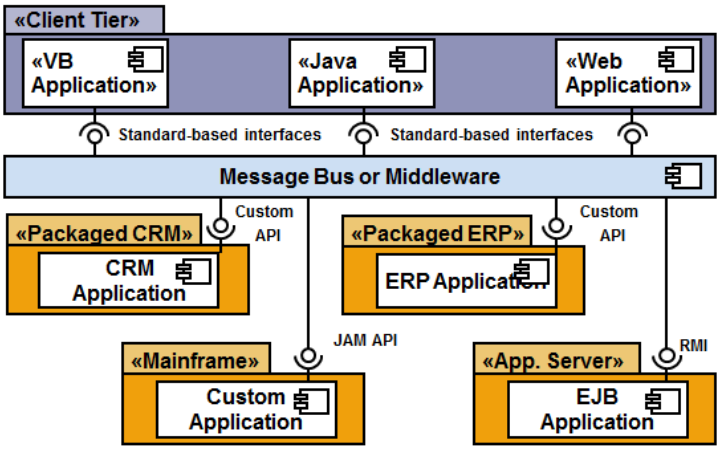

Abbildung 3: Systemintegration mit Hilfe einer EAI Middleware[8]

Die vorhandenen Client-Applikationen interagieren in einer solchen Architektur, wie Abbildung 3 zeigt, nur noch mit der EAI-Middleware und sind mit dieser über standardisierte Schnittstellen verbunden. Auch die Backend-Applikationen kommunizieren untereinander ausschließlich über das Bussystem. Die EAI-Middleware an sich enthält die benötigte Integrationslogik, wie beispielsweise die Transformationslogik für verschiedene Datenformate.

Dieser Architekturansatz weist gegenüber der im vorherigen Abschnitt gezeigten Point-to-Point Architektur einige Vorteile auf. So konnten die direkten Verbindungen der Systeme untereinander aufgelöst werden. Jegliche Form der Kommunikation läuft ausschließlich über den Bus. Weiterhin kann auf Client-Seite nun mit standardisierten Schnittstellen gearbeitet werden, hier existieren keine proprietären Schnittstellen mehr.

Problematisch an diesem Ansatz ist, dass die Integration der Backend-Applikationen mit dem Bussystem noch immer über proprietäre Schnittstellen erfolgt. Allerdings ist die Kopplung hier nicht mehr so stark, wie noch im „Point-

[8] Aus [OSOA01], S. 27

to-Point“-Szenario, denn bei Anpassungen der Schnittstellen auf Seiten der Backend-Systeme müssen an den Clients, zumindest solange sich an deren Schnittstelle nichts ändert, keine Anpassungen vorgenommen werden. Die Anbindung der Backend-Applikationen über proprietäre Schnittstellen ist allerdings noch immer suboptimal. Da die Verantwortung für die Implementierung der spezifischen Integrationslogik noch immer bei den Entwicklern liegt, existieren ähnliche Probleme und Gefahren, die bereits im vorherigen Abschnitt skizziert wurden (Stichwort: Mitarbeiterfluktuation, Know-How-Verlust). Auch die Problematik der Applikationssilos besteht weiterhin.

2.2.3 Moderner Ansatz: SOA Architektur

Zentrales Element einer Service-orientierten Architektur sind die Services und nicht mehr, wie bei den in den vorherigen Abschnitten 2.2.1 und 2.2.2 beschrieben Szenarien, die Applikationen. Eine der Hauptschwierigkeiten in diesem Zusammenhang ist, wie die benötigten Services identifiziert, beziehungsweise auch wie die Transformation von einer applikationszentrierten Architektur hin zu einer Service-orientierten Architektur, erfolgen kann.

Ein erster Schritt weg von den applikationszentrierten Ansätzen hin zum Service-orientierten Ansatz, besteht darin alle Applikationen beziehungsweise deren Funktionalitäten über standardbasierte Schnittstellen zur Verfügung zu stellen. In diesem Zuge erkannte redundante Funktionalitäten der einzelnen Applikationssilos können dann aus den Silos herausgelöst und als ein Service angeboten werden. Für das Beispiel der Kundendatenpflege aus Abschnitt 2.2.1 bedeutet dies, dass ein „CustomerManagement“-Service erstellt werden würde der die benötigte Logik implementiert, um Kundendatensätze sowohl im ERP- als auch im CRM-System verwalten zu können. Client-Applikationen würden im Anschluss ausschließlich mit dem neu erstellten Service und nicht mehr mit den einzelnen Backend-Applikationen kommunizieren.

Eine Möglichkeit die Transition zu einer Service-orientierten Architektur durchzuführen ist, diese, gemäß dem Leitsatz: „Think big, start small!“, in kleinen Schritten durchzuführen. Bei einem solchen Vorgehen wird die vorhandene IT-Systemlandschaft Schritt für Schritt umgestellt. Die verwendeten Vorgehensweisen und Methoden wachsen in diesem Zusammenhang evolutionär mit der entstehenden Service-orientierten Landschaft. Um das ganze Vorhaben in ge-

ordneten Bahnen ablaufen zu lassen, ist es notwendig im Vorfeld eine verbindliche SOA-Strategie zu definieren und zu kommunizieren. In dieser werden die mit dem SOA-Vorhaben verbundenen Ziele definiert. Die SOA-Strategie sollte von der Unternehmensführung getrieben werden. Dabei müssen sowohl fachliche, als auch IT-Treiber berücksichtigt werden. Weiterhin sollte eine SOA-Referenzarchitektur vereinbart werden, um die saubere IT-technische Umsetzung der definierten SOA-Strategie gewährleisten zu können. Ein Beispiel für eine solche Referenzarchitektur zeigt Abbildung 4. Um dem gesamten SOA-Vorhaben einen geregelten zeitlichen Ablauf zu geben, ist es überdies notwendig eine Roadmap mit entsprechenden Meilensteinen zu definieren.

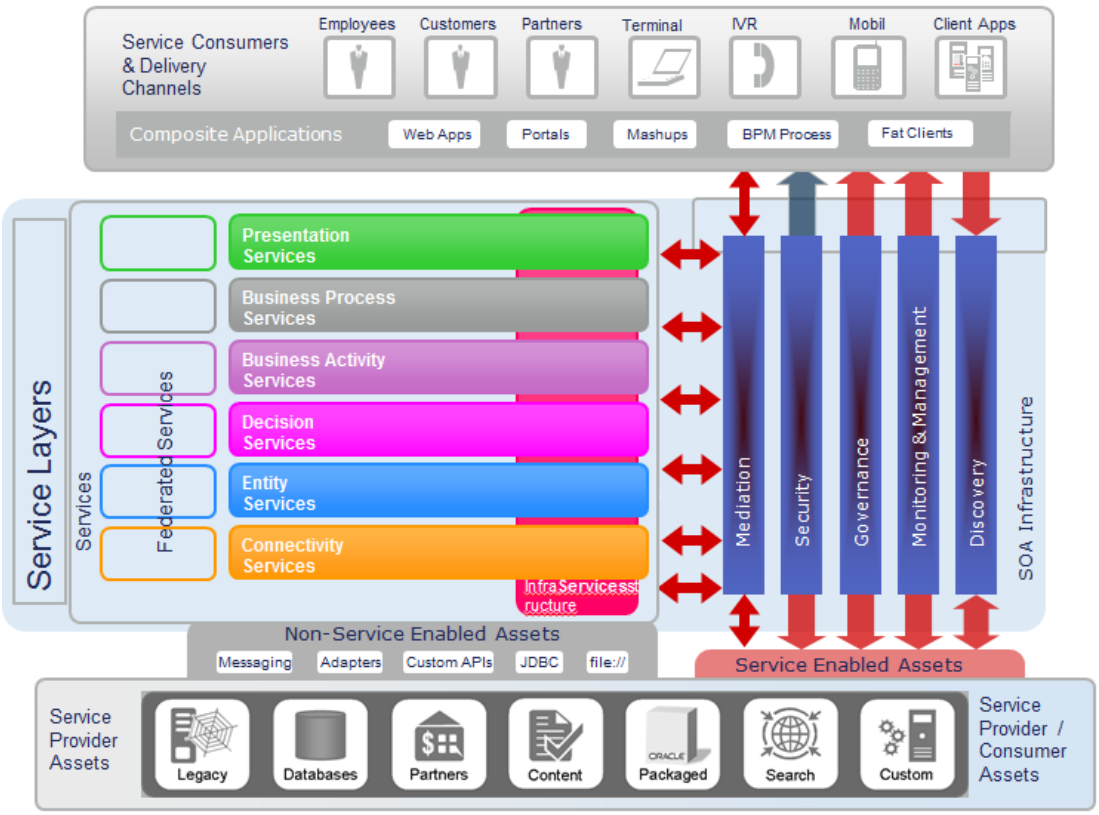

Abbildung 4: Beispiel einer SOA-Referenzarchitektur[9]

Eine SOA-Referenzarchitektur hat, wie die Abbildung zeigt, in der Regel verschiedene Schichten. Auf der untersten Ebene finden sich die im Unternehmen vorhandenen Applikationen beziehungsweise Systeme. In der Ebene darüber findet sich das „Herzstück“: der Service Layer sowie die entsprechende SOA Infrastrukturkomponente. Letztere stellt die Laufzeitumgebung dar und bedient Querschnittsfunktionalitäten für verschiedene Bereiche, wie zum Beispiel die Security oder die Nachrichtenvermittlung („Mediation“). Auf oberster Ebene

[9] Aus [WINT11], S. 6

sind letztlich die Client-Applikationen, auch als „Consumer“ bezeichnet, angesiedelt. Diese interagieren mit den so genannten „Composite Applications“, die wiederrum über die SOA-Infrastruktur mit dem Service Layer und damit auch mit den Backend-Systemen kommunizieren. Bei Composite Applications handelt es sich um zusammengesetzte Applikationen, welche unter Verwendung verschiedener, bereits vorhandener Services aufgebaut werden.

Innerhalb des Service Layers existieren in der Beispielarchitektur verschiedene Serviceschichten. Services aus übergeordneten Serviceschichten können die Funktionalitäten, welche von Services untergeordneter Schichten angeboten werden, verwenden. Dies ist umgekehrt allerdings nicht zulässig. Jede Serviceschicht weist dabei bestimmte Eigenschaften auf, die im Folgenden beschrieben werden.

Connectivity Services (CS)

Services die dieser Serviceschicht angehören sind dafür zuständig Systeme oder Applikationen die nur über proprietäre Schnittstellen verfügen über eine standardbasierte Schnittstelle zur Verfügung zu stellen. Die Systeme sind somit dann „Service-Enabled“. Connectivity Services sind nicht zustandsbehaftet, kurzläufig und werden in der Regel synchron aufgerufen.

Entity Services (ES)

In diese Schicht eingeordnete Services stellen CRUD-Funktionalitäten (Create, Read, Update, Delete) für eine spezifische Entität zur Verfügung. Zudem werden von diesen Transformationen systemspezifischer Datenstrukturen in ein harmonisiertes Datenmodell erledigt. Auch diese Services sind zustandslos, kurzläufig und implementieren eine synchrone Schnittstelle.

Business Activity Services (BAS)

Business Activity Services bilden in der Regel einen atomaren Geschäftsvorfall ab. Im Kontext eines Geschäftsprozesses wäre das ein spezifischer Schritt innerhalb des Prozesses. Business Activity Services sind, ebenso wie die Decision Services, in der Regel nicht zustandsbehaftet, kurzläufig und als synchrone Services implementiert.

Decision Services (DS)

Bei den Decision Services handelt es sich um spezielle Ausprägung von Business Activity Services. Services dieser Schicht fassen Geschäftsregeln zusammen und sind für deren Ausführung zuständig.

Business Process Services (BPS)

Bei den Business Process Services handelt es sich um Services die einen Teil eines Geschäftsprozesses oder auch den gesamten Prozess abbilden. In den Prozessverlauf können menschliche Akteure involviert werden. Business Process Services können entweder langläufig, beispielsweise wenn menschliche Akteure beteiligt werden, oder kurzläufig sein. In Abhängigkeit davon sind Services dieser Schicht zustandsbehaftet oder nicht. In der Regel sind diese Services als asynchrone Services implementiert.

Presentation Services (PS)

Services der Presentation Services-Klasse stellen spezielle Funktionalitäten für die Präsentation in den Client-Applikationen und deren Anzeige zur Verfügung. Dies können beispielsweise Dashboards sein, mit deren Hilfe das fachliche Monitoring von Business Process Services erfolgen kann. Sie sind in der Regel nicht zustandsbehaftet, kurzläufig und haben in der Regel eine synchrone Schnittstelle.

Die skizzierten Serviceschichten sind an dieser Stelle, so wie die gesamte hier vorgestellte Referenzarchitektur, nur als eine mögliche Alternative zu sehen. Es gibt kein Pauschalrezept dafür, wie Services tatsächlich kategorisiert werden sollten und es gibt auch nicht die eine richtige Referenzarchitektur. Die Struktur ist stark davon abhängig, wie sich das konkrete Szenario darstellt, also zum Beispiel der organisatorische Aufbau des Unternehmens, in Verbindung mit den konkreten Zielen des SOA-Vorhabens. Ausgehend von diesen Faktoren muss die SOA-Referenzarchitektur, unter Verwendung von Best Practices, individuell gestaltet und aufgebaut werden.

Bei konsequenter Umsetzung einer Service-orientierten Architektur lassen sich die Probleme und Risiken, die in den klassischen Integrationsansätzen in Verbindung mit proprietären Schnittstellen bestanden, weitestgehend minimieren.

Fragestellungen, wie zum Beispiel Mainframe-Systeme „service-enabled“ werden können, führen jedoch auch im Kontext Service-orientierten Architekturen immer wieder zu Problemen. Meist gibt es in solchen Legacy-Systemen keine saubere Trennung von Verantwortlichkeiten, sodass Masken- und Geschäftslogik häufig vermischt sind. Dies ist aber ein inhärentes Problem von Legacy-Anwendungen generell, das selbst durch die besten Konzepte oder Technologien nicht ganz beseitigt werden kann.

In der Zielarchitektur, welche über die SOA-Referenzarchitektur vorgegeben ist, sind die einzelnen Systeme und Komponenten lose miteinander gekoppelt und mittels standardbasierter Schnittstellen exponiert. Idealerweise existieren keine redundanten Funktionalitäten oder Services mehr. Die Applikationssilos sind aufgebrochen worden. Um das Entstehen neuer Silos, in diesem Fall von SOA-Silos die beispielswiese durch die Entwicklung redundanter Services entstehen können, zu verhindern, müssen organisatorische Maßnahmen im Rahmen einer funktionierenden SOA-Governance etabliert und gelebt werden.

2.3 SOA und User Interfaces (UI)

Nachdem der vorherigen Abschnitt sich hauptsächlich mit der „SOA-fizierung“ der Backend-Applikationen beschäftigt hat, wird es im Folgenden darum gehen, was SOA beziehungsweise ein Backend, das im Sinne der Serviceorientierung aufgebaut wurde, für die Client-Applikationen bedeutet.

Aus Sicht der Client-Applikationen wurde bisher immer direkt mit den Backend-Systemen kommuniziert. Wie bereits die SOA Referenzarchitektur in vorherigem Abschnitt visualisiert (vergleiche Abbildung 4), ist dies so nicht mehr vorgesehen. Die Kommunikation mit den Backend-Systemen funktioniert in der Service-orientierten Welt ausschließlich über die entsprechend zur Verfügung gestellten Services. Auf den ersten Blick müssten zu diesem Zweck nur die bisherigen Backend-Aufrufe durch die korrespondierenden Service-Calls ersetzt werden. Danach sollte alles funktionieren wie zuvor. Dies ist in Realität leider nicht so leicht umsetzbar, wie es sich in der Theorie anhört.

Die direkte Umstellung auf die Service-Calls scheitert häufig an der Art der Datenverarbeitung und dem Interaktionsmuster, dass für die Kommunikation zwischen Clients und den Backend-Systemen verwendet wird. Klassischerweise

sammeln Client-Applikationen Daten, zum Teil über mehrere Seiten, beziehungsweise Masken, hinweg, und leiten diese an das Backend für die Verarbeitung weiter. Da diese Kommunikation in der Regel synchron erfolgt, erhält der Client innerhalb kürzester Zeit eine entsprechende Antwort und kann in Abhängigkeit von dieser auf die nächste Maske springen. Muss eine Client-Applikation nun im Zuge der SOA-fizierung beispielsweise einen Business Process Service aufrufen, funktioniert dies unter der Voraussetzung, dass dieser vollautomatisch abläuft und innerhalb kürzester Zeit antwortet. Dieser Sachverhalt ist in Abbildung 5 dargestellt.

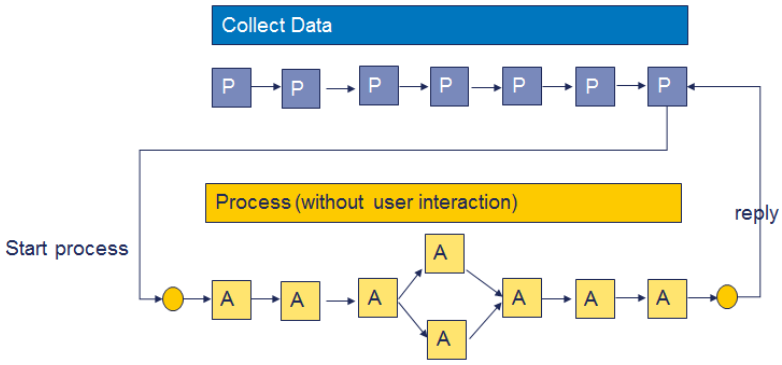

Abbildung 5: UI-Kommunikation mit kurzläufigem BPS[10]

Dies ist aber häufig nicht der Fall. Wie im vorherigen Abschnitt angemerkt, können solche Services langläufig sein und nicht vollautomatisch ablaufen, da beispielsweise das Eingreifen eines menschlichen Akteurs erforderlich ist. Für die Client-Applikation bedeutet dies, dass eine Antwort unter Umständen mehrere Tage dauern kann. In einem solchen Fall ist das Weiterarbeiten in der Client-Applikation nicht mehr möglich, da es hier aufgrund des synchronen Aufrufs zu einem Timeout kommen würde. Zudem könnte ohne die Antwort des Services nicht auf die nächste Seite verzweigt werden.

Das Beispiel zeigt, dass die Kommunikation zwischen Client-Applikation und Service in einem solchen Szenario nicht mehr synchron erfolgen kann, sondern asynchron erfolgen muss. Wie aber ist es in einem solchen Szenario, bei dem der Service für die weitere Verarbeitung Input von einem Benutzer erwartet, möglich, dass der Service mit dem Client beziehungsweise mit einem konkreten Benutzer interagieren kann? Die Lösung hierfür ist, dass der Service Aufgaben (engl. „Tasks“) für einen bestimmten Benutzer oder eine Gruppe von Benutzern

[10] Aus [WINT11], S. 10

erzeugt und mit der weiteren Verarbeitung wartet bis diese bearbeitet wurden. Die Tasks werden an zentraler Stelle in einem so genannten Postkorb (engl. „Inbox“) gesammelt und können von Benutzer mit einer entsprechenden Applikation, bezeichnet als Inbox- oder auch Tasklist-Applikation, bearbeitet werden. Abbildung 6 zeigt diesen Inbox-getriebenen Ansatz. Bei der Verwendung dieses Ansatzes sollte darauf geachtet werden, dass die Inbox und die zugehörigen Services als zentraler Bestandteil der SOA Referenzarchitektur zu implementieren sind.

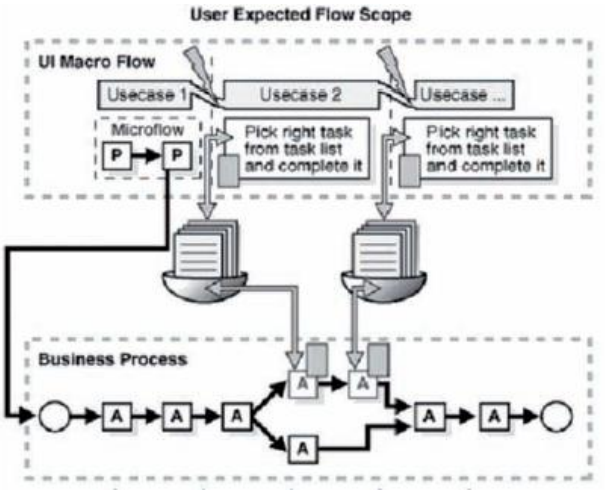

Abbildung 6: Inbox-getriebener Ansatz[11]

Mit Hilfe der Inbox ist es dem Server somit möglich mit einem konkreten Benutzer zu interagieren.

Aus Benutzersicht ist die Umstellung auf ein asynchrones Interaktionsmuster insofern gewöhnungsbedürftig, dass der Service nicht postwendend antwortet. Menschen arbeiten gewohntermaßen am liebsten synchron und erhalten dabei ein direktes Feedback vom Interaktionspartner. Aus Sicht eines Services oder Prozesses ist ein menschlicher Akteur jedoch ein asynchroner Service der aufgerufen wird. Der Service wartet mit der weiteren Verarbeitung dann solange bis die offene Anfrage durch den Menschen beantwortet wurde. Hierbei ergibt sich ein Konflikt bezüglich der bevorzugten Interaktionsstile. Bei einer taskbasierten Interaktion zwischen Service und Benutzer kommt es für den Benutzer immer wieder zu Unterbrechungen, weil zunächst anstehende Tasks abgeschlossen werden müssen, bevor die eigentliche Bearbeitung des Anwendungsfalles fortgesetzt werden kann. Dies ist auch in Abbildung 6 illustriert. Die Er-

[11] Aus [MSOA09], „Teil 10 – SOA und Benutzeroberflächen“, S. 9

zeugung von Tasks erfolgt meist nicht in einer definierten Reihenfolge, was beispielsweise durch Unterschiede im Design von Services oder deren Laufzeitverhalten begründet ist. Für den Benutzer bedeutet dies, dass in der zentralen Inbox in der alle Tasks aller Services auflaufen, zunächst die zum aktuellen Anwendungsfall passende Aufgabe identifiziert werden muss, bevor diese bearbeitet werden kann. Ein Lösungsansatz für die in diesem Absatz angeführten Probleme wird im nächsten Abschnitt vorgestellt.

Bei der Umstellung der IT-Systemlandschaft auf eine serviceorientierte Architektur wird in den meisten Fällen nur über die Backend-Applikationen nachgedacht. Was aber bedeutet ein Service-orientierter Ansatz für die Client-Applikationen? In Client-Applikationen ist es häufig so, dass sehr viel Ablauflogik eines Geschäftsprozesses in Form der Seitenflusslogik implementiert ist. Die Geschäftsprozesse werden in einer SOA allerdings bereits serverseitig in Form eines Business Process Service abgebildet. Hier ergeben sich Redundanzen, die bei Änderungen sowohl Anpassungen am Prozess, als auch an der Client-Applikation bedeuten. Hierbei wird die durch den SOA-Ansatz scheinbar neu gewonnene Agilität wieder relativiert. Hinzu kommt noch, dass es heute meist mehrere redundante Client-Applikationen gibt, welche für dieselben Anwendungsfälle aber für unterschiedliche Vertriebskanäle, beispielsweise Web- und Mobile-Clients, verwendet werden.

Diese Ausführungen zeigen, dass es im Zuge der SOA-Orientierung wichtig ist, auch die Client-Applikationen in die Überlegungen mit einzubeziehen. In einem Artikel von Dr. Werner Steck zum Thema SOA Frontends[9], bei dem es um das Thema der Konsolidierung von Client-Applikationen im Rahmen der SOA-Umstellung geht, wird der in Abbildung 7 visualisierte Ansatz vorgestellt. Hierbei geht es im Kern darum, auf Client-Ebene ein so genanntes „Dialog-Repository“ zu definieren, welches wiederverwendbare Dialoge enthält. Vom Grundprinzip her sollen diese Dialoge so aufgebaut sein, dass sie unabhängig in unterschiedlichen Kontexten wiederverwendbar, beliebig kombinierbar und unabhängig von der verwendeten Frontend-Technologie einsetzbar sind. Für nähere Details sei an dieser Stelle auf den entsprechenden Artikel verwiesen.

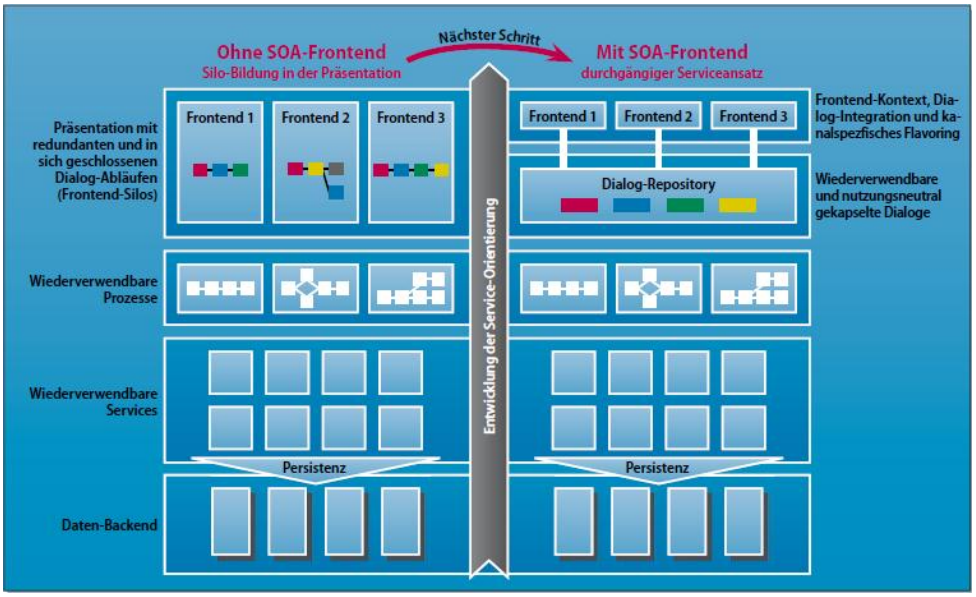

Abbildung 7: SOA-Frontends[12]

2.4 UI Mediator-Pattern

Das von Thomas Erl formulierte Pattern des UI Mediators[13] adressiert das im vorausgegangenen Abschnitt skizzierte Problem der unterschiedlichen Interaktionsmusterpräferenzen von Menschen und Services.

In seinem Pattern schlägt Erl vor, einen so genannten „Mediator Service“ zu etablieren, der dafür zuständig ist dem Benutzer durch konstantes Feedback zu suggerieren, dass der eigentlich asynchrone Serviceaufruf in gewohnter Art und Weise synchron ausgeführt wird. Abbildung 8 skizziert die Idee dieses Pattern.

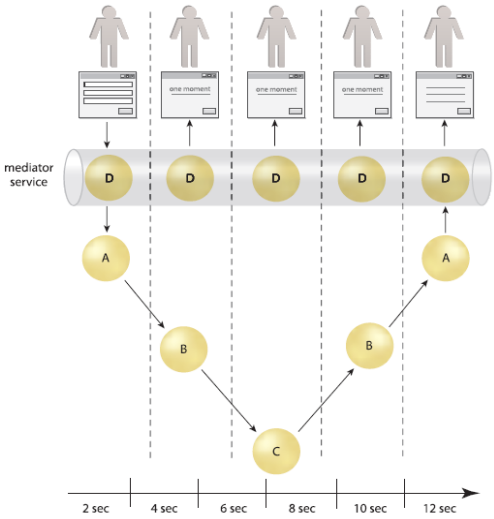

Abbildung 8: UI Mediator nach Erl[14]

[12] Aus [STEW10], S. 2
[13] Siehe dazu auch [ERLT091], S. 366-371
[14] Aus [ERLT091], S. 368

Erl beschreibt die Implementierung relativ abstrakt. Eine konkrete Implementierungsvariante dieses Pattern ist der Service Human Interaction Layer (SHIL)[15]. Der SHIL ist dafür zuständig dem Benutzer den gewohnten synchronen Maskenfluss zu simulieren und für ihn die Interaktion mit dem Postkorb zu übernehmen. Abbildung 9 zeigt, wie der SHIL das Zusammenspiel zwischen GUI und dem Geschäftsprozess synchronisiert.

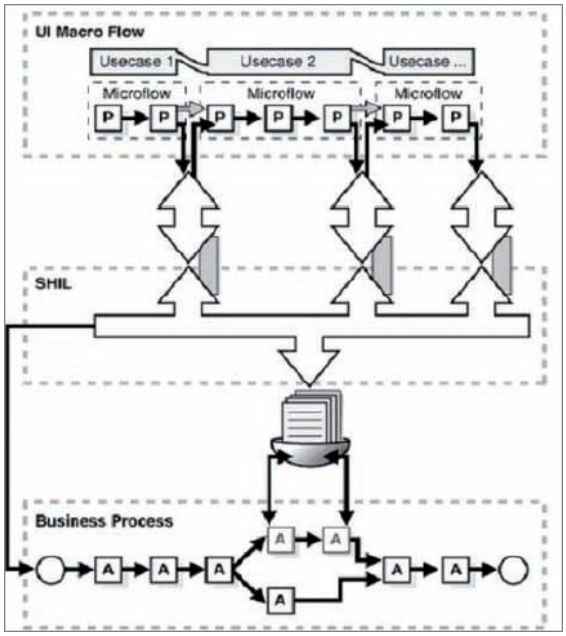

Abbildung 9: SHIL als Implementierungsvariante des UI Mediator Pattern[16]

Kernkomponenten des SHIL sind ein Mirko- und ein Makro-Flow Controller. Der Mikro-Flow Controller ist für die Steuerung der Maskenflüsse auf der Client-Seite zuständig. Ein typischer Vertreter für Mikro-Flow Controller aus dem JEE-Umfeld ist beispielsweise das FacesServlet, das bei Java Server Faces (JSF) als UI Controller eingesetzt wird. Einzelne Maskenflüsse sind anwendungsfallbezogen definiert und besitzen einen logischen Namen über welchen sie angestoßen werden können. Der Makro-Flow Controller ist für die Koordination der Kommunikation zwischen Geschäftsprozess und GUI zuständig. Stellt der Geschäftsprozess in Form eines Tasks eine Benutzeranfrage, wird der Makro-Flow Controller durch dieses Ereignis aktiviert. Dieser teilt dem Mikro-Flow Controller den logischen Flow-Namen mit, der anhand dessen entscheidet welcher Maskenfluss gestartet werden soll. Der logische Name des nächsten Maskenflusses sowie weitere Informationen, die für die Korrelation zwischen serverseitigem Geschäftsprozess und einem spezifischen Client benötigt werden, gibt der Ge-

[15] Nähere Informationen in [MSOA09], S. 10 f.
[16] Aus [MSOA09], S. 10

schäftsprozess als Header-Informationen seines Aufrufs an den Client mit. Die Zwischenschicht, bestehend aus Mikro- und Makro-Flow Controller, darf keine Geschäfts- oder sonstige Anwendungslogik enthalten.

Mit Hilfe des UI Mediator-Patterns beziehungsweise einer konkreten Implementierung SHIL, ist es möglich menschliche Akteure und automatisierte Geschäftsprozesse effektiv miteinander interagieren zu lassen. Allerdings wird der Mensch hierbei durch den Geschäftsprozess gesteuert. Die Akteure haben nicht die Möglichkeit selbst zu bestimmen, wie eine Aufgabe zu lösen ist, das dies durch den Prozess vorgegeben ist. Rücksprünge in Prozessen sind so beispielsweise auch nur dann möglich, wenn dies explizit vorgesehen und modelliert ist. Alle Ausnahme im Geschäftsprozess zu modellieren wird in den meisten Fällen nicht gelingen und führt zu Unübersichtlichkeit sowie einer erhöhten Komplexität.

Um diesem Problem zu begegnen werden adaptive Prozesse benötigt, die nicht den Weg vorgeben wie das Ziel des Geschäftsprozesses, beziehungsweise eines Teilschritts, erreicht werden kann, sondern bei denen das Ziel selbst im Mittelpunkt steht. Wie dieses erreicht wird, liegt in der Verantwortung des menschlichen Akteurs. In diesem Zusammenhang werden Menschen als so genannte „Knowledge workers“[17] gesehen. Dieses, noch sehr junge Themengebiet der adaptiven Prozesse wird auch unter dem Oberbegriff Adaptive Case Management (ACM) subsumiert, auf das an dieser Stelle nicht näher eingegangen werden soll.

[17] Vergleiche hierzu [SWKD10] Kapitel 1, S.8 ff.

3 Anforderungen

Im diesem Kapitel werden die Anforderungen an das zum implementierende GHIA-Framework aufgeführt, welche die unterliegende Task-Engine Implementierung abstrahiert. Als Grundlage für die nachfolgend aufgeführten Anforderungen dienen vor allem die in der WS-HT-Spezifikation[18] beschriebenen Vorgaben an die Schnittstelle zu einer Task-Engine, über die Tasks beispielsweise angelegt oder abgefragt werden können. Hierbei wird allerdings nur ein Ausschnitt dessen was in der WS-HT Spezifikation beschrieben ist, für die Implementierung der Adapterschicht berücksichtigt. Hauptgrund hierfür ist, dass die im Rahmen dieser Arbeit verwendeten BPM-Plattformen nicht alle von der WS-HT Spezifikation geforderten Funktionalitäten unterstützen. Hierauf wird in Kapitel 4 noch einmal detailliert eingegangen.

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen an die Schnittstelle lassen sich in die Kategorien Management, Abfrage und Administration von Tasks einteilen. In den folgenden Abschnitten werden die in diesen Bereichen notwendigen Funktionen kurz aufgeführt.

3.1.1 Taskmanagement

Im Bereich Taskmanagement geht es um Funktionen, die dem Bearbeiten von Tasks beziehungsweise aller zu einem Task zugehörigen Meta- und Detailinformationen dienen.

Referenz	Beschreibung
FM 10	Bearbeiten eines Task (Starten, Zuweisen, Neuzuweisen, Zuweisung aufheben, Unterbrechen, Überspringen, Beenden)
FM 20	Ändern der Taskpriorität
FM 30	Neuanlegen, Ändern und Löschen von Attachments
FM 40	Neuanlegen, Ändern und Löschen von Kommentaren

Tabelle 1: Anforderungen Task-Management

3.1.2 Abfrage von Taskdaten

Im Bereich Abfrage von Taskdaten werden Funktionen definiert, die für Anzeige und Suche von Taskinformationen benötigt werden.

[18] [WSHT], S. 71-98

Referenz	Beschreibung
FQ 10	Abfrage aller Attachments zu einem Task
FQ 20	Abfrage aller Kommentare zu einem Task
FQ 30	Abfrage der Taskhistorie
FQ 40	Abfrage von Taskdetaildaten
FQ 50	Abfrage aller Task-Instanzdaten
FQ 60	Gefilterte Abfrage von Task-Headerinformationen (zum Beispiel Benutzer, Anfangsdatum, Enddatum, etc.)
FQ 70	Synchronisation verschiedener Postkörbe

Tabelle 2: Anforderungen Taskdaten-Abfrage

Die Anforderung FQ 50 ist hier als Zusammenfassung der Anforderungen FQ 10 – FQ 40 zu verstehen. Anstelle von vier Einzelabfragen können die Informationen mit Hilfe dieser Operation mittels einer Anfrage abgerufen werden. Diese mutmaßlich redundante Funktionalität ist in der WS-HT-Spezifikation definiert[19]. Vorteil dieser Definition ist die Flexibilität bei der Abfrage von Taskdetails. Denn je nach fachlichem Kontext werden nicht immer alle Daten benötigt. So ist also die Steuerung des Datenvolumens bei der Übertragung der angefragten Daten möglich. Dies spielt auch bezüglich der Unterstützung möglichst vieler Clienttechnologien eine Rolle.

3.1.3 Administration von Tasks

Im Bereich Administration von Tasks werden spezielle administrative Verwaltungsaufgaben definiert, welche von Business-Administratoren durchgeführt werden sollten.

Referenz	Beschreibung
FA 10	Aktivieren eines Task
FA 20	Administrative (Neu-)Zuweisung eines Tasks

Tabelle 3: Anforderungen Task-Administration

3.2 Qualitätsanforderungen

Die Qualitätsanforderungen, häufig auch als nicht-funktionale Anforderungen bezeichnet, an das zu implementierende Framework sollen sicherstellen, dass das Framework ein Höchstmaß an Flexibilität und Kompatibilität erreicht. Damit einhergehend ist schlussendlich die Akzeptanz bei den Benutzern.

[19] [WSHT], S. 78

Referenz	Beschreibung
QA 10	Kompatibilität: Unterstützung verschiedener Client-Technologien (Web Clients, Rich Clients, Mobile Clients)
QA 20	Kompatibilität: Unterstützung verschiedener Human Workflow-Engines
QA 30	Portabilität: Das Framework muss auf allen gängigen JEE5-Applikationsservern und auch auf einem Servlet-Container, wie beispielsweise Tomcat, gleichermaßen lauffähig sein
QA 40	Flexibilität: Anpassbarkeit beziehungsweise Erweiterbarkeit des Frameworks
QA 50	Security: Sicherstellen Authentizität/Autorisierung von Benutzern

Tabelle 4: Qualitätsanforderungen

3.3 Abnahmekriterien

In diesem Abschnitt werden die Abnahmekriterien definiert, die in der Validierungsphase zur Überprüfung der definierten Anforderungen (siehe Abschnitt 3.1) dienen. Zu diesem Zweck werden Testfälle definiert (siehe Tabelle 5), welche die Anforderungen operationalisieren. Das heißt, dass die Anforderungen mit konkreten Anwendungsfällen hinterlegt werden, welche dann schließlich der Überprüfung der implementierten Lösung dienen. Da wiederrum die Testfälle auf Grundlage der Anforderungen definiert wurden, werden durch die Überprüfung der in den Abnahmekriterien definierten Testfälle auch gleichzeitig die jeweils zugrundeliegenden Anforderungen verifiziert. Der Zusammenhang zwischen den Abnahmekriterien und den Anforderungen ist in Tabelle 6 dargestellt.

Referenz	Beschreibung	Verantwortlichkeit	Erwartetes Ergebnis
TF 10	Zuordnen eines bestehenden, nicht zugeordneten Task zu einem Bearbeiter	Business Administrator, Potential Owner	Der Task wird dem gewählten Bearbeiter zugeteilt und wechselt in den Zustand „Reserved“. Der Task taucht in der Taskliste des Bearbeiters auf.
TF 15	Abschließen eines bestehenden Tasks	Actual Owner	Der Task wird abgeschlossen, wechselt in den Status „Completed“ und der vom Bearbeiter gewählte Outcome wird auf dem Task gesetzt.
TF 20	Hinzufügen eines Attachments und eines Kommentars zu einem bestehenden Task	Actual Owner	Der Kommentar und das Attachment werden hinzugefügt und können entsprechend eingesehen (Über Taskdetails oder Einzelabfrage) werden.
TF 25	Entfernen eines Attachments und eines Kommentars von einem Task	Actual Owner	Der Kommentar und das Attachment werden entfernt. Der Anwender bekommt eine Erfolgsmeldung angezeigt.
TF 30	Abfragen der zu	Actual Owner	Die angeforderten Daten werden

	einem Task zugehörigen Daten aus dem fachlichen Kontext		zurückgeliefert und können eingesehen werden.
TF 35	Abfragen aller zu einem Actual Owner zugehörigen Tasks. Die Herkunft der Tasks (bezogen auf die Task-Engine) soll hierbei keine Rolle spielen	Actual Owner, Business Administrator	Alle Tasks werden entsprechend angezeigt und stehen zur Verarbeitung bereit. Die Bearbeitung der Tasks ist für den Benutzer transparent und muss ohne Kenntnis unabhängig von der Task-Engine erfolgen.
TF 40	Abfragen der Taskhistorie zu einem bereits beendeten Task	Business Administrator	Die Taskhistorie wird zurückgeliefert.
TF 45	Aktivieren eines neu angelegten Tasks	Business Administrator	Der Task wechselt den Zustand zu „Ready“.
TF 50	Neuzuweisen eines bereits zugewiesenen Tasks zu einem Benutzer beziehungsweise einer Gruppe.	Business Administrator	Der Task wechselt seinen Zustand zu „Reserved“ und ist durch für den/die potentiellen Bearbeiter sichtbar. Für den bisherigen Bearbeiter ist der Task nicht mehr sichtbar.
TF 55	Ändern der Taskpriorität	Actual Owner, Business Administrator	Die Priorität des Tasks wird erhöht. Dies wirkt sich auf die Sortierung der Tasks in der Übersichtsliste aus.
TF 60	Wiederabgeben der Taskverantwortung	Actual Owner	Der Task wechselt in den Zustand „Created“, der aktuelle Taskbearbeiter wird zurückgesetzt.

Tabelle 5: Abnahmekriterien

Die in Tabelle 5 festgelegten Testfälle beziehen sich auf die funktionalen Anforderungen des GHIA-Frameworks. Diese sollen mit Hilfe regressionsfähiger, automatisierter Tests verifiziert sowie im Rahmen eines PoC hinsichtlich ihrer Funktionalität geprüft werden.

Der Zusammenhang zwischen den funktionalen Anforderungen und den Abnahmekriterien wird in Tabelle 6 aufgezeigt, wobei ein Testfall mehrere Anforderungen adressieren kann.

Referenz Abnahmekriterium	Referenz Anforderung(en)
TF 10	FM 10, FQ 60
TF 15	FM 10
TF 20	FM 30, FM 40, FM 60, FQ 10, FQ 20, FQ 50
TF 25	FM 30, FM 40
TF 30	FQ 40

TF 35	FQ 60, FQ 70
TF 40	FQ 30, FQ 70
TF 45	FA 10
TF 50	FA 20
TF 55	FM 20
TF 60	FM 10

Tabelle 6: Zuordnung Testfälle zu Anforderungen

Im Rahmen der Validierung werden ebenfalls die Qualitätsanforderungen überprüft. Zur deren Überprüfung werden folgende Tests durchgeführt:

Referenz	Beschreibung	Referenz Anforderung(en)
TQ 10	Deployen auf unterschiedlichen Laufzeitumgebungen (Oracle Weblogic und Apache Tomcat)	QA 30
TQ 20	Anbinden verschiedener Task-Engines (Oracle BPM Suite und Activiti BPM Platform)	QA 20, QA 40
TQ 30	Interagieren mit dem Prototypen über unterschiedlicher Client-Applikationen beziehungsweise über verschiedene Schnittstellen	QA 10
TQ 40	Verbieten des Zugriffs durch nicht berechtigte Benutzer	QA 50

Tabelle 7: Zuordnung Testfälle Qualitätsanforderungen zu Anforderungen

3.4 Definition des Proof of Concept Szenarios

Im PoC wird anhand eines beispielhaften Szenarios überprüft, ob die konzipierte und prototypisch implementierte Software die an sie gestellten Anforderungen erfüllt. In diesem Abschnitt wird das Beispielszenario, anhand dessen die Verifikation des GHIA-Frameworks erfolgen wird, beschrieben. Im Beispielszenario handelt sich um einen vereinfachten Schadenregulierungsprozess des imaginären Versicherungsunternehmens Copital AG.

Der Schadenregulierungsprozess der Copital-Versicherung läuft aktuell nicht optimal. Die Kunden sind aufgrund der langen Bearbeitungszeit für einen Schadensfall unzufrieden. Auch die Mitarbeiter der Coptial AG sind mit den aktuellen Arbeitsabläufen nicht glücklich. Mitarbeiterbefragungen haben ergeben, dass das Hauptproblem die Koordination zwischen den am Prozess beteiligten Akt-

euren (Call-Center-Agents, Sachbearbeiter, externe Sachverständige) ist. Während der Bearbeitung eines Versicherungsfalles kommt es immer wieder zu Missverständnissen, da die Zuständigkeiten unklar sind. Weiterhin mangelt es im aktuellen Prozess an Transparenz. Die Mitarbeiter haben Schwierigkeiten den Überblick über die von ihnen bearbeiteten Schadensfälle und deren Status zu behalten.

Der CIO (Chief Information Officer) wird von der Geschäftsführung beauftragt, den Schadenregulierungsprozess in enger Zusammenarbeit mit der Fachabteilung hinsichtlich seiner Schwächen zu analysieren, diesen zu dokumentieren und schließlich zu optimieren. Die Geschäftsführung sieht dieses Projekt als Pilotprojekt für eine unternehmensweite SOA- und BPM-Initiative. Im Erfolgsfall sollen die im Projektverlauf gesammelten Erfahrungen konserviert werden. Diese sollen das Fundament für eine konkrete Strategie mit entsprechenden Vorgehensweisen und Techniken sein, die im Anschluss unternehmensweit ausgerollt werden kann, um auch in anderen Unternehmensbereichen von solchen Optimierungen profitieren zu können und sich so auf lange Sicht Wettbewerbsvorteile gegenüber den Mitbewerben am Markt zu verschaffen.

Die Analyse ergibt, dass die Hauptschwächen des Schadenregulierungsprozesses Medienbrüche und lange Liegezeiten sind. Der CIO und die Fachabteilung erörtern zunächst gemeinsam den Anwendungsfall „Schadensfall bearbeiten“, um einen ersten Eindruck über alle hieran beteiligten Akteure zu gewinnen und um sich der Tätigkeiten dieses Anwendungsfalles bewusst zu werden.

Use Case „Schadensfall bearbeiten“

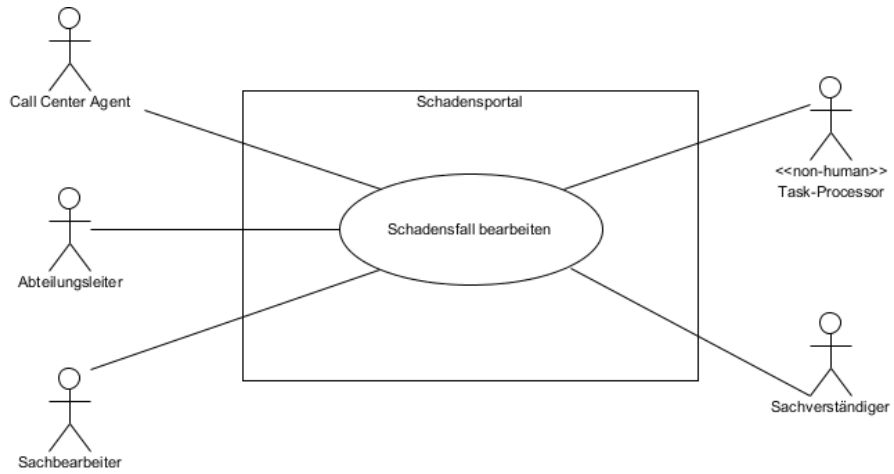

Abbildung 10: Use Case Diagramm Schadensregulierungsprozess

Für den Anwendungsfall werden vereinfacht die in Abbildung 10 dargestellten Akteure identifiziert. Der Ablauf des Anwendungsfalles wird wie folgt beschrieben: Ein Call Center Agent nimmt die telefonische Schadensmeldung des Kunden im Schadensportal, einer webbasierten Applikation, der Versicherungsgesellschaft auf. Die Portalsoftware weist den Fall automatisch der Gruppe „Schadensregulierung“ zu.

Alle Sachbearbeiter, die der Gruppe „Schadenregulierung“ angehören, sehen den neu angelegten Fall in einer Übersichtsliste und können die Bearbeitung eines Falles übernehmen. Nach der Übernahme eines Falles hat ein Sachbearbeiter immer die Möglichkeit diesen wieder abzugeben. Auch der Abteilungsleiter der Abteilung Schadensregulierungen kann die Zuordnung eines Falles zu einem Sachbearbeiter jederzeit aufheben, beziehungsweise ändern. Dies ist vor allem dann erforderlich, wenn der zugeordnete Mitarbeiter beispielsweise im Falle von Krankheit oder Urlaub nicht in der Lage ist, den Fall in einer für den Kunden angemessenen Zeit zu bearbeiten.

Ein Sachbearbeiter der einen spezifischen Schadensfall bearbeitet kann einen externen Sachverständigen beauftragen den Schadensfall zu überprüfen, wenn ein Zweifel an der Validität der Schadensmeldung besteht oder bei einer Schadenssumme die über einer Grenze von 2500 € liegt. Der externe Sachverständige kann die ihm zugewiesenen Fälle über das Versicherungsportal, welche als Webapplikation implementiert ist, oder über eine entsprechende Mobile-Client Applikation einsehen und die Bearbeitung aufnehmen. Die Bearbeitung beinhaltet die Dokumentation der Vorgehensweise sowie letztendlich die Bereitstellung des Gutachtens.

In Abhängigkeit davon, wie das Urteil des externen Sachverständigen ausgefallen ist, sofern dieser eingeschaltet wurde, schließt der Sachbearbeiter die Bearbeitung des Falles ab und legt fest, ob der Kunde den Schaden erstattet bekommt oder nicht.

Soll-Prozess „Schadensregulierung“

Auf Grundlage des Anwendungsfalles erstellt der CIO gemeinsam mit dem Fachbereich einen Soll-Prozess. Dieser wird unter Einsatz der BPMN (Business Process Modelling and Notation) modelliert (siehe Abbildung 11). Der modellierte

Prozess soll zukünftig IT-technisch unterstützt und in einer Process-Engine ausgeführt werden. Der automatisierte Prozess soll weiterhin sicherstellen, dass die Verantwortlichkeiten für bestimmte Aufgaben innerhalb des Prozesses jederzeit klar definiert sind. Hiervon versprechen sich der CIO und die Fachabteilung die gewünschte Transparenz für die Mitarbeiter.

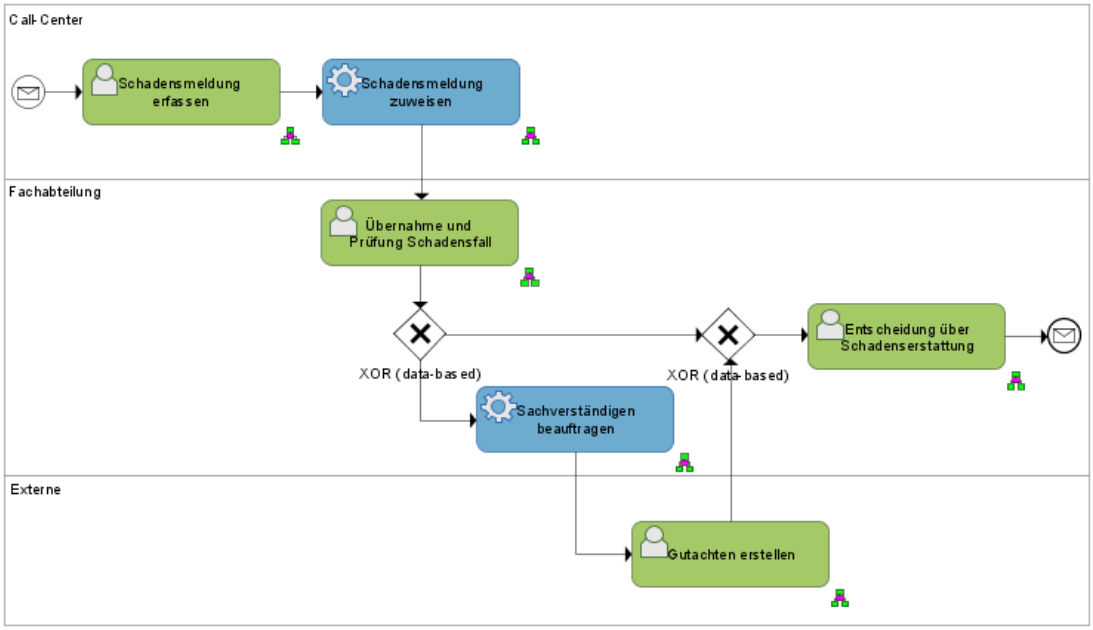

Abbildung 11: Vereinfachter Soll-Prozess Schadenregulierung

In der Pilotprojektphase sollen zwei verschiedene BPM Plattformen evaluiert und hinsichtlich ihrer Einsetzbarkeit bewertet werden. Für den Fall, dass die Pilotphase erfolgreich verläuft und unternehmensweit ähnliche Prozessautomatisierungsprojekte aufgesetzt werden sollen, wird anhand der Ergebnisse der Evaluierung entschieden, welches Produkt als strategische Plattform für weitere Projekte in diesem Kontext eingesetzt werden soll. Der CIO beauftragt die interne Softwareentwicklung damit, die benötigen Software-Artefakte zu konzipieren und umzusetzen.

4 Analyse bestehender Task-Engines ausgewählter BPM-Plattformen

Dieses Kapitel befasst sich mit der Analyse bestehender Task-Engine Implementierungen, am Beispiel zweier BPM-Plattformen. Betrachtet werden hierbei zum einen die Opensource Plattform Activiti und zum anderen mit der Oracle BPM Suite 11g ein Vertreter aus dem kommerziellen Lager. Dabei hat die Analyse der beiden Lösungen zum Ziel, Unterschiede und Gemeinsamkeiten der standardmäßig mitgelieferten Tasklist-Anwendungen sowie der vorhandenen Task APIs beziehungsweise Schnittstellen und deren Architektur aufzuzeigen. Die hierbei gewonnen Erkenntnisse bilden die Grundlage für die nachfolgenden Schritte, Konzeption und Implementierung des GHIA-Frameworks.

In der Analyse wird nicht detailliert auf die Gesamtarchitektur der beiden Plattformen eingegangen oder aber, welche die zugehörigen Komponenten und mögliche Einsatzszenarien für diese sind. Auch wird nicht betrachtet wie die Human Interaction implementiert ist.

4.1 Kurzvorstellung der verwendeten BPM-Plattformen

Die beiden verwendeten BPM-Plattformen unterscheiden sich nicht allein aufgrund der Tatsache, dass es sich bei der Einen um eine Opensource und bei der Anderen um eine kommerzielle Lösung handelt. Es gibt darüber hinaus erhebliche Unterschiede was beispielsweise den Funktionsumfang, die Architektur oder die Integration menschlicher Interaktion in Geschäftsprozesse angeht.

Bei der Opensource BPM Plattform Activiti handelt es sich um eine leichtgewichtige Process-Engine zur Ausführung von Geschäftsprozessen, die in der BPMN 2.0 Notation modelliert worden sind. Die Integration menschlicher Interaktion in die Geschäftsprozesse sowie die Verarbeitung der Tasks werden von der Process-Engine erledigt, es existiert also keine Trennung zwischen Prozessausführungs- und Taskverarbeitungslogik.

Gegenüber der Activiti BPM Plattform handelt es sich bei der Oracle BPM Suite vom Funktionsumfang her um einen deutlich mächtigere Plattform. Neben der Process-Engine, in der sowohl BPMN 2.0- als auch BPEL 2.0-Prozesse ausführbar sind, enthält die Oracle BPM Suite zusätzlich eine Rules-, eine Mediator- so-

wie eine Human Workflow-Komponente. Die Verantwortlichkeiten für die Verarbeitung von Tasks und die Ausführung von Prozessen sind bei Oracle strikt voneinander getrennt („Separation of concerns“). Geschäftsprozesse werden von der Process-Engine und die Tasks von der Human Workflow-Engine verarbeitet.

In der Engine-Architektur der beiden Plattformen zeigt sich ein erster wesentlicher Unterschied was die Human Interaction betrifft: Auf der einen Seite eine Trennung der Verantwortlichkeiten was die Verarbeitung von Tasks und Prozessen angeht, auf der anderen Seite die Erledigung beider Aufgaben durch eine Engine. Die erste Variante ist diejenige, welche von der WS-HT Spezifikation[20] propagiert wird.

Für einen umfangreicheren Vergleich der beiden BPM-Plattformen sei an dieser Stelle auf die im Vorfeld dieser Arbeit entstandene Projektarbeit[21] verwiesen. Darin findet sich eine Gegenüberstellung der beiden Plattformen, beziehungsweise deren Aufbau (vergleiche Kapitel 3) sowie ein detaillierter Vergleich der beiden Human Task-Implementierungen hinsichtlich ihrer Konformität mit den Vorgaben der WS-HT Spezifikation (vergleiche Kapitel 4 und Kapitel 5).

4.2 Die Task-Engines

Die Kommunikation zwischen externen Akteuren, wozu neben den menschlichen Akteuren auch andere Softwaresysteme zählen, und den Task-Engines muss eindeutig definiert sein. Das ist wichtig, denn an der Bearbeitung beziehungsweise Verarbeitung eines Tasks sind in der Regel mehrere Akteure, wie beispielsweise ein den Task initiierender BPMN-Prozess und ein verantwortlicher menschlicher Bearbeiter – unter Umständen sind dies auch mehrere –, beteiligt. Aus diesem Grund sollten die entsprechenden Schnittstellen zu den Task-Engines auf Standards basieren, um eine gute Interoperabilität und einen möglichst hohen Wiederverwendungsgrad gewährleisten zu können.

Die Bearbeitung von Tasks durch menschliche Akteure erfolgt zumeist unter Verwendung spezieller Applikationen. Diese Applikationen werden in der WS-HT Spezifikation als Tasklist-Applikationen bezeichnet, jedoch wird nicht näher

[20] Vergleiche [WSHT], S. 10 ff.
[21] Nähere Informationen siehe [BERS11]

spezifiziert wie diese implementiert sein sollen. Eine solche Applikation wird in der Regel durch die Plattformhersteller mitgeliefert und steht im Anschluss an die Installation zur Verfügung.

Im den nachfolgenden Unterabschnitten wird es darum gehen, Aufbau und Eigenschaften der betrachteten BPM-Plattformen hinsichtlich ihrer Task-Engine Schnittstellen zu untersuchen und deren Spezifika herauszufinden. Weiterhin werden die mitgelieferten Tasklist-Applikationen sowie deren Grundfunktionalitäten und Implementierungsspezifika kurz vorgestellt.

4.2.1 Schnittstellen

Wie einleitend bereits erwähnt, ist es unter anderem aus Gründen der Interoperabilität und Wiederverwendbarkeit wichtig, dass die Task-Engine Schnittstellen auf etablierten Standards basieren. In der WS-HT Spezifikation wird eine SOAP-Schnittstelle[22] definiert, auf deren Grundlage die konkrete Schnittstellenimplementierung als SOAP-Webservice, mit Hilfe einer beliebigen Technologie (Java, .NET, etc.), umgesetzt werden kann.

Oracle BPM Suite

Die Human Workflow-Engine der Oracle BPM Suite bietet zwei verschiedene Schnittstellen an: eine SOAP-Schnittstelle und eine EJB-Schnittstelle. Die Kommunikation erfolgt im Fall der SOAP-Schnittstelle über das etablierte HTTP-Protokoll. Die Kommunikation im Falle der EJB-Schnittstelle läuft entweder über das proprietäre Java-Protokoll RMI (Remote Method Invocation) ab, hierbei wird das so genannte Remote-Interface (Remote EJB) verwendet. Alternativ kann das so genannte Local-Interface (Locale EJB) verwendet werden. Hierbei werden dann simple Java Methodenaufrufe durchgeführt[23]. Die Verwendung des Remote-Interfaces ist immer dann notwendig, wenn sich die aufrufende Client-Applikation auf einem anderen Server, beziehungsweise innerhalb einer anderen Java Virtual Machine (JVM), befindet. Läuft die Client-Applikation auf dem gleichen Server in derselben JVM, können direkte Java-Methodenaufrufe verwendet werden.

[22] Vgl. [WSHT], S. 102-105, S. 114 und S. 120-188
[23] Nähere Details zur EJB-Technologie vgl. [EJB]

Aufgrund der Tatsache, dass bei der EJB-Schnittstelle proprietäre Technologien zum Einsatz kommen, ist die Interoperabilität schlechter als die der SOAP-Schnittstelle. Auf der anderen Seite bietet die EJB-Schnittstelle aber wiederrum leichte Vorteile gegenüber der SOAP-Schnittstelle was die Performance angeht, da hier der Aufwand des Marshalling-/Unmarshalling, also die Umwandlung von Java-Objekten zu XML-Datenstrukturen und umgekehrt, entfällt.

Die Schnittstellen der Oracle BPM Suite bedingen zusätzlich, dass vor der Interaktion mit der Task-Engine eine Authentifizierung des aktuellen Benutzers erfolgt. Hierbei gibt es zwei unterschiedliche Möglichkeiten diese durchzuführen. Einmal ist es möglich, die Authentifizierung mittels Benutzername und des zugehörigen Passworts durchzuführen. Dieser Authentifizierungsmechanismus ist beispielsweise bei der Kommunikation über die EJB-Schnittstelle zu verwenden. Alternativ dazu ist, bei Verwendung der SOAP-Schnittstelle, die Authentifizierung über ein SAML (Security Assertion Markup Language) Security-Tokens möglich. Unabhängig von der verwendeten Authentifizierungsmethode erfolgt die Verifikation der Benutzerinformationen gegen einen konfigurierten Identity Store, was in der Regel das im Unternehmen verwendete Identity Management System (IMS) sein wird[24]. Mit Hilfe der beiden Authentifizierungsmechanismen wird somit sichergestellt, dass nur berechtigte Benutzer mit der Task-Engine interagieren können.

Activiti BPM Plattform

Die Activiti BPM Plattform bietet eine Java- und eine REST-Schnittstelle (Representational State Transfer) an. Als Austauschformat bei der Kommunikation mit der REST-Schnittstelle wird das JSON-Format (JavaScript Object Notation) verwendet. Als Kommunikationsprotokoll dient, wie bei der SOAP-Schnittstelle der Oracle Human Workflow-Engine, ebenfalls das HTTP-Protokoll. Bei Verwendung der Java-Schnittstelle ist zu beachten, dass – wie im Fall der Schnittstelle über Locale EJBs der Oracle BPM Suite – die Activiti BPM Engine und die Client-Applikation in derselben JVM laufen müssen. Läuft die Client-Applikation in einer eigenen JVM, kann die Kommunikation mit der Activiti Process-Engine nur über die REST-Schnittstelle erfolgen.

[24] Vergleiche [BERS11], Abschnitt 4.3.2, S. 28f.

Ähnlich wie im Fall der Oracle Human Workflow-Engine, setzt auch die REST-Schnittstelle der Activiti BPM Plattform voraus, dass sich Anwender zunächst authentifizieren bevor eine Interaktion mit der Activiti Process-Engine erfolgen kann. Als Authentifizierungsverfahren wird in diesem Zusammenhang die „Basic HTTP Authentication“ verwendet. Bei diesem Verfahren werden Benutzername und Passwort als Authentifizierungsinformationen verwendet. Token-basierte Authentifizierungsmechanismen, wie beispielsweise die Authentifizierung über ein SAML-Token, werden bei Activiti nicht unterstützt. Die Verifikation der Benutzerinformationen erfolgt gegen die Activiti-interne Benutzerverwaltung[25]. Bei Verwendung der Java-Schnittstelle ist keine Benutzerauthentifizierung notwendig. Dies ist kein Fehler, sondern soll die Flexibilität bei Anbindung eines bestehenden IMS erhöhen.

4.2.2 Architektur und Aufbau

Nachdem im vorausgegangenen Abschnitt die Schnittstellen zu den Task-Engines im Fokus standen, beschäftigt sich dieser Abschnitt mit dem internen Aufbau beziehungsweise der Architektur. Diese ist nicht Gegenstand der WS-HT Spezifikation, aber für das Verständnis der Konzeption und Implementierung von GHIA notwendig.

Oracle BPM Suite

Die grundsätzliche Architektur der Oracle BPM Suite sowie die zugehörigen Komponenten wurden bereits in der Projektarbeit[26] beschrieben. In den folgenden Ausführungen wird es ausschließlich um die Human Workflow-Engine gehen. Wie Abbildung 12 zeigt ist diese als eigenständige Komponente implementiert und kann über die Unified Runtime mit den anderen Service-Engines, beispielsweise der BPMN-Engine, interagieren. Für externe Client-Applikationen ist die Kommunikation über die im vorherigen Kapitel vorgestellte Schnittstelle möglich.

[25] Vergleiche [BERS11], Abschnitt 4.3.2, S. 29
[26] Vergleiche [BERS11], Kapitel 3, S. 9ff.,

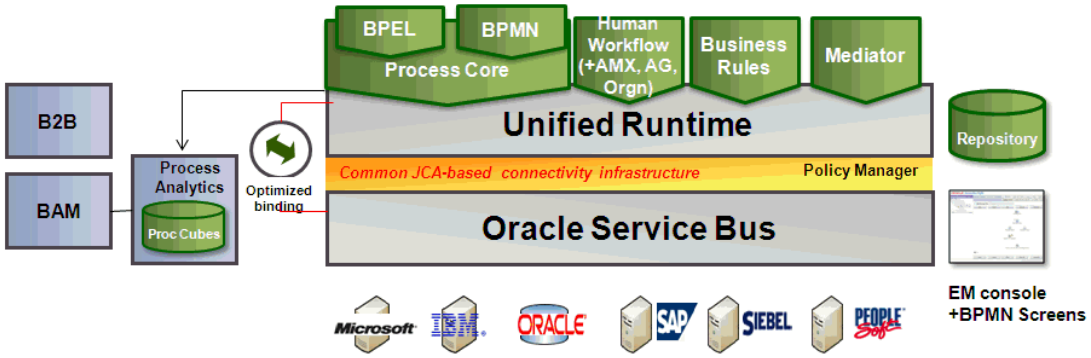

Abbildung 12: Architektur Oracle BPM Suite[27]

Die Oracle Human Workflow-Engine besteht, wie in Abbildung 13 dargestellt, im Kern aus sechs verschiedenen Services. Jeder Service hat hierbei einen klar abgegrenzten Verantwortungsbereich.

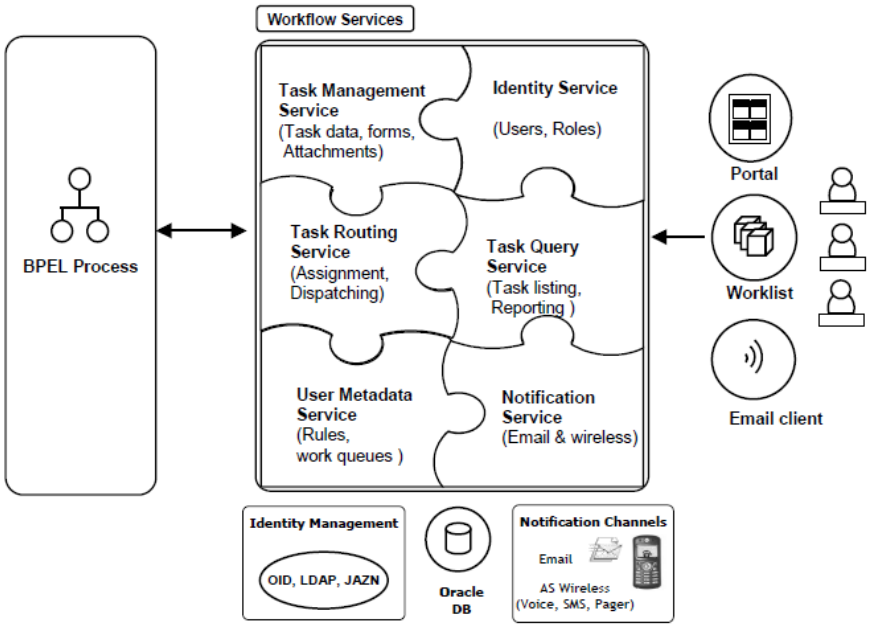

Abbildung 13: Oracle Human Workflow Services[28]

Im Folgenden werden die Funktionalitäten[29] beschrieben, welche die einzelnen Services anbieten:

Der „Task Management Service“, oder auch „Task Service“, ist für das Management des Taskzustandes und die Persistenz der Taskdaten zuständig. Mit Hilfe dieses Services können beispielsweise Tasks abgeschlossen oder zusätzliche Informationen in Form von Attachements oder Kommentaren hinzugefügt werden. Der „Task Routing Services“ beschäftigt sich mit dem der Zuweisung und Weiterleitung von Tasks. Dies ist bezogen auf die in den Task-Metadaten be-

[27] Aus [OSOA02]
[28] Aus [OSOA03]
[29] Angelehnt an [OSOA04], Kapitel 31

schriebenen Routing-Pattern[30] und definierten Zuständigkeiten. Gegenstand sind also die automatischen Zuweisungen und Weiterleitungen. Manuelle Zuweisungen, beziehungsweise Neuzuweisungen, werden mit Hilfe des Task Management Services vorgenommen. Der „User Metadata Service“ dient der Verwaltung von Metadaten für Benutzer, welche für die Bearbeitung von Tasks zugelassen sind. Zu den Metadaten zählen beispielsweise Informationen über Urlaube und entsprechende Vertreterregelungen. Der „Identity Service“ interagiert mit dem Unternehmens-IMS und ist zuständig für die Abfrage von Benutzerinformationen sowie für die Authentifizierung von Benutzern. Mit Hilfe des „Task Query Service“ kann nach Tasks gesucht werden. Die Suche kann unter Verwendung einer Vielzahl von Suchkriterien, wie zum Beispiel dem Taskstatus oder dem Tasktitel, eingeschränkt werden. Über den „Notification Service“ stehen Funktionalitäten bereit, um im Kontext eines Tasks Benachrichtigungen zu versenden, die durch eintretende Ereignisse ausgelöst werden. Beispiele für solche Ereignisse sind Statusübergange oder im das Ablaufen einer Bearbeitungsfrist[31].

Neben diesen sechs Kernservices existieren noch zusätzlich drei spezielle Services, die hier der Vollständigkeit halber ebenfalls erwähnt werden sollen. Dabei handelt es sich um den „Task Report Service“. Dieser ermöglicht fest definierte Reports abzufragen. Beispielsweise gibt es einen Produktivitätsreport, der Auskunft darüber gibt, wie viele Tasks innerhalb einer definierten Zeitperiode durch einen Benutzer oder eine Gruppe abgearbeitet wurden. Des Weiteren existiert noch der „Runtime Config Service“, der die Manipulation von Task-Metadaten ermöglicht und es erlaubt unterschiedliche Darstellungen eines Tasks unter verschiedenen URIs (Unified Resource Identifier) abzulegen. Der letzte Service, der „Evidence Store Service“, ermöglicht die Speicherung digitaler Signaturen und die Verifikation entsprechend signierter Human Tasks.

Alle hier aufgeführten Services können, mit Ausnahme des „Task Routing Service“ und des „Identity Service“, über die im vorherigen Abschnitt beschriebenen Schnittstellen, EJB und SOAP, verwendet werden. Die Verwendung des „Identity Service“ ist nur über die SOAP-Schnittstelle möglich.[32] Der „Routing

[30] Vergleiche auch [BERS11], Kapitel 4.4 „Task Routing“
[31] Vergleiche auch [BERS11], Kapitel 4.5 „Task Escalation“
[32] Vergleiche auch [OSOA04] Kapitel 32, Tabelle 32-1

Service“ wird über keine der beiden Schnittstellen für die externe Verwendung angeboten.

Elementar, in Bezug auf die Verarbeitung von Tasks, sind bei Oracle der „Task Service“ beziehungsweise „Task Management Service“ und der „Task Query Service“. Die übrigen Services sind nicht unwichtig, unterstützen aber eher Spezialanforderungen, die im Sinne der WS-HT Spezifikation, wenn überhaupt, nur eine untergeordnete Rolle spielen.

Activiti BPM Plattform

Bei Activiti ist die Process-Engine auch gleichzeitig die Task-Engine und damit die zentrale Komponente für die Activiti BPM Plattform. Wie Abbildung 14 zeigt, besteht die Engine im Kern besteht aus der Implementierung des BPMN 2.0-Standards, der Process Virtual Machine (PVM), welche die Ausführungskomponente für definierte Prozesse bildet, und den Core Interfaces. Über die Zusatzkomponente „Activiti Spring“ ist es möglich die Activiti Engine mit dem Spring Framework zu integrieren, sodass zum Beispiel in Spring Beans definierte Geschäftslogik direkt aus BPMN 2.0-Prozessen verwendet werden können.

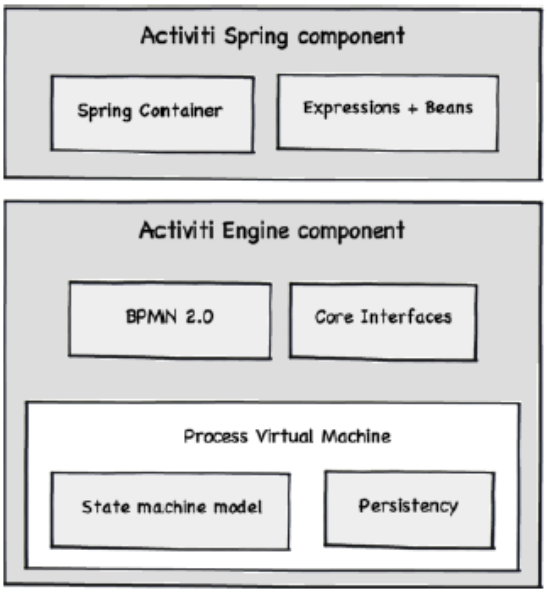

Abbildung 14: Architektur der Activiti Engine[33]

Über die bereits angesprochenen Core Interfaces der Activiti Process-Engine ist die Interaktion von Client-Applikationen mit der Engine möglich. Ähnlich wie bei der Oracle BPM Plattform, bietet die Activiti-Schnittstelle sieben Kernser-

[33] Aus [RADT11], Kapitel 4.1.1, S. 57

vices an. Diese sind in Abbildung 15 dargestellt und werden im Folgenden bezüglich ihrer Funktionalität beschrieben.[34]

Zentrales Element ist die Process-Engine, welche Teile Ihrer Funktionalität über die erwähnten Core Interfaces zur Verfügung stellt. Die für die Engine zentrale Konfigurationsinstanz ist die `ProcessEngineConfiguration`. Für die Konfiguration kann eine spezielle Datei, die `activiti.cfg.xml`, verwendet werden, in der unter anderem Datenbankverbindungsinformationen abgelegt werden.

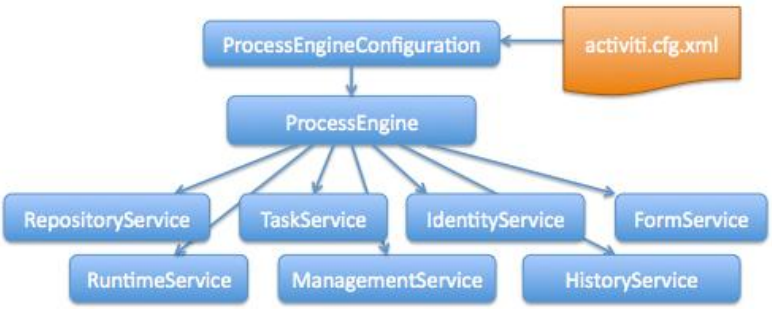

Abbildung 15: Activiti Engine API [35]

Der „Repository Service“ ist für das Deployment-Management verantwortlich. Dieses umfasst Aufgaben, wie beispielsweise die Bereitstellung oder das Löschen von Prozessdefinitionen. Über den „Runtime Service“ ist das Management von Prozessinstanzen möglich. Hierbei ist es möglich neue Prozessinstanzen zu starten sowie Informationen zu laufenden Prozessinstanzen abzufragen. Mit dem „Task Service“ werden Operationen für User Tasks ausgeführt. Beispielsweise können neue Tasks erstellt oder auch bestehende Tasks abgeschlossen werden. Der „Management Service“ kann für Abfragen auf den Activiti-Tabellen und für das Ausführen von Jobs benutzt werden. Jobs werden in BPMN 2.0-Prozessen unter anderem für das Erzeugen von Timer-Events verwendet. Das Abfragen und Management von Benutzer-Authentifizierungsinformationen sowie die Authentifizierung von Benutzern ist über den „Identity Service“ möglich. Der „History Service“ dient der Abfrage von Informationen zu bereits abgeschlossenen Prozessinstanzen und auch zu abgeschlossenen User Tasks. Für die Interaktion mit, von der Activiti Form-Engine generierten, Task Forms zeichnet sich der „Form Service“ verantwortlich. Dieser stellt Funktionalitäten bereit, um

[34] In Anlehnung an [RADT11], Kapitel 4.2, S. 63
[35] Aus [ACTI01], Chapter 5 „Engine API“

beispielsweise alle Daten zu einem Task-Formular abzufragen oder diese zu setzen und so den Task abzuschließen.

Elementar für die Verarbeitung von Human Tasks sind von den hier beschriebenen sieben Kernservices, der „Task Service“ und der „History Service“. „Identity Service“ und „Form Service“ decken ihrerseits Spezialfunktionen bezüglich des Rendering von Task-Forms beziehungsweise der Verwaltung von Benutzerinformationen ab, die allerdings nicht unmittelbar relevant für das Management von Human Tasks und deren Lifecycle sind. Die übrigen Services sind in der Hauptsache für das Management der BPMN 2.0-Prozessbeschreibungen und deren Laufzeitinformationen verantwortlich.

4.2.3 Funktionsvergleich

In diesem Abschnitt werden die, für die Verarbeitung von Human Tasks, als elementar identifizierten Services hinsichtlich ihrer implementierten Funktionalitäten auf Operationsebene betrachtet. Schließlich werden die von der WS-HT Spezifikation definierten Operationen aufgezeigt und bezüglich der Abdeckung durch die beiden Plattformen, beziehungsweise deren Human Workflow Implementierungen, analysiert.

Bei den im Folgenden beschriebenen Operationen werden die Übergabeparameter und Rückgabewerte sowie mögliche Exceptions nicht mit angegeben. Hierfür sei an den betreffenden Stellen aus Gründen der Übersichtlichkeit auf die entsprechenden API-Beschreibungen verwiesen.

4.2.3.1 Oracle: Task Service und Task Query Service

Die aufgeführten Operationen und deren Beschreibungen sind dem Oracle SOA Suite 11g Developer‘s Guide[36] entnommen. Hier finden sich ebenfalls die Beschreibungen für alle anderen, in Abschnitt 4.2.2 beschriebenen Kernservices. Für weitergehende Informationen empfiehlt sich an außerdem ein Blick in die Beschreibung der Java-API[37].

Operation	Beschreibung
acquireTask, acquireTasks	Übernehmen der Verantwortung eines/mehrerer Tasks
addAttachment	Hinzufügen eines Attachments zu einem Task

[36] Vergleiche [OSOA04], Kapitel 32
[37] Vergleiche [OSOA05]

addComment	Hinzufügen eines Kommentars zu einem Task
createToDoTask	Erstellen eines Todo-Task
delegateTask, delegateTasks	Delegieren der Verantwortung für einen Task. Beauftragter und Beauftragender können beide den Task bearbeiten
deleteTask, deleteTasks	Logisches Löschen eines/mehrerer Task
errorTask	Fehlerhaftsetzen eines Task
escalateTask, escalateTasks	Eskalieren eines Tasks
getApprovers	Abfragen vorheriger Bearbeiter
getFutureParticipants	Abfragen zukünftige am Task beteiligte Akteure
getUsersToRequest InfoForTask	Abfragen von Benutzern die bezüglich weiterer Taskinformationen befragt werden können
initiateTask	Starten eines Tasks
mergeAndUpdateTask	Aktualisieren von Taskinformationen: • Nutzdaten • Kommentare • Status • Resultat des Task
overrideRoutingSlip	Überschreiben des in den Task-Metadaten definierten Task-Routing.
purgeTask, purgeTasks	Physikalisches Löschen eines/mehrerer Tasks
pushBackTask	Zurückgeben eines Task an den vorherigen Bearbeiter
reassignTask, reassignTasks	Neuzuordnung eines/mehrerer Tasks
reinitiateTask	Neustarten eines abgeschlossenen Tasks, wobei Historie, Kommentare und Attachments erhalten bleiben
releaseTask, releaseTasks	Abgeben der Verantwortung für einen/mehrere Tasks
removeAttachment	Entfernen eines Attachments
renewTask	Verlängern der Bearbeitungsfrist eines Tasks
requestInfoForTask	Abfragen von Informationen zu einem Task
requestInfoForTask WithReapproval	Abfragen von Informationen zu einem Task, zuvor getroffene Entscheidungen müssen erneut bearbeitet werden
resumeTask, resumeTasks	Fortsetzen der Bearbeitung eines/mehrerer Tasks
routeTask	Ad hoc Routing[38] eines Task zum nächsten Bearbeiter
skipCurrentAssignment	Überspringen der aktuellen Bearbeiterzuordnung
submitInfoForTask	Aktualisieren vorhandener Taskinformationen
suspendTask, suspendTasks	Unterbrechen der Bearbeitung eines/mehrerer Tasks
updateTaskOutcome, updateOutcomeOfTasks	Aktualisieren des Taskresultats
updateTask	Aktualisieren des Task
updateTaskOutcome AndRoute	Aktualisieren des Taskresultats und ad hoc Routing an den nächsten Bearbeiter
withdrawTask, withdrawTasks	Abbrechen der Taskverarbeitung

Tabelle 8: Operationen des Oracle Task Service

[38] Ad hoc Routing: Weiterleiten des Task, das zur Designzeit nicht in den Task-Metadaten definiert wurde.

Operation	Beschreibung
authenticate	Authentifizieren eines Benutzers und Erstellen eines Workflow-Context[39]
authenticate OnBehalfOf	Authentifizieren eines Benutzers im Namen eines Dritten und Erstellen eines Workflow-Context
countTasks	Zählen aller Tasks, die definierten Kriterien genügen
countViewTasks	Zählen aller Tasks, die den in der View[40] definierten Kriterien genügen
createContext	Erstellen des Workflow-Context aus einem authentifizierten HTTP-Request
doesTaskExist	Überprüfen ob ein über Kriterien identifizierter Task existiert
doesViewTaskExist	Überprüfen ob ein über View-Kriterien identifizierter Task existiert
getWorkflowContext	Abfragen des aktuellen Workflow-Context
destroyWorkflow-Context	Zerstören des Workflow-Context
getTaskDetailsById	Abfragen der Details eines Tasks über die ID
getTaskDetailsByNumber	Abfragen der Details eines Tasks über die Nummer
getTaskHistory	Abfragen der Task-Historie
getTaskSequence	Abfragen der Bearbeitungsreihenfolge
getTaskVersionDetails	Abfragen von Task-Versionsinformationen
getWorkflowContext ForAuthenticatedUser	Abfragen des Workflow-Context für einen Benutzer der mittels eines JAAS[41]-Token bereits authentifiziert ist
queryAggregatedTasks	Abfragen aggregierter Taskinformationen, die definierten Kriterien genügen, gruppiert nach einer definierten Spalte
queryTaskErrors	Abfragen aller fehlerhafter Tasks, die den definierten Kriterien genügen
queryTasks	Abfragen aller Tasks, die den übergebenen Kriterien genügen; Komplexe Kriterien und Paging möglich
queryViewAggregated Tasks	Abfragen aggregierter Taskinformationen, basierend auf einer definierten View
queryViewTasks	Abfragen aller Tasks, die den in der View definierten Kriterien genügen

Tabelle 9: Operationen des Oracle Task Query Service

Alle in diesem Kapitel aufgeführten Operationen sind gleichermaßen über die EJB- als auch die SOAP-Schnittstelle verfügbar.

4.2.3.2 Activiti: Task Service und Task History Service

Die Beschreibungen zu den im Folgenden aufgeführten Operationen der Activiti Process-Engine Services sind der Activiti Java API[42] Beschreibung entnommen. In dieser Dokumentation finden sich ebenfalls die Operationsbeschreibungen der übrigen, in Abschnitt 4.2.2 aufgeführten, Core Interfaces. Die zusätzliche, mit

[39] Zentrales Objekt, dass unter anderem das Security-Token enthält
[40] Eine View ist hier als eine gespeicherte Abfrage (selektierte Spalten, Filter, Ordnungskriterien, etc.) zu verstehen. Vom Prinzip her entspricht dies dem View-Konzept Datenbanken.
[41] JAAS (Java Authentication and Authorization Services)
[42] Vergleiche [ACTI02]

der Überschrift „REST“ versehene, Spalte gibt Auskunft darüber, ob eine Operation auch über die von der Engine angebotene REST-Schnittstelle verfügbar ist.

Operation	Beschreibung	REST
addCandidateGroup	Hinzufügen einer Gruppe von Benutzern zur Liste potentieller Taskbearbeiter	×
addCandidateUser	Hinzufügen eines Benutzers zur Liste potentieller Taskbearbeiter	×
addComment	Hinzufügen eines Kommentars zu einem Task	×
addGroupIdentityLink	Hinzufügen einer Gruppe zur Liste potentieller oder zugewiesener Bearbeiter oder Eigentümer für einen Task	×
addUserIdentityLink	Hinzufügen eines Benutzers zur Liste potentieller oder zugewiesener Bearbeiter oder Eigentümer für einen Task	×
claim	Übernahme der Verantwortung eines Tasks	✓
complete	Abschließen eines Tasks	✓
createAttachment	Hinzufügen eines Attachments zu einem Task	✓
createTaskQuery	Erzeugen eines Task-Query zur dynamischen Abfrage von Tasks	✓
delegateTask	Abgeben eines Task an einen anderen Benutzer	×
deleteAttachment	Löschen eines Attachements	×
deleteCandidateGroup	Entfernen einer Gruppe von Benutzern aus der Liste potentieller Taskbearbeiter	×
deleteCandidateUser	Entfernen eines Benutzers aus der Liste potentieller Taskbearbeiter	×
deleteGroupIdentityLink	Entfernen einer Gruppe von Benutzern aus der Liste potentielle oder zugewiesene Bearbeiter oder Eigentümer für einen Task	×
deleteTask, deleteTasks	Physikalisches Löschen eines/mehrerer Tasks	×
deleteUserIdentityLink	Entfernen eines Benutzers aus der Liste potentieller oder zugewiesener Bearbeiter oder Eigentümer für einen Task	×
getAttachment	Abfragen eines bestimmten Attachments	×
getAttachmentContent	Abfragen des Attachment-Inhalts	×
getIdentityLinksForTask	Abfragen aller mit einem Task verknüpften Identitäten	×
getProcessInstanceAttachments	Abfragen aller Attachments einer Prozessinstanz	×
getProcessInstanceComments	Abfragen aller Kommentare einer Prozessinstanz	×
getSubTasks	Abfragen aller Subtasks eines Task	×
getTaskAttachments	Abfragen aller Attachments eines Task	×
getTaskComments	Abfragen aller Kommentare eines Task	×
getTaskEvents	Abfragen aller Events, die bezüglich eines Task aufgetreten sind	×
getVariable	Abfragen einer Task-Variablen, auch außerhalb des Task-Scope[43]	×
getVariableLocal	Abfragen einer Task-Variablen, innerhalb des Task-Scope	×
getVariables	Abfragen mehrerer/aller Task-Variablen, auch außerhalb des Task-Scope	×

[43] Außerhalb des Task-Scopes bezeichnet den Variablen-Scope eines eventuell vorhandenen Parent BPMN-Prozesses.

getVariablesLocal	Abfragen mehrerer/aller Task-Variablen, nur innerhalb des Task-Scope	×
newTask	Erzeugen einer neuen Task-Instanz	×
resolveTask	Anzeigen, dass Taskbearbeitung abgeschlossen wurde; Übergabe an den Eigentümer	×
saveAttachment	Aktualisieren eines Attachments	×
saveTask	Aktualisieren eines Tasks	×
setAssignee	Setzen des Bearbeiters	×
setOwner	Setzen des Eigentümers	×
setPriority	Setzen der Priorität	×
setVariable	Setzen einer Variablen im äußersten Scope (wenn noch nicht vorhanden)	×
setVariableLocal	Setzen einer Variablen auf dem Task (wenn noch nicht vorhanden)	×
setVariables	Setzen von Variablen, im äußersten Scope (wenn noch nicht vorhanden)	×
setVariablesLocal	Setzen von Variablen auf dem Task (wenn noch nicht vorhanden)	×

Tabelle 10: Operationen des Activiti Task Service

Operation	Beschreibung	REST
createHistoricActivityInstanceQuery	Suchen nach abgeschlossenen Activiti-Aktivitäten	×
createHistoricDetailQuery	Suchen nach abgeschlossenen Activiti-Prozess-/Aktivitätendetails	×
createHistoricProcessInstanceQuery	Suchen nach abgeschlossenen Activiti-Prozessinstanzen	×
createHistoricTaskInstanceQuery	Suchen nach abgeschlossenen Activiti-Taskinstanzen	×
deleteHistoricProcessInstance	Löschen abgeschlossener Activiti-Prozessinstanzen	×
deleteHistoricTaskInstance	Löschen abgeschlossener Activiti-Taskinstanzen	×

Tabelle 11: Operationen des Activiti History Service

Die Auflistung der Services und der einzelnen Operationen zeigt, dass die Unterstützung für die meisten Operationen, welche über die Java API verfügbar sind, derzeit in der REST-Schnittstelle fehlt. Diese Tatsache stellt eine Einschränkung für die Arbeit mit der Activiti BPM Plattform dar. Auf dem jetzigen Stand muss die Activiti-Engine in der gleichen JVM, wie die Client-Applikation betrieben werden, um alle Features der Engine hinsichtlich der Taskbearbeitung, welche über die Core-Interfaces zur Verfügung gestellt werden, nutzen zu können.

4.2.3.3 Abdeckungsgrad der Forderungen nach WS-HT

In diesem Abschnitt werden die Operationen vorgestellt, welche dem Verständnis der WS-HT Spezifikation nach, von einer Task-Engine Schnittstelle angeboten werden sollten. In der Spezifikation sind die Operationen in drei Kategorien

eingeteilt: Paticipants-, Query- und Administrative-Operations. Mit Hilfe der beiden zusätzlichen Spalten, welche mit „Oracle“ beziehungsweise „Activiti“ überschrieben sind, wird angezeigt, ob von den zuvor beschriebenen Schnittstellen der Oracle BPM Suite und der Activiti BPM Plattform von der Funktionalität her korrespondierende Funktionalitäten angeboten werden. Dies ist eine wichtige Grundlage für die Umsetzung des GHIA-Frameworks.

Zudem zeigen die nachstehenden Tabellen, ob die jeweilige Operation sich auf den Zustand eines Tasks auswirken kann. Dabei steht das Symbol „=“ für keine Zustandsänderung, das Symbol „≠“ zeigt an, dass die Operation den Taskzustand beeinflussen kann. Auf den Zusammenhang zwischen Operationen und dem Taskzustand wird im Folgeabschnitt näher eingegangen.

Operation	Beschreibung	Zustand	Oracle	Activiti
addAttachment	Hinzufügen von Attachments zu einem Task.	=	✓	✓
addComment	Hinzufügen von Kommentaren zu einem Task.	=	✓	✓
claim	Übernahme der Verantwortung für einen Task	≠	✓	✓
complete	Abschliessen eines Tasks	≠	✓	✓
delegate	Zuweisen eines Task an einen Bearbeiter	≠	✓	✓
deleteAttachment	Löschen eines Attachments	=	✓	✓
deleteComment	Löschen eines Kommentares	=	×	×
deleteFault	Löschen eines Fehlers	=	×	×
deleteOutput	Löschen der Output-Daten eines Task	=	×	×
fail	Fehlerhaftsetzen eines Task	≠	✓	×
forward	Weitergeben eines Task an eine andere organisatorische Instanz	≠	✓	✓
getAttachment	Abfragen eines Attachements	=	✓	✓
getAttachmentInfos	Abfragen von Attachmentinformationen eines Task (ausgenommen die Binärdaetn)	=	✓	✓
getComments	Abfragen aller Kommentare	=	✓	✓
getFault	Abfragen vorhandener Fehlerinformationen	=	×	×
getInput	Abfragen der Task-Inputdaten	=	×	×
getOutcome	Abfragen des Taskresultats	=	✓	✓
getOutput	Abfragen der Task-Outputdaten	=	✓	✓
getParentTask	Abfragen des Parent-Task	=	×	×
getParentTaskIdentifier	Abfragen des Parent-Task-Identifiers		×	×
getRendering	Abfragen eines spezifischen Renderings für einen Task/eine Notification	=	×	×

getRenderingTypes	Abfragen der verfügbaren Renderings	=	×	×
getSubtask Identifiers	Abfragen der Identifier aller vorhandenen Subtasks	=	×	×
getSubtasks	Abfragen aller Subtasks		×	✓
getTaskDescription	Abfragen der Taskbeschreibung	=	✓	✓
getTaskDetails	Abfragen der Task-/Notification-Details	=	✓	✓
getTaskHistory	Abfragen der vorhandenen Task-Historie; kann gefiltert werden	=	✓	✓
getTaskInstanceData	Abfragen aller Taskinformationen (Subtasks, Beschreibung, Details, Historie), bis auf die Attachments	=	✓	✓
getTaskOperations	Abfragen der, für einen Benutzer, möglichen Operationen	=	×	×
hasSubtasks	Ermitteln, ob Subtask existieren	=	×	×
instantiateSubTask	Erzeugen eines Subtask	=	×	×
isSubtask	Ermitteln, ob ein Task ein Subtask ist	=	×	×
release	Abgeben der Verantwortung für einen Task	≠	✓	✓
remove	Physikalisches Löschen einer Notification; nicht anwendbar auf Tasks	≠	✓	✓
resume	Fortsetzen der Taskbearbeitung	≠	×	×
setFault	Setzen von Fehlerinformationen	=	×	×
setOutput	Setzen von Task-Outputdaten	=	×	×
setPriority	Ändern der Taskpriorität	=	×	×
setTaskCompletion DeadlineExpression	Setzen einer Task-Deadline Expression, welche den spätesten Abschlusszeitpunkt festlegt	=	×	×
setTaskCompletion DurationExpression	Setzen einer Task-Duration Expression, welche die Zeitspanne festlegt in der die Bearbeitung spätestens begonnen werden muss	=	×	×
setTaskStart DeadlineExpression	Setzen einer Task-Deadline Expression, welche den spätesten Startzeitpunkt festlegt	=	×	×
setTaskStart DurationExpression	Setzen einer Task-Duration Expression, welche die Zeitspanne festlegt in der die Bearbeitung spätestens abgeschlossen werden muss	=	×	×
skip	Abbrechen der Taskverarbeitung	≠	✓	×
start	Starten der Taskausführung	≠	✓	×
stop	Stoppen der Taskausführung	≠	✓	×
suspend	Unterbrechen der Taskausführung	≠	✓	×
suspendUntil	Unterbrechen der Taskausführung für eine festgelegte Zeitperiode	≠	×	×

updateComment	Aktualisieren eines Task-Kommentars	=	✓	×

Tabelle 12: Participant-Operationen nach WS-HT

Auf den ersten Blick betrachtet kann festgestellt werden, dass die Operationsabdeckung durch die Oracle Human Workflow-Engine beziehungsweise die Activiti Process-Engine nicht sehr hoch ist. Im Fall der Oracle Implementierung ist dies dadurch zu erklären, dass sie bereits lange vor der ersten Veröffentlichung der WS-HT Spezifikation existierte. Der Operationsnamen und Funktionsumfang orientieren sich daher nicht an den Vorgaben der Spezifikation. Bei Activiti liegt der Fokus auf der möglichst vollständigen Implementierung des BPMN 2.0 Standards. Die Interaktionsmöglichkeiten zwischen menschlichen Akteuren und Prozessen ist nur insoweit vorhanden, wie es für die Erfüllung der Anforderungen des BPMN 2.0-Standards erforderlich ist. [44]

Operation	Beschreibung	Zustand	Oracle	Activiti
getMyTaskAbstracts	Abfragen von so genannten „TaskAbstracts“[45]; Filterung möglich, wird in der Regel für die Anzeige einer Übersichtsliste verwendet	=	✓	✓
getMyTaskDetails	Abfragen von Task-Detailinformationen; enthält die Informationen von *getMyTaskDetails* inklusive der Detaildaten	=	✓	✓
query	Abfragen von Taskinformationen; generische Schnittstelle, die Fälle abdecken soll, welche von *getMyTaskAbstracts* oder *getMyTaskDetails* nicht abgedeckt werden können	=	✓	✓

Tabelle 13: Query-Operationen nach WS-HT

Als Anmerkung zu Tabelle 13 ist anzuführen, dass die WS-HT Spezifikation zwischen den so genannten „Simple Query Operations“, `getMyTaskAbstracts` und `getMyTaskDetails`, und der „Advanced Query Operation“ `query` unterscheidet.[46] Die Spezifikation definiert weiter, dass die erstgenannte Gruppe von Operationen zwingend von einer Taks-Engine unterstützt werden muss. Die „Advanced

[44] Weitere Informationen hierzu [BERS11], Kapitel 4 und 5, S. 14ff.
[45] Weitere Informationen zu den verwendeten Datenstrukturen in [BERS11], Abschnitt 4.2.2, S. 20ff.
[46] Vergleiche [WSHT], S. 83 - 89

Query Operations“ hingegen müssen der Spezifikation nach nicht zwingend von einer WS-HT konformen Task-Engine implementiert werden.

Operation	Beschreibung	Zustand	Oracle	Activiti
activate	Aktivieren eines Tasks	≠	×	×
nominate	Zuweisen eines Verantwortlichen (oder einer Gruppe), welcher für die Bearbeitung eines Tasks zuständig ist	≠	✓	✓
setGenericHumanRole	Ersetzen der bisherigen Verantwortung für einen Task	=	✓	✓

Tabelle 14: Administrative-Operationen nach WS-HT

Was die administrativen Aufgaben angeht, ist in der WS-HT Spezifikation wenig definiert, wie Tabelle 14 zeigt. Die Unterstützung durch die beiden Task-Engine Implementierungen ist weitestgehend gewährleistet. Das Aktivieren von Task wird jedoch weder von Activiti, noch von Oracle explizit unterstützt. Dies liegt daran, dass die Aktivierung von Tasks implizit – wie auch das gesamte Management des Task-Lifecycles – durch die Engines erfolgt.

Die Verwendung der hier vorgestellten und nach WS-HT definierten Operationen ist an die Restriktion geknüpft, dass diese nur von entsprechend berechtigten menschlichen Akteuren aufgerufen werden dürfen. Zu diesem Zweck definiert die Spezifikation Rollen, welche ein Benutzer gegenüber einem Task einnehmen kann. In der WS-HT Spezifikation wird hierzu eine Matrix[47] definiert, welche die entsprechenden Zusammenhänge zwischen den Operationen und den Rollen aufzeigt. Schreiboperationen, die beispielsweise den Taskinhalt verändern, sind dabei stärker eingeschränkt nutzbar, als die reinen Leseoperationen, die nahezu von allen Rolleninhabern aufgerufen werden können. Im Falle fehlender Berechtigungen, muss die Task-Engine dies mit Fehlermeldungen quittieren. Für nähere Details zu den Rollen sei an dieser Stelle auf die entsprechende Literatur verwiesen.[48]

Ähnliche Autorisierungskonzepte sind auch in den Task-Engines von Oracle beziehungsweise Activiti umgesetzt (siehe hierzu Abschnitt 4.2.2). Auch hier wer-

[47] Vergleiche [WSHT], S. 89 - 92
[48] Vergleiche [WSHT], S. 19 und [BERS11], S. 24

den die Berechtigungen bezüglich der Tasks über Rollen festgelegt. Diese unterscheiden sich allerdings hinsichtlich der Bezeichnung und der damit verbundenen Verantwortlichkeiten.

4.2.4 Lebenszyklus

Die WS-HT Spezifikation empfiehlt, dass Tasks zu jedem Zeitpunkt einen eindeutig definierten Zustand haben, welcher vom Task-Processor verwaltet wird. Zu diesem Zweck definiert die Spezifikation einen Task-Lifecycle, welcher als UML (Unified Modelling Language) Status-Diagramm definiert ist. Diesem Diagramm, welches in Abbildung 16 zu sehen ist, kann zudem entnommen werden, welche Operationen für die entsprechenden Statusübergänge verantwortlich sind. Dies ist zusätzlich in den Tabellen Tabelle 12 - Tabelle 14 des vorherigen Abschnitts über die Spalte „Zustand“ ersichtlich.

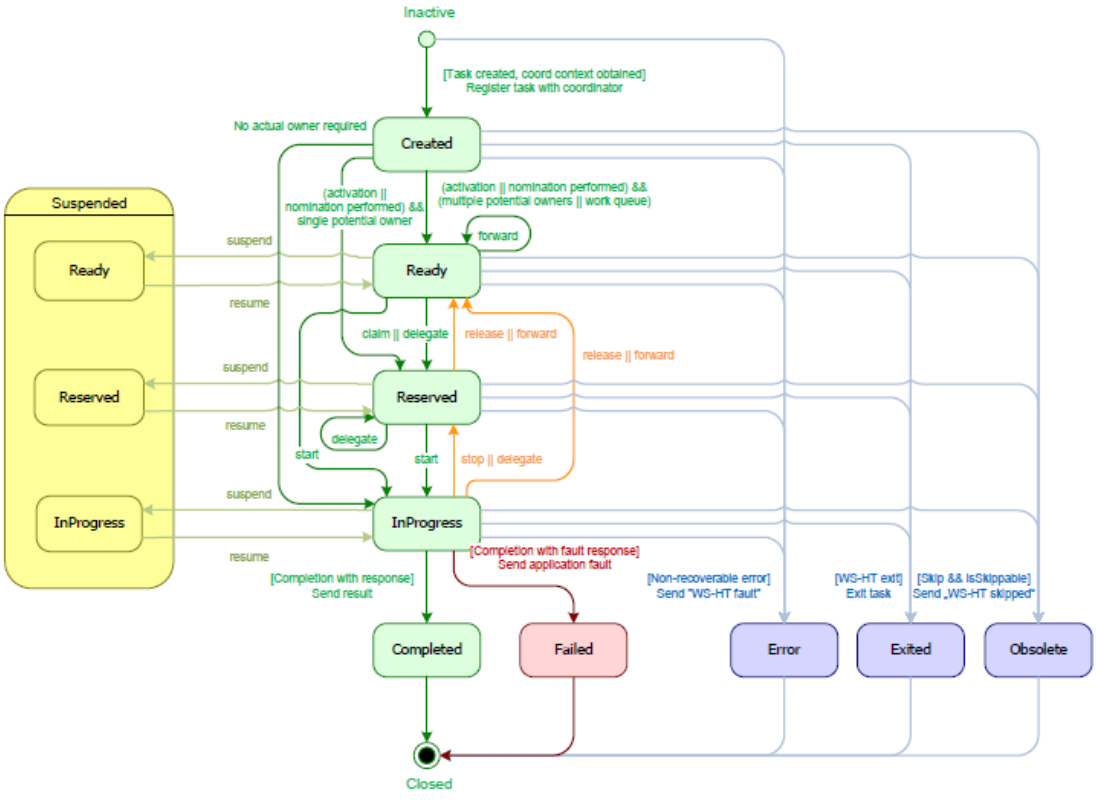

Abbildung 16: Status-Diagramm nach WS-HT[49]

Was dieses Diagramm weiterhin verdeutlicht ist, dass Operationsaufrufe eng mit dem aktuellen Taskzustand zusammenhängen. Das bedeutet um Umkehrschluss, dass Operationsaufrufe die den Taskzustand verändern können, nur dann zulässig sind, wenn der Task sich im vorausgesetzten Zustand befindet.

[49] Aus [WSHT], S. 58

Die WS-HT Spezifikation definiert zusätzlich, dass wenn ein im Sinne des Taskzustandes unzulässiger Operationsaufruf versucht wird, dieser mit einem Fehler quittiert werden muss.

Oracle BPM Suite

Tasks der Human Workflow Service-Engine der Oracle BPM Suite haben jederzeit einen eindeutigen Zustand und einen definierten Lebenszyklus. Unterschiede der WS-HT Variante zur Oracle-Implementierung sind, dass Oracle mit zusätzlichen Substatus arbeitet und zudem die Bezeichnungen der Status anders sind. Der Substatus wird unter anderem dafür verwendet, um einen Fall abzubilden, bei dem aus verschiedenen Status in einen anderen gewechselt werden kann. In einem solchen Fall würde im Substatus der ursprüngliche Wert abgelegt werden. Bezugnehmend auf Abbildung 16, wäre der der Wechsel vom Zustand „Ready“ oder „Reserved“, in den Zustand „Suspended“ ein solches Beispiel. Der Taskstatus wäre in einem solchen Fall „Suspended“ und der Substatus wäre „Ready“ beziehungsweise „Reserved“.

Activiti BPM Plattform

Activiti Human-Tasks weisen keinen komplex definierten Lebenszyklus auf. Sie sind entweder gestartet oder beendet. Die einzigen Statusinformationen, die verwaltet werden, beziehen sich auf die Delegation von Tasks. Diesbezüglich wird festgehalten, ob die Bearbeitung durch den zugewiesenen Akteur noch ausstehend oder bereits erfolgt ist. Die Tatsache, dass kein explizites Statusmanagement vorhanden ist, ist problematische denn es erschwert Tracking und Monitoring von Tasks. So kann beispielsweise nicht nachvollzogen werden, ob gerade an einem Task gearbeitet wird oder ob überhaupt jemand an einem Task arbeitet.

4.2.5 Datenstrukturen und Exceptions

Die Laufzeitdatenstruktur für Tasks, wie sie die WS-HT Spezifikation beschreibt und in den beiden BPM Plattform von Oracle und Activiti Verwendung finden, wurde bereits in der vorab erstellten Projektarbeit erläutert.[50]

[50] Vergleiche [BERS11], S. 20-23

Neben diesen zentralen Datenstrukturen definiert die WS-HT Spezifikation eine weitere schlanke Datenstruktur, welche im Zusammenhang mit der Query-Operation `getMyTaskAbstracts` (vergleiche Kapitel 4.2.3.3) verwendet wird. Sie soll im Speziellen für die Anzeige von Tasklisten verwendet werden und enthält nicht alle Taskattribute. Weder die Oracle BPM Suite, noch die Activiti BPM Plattform definieren bezüglich der Query-Operationen eine ähnliche Struktur. Hier werden immer die vollständigen Task-Datenstrukturen zurückgeliefert.

Die WS-HT Spezifikation beschreibt auch Ausnahmen, welche bei der Verarbeitung von Tasks auftreten können. In vorausgegangenen Kapiteln wurde bereits an der einen oder anderen Stelle angedeutet, dass bei der Interaktion zwischen Task-Engine und Client-Applikation mit einem entsprechenden Fehler reagiert werden muss. Ein Beispiel hierfür sind ungültige Operationsaufrufe hinsichtlich des Taskstatus. Insgesamt definiert die WS-HT Spezifikation in diesem Zusammenhang die folgenden fünf Exceptions[51]:

- `hta:illegalArgumentFault`
  Grund: Übergabe unzulässige Parameter oder falsche Anzahl von Parametern
- `hta:illegalStateFault`
  Grund: Operationsaufruf, der aufgrund des aktuellen Taskzustands nicht zulässig ist
- `hta:illegalAccessFault`
  Grund: Nicht-Autorisierter Aufruf einer Operation zur Bearbeitung von Tasks
- `hta:recipientNotAllowed`
  Grund: Nicht-Autorisierter Aufruf einer Operation zur Bearbeitung von Notifications
- `hta:illegalOperationFault`
  Grund: Unzulässiger Operationsaufruf für einen Task/eine Notification, beispielsweise Aufruf von `remove` auf einem Task.

Auch die Human Workflow Engine von Oracle reagiert auf Fehlerfälle mit Exceptions. In der Regel handelt es sich hierbei um eine `WorkflowException`. In speziellen Fällen kann auch eine `StaleObjectException` geworfen werden. Bei beiden

[51] Vergleiche [WSHT], S. 71

Exception handelt es sich um so genannte „Checked Exceptions“, die eine explizite Behandlung erfordern. Die `StaleObjectException` wird von den schreibenden Taskoperationen zurückgeliefert, wenn das verwendete Task-Objekt in der Zwischenzeit von einem anderen Akteur verändert wurde.

Die Activiti-Engine quittiert fehlerhafte Anfragen grundsätzlich mit einer `ActivitiException`. Im Gegensatz zu den bei Oracle erzeugten Ausnahmen handelt es sich hierbei um „Unchecked Exceptions“, die keine explizite Behandlung erfordern. Die Activiti-Engine reagiert mit solchen Exceptions beispielsweise dann, wenn eine Operation auf einem nicht existenten Task ausgeführt werden soll.

4.2.6 Tasklist-Applikationen

Die Thematik der Client-Applikationen, oder besser gesagt deren Implementierung, ist nicht Betrachtungsgegenstand der WS-HT Spezifikation. In der Spezifikation wird nur generell festgelegt, wie die entsprechenden Schnittstellen zwischen Task-Engine und Tasklist-Applikationen aussehen (vergleiche vorhergehende Abschnitte 4.2.1 - 4.2.5) und, dass für einen Task verschiedene Renderings angeboten werden sollten, um nicht abhängig von einer bestimmten Client-Implementierung zu sein.[52]

In der Regel bringen die Task-Engines bereits eine entsprechende Client-Applikation mit. Auch bei den beiden in dieser Arbeit verwendeten Plattformen ist dies der Fall. Im Folgenden werden die Tasklist-Applikationen von Oracle und Activiti kurz vorgestellt.

Oracle BPM Suite

Die Oracle BPM Worklist-Application ist eine, mit dem Oracle Application Development Framework (ADF) entwickelte, Webanwendung für das Task-Management. Bei Oracle ADF handelt es sich um ein ganzheitliches Entwicklungsframework, welchem eine Model-View-Controller (MVC) Architektur zugrunde liegt und das technologisch auf JEE-Technologien aufsetzt. So werden im Falle der Worklist-Application beispielsweise die Oberflächen für die Taskbearbeitung standardmäßig mit ADF Faces, das auf JSF aufsetzt, erstellt beziehungs-

[52] Vergleiche [WSHT], S. 26

weise generiert.

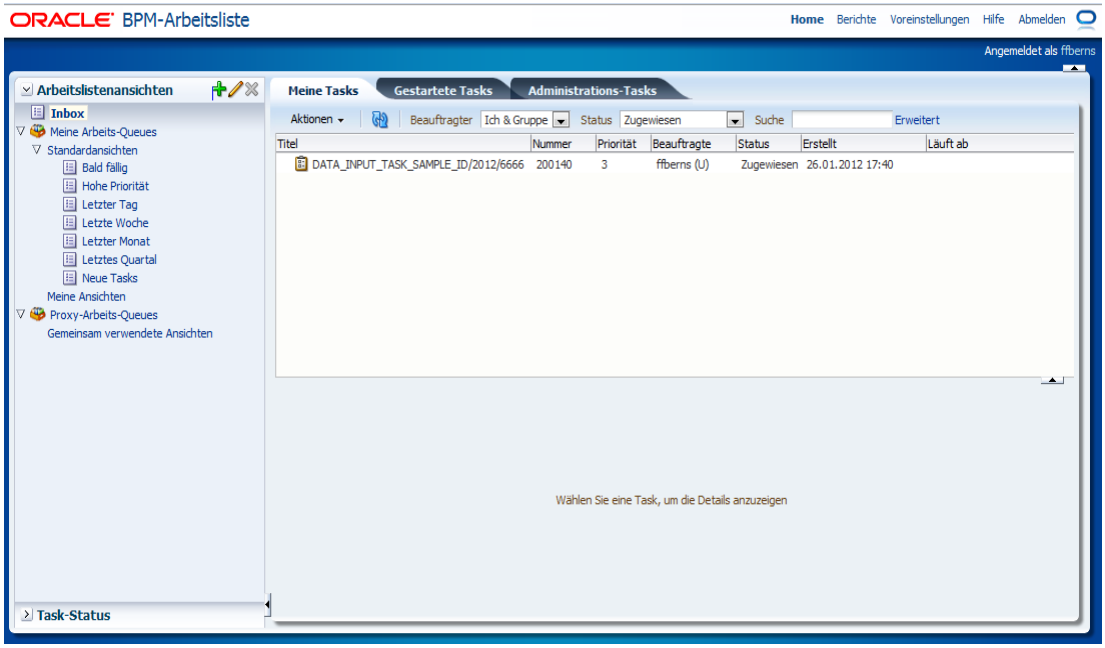

Abbildung 17: Oracle Worklist-Application

Wie Abbildung 17 zeigt, werden in der Oracle Worklist-Applikation die für einen Akteur zu bearbeitenden Tasks angezeigt. Diese können nach verschiedenen Kriterien gefiltert werden. Zudem sind verschiedene Ansichten verfügbar, in denen beispielswiese die Task nach Fälligkeit oder Priorität geordnet angezeigt werden können. Neben den Tasks, bei denen ein Akteur als Bearbeiter eingetragen ist, können auch die Tasks angezeigt werden, für welche ein Akteur als Initiator auftritt oder administrative Aufgaben wahrnehmen kann.

Unabhängig davon, welche Rolle ein Akteur gegenüber einem Task innehat, werden bei der Auswahl eines spezifischen Task, die zugehörigen Details angezeigt, sofern diese vorhanden sind und ein entsprechendes Taskformular, in Form einer Mini-ADF-Applikation, definiert worden ist. Über dieses Taskformular können die Taskdaten editiert und die Bearbeitung des Tasks abgeschlossen werden.

Activiti BPM Plattform

Die Tasklist-Applikation, die von der Activiti BPM Plattform mitgeliefert wird, ist der Activiti-Explorer.

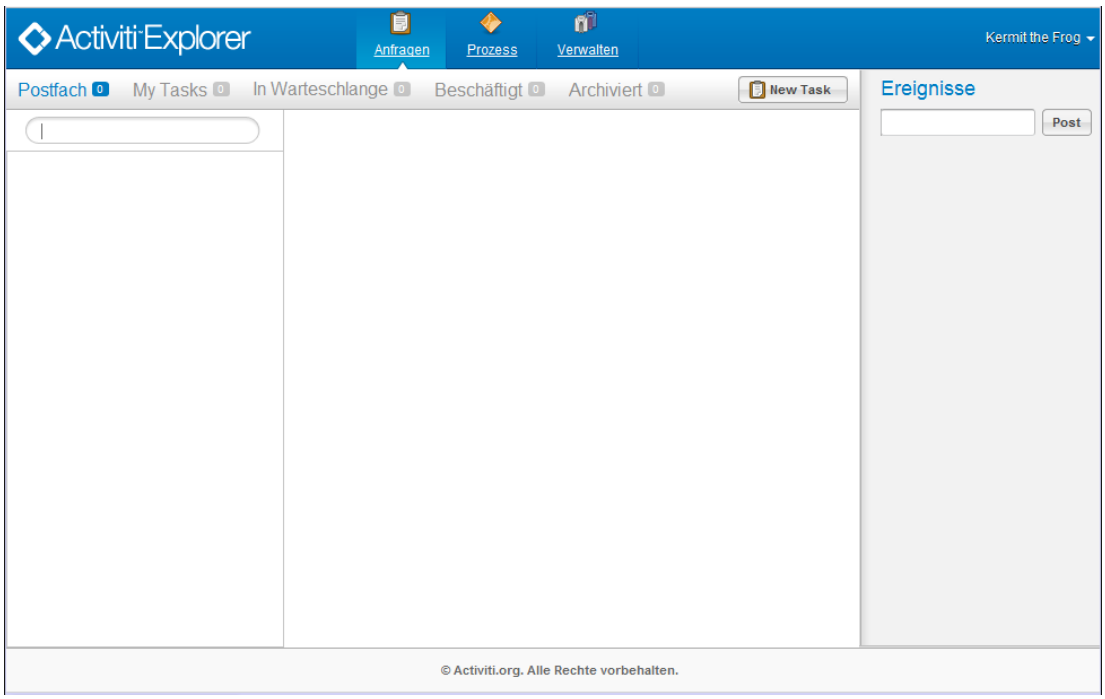

Abbildung 18: Activiti-Explorer

Der Activiti Explorer ist eine mit dem Vaadin Framework[53] erstellte Webapplikation. Bei Vaadin handelt es sich um ein Opensource Java Webframework mit dem moderne Rich Internet Applications (RIA) komplett in Java, ohne das separate Seitenbeschreibungen in HTML oder ähnlichen Technologien zu definieren sind, erstellt werden können. Dabei basieren die Vaadin-Komponenten auf GWT-Widgets (Google Web Toolkit).

Mit Hilfe des Activiti-Explorers sind, ähnlich wie bei der Oracle Worklist-Applikation, das Management und das Monitoring von User-Tasks möglich. Daneben ist mit Hilfe des Explorers die Verwaltung von BPMN-Prozessen möglich. Hierzu zählt beispielsweise das Starten neuer Prozessinstanzen oder das Deployment neuer Prozesse. Zudem kann das unterliegende Datenbankschema eingesehen werden. Hier ist es auch beispielsweise möglich neue Benutzer anzulegen und diesen die entsprechenden Rollen für die Taskbearbeitung zu zuweisen.

[53] Nähere Informationen siehe [VAADIN]

5 Konzeption des GHIA Frameworks

In diesem Kapitel geht es um die Konzeption der Adapterschicht, zur transparenten Interaktion mit unterschiedlichen Task-Engines. Als Grundlage hierfür dienen die in den vorhergehenden Kapiteln erarbeitenden Ergebnisse und Informationen.

Zum Einstieg wird hierbei zunächst die Architektur bestehender Anwendungen aus dem Bereich der Verarbeitung menschlicher Interaktion betrachtet. Im Anschluss daran erfolgen die Konzeption des Core-Frameworks, der Schnittstellen sowie die Darstellung des grundsätzlichen Kommunikationsverlaufs bei der Bearbeitung von Tasks bei Verwendung des GHIA-Frameworks. Abschließend erfolgt ein erstes Zwischenfazit.

5.1 Architektur

In diesem Abschnitt geht es darum wie aktuelle Applikationen, die im Kontext menschlicher Interaktion für die Verarbeitung von Task (zum Beispiel Inbox-Applikationen) verwendet werden, aus architektonischer Sicht aufgebaut sind. Weiterhin wird betrachtet, wie eine mögliche Architektur solcher Applikationen, unter Verwendung des zu erstellenden GHIA-Frameworks, aussehen könnte und welche Vorteile sowie Verbesserungen sich hierdurch für die Applikationsentwicklung ergeben.

5.1.1 Ist-Architektur

Bei den in dieser Arbeit betrachteten Task-Engines von Oracle und Activiti ist der in Abbildung 19 skizzierte Aufbau, was die Realisierung von Inbox-Applikationen angeht, erkennbar. Kernkomponenten dieser Architektur sind die Client-Applikationen, die Kommunikationsschnittstelle und die Task-Engine. Über die Schnittstelle bietet die Task-Engine Operationen an, welche von einer beliebigen Applikation aufgerufen werden können.

Wie die Abbildung zeigt, ist es möglich unter Verwendung verschiedener Protokolle mit der Task-Engine zu interagieren. Ziel hiervon ist, dass möglichst viele technologisch unterschiedliche Applikationen mit dieser kommunizieren können. Allerdings leidet die hierüber gewonnene Interoperabilität, wenn proprietäre Protokolle wie RMI oder auch native Java-Methodenaufrufe für die Kom-

munikation zum Einsatz kommen. Bei den durch die Schnittstellen angebotenen Kommunikationsprotokollen muss weiterhin bedacht werden, dass diese in unterschiedlichem Maße für verschiedene clientseitig eingesetzte Technologien geeignet sind. Für eine mobile Lösung im Bereich der Task-Bearbeitung beispielsweise wäre eine SOAP-Schnittstelle, aufgrund des Marshalling-/Unmarshalling-Prozederes und dem damit verbundenen Rechenaufwand, weniger gut geeignet. Eine REST-Schnittstelle wäre hier besser geeignet, da REST-Webservices leichtgewichtiger sind.

Wie bereits in den Abschnitten 4.2.1 und 4.2.3 deutlich wurde, unterscheiden sich die Schnittstellen, neben unterschiedlichen Kommunikationsprotokollen, auch bezüglich ihres Aufbaus und den angebotenen Operationen. Teilweise verbergen sich hierbei gleiche Funktionalitäten hinter unterschiedlichen Operationsaufrufen, was aus Sicht der Client-Applikationen inkonsistent ist.

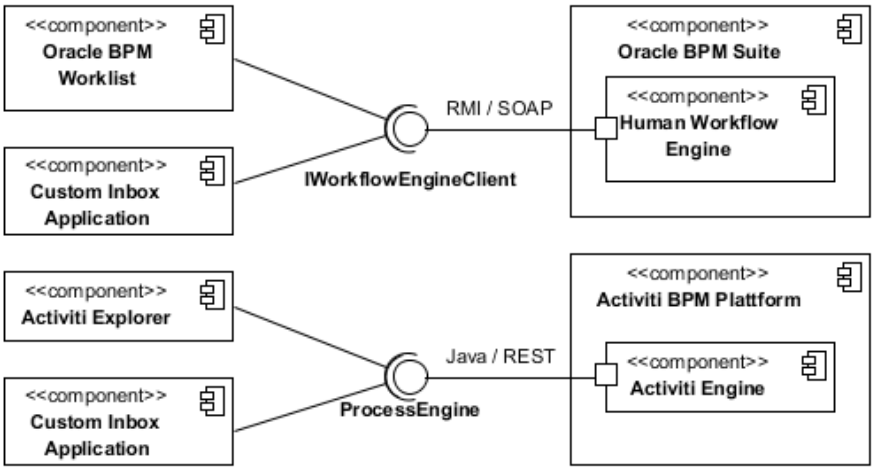

Abbildung 19: Ist-Applikationsarchitektur aktueller Inbox-Applikationen

Der skizzierte Aufbau weist zudem eine enge Kopplung zwischen Task-Engine und den Client-Applikationen auf, was die in Abschnitt 2.2.1 beschriebenen Probleme einer „Point-to-Point“ Kopplung mit sich bringt. Bei Änderung an den Schnittstellen, die sich beispielsweise im Falle von Versionsupdates ergeben können, oder aber in einem Szenario bei dem die gesamte zugrunde liegende Plattform ausgetauscht wird, müssen alle konsumierenden Client-Applikationen angepasst beziehungsweise neu implementiert werden.

Zusammenfassend betrachtet sind also die Probleme der Architektur heutiger Inbox-Applikationen: enge Kopplung zwischen Client-Applikationen und Task-Engine, Verwendung proprietärer Technologien für die Kommunikation und

eine nicht-standardisierte Schnittstelle. Diese Punkte werden in der Soll-Architektur mit Hilfe des zu erstellenden GHIA-Frameworks adressiert.

5.1.2 Soll-Architektur

Im Folgenden wird eine Soll-Applikationsarchitektur beschrieben, welche die im vorherigen Abschnitt erkannten Probleme mit Hilfe einer Adapterschicht (GHIA) adressiert. Abbildung 20 skizziert den Aufbau, wobei hier aus Gründen der Übersichtlichkeit die von den Herstellern mitgelieferten Applikationen für die Taskbearbeitung nicht mehr berücksichtigt werden. Diese benutzen nach wie vor direkt die proprietären Schnittstellen `IWorkflowEngineClient` beziehungsweise `ProcessEngine`.

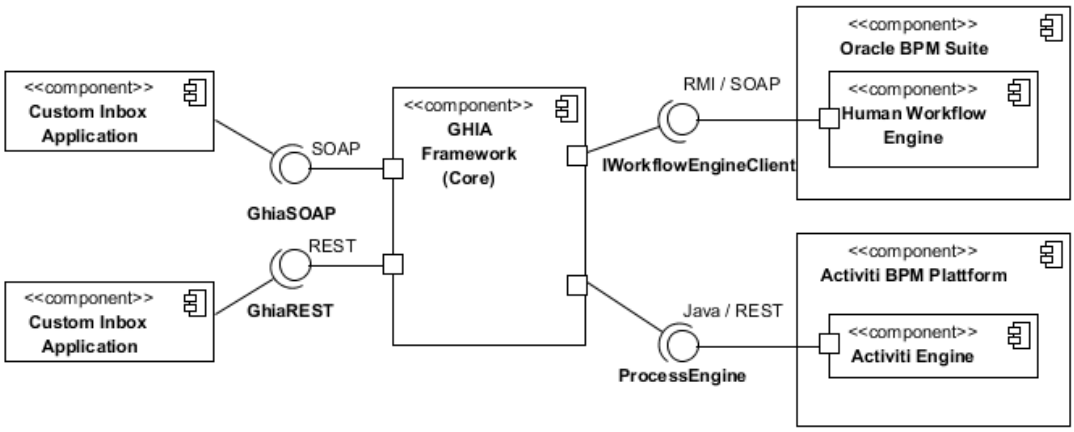

Abbildung 20: Soll-Applikationsarchitektur zukünftiger Inbox-Applikationen

Das GHIA-Framework kapselt, wie die Abbildung zeigt, die herstellerspezifischen Implementierungsdetails und bietet standardisierte Schnittstellen für Client-Applikationen an. Die zu dem Adapterframework zugehörigen Bestandteile sind:

- Kernframework („Core“)
  Enthält die Kernelemente der Adapterschicht, wie Steuerungslogik für die Verarbeitung von eingehenden Anfragen und stellt Basisfunktionalitäten für andere Bestandteile und Erweiterungen zur Verfügung.

- SOAP-Schnittstellenimplementierung
  Implementiert die Funktionalität, die für die Verarbeitung von SOAP-Nachrichten benötigt wird und reicht diese transformiert in Form eines GHIA- Data Transfer Objects (DTO) an das Kernframework weiter. Das Schnittstellendesign orientiert sich an den Vorgaben von WS-HT.

- REST-Schnittstellenimplementierung
  Implementiert die Funktionalität, die für die Verarbeitung von REST-Anfragen benötigt wird. Das Vorgehen bei der Nachrichtenverarbeitung läuft analog zur SOAP-Schnittstelle.

- Task-Engine-spezifische Schnittstellenimplementierung
  Implementiert die spezifische Logik, die benötigt wird um einen eingehenden Request verarbeiten zu können und übernimmt benötigte Überführungen des herstellerspezifischen Datenformats in das vom GHIA-Core verstandene Datenformat. In der ersten Ausbaustufe existieren spezifische Implementierungen für Oracle sowie Activiti.

Mit Hilfe des GHIA-Frameworks kann somit die enge Kopplung zwischen den Client-Applikationen und den Task-Engines aufgelöst werden. Änderungen an den eigentlichen Schnittstellen der Engines führen nicht mehr dazu, dass alle Client-Applikationen angepasst werden müssen. In einem solchen Fall muss lediglich die Task-Engine-spezifischen Schnittstellenimplementierung für die Engine angepasst werden. Auch die Schnittstellenproblematik in Richtung der Client-Applikationen wird insofern entschärft, dass die Interaktion über die standardisierten Kommunikationsprotokolle SOAP oder REST läuft und sich zudem das Schnittstellendesign, zumindest was die SOAP-Schnittstelle angeht, an den Vorgaben der WS-HT Spezifikation orientiert.

5.2 Aufbau des Frameworks

Im vorherigen Abschnitt wurden bereits die Bestandteile des GHIA-Framework mit dem Kernframework, der SOAP- und der REST-Schnittstellen sowie den Task-Engine-spezifischen Schnittstellenimplementierungen kurz vorgestellt. In den folgenden Passagen wird detaillierter auf den Aufbau der einzelnen Komponenten eingegangen.

5.2.1 Core

Das Herzstück des Frameworks ist Gegenstand dieses Abschnitts. Das in Abbildung 21 dargestellte Klassendiagramm skizziert den Aufbau.

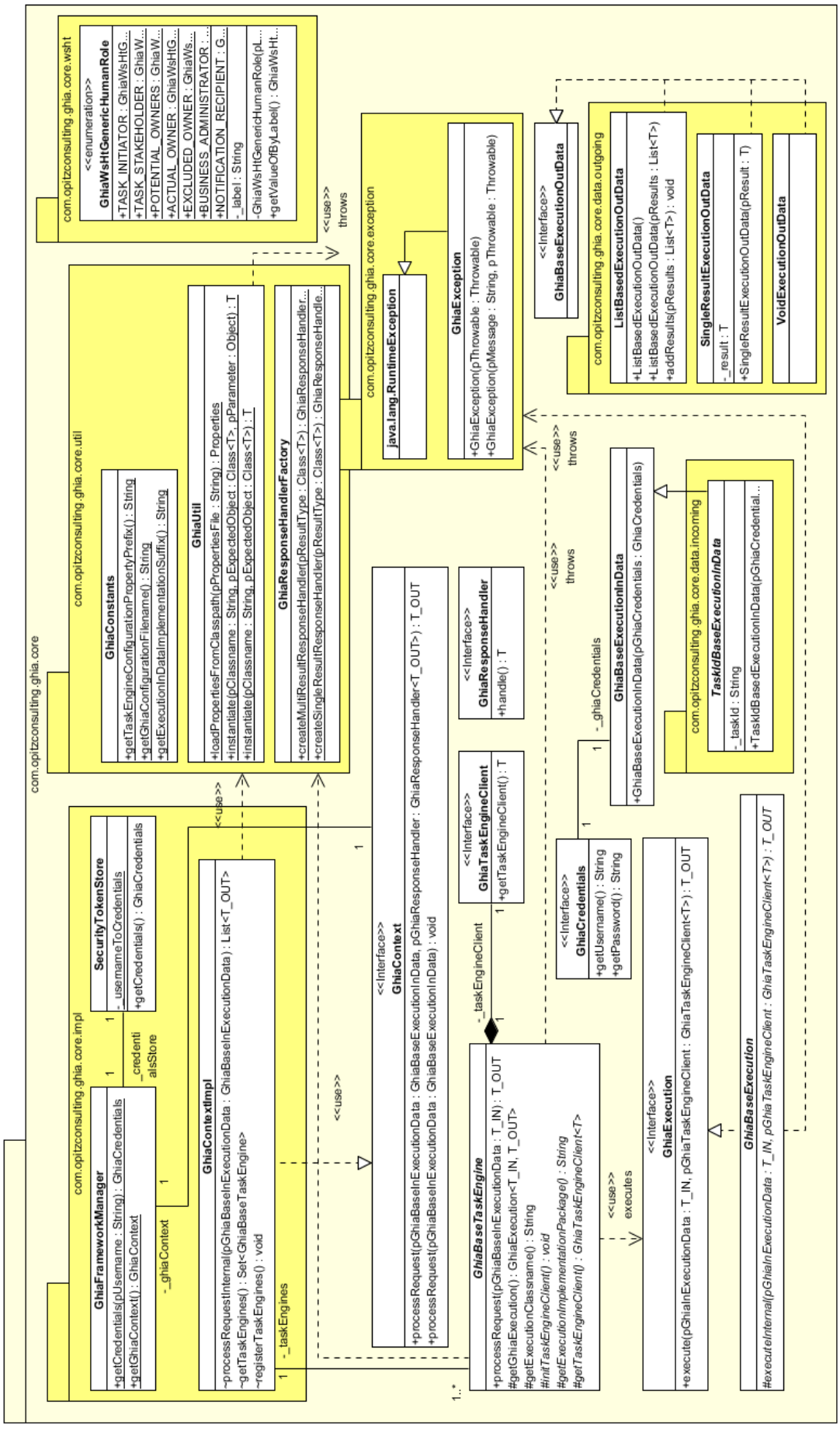

Abbildung 21: Klassendiagramm GHIA Core

In dem Klassendiagramm des Core-Frameworks werden alle Klassen, Interfaces und Enumerations sowie deren Paketierung visualisiert. Eine Ausnahme bilden die `com.opitzconsulting.ghia.core.data`-Packages. Hier werden in dieser Gesamtansicht aus Gründen der Übersichtlichkeit nur exemplarisch ein bis zwei Klassen dargestellt. Eine weitere Detailierung erfolgt später bei der Beschreibung des Packages.

Für jede Komponente werden die vorhandenen Methoden dargestellt. Ausnahme bilden hierbei triviale Getter-Methoden für den Zugriff auf private Membervariablen die zwar implementiert, aus Gründen der Übersichtlichkeit im Diagramm jedoch nicht dargestellt werden. Auch Konstruktoren sind nur dann aufgeführt, wenn diese explizit implementiert werden.

Verwendungsbeziehungen („Usage-Relations“, markiert mit `<<use>>`) werden nur in solchen Fällen verwendet, in denen es nicht explizit ersichtlich ist, in welchem Zusammenhang die Klassen zueinander stehen. Explizit heißt hier beispielsweise die Verwendung von Klassen in Methodensignaturen als Übergabeparameter, beziehungsweise Rückgabewert. Bei paketübergreifenden Verwendungsbeziehungen wird aus Gründen der Übersichtlichkeit darauf verzichtet, diese auf Klassenebene zu definieren. Hier wird die Beziehung durch einen entsprechenden Pfeil, ausgehend von der aufrufenden Klasse zum Package in dem die benutzte Klasse definiert ist, angedeutet.

In den Signaturen einiger Methoden finden sich die generischen Platzhalter `T`, `T_IN` und `T_OUT`. Für diese gilt die Konvention, dass `T` nicht weiter eingeschränkt ist und zur Laufzeit durch alle Typen repräsentiert werden kann deren Klasse von `java.lang.Object` erben. `T_IN` hingegen ist dahingehend eingeschränkt, dass hier zur Laufzeit nur Typen erlaubt sind, deren Klassen von `com.opitzconsulting.ghia.core.GhiaBaseExecutionInData` ableiten und für `T_OUT` sind nur solche Typen zulässig deren Klassen das Interface `com.opitzconsulting.ghia.core.GhiaExecutionOutData` implementieren.

Im Folgenden werden die einzelnen Packages, beziehungsweise die darin enthaltenen Klassen und deren Funktionalität, kurz beschrieben.

Package com.opitzconsulting.ghia.core

In diesem Package sind Interfaces und Basisklassen enthalten, die innerhalb des Core-Frameworks und auch von anderen Bestandteilen des Frameworks benötigt und verwendet werden.

Klassenname	Beschreibung
`GhiaBaseExecution`	Basisklasse für alle Executions (siehe hierzu auch GhiaExecution), ist zuständig für die Fehlerbehandlung, sodass in einer speziellen Execution nur die benötigte Logik implementiert werden muss.
`GhiaBaseExecutionInData`	Basisklasse für DTO-Klassen eingehender Anfragen.
`GhiaBaseTaskEngine`	Basisklasse für alle Task-Engine Implementierungen, muss implementiert werden, wenn eine spezifische Task-Engine bei der Request-Verarbeitung vom Framework berücksichtigt werden soll.
`GhiaContext`	Basis-Interface für den GhiaContext, der die zentrale Verarbeitungseinheit des Frameworks darstellt und für das Management der Task-Enignes zuständig ist.
`GhiaCredentials`	Basis-Interface für die Übergabe von Authentifizierungsinformationen. Aktuell werden nur Benutzername und Passwort unterstützt.
`GhiaExecution`	Basis-Interface für eine Execution, welche die Ausführungslogik für einen Operationsaufruf implementiert. Jede spezifische Task-Engine Implementierung muss pro Operation eine Execution zur Verfügung stellen. Das Prinzip der Executions orientiert sich am „Command“ Design-Pattern der Gang of Four.
`GhiaExecutionOutData`	Marker-Interface für DTOs ausgehender Anfragen.
`GhiaResponseHandler`	Basis-Interface für spezifische ResponseHandler-Implementierungen. Ein ResponseHandler implementiert die Logik die benötigt wird, um im Falle eines Client-Aufruf der an mehrere Task-Engines delegiert wird, die Ergebnisse zu verarbeiten.
`GhiaTaskEngineClient`	Basis-Interface für die Kapselung einer spezifischen Task-Engine Schnittstelle in einer Wrapper-Klasse (beispielsweise der `IWorkflowServiceClient` im Falle der Oracle Task-Engine).

Tabelle 15: Beschreibung der Klassen aus com.opitzconsulting.ghia.core

Die 1:1-Beziehung zwischen den Klassen `GhiaBaseExecutionInData` und `GhiaCredentials` dient der Entkopplung von Nutzdaten und den Authentifizierungsinformationen. Perspektivisch gesehen soll es so möglich sein, neben Benutzername und Passwort, weitere beziehungsweise andere Authentifizierungsinformationen zu übergeben.

Die Entscheidung die Beziehung zwischen den Klassen `GhiaBaseTaskEngine` und `GhiaTaskEngineClient` als 1:1-Komposition abzubilden liegt darin begründet,

dass ohne die entsprechende Task-Engine beziehungsweise deren Konfiguration auch nicht deren Schnittstelle zur Verfügung steht. Weiterhin dient die Verwendung dieser Wrapper-Klasse dazu, dass die proprietäre Schnittstelle der Task-Engines nur in den jeweils spezifischen Implementierungen der `GhiaBaseTaskEngine` und den zugehörigen speziellen `GhiaExecution`-Implementierungen bekannt sein müssen.

Package com.opitzconsulting.ghia.core.impl

Dieses Package enthält die Hauptklassen des Framework-Kerns was Initialisierung und Management angeht. Dabei ist der `GhiaFrameworkManager` verantwortlich für die Initialisierung sowie die Verwaltung des `GhiaContext` und fungiert für diesen als Provider-Schnittstelle. Der Zugriff auf den `GhiaContext` muss also immer über den `GhiaFrameworkManager` erfolgen. Dieser stellt seinerseits sicher, dass es zu jedem Zeitpunkt nur eine Instanz des `GhiaContext` gibt. Das ist auch der Grund dafür, warum die Beziehung zwischen `GhiaContext` und `GhiaFrameworkManager` als 1:1-Assozation modelliert ist. Überdies ist der `GhiaFrameworkManager` – zumindest in dieser ersten Ausbaustufe – für das Management der `GhiaCredentials` verantwortlich. Für diesen benutzt er den `SecurityTokenStore`, der die Authentifizierungsinformationen der Benutzer verwaltet.

Die Klasse `GhiaContextImpl`, als konkrete Implementierungsklasse des `GhiaContext`, ist die zentrale Managementinstanz des Frameworks. Sie ist zuständig für Registrierung und Verwaltung einer oder mehrerer konkreter Task-Engine Implementierungen und steuert die Verabeitung der eingehenden Anfragen.

Package com.opitzconsulting.ghia.core.data

In den beiden Sub-Packages `*.incoming` beziehungsweise `*.outgoing` finden sich die Implementierungen der DTO-Klassen, die für den Austausch der Anfrage- und Ergebnisdatenstrukturen mit den spezifischen Executions benutzt werden. Wie eingangs bereits angemerkt sind in dem in Abbildung 21 dargestellten Klassendiagramm nicht alle Klassen der beiden Sub-Packages dargestellt. Dies gilt vor allem für die Anfragedatenstrukturen die im Sub-Package `*.incoming` abgelegt werden.

Im Falle der Ergebnisdatenstrukturen, die im Sub-Package `*.outgoing` definiert werden, sind allerdings bereits alle Datenstrukturen dargestellt. Das liegt darin

begründet, dass Operationen entweder keinen, einen oder mehrere Rückgabewert haben können. Diese drei Möglichkeiten werden wie folgt abgedeckt:

- Kein Ergebnis: Benutzen der entsprechenden `GhiaContext`-Methode ohne Rückgabewert: `void processRequest(T_IN pGhiaBaseInExecution)`
- Ein Ergebnis: Die `GhiaExecution` liefert ein Objekt vom Typ `SingleResultExecutionOutData`, welche das WS-HT-spezifische Rückgabeobjekt enthält
- Mehrere Ergebnisse: Die `GhiaExecution` liefert ein Objekt vom Typ `ListBasedExecutionOutData`, worin die WS-HT-spezifischen Rückgabewerte enthalten sind

Die Einschränkung bei mehreren Rückgabewerten ist, dass hier eine Liste mit Objekten des gleichen Typs zurückgeliefert wird.

Abbildung 22 zeigt ein vollständiges Klassendiagramm des Sub-Packages `*.incoming`. Vollständig in dem Sinne, dass alle aktuell implementierten Klassen die für die Erstellung des Prototyps benötigt werden enthalten sind.

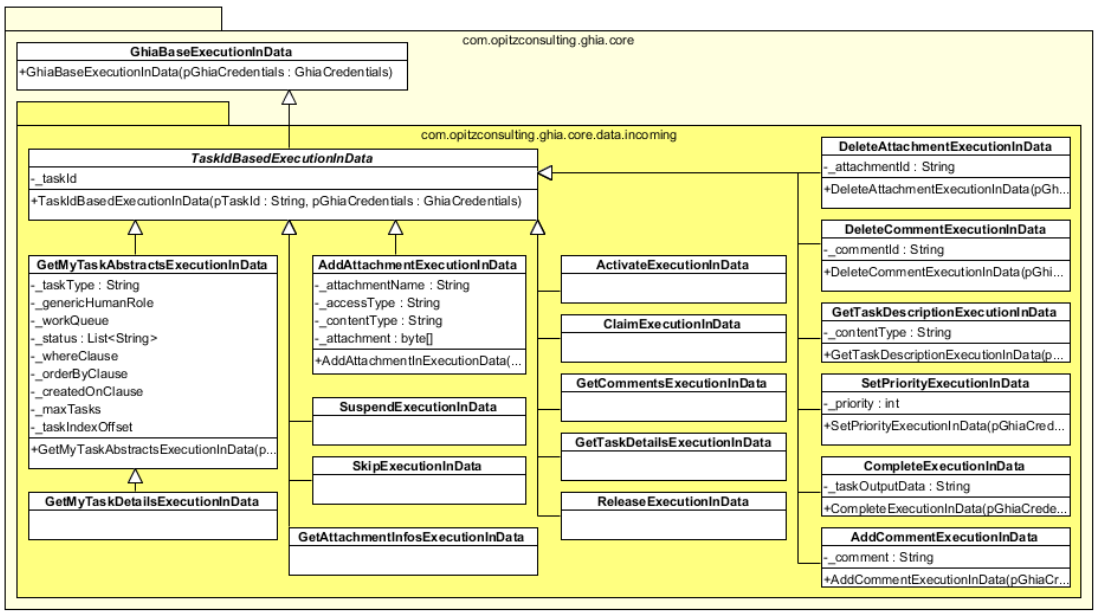

Abbildung 22: Klassendiagramm Package `*.incoming`

Package com.opitzconsulting.ghia.core.exception

In diesem Package werden Framework-spezifische Exceptions, beziehungsweise Exception-Handler Klassen, definiert. Derzeit ist hier nur eine Unchecked

Exception, die `GhiaException`, definiert. Dies liegt vor allem daran, dass in dieser ersten Ausbaustufe nur eine rudimentäre Fehlerbehandlung stattfindet.

Package com.opitzconsulting.ghia.core.util

Die Klassen in diesem Package beinhalten Methoden, deren Funktionalität in verschiedenen Bereichen benötigt wird. In der Klasse `GhiaConstants` werden alle konstanten Werte, die verwendet werden sollen, hinterlegt. Das hat den Hintergrund, dass diese so an zentraler Stelle definiert sind und Änderungen so einfacher sind als wenn verwendete Konstanten, beispielsweise als statische Klassenvariablen, über mehrere Klassen verteilt definiert werden.

Bei der Klasse `GhiaUtil` handelt es sich um eine reine Utility-Klasse, die verschiedene Hilfsfunktionalitäten anbietet. Die Klasse `GhiaResponseHandlerFactory` stellt eine Factory[54]-Klasse dar, die verschiedene Ausprägungen des `GhiaResponseHandler`-Interfaces liefern kann. Analog zu den Klassen im Package `*.outgoing` gibt es auch hier nur eine begrenzte Anzahl an konkreten `GhiaResponseHandler`-Implementierungen die benötigt und an mehreren Stellen wiederverwendet werden. Es gibt einen Handler für Listen und einen für die Behandlung von Einzelergebnissen; für die Operationen ohne Rückgabewert werden keine Handler benötigt.

Package com.opitzconsulting.ghia.core.wsht

In diesem Package werden Klassen, Interfaces oder Enumerations definiert, die sich aus der WS-HT Spezifikation ableiten lassen, jedoch nicht Teil der Schnittstellenbeschreibung sind.

Die Enumeration `GhiaWsHtGenericHumanRole` bildet die in WS-HT definierten Rollen, die ein Akteur gegenüber einem Task annehmen kann, ab. Diese wird unter anderem im Bereich der Query-Operationen für das Mapping auf die herstellerspezifischen Rollen benötigt.

5.2.2 Schnittstellen

Bei den Schnittstellen gibt es, wie Abbildung 20 zeigt, zwei Seiten: einmal vom GHIA-Framework zu den spezifischen Task-Engines und auf der anderen Seite die Schnittstelle zu den konsumierenden Client-Applikationen.

---

[54] Nähere Details zum Factory-Pattern [PAUJ11]

Die Schnittstelle zu einer Task-Engine wird mit Hilfe der Basisklasse `GhiaBaseTaskEngine` und den entsprechenden spezifischen Execution-Implementierungen realisiert. Die Anbindung von spezifischen Task-Engines ist somit trivial. Es ist lediglich erforderlich eine konkrete Implementierung der `GhiaBaseTaskEngine` bereitzustellen, welche die Konfigurationsdetails für die jeweils verwendete Task-Engine kapselt. Zusätzlich dazu muss für jede Operation, die durch eine Task-Engine ausgeführt werden soll, eine konkrete `GhiaExecution` bereitgestellt werden. Als Übergabeparameter und Rückgabewerte verwenden die Execution-Klassen die im Core-Framework, in den Packages `*.incoming` und `*.outgoing`, definierten DTOs.

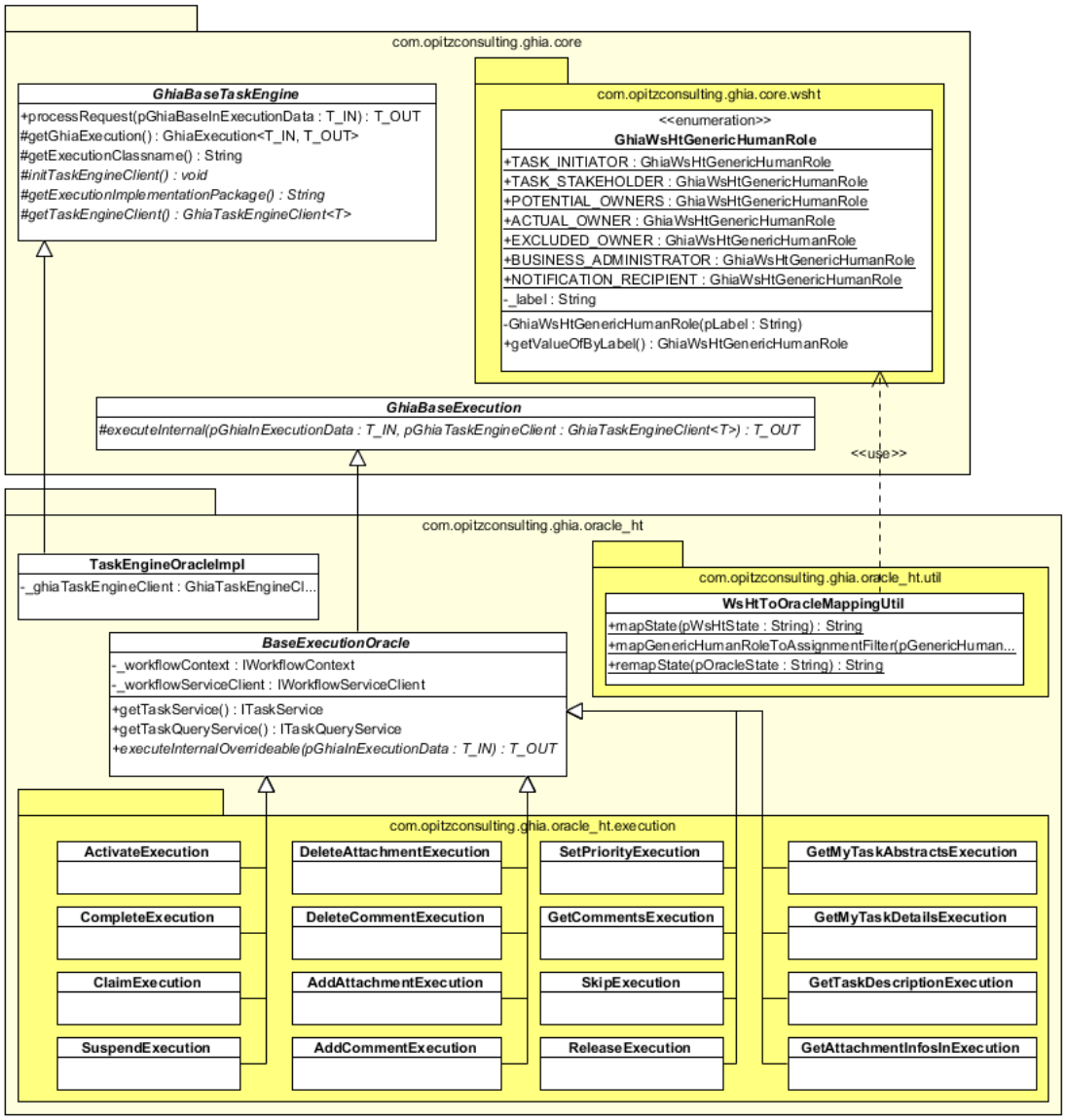

Abbildung 23: Anbindung einer spezifischen Task-Engine

Das in Abbildung 23 dargestellte Klassendiagramm skizziert das beschriebene Vorgehen exemplarisch am Beispiel der Oracle Human Workflow-Engine. Was

das Diagramm zudem zeigt, ist die Verwendung der Enumeration `GhiaWsHtGenericHumanRole`. Diese wird von der Utility Klasse `WsHtToOracleMappingUtil` für das Mapping der Rollen von den WS-HT Bezeichnungen auf die Oracle-spezifischen Bezeichnungen verwendet. Vom Prinzip her funktioniert die Anbindung beliebiger Task-Engines, wie zum Beispiel der Activiti Process-Engine, analog zu der dargestellten Anbindung der Oracle Human Workflow-Engine.

Die Konzeption der Schnittstellen in Richtung der Client-Applikationen, sieht zwei alternative Varianten vor, um mit der Adapterschicht beziehungsweise den dahinterliegenden Task-Engines interagieren zu können. GHIA wird einmal eine standardbasierte SOAP-Webserviceschnittstelle, die den in der WS-HT Spezifikation spezifizierten Vorgaben genügt, anbieten. Für den Prototyp werden hier allerdings noch nicht alle definierten Operationen implementiert, sondern nur die, die für die Erfüllung der Anforderungen notwendig sind. Neben der SOAP-Webserviceschnittstelle soll eine alternative REST-Webserviceschnittstelle zur Verfügung gestellt werden, welche dieselben Funktionalitäten zur Verfügung stellt, wie die SOAP-Schnittstelle. Die zu implementierende REST-Webserviceschnittstelle ist nicht Betrachtungsgegenstand der WS-HT Spezifikation. Sie soll vor allem zugunsten einer besseren Interoperabilität und der damit verbundenen besseren Wiederverwendbarkeit, bereitgestellt werden. So ist der REST-Kommunikationsstil beispielsweise besser für mobile Applikationen geeignet, da REST-Webservices bezüglich Bandbreitennutzung und der Verarbeitungsgeschwindigkeit auf dem Client-Device leichtgewichtiger sind, da hier das Marshalling und Unmarshalling der Java-Objekte zu XML und umgekehrt entfällt. Eine weitergehende Diskussion der beiden Webservicetechnologien soll an dieser Stelle nicht geführt werden. Zu diesem Thema gibt es immer wieder kontroverse Diskussionen. Als Quintessenz dieser kann festgehalten werden: Für beide Technologien gibt es Anwendungsfälle und es kann nicht pauschal gesagt werden REST ist besser oder schlechter als SOAP.[55] Dies soll an dieser Stelle ausreichend für die Zwecke der vorliegenden Arbeit sein.

5.3 Interaktionsverhalten

In diesem Abschnitt soll darauf eingegangen werden, wie das Zusammenspiel der einzelnen Komponenten Client, Adapterschicht und Task-Engine bei Ver-

[55] Weitere Informationen siehe [ROZM10]

wendung des GHIA-Frameworks abläuft. Der Kommunikationsverlauf ist in Abbildung 24 in Form eines Sequenzdiagramms visualisiert und stellt diesen an einem beispielhaften Aufruf der Methode `getMyTaskAbstracts` dar. Aus Gründen der Übersichtlichkeit wird in der Darstellung nur die Einbeziehung einer Task-Engine berücksichtigt. Prinzipiell laufen Methodenaufrufe aber immer wie in der Abbildung dargestellt ab. Der hier visualisierte Kommunikationsverlauf kann als die Regel im Gutfall bezeichnet werden. Auf Fehler wird mit entsprechenden Exceptions reagiert. Dies wird allerdings hier nicht dargestellt. Im Folgenden wird der in der Abbildung dargestellte Interaktionsverlauf textuell beschrieben.

Ein Benutzer ruft in einer Client-Applikation die von der Schnittstelle angebotene Methode `getMyTaskAbstracts` auf. Die Webservice-Schnittstelle, die an dieser Stelle nicht näher definiert wird weil es für den Kommunikationsverlauf unerheblich ist über welche Schnittstelle – SOAP oder REST – die Nachricht eingeht, nimmt den Aufruf entgegen. Anschließend erfragt sich die Schnittstellenimplementierung vom `GhiaFrameworkManager` die aktuellen `GhiaCredentials`. Diese werden im `SecurityTokenStore` abgelegt und können hier unter Angabe des Benutzernamens vom `GhiaFrameworkManager` angefragt werden. Im Anschluss daran wird eine Instanz der Klasse `GetMyTaskAbstractsExecutionInData` erzeugt. Bei der Instanziierung werden dieser, neben den `GhiaCredentials`, alle Aufrufparameter der Methode `getMyTaskAbstracts` übergeben.

Mit Hilfe des erzeugten DTOs kann der eigentliche Operationsaufruf gestartet werden. Die Schnittstellenimplementierung holt sich über den `GhiaFrameworkManager` die aktuelle Referenz auf den `GhiaContext`. Dieser muss gegebenenfalls zunächst noch initialisiert werden. Im Falle der erstmaligen Erzeugung des `GhiaContext` werden auch die konkreten Task-Engine Implementierungen instanziiert. Da die Initialisierung nicht Teil der Regelkommunikation ist, sondern nur einmalig erfolgt, wird dies nicht im Diagramm dargestellt.

Da die aufzurufende Methode einen Rückgabewert hat, wird für die Behandlung der Task-Engine-Antwort ein `GhiaResponseHandler` benötigt, der von der Factory-Klasse `GhiaResponseHandlerFactory` zur Verfügung gestellt wird. Im Anschluss daran werden die `GetMyTaskAbstractsExecutionInData` und der `GhiaResponseHandler` an die `processRequest`-Methode des `GhiaContext` übergeben.

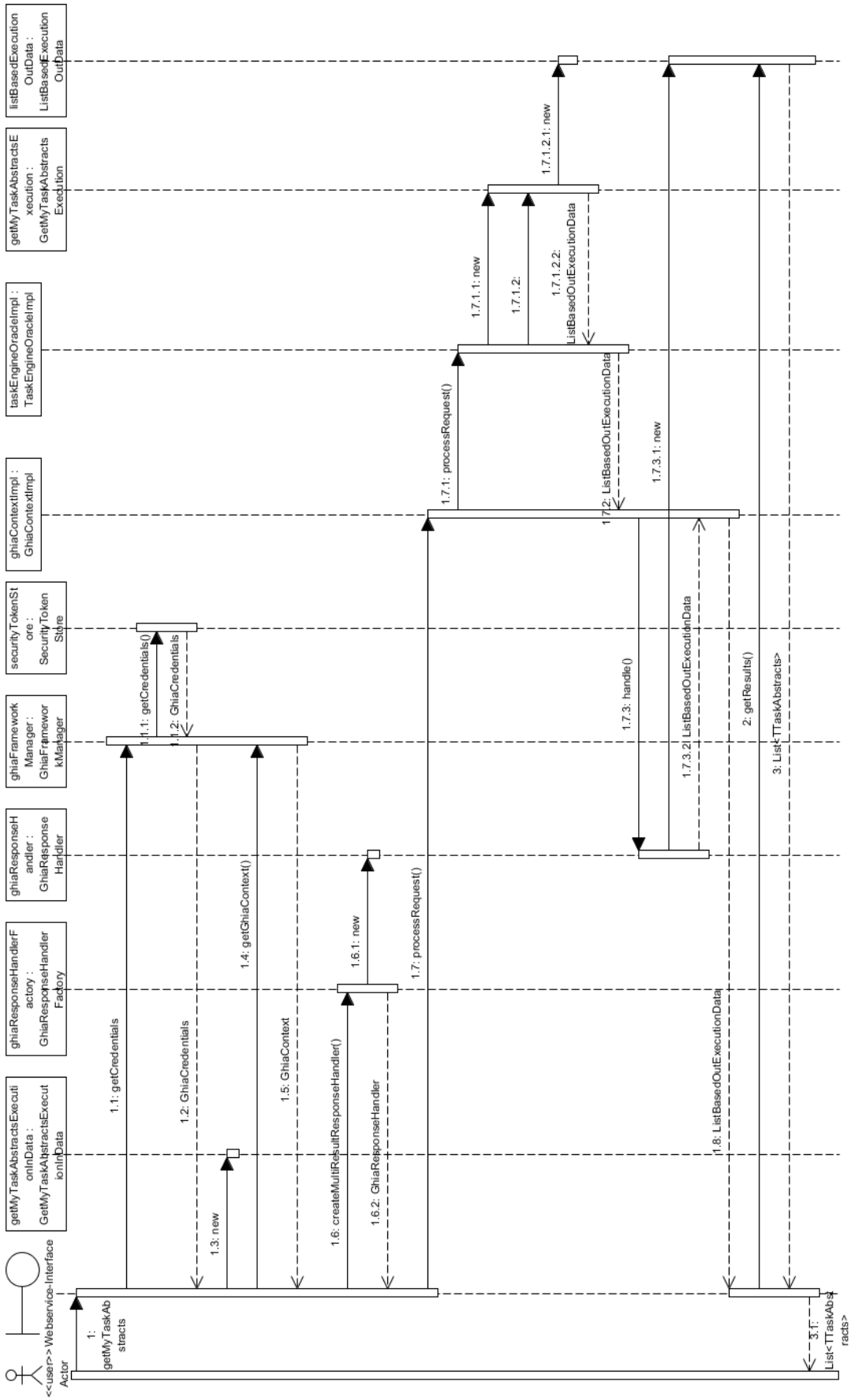

Abbildung 24: Sequenzdiagramm Kommunikation bei Operationsaufrufen

Dieser delegiert den Aufruf dann weiter an die konkrete Task-Engine Implementierung, in diesem Fall die `TaskEngineOracleImpl`.

Die Task-Engine erzeugt ihrerseits die benötigte Execution-Klasse `GetMyTaskAbstractsExecution`, welche die Logik für die Ausführung der Operation `getMyTaskDetails` kapselt. Im Verlauf der Verarbeitung wird eine Instanz der Klasse `ListBasedExecutionOutData` erstellt und als Ergebnis des Operationsaufrufs von der `TaskEngineOracleImpl` an den `GhiaContext` zurückgeliefert. Dieser gibt das, durch den `GhiaResponseHandler` behandelte, Ergebnis an die Webservice-Schnittstelle zurück, die ihrerseits konkreten Ergebnissen von der DTO-Klasse erhält. Diese, hier eine Liste mit `TTaskAbstract` Objekten, werden als Resultat des Operationsaufrufs an die Client-Applikation zurückgegeben.

5.4 Zwischenfazit

Mit Hilfe der im Verlauf dieses Kapitels konzipierten Architektur der Adapterschicht GHIA werden hauptsächlich die in Abschnitt 3.2, Tabelle 4 definierten Qualitätsanforderungen aus den Bereichen Kompatibilität, Portabilität, Flexibilität und Security adressiert. Wie genau die einzelnen Anforderungen aus den verschiedenen Bereichen adressiert werden, wird im Folgenden diskutiert.

Kompatibilität

Der Anforderung Q10, welche die Unterstützung verschiedener Client-Technologien fordert, wird über die Bereitstellung zweier technologisch unterschiedlicher Webserviceschnittstellen, der SOAP- und der REST-Variante, Rechnung getragen. Die Anforderung Q20 wird durch den `GhiaContext` adressiert, der für die Verwaltung der konkreten Task-Engine Implementierungen zuständig ist. Des Weiteren übernimmt dieser die Koordination bei der Verarbeitung von Webservice-Requests, indem diese an die registrierten Task-Engines delegiert werden. Zudem obliegt dem `GhiaContext`, unter Verwendung der `ResponseHandler`, die Behandlung der einzelnen Resultate.

Portabilität

Die Portabilitätsanforderung Q30 verlangt, dass das GHIA-Framework auf allen gängigen Plattformen, sowohl auf einem JEE5-konformen Applikationsserver als auch auf einem reinen Servlet-Container, betrieben werden kann. Da aus kon-

zeptioneller Sicht für die Implementierung des GHIA-Frameworks ausschließlich Bestandteile der Java Standard Edition (JSE) benötigt werden, ist auch diese Forderung erfüllt. Für die Implementierung der Adapterschicht ergibt sich somit die Einschränkung, dass keine JEE-Features, die zwingend den Einsatz eines entsprechenden Applikationsservers voraussetzt, benutzt werden dürfen.

Flexibilität

Die Flexibilität des Frameworks ist Gegenstand der Qualitätsanforderung Q40. Unter Flexibilität wird verstanden, dass das GHIA leicht um weitere spezifische Task-Engine Implementierungen ergänzt werden kann. Wie bereits in Abschnitt 5.2.1, Abbildung 23 gezeigt, ist dies einfach über die Implementierung einer Klasse die von `GhiaBaseTaskEngine` ableitet und den zugehörigen Unterklassen der `GhiaBaseExecution` möglich. Die in diesem Kapitel angestellten Überlegungen hinsichtlich der Konzeption der Adapterschicht werden also auch dieser Anforderung gerecht.

Security

Die Anforderung Q50 bezieht sich auf die Sicherstellung von Authentizität und Berechtigung eines Benutzers, sodass Aktionen hinsichtlich eines Tasks nur für entsprechend berechtigte Benutzter zulässig sind. Diese Anforderung verbleibt im Kern bei der jeweiligen Task-Engine Implementierung. Konzeptionell wird diese insofern adressiert, dass der `GhiaFrameworkManager` in der ersten Ausbaustufe für die Verwaltung der Benutzerinformationen zuständig ist. Eine weitergehende Betrachtung des Themas Security im Rahmen in dieser Abschlussarbeit wird nicht erfolgen, da das Thema zu umfassend ist, um es vollständig und angemessen behandeln zu können.

Wie die Analyse zeigt werden durch die vorliegende Konzeption die Qualitätsanforderungen berücksichtigt. Die Implementierung darf hieran nichts ändern, da die Konzeption als verbindlicher Leitfaden hierfür anzusehen ist. Die Verifikation der in Abschnitt 3.1 definierten funktionalen Anforderungen kann dann nach der Implementierung des GHIA-Frameworks und der entsprechenden Task-Engine-spezifischen Funktionalitäten erfolgen.

6 Implementierung des GHIA-Frameworks

Nachdem im vorherigen Kapitel konzeptionelle Aspekte des Adapterframeworks beschrieben wurden, soll im Folgenden ein Überblick darüber gegeben werden, wie das GHIA-Framework von Implementierungsseite her aufgebaut ist. Hierbei geht es zunächst um allgemeine Aspekte, bevor im nächsten Schritt der strukturelle Aufbau, also die Aufteilung in verschiedene Module, beziehungsweise Projekte, erläutert wird. Im daran anschließenden Abschnitt werden Konfigurationseinstellungen sowie Konventionen, die bei der Erweiterung beziehungsweise der Weiterentwicklung des GHIA-Frameworks zu beachten sind, beschrieben. Das Deployment und die zugehörige Paketierung der Artefakte ist Gegenstand des letzten Abschnitts.

6.1 Allgemeine Aspekte

Dieser Abschnitt beschäftigt sich mit dem allgemeinen Implementierungsrahmen. Hierbei handelt es sich vor allem um technische Aspekte, wie die verwendeten Frameworks sowie das Build- und Dependency-Management.

6.1.1 Verwendete Frameworks und Libraries

In Abschnitt 5.4 wurde bereits die Einschränkung definiert, dass für die Entwicklung des GHIA-Frameworks nur Funktionalitäten der JSE eingesetzt werden dürfen. Folglich wird für die Entwicklung des Frameworks auf den Einsatz von JEE-Features verzichtet. Basis für die Entwicklung ist das JDK 6 (Java Development Kit). Für die Entwicklung der Schnittstellen für die Client-Applikationen werden die Java-Standards JAX-WS[56], für die Realisierung der SOAP-Schnittstelle, beziehungsweise JAX-RS[57], für die Implementierung der REST-Schnittstelle, eingesetzt. Als Webservice-Framework für die Entwicklung kommt in diesem Zusammenhang Apache CXF[58] zum Einsatz, das die beiden Java-Standards in den aktuellen Versionen vollständig unterstützt. CXF selbst basiert im Kern auf dem Spring-Framework und ist unabhängig von JEE-Spezifika.

[56] Nähere Informationen, siehe [JAXWS]
[57] Nähere Informationen, siehe [JSR311]
[58] Nähere Informationen, siehe [CXF]

Alternative Frameworks für die Umsetzung der Schnittstellen wären beispielsweise der Metro-Stack, als JAX-WS Referenzimplementierung von Oracle oder Apache Axis für den SOAP-Bereich gewesen. Für die Umsetzung des REST-Webservices hätten zum Beispiel die JAX-RS Referenzimplementierung Jersey oder das von JBoss zur Verfügung gestellte Framework RESTeasy verwendet werden können. Hauptargument auf Apache CXF als Implementierungsframework für die entsprechenden Webservice-Schnittstellen zu setzen ist, dass mit einem Framework sowohl JAX-WS, als JAX-RS Webservices umgesetzt werden können. Zudem wird die Weiterentwicklung des Frameworks durch eine große und lebhafte Community sowie die Firma Talend, die CXF als Kernkomponente für ihren ESB benutzt und einige der Kern-Commiter beschäftigt, sehr stark getrieben. Infolgedessen ist das Framework entsprechend stabil, robust und bietet einen sehr guten Support durch die Community. Außerdem ist CXF leicht mit dem Spring-Framework integrierbar, da es wie bereits erwähnt auf demselben basiert. Dies ist vor allem dann interessant, wenn neben den Vorteilen der Dependency Injection komplexere Webservices mit Transaktionslogik oder ähnlichem erstellt werden müssen. Hier bietet Spring in vielen Bereichen entsprechende Unterstützung.

Neben dem Einsatz von Apache CXF und der damit verbundenen indirekten Verwendung von Spring, werden keine weiteren Frameworks eingesetzt. Als zusätzliche Libraries kommen weiterhin JUnit, für die Definition von Tests, sowie Google Guava, eine Utiltiy-Sammlung, die unter anderem das Handling von Collections erleichtert, zum Einsatz.

6.1.2 Build- und Dependency-Management

Für das Erstellen der Deployment-Artefakte, den Build, und die automatische Verwaltung der Abhängigkeiten („Depedencies“), also der externen Libraries (JAR-Files, Java Archive) die beispielsweise vom Apache CXF-Framework benötigt werden, wird Apache Maven[59] verwendet.

Die benötigten JAR-Files werden automatisch von Maven aus einem lokalen Repository auf dem eigenen Rechner oder, sollten die entsprechenden Dateien lokal nicht verfügbar sein, aus öffentlich verfügbaren Remote Repositories geladen. Welche Abhängigkeiten bestehen, wird in einer zentralen Datei, der

[59] Nähere Informationen, siehe [MAVEN]

`pom.xml`, festgelegt welche von jedem Maven-Projekt zu definieren ist. In dieser werden neben den Abhängigkeiten weitere Eigenschaften des zu erstellenden Artefaktes sowie des Build-Laufs definiert.

Eine Alternative zu Maven wäre Apache Ant. Ausschlaggebend für die Verwendung von Maven ist letztlich, die automatische Verwaltung der Dependecies in einem zentralen Repository, sodass ein manuelle Suchen, Einfügen und Auflösen benötigter Libraries, unter Berücksichtigung der entsprechenden Versionen, entfällt.

6.2 Struktureller Aufbau

Im Folgenden wird es um den strukturellen Aufbau des GHIA-Frameworks gehen. Hiermit ist der Aufbau der Projektstruktur beziehungsweise die Modularisierung gemeint.

Im Kern besteht GHIA aus den drei Modulen `ghia-wsht-spec`, `ghia-core` und `ghia-ws`. Des Weiteren existieren zurzeit die zwei Module, `ghia-oracle-ht` und `ghia-activiti-ht`, für die Anbindung der spezifischen Task-Engines von Oracle und Activiti. Abbildung 25 visualisiert die Abhängigkeiten und Zusammenhänge der Komponenten untereinander. Abhängigkeiten zu externen Libraries, wie beispielsweise CXF, werden nicht dargestellt. Dies würde das Diagramm unnötig aufblähen und unübersichtlich machen. Für nähere Details zu den Dependencies sei an dieser Stelle auf die `pom.xml`-Dateien verwiesen, in denen die einzelnen Libraries referenziert werden.

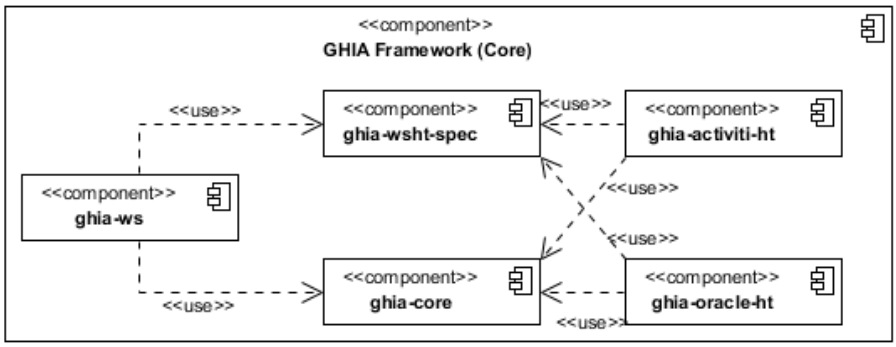

Abbildung 25: Abhängigkeiten der Module des GHIA-Frameworks

Im Kern zeigt die Abbildung, wie die in Abbildung 20 dargestellte Blackbox „Ghia Framework (Core)“ im Inneren aufgebaut ist. Hierbei wird deutlich, dass die Komponenten `ghia-wsht-spec` und `ghia-core` im Fokus stehen und das Herz-

stück des Frameworks bilden. Die Aufgaben der einzelnen Module des GHIA-Frameworks, beziehungsweise deren Verantwortlichkeiten, werden in den nachfolgenden Passagen skizziert.

Im Modul `ghia-wsht-spec` ist die Schnittstellendefinition der, in der WS-HT Spezifikation beschriebenen, SOAP-Schnittstelle enthalten. Zudem sind hier die mit Hilfe des CXF Maven-Plugins aus der WSDL generierten Java-Artefakte, also Klassen für die Datentypen sowie weitere Webservice-Stub Klassen, abgelegt. Diese Separierung von Schnittstellendefinition und der eigentlichen Webservice-Implementierung führt zu einer höheren Flexibilität des Gesamtframeworks, da dieses Modul von fast allen anderen Modulen referenziert und benötigt wird. So zum Beispiel auch von den Modulen zur Anbindung der Task-Engines.

Das Modul `ghia-core` wurde in dieser Ausarbeitung bereits intensiv erläutert und detailliert dargestellt (vergleiche Abschnitt 5.2.1). Es enthält die Kernklassen und –interfaces des GHIA-Frameworks, die beispielsweise für die Anbindung neuer Task-Engines benötigt werden.

Die beiden Module `ghia-oracle-ht` und `ghia-activiti-ht` beinhalten die Logik, welche für die Anbindung der jeweiligen Task-Engine von Oracle beziehungsweise Activiti an die GHIA-Adapterschicht benötigt werden. Wie dies im Detail, in Bezug auf den zu erstellenden Prototyp, funktioniert wurde bereits in Abschnitt 5.2.2 am Beispiel der Oracle Human Workflow-Engine dargestellt. Eine Besonderheit existiert hierbei bezüglich des Einbindens Engine-spezifischer Libraries über Maven. Die benötigten Libraries der Oracle Human Workflow-Engine sind nicht in öffentlichen Maven Repositories verfügbar. Die Installation dieser Dependencies in das lokale Repository muss somit manuell erfolgen.

Schließlich existiert noch das Modul `ghia-ws`, das die Implementierung der Webservice-Schnittstellen für die Client-Applikationen enthält. Dieses enthält sowohl die SOAP-, als auch die REST-Artefakte. Für die REST-Schnittstelle werden die Artefakte der SOAP-Schnittstelle, soweit wie möglich, wiederverwendet. Im Prinzip wurde für die Realisierung des Prototyps nur ein zusätzliches Interface angelegt und dessen Methoden mit entsprechenden JAX-RS Annotationen versehen. Das Aufrufen des Webservices ist somit im SOAP-Stil, wobei als Aus-

tauschformat XML verwendet wird und im REST-Stil, bei dem ein JSON-Datenaustauschformat verwendet wird.

Das GHIA-Framework ist aus Maven-Sicht als Multi-Module-Projekt definiert. Dies bedeutet, dass eine so genannte „Parent POM“ (Project Object Model) existiert, welche mehrere Submodule referenziert. Die Parent POM definiert beispielsweise gemeinsam genutzte Dependencies, die in allen Submodulen verwendet werden. Die Submodule wiederrum definieren eigene POM-Dateien, in denen die, für das Submodul spezifischen, Build-Konfigurationen vorgenommen werden. Die Durchführung eines Builds über die Parent POM führt dazu, dass auch alle definierten Submodule, in der definierten Reihenfolge, gebuilded werden. Einzelbuilds der Submodule sind dabei ebenfalls möglich. Derzeit besteht das GHIA-Framework aus fünf Submodulen (siehe hierzu Abbildung 25).

6.3 Konfiguration und Erweiterung des Frameworks

Der folgende Abschnitt befasst sich mit der Thematik, wie die Konfiguration beziehungsweise die Anbindung spezifischer Task-Engines erfolgen kann und was in diesem Zusammenhang zu beachten ist. Das Vorgehen wird anhand der beiden, im Rahmen dieser Arbeit verwendeten, Engine-Implementierungen von Oracle und Activiti skizziert. Dabei wird auch auf Besonderheiten bei der Anbindung der beiden Engines eingegangen. Die vorgestellte Herangehensweise ist im Anschluss als Anleitung für beliebige Task-Engines adaptierbar.

6.3.1 Implementierung einer Task-Engine

Für jede anzubindende Task-Engine kann entweder ein neues Maven-Projekt angelegt oder die benötigten Klassen können innerhalb eines bestehenden Projektes definiert werden. Aus Gründen der Flexibilität wird für die initiale Entwicklung des GHIA-Frameworks die Variante gewählt, für jede anzubindende Task-Engine ein eigenes Projekt anzulegen.

Das Herzstück bei der Anbindung einer Task-Engine ist eine konkrete Implementierung der abstrakten Basisklasse `GhiaBaseTaskEngine`. Diese Basisklasse definiert drei abstrakte Methoden die zu überschreiben sind:

- `initTaskEngineClient`
  Konfiguration und Aufbau der Verbindung mit der Task-Engine zum Konstruktionszeitpunkt des Objekts.
- `getTaskEngineClient`
  Rückgabe der konkreten Task-Engine Implementierung, gekapselt als `GhiaTaskEngineClient`, welche im Kontext der Ausführung einer konkreten Execution benötigt.
- `getExecutionImplementationPackage`
  Rückgabe des Implementierungspackages für die spezifischen Execution-Klassen als String; wird im Zusammenhang mit der Ermittlung der zu verwendenden Execution benutzt.

Für Beispiele wie die konkrete Implementierung aussieht sei an dieser Stelle auf die beiden Klassen `com.opitzconsulting.ghia.activiti_ht.TaskEngineActivitiImpl` beziehungsweise `com.opitzconsulting.ghia.oracle_ht.TaskEngineOracleImpl` verwiesen.

Neben der Implementierung der konkreten Task-Engine Klasse, müssen zusätzlich noch die konkreten Executions, die von der abstrakten Basisklasse `GhiaBaseExecution` ableiten müssen, definiert werden. Hierbei ist die abstrakte Methode `executeInternal` der Basisklasse zu überschreiben. Exceptions die innerhalb der konkreten Execution auftreten, müssen nicht behandelt werden, da das Exception-Handling konsistent durch die Basisklasse erledigt werden soll. Im ersten Prototyp ist das Exception-Handling allerdings noch recht rudimentär umgesetzt.

Wie bereits in Kapitel 5.2.2 angemerkt wurde, kapseln die Executions die Logik für die Ausführung eines Operationsaufrufs. Die konkreten Implementierungsklassen der `GhiaBaseExecution` sollten in dem über die Methode `getExecutionImplementationPackage` definierten Package liegen, da diese ansonsten zur Laufzeit nicht gefunden werden können. Wird zu einer Operation keine zugehörige Execution-Implementierung gefunden, quittiert das Framework dies mit einer Exception.

Besonderheiten bei der Anbindung der Oracle und Activiti Task-Engine

Für die Initialisierung der Task-Engines in der Methode `initTaskEngineClient` kann für beide Task-Engines auf unterschiedliche Weise erfolgen. Es ist möglich die Engines vollständig programmatisch oder mittels einer XML-Konfigurationsdatei, die beim Aufrufen einer Engine-spezifischen Factory-Methode gelesen wird, zu konfigurieren. Für die Implementierung im GHIA-Framework wurde diesbezüglich die Variante mit den XML-Konfigurationsdateien gewählt, die im Folgenden beschrieben wird. Dies hat ästhetische Gründe, denn so tauchen Infrastrukturdetails, wie zum Beispiel Servernamen oder Ports, nicht direkt im Quellcode auf. Diese sind bei dieser Variante sauber innerhalb einer Konfigurationsdatei gekapselt.

Im Falle der Oracle Human Workflow-Engine wird hierzu eine Datei mit dem Namen „`wf_client_config.xml`“ im Classpath erwartet, in welcher im Idealfall alle verfügbaren Clientarten konfiguriert sind (SOAP, Remote EJB, Local EJB). Zusätzlich dazu wird eine weitere Konfigurationsdatei mit dem Namen „`jps-config.xml`“ benötigt, in der Informationen die OPSS-Konfiguration (Oracle Platform Securtiy Service) betreffend enthalten sind.

Für die Konfiguration der Activiti Process-Engine muss im Classpath eine Datei mit dem Namen „`activiti.cfg.xml`“ liegen. Bei dieser Datei handelt es sich um einen Spring Application-Context[60]. Die Konfiguration der Activiti-Engine erfolgt, im Falle der Verwendung der XML-Datei, also mittels Spring (vergleiche hierzu auch Abschnitt 4.2.2).

6.3.2 Konfiguration einer Task-Engine

Nachdem die benötigten Artefakte für die Anbindung einer Task-Engine wie im vorherigen Abschnitt beschrieben erstellt worden sind, muss dem GHIA-Framework mitgeteilt werden, dass die entsprechende Task-Engine Implementierung existiert, sodass diese bei der Verarbeitung von Anfragen durch den `GhiaContext` berücksichtigt wird.

Zu diesem Zweck existiert im Modul `ghia-core` im Verzeichnis `src/main/resources` die Konfigurationsdatei `ghia-config.properties`. In dieser werden die verwendeten Task-Engines, beziehungsweise deren vollqualifizierte Klassen-

[60] Nähere Informationen, siehe [SPRING]

namen, als Key-Value-Pair hinterlegt. Diese Datei wird bei der Initialisierung des `GhiaContext` ausgelesen und die darin enthaltenen Klassen als konkrete Task-Engine Implementierungen registriert. Die Registrierung erfolgt mit Hilfe der Java Reflection[61] API was der Grund dafür ist, warum in der Konfigurationsdatei der vollqualifizierte Klassenname anzugeben ist. Die Einträge für die Oracle Human Workflow-Engine und die Activiti Process-Engine sehen dabei wie folgt aus:

```
ghia.taskengine.config.oracle=
    com.opitzconsulting.ghia.oracle_ht.TaskEngineOracleImpl

ghia.taskengine.config.activiti=
    com.opitzconsulting.ghia.activiti_ht.TaskEngineActivitiImpl
```

Listing 1: Konfigurationseinträge Oracle und Activiti in ghia-config.properties

Wurde die Implementierung der Task-Engine als separates Modul realisiert, muss dieses als Submodul in der Parent POM registriert werden, sodass das neu hinzugefügte Modul von Maven auch als solches erkannt und in den Build miteinbezogen wird. Die Registrierung erfolgt über das Hinzufügen eines neuen Eintrages in der Datei `ghia-framework/src/pom.xml`. Wie dies aussieht, wird in Listing 2 dargestellt, in dem die Einträge für die Task-Engine Submodule hervorgehoben sind.

```
…
<modules>
  <module>ghia-wsht-spec</module>
  <module>ghia-core</module>
  <module>ghia-oracle-ht</module>
  <module>ghia-activiti-ht</module>
  <module>ghia-ws</module>
</modules>
…
```

Listing 2: Hinzufügen eines neuen Submoduls in der Parent POM

Zu beachten ist hierbei die Definitionsreihenfolge der Module. Wie Abbildung 25 bereits gezeigt hat, bestehen zwischen den Modulen Abhängigkeiten. Demnach ist es wichtig diese auch beim Build zu berücksichtigten. Aus diesem Grund müssen zuerst die Module `ghia-wsht-spec` sowie `ghia-core` gebuilded werden. Zudem ist es wichtig zu beachten, dass das `ghia-ws` Modul zuletzt erstellt wird.

[61] Nähere Informationen in [ULLC11], Kapitel 25 „Reflection und Annotationen“

Warum diese Reihenfolge eingehalten werden müssen, wird im Abschnitt „Deployment und Verteilung“ noch einmal näher erläutert.

Wurde für eine neu anzubindende Task-Engine kein eigenes Submodul erzeugt, ist in Konsequenz auch kein Eintrag in der Parent POM zu ergänzen. Unabhängig davon wie die Task-Engine Anbindung implementiert worden ist, muss in jedem Fall ein Build für das gesamte Projekt durchgeführt und ein neues Deployment vorgenommen werden.

6.3.3 Konventionen

Nachdem sich die vorausgegangenen Abschnitte mit den Bereichen Implementierung und Konfiguration beschäftigt haben, wird es im Folgenden darum gehen, welche Konventionen zu beachten sind, beziehungsweise welche Annahmen zum Beispiel hinsichtlich der Namensgebungen, seitens des Frameworks gemacht werden.

Listing 1 im vorherigen Abschnitt zeigt die Konfigurationseinträge, die in der `ghia-config.properties` für die Registrierung der Task-Engines von Oracle und Activiti gemacht werden müssen. Hierbei ist festgelegt, dass die Property-Key Namen mit dem Wert „`ghia.taskengine.config`“ beginnen müssen. Um die Eindeutigkeit dieses Schlüsseleintrages zu gewährleisten, muss zusätzlich ein eindeutiges Suffix, in diesem Fall der Herstellername der Task-Engine, angehängt werden. Eine Missachtung dieser Konvention führt dazu, dass die betroffene Implementierung nicht im `GhiaContext` registriert und somit bei Request-Verarbeitung ignoriert wird.

Eine weitere Konvention, die beachtet werden muss, besteht in Bezug auf die Klassennamen von konkreten `GhiaExecution`-Klassen und den zugehörigen konkreten `GhiaBaseExecutionInData`-Implementierungen. Aus Gründen der besseren Verständlichkeit des Codes und der Zusammenhänge sollten die DTO-Klassen für eingehende Requests mit dem Operationsnamen in „Camel-Case“-Schreibweise als Präfix definiert werden. Das Ermitteln der auszuführenden `GhiaExecution` wird durch die `GhiaBaseTaskEngine` erledigt. Hierbei wird der Name des eingehenden DTOs auf eine spezifische Execution-Implementierung gemappt, indem das Suffix „InData“ des DTO-Namens entfernt wird. Hieraus ergibt sich der Name der auszuführenden Execution-Klasse. Mit Hilfe des, über die Me-

thode `getExecutionImplementationPackage` festgelegten, Package-Namens wird der Klassenname vollqualifiziert, wodurch mit Java Reflection die konkrete `GhiaExecution` instanziiert werden kann. Zusammenfassend betrachtet ergibt sich für die Benennung einer konkreten `GhiaExecution` beziehungsweise deren `GhiaBaseExecutionInData` die Namenskonvention:

- `<Operationsname>Execution`
- `<Operationsname>ExecutionInData`

Wird diese Konvention nicht beachtet, kann die richtige Execution nicht ermittelt werden, was zu einer Exception führt. Allerdings hat die Konvention des Operationsnamen als Präfix ästhetischen Charakter und muss nicht zwingend befolgt werden, solange die Namen dem dargestellten Muster folgen. Der Platzhalter `<Operationsname>` ist somit flexibel und beliebig definierbar.

6.4 Deployment und Verteilung

Nachdem bereits Aspekte aus den Bereichen Implementierung und Konfiguration angesprochen wurden, wird im Folgenden beschrieben wie Deployment und Verteilung im Falle des GHIA-Adapterframeworks aussehen.

In einem der vorausgegangenen Abschnitte (vergleiche Abschnitt 6.2) sind die einzelnen Module des Frameworks bereits vorgestellt und erläutert worden. Die einzelnen Module des GHIA-Frameworks werden für das Deployment in Form eines JEE-Standardkonformen WAR-Files[62] (Web Archive) zusammengepackt. Die Bestandteile dieser Datei wird in Abbildung 26 skizziert.

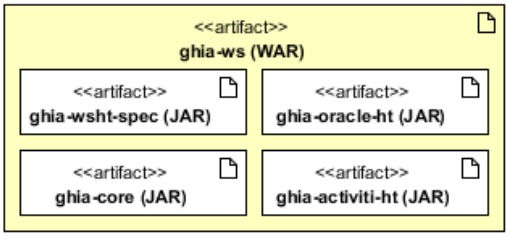

Abbildung 26: GHIA Deploymentarchiv

Hierbei ist zu beachten, dass das WAR-File nicht einfach nur die JAR-Files enthält, sondern zusätzlich verschiedene Class- sowie Konfigurationsdateien, wie beispielsweise Deployment Deskriptoren. Weiterhin werden die JAR-Files standardmäßig im Verzeichnis `WEB-INF/lib` des WAR-Archivs abgelegt.

[62] Weiterführende Informationen zu dem WAR-Deploymentformat unter [OJEE01]

In der Abbildung ist weiterhin erkennbar, dass – außer dem GHIA-Submodul `ghia-ws` – alle anderen Submodule in Form eines JAR-Files vorliegen, zu denen das WAR-File direkte oder indirekte Abhängigkeiten besitzt. Die Abhängigkeiten der einzelnen Artefakte untereinander wurden bereits in Abbildung 25 visualisiert. Weiterhin bestehen Abhängigkeiten zu den Libraries des verwendeten CXF Frameworks sowie zu Task-Engine spezifischen Libraries, die hier nicht dargestellt sind.

Da es sich bei dem erstellten WAR-Artefakt um ein Java-Standard Archiv handelt ist es theoretisch auch einfach auf jedem JEE-konformen Applikationsserver und auch einem Servlet-Container deploybar. Theoretisch insofern, als dass es durch Applikationsserver-Spezifika zu Classloading-Problemen kommen kann, die mit Hilfe systemspezifischer Deployment-Deskriptoren behoben werden müssen. Dieses Problem zunächst außer Acht gelassen, ergeben sich für das Deployment des GHIA-Frameworks, verschiedene Möglichkeiten, was die Verteilung angeht.

Abbildung 27 zeigt eine Variante bei der die Activiti Process-Engine als Teil des GHIA-Frameworks in derselben JVM deployed wird. Die Laufzeitdaten zu Tasks oder BPM-Prozessinstanzen werden in einer solchen Konfiguration in einer In-Memory Datenbank abgelegt. Redeployments oder Neustarts des Servers haben in dieser Konstellation die Konsequenz, dass alle Laufzeitdaten verloren gehen.

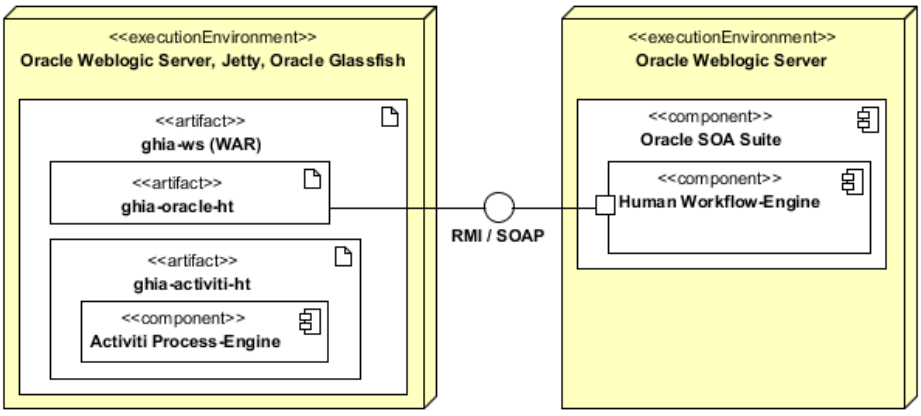

Abbildung 27: Deployment und Verteilung (Variante 1 - Activiti Lokal)

Als Laufzeitumgebung („Execution Environment“) kommen verschiedene Server in Frage, was im Diagramm durch die verschiedenen Serverbezeichner angedeutet werden soll. Diese Liste kann im Prinzip beliebig erweitert werden. Die Kommunikation mit der Activiti Process-Engine kann in diesem Fall nativ über

die Java-Schnittstelle erfolgen. Die Interaktion mit der Oracle Human Workflow-Engine ist hingegen nur via SOAP oder RMI möglich, da diese auf einem anderen Server deployed ist.

In einer anderen Variante, die in Abbildung 28 visualisiert wird, werden sowohl die Oracle, als auch die Activiti Engine in einer eigenen Laufzeitumgebung betrieben.

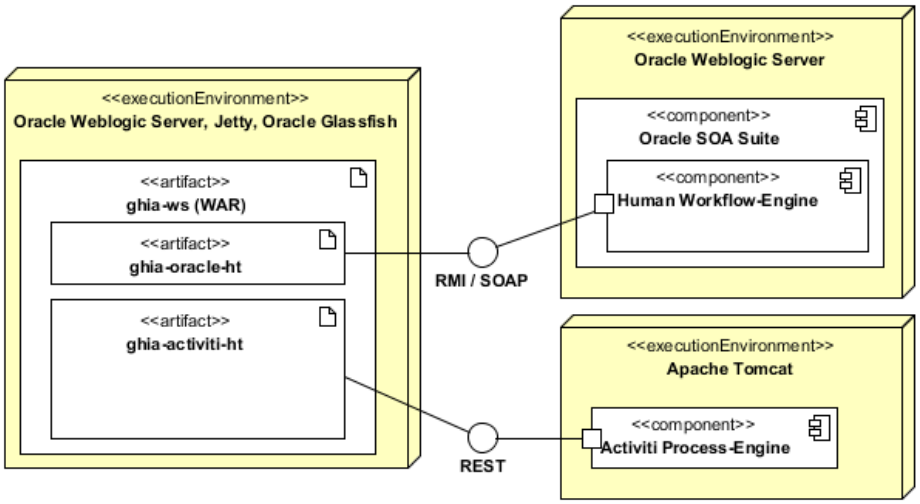

Abbildung 28: Deployment und Verteilung (Variante 2 - Activiti Remote)

In einer solchen Konstellation, kann die Kommunikation ausschließlich über die Remote-Schnittstellen, RMI oder SOAP bei Oracle und REST bei Activiti, erfolgen.

Grundsätzlich sind auch noch weitere Szenarien möglich. So können beispielsweise auch alle Artefakte und Komponenten gemeinsam auf einem Server, in diesem Fall auf einem Oracle Weblogic Server, auf dem auch die BPM Suite betrieben wird, in einer JVM deployed werden. In diesem Fall könnte die Kommunikation ausschließlich über die lokalen Java-Schnittstellen der beiden Engines erfolgen. Zudem könnte auch nur die Activiti Process Engine auf einem separaten Server deployed sein, wohingegen das GHIA-Framework auf dem Oracle BPM Server verbleibt.

Für die Implementierung des im Rahmen dieser Arbeit entstehenden Prototyps wird Deployment- beziehungsweise Verteilungs-Variante 1, dargestellt in Abbildung 27, verwendet. Hierbei besteht der Vorteil die Java-Schnittstelle der Activiti-Engine benutzen zu können, die mehr Funktionalitäten zur Verfügung stellt als ihr REST-Pendant und zudem im ersten Schritt leichter realisierbar ist.

7 Test und Validierung

Konzeption und Implementierung des GHIA-Frameworks sind nunmehr prototypisch abgeschlossen. Ob die in Kapitel 3 definierten Anforderungen hierbei auch adressiert werden konnten, wird in diesem Kapitel dargelegt. Dabei wird zunächst das Testvorgehen beziehungsweise der Testablauf beschrieben, bevor im Anschluss die funktionalen Anforderungen (vergleiche Abschnitt 3.1), unter Berücksichtigung der definierten Abnahmekriterien (vergleiche Abschnitt 3.3), überprüft werden. Daran anschließend erfolgt noch die Überprüfung für die Qualitätsanforderungen, die nicht allein von konzeptioneller Seite her abgedeckt werden können.

7.1 Vorgehen und Ablauf

Wie bereits im Rahmen der Definition der Abnahmekriterien beschrieben wurde, erfolgt die Validierung der funktionalen Anforderungen in zwei Schritten. Es erfolgt eine technische Prüfung mit deren Hilfe der die Funktionsfähigkeit des Frameworks sichergestellt werden soll. Dazu wird jede implementierte Operation, also jede spezielle `GhiaExecution`, mit einem JUnit-Test hinterlegt. Weiterhin erfolgt ein „End-to-End“-Test indem, für das in Abschnitt 3.4. definierte PoC-Szenario, eine Beispielapplikation implementiert wird. Hierbei wird das GHIA-Framework in seiner Zusammenarbeit mit den spezifischen Task-Engines sowie einer Client-Applikation getestet.

7.1.1 Technische Testfälle

Die technischen Testfälle sind, wie bereits erwähnt, als JUnit-Tests implementiert. Diese werden beim Build der Anwendung, sofern dies nicht explizit ausgeschlossen wird, automatisiert von Maven ausgeführt. Kommt es bei der Ausführung dieser Tests zu einem Fehler, schlägt auch der Build-Lauf fehl. Derzeit konzentrieren sich die Tests auf die Funktionalitäten der einzelnen Executions.

Für die Ausführung eines Tests im Falle der Oracle BPM Suite muss beachtet werden, dass eine entsprechende Verbindung zum BPM Suite-Server besteht. Auf dem Zielserver muss weiterhin eine Beispiel-Applikation bereitgestellt werden, welche der Erstellung von Tasks dient. Pro Testfall wird dabei ein eigener Task erstellt der, wenn nötig, für das jeweilige Testszenario vorbereitet wird. Im Anschluss daran erfolgt die Ausführung der eigentlichen Testlogik. Im

Falle der Activiti BPM Platform existiert diese Einschränkung nicht, da es sich bei dieser um eine leichtgewichtige Komponente handelt, die für Testzwecke In-Memory hochgefahren und im Rahmen eines Tests benutzt werden kann. Es ist keine schwergewichtige Laufzeitumgebung, wie im Falle der Oracle BPM Suite, notwendig. Dadurch bedingt weisen die Testfälle auch Laufzeitunterschiede auf. Während die Laufzeiten der Activiti-Operationen im Millisekunden-Bereich liegen, betragen die Laufzeiten im Falle der Oracle BPM Suite derzeit mehrere Sekunden, was zum Beispiel dem Aufbau einer Serververbindung geschuldet ist.

Die definierten JUnit-Tests werden im jeweiligen Modul unter `src/test/java` abgelegt, werden hier per Konvention in derselben Package-Struktur wie die eigentlichen Implementierungsklassen organisiert und erhalten auch denselben Klassennamen mit dem Suffix „Test“.

Referenz	Name	Engine	Vorbedingungen
TT 10	AddAttachmentExecutionTest	Oracle	-
TT 15	AddCommentExecutionTest	Oracle	-
TT 20	ClaimExecutionTest	Oracle	-
TT 25	CompleteExecutionTest	Oracle	Übernahme der Verantwortung für einen Task
TT 30	DeleteAttachmentExecutionTest	Oracle	Vorhandensein eines Attachments
TT 35	DeleteCommentExecutionTest	Oracle	Vorhandensein eines Kommentares
TT 40	GetAttachmentExecutionTest	Oracle	Vorhandensein eines Attachments
TT 45	GetCommentsExecutionTest	Oracle	Vorhandensein eines Kommentares
TT 50	GetMyTaskAbstractsExecutionTest	Oracle, Activiti	-
TT 55	GetMyTaskDetailsExecutionTest	Oracle, Activiti	-
TT 60	GetTaskDetailsExecutionTest	Oracle	-
TT 65	NominateExecutionTest	Oracle	-
TT 70	ReleaseExecutionTest	Oracle	Übernahme der Verantwortung für einen Task
TT 75	ResumeExecutionTest	Oracle	Übernahme der Verantwortung für einen Task; Unterbrechen der Task-bearbeitung
TT 80	SetPriorityExecutionTest	Oracle	-
TT 85	SkipExecutionTest	Oracle	Übernahme der Verantwortung für einen Task
TT 90	SuspendExecutionTest	Oracle	Übernahme der Verantwortung für einen Task
TT 95	GetTaskHistoryExecutionTest	Oracle	-

Tabelle 16: Execution-Testfälle

Derzeit existieren, was die Executions angeht, die in Tabelle 16 dargestellten technischen Testfälle. Die Spalte „Name“ enthält dabei den Namen der Testklas-

se, die Spalte „Engine“ die Task-Engine für die ein Test und damit auch eine entsprechende Execution vorhanden ist. In der letzten Spalte „Vorbedingung“ finden sich Informationen darüber welche Ausgangssituation gegeben sein muss, um eine Operation erfolgreich ausführen zu können. Dabei wird das Vorhandensein eines Tasks implizit vorausgesetzt und somit nicht explizit als Vorbedingung aufgeführt. Derzeit werden in den Testfällen nur die Gutfälle betrachtet und abgedeckt.

Die meisten Testfälle und dementsprechend auch die meisten Executions sind, wie der Tabelle entnommen werden kann, derzeit für die Oracle BPM Suite implementiert. Dies liegt darin begründet, dass für den Prototyp in beiden Engines nur die Funktionalitäten für das Abfragen von Tasks vorhanden sein müssen (vergleiche Abschnitt 3.1.2, Tabelle 2, FQ 80). Für die Validierung der übrigen Anforderungen reicht die Implementierung einer Funktionalität für eine spezifische Task-Engine aus. Dass die Wahl hier auf die Oracle BPM Suite als führendes System gefallen ist, hat mit der Praxisrelevanz der Plattform zu tun, die höher ist als die der Opensource BPM Plattform von Activiti.

7.1.2 „End-to-End“-Test

Wie eingangs bereits erwähnt wird bezüglich des „End-to-End“-Tests das PoC-Szenario benutzt, um die Anforderungen aus der Perspektive eines Anwenders, unter Verwendung einer Client-Applikation die über das GHIA-Framework mit den Task-Engines interagiert, zu validieren.

Vom Vorgehen her werden hierbei verschiedene Szenarien, die innerhalb des Schadenregulierungsprozesses auftreten können, durchgespielt. Anhand dessen erfolgt die Überprüfung der definierten Abnahmekriterien. Szenarien die diesbezüglich zu durchlaufen sind, werden in Tabelle 17 dargestellt. Zu einem Testszenario wird eine knappe Beschreibung darüber gegeben, was getan werden soll (Spalte „Beschreibung“) und wer für die Ausführung zuständig ist (Spalte „Verantwortlichkeit“). Die Verantwortlichkeiten beziehen sich dabei auf die im Use-Case Diagramm in Abschnitt 3.4 identifizierten Akteure.

Referenz	Beschreibung	Verantwortlichkeit
TS 10	Auswahl eines zu bearbeitenden Schadenfalles, aus einer Liste aller offener Fälle, zur Prüfung	Sachbearbeiter
TS 20	Neuzuweisung eines Bearbeiters, aufgrund einer Erkrankung eines bisherigen Bearbeiters	Abteilungsleiter

TS 30	Abschluss der Prüfung und des Schadenfalles	Sachbearbeiter
TS 40	Abschluss der Prüfung mit Konsultation eines externen Gutachters; Begründung der Konsultation durch Kommentierung und gegebenenfalls Anhängen von Dokumenten	Sachbearbeiter
TS 50	Übernahme eines Schadenfalles zur Begutachtung des dargelegten Sachverhalts	Externer Gutachter
TS 60	Anhängen des Gutachtens als Datei; Abschluss des Gutachten	Externer Gutachter
TS 70	Überprüfung abgeschlossener Schadensfälle	Abteilungsleiter
TS 80	Erhöhung der Priorität, sodass ein Schadensfall vorrangig bearbeitet wird	Abteilungsleiter
TS 90	Wiederabgeben der Verantwortung für einen Task, da dieser fälschlicherweise übernommen wurde	Sachbearbeitung

Tabelle 17: Testszenarien für den PoC

Die dargestellten PoC-Szenarien sind technologieunabhängig beziehungsweise unabhängig von einer Task-Engine definiert. Generell ist es so, dass sowohl die Activiti- als auch die Oracle-Engine mit Hilfe des GHIA-Frameworks angebunden werden können. Einschränkung hierbei ist, dass alle schreibenden Operationen nur für Oracle-Tasks ausgeführt werden. Für die Demonstration lesender Operationen werden in Activiti einige Dummy-Tasks angelegt, um zu zeigen, dass beispielsweise die Synchronisation verschiedener Postkörbe mit Hilfe des Frameworks transparent funktioniert.

7.2 Validierung der funktionalen Anforderungen

In diesem Abschnitt soll die Adressierung der Abnahmekriterien mit den im vorherigen Abschnitt beschriebenen technischen Testfällen sowie den Testszenarien des „End-to-End“-Tests aufgezeigt werden. Auf Grundlage dieser Gegenüberstellung können eine Bewertung des Erfüllungsgrades der Anforderungen und darüber hinaus auch eine Beurteilung der Implementierung und Konzeption des GHIA-Frameworks erfolgen.

Tabelle 18 gibt diesbezüglich einen Überblick, welche Abnahmetestfälle (vergleiche Abschnitt 3.3, Tabelle 5) durch welche JUnit-Tests (vergleiche Abschnitt 7.1.1, Tabelle 16) und Testszenarien (vergleiche Abschnitt 7.1.2, Tabelle 17) adressiert werden. Weiterhin gibt die Tabelle Auskunft über den Status des Abnahmetestfalls (Spalte „Referenz TF“) sowie der einzelnen JUnit-Tests (Spalte „Referenz TT“) und Testszenarien (Spalte „Referenz TS“). Im Falle der TT beziehungsweise der TS wird der Status durch eine entsprechende farbliche Markie-

rung angezeigt. Die Farbe Grün steht dabei für „Akzeptiert“ und die Farbe Rot für „Nicht Akzeptiert“. Besonderheiten die sich in Bezug auf einen Abnahmetestfall, beziehungsweise die hiermit in Verbindung stehenden JUnit-Tests oder Testszenarien ergeben, sind in der Spalte „Bemerkung“ entsprechend vermerkt.

Referenz TF	Referenz TT	Referenz TS	Gesamtstatus	Bemerkung
TF 10	TT 65 TT 20	TS 20 TS 50	Akzeptiert	-
TF 15	TT 25	TS 30 TS 40 TS 60	Akzeptiert	-
TF 20	TT 10 TT 15	TS 40 TS 60	Fehlgeschlagen	Speichern von Attachments: Speichern der Metadaten funktioniert, allerdings werden die Binärdaten nicht mitabgespeichert.
TF 25	TT 30 TT 35	-	Akzeptiert	Löschen von Kommentaren nicht möglich, weder bei Activiti noch bei Oracle.
TF 30	TT 60	TS 30 TS 40 TS 50 TS 60	Akzeptiert	-
TF 35	TT 50 TT 55	TS 10 TS 50 TS 70	Akzeptiert	-
TF 40	TT 75	TS 70	Akzeptiert	-
TF 45	-	-	Akzeptiert	Neuangelegte Tasks werden implizit aktiviert.
TF 50	TT 65	TS 20	Akzeptiert	-
TF 55	TT 80	TS 80	Akzeptiert	-
TF 60	TT 70	TS 90	Akzeptiert	-

Tabelle 18: Abbildung JUnit-Tests und Testszenarien auf Abnahmetestfälle

Wie aus der Tabelle hervorgeht, konnte ein Großteil der Abnahmetestkriterien voll erfüllt werden. Zu den Kriterien, bei denen Einschränkung, Bemerkungen oder Besonderheiten bestehen, sollen im Folgenden noch einige Ausführungen gemacht werden.

Bezüglich des Abnahmekriteriums TF 20 ist anzumerken, dass das Speichern von Kommentaren sowie der Attachments grundsätzlich funktioniert. Wie bereits in der Bemerkung beschrieben ist, werden die Binärdaten allerdings nicht übernommen. Die Ursache des Problems konnte nicht klar eingegrenzt werden. Grundsätzlich wird diese Funktionalität von der Oracle Human Workflow-Engine angeboten, sodass die Vermutung naheliegt, dass es sich um ein Problem der `AddAttachementExecution`-Implementierung handelt. Da dies eine wesentli-

che Einschränkung der intendierten Funktionalität darstellt, wird das Abnahmekriterium als „Fehlgeschlagen“ deklariert“.

Im Falle des Abnahmekriteriums TF 25, bezüglich des Entfernens von Kommentaren, handelt es sich um ein Problem der verwendeten Task-Engines, das unabhängig von der Funktionalität des GHIA-Frameworks ist. Würden die Engines diese Operation unterstützen, hätte die Anforderung auch entsprechend umgesetzt werden können. Da aber das ebenfalls geforderte Löschen von Attachements funktionieren, und das Löschen von Kommentaren nur aufgrund technischer Limitationen nicht funktioniert, wird das Kriterium TF 25 im Gesamtstatus mit „Akzeptiert“ bewertet.

Die Funktionalität des expliziten Aktivierens eines Tasks, welche in TF 45 gefordert wird, ist ebenfalls nicht über die Engine-API von Oracle oder Activiti verfügbar. Dies wird implizit von den Task-Engines übernommen und soll nicht explizit durch einen externen Akteur, sei es ein menschlicher Akteur oder ein weiterer Service, erfolgen können. Ähnlich wie im Fall von TF 25 könnte das Framework die geforderte Funktionalität aber abdecken, sofern diese von den Task-Engines angeboten würden. Aus diesem Grund wird auch dieser Testfall als „Akzeptiert“ gekennzeichnet.

7.3 Validierung der Qualitätsanforderungen

Nachdem im vorherigen Abschnitt ausschließlich die Validierung der funktionalen Anforderungen erfolgt ist, soll im Folgenden die Validierung der Qualitätsanforderungen vorgenommen werden. Die hierzu gehörigen Abnahmekriterien sind in Abschnitt 3.3, Tabelle 18 definiert.

Wie bereits in Abschnitt 5.4 angesprochen, konnten einige Anforderungen dieses Bereichs und damit auch die Abnahmekriterien bereits auf konzeptioneller Ebene adressiert werden. Die Qualitätsanforderungen hinsichtlich Flexibilität und Kompatibilität (vergleiche Abschnitt 3.2, Tabelle 4 QA 20 und QA 40) werden bereits durch die Konzeption des GHIA-Frameworks komplett abgedeckt und auch im Rahmen der funktionalen Abnahmetest verifiziert. Ein Beispiel ist das Abfragen von Taskdaten der beiden Task-Engines von Oracle und Activiti. Somit ist das Abnahmekriterium TQ 20 (vergleiche Abschnitt 3.3, Tabelle 7) bereits validiert und somit auch erfüllt. Auch der Qualitätsanforderung QA 10

wurde schon in der Konzeption durch die Definition einer SOAP- und einer REST-Webserviceschnittstelle Rechnung getragen. Allerdings muss diese noch zusätzlich durch die Verifizierung mit Hilfe funktionaler Tests abgedeckt werden. Dazu muss sichergestellt werden, dass die in Tabelle 17 definierten Testszenarien auch über die REST-Schnittstelle funktionieren. Hierbei ist anzumerken, dass es für die Erfüllung dieser Anforderung im ersten Schritt ausreicht nur einen Teil der Funktionalitäten über die REST-Schnittstelle abzudecken.

Das einzige Abnahmekriterium im Bereich der Qualitätsanforderung, das in diesem Abschnitt noch betrachtet werden muss da es in keinem der vorhergehenden Kapitel adressiert wurde, ist die Portabilität. Von Konzeptions- und Implementierungsseite her besteht diesbezüglich die Aussage, dass das GHIA-Framework grundsätzlich auf jeder Laufzeitumgebung deploybar sein muss, da keine JEE-spezifischen Features verwendet werden. Zur Validierung dieser These wird das GHIA-Framework beziehungsweise der GHIA-Adapterservice auf unterschiedlichen Laufzeitumgebungen bereitgestellt. Anschließend werden Beispielanfragen durchgeführt. Liefern diese Beispielanfragen ein entsprechendes Ergebnis zurück, wird der Testfall als erfolgreich gewertet.

Server	JEE	Status	Konfiguration / Bemerkungen
Apache Tomcat Server v. 7.0.19	-	Fehlerhaft	Konfiguration: Standard; Bemerkung: Classloading-Probleme
Mortbay Jetty v. 6.2.26	-	Akzeptiert	Konfiguration: Standard, gestartet über Maven Jetty-Plugin
Oracle Glassfish 3.1.2 (Community Edition)	✓	Akzeptiert	Konfiguration: Standard
Oracle Weblogic 11g (v. 10.3.6)	✓	Akzeptiert	Konfiguration: Weblogic-Domain mit einem Administration-Server und einem Managed-Server (keine SOA Suite, kein OSB)
Oracle Weblogic 12c	✓	Akzeptiert	Konfiguration: Weblogic-Domain mit einem Administration-Server und einem Managed-Server

Tabelle 19: Ergebnisse Portabilitätstests

Tabelle 19 zeigt die für den Test verwendeten Server und gibt Auskunft darüber, ob es sich um einen vollwertigen JEE-Laufzeitcontainer handelt oder nicht. Des Weiteren werden der Teststatus sowie, im Falle von Besonderheiten, eine entsprechende Bemerkung angegeben. Die dargestellte Liste beinhaltet natürlich bei weitem nicht alle am Markt vorhandenen Server. Hier jedoch alle aufzu-

führen und in den Test miteinzubeziehen wäre nur mit einem hohen Aufwand zu bewerkstelligen. Die Auswahl beschränkt sich daher auf die Server, die zum Zeitpunkt der Erstellung dieser Abschlussarbeit in entsprechenden Testumgebungen zur Verfügung standen.

Wie die Tabelle zeigt, ist ein Testfall fehlgeschlagen. Ursächlich hierfür sind, wie auch in der Bemerkung vermerkt, Classloading-Probleme. Das Deployment des GHIA-Frameworks beziehungsweise des GHIA-Webservices funktioniert zwar problemlos, zur Laufzeit kommt es dann zu Fehlermeldungen, da der Server einige Klassen nicht im Classpath finden kann. Die Erwartungshaltung, dass es zu derartigen Problemen kommen kann, wurde bereits in Abschnitt 6.4 formuliert. Für die Lösung des Problems bedarf es einer eingehenden Analyse sowie eine Anpassung des Deployments für den Tomcat Servlet-Container. Dies soll aber nicht im Rahmen der vorliegenden Ausarbeitung geschehen. Trotz dieses Problems, lassen die ermittelten Ergebnisse jedoch den Schluss zu, dass das GHIA-Framework grundsätzlich in jede Laufzeitumgebung deployed werden kann. Das Abnahmekriterium TQ 10 ist somit auch verifiziert.

8 Abschlussbetrachtung

Im letzten Kapitel dieser Ausarbeitung werden die erarbeiteten Erkenntnisse und Ergebnisse zusammengefasst. Zudem erfolgt eine Bewertung der Arbeitsergebnisse hinsichtlich der in Abschnitt 1.2 definierten Zielsetzung. Außerdem werden mögliche praktische Einsatzgebiete und Anwendungsszenarien für das GHIA-Framework skizziert. Abschließend werden mögliche und sinnvolle Erweiterungen sowie Verbesserungen des Frameworks betrachtet.

8.1 Zusammenfassung

In der Erstellung des GHIA-Frameworks bestand das Hauptziel der vorliegenden Arbeit. Mit Hilfe des Frameworks wird die standardbasierte Integration menschlicher Interaktion in Geschäftsprozesse, beziehungsweise allgemein in Service-orientierte Architekturen, ermöglicht. Dieses Ziel wurde weiter in die in Abschnitt 1.2 beschriebenen Teilziele

- Konzeption auf Basis der WS-HT Spezifikation,
- Verifizierung der Konzeption mit Hilfe einer prototypischen Implementierung und
- Bewertung der Praxistauglichkeit der WS-HT Spezifikation als generische Fassade für Task-Engines

untergliedert.

Die Definition von funktionalen Anforderungen und Qualitätsanforderungen war, neben den Vorgaben der WS-HT Spezifikation, die Basis für Konzeption und Implementierung des GHIA-Frameworks. Für die Simulation eines realen Anwendungsszenarios wurde ein PoC-Szenario definiert, dessen Implementierung Aussagen in Richtung der Praxistauglichkeit des GHIA-Frameworks und der WS-HT Spezifikation erlaubt.

Um Aspekte zu identifizieren, die im Rahmen der Konzeption und Implementierung des Weiteren berücksichtigt werden mussten, wurde ein Vergleich der verwendeten Human Workflow-Engines von Oracle und Activiti gegen die WS-HT Spezifikation durchgeführt. Relevante Punkte in diesem Zusammenhang waren zum Beispiel das Schnittstellendesign für Client-Applikationen und die von den Schnittstellen zur Verfügung gestellten Operationen. Das Ergebnis der Ana-

lyse war, dass sich die Task-Engines untereinander sowie im Vergleich mit der Spezifikation hinsichtlich Schnittstellenarchitektur, Funktionsumfang und auch bezüglich der ausgetauschten Datenstrukturen voneinander unterscheiden.

Die, unter Berücksichtigung dieser Analyseergebnisse, durchgeführte Konzeption lieferte im Ergebnis eine Soll-Architektur (siehe hierzu Abschnitt 5.1.2, Abbildung 20), Schnittstellen für die Anbindung beliebiger Task-Engines sowie Schnittstellen für die Anbindung verschiedener Client-Applikationen. Innerhalb des Frameworks und in Richtung der Client-Applikationen erfolgt die Kommunikation unter Verwendung der in WS-HT spezifizierten Datenstrukturen. Die proprietären Hersteller-APIs werden hierbei ausschließlich in den spezifischen Engine-Implementierungen verwendet. Eine Erkenntnis der Konzeption war, dass die Qualitätsanforderungen aus konzeptioneller Sicht bereits vollständig adressiert werden konnten.

Das Ergebnis der Umsetzung der definierten Soll-Architektur war eine erste prototypische Version des GHIA-Frameworks. In dieser ersten Version lag das Hauptaugenmerk auf der Implementierung des Core-Frameworks sowie der SOAP-Schnittstelle nach WS-HT. Die REST-Schnittstelle befindet sich derzeit noch im Aufbau und bietet nur eine sehr rudimentäre Unterstützung der in dieser Arbeit beschriebenen Funktionalitäten. Als Erkenntnis der Implementierung kann festgehalten werden, dass neue Task-Engines durch den modularen sowie flexiblen Aufbau des GHIA-Frameworks schnell und mit wenig Aufwand angebunden werden können.

Abschließend zeigte die Validierung des implementierten Prototyps, gegen die definierten Anforderungen, dass ein Großteil der Anforderungen abgedeckt ist. Die im Rahmen der Konzeptionsphase gewonnen Einschätzung hinsichtlich der Abdeckung der Qualitätskriterien wurde im Zuge der Validierung ebenfalls bestätigt.

8.2 Fazit

Wie das Validierungskapitel zeigt, sind 10 der insgesamt 11 Abnahmekriterien aus dem Bereich der funktionalen Anforderungen und 4 der insgesamt 5 Abnahmekriterien aus dem Bereich der Qualitätsanforderungen positiv validiert worden. Mit Hilfe des umgesetzten PoC-Szenarios der Schadensregulierung

konnte zudem für einen einfachen Anwendungsfall die Praxistauglichkeit des Frameworks und damit auch der WS-HT Spezifikation unter Beweis gestellt werden. Als Gesamtresümee kann somit festgehalten werden, dass das in Abschnitt 1.2 definierte Ziel, die Erstellung des GHIA-Frameworks und seiner Teilziele vollständig erreicht worden ist.

Mit Hilfe des GHIA-Frameworks ist es möglich die Abhängigkeit zu proprietären APIs aufzulösen. Weiterhin ist das Framework zukunfts- und investitionssicher, da standardbasierte Technologien und Protokolle dessen Grundlage bilden. Zudem besteht ein verbindlicher und robuster Servicekontrakt, der unabhängig von der unterliegenden spezifischen Task-Engine ist. Neben diesem Entkopplungsaspekt ergeben sich für das GHIA-Framework aus Praxissicht weitere Einsatzszenarien, die im Folgenden erläutert werden.

Mit Hilfe des GHIA-Frameworks besteht die Möglichkeit mehrere Task-Engines unterschiedlicher Hersteller anzubinden. Die proprietären Engines werden in einer solchen Konstellation über eine standardbasierte Schnittstelle angesprochen. Die Taskbearbeitung ist somit transparent über eine einheitliche Client-Applikation möglich. Dieses Szenario könnte zum Beispiel für Unternehmen interessant sein, die, bedingt durch Unternehmenszukäufe, verschiedene Task-Engines im Einsatz haben. In solchen Szenarien ist es aufgrund der inhärenten Komplexität der vorhandenen IT-Systemlandschaften in der Regel nicht möglich diese komplett zu reorganisieren. Bereits bestehende Task und die zugehörigen Prozessinstanzen auf eine dedizierte Plattform zu migrieren, ist in diesem Zusammenhang zu aufwendig. Um diese Komplexität überschaubar und beherrschbar zu machen werden heterogene Systeme in solchen Szenarien, beispielsweise unter Verwendung eines ESB, integriert. Das GHIA-Framework könnte hierbei als Integrationslösung für eventuell vorhandene Human Workflow-Engines eingesetzt werden.

Neben den genannten Szenarien der Entkopplung und der Integration ist es weiterhin möglich das GHIA-Framework gezielt in Migrationsszenarien einzusetzen. In solchen Szenarien sind sowohl der Entkopplungsaspekt als auch der Integrationsaspekt von elementarer Bedeutung für einen sauberen und unkomplizierten Migrationsverlauf. Weil sich im Falle von Release-Wechseln, unabhängig davon ob es sich um ein Major- oder ein Minor-Release handelt, Ände-

rungen an den proprietären Schnittstellen der Task-Engines ergeben können, ist die Entkopplung hier wichtig um Auswirkungen auf Consumer-Applikationen zu vermeiden.

Geschäftsprozesse, die Menschen involvieren, sind zudem meist langläufig. Weil im Normalfall nicht alle laufenden Geschäftsprozessinstanzen und die zugehörigen Tasks zu einem definierten Zeitpunkt abgearbeitet sind, ist in Migrationsszenarien ein Parallelbetrieb erforderlich. In der Regel gibt es einen Übergangszeitraum, in dem die bereits vorhandenen Instanzen und Tasks über die alte Version der Plattform abgewickelt werden und alle neu angestoßenen Instanzen bereits von der Plattform in der aktuelleren Version verarbeitet werden. Im Anschluss an diese Übergangsphase kann dann die ältere Version der Plattform abgeschaltet werden. Mit Hilfe der Integrationsfunktion des GHIA-Frameworks ist es möglich diesen Parallelbetrieb hinsichtlich der Taskbearbeitung sicherzustellen.

Mobile Endgeräte sind im Durchschnitt weniger leistungsstark als normale Arbeitsplatzrechner. Infolgedessen ist der Zusatzaufwand, der im Falle von SOAP-Webservices bei der Umwandlung der Java-Objekte zu XML-Daten und umgekehrt entsteht, meist zu hoch. Daher werden in diesem Bereich in der Regel die schlankeren REST-Webservices eingesetzt bei denen diese Umwandlung entfällt. Vor dem Hintergrund des stetigen Wachstums bei der Verbreitung mobiler Endgeräte ist es also wichtig eine REST-Schnittstelle alternativ zu bestehenden Schnittstellen anzubieten. Das GHIA-Framework bietet daher alternativ zu der auf WS-HT basierenden SOAP-Schnittstelle auch eine REST-Schnittstelle an, um die Interoperabilität und Akzeptanz des Frameworks zu steigern. In der ersten vorliegenden Version des GHIA-Frameworks eignet sich die REST-Schnittstelle jedoch noch nicht für den produktiven Einsatz, weil diese sich aktuell im Status „Experimentell“ befindet.

Zusammenfassend betrachtet kann gesagt werden, dass für das GHIA-Framework aus Praxissicht interessante Einsatz- und Anwendungsszenarien existieren. Zudem zeigt die Implementierung des Frameworks, dass es möglich ist bestimmte Aspekte aus dem Bereich der Human Interactions zu vereinheitlichen und Funktionalitäten standardbasiert zur Verfügung zu stellen. Derzeit

existiert keine alternative Referenzimplementierung die der Funktionalität des GHIA-Frameworks gleichkommt.

Interessante Fragestellungen, die von der aktuellen Implementierung des GHIA-Frameworks derzeit nicht adressiert werden sind:

- Wie kann die transparente und einheitliche Authentifizierung und Autorisierung von menschlichen Akteuren erfolgen?
- Existiert die Möglichkeit Engine-spezifische Exceptions, gegebenenfalls konfigurierbar, in Standard-Exceptions zu übersetzen?
- Können standardisierte, von einer bestimmten Client-Technologie unabhängige und wiederverwendbare Komponenten definiert werden, um diese für die Darstellung von Taskdetaildaten zu verwenden?

Von den genannten offenen Punkten ist der Security-Aspekte für den Einsatz im Tagesgeschäft von zentraler Bedeutung. Bei der Security-Thematik handelt es sich um eine Querschnittsfunktionalität. Aus diesem Grund ist es wichtig die Überprüfung von Authentizität und Autorisierungen transparent an zentraler Stelle durchzuführen. Dies ist derzeit nicht möglich.

Das Exception-Handling ist derzeit nur rudimentär implementiert und muss dahingehend verfeinert werden, die spezifischen Exceptions in die von der der WS-HT Spezifikation geforderten zu übersetzen. Das bedeutet, dass auf Seite der Client-Applikationen derzeit noch kein Exception-Handling nach WS-HT möglich ist. Die Grundfunktionalität des Frameworks beeinträchtigt dies aber nicht wesentlich.

Beim letzten offenen Punkt bezüglich der Definition wiederverwendbarer Client-Komponenten handelt es sich um ein „Nice-to-have“-Feature, das die Entwicklung der Client-Applikationen für die Taskbearbeitung erleichtern soll. Für die Entwicklung der Client-Applikationen bietet das GHIA-Framework derzeit keine Unterstützung, da der Fokus auf der Konsolidierung der Server-Applikationen liegt. Client-Entwickler müssen sich derzeit selbst darum kümmern, wie beispielsweise die Taskdetaildaten unter Verwendung einer bestimmten Technologie dargestellt werden können. Weil dies aber in der Regel immer in der Verantwortung der Client-Entwickler liegt, stellt das keine Einschränkung für den praktischen Einsatz des Frameworks dar.

8.3 Ausblick

Auch wenn, wie der vorherige Abschnitt zeigt, im Nachhinein noch offene Fragen bezüglich fehlender oder nicht berücksichtigter Funktionalitäten existieren, konnten doch die anfangs definierten Ziele der vorliegenden Abschlussarbeit vollständig erreicht werden. Diese offenen Punkte zeigen auch, dass das GHIA-Framework Potential hat und gute Weiterentwicklungsmöglichkeiten bietet. Es ist zudem davon auszugehen, dass in Praxissituationen neue Anforderungen entstehen, die zu Erweiterungen des Frameworks führen.

Ein direkter Ansatzpunkt für eine Erweiterung ist die im vorherigen Abschnitt angesprochene Security-Problematik. Das GHIA-Framework behandelt den Security Aspekt derzeit nur am Rande, weil sich die Task-Engines selbst um die Authentifizierung und Autorisierung von Benutzern kümmern (vergleiche hierzu Abschnitt 5.4). Dies stellt solange kein Problem dar, bis die Integration mit einer Client-Applikation implementiert werden muss. An dieser müssen sich Benutzer in der Regel anmelden. Für die Bearbeitung von Tasks erfolgt wiederrum eine Authentifizierung gegenüber der Task-Engine. Das Problem hierbei ist, dass die Authentifizierung im Grunde doppelt durchgeführt werden muss. Abhilfe würde hier eine Art „Single-Sign-On“-Erweiterung für das GHIA-Framework schaffen, die transparent von Client- und Serverkomponenten benutzt werden kann. Mögliche Ansätze für die Umsetzung einer solchen Erweiterung sind OAuth[63] oder auch die Apache CXF Erweiterung STS[64] (Security Token Store).

Wie bereits in der Zusammenfassung in Abschnitt 8.1 angemerkt wurde, ist die Erweiterung des Frameworks von Implementierungsseite einfach und schnell möglich. Allerdings sollte im Vorfeld eine Analyse, ähnlich zu der in Kapitel 4, durchgeführt werden, um die Besonderheiten einer spezifischen Task-Engine zu identifizieren. Ähnlich zu den im Rahmen dieser Arbeit betrachteten Engines, wird auch die Kompatibilität anderer Task-Engines zur WS-HT Spezifikation eher gering ausfallen. Recherchen im SAP[65]- oder IBM[66]-Umfeld zeigen, dass hier bezüglich der Human Interactions proprietäre Schnittstellen zum Einsatz

[63] Details hierzu unter [OAUTH]
[64] Details hierzu unter [STS11]
[65] Nähere Informationen unter [SAPHT]
[66] Nähere Informationen unter [IBMHT]

kommen. Eine fundierte Aussage bezüglich des Abdeckungsgrad der von WS-HT geforderten Funktionalitäten kann allerdings mit dem derzeitigen Wissenstand nicht getroffen werden.

Wie eingangs bereits erwähnt, ist das Potential bezüglich möglicher Erweiterungen des GHIA-Frameworks vorhanden. In einem nächsten Schritt wäre es wichtig die skizzierte Security-Erweiterung des Frameworks durchzuführen. Im Anschluss daran ist ein erster produktiver Einsatz in einem kleineren Umfeld aus technischer Sicht problemlos möglich. Zudem ist es denkbar, das GHIA-Framework als Opensource-Projekt zu veröffentlichen und so einer breiteren Community zugänglich zu machen. Produktive Einsätze des Frameworks in der Praxis auf der einen Seite und eine breit aufgestellte Community auf der anderen Seite dienen der Stabilisierung sowie der Steigerung der Interoperabilität des Frameworks und fördern dessen Verbreitung. Mit der Hilfe einer breiten Community könnte zudem die Weiterentwicklung des Frameworks forciert und weiter vorangetrieben werden.

Anhang A: Glossar

Build *Synonyme*: Build-Lauf, Build-Prozess

Der Build beziehungsweise der Build-Prozess bezeichnet das Erzeugen der Deployment-Artefakte. Im Verlauf des Build-Prozesses erfolgt die Auflösung der Abhängigkeiten, die Kompilierung der Source-Dateien, die Ausführung vorhandener Test sowie die Erzeugung der Deployment-Artefakte.

Classpath *Synonyme*: -

Der in der Regel als Parameter gesetzte Klassenpfad zeigt der Java Virtual Machine zur Laufzeit an, wo nach benutzerdefinierten Libraries und Klassen zu suchen ist.

Dependency *Synonyme*: Abhängigkeit, Library

Ist eine Abhängigkeit zu Klassen eines anderen JAR-Files.

Dependency Injection *Synonyme*: -

Entwurfsmuster, welches dem IOC-Prinzip („Inversion of Control“) folgt. Hierbei sind die Objekte nicht mehr selbst für die Auflösung der benötigten Abhängigkeiten zuständig, sondern dies wird von einem externen Framework übernommen. (Beispiele: Spring Framework, Google Guice)

Deployment *Synonyme*: Bereitstellung

Bezeichnet die zentrale Bereitstellung einer Applikation in einem Laufzeit-Container (Application Server).

Human Interaction *Synonyme*: Menschliche Interaktion

Bezeichnet die Interaktion eines automatisierten Geschäftsprozesses mit einem menschlichen Akteur

Paging *Synonyme*: -

Ist eine Strategie für das Abfragen großer Datenmengen. Die angefragten Daten werden paketweise in einer festzulegenden Größe („Page-Size“) geladen.

Process-Engine *Synonyme*: BPM-Engine

Stellt eine Laufzeitumgebung für automatisierte Geschäftsprozesse dar.

Task *Synonyme*: Aufgabe, Human Task

Definition einer Aufgabe die, im Kontext menschlicher Interaktion in automati-

siert ablaufender Geschäftsprozesse, von einem Menschen erledigt werden muss.

Task-Engine *Synonyme*: Human Workflow-Engine, Task-Processor

Bezeichnet eine Laufzeitumgebung, die für die Verarbeitung und Management von Human Tasks zuständig ist.

Tasklist-Applikation *Synonyme*: Worklist-Applikation, Inbox-Applikation, Postkorb-Applikation

Stellt eine Applikation dar, die authentifizierten Benutzern die Bearbeitung von Human Tasks ermöglicht.

Anhang B: Inhalt der mitgelieferten CD

Auf der CD befinden sich neben einer elektronischen Fassung dieser Ausarbeitung, welche im Rahmen der Ausarbeitung verwendeten elektronischen Dokumente sowie die Quellcode-Dateien des GHIA-Frameworks.

Zur Strukturierung: Im Ordner „*src*“ findet sich der Quellcode des GHIA-Frameworks. Im Ordner „*docs*“ finden sich die, im Literaturverzeichnis referenzierten, elektronischen Dokumente.

Die im Javamagazin erschienene Artikelserie „SOA aus dem wahren Leben“, wurde der Vollständigkeit halber komplett in das „*docs*“ übernommen. Diese besteht aus den folgenden PDF's:

- javamagazin_2008_11_winterberg_rent_your_legacy_car.pdf
- javamagazin_2008_12_winterberg_soa_blueprint.pdf
- javamagazin_2009_01_winterberg_von_klein_nach_gross.pdf
- javamagazin_2009_02_winterberg_service_kategorisierung.pdf
- javamagazin_2009_03_winterberg_lose_kopplung.pdf
- javamagazin_2009_04_winterberg_was_macht_einen_ guten_service_aus.pdf
- javamagazin_2009_05_winterberg_governance.pdf
- javamagazin_2009_06_winterberg_security.pdf
- javamagazin_2009_07_winterberg_compensation.pdf
- javamagazin_2009_08_winterberg_soa_und_benutzeroberflaechen.pdf
- javamagazin_2009_09_winterberg_kanonisches_datenmodell.pdf
- javamagazin_2009_10_winterberg_soa_und_change_management.pdf
- javamagazin_2009_11_winterberg_data_access_services.pdf
- javamagazin_2009_12_winterberg_event-driven_soa.pdf

Anhang C: Eidesstaatliche Erklärung

Eidesstattliche Erklärung

Ich, Sven Bernhardt, versichere an Eides statt, dass ich die vorliegende Ausarbeitung selbstständig angefertigt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht.

Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

…… ……

(Ort, Datum) (Unterschrift)

Literaturverzeichnis

[ACTI01] *Activiti 5.8 User Guide*. (kein Datum). Abgerufen am 01.12.2011 von http://activiti.org/userguide/index.html

[ACTI02] Alfresco. (2012). *Activiti - Engine 5.9 API*. Abgerufen am 19.01.2012 von Activiti - Engine 5.9 API

[BERS11] Bernhardt, S. (23.09.2011). Integration von Menschen in automatisiert ablaufende Geschäftsprozesse – Ein Vergleich zweier Ansätze am Beispiel der Oracle BPM Suite und der Activiti BPM Platform. Köln, Nordrhein-Westfalen, Deutschland.
(CD-Referenz. docs/ Integration_von_Menschen_in_automatisiert_ablaufende_Geschäftsprozesse.pdf)

[BLOJ08] Bloch, J. (2008). *Effective Java - Second Edition.* Massachusetts, USA: Addison-Wesley.

[CXF] Apache Software Foundation. (2012). *Apache CXF: An Open-Source Services Framework*. Abgerufen am 13.03.2012 von Apache CXF: http://cxf.apache.org/

[DASM12] Das, M., Deb, M., & Wilkins, M. (2012). *Oracle Business Process Management Suite 11g Handbook.* USA: McGraw-Hill.

[EJB] Oracle. (kein Datum). *Enterprise Java Beans 3.0 Final Release.* Abgerufen am 12. 01.2012 von Oracle Technology Network (OTN): http://download.oracle.com/otn-pub/jcp/ejb-3_0-fr-eval-oth-JSpec/ejb-3_0-fr-spec-ejbcore.pdf
(CD-Referenz. docs/ejb-3_0--spec.pdf)

[ERLT091] Erl, T. (2009). *SOA Design Patterns.* USA: Prentice Hall.

[ERLT092] Erl, T. (2009). *Standardized Service Contract*. Abgerufen am 18.02.2012 von SOA Principles: http://www.soaprinciples.com/standardized_service_contract.php

[HISE] Apache Software Foundation. (14.03.2012). *Apache HISE - Open Source Implementation of WS-Human-Task Specification*. Abgerufen am 26.03.2012 von Apache HISE: http://incubator.apache.org/hise/index.html

[IBMHT] IBM. (kein Datum). *IBM Business Process Manager, Version 7.5: Authorized roles for actions on task instances*. Abgerufen am 10.04.2012 von http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=%2Fcom.ibm.wbpm.bpc.doc%2Ftopics%2Frhtmapi_taskinstanceactions.htm

[JAXWS] Oracle. (2012). *JAX-WS*. Abgerufen am 13.03.2012 von http://jax-ws.java.net

[JELL11] Jellema, L. (2011). *Oracle SOA Suite 11g Handbook.* McGraw-Hill.

[JSR311] Oracle. (2012). *JSR 311 (REST)*. Abgerufen am 13.03.2012 von http://jsr311.java.net

[MAVEN] Apache Software Foundation. (2012). *Apache Maven Project*. Abgerufen am 13.03.2010 von http://maven.apache.org

[MSOA09] Maier, B., Normann, H., Trops, B., Utschig-Utschig, C., & Winterberg, T. (08 2009). Artikelserie: SOA aus dem wahren Leben.

[OAUTH] *OAuth*. (kein Datum). Abgerufen am 02.04.2012 von http://oauth.net

[OJEE01] Oracle. (2010). *The Java EE 5 Tutorial - Webmodules*. Abgerufen am 15.03.2012 von http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html

[OSOA01] Oracle. (08 2009). Oracle SOA Suite 11g - Essential Concepts.

[OSOA02] Oracle. (kein Datum). *Oracle® Fusion Middleware Performance and Tuning Guide 11g Release 1 (11.1.1)*. Abgerufen am 14.01.2012 von http://docs.oracle.com/cd/E21764_01/core.1111/e10108/part_soa.htm

[OSOA03] Clugage, K., Shaffer, D., & Nainani, B. (kein Datum). *BPEL & Human Workflow Feature Preview Webinar Series.* Abgerufen am 01.12.2011 von Oracle Technetwork: http://www.oracle.com/technetwork/middleware/ias/bpel-workflowservices-1013-133562.pdf
(CD-Referenz. docs/BPEL_And_Human_Workflow.pdf)

[OSOA04] Oracle. (kein Datum). *Oracle® Fusion Middleware Developer's Guide for Oracle SOA Suite 11g Release 1 (11.1.1.6.0)*. Abgerufen am 07.04.2012 von http://docs.oracle.com/cd/E23943_01/dev.1111/e10224/toc.htm

[OSOA05] Oracle. (2011). *Oracle Fusion Middleware Workflow Services Java API Reference for Oracle SOA Suite 11g Release 1 (11.1.1.6.0)*. Abgerufen am 07.04.2012 von http://docs.oracle.com/cd/E23943_01/apirefs.1111/e10660/toc.htm

[PAUJ11] Paul, Javin. (02.12.2011). *What is Factory method Design Pattern in Java with Example-Tutorial.* Abgerufen am 12.04.2012 von http://javarevisited.blogspot.de/2011/12/factory-design-pattern-java-example.html

[PÜCH09] Pütter, C. (19.08.2009). *Der neue Gartner Hype Cycle 2009*. Abgerufen am 08.03. 2012 von CIO: http://www.cio.de/strategien/analysen/895084/

[RADT11] Rademakers, T., & von Liempd, R. (2011). Activiti in Action (MEAP Edition).

[REYA10] Reynolds, A., & Wright, M. (2010). *Oracle SOA Suite 11g R1 Developer's Guide.* Birmingham: Packt Publishing Ltd.

[ROZM10] Rozlog, M. (01.04.2010). *REST and SOAP: When Should I Use Each (or Both)?* Abgerufen am 05.03.2012 von InfoQ: http://www.infoq.com/articles/rest-soap-when-to-use-each

[SAPHT] SAP. (kein Datum). *Working with the BPM APIs*. Abgerufen am 10.04.2012 von http://help.sap.com/saphelp_nw73ehp1/helpdata/en/9a/eebbdf59b94c3c82af598db638b0d0/content.htm

[SOARM] OASIS. (12.10.2006). *OASIS.* Abgerufen am 08.02.2012 von OASIS SOA Reference Model TC: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf (CD-Referenz. docs/OASIS_SOA_Reference_Model.pdf)

[SPRING] Springsource. (16.02.2012). *Spring Framework Documentation - 4. The IoC container*. Abgerufen am 08. 04 2012 von http://static.springsource.org/spring/docs/3.1.1.RELEASE/spring-framework-reference/html/beans.html

[STEW10] Steck, W. (04 2010). *SOA-Frontends: Serviceorientierte Ansätze helfen bei der Konsolidierung von Client-Landschaften.* Abgerufen am 18.03.2012 von http://www.wirtschaftsinformatik.de/index.php;do=show/site=wi/sid=20189218574f8425e6455d4334421999/alloc=12/id=2689

[STS] hEigeartaigh, C. O. (19.09.2011). *Apache CXF STS documentation - part I*. Abgerufen am 02.04.2012 von Open Source Security: http://coheigea.blogspot.de/2011/10/apache-cxf-sts-documentation-part-i.html

[SWKD10] Swenson, K. D. (2010). *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done.* Tampa, Florida: Meghan-Kiffer Press.

[ULLC11] Ullenboom, C. (2011). *Java ist auch eine Insel.* Bonn: Galileo Press.

[VAADIN] *Vaadin*. (kein Datum). Abgerufen am 29.01.2012 von https://vaadin.com

[WINT01] Maier, B., & Winterberg, T. (24.02.2011). SOA und UI. Düsseldorf, Nordrhein-Westfalen, Deutschland.
(CD-Referenz. docs/SOA_and_UI_Winterberg_V10.pptx)

[WSHT] Clément, L., König, D., Mehta, V., Mueller, R., Rangaswamy, R., Rowley, M., et al. (17.08.2010). *WS-HT Spezifikation.* Abgerufen am 15.08.2011 von OASIS: http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cs-01.doc
(CD-Referenz. docs/ws-humantask-1.1-spec-cs-01.doc)