technische universität
dortmund

fakultät für informatik

PG SPorTs

Semantic Portal Technologies

Endbericht

April 2009 - April 2010

interne berichte
internal reports

Lehrstuhl 14 - Software Engineering

Betreuer:
Prof. Dr. Jakob Rehof
Dipl.-Inform. Martin Sugioarto, Dipl.-Inform. Markus Doedt

fakultät für
informatik

GERMANY · D-44221 DORTMUND

Veranstalter

Prof. Dr. Jakob Rehof
Technische Universität Dortmund
Fakultät Informatik
Lehrstuhl 14
(Geschoßbau IV)
Baroper Straße 301
44227 Dortmund

Betreuer

- Lehrstuhl 14
  Dipl.-Inform. Martin Sugioarto
  Dipl.-Inform. Markus Doedt

Teilnehmer

Oliver Becker
Timm Berens
Fateh Farshi
Eugen Fomin
Olga Galytska
Stephan Karlinski
Shihong Li
Marin Lohsträter
Ayla Tasbas
Henning Schröder
Matthias Wiedenhorst

Kontakt

`pg-sports@lists.cs.tu-dortmund.de`

Zeitraum

Sommersemester 2009 und Wintersemester 2009/2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einführung

Die Projektgruppe **S**emantic **Por**tal **T**echnologie**s** (SPorTs) startete im Sommersemester 2009 am Lehrstuhl 14 der Fakultät für Informatik an der TU Dortmund. Thema der Projektgruppe war die *„CodeGenerierung und automatisierte Integration von semantisch beschriebenen Prozessen am Beispiel von E-Government mit Hilfe von Web-Portaltechnologien“*[1].
Der vorliegende Endbericht dokumentiert die Motivation für die Projektgruppe, die Vorgehensweise bei der Bearbeitung der Problemstellung, organisatorische Entscheidungen und die im Rahmen der Projektgruppe entwickelte Software.
Die folgenden Abschnitte behandeln die Motivation der Projektgruppe, sich mit dem Thema auseinander zu setzen, die im Rahmen der Projektgruppe zu bewältigenden Aufgaben und die konkreten Ziele, also wie diese Aufgaben gelöst werden sollten. Schließlich wird auf die Organisation innerhalb der Projektgruppe eingegangen.

1.1. Motivation

Portale werden häufig genutzt, um Kunden und Mitarbeitern einen einfachen und einheitlichen Zugang zu Unternehmensprozessen zu bieten. Die Bereitstellung eines existierenden Ablaufes auf einem Portal ist jedoch häufig mit aufwendiger Handarbeit verbunden und folgt keinem einheitlichen Muster. Dies sorgt spätestens bei einer größeren Anzahl von bereitzustellenden Abläufen für Schwierigkeiten. Das Erstellen von Portalseiten unter Berücksichtigung von Abhängigkeiten einzelner Schritte eines Ablaufes sollte deshalb möglichst einfach und wenig arbeitsintensiv mit Hilfe von Softwaretools vonstatten gehen. Die Modellierung solcher Abläufe sollte von Menschen mit hohem Fachwissen geschehen. Dazu sollte technisches Wissen über das System, das diesen Ablauf steuert und zur Verfügung stellt, nicht nötig sein.

1.2. Aufgabenstellung

Aufgabe der Projektgruppe war es, ein System zu entwickeln, in dem Benutzer Abläufe aus dem Bereich E-Government modellieren bzw. bestehende (in anderer Form vorlie-

gende) Abläufe in ein Format bringen können, aus welchem möglichst automatisch eine Portalanwendung generiert wird, die diese Abläufe als Dienste zur Verfügung stellt. Dabei soll eine fachliche Sicht ermöglicht werden, d.h. den Benutzer soll kein Wissen über die technischen Aspekte eines Portalservers und die Art, wie dort Dienste zur Verfügung gestellt werden können, abverlangt werden. Die Sprache, in der Abläufe modelliert werden, muss mächtig genug sein, um die Semantik eines Ablaufs (Welche Informationen werden benötigt und müssen in welcher Reihenfolge von wem bereitgestellt werden?) nach der Modellierung und der Generierung der entsprechenden Komponenten des Portals in der vom Benutzer beabsichtigten Form zu verwirklichen. Insbesondere muss auf die verschiedenen Rollen innerhalb eines Ablaufs und die Konsequenzen von bestimmten Eingaben im weiteren Verlauf eingegangen werden. Die Darstellung auf dem Portal muss den Anforderungen des E-Government genügen. Dazu zählen etwa die Bedürfnisse der verschiedenen an einem Ablauf beteiligten Rollen. Für die Bürgerseite bedeutet dies unter anderem, dass Daten, welche schon einmal abgefragt wurden, nicht noch einmal abgefragt werden. Für Beamte ist es andererseits vorteilhaft, wiederkehrende, gleiche Arbeitsabläufe auch gleichmäßig darzustellen, um den Arbeitsfluss nicht zu unterbrechen.

1.3. Ziel des Projektes

Im Folgenden wird auf die konkrete, von der Projektgruppe anvisierte Lösung der zuvor vorgestellten Aufgabe eingegangen. Hierbei soll ein grober Überblick genügen und für Details der einzelnen Komponenten auf die späteren Kapitel verwiesen werden. Im ersten Semester wurde versucht, durch WS-BPEL (Business Process Execution Language for Web Services) beschriebene Prozesse gesteuert durch das Portal auf einer BPEL-Engine auszuführen. WS-BPEL ist eine Sprache zur Orchestrierung von Web Services. Jede Aufgabe wird in WS-BPEL als Web Service modelliert und auch aufgerufen. Nach außen wird auch der Prozess selbst wie ein Web Service modelliert. Dazu sollten zuerst die einzelnen Schritte in solch einem Prozess, repräsentiert durch Web Services, in einem Editor mit den zur Generierung des Portals nötigen semantischen Beschreibungen annotiert und zu einer Prozessbeschreibung zusammengefasst werden. Aus dieser Ablauf-Beschreibung sollte BPEL-Code generiert werden. Leider musste die Projektgruppe nach Ablauf des ersten Semesters feststellen, dass die Aufgabe nicht in der gewünschten Zeit gelöst werden würde, ohne gravierende Änderungen am bisher verfolgten Ziel vorzunehmen. Es wurde deshalb darauf verzichtet, ausführbare Geschäftsprozesse zu generieren und das Augenmerk auf die Generierung von Portalseiten gelegt. Ausgangspunkt für die

Lösung der Aufgabe waren nun Abläufe, welche durch das Ausfüllen und Verschicken von Meldungen oder Anträgen in Form von Formularen und Entscheidungen über diese beschrieben werden können. Dabei werden elektronische Formulare in Form von XForms vorausgesetzt. Mit einem grafischen Editor werden diese dem jeweiligen Schritt zugeordnet. Einzelne Schritte sind wiederum einer Rolle zugeschrieben. Die so entstandenen Ablaufdateien werden daraufhin von einem Codegenerator gelesen und die Dateien für die Darstellung auf dem Portal werden erzeugt. Diese können dann auf dem Portalserver zur Verfügung gestellt werden.
Durch den Fokus auf gesamte Formulare, anstatt auf einzelnen Eingabefelder, kann die Abfolge der Verarbeitungsschritte von bestehenden Abläufen originalgetreuer in digitaler Form abgebildet werden. Es werden nicht einzelne Daten (z. B. Name, Steuernummer, Name des Ehegatten) als Eingaben für einen Ablauf gesehen, sondern alle zusammen als eine Einheit in Form von digitalen Formularen. Die Erwartung ist, dass dies für eine Person mit Kenntnissen der Abläufe weniger Umgewöhnung bedeutet und somit die Akzeptanz von Veränderungen erhöht wird.

1.4. Organisation

Neben der Auseinandersetzung mit Technologien zur Erreichung des Projektziels und der Übung in Programmierung zählt zum Lernziel von Projektgruppen auch die selbständige Organisation der Gruppe. In diesem Abschnitt wird kurz auf die Art und Weise, wie dies in der PG SPorTs geschehen ist, eingegangen.

1.4.1. Gruppenarbeit

Die Regelung der offiziellen Treffen der Projektgruppe wurde in der ersten Sitzung festgelegt. Es wurde beschlossen sich einmal pro Woche zu treffen. Es wurden Zeit und Tag ausgesucht, die sowohl für jedes Mitglied der Projektgruppe als auch für unsere Betreuer passend sind. In jedem Treffen wurden die Aufgaben verteilt, die wichtigsten Fragen besprochen und die Ergebnisse der Arbeit den anderen Mitgliedern und den Betreuern vorgestellt. Die restliche Zeit diente der selbstständigen Arbeit, was auch Treffen in kleinen Teilgruppen beinhaltete. Für die schnelle und einfache Kommunikation wurde eine Mailingliste eingerichtet, welche von Betreuern und Teilnehmern gleichermaßen genutzt werden konnte.
Es wurde beschlossen, dass bei jedem Treffen ein Protokoll geführt wird. Die Vorlage für das Protokoll wurde nach der ersten Sitzung erstellt und durch das Wiki für alle

Protokollanten bereitgestellt. In erster Linie diente das Protokoll dazu, die besprochenen Punkte und auch gewonnenen Informationen (beispielsweise durch Gastvorträge) schriftlich festzuhalten. Somit sollten auch in der Sitzung nicht anwesende Teilnehmer in die Lage versetzt werden, sich nachträglich über den Inhalt der Sitzung zu informieren. Zusätzlich wurden in den Protokollen die getroffenen projektrelevanten Entscheidungen festgehalten, so dass jedes Mitglied im Idealfall immer eine grobe Übersicht über alle Teile des Projektes behalten konnte. Die Protokolle wurden möglichst zeitnah nach jeder Sitzung in Reinform geschrieben und im Wiki abgelegt, um diese jedem Mitglied und Betreuer schnellstmöglich zugänglich zu machen. Aus Gründen der Fairness wurde zudem beschlossen, dass der Protokollant in regelmäßigen Abständen ausgewechselt wird. Festgelegt wurde hierbei ein zweiwöchiger Rhythmus.
Außerdem wurde jede Sitzung von einem Moderator geführt. Die Aufmerksamkeit wurde dann zunächst auf die erzielten Ergebnisse der letzten Woche gerichtet. Das Team und die Betreuer hatten somit einen konkreten Anfangspunkt, auf dem aufgebaut werden konnte. Zu weiteren Aufgaben des Moderators gehörte, die Diskussion anzuregen, darauf zu achten, dass der zeitliche Rahmen eingehalten wird und die Diskussion in einem geordneten Rahmen verlaufen zu lassen. Auch für die Moderatoren galt eine Rotationslösung. Jeder Teilnehmer der Projektgruppe führte zunächst jeweils in zwei aufeinanderfolgenden Sitzungen Protokoll und moderierte dann die zwei darauf folgenden Sitzungen.
In der ersten Sitzung wurde ein Projektleiter bestimmt. Dieser sollte einen Überblick über das gesamte Projekt haben und möglichst zu jedem Zeitpunkt konkret wissen, von wem welche Aufgabe bearbeitet wird oder bearbeitet werden soll. Zu den Aufgaben des Projektleiters gehörte auch die grobe Zeitplanung des Projektes. Dazu gehörten die Fertigstellungszeiträume der zugeteilten Aufgaben und die Abgabetermine von Dokumenten wie z.B. Protokolle. Zudem wurden Leute ausgewählt, die sich um die Infrastruktur der Projektgruppe kümmern. Das Infrastruktur-Team kümmerte sich darum, dass eine möglichst zusammenhängende und komfortable Lösung gefunden wurde.

1.4.2. Infrastruktur

Zur Unterstützung der Arbeit, gerade im Hinblick auf die Kommunikation der PG Mitglieder untereinander und auf die Verwaltung des Projekts, hat sich die PG entschieden, verschiedene Softwareprodukte zu benutzen. Noch während der Seminarphase wurden drei Mitglieder bestimmt, um Vorschläge für mögliche Systeme zu erarbeiten und diese dann bei Bedarf umzusetzen.

Nach einer ca. zweiwöchigen Evaluierungszeit entschieden sich die Mitglieder für eine direktere Kommunikation parallel zur schon existierenden Maillingliste. Hierfür wurde ein XMPP-Server von den Mitarbeitern des Lehrstuhls 14 eingerichtet und zur Verfügung gestellt. XMPP ist ein Protokoll zur Echtzeitkommunikation, auch Instant Messaging genannt [2].

Zur Installation der anderen Komponenten wurde der PG die Möglichkeit eröffnet, virtuelle Maschinen auf dem lehrstuhleigenen „ESX Server“ zu installieren. Der „ESX Server“ ist Teil der Programmsuite „VMWare Infrastructure“ der US amerikanischen Firma „VMware, Inc.“, welche die Virtualisierung von Computersystemen realisiert.

Es wurden zwei Server für die PG eingerichtet. Die Hostnamen der Systeme waren „lizzy.cs.tu-dortmund.de“ und „ralph.cs.tu-dortmund.de“. Beide Systeme benutzten als Betriebssystem CentOS 5.3. Die Wahl fiel auf diese Distribution, da sie innerhalb der „Infrastruktur Gruppe“ die bekannteste war.

CentOS ist ein Akronym für „Community Enterprise Operating System“ und baut auf der Distribution RHEL („Red Hat Enterprise Linux“) der Firma „Red Hat“ auf. CentOS ist vollständig binärkompatibel zu RHEL. Somit profitierte die Projektgruppe direkt vom Support im Bereich Updates von „Red Hat“, ebenso wie den zahlreichen Hilfen und Anleitungen im Internet.

Zum Datenaustausch wurde ein Wiki installiert. Die Wahl fiel hierbei auf das Programm „DekiWiki“, in der Version 9.02.1. Primär wurde das Wiki zum Austausch von Dateien genutzt, wie der wöchentlichen Sitzungsprotokolle und zur Koordination der Gruppe. Beispielweise geschah die Organisation des Zwischenberichts über einen Artikel (inklusive mehrerer Unterartikel).

Die Organisation des geschriebenen Quelltextes geschah über ein SVN Repository. Dieses bot die größtmögliche Flexibilität im Hinblick auf Synchronisation zwischen den einzelnen PG Mitgliedern und stellte zusätzlich indirekt eine Backupfunktionialität dar. Allerdings wurde innerhalb des Systems auf jegliche Rechteverwaltung verzichtet. Durch einen Passwortschutz ist das Repository nur für die Mitglieder der PG zugänglich, aber innerhalb des Systems hat der Benutzer volle Lese- und Schreibrechte auf alle Projekte.

Zur Planung des Projekts, zum Definieren von Milestones und als Bugtracker wurde das Programm „trac“ eingesetzt. Das Programm bietet einen umfassenden Katalog von Funktionen für die Planung und Durchführung eines Projektes, wie es in dieser PG angegangen wurde.

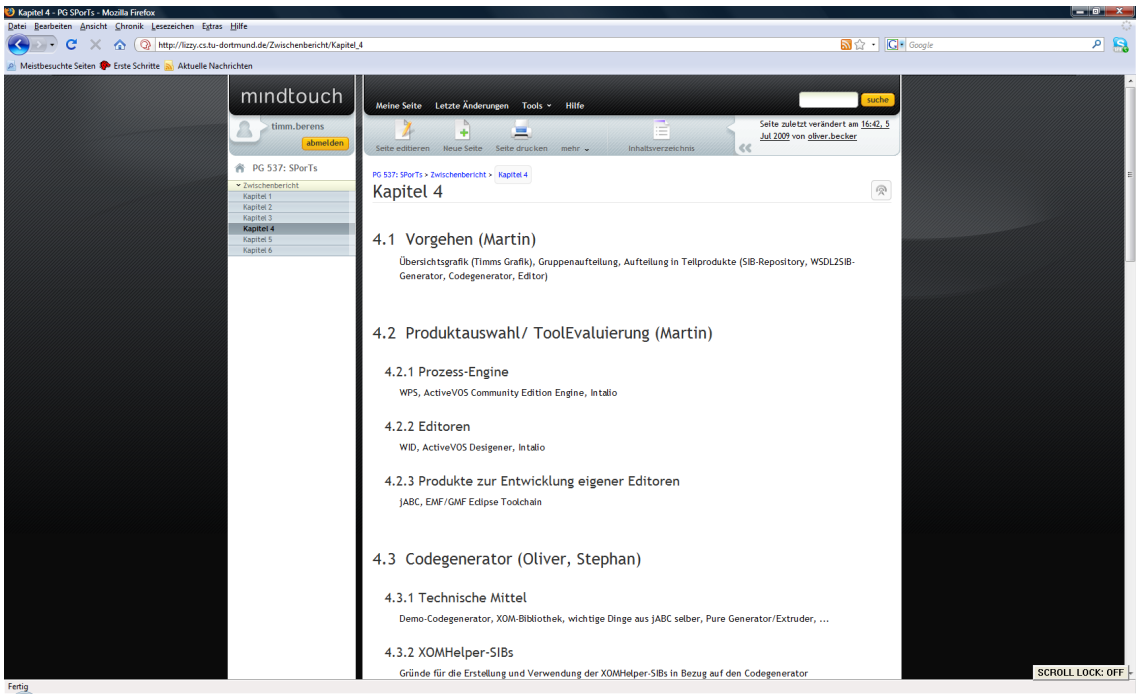


Abbildung 1.1.: Dekiwiki 9.02.1

1.5. E-Government im Portalumfeld

Die Definition des E-Governments ist in der wissenschaftlichen Diskussion noch nicht abgeschlossen. Auffällig ist jedoch in nahezu allen Ansätzen der inhaltliche Bezug auf technologische Aspekte, der auch im folgenden auftaucht:

„E-Government umfasst alle Prozesse der öffentlichen Willensbildung, der Entscheidungsfindung und Leistungserstellung in Politik, Staat und Verwaltung, soweit diese unter weitestgehender Nutzung von Informations- und Kommunikationstechnologien stattfinden. Dabei sind die Nutzungsmöglichkeiten sehr vielfältig. Sie fangen an bei der Verwaltungsmodernisierung durch elektronische Vorgangsbearbeitung, gehen über die Bereitstellung von Verwaltungsinformationen auf Behördenportalen im Internet, bis hin zu interaktiven elektronischen Bürgerdiensten“[4, E-Government].

„Unter E-Government versteht man die Abwicklung geschäftlicher Prozesse im Zusammenhang mit Regieren und Verwalten mit Hilfe von Informations- und Kommunikationstechniken über elektronische Medien. Aufgrund der technischen Entwicklung nehmen wir an, dass diese Prozesse künftig sogar vollständig elektronisch durchgeführt werden

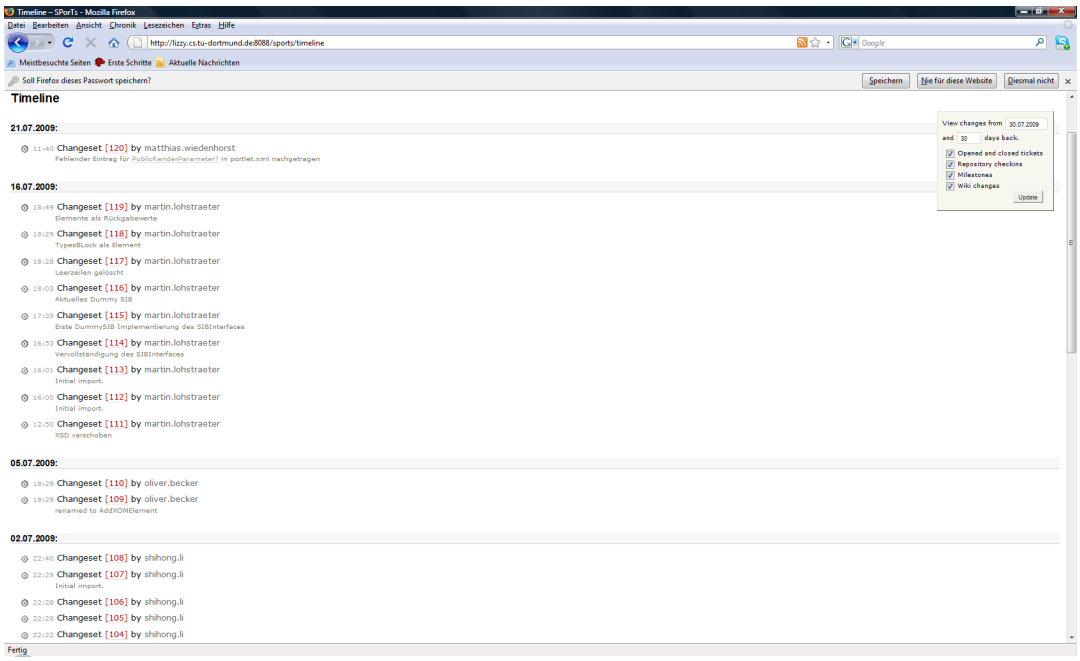


Abbildung 1.2.: Timelineansicht von trac

können. Diese Definition umfasst sowohl die lokale oder kommunale Ebene, die regionale oder Landesebene, die nationale oder Bundesebene sowie die supranationale und globale Ebene. Eingeschlossen ist somit der gesamte öffentliche Sektor, bestehend aus Legislative, Exekutive und Jurisdiktion sowie öffentlichen Unternehmen“[5, S. 1].

Laut Studien aus vielen Ländern verbessert der Einsatz von E-Government die Effizienz, Qualität und Produktivität bei der Verwaltung und öffentlichen Dienstleistung. Außerdem kann der Staat dadurch viel Zeit und Kosten sparen. Dies ermöglicht auch ständige und unmittelbare Verfügbarkeit der Dienstleistungen. Dadurch werden mehr Kapazitäten für die persönliche Beratung in Sonderfällen erreicht. Noch zu beachten ist, dass man mit Hilfe von E-Government die Transparenz der Funktion öffentlicher Institutionen erhöht und Korruption und Betrug leichter bekämpft werden kann.
„Gleichzeitig kommt E-Government eine weitere Rolle zu. Der Staat wird zum Nachfrager und Förderer von Zukunftstechnologien und sichert die erforderlichen Infrastrukturen der Informationstechnik. Durch den Einsatz innovativer und neuer Techniken wird er zum Impulsgeber und zum Partner der Wirtschaft“[3, S. 4].

1.5.1. E-Government im Rahmen der Projektarbeit

Es wurde aus dem Themengebiet E-Government die gesamte Entwicklungsbahn anhand zweier Szenarien entworfen und implementiert. Dafür haben wir zwei Ablauf-Dateien erstellt nämlich, *Beschwerde einreichen* und *Hund anmelden*. Die ersten Überlegungen sind in ganz simplen Beispielszenarien dargestellt (Siehe Anhang D.1 und D.2), die später als Prozesse modelliert wurden. Die näheren Erläuter-ungen bzgl. Prozessmodellierung sind in den weiteren Kapiteln zu finden. Anhand dieser Prozesse wurden dann Formulare erstellt. Zur Generierung von Portalseiten reichen die Informationen, welche die Geschäftsprozesse bereitstellen nicht aus. Zusätzliche Informationen wurden für Rollenzuweisung und Abhängigkeiten von Daten gebraucht. Benutzer von E-Government Portalen sind einer von mindestens zwei Rollen zugeordnet. Bürger auf der einen Seite und Beamte bzw. Sachbearbeiter auf der anderen Seite. Beide Rollen können natürlich noch weiter aufgeteilt werden. Eine Unterscheidung in Privat- und Geschäftskunden oder in verschiedenen Dienstgrade wäre denkbar.

1.5.2. Wie soll E-Government funktionieren?

Öffentliche Dienste müssen nur in einer sicheren Umgebung betrieben werden. Dies müssen auch garantieren können, dass die jeglichen digitalen Transaktionen und private Daten jederzeit von außen geschützt sind. Ein sicheres und zuverlässiges System für eine funktionsfähige Informationsgesellschaft ist eine notwendige Voraussetzung. Die Bürger müssen jederzeit auf ihre persönlichen Daten zugreifen und mögliche Änderungen vornehemen können.
„Die Einführung elektronischer Behördendienste ist keineswegs einfach. Um Dienstleistungen zu erbringen, in deren Mittelpunkt der Bürger steht und um unnötigen Verwaltungsaufwand abbauen zu können, müssen Informationen über Abteilungsgrenzen und Verwaltungsebenen hinweg gemeinsam genutzt werden (z.B. zwischen örtlicher und nationaler Ebene). In den meisten Fällen sind dafür organisatorische Veränderungen notwendig. Dies erfordert den Willen, bisherige Arbeitsmethoden in Frage zu stellen und stößt häufig auf Widerstände. Außerdem sind elektronische Behördendienste nicht umsonst zu haben und machen sich oft erst langfristig bezahlt“[6, S. 4].
Die Geschäftsprozesse (Personalmeldungen, Statistikpflichten, Anträge, Genehmigungen, Kontakte zu Finanzämtern) stellen in der Regel die üblichen Abläufe dar, die bei einer Verwaltung in einem Unternehmen nötig sind. „Ziel ist, für ausgewählte Verwaltungsverfahren sowohl die Geschäftsabläufe wie auch die IT-Verfahren der Beteiligten zu

integrieren. Grundlage sind optimierte Geschäftsabläufe in den Unternehmen und Behörden sowie entsprechende Standards für Schnittstellen und Austauschformate zwischen den IT-Verfahren“[3, S. 12].

2 Fachliche Abläufe

In diesem Kapitel geht es hauptsächlich um die Erstellung der fachlichen Abläufe. Zunächst werden die verschiedenen Abläufe mit den zugehörigen Modellen aufgezeigt. Danach folgt die Erklärung zu der Erstellung der Formulare.

2.1. Ausarbeitung der Vorgänge

Ein Teil der Projektgruppe beschäftigte sich hauptsächlich mit der Suche nach geeigneten speziellen Abläufen aus dem Bereich E-Government, welche modelliert werden sollten. Die Suche verlief erfolgreich und es standen verschiedene Abläufe zu Auswahl. Überwiegend hat sich die Projektgruppe an Rathäusern, bzw. am virtuellen Rathaus von Dortmund orientiert und sich somit passende Abläufe ausgesucht. Unter anderem waren es Abläufe wie Briefwahl beantragen, Beschwerde, Umzug, Hund anmelden, Kind anmelden und Formulare beantragen. Für die Erfassung und Analyse solcher oft komplexen Verwaltungsabläufen war es sinnvoll, diese zu visualisieren. Deshalb hat die Gruppe einfache Modelle für den Vorgang der genannten Abläufe per Hand erstellt. Im Anhang ist in Abbildung D.3 ein solches Modell dargestellt.

2.1.1. Briefwahl

In der erwähnten Abbildung handelt es sich um die Beantragung eines Formulars für eine Briefwahl. Da die Abläufe und deren Vorgang auf Formularen basieren, ist dies auch im Modell dargestellt. Der Sachbearbeiter erstellt und sendet den Bürgern die Wahlbenachrichtigung. Der Bürger erhält die Wahlbenachrichtigung und möchte eine Briefwahl beantragen. Er füllt die Wahlbenachrichtigung aus und beantragt somit eine Briefwahl. Nachdem Ausfüllen der Formulare wird diese durch den Bürger an den Sachbearbeiter geschickt. Der Sachbearbeiter füllt einen internen Antrag aus und löscht den Bürger aus der Wählerliste. Der interne Antrag ist nicht definiert. Der Sachbearbeiter schickt dem Bürger den Wahlbogen. Der Bürger erhält den Wahlbogen, füllt dieses Formular aus und schickt es dann wieder dem Sachbearbeiter. Der Sachbearbeiter füllt erneut einen internen Antrag aus und legt den Wahlbogen ab. Fr̈ alle genannten Abläufe wurde ein solches oder ähnliches einfaches Modell entwickelt.

2.1.2. Vorgang Bechwerde

Briefwahl war einer der Abläufe die zu Auswahl standen. Doch weitere Beachtung schenkten wir zwei ganz anderen Abläufen, und zwar den Abläufen Beschwerde und Hund anmelden. In Abbildung D.2, welches sich im Anhang befindet, wird der Ablauf in einer anderen Modellvariation dargestellt als in Abbildung D.3. Hierbei handelt es sich um den Vorgang Beschwerde. Der Bürger füllt ein Formular aus und sendet dies dem Sachbearbeiter. Der Bürger kann in diesem Formular seine Identität angeben, aber auch anonym bleiben. Weiterhin gibt der Bürger den Grund und die Tatadresse für seine Beschwerde an. Nachdem der Sachbearbeiter das Formular überprüft und ein internes Protokoll ausgefüllt hat, benachrichtigt er einen Außendienstmitarbeiter für die Aufklärung vor Ort.

2.1.3. Vorgang Hund anmelden

Der Ablauf Hund anmelden ist natürlich etwas komplexer als der Ablauf der Beschwerde. Hierfür werden einige Schritte und Formulare mehr benötigt. Der Bürger muss ein Formular ausfüllen, um einen oder auch mehrere Hunde anzumelden. Die ausgefüllten Formulare schickt er dem Sachbearbeiter zu. Der Sachbearbeiter überprüft die Formulare auf Vollständigkeit und Korrektheit. Sobald die Formulare vollständig sind, werden sie zur Verarbeitung weitergereicht. Nach der Bearbeitung der Unterlagen findet ein externer Ablauf statt. Eine Hundemarke soll erstellt werden. Weiterhin muss der Sachbearbeiter dem Bürger die Steuerunterlagen zuschicken. Falls die Unterlagen des Bürgers nicht vollständig sind, werden diese dem Bürger zur Korrektur zurückgeschickt. Der Bürger überprüft seine Angaben, korrigiert diese und sendet das Formular erneut dem Sachbearbeiter. Wieder wird die Vollständigkeit geprüft und je nachdem werden die Daten entweder bearbeitet oder an den Bürger zurückgesendet. In Abbildung D.1 im Anhang ist das Modell für den Ablauf Hund anmelden dargestellt.

2.2. Erstellung der fachlichen Formulare

Nachdem die Teilgruppe die Modelle für die verschiedenen Abläufe erstellt hatte, mussten sie, da die Abläufe auf Formularen basieren, Formulare erstellen. Nach einer gründlichen Recherche wurden einfache Formulare per Hand erstellt, welche als Fundament für spätere Arbeiten dienen sollten. Von einigen Alternativen wurden zwei ausgewählt.

2.2.1. Beschwerde

In Abbildung C.1 im Anhang ist ein Formular für den Beschwerdefall dargestellt. Die Felder, die unbedingt ausgefüllt werden müssen, sind mit einem Sternchen definiert. Wie vorhin erwähnt, kann der Bürger seine Beschwerde auch anonym absenden. Es ist wichtig die Adresse vollständig anzugeben, damit der Außendienstmitarbeiter weiß, wo sich der Tatort befindet. Weiterhin ist auch noch ein freies Textfeld verfügbar. Hier kann der Bürger seine konkrete Beschwerde erläutern. Als letztes muss der Bürger nur noch Ort und Datum angeben und kann somit seine Beschwerde absenden.

2.2.2. Hund anmelden

Das Formular für den Ablauf Hund anmelden ist in Abbildung C.2 im Anhang zu sehen. In diesem Formular muss der Hundehalter zunächst Angaben zu seiner eigenen Person machen. Angaben wie Name, Vorname, Anschrift, Geburtsdatum, Geburtsort und Telefon müssen ausgefüllt werden. Danach gibt der Bürger die Daten seines Hundes an. Wurftag/Alter, Rasse, Größe/Gewicht und Anschaffungstag sind wichtige Daten des Hundes, die der Bürger angeben muss. Neben Wurftag steht auch noch das Alter zur Verfügung, falls der Hundehalter den Wurftag des Hundes nicht mehr in Erinnerung hat. Weiterhin ist es wichtig anzugeben, wie viele Hunde sich im Haushalt bereits befinden. Dies hat nämlich eine Auswirkung auf die Hundesteuer. Als letztes versichert der Bürger noch die Richtigkeit der Daten, gibt Ort und Datum an und verschickt das Formular an den Sachbearbeiter.

3 Prozessmodellierung

Ein Ziel des Projekts besteht darin, beliebige Vorgänge in verschiedenen Lebenslagen von Bürgern innerhalb der Domäne des E-Governments auf ein Webportal abbilden zu können. Diese Vorgänge müssen dazu auf fachlicher Ebene modelliert werden können, ohne umfangreiche technologische Kompetenzen vorauszusetzen, die zur Umsetzung eines E-Government-Portals erforderlich sind. Um eine solche Trennung der Kompetenzen zu erreichen, werden mehrere Modellschichten mit angepassten Schnittstellen benötigt, die benutzerfreundliche Übergänge zwischen den einzelnen Ebenen erlauben.
In diesem Kapitel werden wir uns zuerst der fachlichen Modellschicht widmen und unser Vorgehen bei der Suche nach geeigneten Spezifikationen und Werkzeugen beschreiben. Anschließend stellen wir unsere Auswahl im Detail vor und umreißen die notwendigen Schritte unserer Modellierung. Auf die Modellschicht, die sich zwischen der fachlichen Darstellung von Geschäftsprozessen und unserer Portal-Ebene befindet, gehen wir im letzten Abschnitt ausführlich ein.

3.1. Evaluierung

Das Konzept eines formularbasierten Ablaufs bedarf eines geeigneten Editors zur grafischen Modellierung der verwendeten Formulare und deren Bearbeitungsreihenfolge. Unter der Prämisse, dass die Verwendung des Editors kostenlos ist, stellen die Möglichkeit der Modellierung von Kontrollflüssen und die Möglichkeit der Annotation der Formulare wesentliche Anforderungen dar. Diese Annotationen sollten möglichst strukturiert vorgenommen werden können, d.h. dass der Editor für bestimmte Informationen vordefinierte Eingabeoberflächen bereitstellt, wie z. B. die zu einem Prozessschritt zugehörige ausführende Person. Diese Struktur ist jedoch immer von der verwendeten grafischen Notation abhängig. Diese Anforderungen an den Editor regten die Diskussion an, ob man nun mit Hilfe geeigneter Werkzeuge wie *Eclipse GMF*[1] einen auf die Anforderungen zugeschnittenen Editor entwirft oder bereits existierende Editoren bezüglich dieser Anforderungen untersucht. Diese stellen eventuell weitere Funktionen, wie z.B. den Export in ein anderes Format zur Verfügung, der für die weitere Entwicklung nützlich sein könnte. Dazu

[1] [7] Das *Eclipse Graphical Modeling Framework* bietet eine generative Struktur zur Erzeugung von grafischen Editoren.

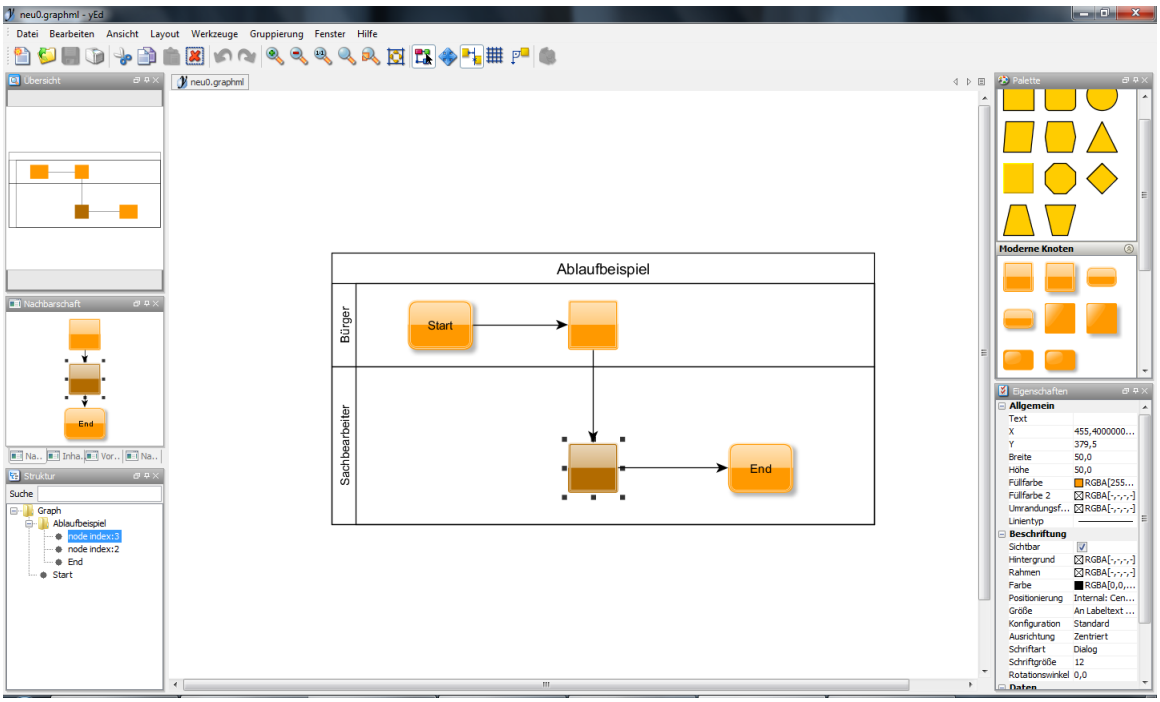


Abbildung 3.1.: Benutzeroberfläche des *yEd* Graph-Editors

wurden parallel das *GMF* und bereits existierende Editoren wie *ORYX*[2] und der *BizAgi Process Modeler*[3], die *BPMN* verwenden sowie verschiedene *UML*-Editoren und der Graph-Editor *yEd*[4] bezüglich der Anforderungen untersucht.

Der Graph-Editor *yEd*, dessen Benutzeroberfläche in Abb. 3.1 dargestellt ist, und die *UML*-Editoren waren jedoch nur bedingt für die Modellierung unserer Abläufe geeignet. Sie boten alle die Möglichkeit verschiedene Kontrollflüsse zu modellieren und sie unterstützten die Konstrukte des Aktivitätsdiagramms zur Modellierung von Rollen sowie die Modellierung von Objektflüssen. Es existierten jedoch keine weiteren Annotationsmöglichkeiten, um die Modelle mit zusätzlichen Informationen anreichern zu können. Zwar war es bei allen Editoren möglich, für jeden Knoten eine Beschreibung hinzuzufügen. Jedoch konnten beim *yEd Graph Editor* keine erweiterten Attribute, die eine allgemeine Möglichkeit bieten, eine Vielzahl an Zusatzinformationen zu annotieren, hinzugefügt werden.

Zwei weitere von uns untersuchte Editoren, die *BPMN* zur Modellierung verwenden, waren der *BizAgi Process Modeler* (Abb. 3.3) und *ORYX* (Abb. 3.2). Besonders der *BizAgi*

[2] [8] ist ein webbasierter Editor, der an der Universität Potsdam entwickelt wird
[3] [9] Der *BizAgi Process Modeler* ist ein frei erhältlicher Teil der *BizAgi BPM Suite*.
[4] [10] Der *yEd Graph Editor* ermöglicht die Erstellung allgemeiner Grafiken.

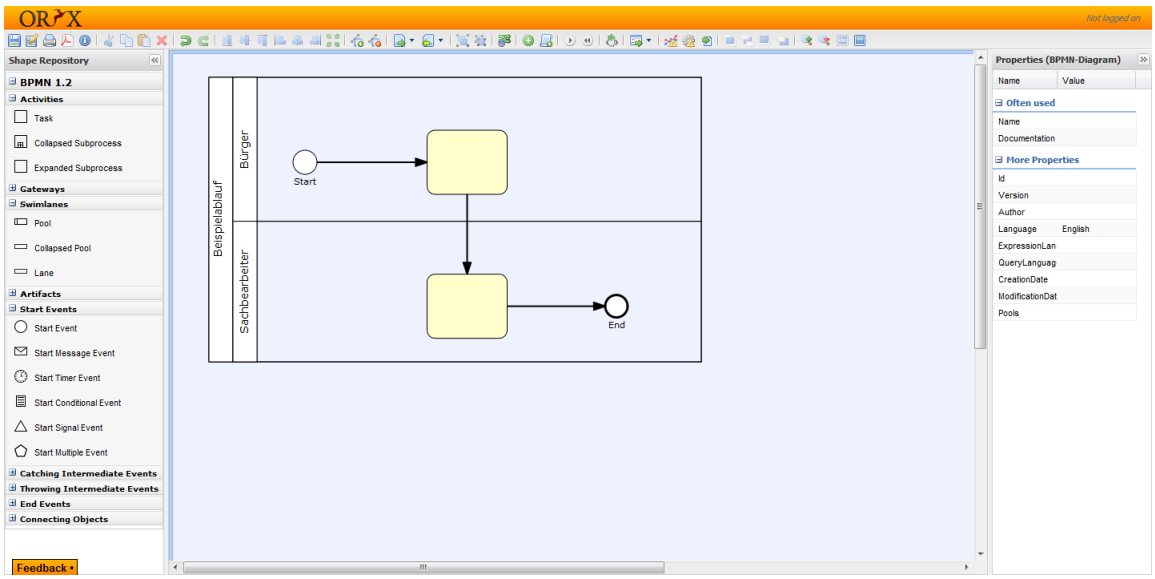


Abbildung 3.2.: Benutzeroberfläche des *ORYX BPMN*-Editors

Process Modeler bietet durch die Verwendung der *BPMN* und dem in der Spezifikation enthaltenen Konstrukt der erweiterten Attribute. Weiterhin ist es möglich über vordefinierte Eingabeoberflächen die verschiedenen Knoten mit einer oder mehreren ausführenden Personen zu verknüpfen, sowie die Ein- und Ausgaben in Form von Datenobjekten für die einzelnen Knoten festzulegen, die in unserem Fall die verschiedenen benötigten Formulare darstellen.
Eine weitere Funktion, die der *Process Modeler* von *BizAgi* bereithält, ist die *XPDL*[5] -Exportfunktion, die es ermöglicht, den ganzen Prozess mit allen Elementen und Zusatzinformationen in ein maschinell lesbares Format zu transformieren. *XPDL* ist eine *XML*-basierte Sprache zur Beschreibung von Abläufen. Sie wird dann relevant, wenn mittels der *BPMN* erstellte *Business Process Diagrams* gespeichert oder ausgetauscht werden sollen. Diese Funktion führte dazu, dass wir zusätzlich zu den bereits genannten Editoren gezielt nach *XPDL*-Editoren suchten und diese als eine weitere Möglichkeit in Betracht zogen, um die jeweiligen Abläufe zu modellieren und so eine Kombination aus Modellierung und maschinenlesbarem Export zu erreichen. Dabei untersuchten wir den Editor *Enhydra JaWe*[6], der die Möglichkeit bietet *XPDL*-Dateien zu erstellen, die konform zur *XPDL*-Spezifikation 1.0 sind.
Zusammen mit der Exportfunktion stellten die Editoren von *BizAgi*, *ORYX* und *Enhydra* und somit *XPDL* und *BPMN* die beste Möglichkeit dar, die formularbasierten

[5][11] Standard der *WfMC* zur Beschreibung von Geschäftsprozessen.
[6][12] Der *Enhydra JaWe* ist ein grafischer *XPDL*-Editor.

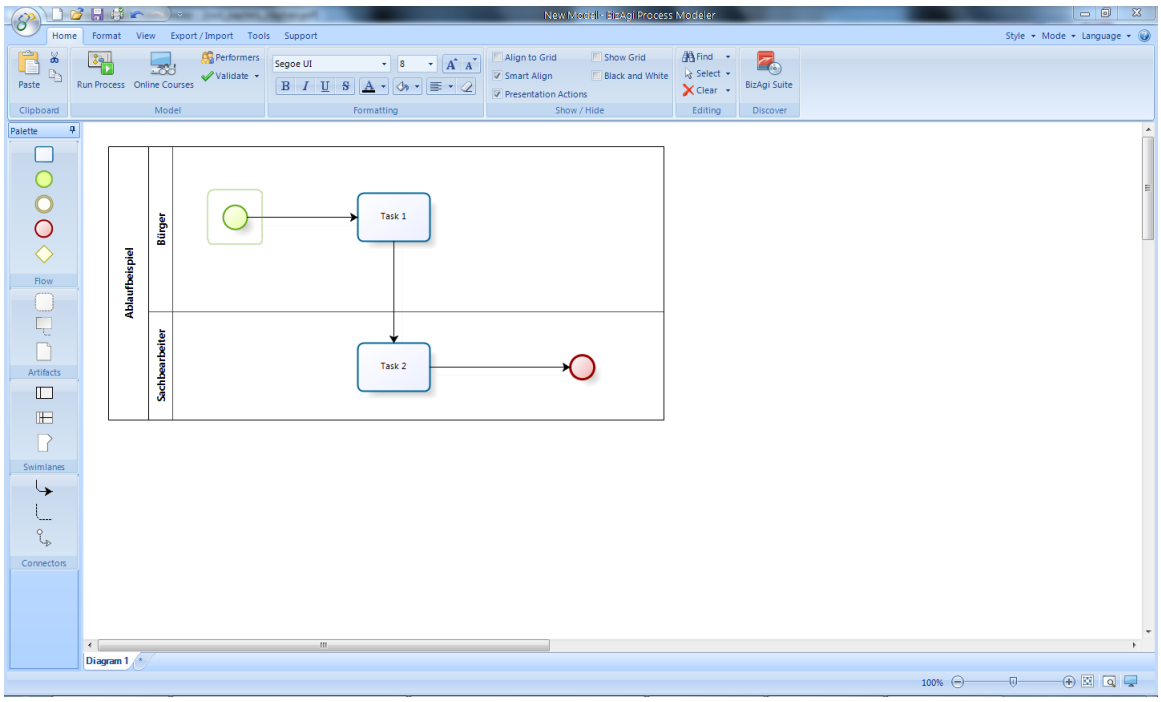


Abbildung 3.3.: Benutzeroberfläche des *BizAgi Process Modelers*

Abläufe darzustellen. Letztendlich entschieden wir uns für das Produkt von *BizAgi*, das zusammen mit der dort verwendeten *Business Process Modeling Notation* in Unterkapitel 3.2 detailliert beschrieben wird. Neben der Tatsache, dass der *BizAgi Process Modeler* im Gegensatz zu den anderen Editoren zu diesem Zeitpunkt mit der aktuellsten Version von *XPDL* arbeitete, gestaltete sich das Modellieren mit dem *Process Modeler* von *BizAgi* intuitiver als mit dem *Enhydra JaWe*. Weiterhin war es dort auch nicht in dem Maße möglich unterschiedliche Zusatzinformationen zu annotieren. Der *BPMN*-Editor von *ORYX* hatte den Nachteil, dass dort keine grafische Unterscheidung der Aktivitätstypen vorgenommen werden konnte. Die Möglichkeit, einen eigenen Editor zu erstellen wurde wegen der Eignung des *BizAgi Process Modelers* als Editor zur Modellierung der formularbasierten Abläufe verworfen.

3.2. Fachmodell

Es gibt viele Möglichkeiten den Verlauf eines Geschäftsprozesses[7] fachlich abzubilden. Zur Darstellung der Vorgänge haben wir uns für die *Business Process Modeling Notation* entschieden und werden deshalb zunächst auf die grundlegenden Merkmale dieser

[7]Die Begriffe Ablauf, Vorgang und Prozess werden im Folgenden bedeutungsgleich verwendet.

Spezifikationssprache eingehen. Anschließend zeigen wir, wie man die Prozesse mit dem *BizAgi Process Modeler* in *BPMN*-Modelle umsetzen kann, und wie der Übergang zu einem Zwischenmodell gelöst wurde.

3.2.1. Business Process Modeling Notation

Die *Business Process Modeling Notation (BPMN)* wurde von Stephen A. White, Mitarbeiter von *IBM*, zusammen mit Mitgliedern der *BPMI Notation Working Group* erstellt und ist seit 2006 offiziell ein Standard der *OMG*[8]. Sie definiert ein Diagramm, das sogenannte *Business Process Diagram (BPD)*.
Sie wurde entwickelt, um allen Benutzern eine Möglichkeit zu bieten, Geschäftsprozesse in einer leicht verständlichen grafischen Darstellung repräsentieren zu können, jedoch gleichzeitig die Modellierung komplexer Geschäftsprozesse zu ermöglichen. Ein weiteres Ziel ist die Herstellung einer Verbindung zu *XML*-basierten Sprachen zur Ausführung von Geschäftsprozessen [13]. Die grafischen Elemente von *BPMN*(Abb. 3.4) können in folgende Kategorien eingeteilt werden, wobei wir uns auf die relevanten Elemente einschränken.

- **Flow Objects** beschreiben die Knoten innerhalb des Diagramms, wie z.B. Aufgaben oder Gateways.
- **Connecting Objects** sind die verschiedenen Arten von verbindenen Kanten.
- **Swimlanes** sind die Bereiche zur Darstellung der Aktoren und Systeme.
- **Artifacts** beschreiben Elemente wie Datenobjekte, Gruppen und Annotationen.

Syntax und Semantik

Ereignis Das Ereignis wird durch einen Kreis dargestellt, der in verschiedenen Varianten auftreten kann (Abb. 3.4). Es stellt Zustandsänderungen dar und beeinflusst dadurch den Prozessablauf. Weiterhin weist es meistens einen Auslöser oder ein Ergebnis auf. Es gibt drei verschiedene Arten von Ereignissen: Startereignis, Zwischenereignis und Endereignis.

[8][14] Die Object Management Group ist ein 1989 gegründetes Konsortium, das sich mit der Entwicklung von Standards für die herstellerunabhängige systemübergreifende objektorientierte Programmierung beschäftigt.

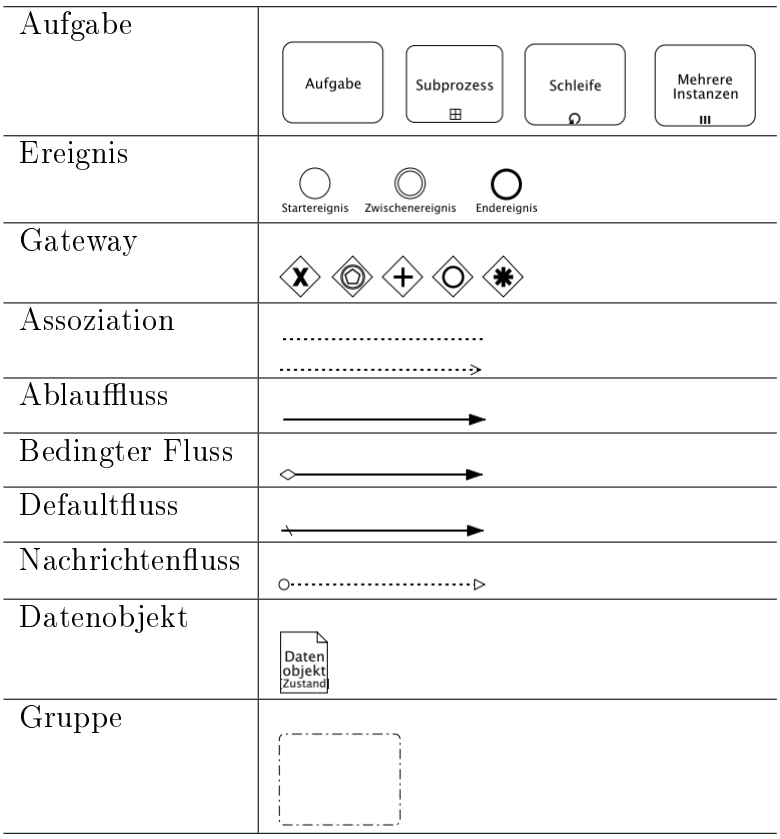


Abbildung 3.4.: Auswahl von *BPMN*-Symbolen

Aufgabe Die Aufgabe wird durch ein abgerundetes Rechteck dargestellt und symbolisiert die Ausführung einer Tätigkeit, die innerhalb eines Prozesses ausgeführt wird. Sie kann entweder atomar oder zusammengesetzt sein. Atomare Aktivitäten werden als Aufgabe bezeichnet, zusammengesetzte Aktivitäten als Subprozess. Aktivitäten können mehrmals hintereinander ausgeführt werden, solange die Bedingung erfüllt ist, und es kann zu einer Aktivität mehrere Instanzen geben.

Ablauffluss, bedingter Fluss und Defaultfluss Der Ablauffluss wird durch einen durchgehenden schwarzen Pfeil dargestellt und legt die Reihenfolge der Abarbeitung der verschiedenen Ablaufelemente innerhalb des Prozesses fest. Die Quelle und das Ziel können dabei Aktivitäten, Gateways oder Ereignisse sein. Der bedingte Fluss wird nur dann durchlaufen, wenn eine bestimmte Bedingung erfüllt ist. Der Defaultfluss wird nur dann durchlaufen, wenn kein anderer Ablauffluss durchlaufen werden kann.

Nachrichtenfluss Der Nachrichtenfluss dient zur Verbindung zwischen zwei Pools innerhalb eines Prozesses und kann ausschließlich dafür verwendet werden. Er wird mit einem gestrichelten Pfeil mit offener Pfeilspitze dargestellt.

Assoziation Eine Assoziation wird durch einen gestrichelten Pfeil dargestellt und dient hauptsächlich zur Definition der Ein- und Ausgabedaten für Aktivitäten.

Gateway Das Gateway hat die Aufgabe der Verzweigung bzw. der Zusammenführung innerhalb des Prozessflusses. Es wird als Raute dargestellt und das Kontrollverhalten wird durch verschiedene Symbole (Abb. 3.4) im Inneren der Raute festgelegt. Es sind nun folgende Möglichkeiten zu unterscheiden:

- **Datenbasiertes exklusives Gateway** Das datenbasierte exklusive Gateway wird durch eine Raute mit einem darin enthaltenden Kreuz gekennzeichnet. Bei der Verzweigung wird der Ablauffluss abhängig von der jeweiligen Bedingung genau zu einer ausgehenden Kante geleitet. Bei der Zusammenführung wird der Ablauffluss aktiviert, wenn eine der eingehenden Kanten aktiv ist.

- **Ereignisbasiertes exklusives Gateway** Der Sequenzfluss wird zu dem zuerst eintretenden Ereignis geleitet.

- **Paralleles Gateway** Das parallele Gateway wird durch eine Raute mit einem darin enthaltenden Pluszeichen gekennzeichnet. Bei der Verzweigung werden alle ausgehenden Ablaufflüsse parallel ausgeführt. Bei der Zusammenführung ist es für die Aktivierung des ausgehenden Ablaufflusses erforderlich, dass alle eingehenden Ablaufflüsse anliegen.

- **Inklusives Gateway** Das inklusive Gateway wird durch eine Raute mit einem darin enthaltenden Kreis gekennzeichnet. Bei einer Verzweigung werden ein oder mehrere Ablaufflüsse nach Prüfung der jeweiligen Bedingungen aktiviert. Bei der Zusammenführung müssen wieder alle eingehenden Ablaufflüsse anliegen.

- **Komplexes Gateway** Einer oder mehrere Ablaufflüsse werden nach Prüfung einer komplexen Bedingung oder einer verbalen Beschreibung aktiviert. Es sollte aber nur genutzt werden, falls kein anderes Gateway anwendbar sein sollte.

Abbildung 3.5.: Pool mit zwei Lanes

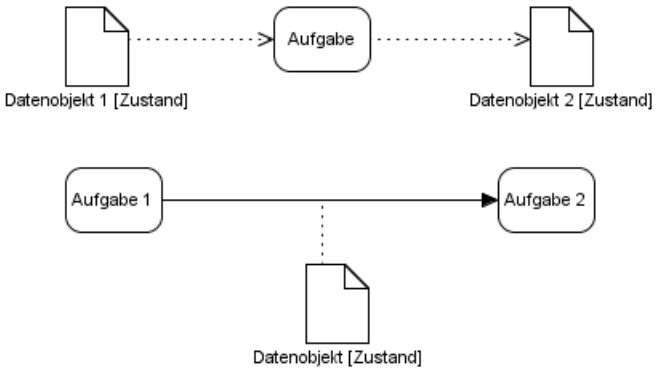


Abbildung 3.6.: Benutzung des Datenobjekts

Pool und Lane Ein Pool stellt eine Organisation dar und kann in mehrere Lanes unterteilt werden, es muss jedoch mindestens eine existieren (Abb. 3.4). Lanes repräsentieren Untereinheiten der Organisation, wie z.B. Abteilungen und organisieren und kategorisieren die im Pool enthaltenen Aktivitäten bezüglich der unterschiedlichen Zuständigkeiten, Rollen und Rechte dieser Untereinheiten innerhalb des Prozesses.

Datenobjekt Ein Datenobjekt ist optional und beeinflusst nicht den Prozessablauf. Es stellt Ressourcen dar, die innerhalb eines Prozesses benötigt oder erzeugt werden und dient der Definition des In- und Outputs von Aktivitäten. Jedes Datenobjekt hat einen Zustand, der in eckigen Klammern annotiert wird. Es wird entweder durch eine Assoziation mit der Aktivität verbunden oder durch eine gestrichelte Linie an den Ablaufffluss angebracht (Abb. 3.6).

Gruppe In einer Gruppe, die durch ein gestricheltes, abgerundetes Rechteck dargestellt wird, können verschiedene Elemente innerhalb des Diagramms zusammengefasst werden. Sie dienen lediglich der Strukturierung und Analyse und haben keinen Einfluss auf den Prozessablauf.

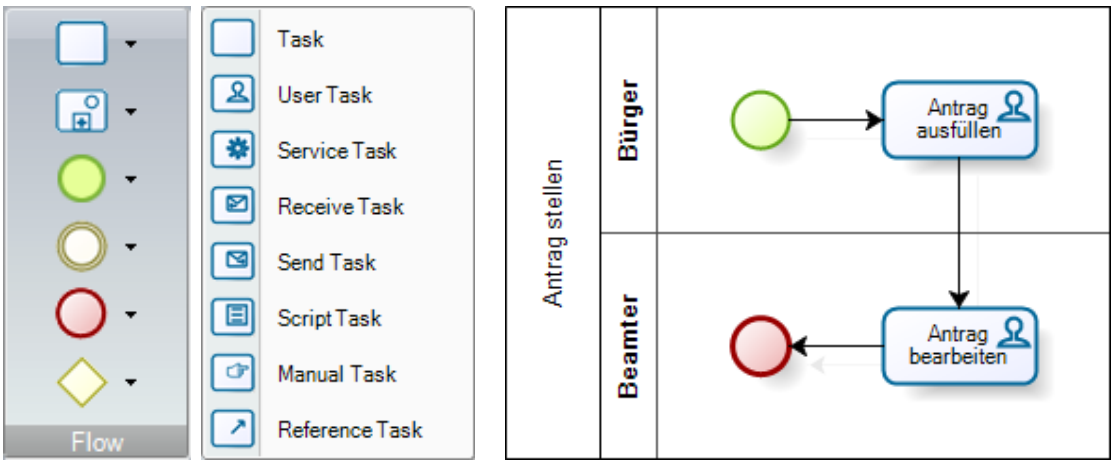


Abbildung 3.7.: Einfacher Vorgang mit zwei beteiligten Rollen

3.2.2. BizAgi Process Modeler als Werkzeug

In der vielfältigen Menge der Modellierungswerkzeuge fiel unsere Wahl auf ein Produkt der Firma *BizAgi*. Der *BizAgi Process Modeler* ist als Freeware verfügbar und unterstützt die Modellierung der Geschäftsprozesse in der aktuellen Version 1.2 der *BPMN* Spezifikation. Im Folgenden wollen wir nun demonstrieren, wie man damit gültige Fachmodelle der Vorgänge erstellen kann, die wir später als Eingabe zur Portalgenerierung verwenden. Neben allgemeinen Vorgängen dienen als Beispiele auch die beiden einfachen Szenarien *Beschwerde* und *Hundeanmeldung*, die wir als mögliche Anliegen in einem E-Government-Portal im gesamten Projekt beispielhaft umgesetzt haben. Die Reihenfolge der nachfolgenden Schritte während der Modellierung ist teilweise nicht bindend, da sie nicht unbedingt alle voneinander abhängig sind.

Analyse und Unterteilung Zunächst sollte die fachliche Beschreibung des Vorgangs, der abgebildet werden soll, untersucht und sinnvoll in einzelne Schritte[9] unterteilt werden. Diese Teilschritte lassen sich entsprechend den *BPMN*-Aktivitäten als *Tasks* darstellen. So würde man einen allgemeinen Antrag grob in die beiden Schritte *Bürger füllt Antrag aus* und *Beamter bearbeitet Antrag* unterteilen und durch jeweilige Aktivitäten darstellen (Abb. 3.7). Durch das *Start Event* (grüner Kreis) muss eindeutig festgelegt werden, wo der Prozess beginnt. Dementsprechend symbolisiert das *End Event* (roter Kreis) das Ende, das aber nur in der grafischen Darstellung zur besseren Anschauung

[9] Im Folgenden werden die Ausdrücke Schritt (*Step*), Aufgabe (*Task*) und Aktivität (*Activity*) synonym eingesetzt, um die einzelnen Aktionen eines Vorgangs zu bezeichnen.

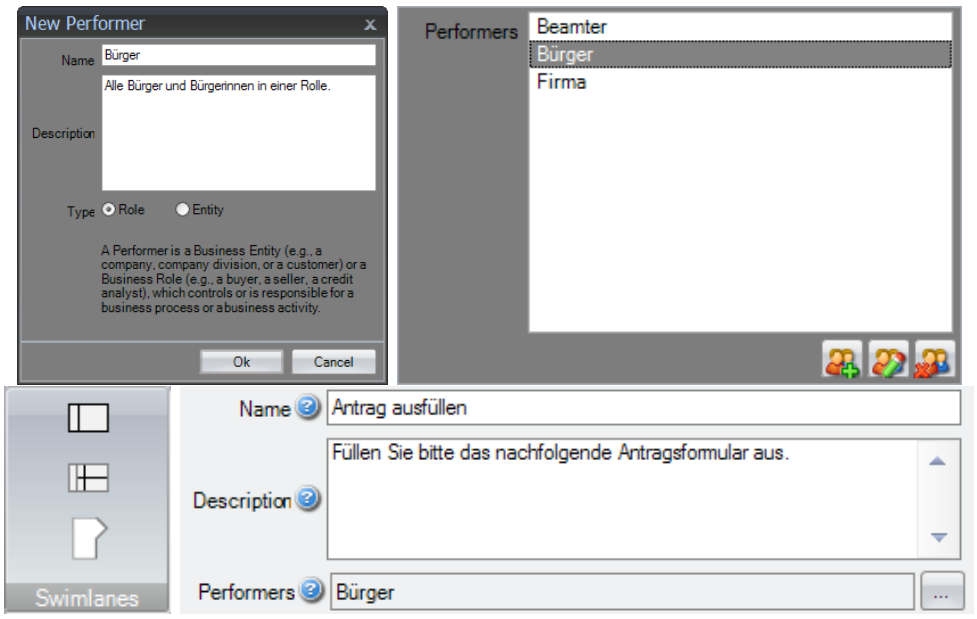


Abbildung 3.8.: Festlegung der beteiligten Rollen

dient und während der Generierung nicht zwingend erforderlich ist.

Typisierung und Aufgabenteilung In *BPMN* stehen mehrere *Activity*-Arten zur Verfügung, die wir zur Unterscheidung der Aufgabentypen benötigen. Es ist also für die reibungslose Weiterverarbeitung notwendig, dass die *Tasks* den richtigen Typ zugewiesen bekommen (Abb. 3.7). Dabei haben wir die Auswahl für unsere Zwecke auf die folgenden Typen beschränkt:

User Task: Aufgaben, die zu festgelegten Personengruppen zugeordnet werden müssen und deshalb die Schnittstelle für die Benutzer des Portals definieren.

Send Task: Diese Elemente repräsentieren in unseren Modellen einen Webservice, der später innerhalb des Portals aufgerufen wird.

Alle anderen *Task*-Typen können zwar ohne Einschränkungen verwendet werden, um die Aussagekraft des grafischen Modells zu verstärken, spielen aber während der Generierung keine Rolle. Die Zuständigkeit der Aufgaben lässt sich in *BPMN* übersichtlich durch *Pools* und *Lanes* grafisch repräsentieren. Zwar erlaubt der *BizAgi Process Modeler* das komfortable Hinzufügen mehrerer *Lanes* (Abb. 3.8), in die man die *Tasks* einfach ziehen kann, allerdings erfolgt dabei leider keine Zuweisung der Zuständigkeit zu den Rollen. Das liegt daran, dass man *Lanes* beliebig beschriften kann, sie aber nicht zu bestimmten

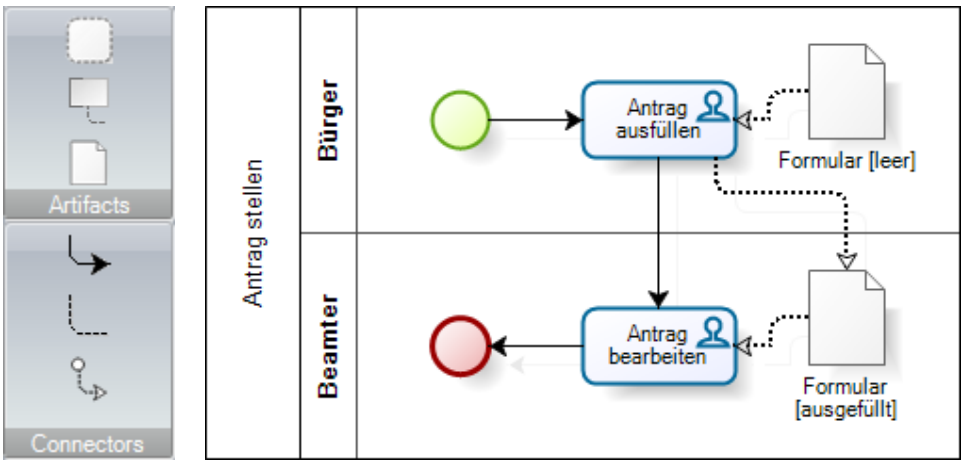


Abbildung 3.9.: Erweiterung des Beispiels aus Abb. 3.7 durch *Datenobjekte*

Benutzern zugeordnet werden können. Allerdings ist die explizite Zuweisung zu einer Rolle[10] nötig, um die Verteilung der Aufgaben später automatisch ermessen zu können. Dazu bedarf es eines kleinen Umwegs über die Eigenschaften einer Aktivität, wo man die zugehörige Rolle konkret zuweisen muss, wenn man alle *Performer* für den Prozess zuvor zentral festgelegt hat (Abb. 3.8).

Eingaben und Datenfluss Mithilfe von *Datenobjekten* können wir den einzelnen Schritten in gewisser Weise Eingabeparameter zuordnen. In unserem Fall sind es Formulare aus dem Bereich der öffentlichen Verwaltung, die wir in digitaler Form auf das Portal abbilden. Wir legen also für jedes benötigte Formular ein *Datenobjekt*[11] an und verbinden es durch gerichtete *Assoziationen*[12] mit einer Aufgabe, in der es als Eingabeobjekt dienen soll (Abb. 3.9). Die Richtung der Zuordnungsverbindungen ist entscheidend, denn sie bestimmt den Ort der Eingabe und damit auch den Datenfluss. Außerdem ist der Name des *Datenobjekts* so zu wählen, dass er mit dem entsprechenden Formularnamen übereinstimmt, damit es künftig auf das passende elektronische Formular abgebildet werden kann.

[10] Insbesondere müssen in unserem Szenario die beiden Rollen *Bürger* und *Sachbearbeiter* lauten, um sie auf die richtigen Rollen im Portal abbilden zu können.
[11] *Data Object* sind unter den *Artifacts* zu finden und werden durch ein Blattsymbol repräsentiert.
[12] *Associations* sind für *Datenobjekte* gedachte Kanten und sind bei den *Connectors* aufgeführt.

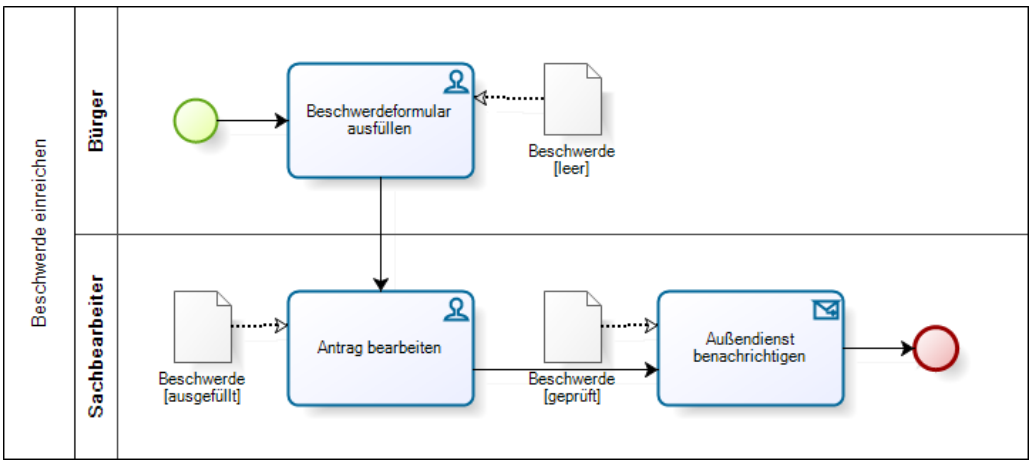


Abbildung 3.10.: Prozess mit externem Schritt als Webservice

Externe Webservices Manchmal ist es wünschenswert oder sogar erforderlich, auswärtige Dienste in das Modell aufzunehmen. Dazu stehen *Service Tasks* und *Send Tasks* zur Verfügung, die allgemeinen oder bestimmten Diensten und externen Einheiten zugeordnet werden können. Beide Elementarten können einen Webservice meinen, aber unterscheiden sich prinzipiell. Während ein *Service Task* einen Dienst versteht, der eine Antwort liefert, handelt es sich bei einem *Send Task* um eine einfache Nachricht an außenstehende Teilnehmer, die keiner Antwort bedarf. Wir haben uns im Beispiel 3.10 dementsprechend dafür entschieden, im Falle einer Beschwerde die Benachrichtigung des Außendienstes durch einen *Send Task* darzustellen. Bei der Abbildung ins Portal wird dieser Schritt dem passenden Webservice zugewiesen und in die vorhergehende Aktivität als dessen Aufruf integriert.

Verzweigung und Zusammenführung Oft enthalten fachliche Vorgänge an bestimmten Stellen mehrere Fälle, die es zu unterscheiden gilt und die zu verschiedenen Folgen führen können. Auch mit dem *BizAgi Process Modeler* ist es gemäß *BPMN* möglich, Verbindungen mithilfe der *Gateways* zwischen Objekten zu verzweigen. Für unsere Anwendung ist zusätzlich darauf zu achten, dass jede ausgehende Kante eines *Gateways* eine eindeutige Beschriftung erhält, die die Bedingung für den Eintritt des entsprechenden Falls sinnvoll beschreibt. Im Beispiel 3.11 kann man erkennen, dass nachdem der Antrag überprüft wurde, dieser entweder korrigiert werden muss, genehmigt oder abgelehnt werden kann. *Gateways* sind die einzigen Elemente, die mehrere eingehende oder ausgehende Kanten in *BPMN* regelkonformen Diagrammen aufweisen sollten. Das bedeutet,

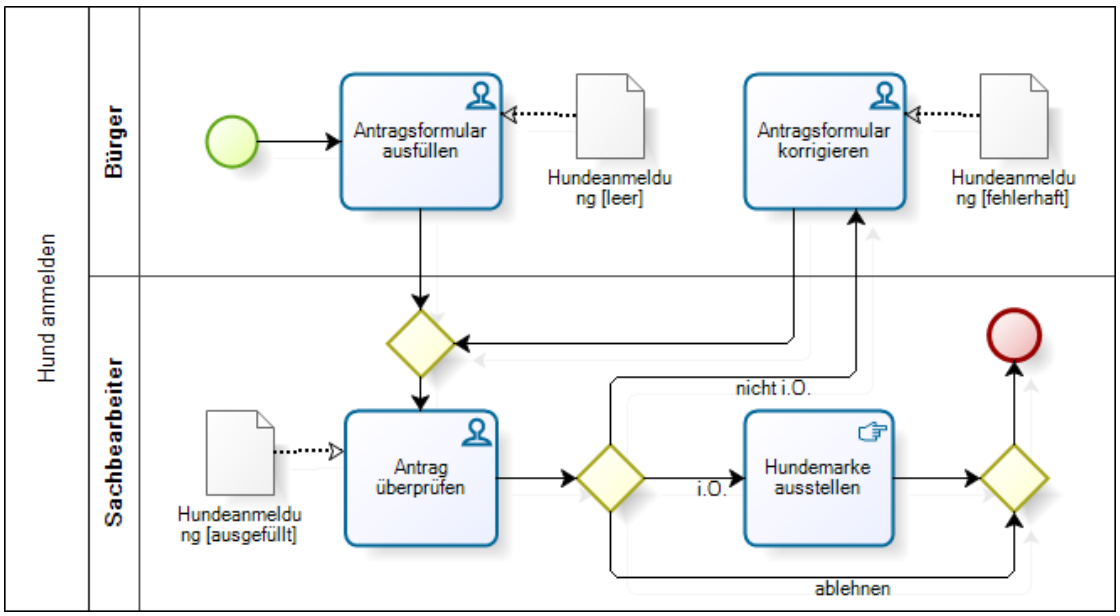


Abbildung 3.11.: Modell eines Prozesses mit mehreren möglichen Verläufen

dass *Gateways* immer dann nötig sind, wenn Kanten verzweigt oder zusammengeführt werden müssen.

Konventionen und Beschränkungen Damit die Abbildung auf das Portal problemlos funktioniert, mussten wir einige Bedingungen an korrekte Modelle der Geschäftsvorfälle stellen, aber auch den Sprachumfang von *BPMN* und die Möglichkeiten unseres Modellierungswerkzeugs bewusst einschränken. Einige dieser Konventionen wurden bereits erwähnt und werden an dieser Stelle mit den restlichen Regelungen kompakt zusammengefasst.

Rollen: Grundsätzlich stellt ein Modell den Ablauf einer Dienstleistung dar, die Bürgern durch die öffentliche Verwaltung zur Verfügung gestellt wird. Nur die beiden Rollen *Bürger* und *Sachbearbeiter* müssen also als *Performer* im Modell aufgeführt sein.

Aufgabenzuordnung: Soll eine Aktivität als Portlet im Portal dargestellt werden, muss sie der passenden Rolle zugeordnet werden und als *User Task* deklariert werden. Alle Schritte der Rolle *Bürger* werden später entsprechend im Bürgerportal angezeigt. Die korrekte Positionierung von *Tasks* innerhalb der *Lanes* ist empfehlenswert aber nicht hinreichend.

Anfang: Das *Start Event* muss mit einer Aktivität verbunden sein und somit den Anfang

Abbildung 3.12.: *XPDL*-Export als zentrale Schnittstelle zum Zwischenmodell

des Prozesses festlegen. Endknoten dienen der Übersichtlichkeit, sind aber für das Zwischenmodell entbehrlich.

Formulare: Die Eingabe für einen Schritt wird durch ein Formular parametrisiert, das als *Assoziation* mit dem zugehörigen *Datenobjekt* dargestellt wird. Der Name des *Datenobjekts* muss mit dem referenzierten Formular übereinstimmen, damit es auf das passende Formularobjekt abgebildet werden kann.

Webservices: Um externe Dienste ins Modell einbringen zu können, müssen Schritte als *Service Tasks* oder *Send Tasks* deklariert und nach der Webservice-Referenz benannt werden.

Verzweigungen: Sollen mehrere Fälle nach einer Aktivität unterschieden werden, müssen datenbasierte exklusive *Gateways* verwendet werden, um Verläufe zu verzweigen oder wieder zu vereinen. Alle von einem *Gateway* ausgehenden Kanten müssen mit unterschiedlichen Bedingungen beschriftet werden.

Allgemeines: Andere *BPMN*-Elemente finden in unseren Modellen bislang keine Verwendung und sollten zur Wahrung der Verträglichkeit mit dem Zwischenmodell nicht benutzt werden.

Export Erfüllt ein erstelltes Modell als Ganzes die oben genannten Konventionen, dann lässt es sich mithilfe der vielfältigen Exportfunktionen des *BizAgi Process Modelers* im *XPDL*-Format abspeichern (Abb. 3.12). Diese Dateien stellen unser Zwischenmodell dar und werden bei der Generierung als zentraler Teil der Eingaben verwendet.

3.3. XPDL als Zwischenmodell

In Bezug auf die spätere Generierung des Portals bedarf es der Transformation des Graphen in ein Zwischenmodell, sodass die mit dem *BizAgi Process Modeler* modellierten

Abläufe und die dazugehörigen Zusatzinformationen wie z.B. Rollenzuweisungen maschinenlesbar sind und von weiteren Komponenten verarbeitet werden können.

Genauer gesagt ist das Ziel die Erstellung einer Ablaufdatei, die neben der Ablaufstruktur mit den dazugehörigen Aktivitäten, Gateways und Transitionen, die jeweiligen Verantwortlichkeiten, die mit einer Aktivität verknüpften Formulare, den Typ der Aktivität und deren eindeutige Kennung abbildet.

Neben der Möglichkeit der Erstellung eines eigenen Schemas zur Abbildung des Ablaufs, bietet der *BizAgi Process Modeler* die bereits erwähnte *XPDL*-Exportfunktion, die es in Bezug auf die Eignung als Zwischenmodell zu untersuchen galt. Die aktuelle *XPDL* Version 2.1 unterstützt alle *BPMN*-Konstrukte und da der *BizAgi Process Modeler* diese Version der *XPDL*-Spezifikation verwendet, bot sich die Verwendung dieser generierten Dateien an.

Anhand der beiden Beispielprozesse musste nun noch geprüft werden, inwiefern die Exportfunktion die zusätzlichen Annotationen und *BPMN*-Konstrukte konform zur *XPDL* 2.1 Spezifikation exportiert. Wir kamen zu dem Ergebnis, dass die generierten *XPDL*-Dateien für unsere Ansprüche ausreichend waren, da alle modellierten Aktivitäten und annotierten Zusatzinformationen dort abgebildet wurden. Der Ansatz der Erstellung eines eigenen Schemas wurde somit verworfen, da die *XPDL*-Dateien nicht mehr angepasst werden mussten und die Exportfunktion somit eine komfortable Lösung darstellte. Zur Modellierung des Ablaufs und der Generierung eines Zwischenmodells ist daher nur eine Anwendung notwendig.

Im Folgenden werden die wichtigsten Elemente einer *XPDL*-Datei und deren hierarchische Struktur näher beschrieben (siehe auch Abb. 7.3). Dabei werden wir uns auf die Elemente beschränken, die gemäß unseren Anforderungen relevant sind. Das oberste Element ist das *Package*, das alle weiteren Elemente als Subelemente enthält. Das erste Subelement, der *PackageHeader*, enthält Informationen zur verwendeten *XPDL*-Version, dem Autor und dem Erstellungsdatum. Auf gleicher Ebene befindet sich das nächste wichtige Element, die *Participants*. Dort werden alle in dem Ablauf existierenden Rollen bzw. ihre Namen aufgelistet. Wie jedes *BPMN*-Konstrukt erhält auch jeder *Participant* ebenfalls eine eindeutige *Id*. Die Datenobjekte bzw. die Formulare mit dem dazugehörigen Namen und dem Zustand werden unter dem Element *Artifacts* und die Verbindungen zwischen den *Artifacts* und den *Activities* unter dem Element *Associations* zusammengefasst. Der eigentliche Prozess wird unter dem Element *WorkflowProcess* abgebildet. Dort sind alle im Prozess existierenden Aktivitäten unter dem Element *Activities* mit allen dazugehörigen Attributen und Subelementen aufgelistet. Für jede

Aktivität werden neben einer Beschreibung der Aktivitätstyp und der Aktivitätsname gespeichert. Weiterhin werden dort mittels einer Referenz auf einen oder mehrere *Participants* die zugehörigen Rollen angegeben. Innerhalb der Aktivitätsstruktur werden die verwendeten Formulare nach Ein- und Ausgabe mittels des *Input*- und *Output*-Elements unterschieden. Die Angabe erfolgt mittels einer Referenz auf die verschiedenen Elemente vom Typ *Artifact*. Sofern weitere Zusatzinformationen annotiert wurden, werden diese unter dem Element *ExtendedAttributes* zusammengefasst. Neben den Aktivitäten enthält der *WorkflowProcess* alle Transitionen. Um einen beispielhaften Einblick in den Aufbau einer *XPDL*-Datei zu geben werden im Folgenden für die *Transition* der Auszug aus der *XPDL*-Spezifikation beschrieben und eine konkrete Instanz, die mit dem *BizAgi Process Modeler* erstellt wurde, dargestellt:

```
<xsd:element name="Transition">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="xpdl:Condition" minOccurs="0"/>
        <xsd:element ref="xpdl:Description" minOccurs="0"/>
        <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
        <xsd:element ref="xpdl:Assignments" minOccurs="0"/>
        <xsd:element ref="xpdl:Object" minOccurs="0"/>
        <xsd:element ref="xpdl:ConnectorGraphicsInfos" minOccurs="0"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="Id" type="xpdl:Id" use="required"/>
      <xsd:attribute name="From" type="xpdl:IdRef" use="required"/>
      <xsd:attribute name="To" type="xpdl:IdRef" use="required"/>
      <xsd:attribute name="Name" type="xsd:string" use="optional"/>
      <xsd:attribute name="Quantity" type="xsd:int" use="optional" default="1">
        <xsd:annotation>
          <xsd:documentation>Used only in BPMN. Specifies number of tokens on outgoing
              transition.</xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
  </xsd:element>
```

Die Transitionen werden mittels Referenzen auf die Aktivitäten als Quell- und Zielaktivität in den Attributen *From* und *To* abgebildet. Für jede Transition kann neben einem Namen eine Bedingung angegeben werden, die unter dem Subelement *Condition* gespeichert wird. Anhand der Transitionen ist es so möglich den kompletten Graphen zu rekonstruieren und eventuelle Nachfolger bzw. Vorgänger implizit zu identifizieren. Der folgende Ausschnitt aus der generierten *XPDL*-Datei zeigt eine Instanz einer Transition:

```
<Transition Id="c4558c0f-abbb-4165-81c8-e1e86dd42c46" From="510ddf30-3178-4769-9034-fe3cbe2b167a"
    To="bf0aac39-bd9f-48f0-8054-16935a1c30b2" Name="nicht i.O.">
      <Condition Type="CONDITION">
        <Expression />
      </Condition>
      <Description />
      <ExtendedAttributes />
      <ConnectorGraphicsInfos>
        <ConnectorGraphicsInfo ToolId="BizAgi_Process_Modeler" BorderColor="0" FromPort="1"
            ToPort="7">
          <Coordinates XCoordinate="380" YCoordinate="280" />
          <Coordinates XCoordinate="380" YCoordinate="240" />
          <Coordinates XCoordinate="526" YCoordinate="240" />
          <Coordinates XCoordinate="526" YCoordinate="140" />
        </ConnectorGraphicsInfo>
      </ConnectorGraphicsInfos>
    </Transition>
```

Wie auch für jede Aktivität ist es möglich den gesamten *WorkflowProcess* mit zusätzlichen Informationen anzureichern, um ihn beispielsweise mit anderen verwandten Abläufen in Beziehung zu setzen.
Durch das *XPDL*-Format haben wir unser Ziel, ein geeignetes Zwischenmodell zu finden, erreicht und werden dieses für die Generierung des Portals verwenden, die in Kapitel 7 detailliert beschrieben wird.

4 Portal-Technologien

Das Konzept der Portale erfreut sich in den letzten Jahren immer größerer Beliebtheit. Viele verschiedene Dienste bzw. Anwendungen, wie zum Beispiel Nachrichten, Wetterberichte oder Stauwarnungen, können in ein Portal integriert werden und nutzen eine einheitliche, durch das Portal vorgegebene Oberfläche, die jedoch durch jeden Nutzer individuell konfiguriert werden kann. *Single Sign On*, das heißt sich nur einmal am Portal anmelden zu müssen und dann automatisch bei allen in das Portal integrierten Anwendungen ebenfalls angemeldet zu sein, ist ein weiteres wesentliches Dienstmerkmal.
All dies macht das Portal zu einer komfortablen, dem Nutzer so viel Arbeit wie möglich abnehmenden Art und Weise der Webnutzung.

4.1. Auswahl des Portalservers

Im Bereich der Portalserver standen drei Produkte zur näheren Auswahl, dieses waren der IBM WebSphere Portal Server [18], die Exo Portal Platform [19] und der JBoss Portal Server [20]. Alle drei bieten spezifikationskonforme Portletunterstützung sowie eine umfangreiche, fast identische Liste von Funktionalitäten. Letztlich überzeugte der JBoss Portal Server durch unkomplizierte und schnelle Installation, einfache Bedienung und umfangreiche Dokumentation. Dass er außerdem bereits auf eine längere Entwicklungs- und Nutzungsphase zurückblicken kann, spricht dafür, dass es sich um ein ausgereiftes Produkt handeln sollte.
Zu guter Letzt kann noch hervorgehoben werden, dass es sich beim JBoss Portal Server um „Free Open Source Software“ handelt, also keine Anstrengungen im Bereich Lizenzerlangung von Nöten waren.

4.2. Auswahl der Entwicklungsumgebung

Für die Erstellung der Portlets haben wir zunächst einige Entwicklungsumgebungen ausprobiert, bevor wir uns für eine davon entschieden haben. In die nähere Auswahl haben wir drei Produkte einbezogen. Namentlich waren dies der kommerzielle Rational Application Developer von IBM, das Open-Source Produkt Eclipse mit dem Eclipse Portalpack Plugin sowie das ebenfalls frei verfügbare Sun Netbeans mit dem Netbeans Portal Pack. Alle drei Produkte sind auf einem Entwicklungsstand, der die Erstellung

von Portlets nach dem neueren Standard JSR 286 ermöglicht.

4.2.1. Rational Application Developer

Diese von IBM grundsätzlich kommerziell vertriebene Anwendung bietet einen sehr großen Funktionsumfang und zahlreiche Templates für die verschiedenen Möglichkeiten von Portlets an [21].
Als großes Problem erwiesen sich die hohen Hardwareanforderungen, die diese Entwicklungsumgebung stellt. Die privat vorhandenen Computer der PG-Teilnehmer konnten diese Anforderungen fast alle nicht erfüllen, so dass mit dieser Anwendung nur auf den Pool-Rechnern gearbeitet werden konnte. Da dieses letztendlich ein Arbeiten von zu Hause ausgeschlossen hätte, haben wir uns gegen dieses Tool entscheiden müssen.

4.2.2. Eclipse

Diese Software bereitete auf den Testrechnern Probleme bei der Installation, beziehungsweise Konfiguration. Es gelang uns nicht, einen lokalen Server in das Programm einzubinden. Dieses hätte zur Folge gehabt, dass ein direktes Testen der erstellten Portlets aus der Entwicklungsumgebung heraus nicht möglich gewesen wäre. Stattdessen hätten die Webanwendungen jedes Mal auf manuelle Weise auf dem Testrechner deployt werden müssen [22].
Auch für dieses Produkt haben wir uns folglich nicht entschieden.

4.2.3. NetBeans

Im Funktionsumfang unterscheidet sich NetBeans nicht merklich von Eclipse [23]. Es wird das templatebasierte Erstellen von Portlets, der zugehörigen JSPs sowie des Deployment Deskriptors unterstützt. Jedoch werden nicht alle denkbaren Einstellungen abgefragt und automatisch eingetragen, manches muss manuell an den entsprechenden Stellen eingegeben werden. Dies gilt insbesondere für Optionen, die im Deployment Deskriptor eingestellt werden müssen. Die Installation gestaltete sich jedoch erheblich problemloser. Insbesondere wurde der lokale Server immer automatisch gefunden und korrekt eingebunden.
Für diese Entwicklungsumgebung haben wir uns letzten Endes entschieden. Wir benutzen die zu Beginn der PG aktuellsten Versionen, nämlich NetBeans 6.5.1 mit Portal Pack Version 3.0.2.

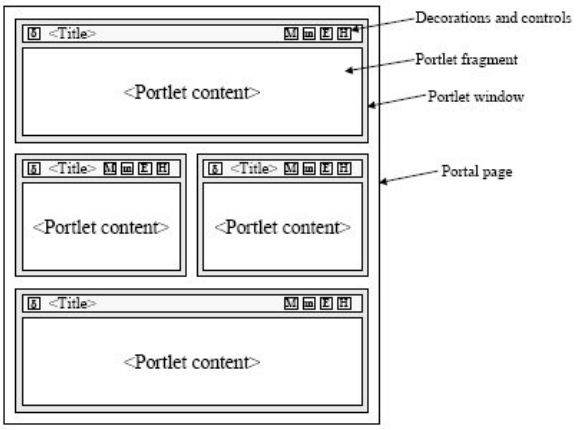


Abbildung 4.1.: Aufbau einer Portalseite [25]

4.3. Grundlagen der Portlet-Technologie

Ein Portlet ist eine Komponente der Portalseite, die statischen oder dynamischen Inhalt bereitstellt. Den Inhalt, den ein Portlet generiert, also zum Beispiel den anzuzeigenden HTML-Code, nennt man auch Fragment. Das generierte Fragment kann von Nutzer zu Nutzer unterschiedlich sein, abhängig von den nutzerspezifischen Einstellungen. Alle Fragmente bilden gemeinsam die Portalseite. Abbildung 4.1 zeigt schematisch den Aufbau eines Beispielportals. Zur Standardisierung von Portlets in Java wurden zwei JSR (Java Specification Request) herausgegeben. Dies sind „JSR 168: Portlet Specification“ [24] aus dem Jahr 2002 sowie der Nachfolger dieser Spezifikation, „JSR 286 Portlet Specification 2.0“ [25] aus dem Jahr 2008. Die Inhalte dieser Spezifikationen werden im Folgenden eingehender vorgestellt.

Grundsätzlicher Ablauf eines Portal-Aufrufs

- Ein Client (z.B. Webbrowser) sendet einen HTTP-Request an das Portal.
- Das Portal empfängt den Request und prüft, ob er eine Aktion für eines der dem Portal zugeordneten Portlets enthält.
- Falls eine Aktion enthalten ist, fordert das Portal den Portlet-Container auf das entsprechende Portlet zur Verarbeitung der Aktion aufzurufen.
- Das Portal fordert alle enthaltenen Portlets durch den Portlet-Container auf, die in die Portalseite zu integrierenden Inhalts-Fragmente bereitzustellen.

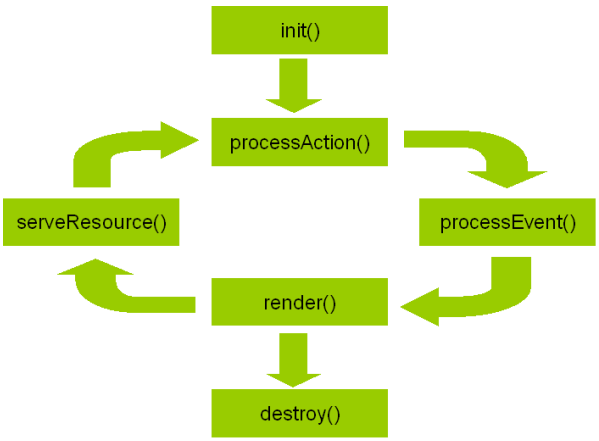


Abbildung 4.2.: Lebenszyklus eines Portlets

- Das Portal setzt die einzelnen Fragmente zu einer Portalseite zusammen und sendet diese an den aufrufenden Client.

4.3.1. Lebenszyklus

Jedes Portlet durchschreitet einen wohldefinierten Lebenszyklus. Darin ist festgelegt, wie das Portlet initialisiert wird, wie Anfragen verarbeitet werden und wie es wieder beendet wird. Dies wird durch die entsprechenden und im Folgenden beschriebenen Methoden des Portlet-Interfaces ausgedrückt. Die Abfolge der einzelnen Phasen ergibt sich aus der genannten Spezifikation und ist in Abbildung 4.2 dargestellt.

init()

Bevor ein Portlet aufgerufen werden kann, muss es initialisiert werden. In der `init()`-Methode können dabei zum Beispiel aufwändige Backend-Verbindungen erstellt werden. Die nötigen Initialisierungsparameter stammen in aller Regel aus dem Deployment Deskriptor und werden in Form eines PortletConfig-Objekts übergeben. Ob diese Initialisierung einmalig beim Starten des Containers oder separat für jeden Aufruf des Portlets erfolgt, bleibt der konkreten Implementierung überlassen.

processAction()

Benutzer, die eine Aktion im Portlet ausführen, generieren dadurch zumeist einen ActionRequest genannten Aufruf. Dieser wird in der `processAction()`-Methode verarbeitet. Zum Beispiel kann das Portlet seinen Status in Abhängigkeit von den übermittelten

Anfrageparametern verändern. Außerdem können andere Portlets durch die Erzeugung von Events über Änderungen informiert werden.

processEvent()

In der `processEvent()`-Methode werden die für das jeweilige Portlet relevanten Events verarbeitet. Diese können genutzt werden, um den Zustand zwischen verschiedenen Portlets zu koordinieren. Als Reaktion auf den EventRequest genannten Aufruf dieser Methode kann das Portlet zum Beispiel seine Eigenschaften, also etwa die Darstellungsgröße, ändern. Neue Render-Parameter können währenddessen ebenfalls gesetzt werden.

render()

Während des RenderRequests erzeugt das Portlet den auszugebenden Inhalt (sein „Fragment“) in Abhängigkeit von den übermittelten Anfrageparametern, also zum Beispiel von Portlet Mode oder Einstellungen. Die Antwort kann direkt, durch ein Servlet oder eine JSP-Seite erzeugt werden.

serveResource()

Der Aufruf dieser Methode durch den Portlet-Container wird als ResourceRequest bezeichnet. Er wird ausgelöst durch den Aufruf einer in die HTML-Seite eingetteten sogenannten ResourceURL durch den Client. Die angeforderte Ressource wird erzeugt und direkt zurückgesendet, ohne dass der Portlet-Container weiteren Inhalt einfügt. Dies kann beispielsweise genutzt werden, um AJAX-Technik zu realisieren.

destroy()

Soll ein Portlet beendet werden, geschieht dies durch Aufruf der `destroy()`-Methode. Hiermit hat das Portlet die Gelegenheit seinen eventuell persistenten Zustand abzuspeichern und belegte Ressourcen freizugeben. Anschließend wird das Objekt freigegeben. Analog zur `init()`-Methode hängt es von der konkreten Implementierung ab, wann genau ein Portlet beendet wird.

4.3.2. Request / Response

Das aus dem Bereich des HTTP [26] bekannte und bewährte Paradigma der Kommunikation über Requests und Responses findet sich auch bei den Portlets wieder. Die vom Nutzer gesendeten HTTP-Anfragen werden vom Portal in Portlet-Anfragen umge-

wandelt und bearbeitet. Die erzeugten Portlet-Antworten werden vom Portal in HTTP-Antworten gewandelt und an den Nutzer übermittelt.

Portlet Request

Die Anfrage eines Clients wird in ein Anfrageobjekt umgesetzt. Dieses enthält alle Daten der Anfrage, zum Beispiel Parameter, angeforderte Daten oder den aktuellen Portlet-Zustand. Die Gültigkeit eines Anfrageobjekts endet, sobald die aufgerufene Verarbeitungsmethode beendet wird.
Wie bereits zuvor beschrieben gibt es verschiedene Anfragearten, nämlich Actionrequests, Eventrequests, Renderrequests und Resourcerequests, welche dann jeweils an die zugehörigen Verarbeitungsmethoden übergeben werden.

Portlet Response

Das Response-Objekt enthält alle Informationen, die während eines Requests vom Portlet an den Portlet-Container zurückgegeben werden, also zum Beispiel eine Weiterleitung, einen Moduswechsel oder den erzeugten Inhalt. Der Portlet-Container nutzt diese Informationen, um die Antwort, also die Portalseite zusammenzustellen, die an den Aufrufer gesendet wird.

4.3.3. Interportlet-Kommunikation

Es existieren zwei grundlegende Methoden zur Kommunikation zwischen Portlets. Die erste ist eine passive Methode, bei der ein Portlet seine Werte einfach in einer Variablen abspeichert und andere Portlets diese Variable abfragen müssen um etwaige Änderungen zu sehen. Die zweite ist eine aktive Methode, bei der die Portlets explizit informiert werden, wenn sich Änderungen ergeben haben.

PublicRenderParameter

Dies ist die passive Variante. Ein Parameter wird im Deployment Deskriptor als PublicRenderParameter deklariert, er kann so von allen Portlets aus zugegriffen werden. Ein Portlet, das Informationen übergeben will, schreibt diese nun in den Parameter, alle Portlets, die diese Information nutzen wollen, fragen den Parameter ab. Da alle Portlets außer dem Aufgerufenen dieses nur in der Renderphase machen können, sind die möglichen Reaktionen sehr eingeschränkt.

Event-Mechanismus

Um diese Einschränkungen zu vermeiden, gibt es auch eine aktive Variante. Ein Portlet das Informationen übergeben will, erzeugt in diesem Fall ein Event. Bei allen Portlets, die Interesse an diesem Event bekundet haben, wird daraufhin nacheinander die ProcessEventphase aufgerufen. In dieser Phase können die Portlets alle Aktionen auslösen, die ihnen auch in der ProcessAction-Phase zur Verfügung stehen.

4.3.4. Packaging und Deployment

Eine der Anforderungen an eine standardkonforme Portletanwendung ist es, auf jedem ebenfalls standardkonformen Portalserver lauffähig zu sein, ohne dass größere Anpassungen erforderlich werden. Um dies zu erreichen, muss die Anwendung in einem speziellen Format vorliegen und die notwendigen Informationen zur Konfiguration in einer Datei, dem sogenannten Deployment Deskriptor, mitliefern.

Deployment Deskriptor

Der Deployment Deskriptor ist XML-basiert und enthält Angaben über Elemente und Konfigurationsinformationen von Portlet-Anwendungen und überträgt diese an das ausführende Portal bzw. den Portlet-Container. Eine Portlet-Anwendung benötigt zwei Deskriptoren, einen der Web-Ressourcen (`web.xml`), sowie einen, der die Portlet-Ressourcen spezifiziert (`portlet.xml`). Im Portlet Deskriptor stehen zum Beispiel der Name und die Standardeinstellungen des Portlets. Auch die erwähnten PublicRenderParameter und Events werden hier deklariert.

WAR-Datei

Alle Ressourcen, Portlets und Deployment Deskriptoren werden in einer genau definierten Verzeichnisstruktur abgelegt. Diese Struktur wird dann in ein Web Application Archive (WAR-Datei) zusammengefasst und kann so als Portletanwendung auf einem entsprechenden Portalserver veröffentlicht werden.

4.4. Erste Schritte

Zur Einarbeitung in den Themenkomplex der Portleterstellung haben wir zunächst einen ersten Übungsfall erarbeitet. Hierbei handelt es sich um eine Anwendung, die aus zwei Portlets besteht und auf einem JBoss-Portalserver veröffentlicht wurde. Das erste der

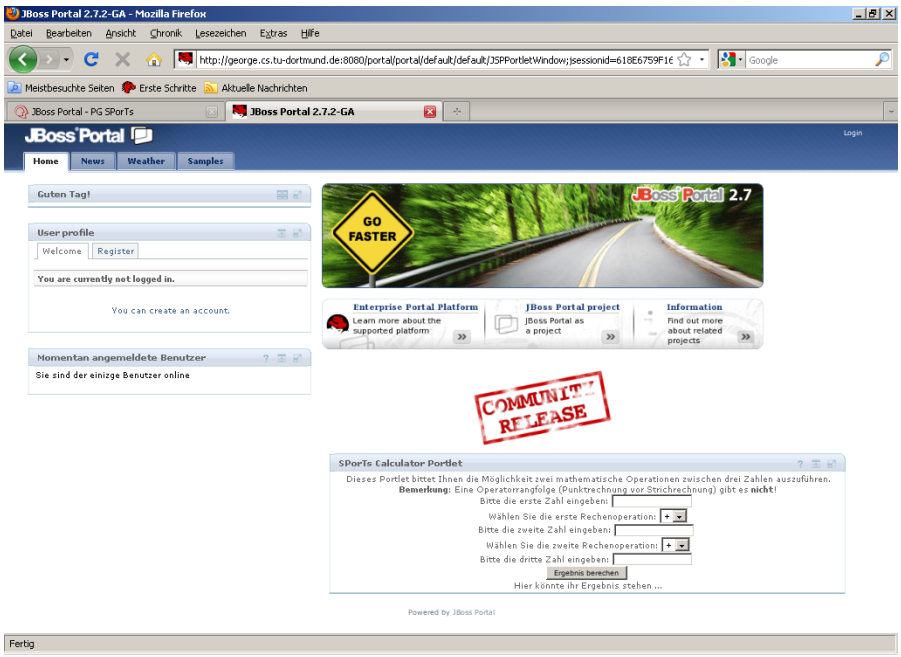


Abbildung 4.3.: Standard - Portalseite auf JBoss

beiden Portlets nimmt drei Zahlen sowie zwei Rechenoperationen entgegen und ruft mit dieser Eingabe einen ebenfalls online verfügbaren Prozess auf. Dieser Prozess führt daraufhin die Berechnung aus und sendet das Ergebnis zurück an das Portlet. Dieses wiederum bindet das Ergebnis in seine Ausgabe ein, so dass es dem Benutzer angezeigt wird. Das zweite Portlet nutzt ein Verfahren zur Interportlet-Kommunikation, um ebenfalls das Ergebnis des ersten Portlets anzeigen zu können.
Die Portlets bestehen jeweils aus einer JSP-Datei sowie der eigentlichen Portlet-Klasse. Die Darstellung des Portlets, also der angezeigte HTML-Code, wird von der JSP übernommen. Die Logik des Portlets, beispielsweise der Aufruf des Prozesses, wird in der entsprechenden `processAction()`-Methode umgesetzt. Abbildung 4.3 zeigt die Startseite der Übungsanwendung, in der Mitte unten erkennt man das beschriebene Berechnungsportlet.

5 Formulartechnologien

Die fachlichen Formulare wurden von uns mit Hilfe von XForms erstellt (Details in Kapitel 5.1). XForms ist eine Anwendung von XML um erweiterte Formularfunktionen in andere Auszeichnungssprachen wie XHTML zu integrieren [38] (siehe Abbildung 5.1). Wir verwenden XForms im Zusammenspiel mit der serverseitigen Software Orbeon Forms (siehe Kapitel 5.2). Sowohl XForms als auch Orbeon werden in den folgenden Kapiteln ausführlich vorgestellt. Als Beispiel zur Demonstration werden Quellcodeausschnitte und Bilder aus dem Vorgang „Hund anmelden“ gezeigt. Auf die jeweiligen Besonderheiten bei der Umsetzung wird an den passenden Stellen genau eingangen, da es sich meist um eine Kombination vieler eingesetzter Technologien handelt, um das gezeigte Ergebnis darzustellen.

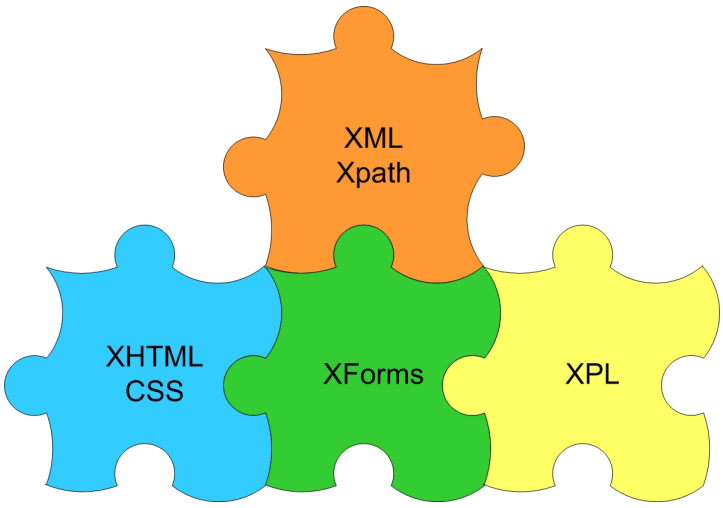


Abbildung 5.1.: Zusammenspiel der eingesetzten Technologien

Generell ist zu sagen, dass bei dem von uns gewählten Ansatz die Formulare nicht automatisch generiert werden. Die Formulare wurden per Hand mit einem Texteditor entwickelt, da die von uns betrachteten XForms-Editoren nicht unseren Ansprüchen genügten. Deshalb war keine Trennung zwischen Entwicklung und Design der Formulare möglich und beide Aufgaben wurden gleichzeitig erledigt (siehe dazu Kapitel 5.2.2). Anschließend wurden die Formulare unter einer URL innerhalb der Serverumgebung zur Verfügung gestellt. Diese URL setzt sich aus dem Präfix Vorgangsname und dem Suffix

`_buerger` bzw. `_beamter` zusammen, um eine Trennung zwischen beiden Benutzergruppen zu erreichen. In der Portalanwendung selber sind diese URLs nicht sichtbar und nicht vom Benutzer beeinflussbar, sie werden alleine als Eingabeparameter zur Generierung in passende Portlets geschrieben. Die Formulardateien sind nicht vom Benutzer änderbar, sondern müssen im Vorfeld von fachkundigen Entwicklern erstellt werden.

5.1. XForms

Die benötigten Formulare wurden von uns mit XForms [38] umgesetzt. XForms ist ein W3C-Standard für elektronische Formulare und derzeit in Version 1.1 (siehe [39]) vom 20. Oktober 2009 verfügbar. Es wurde als Format für dynamische Formulare entwickelt, das in den folgenden Abschnitten vorgestellt wird.
XForms selber ist zwar plattformunabhängig, jedoch ist derzeit eine „native Darstellung“ nur sehr eingeschränkt möglich und benötigt fast immer zusätzliche technische Hilfsmittel wie Plugins. Zahlreiche Projekte sind in Entwicklung, die eine Darstellung in populären Browsern wie Firefox (vgl. [42]) oder generell auf Clientseite ermöglichen sollen (vgl. [41]). Auch serverseitig gibt es Software zum einfachen Rendern von XForms, z. B. Orbeon Forms (siehe Kapitel 5.2). Mittlerweile unterstützen auch Programme wie OpenOffice.org XForms (vgl. [40]) und bieten Editoren an.
Einige der wichtigen Vorteile von XForms sind in der Spezifikation [38] genannt. Für uns besonders interessant waren die folgenden Argumente:

- Starke Typisierung: In XForms ist es möglich die verwendeten Daten zu typisieren. Wir benutzen dazu eine XML Schema Definition, auf die in Kapitel 5.1.1 eingangen wird. Die direkte serverseitige Validierung der Daten mit entsprechender Fehlerbehandlung beschleunigt das Ausfüllen der Formulare.

- XML Submission: XForms kann direkt mit XML-strukturierten Daten umgehen. Das heißt es entfallen unnötige Datenkonvertierungen oder Formatprobleme, sowohl auf Seite des Clients als auch auf dem Server können alle Eigenschaften von XML direkt ausgenutzt werden (z. B. die Validierung).

- Internationalisierung: Durch die Nutzung von XML 1.0 für die verwendeten Datensätze ist eine Unterstützung von internationalen Zeichen-Codierungen sichergestellt.

Der in unserer Anwendung typische Aufbau eines XForms Dokumentes (eingebettet in XHTML) sieht wie in Listing 5.1 aus:

```
1  <html <!-- Namespaces --> >
2    <head>
3      <title>Titel</title>
4      <xforms:model schema=<!-- Pfad zur .xsd-Datei -->>
5        <!-- Definition der XML-Instanzen -->
6        <!-- Definition der XForms-Submissions -->
7        <!-- Datentransformationen und Eventbehandlung -->
8      </xforms:model>
9      <!-- CSS Definition -->
10   </head>
11   <body>
12     <!-- Formularstruktur -->
13     <!-- Datentransformationen und Eventbehandlung -->
14     <!-- evtl. Widgets zum Debugging -->
15   </body>
16 </html>
```

Quellcode 5.1: Allgemeiner Aufbau der Formulardateien

Wir stellen nun die wichtigsten Abschnitte einzeln vor.

5.1.1. XML-Instanzen

Eine der wichtigsten Eigenschaften von XForms ist der einfache und effiziente Umgang mit Daten, die im XML-Format vorliegen. Dazu kann man Instanzen definieren, die im Prinzip Grundgerüste für die benutzten Daten darstellen. So bleibt dem Formularersteller die volle Kontrolle über die Struktur der Daten erhalten. Diese Daten können, wie hier in Beispiel 5.2, direkt validiert werden. Dazu gibt man eine entsprechende XML Schema Definition als Parameter an. Sollten die Daten diesem Schema nicht entsprechen, wird ein passendes XML-Event geworfen, welches anschließend ausgewertet wird (Beispiel 5.2). Jede Instanz erhält eine eindeutige ID, um den Umgang mit mehreren Instanzen zu erleichtern. Eine Instanz muss genau ein Root-Element für ihre Daten haben, im Beispiel ist dies `registration`. Auch hier kann ein Namespace angegeben werden. Im Zusammenhang mit dem Einsatz von Orbeon ist die Angabe allerdings nicht mehr optional, es muss der `xmlns`-Parameter gesetzt werden, auch wenn der Namespace-Pfad z. B. ein leerer String ist.
Unterhalb des Root-Elementes können nun die einzelnen Dateneinträge beginnen oder es können weitere untergeordnete Root-Elemente folgen. Nützlich ist dies vor allem bei Instanzen, die z.B. mit mehreren Einträgen aus Datenbankabfragen befüllt werden sollen.
Die Elemente können, müssen aber nicht mit einem Initialwert vorbelegt sein (siehe `id`

bzw. `lasteditor`). Üblicherweise setzen wir Initialwerte nur dort, wo sie für die weitere Bearbeitung nötig sind und immer ein Wert erwartet wird (z.B. `id` für folgende Datenbankabfragen). Die angegebenen Daten müssen natürlich den Defitionen im zugehörigen Schema entsprechen. Als Besonderheit kann in Instanzen auch mittels `src`-Attribut direkt ein externer Datensatz zur Initiierung dienen. Die hier definierten Instanzen sind die Grundlage für den Umgang mit Formulardaten.

```
<xforms:model schema="oxf:/apps/hundanmelden_buerger/schema.xsd">
    <xforms:instance id="form-instance">
        <registration xmlns="">
            <id>1</id>
            <!-- [...] -->
            <note>Neues Formular</note>
            <changedate>2010-01-01T00:00:00.000</changedate>
            <lasteditor/>
        </registration>
    </xforms:instance>
</xforms:model>
```

Quellcode 5.2: Beispiel zu XML-Instanzen

5.1.2. XForms Submissions

XForms Submissions sind die von XForms benutzte Lösung, um Instanz-Daten zu sammeln, serialisieren und, wenn nötig, mittels eines Protokolls an externe Ziele zu übermitteln. Die von uns genutzen Submissions dienen hauptsächlich für Datenbankabfragen und nutzen nur einen geringen Teil der Möglichkeiten aus. Eine ausführliche Betrachtung der verschiedenen Submissions findet sich in der XForms-Spezifikation [38].
Am dargestellten Beispiel 5.3 zeigt sich der generelle Aufbau einer Submission. Der Submission wird ebenfalls eine eindeutige ID zugeordnet um den Umgang zu erleichtern. Über diese ID wird sie auch später aufgerufen. Das Attribut `ref` gibt die Quelle an, von der die Daten für die Submission stammen. Hier wird explizit eine Instanz mit der ID `basic-data` genannt, es ist aber z. B. auch die Nennung einzelner Elemente aus Instanzen möglich. Das Attribut `replace` bestimmt den Umgang mit den Antwortdaten der Submission. Der Wert `instance` bestimmt, dass die Antwort-Daten eine angegebene Instanz vollständig überschreiben. Diese Zielinstanz kann in einem weiteren Attribut `instance` angegeben werden. Fehlt dieses Attribut, wie hier im Beispiel, wird standardmäßig die Instanz mit den Quelldaten der Submission überschrieben. Der Wert des Attributes `method` bestimmt nun die Methode, mit der die Daten verschickt werden. Üblicherweise benutzen wir in unseren Formularen `post`. Wie der Dokumentation [38]

zu entnehmen ist, können als mögliche Werte auch `get`, `delete`, `put`, spezielle Werte für Binärdaten oder sogar völlig eigene Werte benutzt werden. Das letzte Attribut `action` gibt schließlich die Empfänger-URL der Submission-Daten an.
Ebenfalls an Beispiel 5.3 erkennbar ist unser Vorgehen, die Submissions möglichst einheitlich und selbsterklärend zu benennen. Es wurde versucht die zugehörigen Actions unter einer dazu passenden URL abzulegen um Verwechslungen zu vermeiden.

```
1 <xforms:submission
2     id="sub-psql-select-current-user-row"
3     ref="instance('basic-data')" replace="instance"
4     method="post"
5     action="/hundanmelden_buerger/psql-select-current-user-row" />
```

Quellcode 5.3: Beispiel 1 zu XForms Submissions

Mit Hilfe der Submissions ist es auch möglich, externe Webservices einfach und effizient einzubinden. Der Quellcode-Abschnitt 5.4 zeigt den Aufruf eines Services zur Ermittlung der IP des Aufrufers. Hierbei ist zu beachten, dass die Quelldaten ein entsprechend gültiges Format aufweisen (z.B. SOAP-Envelope/Body). Das Ergebnis des Aufrufes kann wie üblich in den Instanzen weiterverarbeitet werden. Trotz der Einfachheit wurde dieser Ansatz zur Einbindung externer Webservices nicht weiter verfolgt, da sich die Generierung nicht praktikabel gestaltete. Somit ist das Beispiel 5.4 als Proof-of-Concept anzusehen.

```
1 <xforms:submission
2     id="ip-submission"  method="post"
3     action="http://www.innergears.com/WebServices/GetExternalIP.asmx"
4     ref="instance('ws-request')" replace="instance"
5     instance="ws-response"
6     mediatype="application/soap+xml;_action=&quot;http://www.innergears.com/
7 _________WebServices/GetExternalIP/getExternalIp&quot;"/>
```

Quellcode 5.4: Beispiel 2 zu XForms Submissions

5.1.3. Datentransformationen und Eventbehandlung

Die hier unter dem Begriff Datentransformationen vorgestellen Operationen sind Beispiele der sogenannten XForms Actions (weitere Actions finden sich in [38, Kapitel 10]). Diese können an verschiedenen Stellen im Formular auftreten. Je nach Verwendungszweck wird beschrieben an welcher Stelle welche Aktion sinnvoll ist.
Grundsätzlich gibt es eine Vielzahl von Aktionen, die innerhalb der XForms Formulare ausgeführt werden können. Ein `action`-Element kann mehrere weitere Aktionen beinhalten. Das Beispiel 5.5 zeigt die Kombination mehrerer Technologien und stammt aus

dem `head`-Bereich eines Formulares. Ein `action`-Element kann eine Bedingung haben und wird nur dann aktiv, wenn der Wert des `if`-Attributes `true` erreicht. Hier wird überprüft, ob ein Instanz-Wert ein leerer String ist. Näheres zu Bedingungen und XPath folgt in Kapitel 5.1.5.

```
1 <xforms:action if="string-length(instance('user-data')/pid) = 0" ev:event="xforms-ready">
2     <xforms:setvalue ref="instance('control-instance')/edit">0</xforms:setvalue>
3 </xforms:action>
```

Quellcode 5.5: Beispiel 1 zu XForms Actions

Das zweite wichtige Attribut `event` ist der Auslöser der Action. Events können an verschiedenen Stellen im Umgang mit XForms erzeugt werden. So gibt es Initialization Events, die bei einzelnen Teilschritten der ersten Darstellung des Formulares auftreten können. Dazu kommen Notification Events, die auf bestimmte Ereignisse innerhalb des Formulares reagieren können. Eine vollständige Liste mit weiteren Event-Klassen findet sich unter [38, Kapitel 4.1]. Das in Beispiel 5.5 genannte Event `xforms-ready` gehört zu den Initialization Events. Es eignet sich dadurch für Initialisierungen und ähnliche Vorgänge, bevor das Formular vom Benutzer selber verändert werden kann. Auch wenn die Actions grundsätzlich an fast jeder Stelle im Quellcode eingefügt werden können, haben wir uns darauf geeinigt, grundlegende Vorbereitungen als Actions in den Dokument-Header zu schreiben. Die konkrete Aktion im Beispiel wird durch `setvalue` beschrieben. Hier wird exemplarisch ein Wert in der referenzierten Instanz auf `0` gesetzt. XForms Actions ermöglichen es auch, komplexe Vorgänge innerhalb des Formulares zu definieren. In Beispiel 5.6 wird automatisch bei der abgeschlossenen korrekten Erzeugung des Formulares (Event `xforms-ready`) eine Submission aufgerufen (mittels der Action `send`). So können wir direkt beim Aufruf des Formulares im Hintergrund intiale Werte aus Datenbanken in Formulare laden oder auf andere Parameter reagieren. Die Submission wird, wie oben angesprochen, über ihre ID referenziert.

```
1     <xforms:send
2         submission="sub-psql-select-current-user-row"
3         ev:event="xforms-ready" />
```

Quellcode 5.6: Beispiel 2 zu XForms Actions

Die Actions werden jedoch auch im eigentlichen Formularteil eingesetzt, wenn auch meist im Zusammenhang mit Triggern, auf die im nächsten Kapitel 5.1.4 eingegangen wird.

5.1.4. Formularstruktur

Nach den vorherigen Elementen wird nun der Aufbau des Formulares mit Hilfe von XHTML beschrieben. Der grundsätzliche Umgang mit XHTML wird hier nicht genauer beschrieben. Der Fokus liegt auf Besonderheiten im Zusammenhang mit XForms.
Hierbei sind vor allem die XForms Core Form Controls zu erwähnen, die die vorher beschriebenen Elemente verbinden können. Die einfachsten Elemente sind `input` und `output`.

```
1  <xforms:output ref="instance('basic-data')/lname">
2      <xforms:label class="form-label">Nachname*</xforms:label>
3  </xforms:output>
4
5  <xforms:input ref="instance('form-instance')/cnr">
6      <xforms:label class="form-label">Hausnummer</xforms:label>
7      <xforms:action ev:event="xforms-valid">
8          <xforms:setvalue ref="instance('control-instance')/finish" value="instance('control-instance')/finish +1"/>
9      </xforms:action>
10     <!-- [...] -->
11     <xforms:hint>Maximal 10 Zeichen.</xforms:hint>
12     <xforms:alert>Die Angabe entspricht nicht dem gewünschten Format oder ist zu lang.</xforms:alert>
13 </xforms:input>
```

Quellcode 5.7: Beispiel 1 zu XForms Form Controls

Das Listing 5.7 stellt die Funktion beider Elemente vor. `output` dient zur Ausgabe eines referenzierten Wertes, der wie üblich durch den Wert des `ref`-Attributes angegeben wird. Hier wird ein passendes Element einer Instanz angegeben. Das `label`-Unterelement ist bei vielen XForms Core Form Controls wichtig und dient zur Beschriftung. Die tatsächliche Darstellung hängt allerdings vom verwendeten Renderer ab. Bei unseren Formularen wird das Label ähnlich einer Tabellenspalte vor das entsprechende Core Form Control ausgegeben. Output-Felder sind nicht mehr editierbar, können jedoch neben einfachen Strings auch andere Datenstrukturen wie z.B. Bilder ausgeben und sind universell einsetzbar. Der zweite Teil des Beispiels 5.7 zeigt ein `input`-Feld. Es dient zur Eingabe von

Abbildung 5.2.: Beispiel zum output-Element

Daten und bietet dabei einige zusätzliche Features an, die sehr nützlich sind. So wird hier im Beispiel direkt auf ein auftretendes Event `xforms-valid` reagiert. In diesem konkreten Fall werden die eingegebenen Daten mit der zuvor angegebenen XML-Schema Datei

validiert. Sollten die Daten korrekt sein, tritt das Event auf und es wird als Action der Wert des Elementes `finish` in einer Instanz um `1` erhöht. Durch diese Kombination der Core Form Controls mit passenden Actions können wir direkt bei Eingabe der Daten auf fehlerhafte Werte reagieren und Warnungen ausgeben. Diese Warnungen werden im Element `alert` definiert. Das entsprechende `hint`-Element gibt Hinweise zur Benutzung des `input`-Feldes aus. Wie `hints`, `alerts` und auch `input`-Felder tatsächlich dargestellt werden, hängt allerdings ebenfalls vom verwendeten Renderer ab. Das hier gezeigte Bild 5.3 bezieht sich also nur auf Orbeon. Analog zum `input`-Element gibt es auch `textarea`-

Abbildung 5.3.: Beispiel zum input-Element

Felder für mehrzeiligen Text und `secret`-Elemente für Passwortfelder.
Der weitere Formularaufbau wird mit anderen Elementen gestaltet. Das `trigger`-Element entspricht in etwa der Vorstellung eines Buttons und wird von uns für abschließende Aktionen seitens des Benutzers eingesetzt (z.B. Formular abschicken).

```
1  <xforms:trigger>
2      <xforms:label>Angaben korrigieren</xforms:label>
3          <xforms:toggle ev:event="DOMActivate" case="formular"/>
4  </xforms:trigger>
5  <xforms:trigger>
6      <xforms:label>Beschwerde abschicken</xforms:label>
7      <xforms:send ev:event="DOMActivate" submission="sub-psql-select-maxid-process"/>
8      <xforms:setvalue ev:event="DOMActivate" ref="instance('process-instance')/id"
           value="(instance('maxid-data')/maxid) + 1"/>
9      <xforms:setvalue ev:event="DOMActivate" ref="instance('form-instance')/process_id"
           value="instance('process-instance')/id"/>
10     <!-- [...] -->
11 </xforms:trigger>
```

Quellcode 5.8: Beispiel 2 zu XForms Form Controls

In Beispiel 5.8 ist der Quellcode für zwei `trigger`-Elemente dargestellt. Der erste Trigger ist ein Button, der eine Zurück-Funktion implementiert. Dazu wird innerhalb des Formulares durch den Druck auf den Button das Event `DOMActivate` ausgelöst und anschließend der Kontext des ganzen Formulares auf einen anderen `case` geändert. Diese unterschiedlichen Fälle lassen sich mit einem `switch`-Element abfangen und zur besseren

Strukturierung des Formulares einsetzen. In unseren Formularen werden sie als eine Art Blätterfunktion benutzt (vgl. Beispiel 5.9), wir realisieren damit verschiedene Schritte eines Ablaufes. Üblicherweise ist der erste Fall das editierbare Formular mit Eingabemasken zur Eingabe der Daten durch den Benutzer. Der zweite Fall dient zur Kontrolle der getätigten Eingaben und bietet eine Möglichkeit, zum ersten Fall zurückzukehren um Korrekturen vornehmen zu können. Der dritte Fall wird meistens nicht mehr selber angezeigt, sondern dient nur intern zur Strukturierung. In ihm werden die finalen Actions aufgerufen und das Formular „abgeschickt“, d.h. seine Zustände in der Datenbank gesichert.

```
<xforms:switch>
   <xforms:case id="control">
      <!-- Seite 1 des Formulares -->
   </xforms:case>
   <xforms:case id="end">
      <!-- Seite 2 des Formulares -->
   </xforms:case>
</xforms:switch>
```

Quellcode 5.9: Beispiel 3 zu XForms Form Controls

In Beispiel 5.8 sieht man dazu einen zweiten Trigger mit Beschriftung „Beschwerde abschicken“. Der hier gekürzte Quelltext zeigt die Kombinationsmöglichkeiten der Actions mit den Form Controls. Auf Knopfdruck werden direkt die angegebenen Anweisungen ausgeführt.
Die angesprochenen `switch-case`-Elemente gehören allerdings zur Klasse der Container Form Controls. Hier sind zahlreiche nützliche Elemente gesammelt, die auf höherer Ebene z.B. Core Form Controls kombinieren können. Jeweils eine „Seite“ unserer Formulare kann man also als entsprechenden mit Core Form Controls befüllten Container im passenden `switch-case`-Teil auffassen. Ein weiteres wichtiges Element dieser Klasse ist `repeat`.

```
<xforms:repeat nodeset="instance('dogdata-instance')/dog" id="dogrepeat">
   <tr>
      <td class="form-td">
         <forms:output ref="dogbirthday">
            <xforms:label class="form-label">Wurftag</xforms:label>
         </xforms:output>
         <!-- [...] -->
      </td>
   </tr>
</xforms:repeat>
```

Quellcode 5.10: Beispiel 4 zu XForms Form Controls

Ein `repeat`-Element kann wie in Beispiel 5.10 benutzt werden, um Iterationen zu programmieren. Im Beispiel benutzen wir die Instanz mit der ID `dogdata-instance`. Diese hat wie verlangt nur ein Root-Element `dogs`, darin aber mehrere Elemente `dog`. Über diese Elemente `dog` wird nun iteriert und jeweils eine neue Zeile zu einer Tabelle mit den passenden Daten hinzugefügt.
Die hier vorgestellten Form Controls sind nur von uns eingesetzte Beispiele. Eine vollständige Auflistung aller möglichen Elemente findet sich in den jeweils dazu genannten Quellen.

5.1.5. XPath Expressions

Ein weiteres nützliches Feature von XForms ist die Integration von XPath Expressions ([38, Kapitel 7]). XPath wird derzeit in Version 1.0 unterstützt und bietet eine einfache Möglichkeit an, per Angabe des Pfades XML-strukturierte Daten zu benutzen. Mit XPath-Ausdrücken wird in den Beispielen auf Instanzen oder Elemente in den XML-Instanzen zugegriffen.

```
1 <xforms:trigger>
2     <xforms:setvalue ev:event="DOMActivate" ref="instance('process-instance')/id"
          value="(instance('maxid-data')/maxid) + 1"/>
3     <!-- [...] -->
4 </xforms:trigger>
```

Quellcode 5.11: Beispiel zu XPath Expressions

XPath bietet aber auch zahlreiche weitere Funktionen wie etwa Kontrollstrukuren oder die Auswertung mathematischer Ausdrücke (siehe [38, Kapitel 7]), die deshalb nicht zusätzlich von uns implementiert werden mussten. Im dargestellten Quelltext 5.11 wird die Adressierung des zu ändernden Wertes wie angedeutet mittels eines Pfades realisiert. Das heißt in der ersten Zeile wird die passende Instanz mit ID `process-instance` gesucht und darin das Element `id` unterhalb des Root-Elements ausgewählt. Hier wird nun eine numerische Funktion aus XPath eingesetzt. Im `value`-Attribut wird aus einer anderen Instanz ein Wert eingelesen und um `1` erhöht. Der ganze Vorgang ist in der Action `setvalue` gekapselt. Ein ausführlicheres Beispiel (Listing 5.5) wurde bereits vorher angegeben. Ein XPath-Ausdruck wird dort zur Formulierung einer `if`-Bedingung der angegebenen Action verwendet und überprüft die Länge der Zeichenkette des referenzierten Wertes. Ist dieser Wert leer, wird die Action ausgeführt. Eine solche Struktur wurde von uns eingesetzt, um leere Übergabeparameter des aufrufenden Portlets abzufangen und stattdessen sinnvolle Initialwerte zu erzeugen. In den von uns erzeugten Fomularen

wurde nur ein Bruchteil der möglichen Funktionen eingesetzt, eine vollständige Referenz findet sich in der angegebenen Quelle.

5.2. Orbeon Forms

Wie auf der Orbeon Forms Homepage [30] und im Orbeon Forms Wiki [31] beschrieben wird, ist Orbeon Forms im Kern eine XForms Engine. Auf Basis dieser Engine wurde eine browserbasierte WYSIWYG („What you see is what you get“, „Was man sieht, ist was man bekommt“) Entwicklungsumgebung implementiert (Orbeon Forms Form Builder), mit der formularbasierte Anwendungen gebaut werden können. Solche Anwendungen können dann direkt auf der zugehörigen Laufzeitumgebung (Orbeon Forms Form Runner) veröffentlicht und gestartet werden. Geöffnet werden diese Anwendungen in einem einfachen Browserfenster und benötigen keine weitere Software für ihre Ausführung.
Orbeon Forms liefert eine Reihe von Technologien [31], die bei der Entwicklung von formularbasierten Anwendungen nützlich sind. Dazu gehören AJAX[35][34][36], eine XML Pipeline Engine zur Ausführung von XPL-Dateien (s. Kapitel 5.2.5) und einen Anwendungscontroller (Page-Flow-Controller, PFC; s. Kapitel 5.2.4). Außerdem werden Standards und Technologien wie XPath, XSLT oder die Portlet Standards JSR-168 und JSR-268 unterstützt.

5.2.1. Installation

Orbeon Forms kann auf verschiedenen Anwendungsservern installiert werden. Dazu zählen Apache Tomcat, IBM WebSphere Application Server als auch der von uns eingesetzte JBoss. Nach der Installation ist die Orbeon Forms Hauptseite unter der URL `http://localhost:8080/orbeon` erreichbar. Auf dieser Seite sind die Dokumentationen, Tutorials und Beispielanwendungen verlinkt. Für das Zusammenspiel mit Portlets, dem JBoss Portal und unserer Postgres Datenbank waren noch einige kleinere Konfigurationen der XML-Konfigurations-Dateien (web.xml, portal-postgres-ds.xml, etc.) im Verzeichnis `[ORBEON-ROOT]/WEB-INF/` nötig.

5.2.2. Form Builder und Form Runner

Der Form Builder und der Form Runner befinden sich noch in der Entwicklung und derzeit im „Alpha“-Stadium [31]. Leider gibt es noch keine ausführliche Dokumentation [31], sondern nur Benutzeranleitungen, die den Umgang mit der Entwicklungsumgebung

beschreiben. Der Form Builder hält sich laut dem Orbeon Forms Wiki [31] an spezielle Konventionen, weshalb es nicht empfohlen wird externe Formularanwendungen zu importieren. Außerdem wird teilweise Form Runner spezifischer Code erzeugt, der es verhindert die Formulare ohne erheblichen Anpassungsaufwand auf der reinen XForms-Engine von Orbeon Forms zu verwenden. Deshalb und mangels Dokumentation ist es nur möglich, die im Form Builder erstellten Formularanwendungen auf der Form Runner Engine zu veröffentlichen und auszuführen. Da wir die Formulare als Teil eines Ablaufs in einem Portal verwenden wollten, war dieser Weg leider keine Option für uns. Somit haben wir uns darauf beschränkt, Orbeon Forms allein wegen seiner XForms-Engine zu verwenden und unsere Formulare entsprechend zu entwickeln. Das bedeutete, dass wir mangels geeigneter Entwicklungsumgebungen alle nötigen Dateien per Hand in einem Editor erstellen und bearbeiten mussten.

5.2.3. Aufbau einer Formularanwendung

„Anders als bei anderen Web-Anwendungs-Plattformen, die auf Java Objeketen oder Skriptsprachen basieren, basiert Orbeon Forms auf XML Dokumenten und XForms.“ [31]
Für eine Formularanwendung in Orbeon Forms werden also lediglich XML-Dokumente oder Dokumente, deren Aufbau einer XML-Datei ähnelt (z. B. XPL-Dateien siehe Kapitel 5.2.5), benötigt. Eine simple Formularanwendung besteht aus einer mit XForms angereicherten XHTML-Datei und einer Page-Flow-Datei (s. Kapitel 5.2.4). Zunächst muss ein neuer Ordner für die Formularanwendung (z.B. HelloWorld) im Verzeichnis `Orbeon_Verzeichnis\\WEB-INF\\resources\\apps\\` angelegt werden. Die XHTML-Datei und die Page-Flow-Datei müssen in dieses Verzeichnis kopiert werden. Danach ist das beschriebene Formular unter `http://localhost:8080/orbeon/HelloWorld/` erreichbar. Über die Page-Flow-Datei können URL-Suffixe definiert werden, die die Erreichbarkeit auf speziellere URLs einschränkt. Wichtig ist jedoch das abschließende „/“, da ohne dieses das Formular nicht aufgerufen wird.
Alle weiteren Dateien, die für die Formularanwendung benötigt werden, können auch in anderen Verzeichnissen liegen, da sie über die Page-Flow-Datei referenziert werden. Um Probleme zu vermeiden, haben wir jedoch jede nötige Datei in das Verzeichnis der entsprechenden Formularanwendung kopiert.

5.2.4. Page-Flow-Controller (PFC)

Eine Formularanwendung kann aus mehr als nur einer Formulardatei (XForms und XHTML) und der Page-Flow-Datei Dateien bestehen. Zum Beispiel können mehrere Formulare oder mit XPL (siehe Kapitel 5.2.5) erstellte Dienste in einer Anwendung benutzt werden. Damit die Dateien aufgerufen werden können, müssen sie in der Page-Flow-Datei referenziert werden. Wie eine solche Page-Flow-Datei aussehen muss, zeigt Listing 5.12.

```
<config xmlns="http://www.orbeon.com/oxf/controller" xmlns:xu="http://www.xmldb.org/xupdate">

    <!-- XHTML+XForms page -->
    <page path-info="/[Formularanwendungsname]/" view="[Dateiname].xhtml" />

    <!-- Services -->
    <page path-info="/[Formularanwendungsname]/[Dienstname]" view="[Dienstdateiname].xpl"/>

    <epilogue url="oxf:/config/epilogue.xpl" />

</config>
```

Quellcode 5.12: Aufbau einer Page-Flow-Datei

Jede benötigte Datei muss einen Eintrag in der Page-Flow-Datei haben. Ein solcher Eintrag hat zwei Attribute: `path-info` und `view`.
Der Wert des `view`-Attributs gibt die Position der Datei an. Zusätzlich zum Dateinamen kann ein absoluter oder relativer Pfad verwendet werden. Ist nur der Dateiname angegeben, wird im Formularanwendungsverzeichnis gesucht.
Der Wert der `path-info` muss in der Page-Flow-Datei eindeutig sein und gibt den URL-Suffix an, unter der die Datei verfügbar ist. Dieser Wert muss beim Aufruf über eine XForms-Submission angegeben werden. Eine XForms-Submission für eine Formulardatei unterscheidet sich von einer Submission für eine XPL-Datei. Listing 5.13 und Listing 5.14 zeigen die Syntax der entsprechenden Submissions.

```
<xforms:submission id="[Submissionname]" ref="instance('[zu übergebene XML-Instanz]')" replace="instance"
      method="post" action="/[Formularanwendungsname]/[Dienstname]" />
```

Quellcode 5.13: XForms-Submission zum Aufruf einer XPL

```
<xforms:submission id="[Submissionname]" resource="/[Formularanwendungsname]/[neueFormularseite]"
      method="post" replace="all"/>
```

Quellcode 5.14: XForms-Submission zum Aufruf einer Formulardatei

Wie bereits in Kapitel 5.1.2 beschrieben, besteht eine Submission aus verschiedenen Attributen. Wie in den Listings zu sehen ist, werden für den Aufruf einer neuen Formularseite nur die Attribute `id`, `resource`, `method` und `replace` benötigt. Die in `resource` angegebene URL bzw. das Pfadfragment wird vom Page-Flow-Controller verarbeitet und die in der Page-Flow-Datei spezifierte Datei ausgeführt.
Für den Aufruf eines Services wird zusätzlich noch das Attribut `ref` benötigt, um die XML-Instance anzugeben, die an den Service übermittelt wird. Die zurückgelieferten Daten werden dann wieder in dieser Instanz gespeichert.
In unserem Projekt wurden die Formulare in einem Portlet auf dem JBoss Portal Server dargestellt. Jedoch gab es bei der Verwendung von Page-Flows zum Formularwechsel leider das Problem, dass bei einem Aufruf einer neuen Formularseite das neue Formular nicht mehr im Portlet angezeigt wurde. Dieses Problem konnte für unsere Infrastruktur leider nicht gelöst werden. Deshalb haben wir den Aufruf verschiedener Formularseiten mit `switch`- und `case`-Elementen gelöst. So muss zwar jede einzelne Formularseite in einer einzigen Formulardatei implementiert werden, jedoch kann die gesamte Anzeige innerhalb eines Portlets geschehen.
Der Aufruf von XML Pipelines konnte weiterhin mit der Page-Flow-Datei realisiert werden.

5.2.5. XML Pipeline Language (XPL)

Im Wiki von Orbeon Forms [31] werden XML Pipelines und ihre Verwendung in Orbeon Forms kurz beschrieben.
XML Pipelining ist ein Ansatz der Verarbeitung von XML-Daten, bei dem die Eingaben und Ausgaben mehrerer Verarbeitungsschritte verbunden werden. Für Orbeon Forms wurde eine XML Pipeline Engine entwickelt, die die deklarative XML Pipeline Language XPL ausführen kann.
Eine XML Pipeline besteht aus sogenannten XML Prozessoren. Dies sind Softwarekomponenten, die XML-Daten verarbeiten oder ausgeben. Für die Implementierung neuer XML Prozessoren ist Java die bevorzugte Programmiersprache. Doch Orbeon Forms liefert viele nützliche XML Prozessoren bereits mit, so dass es meist nicht nötig ist, neue zu implementieren. Zu den mitgelieferten Prozessoren gehören XSLT Prozessoren, Datenbankprozessoren und Prozessoren für die Serialisierung von XML Dokumenten.
Im Februar 2005 wurde ein Entwurf einer Spezifikation [37] dem W3C übermittelt. Im Dezember 2005 hat die XML Processing Model Working Group beim W3C begonnen an

der Spezifikation zu arbeiten. Bis heute ist die Spezifikation nicht fertiggestellt.
Im Rahmen der Projektgruppe werden XML Pipelines lediglich für die Kommunikation mit unserer Postgres Datenbank verwendet. Listing B.1 im Anhang zeigt am Beispiel einer SQL-Select-Anweisung, wie eine XPL Datei aufgebaut ist.

5.2.6. Orbeon spezifische Features

Orbeon bietet einige nützliche Features um Formulare noch benutzerfreundlicher zu gestalten. Viele dieser Features sind mit JavaScript umgesetzt. So wird ein als Datum spezifiziertes Eingabefeld zum Beispiel automatisch um einen Datepicker erweitert, so dass der Benutzer das Datum aus einem kleinen Kalender auswählen kann (Bild 5.4). Neben

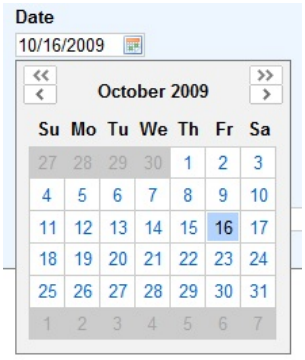


Abbildung 5.4.: Datumsauswahl

diesen automatischen Ergänzungen gibt es auch noch zusätzliche XForms-Elemente. Für diese Elemente muss der in Listing 5.15 gezeigte Namespace eingebunden werden. Danach können über den Präfix `xxforms` die Orbeon-spezifischen Elemente verwendet werden. Zum Beispiel gibt es ein mehrzeiliges Textfeld 5.5, welches automatisch die Größe an die Anzahl der Zeilen des eingegebenen Textes anpasst 5.5.

```
1 xmlns:xxforms="http://orbeon.org/oxf/xml/xforms"
```

Quellcode 5.15: Namespace für Orbeon-spezifische XForms-Elemente

Ein Element, das unsere Arbeit beim Debuggen der Formulare sehr erleichtert hat, ist der Instance Viewer. Er ist eines der vielen Widgets, die Orbeon Forms zur Verfügung stellt. Mit einer einzigen Code-Zeile (siehe Listing 5.16) kann er an einer beliebigen Stelle im Formular eingebaut werden.

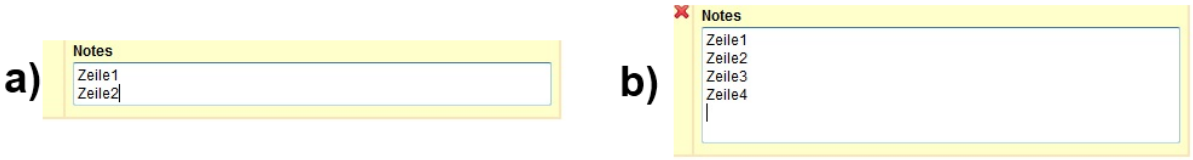


Abbildung 5.5.: Dynamisches Textfeld: a) klein, b) groß

```
1 <widget:xforms-instance-inspector xmlns:widget="http://orbeon.org/oxf/xml/widget" />
```

Quellcode 5.16: Integration des Instance Viewer Widgets

Wie im Bild 5.6 zu sehen ist, können mit Hilfe der Bedienelemente die XML-Instanzen des Formulars untersucht werden. Es kann eine Instanz ausgewählt werden, um sie sich formatiert oder als Plain-Text anzuzeigen. Alternativ können XPath Ausdrücke ausgewertet werden.

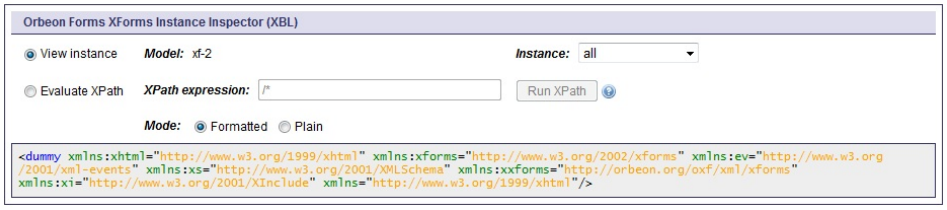


Abbildung 5.6.: Instance Viewer Widget

5.2.7. Kommunikation mit Portlets

Wie Daten vom Portlet an die Formulare übermittelt werden, wird in Kapitel 6.4.2 beschrieben. Um Daten von den Formularen an die Portlets zu senden, verwenden wir eine XForms-Submission. Mit der Submission wird eine HTML-POST-Anfrage an eine URL gesendet. Die angegebene URL gibt die Position des Portlets innerhalb der Portalstruktur und die aufzurufenen Methode an. An diese Methode wird eine XML-Instanz übergeben, die dann von der Portlet-Methode verarbeitet wird. Listing 5.17 zeigt ein Beispiel einer solchen Submission.

```
1 <xforms:submission id="sub-send-data-to-portlet" ref="instance('portlet-instance')" replace="all"
      method="post" action="http://localhost:8080/portal/auth/portal/default/Portal/Portlet?action=1" />
```

Quellcode 5.17: Aufbau einer Submission um Daten an ein Portlet zu senden

5.3. Fazit zur Arbeit mit Formulartechnologien

Die oben vorgestellten Formulare sind das Ergebnis teilweise mühsam erarbeiteter Erfahrungswerte. Orbeon erzeugt zwar Logdateien, diese sind allerdings nicht unbedingt als lesbar einzuschätzen. Es kam durchaus vor, dass wir in drei Stunden intensiver Arbeit 200 MB Logdatei-Einträge erzeugt haben. Diese sind alleine aufgrund ihrer Größe natürlich schwierig auf eventuelle Fehler zu durchsuchen. Dazu kommen die oft nicht sehr präzisen Fehlermeldungen. Ein nicht korrekt geschlossener XML-Tag generiert z.B. eine gut 150 Zeilen lange Java-Exception, in der aber kein Wort über XML oder seine Tags verloren wird. Besonders die Fehlersuche erfordert also einige Erfahrung.
Weiterhin ist die Dokumentation zu Orbeon stellenweise sehr lückenhaft. Ein grundlegendes Tutorial (siehe [43, Orbeon Forms Tutorial]) bietet einen guten Einstiegspunkt, leider sucht man gerade vertiefende Problemlösungen vergebens. Dazu ist die Dokumentation auf die Hauptseite (siehe [43]) und ein zusätzliches Wiki verteilt (siehe [31]). Auch wenn sich anscheinend sehr um einen möglichst reibungslosen Einstieg in XForms mit Orbeon bemüht wird, eine etwas klarere Struktur wäre wünschenswert. Viele Problemlösungen findet man auf der User-Mailingliste, auf der auch einige Orbeon-Entwickler sehr aktiv und hilfsbereit sind (vgl. [31, FAQ - Orbeon Forms Support]).
Eine weitere Anmerkung zu Orbeon betrifft den mitgelieferten Form Builder (siehe [44]). Der Form Builder kann zur WYSIWYG-Erzeugung von XForms verwendet werden und bietet zahlreiche nützliche Features. Leider ist der generierte Code sehr von Orbeon abhängig, d.h. ohne Orbeon sind die gebauten Formulare nicht lauffähig. Unsere erste Hoffnung mit der Wahl von Orbeon auch einen passenden einfach zu bedienenden Editor zu bekommen, wurde so leider nicht erfüllt. Trotz ausführlicher Recherche scheint es derzeit keinen Editor zu geben, der tatsächlich „reine“ XForm-Dateien erzeugen kann, die prinzipiell ohne genau passende Engine benutzbar sind. So blieb uns letzten Endes nur die Erstellung der Formulare per Hand.
Auch bei dieser Erstellung ist allerdings Erfahrung gefragt. Es gibt hier viele Wege, die zum Ziel führen und nicht immer im Quelltext klar ersichtlich sind. Die Mischung aus Struktur des Formulars mit logischen Abfragen, Verarbeitung und Veränderung der enthaltenen Daten, gleichzeitiger Validierung der Daten und per CSS-Style-Sheets auch noch generelle Optik des Formulars lassen die eigentlichen Quelltextdateien sehr aufgebläht erscheinen. Ein nicht fachkundiger Benutzer wird einige Zeit benötigen die im Formular enthaltenen Abläufe nachvollziehen zu können. Orbeon bietet zwar gewisse Hilfen an (z.B. ein Widget um Instanzen anzeigen zu lassen), trotzdem wird der Aufbau

eines Formulares schnell sehr komplex. Einige Fehler konnten erst nach mühevoller Suche beseitigt werden und hätten unserer Meinung nach bessere Hinweise verdient. Ein `output`-Feld verschwindet z.B. einfach komplett, wenn in der zugehörigen Instanz kein Wert eingetragen ist. Ein leerer String dagegen gilt als Wert. Instanzen werden nicht korrekt behandelt wenn ihnen kein Namespace zugeordnet ist, auch wenn dieser Namespace dann wiederum leer ist.
Wir sind der Überzeugung, dass der Einsatz von XForms mit Orbeon die Formularerstellung vereinfacht hat. Die mitgelieferten Möglichkeiten zur Kombination zahlreicher Technologien ersparten uns die Programmierung einer eigenen Formular-Engine. Bei der Formularerstellung mussten wir uns sogar eher einschränken, um nicht zu große Teile der Ablauflogik des Portals direkt zu integrieren. Dadurch wären die Portlets zu „Darstellungsfenster“ der Formulare reduziert worden, was nicht in unser Gesamtkonzept gepasst hätte.

6 SportsPortal - Manuelle Erstellung einer Portletanwendung

Um sich weiter in die Technologie der Portleterstellung zu vertiefen und um eine Vorlage für den zu erstellenden Generator zur Verfügung zu haben, wurde zunächst eine Version der fortan *SportsPortal* genannten E-Government-Anwendung manuell programmiert. Dieses manuell erstellte Portal wurde laufend um neue Fähigkeiten erweitert, welche anschließend dann in den später beschriebenen Generator integriert werden konnten.
Obwohl das *SportsPortal* dadurch nach und nach gewachsen ist, wird es der Übersichtlichkeit halber im Folgenden als Komplettpaket behandelt und nach Top-Down-Methodik vorgestellt.

6.1. Aufbau des SportsPortal

Das Portal verfügt über drei Bereiche, die den vorgesehenen Nutzerrollen entsprechen. Auf diese Bereiche teilen sich die insgesamt 12 Seiten auf. Jede Seite verfügt über ein Portlet zur Formularanzeige und ein Hilfeportlet zur Anzeige einer vorgangsbezogenen Hilfeseite. Einige Seiten haben zusätzlich ein Posteingangsportlet zur Anzeige von offenen Fällen oder ein Hinweisportlet, um auf verwandte Fälle hinzuweisen.
Das Portal sieht eine Aufteilung der Nutzer auf zwei Rollen vor. Dies ist zum einen die Rolle *buerger* und zum anderen die Rolle *sachbearbeiter*, wobei jeder Nutzer nur zu einer Rolle zugeordnet sein soll. Die Missachtung dieser Einschränkung kann zu doppelten Einträgen in den Posteingängen führen.
Im Detail ist das *SportsPortal* wie folgt aufgebaut:

- Alle: Diese Seite ist als Startseite festgelegt. Sie enthält einen kurzen Begrüßungstext sowie ein Portlet zur Erstellung und Verwaltung des Benutzerkontos. Sie besitzt zwei Unterseiten, die während der Entwicklung benötigte Testfunktionen realisieren.
  - Forms-Tester: Enthält ein Portlet, das eine Formular-URL abfragt und dann das entsprechende Formular anzeigt. Dies dient zum Testen der Formulardarstellung innerhalb des Portals.
  - KommunikationsTester: Enthält ein Portlet, das sämtliche ihm übergebenen

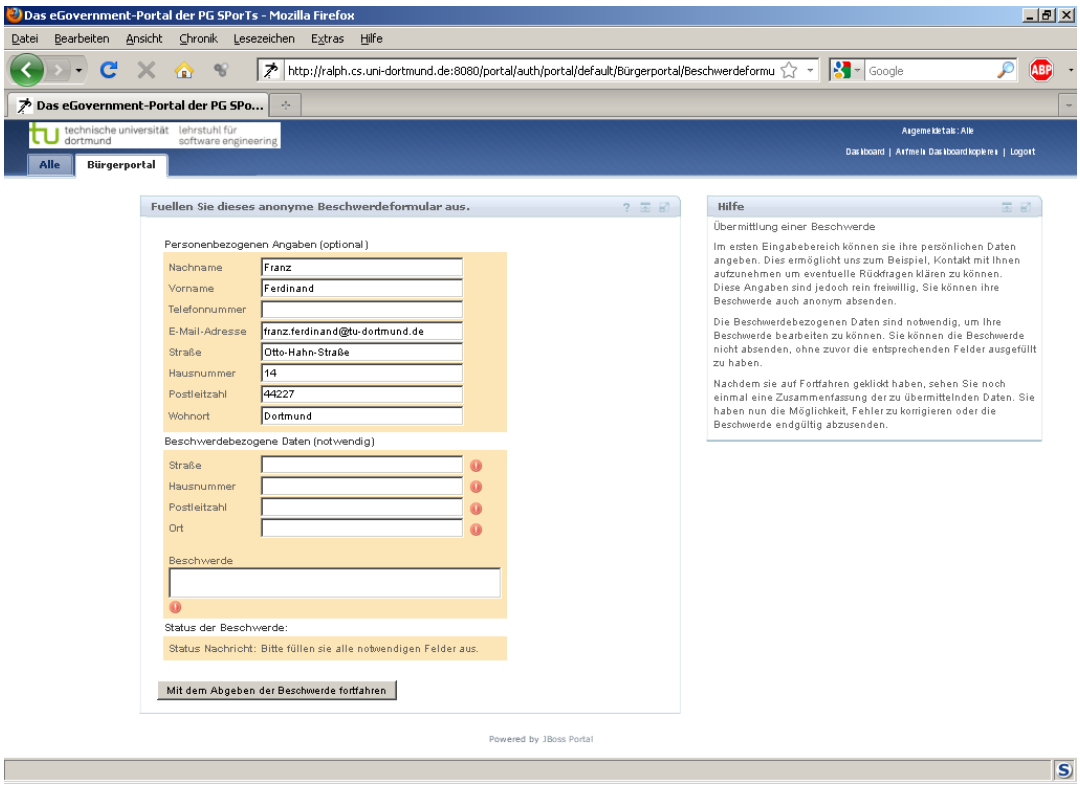


Abbildung 6.1.: Aussehen des SportsPortals

Parameter ausgibt. Dies dient zum Testen der Kommunikation zwischen Portal und Formular.

- Bürgerportal: Enthält einen kurzen Begrüßungstext. Unterhalb dieser Seite finden sich die Funktionen und Anträge, die ein Bürger benötigt. Auf diese Seite und ihre Unterseiten dürfen folglich nur Nutzer der Rolle *buerger* zugreifen.
  - Stammdaten: Enthält ein Portlet, das die in der Datenbank gespeicherten Stammdaten des angemeldeten Benutzers anzeigt und die Eingabe und Änderung dieser Daten ermöglicht.
  - BuergerPosteingang: Enthält ein Portlet, das sämtliche dem angemeldeten Nutzer zugeordneten Ablaufinstanzen anzeigt. Sollten für einen Ablauf weitere Angaben benötigt werden, so können diese hier eingegeben werden. Außerdem können noch nicht abschließend bearbeitete Anträge zurückgezogen werden.
  - Beschwerde: Enthält ein Portlet, welches das Beschwerdeformular anzeigt.
  - Hund anmelden: Enthält ein Portlet, welches das Formular zur Anmeldung

von Hunden anzeigt.

- Sachbearbeiterportal: Enthält einen kurzen Begrüßungstext. Unterhalb dieser Seite finden sich die Formulare, die ein Sachbeabeiter benötigt. Auf diese Seite und ihre Unterseiten dürfen folglich nur Nutzer der Rolle *sachbearbeiter* zugreifen.
  - Stammdaten: Enthält ein Portlet, das die in der Datenbank gespeicherten Stammdaten des angemeldeten Benutzers anzeigt und die Eingabe und Änderung dieser Daten ermöglicht.
  - Beschwerdefall: Enthält ein Posteingangsportlet, das alle offenen Beschwerdefälle anzeigt und dem Bearbeiter die Auswahl des als Nächstes von ihm zu bearbeitenden Falls ermöglicht. Weiterhin enthält die Seite ein Portlet, das das Formular des ausgewählten Vorgangs anzeigt und dessen Bearbeitung ermöglicht.
  - HundAnmeldenfall: Enthält ein Posteingangsportlet, das alle offenen Hundanmeldungen anzeigt und dem Bearbeiter die Auswahl des als Nächstes von ihm zu bearbeitenden Falls ermöglicht. Weiterhin enthält die Seite ein Portlet, das das Formular des ausgewählten Vorgangs anzeigt und dessen Bearbeitung ermöglicht.

6.2. Struktur und Inhalt der zugehörigen WAR-Datei

Alle Dateien, die für das zuvor beschriebene Portal benötigt werden, sind in einem *Web Application Archive* zusammengefasst. In diesem Kapitel werden zunächst die einzelnen Bestandteile dieser WAR-Datei vorgestellt. Die wesentlichen Bestandteile werden anschließend in eigenen Kapiteln näher betrachtet.
Auf der obersten Ebene befinden sich zwei Verzeichnisse, `images` und `WEB-INF`, und eine CSS-Datei. Letztere enthält das Stylesheet, das für das gesamte Layout des Portals verantwortlich ist, der Ordner `images` enthält die zugehörigen Grafiken. Alle restlichen Dateien befinden sich im Ordner `WEB-INF`. Dieser enthält insgesamt fünf Deployment Deskriptoren, die jeweils bestimmte Konfigurationsdaten enthalten. Außerdem befinden sich in ihm zwei weitere Verzeichnisse, nämlich der Ordner `classes`, der für jedes Portlet die zugehörige class-Datei enthält, sowie der Ordner `jsp`, der die im Portal benutzten *Java Server Pages* enthält.

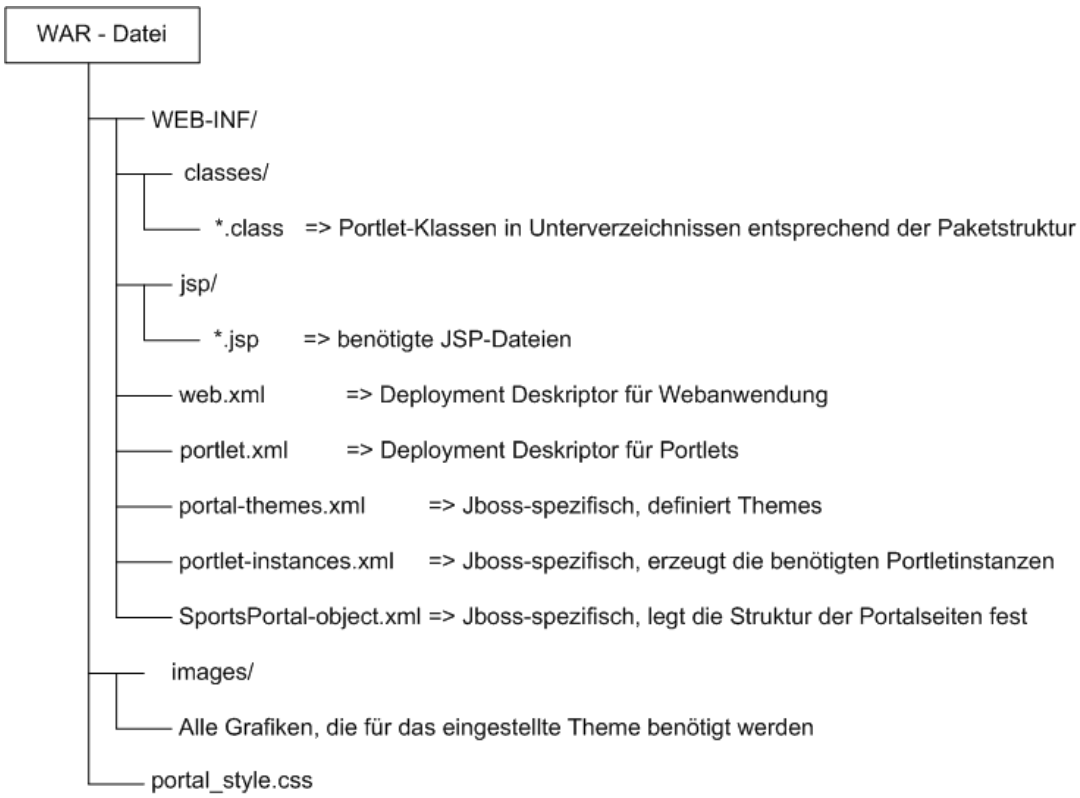


Abbildung 6.2.: Struktur der WAR-Datei

6.3. Beschreibung der Deployment Deskriptoren

Das *SportsPortal* benötigt insgesamt fünf Konfigurationsdateien, die sogenannten Deployment Deskriptoren. Zwei davon werden von der Portletspezifikation, die restlichen von dem bei uns eingesetzten JBoss Portal Server gefordert. Bei den Deskriptoren handelt es sich um XML-Dateien. Sie beschreiben beispielsweise den Aufbau des Portals oder geben bestimmte Einstellungen, die der Portalserver beachten soll, bekannt. Die genaue Funktion der einzelnen Deskriptoren wird in den folgenden Unterkapiteln beschrieben.

6.3.1. Deskriptor web.xml

Dieser Deskriptor ist notwendig, da eine Portletanwendung immer auch eine Webanwendung ist und eine solche nach der zugehörigen Spezifikation JSR 315 [27] diesen Deskriptor besitzen muss.
Abgesehen davon hat er für das Portal aber keine Bedeutung.

6.3.2. Deskriptor portlet.xml

Dieser Deskriptor wird von der Portlet-Spezifikation [25] vorgeschrieben. Er enthält für jedes im Portal vorkommende Portlet einen `<portlet>`-Block, der grundlegende Eigenschaften angibt. Insbesondere werden Name, Verweis auf die zugehörige Java-Klasse sowie erzeugte und konsumierte Events angegeben. Außerdem müssen alle im Portal vorkommenden Events in diesem Deskriptor explizit deklariert werden. Ein Codebeispiel findet sich im Anhang E.1

6.3.3. Deskriptor portal-themes.xml

In diesem JBoss-spezifischen Deskriptor werden Themes, also mögliche Erscheinungsbilder für das Portal deklariert. Eine solche Deklaration enthält neben einem Namen für das Theme vor allem einen Verweis auf die zugehörige Stylesheet-Datei. Das Sports-Theme benutzt zusätzlich noch ein eigenes sogenanntes Favicon [29], also eine kleine Grafik die Browser zu Lesezeichen hinzufügen können.

6.3.4. Deskriptor portlet-instances.xml

Auch dieser Deskriptor ist speziell für den JBoss Portal Server anzulegen. Er dient zur Instanziierung der im Portal anzuzeigenden Portlets. Dabei ist es auch möglich, mehrere Instanzen von demselben Portlet zu erzeugen. Zusätzlich können für die einzelnen Instanzen noch Parameter festgelegt werden, welche das Portlet zur Laufzeit auswerten kann. Dieser Mechanismus wird im *SportsPortal* beispielsweise benutzt, um die richtigen JSPs in den Hilfeportlets anzuzeigen.
Für jede zu erzeugende Instanz wird ein `<deployment>`-Block benötigt. In diesem Block steht zum Einen eine Anweisung die bewirkt, dass zum Deploy-Zeitpunkt eventuell vorhandene Instanzen des selben Namens überschrieben werden, sowie zum Anderen ein `<instance>`-Block. In diesem Block wird die Instanz mit einer eindeutigen ID versehen und es wird angegeben, von welchem Portlet die Instanz abgeleitet werden soll. Falls gewünscht, können noch Parameter jeweils mit Namen und Wert festgelegt werden. Ein Codebeispiel findet sich im Anhang E.2

6.3.5. Deskriptor SportsPortal-object.xml

Der dritte JBoss-spezifische Deskriptor beschreibt die Struktur des Portals. Neben der Deklaration des Portals selber sowie einigen Einstellungen dafür, sind dies die Seiten und Unterseiten, aus denen das Portal bestehen soll, sowie die Verteilung und Anordnung der zuvor erzeugten Portlet-Instanzen auf die einzelnen Seiten. Im ersten Teil des Deskriptors (Anhang E.3) wird das Portal deklariert und mit dem Portalnamen `default` als Standardportal festgelegt. Weiterhin wird eingestellt, dass die Portlets des *SportsPortals* nur den View-Modus unterstützen sollen, welches Layout und welches Theme verwendet werden soll, dass die Seite *Alle* die Startseite des Portals sein soll und dass der Zugriff auf das Portal keine Authentifizierung erfordert. Im zweiten, nur Aussschnittsweise wiedergegebenen Teil des Deskriptors (Anhang E.4), folgt die Festlegung der Struktur des Portals. In erster Ebene besteht es aus Seiten (`pages`). Für jede Seite können eigene Zugriffsrechte festgelegt werden, im Beispiel oben wird festgelegt, dass nur Nutzer der Rolle *buerger* die Seite *Bürgerportal* und deren Unterseiten sehen dürfen. Über die Eigenschaft `order` kann die Sortierung im Navigationsmenü beeinflusst werden.
Jede Seite kann weitere Seiten, welche dann als Unterpunkte im Navigationsmenü angezeigt werden, und Fenster (`windows`) beinhalten. Für jedes Fenster wird dabei festgelegt, welche Portletinstanz in diesem dargestellt wird und wo es auf der Seite angeordnet werden soll.

6.4. Portletklassen

Die verschiedenen, für das *SportsPortal* erstellten Portlets können entsprechend ihrer Funktionsweise in unterschiedliche Gruppen eingeteilt werden. Im Wesentlichen gibt es Portlets, die ein Orbeon-Formular abrufen und darstellen, Portlets, die einen Posteingang realisieren und Portlets, die sonstige Hilfsfunktionalitäten zur Verfügung stellen.
Allen Gruppen gemeinsam ist dabei das folgende Grundgerüst einer Portletklasse mit der Methode `doView()` zur Verarbeitung von GET-Anfragen, in der Portletspezifikation RenderRequests genannt, und der Methode `processAction()` zur Verarbeitung von POST-Anfragen, in der Portletspezifikation ActionRequests genannt (Siehe auch Kapitel 4.3).

```
1  package sports.portlets;
2  import javax.portlet.*;
3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import sports.support.*;
6  //ggf. weitere benötigte imports
7
8  public class XYPortlet extends GenericPortlet {
9      public void processAction(ActionRequest request, ActionResponse response) throws
           PortletException,IOException {
10   //place processAction-Code here
11   }
12
13   public void doView(RenderRequest request, RenderResponse response) throws PortletException,IOException
           {
14   //place doView-Code here
15   }
16 }
```

Quellcode 6.1: Grundlegender Portletaufbau

6.4.1. Posteingangsportlet für den Bürger

Der Bürgerposteingang zeigt alle vorhandenen Vorgänge des aktuell angemeldeten Nutzers mit ihrem gegenwärtigem Status an. Der Bürger hat hier die Möglichkeit, einen noch nicht abschließend bearbeiteten Antrag zurückzuziehen oder einen vom Sachbearbeiter als fehlerhaft markierten Vorgang zu korrigieren.

6.4.2. Formularportlets für den Bürger

Zu dieser Gruppe gehören die Portlets mit den Antragsformularen ebenso wie die Portlets zur Anzeige der Stammdaten. Alle Portlets rufen jeweils ein Orbeon-Formular ab und zeigen dieses dem Bürger an. Dabei werden neben der Formularadresse noch einige weitere Daten an den Server übermittelt. Konkret können dies beispielsweise die Typ-ID des Prozesses oder der Name und die Rolle des aktuellen Benutzers sein. Diese Daten werden dabei als Eigenschaften (`properties`) im Header des HTTP-Requests übermittelt.
Der Paramter `submitted` dient zur Unterscheidung, ob ein Formular noch ausgefüllt wird oder schon abgeschickt wurde. Nach dem Abschicken des Formulars wird dieser Parameter auf `true` gesetzt, so dass ein kurzer Rückmeldungstext anstatt eines neuen Formulars eingeblendet wird. Der zugehörige Codeausschnitt befindet sich im Anhang

E.5.

Die `processAction()`-Methode wird nach dem Abschicken des Formulars aufgerufen. Dabei übermittelt das Formular im Body des HTTP-Aufrufs eine XML-Struktur mit der ID des eingegebenen Datensatzes, der in der Datenbank angelegt wurde. Diese wird benötigt, um die Status-ID eben dieses Vorgangs korrekt auf den neuen Wert setzen zu können. Außerdem wird noch der Parameter `submitted` auf `true` gesetzt, damit in der folgenden `doView()`-Methode der Rückmeldungstext angezeigt wird.
Sollte der Benutzer in diesem Rückmeldungstext auf den Button *neues Formular* klicken, so enthält der ausgelöste ActionRequest den Parameter `new`. Das führt dazu, dass `submitted` wieder auf `false` gesetzt und damit durch die folgende `doView()`-Methode ein neues Formular angezeigt wird.

```
1  public void processAction(ActionRequest request, ActionResponse response) throws
       PortletException,IOException {
2      String temp = request.getParameter("new");
3      if (temp != null && temp.equals("new")){
4          response.setRenderParameter("submitted", "false");
5      }else{
6          try {
7              DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
8              DocumentBuilder db = dbf.newDocumentBuilder();
9              Document doc = db.parse(request.getPortletInputStream());
10             NodeList ids = doc.getElementsByTagName("process_id");
11             if (ids.getLength() > 0) {
12                 response.setRenderParameter("submitted", "true");
13                 DataBase.dbUpdate("UPDATE process SET process_state_id=4 WHERE
                       id="+ids.item(0).getTextContent());
14             }
15         } catch(Exception e) {
16             e.printStackTrace();
17         }
18     }
19 }
```

Quellcode 6.2: processAction()-Methode eines Formularportlets für Bürger

6.4.3. Posteingangsportlets für die Sachbearbeiter

Der Übersichtlichkeit halber existiert für jeden Sachbearbeiterschritt ein eigener Posteingang, der alle noch zu bearbeitenden Fälle dieses Schritts anzeigt. Dazu werden die offenen Fälle in der Datenbank abgefragt und zeilenweise ausgegeben. Für jeden Fall wird ein Button erzeugt, der beim Drücken die entsprechende ID an das Portlet übermittelt. Diese ID wird in der `processAction()`-Methode ausgelesen und mittels

eines Events an das zugehörige Sachbearbeiterportlet übermittelt. Dies ist im folgenden Listing dargestellt.

```
1  public void processAction(ActionRequest request, ActionResponse response) throws
       PortletException,IOException {
2      String id = request.getParameter("id");
3      Integer zahl;
4      try {
5          zahl = Integer.parseInt(id);
6      } catch (Exception e) {
7          zahl = 0;
8      }
9      response.setEvent("sports.hundanmeldenauswahlevent", zahl);
10 }
```

Quellcode 6.3: processAction()-Methode eines Posteingangsportlets

6.4.4. Formularportlets für den Sachbearbeiter

Im Vergleich zu den Formularportlets für den Bürger enthält diese Gruppe von Portlets etwas mehr Logik. Sie implementieren zusätzlich die `processEvent()`-Methode zur Verarbeitung der durch die Posteingänge ausgelösten Events. Ein solches Event enthält wie zuvor beschrieben die ID des Vorgangs, der dem Sachbearbeiter im Portlet angezeigt werden soll. Diese ID wird mittels Parameter an die folgende `doView()`-Methode weitergeleitet und von dort an den Formularserver übermittelt, so dass dieser das Formular mit den entsprechenden Vorgangsdaten befüllen kann. Das folgende Listing zeigt die beschrieben `processEvent()`-Methode.

```
1  public void processEvent(EventRequest request, EventResponse response) throws PortletException,IOException {
2      Event event = request.getEvent();
3      Integer id = 0;
4      id = (Integer)event.getValue();
5      response.setRenderParameter("HundAnmeldenevent", id.toString());
6  }
```

Quellcode 6.4: processEvent()-Methode eines Formularportlets für Sachbearbeiter

Auch die `processAction()`-Methode ist bei diesen Portlets geringfügig umfangreicher. Da der Sachbearbeiter üblicherweise mehrere Wahlmöglichkeiten, zum Beispiel annehmen oder ablehnen, für den weiteren Prozessverlauf hat, muss sich die Statusaktualisierung in der Datenbank natürlich der getroffenen Entscheidung anpassen. Dazu wird die getroffene Entscheidung in Form einer Zeichenkette an diese Methode übermittelt und in Abhängigkeit davon das korrekte SQL-Statement ausgewählt. Der zugehörige Codeausschnitt befindet sich im Anhang E.6.

6.4.5. Sonstige Portlets

Von den sonstigen Portlets lohnt es sich noch, das Hilfeportlet und das Hinweisportlet näher zu betrachten. Sie dienen zur Darstellung der Hinweis- und Hilfetexte in der rechten Spalte des Portals und funktionieren beide nach dem selben Prinzip. Sie generieren die Ausgabe nicht selber, sondern leiten dazu an eine JSP-Datei weiter. Die Adresse wird dabei aus einer Einstellungsvariablen genommen, die bei der Instanziierung im `instances`-Deskriptor gesetzt wurde.

```
1 public void doView(RenderRequest request,RenderResponse response) throws PortletException,IOException {
2     response.setTitle("Hilfe");
3     response.setContentType("text/html");
4     String page = request.getPreferences().getValue("page", "NoHelp");
5     PortletRequestDispatcher dispatcher =
6     getPortletContext().getRequestDispatcher("/WEB-INF/jsp/" +page+ "Portlet_help.jsp");
7     dispatcher.include(request, response);
8 }
```

Quellcode 6.5: doView()-Methode des Hilfeportlets

Weiterhin verfügt das *SportsPortal* noch über das Begrüßungsportlet, dass lediglich einen kurzen Begrüßungstext ausgibt sowie über zwei Testportlets, die während der Entwicklung benötigt wurden. Letztere dienten im Wesentlichen dazu, die interne Kommunikation zwischen den Formularen und dem Portal anzuzeigen und nachvollziehen zu können.

6.4.6. Verwendete Java Server Pages

JSP-Dateien werden im Portal nur für die Anzeige der Hilfe- und Hinweistexte verwendet, für jede Seite des Portals wird dabei eine Hilfe- und eine Hinweisdatei benötigt. Im Prinzip handelt es sich dabei um HTML-Dateien, die auf spezielle Art mit Java-Befehlen angereichert werden können. Ein typisches Beispiel zeigt das folgende Listing.

```
1 <%@page contentType="text/html"%>
2 <%@page pageEncoding="UTF-8"%>
3 <%@ page import="javax.portlet.*"%>
4 <%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet"%>
5 <portlet:defineObjects />
6 <%PortletPreferences prefs = renderRequest.getPreferences();%>
7
8 <b>
9     Bearbeitung von Bürger-Beschwerden.
10 </b>
11 <p>
12     Bitte wählen sie zunächst den Beschwerdevorgang, den Sie bearbeiten wollen aus der Tabelle aller offenen
          Vorgänge aus.
```

```
13 </p>
14 <p>
15     Nach einem Klick auf die entsprechende Schaltfläche erscheint der entsprechende Vorgang im unteren
           Portlet.<br>
16     Sie können ihn nun wie gewohnt bearbeiten.
17 </p>
```

Quellcode 6.6: Beispielhafter Aufbau einer JSP

6.5. Ablauf der Kommunikation

Nachdem nun das Portal mit seinen Bestandteilen vorgestellt wurde, kann auch die bisher nur in Einzelteilen erwähnte Kommunikation in ihrer Gesamtheit betrachtet werden. Dafür wird beispielhaft der Arbeitsablauf eines Sachbearbeiters untersucht. Die Darstellung erfolgt in einer stark an Sequenzdiagrammen angelehnten, grafischen Form.

6.5.1. Aufruf einer Sachbearbeiterseite

Zunächst ruft der Sachbearbeiter die zu einem Ablauf gehörende Seite auf. Das bedeutet, dass sein Browser einen GET-Request an das Portal schickt. Dieses fordert daraufhin die sich auf der angeforderten Seite befindlichen Portlets über ihre `doView()`-Methoden auf, ihren darzustellenden Inhalt zurückzuliefern. Um das tun zu können, fragt das Posteingangsportlet die offenen Fälle in der Datenbank ab. Die Rückgaben der Portlets werden vom Portal zusammengefasst und an den anfragenden Browser des Sachbearbeiters ausgeliefert. Siehe Abbildung 6.3.

6.5.2. Auswahl eines Vorgangs

Hat sich der Sachbearbeiter für eine Ablaufinstanz entschieden, wählt er sie durch einen Klick auf den zugehörigen Button aus. Der Klick bewirkt, dass der Browser einen POST-Request an das Portal sendet, was dieses in den Aufruf der `processAction()`-Methode des Posteingangs umsetzt. In dieser wiederum wird ein Event zur Übermittlung der Instanz-ID an das Sachbearbeiterportlet ausgelöst, welches diese ID dann in einem Parameter ablegt, um sie in der nächsten `doView()`-Methode greifbar zu haben.
Der nun folgende `doView()`-Aufruf für den Posteingang unterscheidet sich nicht vom Vorherigen. Im Sachbearbeiterportlet jedoch wird der zuvor gesetzte Parameter ausgelesen und per GET-Request an das Orbeon-Formular übermittelt. Dieser lädt den

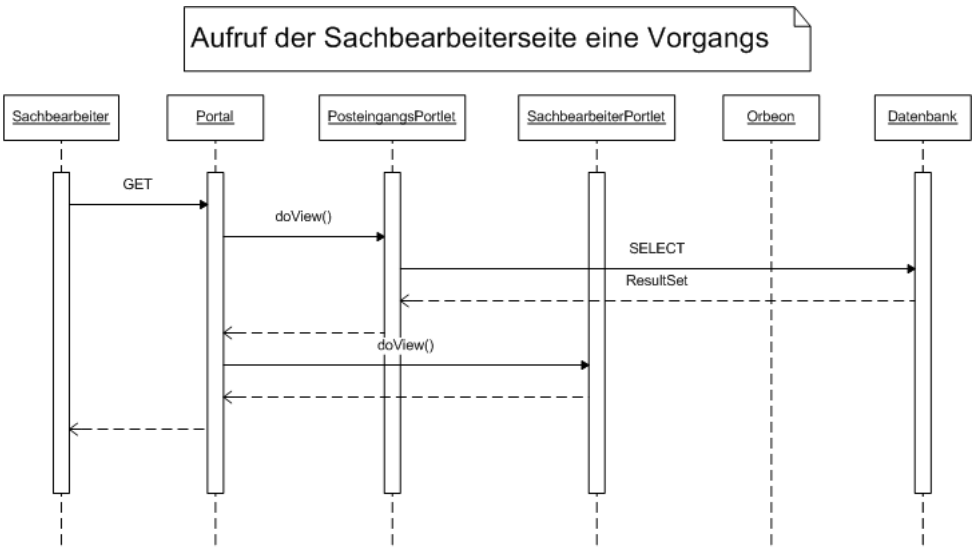


Abbildung 6.3.: Aufruf der Sachbearbeiterseite eines Vorgangs

zugehörigen Datensatz aus der Datenbank, baut ihn in das Formular ein und liefert dieses als Antwort zurück. Das Portal fasst die Fragmente wieder zu einer HTML-Antwort zusammen und schickt diese an den aufrufenden Browser. Zur Veranschaulichung dient 6.4.

6.5.3. Absenden eines Formulars

Nachdem der Sachbearbeiter das Formular ausgefüllt hat, schickt er es ab. Die Daten werden dabei direkt an das Orbeon-Formular übermittelt, welches sie dann in die Datenbank abspeichert. Anschließend sendet das Formular einen POST-Request an das Portal, dieser beinhaltet die Entscheidung des Sachbearbeiters über den nächsten Zustand des Ablaufs, repräsentiert durch eine Zeichenkette. Das Portal setzt diesen Request in einen Aufruf der `processAction()`-Methode des Sachbearbeiterportlets um, dieses aktualisiert daraufhin den Zustand des Ablaufs in der Datenbank. Außerdem wird noch der ID-Parameter zurückgesetzt, damit das Sachbearbeiterportlet das Formular nicht mehr anzeigt.
Der folgende `doView()`-Zyklus entspricht dem vom Aufruf der Sachbearbeiterseite. Dieser Ablauf ist auch in Abbildung 6.5 dargestellt.

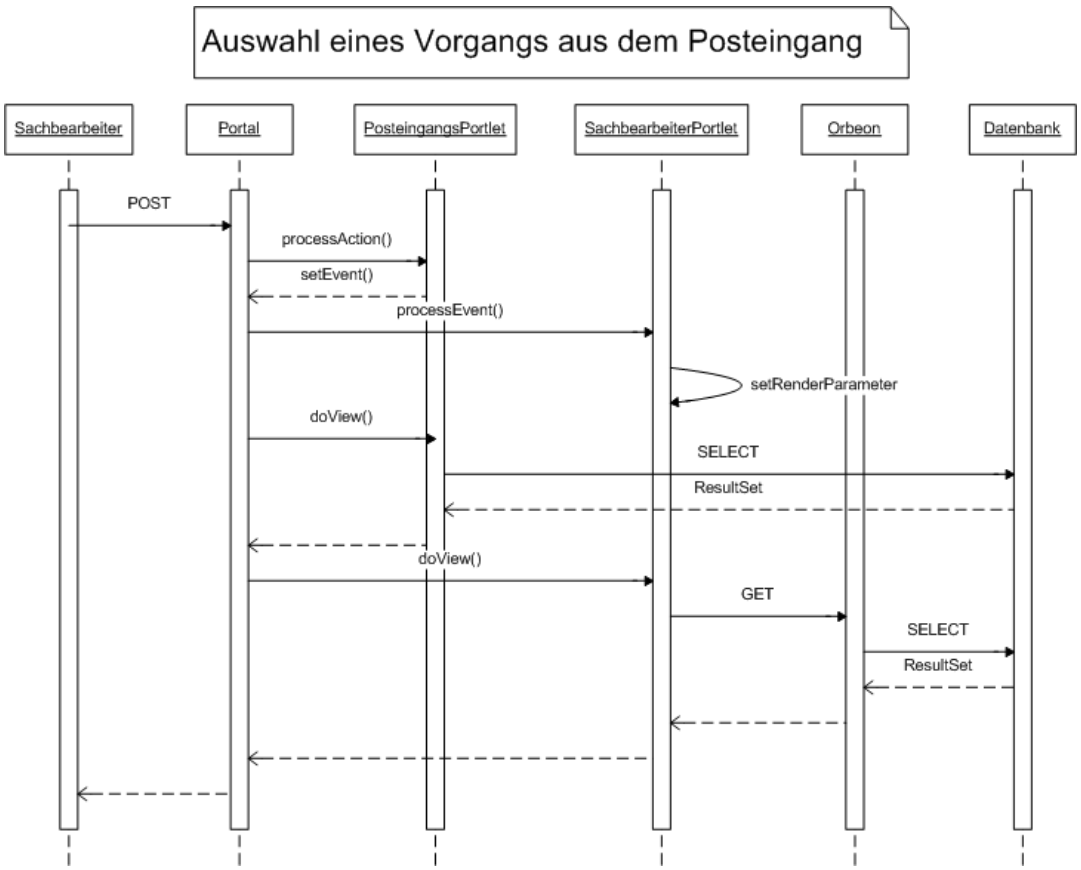


Abbildung 6.4.: Auswahl eines Vorgangs aus dem Posteingang

6.6. Anbindung von Fremdsystemen

Nachdem im vorherigen Abschnitt auf die Portalprototypen eingegangen wurde, wird hier gemäß Systemanforderung auf Portalseite ein zusätzlicher Client eingeführt, welcher zur Kommunikation mit Fremdsystemen dient.
Im Folgenden wird zunächst die Anforderung an die Anbindung von Fremdsystemen kurz vorgestellt. Anschließend bekommt man in Kapitel 6.6.2 einen Überblick über die grobe Architektur und relevante Technologien. Anhand gegebener Architektur erfolgt in den späteren Abschnitten 6.6.3 und 6.6.4 eine detaillierte Beschreibung der einzelnen Komponenten (Serverseitig, Clientseitig).

6.6.1. Anforderungsspezifikation

In manchen Situationen ist es in E-Government-Systemen beim Austausch von Formulardaten zwischen verschieden Rollen (z.B. Bürger bzw. Beamter) notwendig, die Formu-

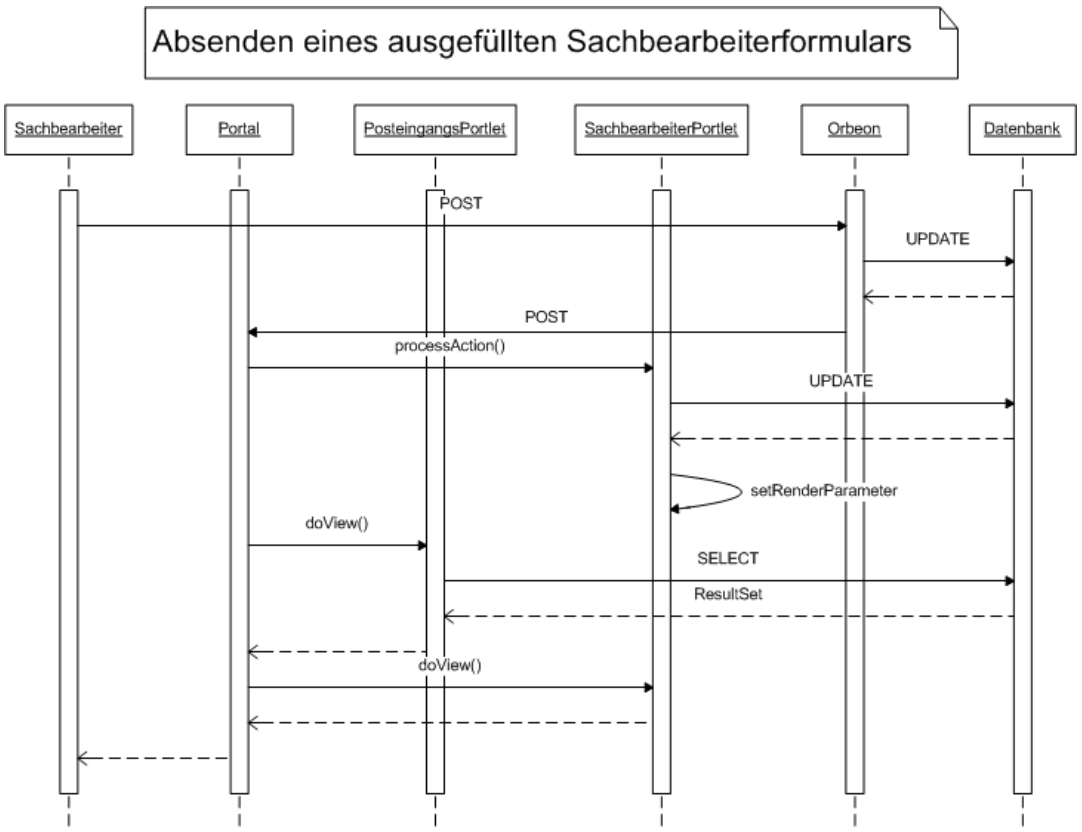


Abbildung 6.5.: Absenden eines ausgefüllten Sachbearbeiterformulars.

lardaten an ein Fremdsystem weiterzuleiten, damit die Daten weiter verarbeitet werden können oder zur künftigen Anwendung einfach in ein spezielles Repository abgelegt werden können.

Hierbei muss man einerseits im Rahmen der PG die Anbindung von Fremdsystem sowohl in das vorhandene Portal integrieren, als auch als ein Schritt in den Ablauf einfügen. Die einzige Operation dabei ist „sendFormDataToExternalService“. Andererseits stellt das Fremdsystem typischerweise zum Teil eine Datenbank bzw. ein Repository zur Speicherung der Daten zur Verfügung, allerdings ist das Fremdsystem nicht Ziel der PG. Um unnötige Komplexität zu vermeiden, braucht man dafür lediglich ein „stub Fremdsystem“ einzurichten, d.h. Speicherung und Verarbeitung der Formulardaten „on the fly“ gewährleisten.

Nach einer kurzen Vorstellung von Anforderung wird im Folgenden der konkrete Entwurf eingehend beschrieben.

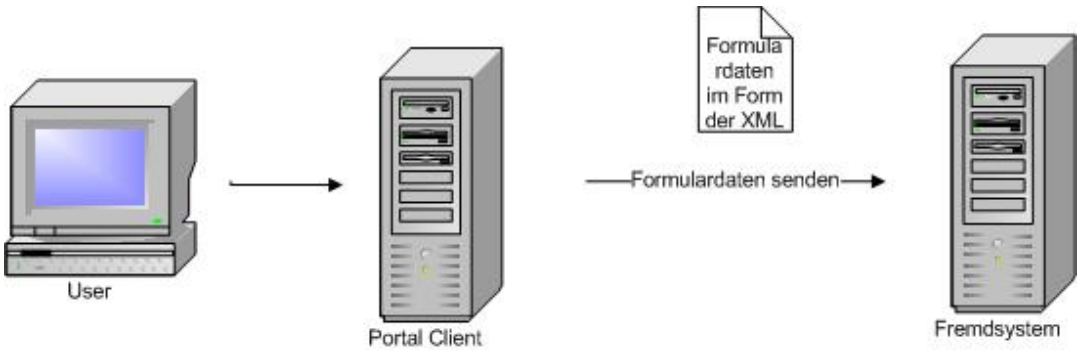


Abbildung 6.6.: Architektur der Anbindung von Fremdsystem

6.6.2. Architektur der Anbindung von Fremdsystem

Abbildung 6.6 illustriert die grobe Architektur der Anbindung des Fremdsystems, die nur aus nur zwei Komponenten besteht:

- Portal Client

  Dieser Teil dient hauptsächlich zum Extrahieren der Formulardaten und weiterhin zur Übermittelung an das Fremdsystem.

- Fremdsystem (stub)

  Wie oben erwähnt, sorgt das Fremdsystem für eine temporäre Speicherung der empfangen Daten und dann für die Ausgabe der entsprechenden Daten auf der Konsole.

Was die Formulardaten angeht, würden diese in Form vom XML repräsentiert werden, da XML für den Austausch von Information weit verbreitet ist.
Für das gegebene Problem gibt es prinzipiell mehrere verschiedene Lösungenen, aber durch diese Systemanforderung liegt es nahe, die Verbindung zum Fremdsystem durch Webservices zu realisieren.
Webservices sind in den letzten Jahren im Bereich der verteilten Systeme als Forschungsthema immer wichtiger geworden. Zudem ist dies normalerweise als eine der Haupttechnologien zur Implementierung des Servicekonzeptes betrachtet worden. Darüber hinaus beinhaltet das Themengebiet Webservices mittlerweile eine Vielzahl verschiedener Unterthemen, sowohl aus theoretischer als auch aus praktischer Sicht. Manchen Themen sind sogar in Entwicklung, beispielsweise die Verbesserung der vorhandenen Standards

und QoS von Web Service, etc. Kurz gesagt, Web Service ist ein umfangreiches Thema in verteilten Systemen.
Ausgehend von der oben erwähnten Anforderung und der ermittelten Architektur, wird jede einzelne Komponente in den folgenden Unterkapiteln näher beschrieben.

6.6.3. Serverseitiger Entwurf des Systems

In diesem Abschnitt geht es vor allem um den serverseitigen Entwurf von Fremdsystemen. Hier liegt das Hauptproblem darin, wie durch Webservices der Stub des Fremdsystems realisiert werden kann.
Wie zuvor angesprochen, sind Webservices ein recht umfangreiches Thema. Um möglichst einfach die Sachen aufzuklären, wird der Abschnitt in einen Theorie und einen Praxisteil unterteilt. Jeder Teilabschnitt wird lediglich den für das Projekt relevanten Bereich vorstellen. Die konkreten Technologien, wie beispielsweise SOAP [50], WSDL [51], UDDI [52], etc., werden nicht näher betrachtet, da dies nicht im Problembereich der Projektgruppe liegt. Schließlich wird anhand der Webservice-Charakteristik ein statisches Modell zur Gestaltung des Fremdsystems erstellt.

Webservices-Theorie auf der Serverseite

Webservices sind als eine Standardtechnologie ist prinzipiell unabhängig von irgendeiner Plattform oder Programmiersprache. Die grundlegende Theorie kann man in [45], [46], [47] finden, in denen viele Hauptstandards, z.B. SOAP, WSDL, UDDI usw. bereits ausführlich erklärt werden.
Hier in der Projektgruppe basiert die Webservice-Kommunikation auf SOAP-Nachrichten, die ebenfalls im XML-Format sind. Daraus ergibt sich die Frage, wie sich die Formulardaten in diese SOAP-Nachrichten verpacken lassen. Dazu führt der SOAP-Standard das SOAP Message Model ein. Das SOAP Message Model ist vor allem definiert durch den „SOAP encoding style“ und „SOAP communication style“.

- SOAP-Kodierungsstil

  SOAP-Kodierungsstil legt die Kodierungsregeln fest, so dass die unterschiedlichen Datenstrukturen im Vorfeld entsprechenden Strings zugewiesen werden.

- SOAP-Kommunikationsstil

  Der SOAP-Standard bietet im Wesentlichen vier Typen an: RPC/literal, Document /literal, RPC/encoded, Document/encoded.

SOAP-Kommunikationsstil ermöglicht die Übermittlung der Formulardaten via SOAP-Nachrichten. Man unterscheidet vor allem zwischen zwei Arten:

- RPC-Stil

  Analog zu RPC, das „Request“ auf der Clientseite verhält sich wie ein Methodenaufruf mit einer Menge von Argumenten. Der Webservice gibt dann das zugehörige „Response“ zurück. Im Allgemeinen sind solche Werte primitive Datentypen. Die SOAP-Nachrichten enthalten nur die Parameterwerte bzw. den Rückgabewert.

- Document-Stil

  Typischerweise betrachtet man den unkodierten XML-Inhalt im SOAP-Nachrichtenrumpf als Document-Stil. D.h. der Client schickt das ganze XML-Dokument, das im SOAP-Nachrichtenrumpf enthaltet ist, sondern nicht die Parameter im SOAP-Nachrichtenrumpf.

Somit kann mit Hilfe des Document-Stil sichergestellt werden, dass die Formulardaten einheitlich übermittelt werden können.

Webservices-Praxis auf der Serverseite

Viele Hersteller haben bereits ihre konkrete Implementierungslösung, die Webservice-API, sowie die entsprechenden Werkzeuge auf den Markt gebracht, wobei APIs wie Axis [53] und JAX-WS [54] etc., die am meisten verwendet werden.
Im Vergleich zu Axis hat JAX-WS die Webservice Entwicklung sehr vereinfacht. Sun hat mit JAX-WS als Nachfolger von JAX-RPC ein neues Webservice-Programmiermodell definiert. Der wichtigste Vorteil ist, dass dieses folgende Features Angebotn hat:

- bessere Plattformunabhängigkeit
- Annotation
- besserer Aufruf von Webservices via sync. oder async.
- Unterstützung der „Top-down“ und „Bottom-up“ Entwicklungsansätze.

Die PG hat sich grundsätzlich für JAX-WS entschieden, jedoch besteht noch ein weiteres Problem bezüglich des „Webservice Runtime Environment“. Nicht alle Applikationsserver stellten das gleiche oder ein kompatibeles „Webservice Runtime Environment“ zur Verfügung, z. B. verfügt der JBoss Applikationsserver, der im Voraus in der PG verwendet

wurde, über sogenannte JBoss-WS [48] als eigene „Webservice Runtime Environment“. Obwohl JBoss-WS zu einem gewissen Grad auf JAX-WS basiert, wurden in JBoss-WS noch neue Features eingefügt und manche Features geändert. Somit müssen bei der Erstellung eines Webservices einige Einstellungen selbst vorgenommen werden, z. B. der Deployment Descriptor.
Weiterhin muss neben den zwei oben angesprochenen Punkten der wichtige Erstellungsansatz gewählt werden. Im Allgemeinen existieren zwei Möglichkeiten: „Top-Down Approach“ und „Bottom-Up Approach“.

- Top-Down Approach

  Dies bedeutet, dass zuerst die WSDL-Datei manuell erstellt werden muss. Dann werden mit Hilfe der WSDL-Datei durch spezifische Werkzeuge automatisch die benötige „skeleton“ Klassen erstellt, die durch die Entwickler noch implementiert werden müssen.

- Bottom-Up Approach

  Im Gegensatz zum „Top-Down Approach“, sind in diesem Fall die konkreten implementierten Klassen bereits im Voraus vom Entwickler erstellt worden. Diese müssen dann nur noch an den richtigen Stellen annotiert werden, damit ebenfalls durch spezifische Werkzeuge die passende WSDL-Datei und ein entsprechender Webservice erstellt werden.

Wegen einem eigenen „Webservice Runtime Enviroment“ stellt JBoss-WS die für die Umwandlung benötigte Werkzeuge zur Verfügung. (siehe [49]) Die konkreten Implementierungen können in [49] [55] gefunden werden.
Hier wurde der „Bottom-Up Approach“ zur Erstellung des Webservices verwendet, damit es die Bau der Webservices viel vereinfacht hat.

Statisches Modell des Systems

Ausgehend von den oben angesprochenen Punkten (Webservice-Theorie 6.6.3 und Praxis 6.6.3) ergibt sich das in Abbildung 6.7 dargestellte statische Modell in Form eines Klassendiagramms. Dort sind die beteiligten Klassen dargestellt, die zur Modellierung verwendet werden.
Hierbei ist die „ExternalService“ Klasse als Webservice aufgebaut, daher kann man alle benötige funktionale Logik realisieren, z.B. „sendToExternalService“, „processFormData“, etc. Zu dieser Klasse muss der Programmierer nur noch die JAX-WS spezifische Annotation an der gewünschten Methode vornehmen.

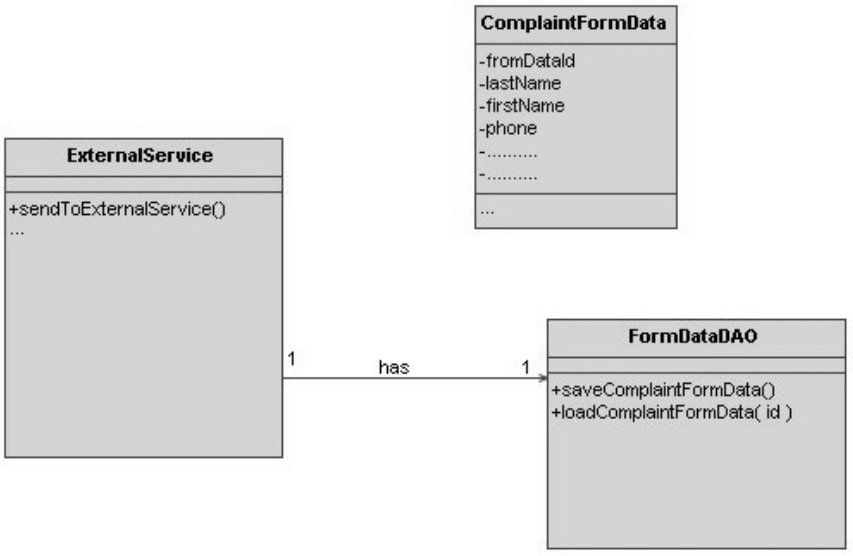


Abbildung 6.7.: Statisches Modell des Fremdsystems

Um Formulardaten im „Document Style“ zu übermitteln, muss hierbei das Objekt „ComplaintFormData“ erstellt werden, das alle Daten des Formulars enthalten sollte. Dieses Objekt wird bei der SOAP-Nachrichten-Übertragung automatisch in ein ganzes XML-Document umgewandelt, welches dann in SOAP-Nachrichtenrumpf hinzugefügt werden kann.
Normalerweise ist es nötig, die empfangenen Daten irgendwo abzulegen, z. B. in einer Datenbank. Dazu bietet die Klasse „FormDataDAO“ Zugriffslogik an, damit dadurch die Persistenzschicht flexibel manipuliert werden kann. Wegen des „stub Fremdsystem“ wird hier die persistente Speicherung durch eine normale Datenstruktur wie beispielsweise „HashMap“ ersetzt.

6.6.4. Clientseitiger Entwurf des Systems

Im Vergleich zur Serverseite ist die Clientseite genauso wichtig. Auf Details zur Theorie und Praxis wird an dieser Stelle ebenfalls eingegangen.

Webservices-Theorie auf der Clientseite

Eines der wichtigsten charakteristischen Merkmale von Webservices ist, dass diese lose gekoppelte Softwaremodule sind. Wegen dieses Merkmals lassen sich viele weitere Webservice-Theorien, Features und entsprechende Technologien herleiten.

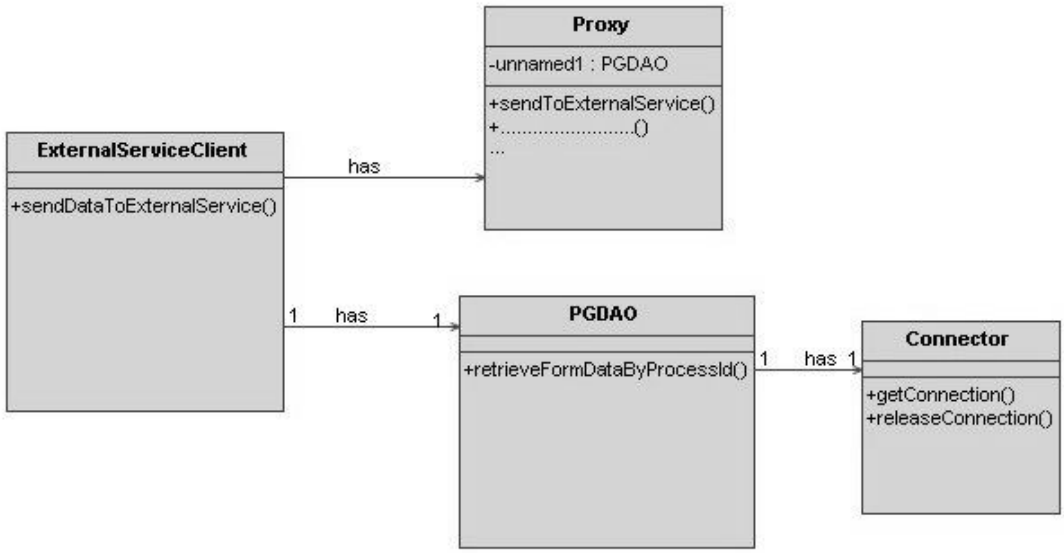


Abbildung 6.8.: Statisches Modell des Clients

Hier in der PG bedeutet es, dass die clientseitige Anbindung durch eine beliebige Plattform, Programmiersprache, und Technologie realisiert werden kann, wobei es nicht notwendig ist, herauszufinden, welche konkrete Implementierung sich auf der Serverseite befindet. Die einzige benötigte Information ist nur das Service-Interface.

Webservices-Praxis auf der Clientseite

Demnach fiel die Entscheidung auf eine unabhängige clientseitige Applikation „Standalone application“ mit Hilfe von JAX-WS. Wie oben beschrieben, ermöglicht JAX-WS die Erzeugung von Webservices. Dabei unterstützt es auch die clientseitige Erzeugung. Mittels der JAX-WS zugehörigen Werkzeuge werden alle dazu benötigten Interfaces, abstrakten Klassen und Klassen, die man „Proxy“ nennt erstellt. Anhand dieses Proxys ruft die Clientseite die Webservices, als wäre auf Clientseite eine lokale Methode aufgerufen worden. (konkrete Implementierung: siehe [55])

Abgesehen davon, obwohl JAX-WS ein neues Feature in JDK 1.6 ist, ist es eigentlich ein separates Projekt, das man mit der JDK 1.5 in der PG durch geeignete Einstellungen ebenfalls direkt verwenden kann.

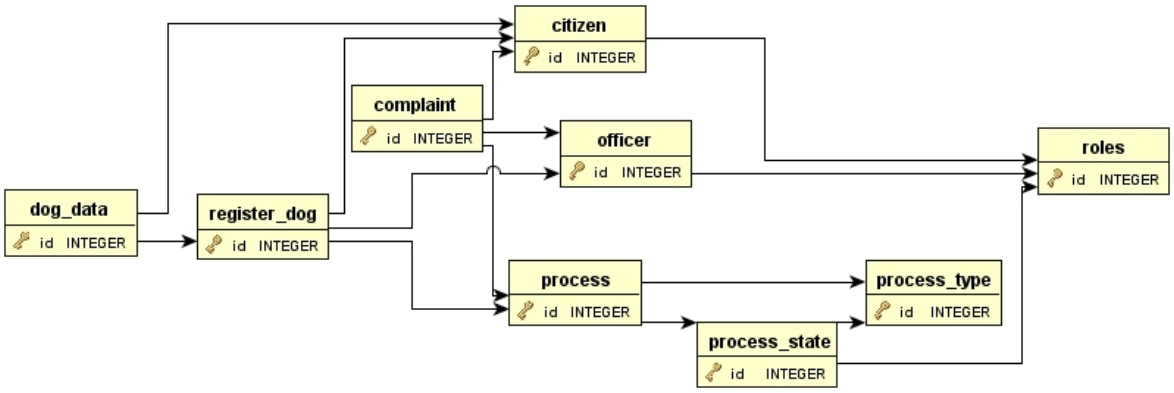


Abbildung 6.9.: Übersicht über das Datenbankschema

Statisches Model der Clientseite

Wie die Abbildung 6.8 veranschaulicht, bewirkt „ExternalServiceClient“, dass die Clientseite den Webservice aufrufen kann. Hierfür sind die Methoden des Proxys entscheidend. Der Proxy repräsentiert eigentlich nicht nur eine Klasse, sondern mehrere Interfaces und Klassen, die sich um die zugrunde liegende Aufgabe gekümmert haben, beispielsweise die Kodierung der Datentypen, Analyse des Service-Interfaces, XML-Binding, etc. Aber hier ist die Hauptsache, dass der Proxy die Operation des Webservices angeboten hat.
Neben dem Aufruf des Webservices muss die Clientseite noch auf die Datenbank zugreifen, um die Formulardaten zu extrahieren. Hierbei wird auch eine „PGDAO“ Klasse erstellt, die die notwendige Datenbank-Zugriffslogik umfasst. Unter Koordination der „Connector“-Klasse ist der Client durch JDBC an die Datenbank angeschlossen.

6.7. Datenbankschema

Für das Ausführen und Verarbeiten langlebiger und asynchroner Prozesse, wie die von uns beschriebenen Abläufe, müssen Daten gespeichert werden. Um die persistente Speicherung der Daten zu realisieren, haben wir uns für ein Datenbanksystem entschieden. Bei der Auswahl gab es hierfür keine Präferenzen. Wir wählten PostgreSQL [32]. PostgreSQL ist ein Open-Source Datenbank Management System, welches im Gegensatz zu MySQL [33] auch Fremdschlüsselbedingungen korrekt umsetzt.
Das Datenbankschema musste zusätzlich zu den Nutzdaten der Formulare und Abläufe auch weitere Metadaten enthalten. Diese Metadaten sollen eine konsistente Wiederaufnahme der Abläufe nach einem Systemneustart oder sogar eines Systemausfalls gewährleisten. So entstand bereits für unsere beiden einfachen Beispielabläufe ein recht

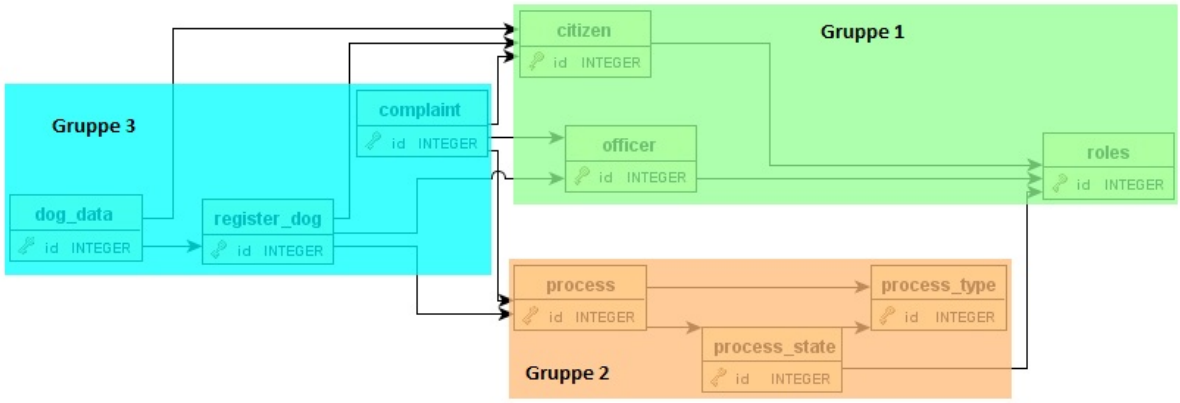


Abbildung 6.10.: Einteilung der Tabellen in Gruppen

komplexes Datenbankschema, dass im folgenden genauer beschrieben und erklärt wird.

6.7.1. Gesamtüberblick

Abbildung 6.9 gibt eine grobe Übersicht über die vorhandenen Tabellen und ihrer Beziehungen. Die Tabellen können wie in Abbildung 6.10 in drei Gruppen eingeteilt werden: Benutzertabellen, Ablauftabellen und Formulartabellen.
Die erste Gruppe ist die der Benutzertabellen, die Daten für die Benutzerverwaltung speichern. **citizen** und **officer** speichern die persönlichen Daten für Bürger und Beamte. Zusätzlich gibt es die Tabelle **roles** für die verschiedenen Rollen. Hier werden die Rollen aus dem Portal abgebildet.
Die zweite Gruppe sind die Ablauftabellen, in denen Daten der Ablaufinstanzen gespeichert werden. Die Tabelle **process** spielt in dieser Gruppe die zentrale Rolle, da in ihr die Daten aller gestarteten, abgeschlossenen und abgebrochenen Ablaufinstanzen gespeichert werden. Die Tabellen **process_type** und **process_state** beinhalten die existierenden Ablauftypen und Ablaufzustände.
Als letzte Gruppe gibt es die Formulartabellen, die die Nutzdaten der Formulare und somit der einzelnen Ablaufinstanzen speichern. Dazu gehören **complaint** für den Beschwerde-Ablauf und **register_dog** und **dog_data** für den Hund-Anmelden-Ablauf.

6.7.2. Benutzertabellen

Zu den Benutzertabellen gehören **citizen** für die Speicherung der Bürgerdaten, **officer** für die Speicherung der Beamtendaten und **roles** für die Speicherung der Rollen. Abbil-

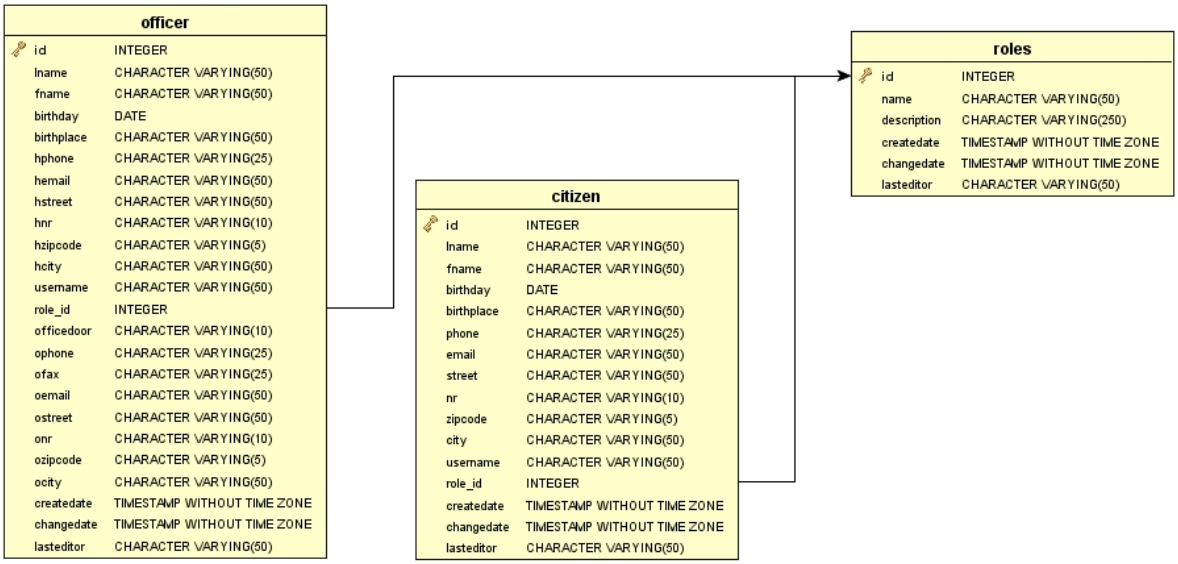


Abbildung 6.11.: Die Benutzer in der Datenbank

dung 6.11 zeigt eine detailierte Ansicht der Tabellen und ihrer Abhängigkeiten.
Zunächst fällt auf, dass **officer** wesentlich mehr Spalten enthält als **citizen**. Das liegt daran, dass wir uns entschieden haben, zusätzlich zu den persönlichen Informationen auch Daten zu dem Büro des Beamten mit in diese Tabelle aufzunehmen. An die Stelle einer Beamtentabelle kann natürlich auch eine Mitarbeitertabelle eingesetzt werden. Die Spalten können dann von Unternehmen zu Unternehmen variieren. Für die Bürgertabelle gilt das gleiche. Hier könnte eine mögliche Kundentabelle angelegt werden, die die Daten der Kunden des Unternehmens enthält. Die entscheidende Spalte ist **roles_id**. Sowohl **citizen** als auch **officer** haben über diese Spalte eine Fremdschlüsselbeziehung zu **roles**. Dies ist leicht nachvollziehbar, da sowohl Bürger als auch Beamte eine ganz spezielle Rolle einnehmen. In unseren Beispielen und in der E-Government Domäne wird dies für die Bürger wahrscheinlich nur die Rolle eines **Bürgers** sein. Ausgeweitet auf ein Unternehmensportal sind hier allerdings mehr Rollen, wie **Lieferanten** oder **Händler**, denkbar.
Es wurde entschieden, die Zuweisung der Beamten in verschiedene Abteilungen nicht über eine zusätzliche Tabelle, sondern mit Hilfe der bereits existierenden Tabelle **roles** zu realisieren. So müssten gleiche Stellungen in verschiedenen Abteilungen als zwei unterschiedliche Rollen angelegt werden. Für den Abteilungsleiter der Abteilung 1 würde eine Rolle **Abteilungsleiter_Abteilung1** und für den Abteilungsleiter der Abteilung 2 würde eine Rolle **Abteilungsleiter_Abteilung2** angelegt werden. Das Portal ist die

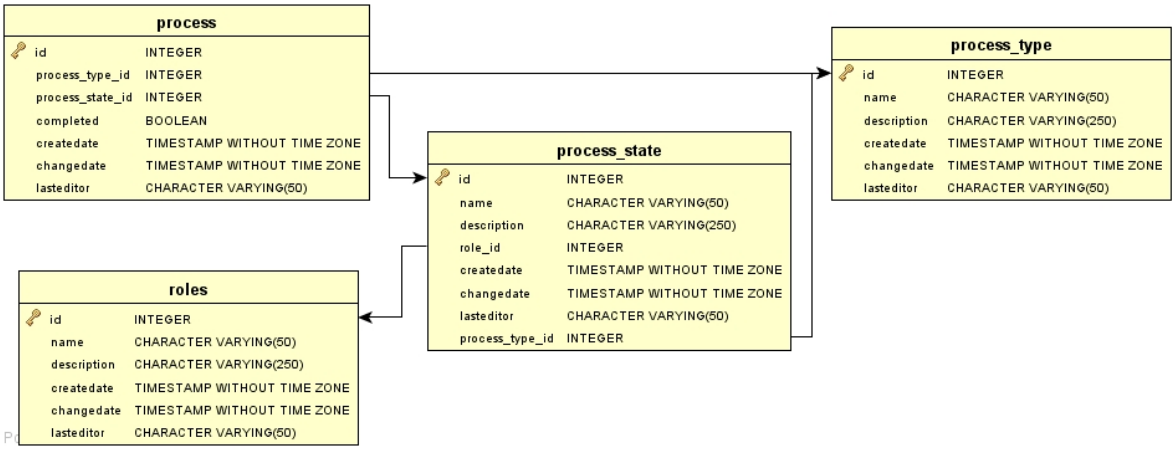


Abbildung 6.12.: Aufbau der Abläufe in der Datenbank

zentrale Stelle, an der die Benutzer und Rollen verwaltet werden. Die Tabelle **roles** soll diese Einstellungen nur abbilden. Dadurch ist sie eng an die Rollenverwaltung des Portalservers gekoppelt.

6.7.3. Ablauftabellen

Zu den Ablauftabellen gehören **process_type** für die Speicherung der Ablauftypen, **process_state** für die Speicherung der Ablaufzustände und **process** für die Speicherung der Informationen zu den Ablaufinstanzen. Grafik 6.12 zeigt eine detailierte Ansicht der Tabellen und ihrer Abhängigkeiten.
Die Tabelle **process_type** ist eng an die existierenden Abläufe gekoppelt. In dieser Tabelle werden eine Bezeichnung und eine Beschreibung der Abläufe gespeichert.
Auch in der Tabelle **process_state** werden Bezeichung und Beschreibung der einzelnen Ablaufzustände gespeichert. Zusätzlich gibt es allerdings noch eine Fremdschlüsselbeziehung zu **process_type** und **roles**. Als Ablaufzustände haben wir die einzelnen Knotenpunkte der Ablaufdatei verstanden. Dazu gehören die Schritte und die Start- und Endzustände. In einem Ablauf wie er in Abbildung 3.10 zu sehen ist, gäbe es somit 5 Zustände. Damit wir in der Tabelle nachsehen können, welche Rolle den aktuellen Schritt bearbeiten darf, benötigen wir die Referenz auf die Rollentabelle. In dieser Tabelle werden alle Zustände aller Abläufe gesammelt. Um die Zustände auch nach Ablauftyp filtern zu können, haben wir einen Verweis auf den zugehörigen Ablauftyp hinzugefügt. In der Tabelle **process** wird nun für jede gestartete Ablaufinstanz ein Eintrag mit einer

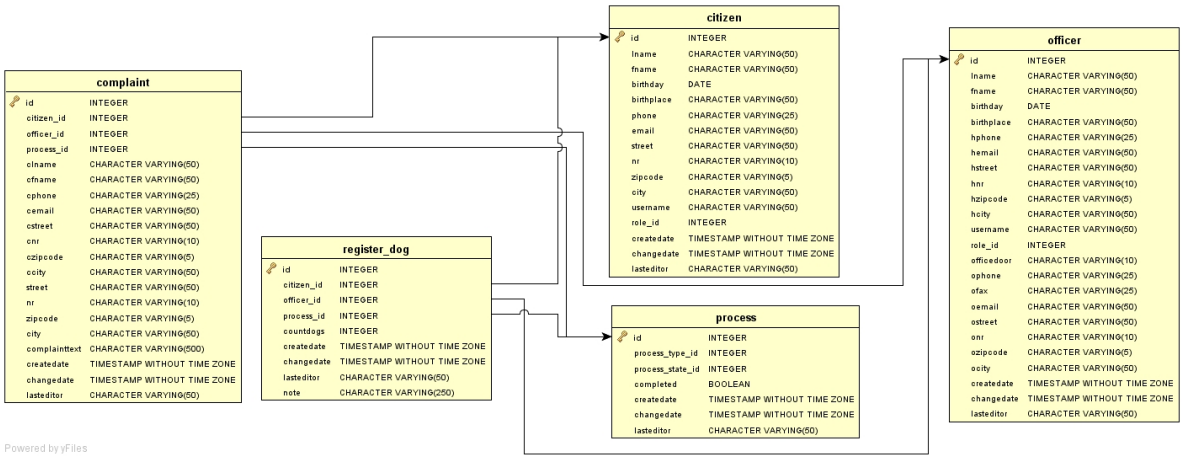


Abbildung 6.13.: Die fachlichen Abläufe in der Datenbank

Referenz auf den Ablauftyp und den aktuellen Ablaufzustand angelegt. So können wir für jede Ablaufinstanz herausfinden von welchem Typ die Instanz ist und in welchem Zustand sie sich momentan befindet. Damit zwischen erfolgreich durchgelaufenen und abgebrochenen Abläufen unterschieden werden kann, gibt es eine Spalte **completed**, in der der Boolean-Wert auf **false** gesetzt ist, falls der Ablauf noch aktiv ist, oder auf **true** gesetzt ist, falls der Prozess nicht mehr aktiv ist. Eine erfolgreiche Ablaufinstanz würde somit das „Completed-Flag“ auf **true** gesetzt und eine Referenz auf einen als Endzustand beschriebenen Zustand haben.
In der **process**-Tabelle werden keine Referenzen auf Formulartabellen gesetzt. Dies ist so beabsichtigt, da nicht vorhersehbar ist, wieviele verschiedene Formulare für einen Ablauf benötigt werden können. Damit nun nicht alle Formulardaten in eine einzige Tabelle gepresst werden müssen, wurde das Datenbankschema so aufgebaut, dass stattdessen in den Formulartabellen eine Referenz auf eine Ablaufinstanz gesetzt wird. Eine Formularinstanz kann offensichtlich nur zu einer einzigen Ablaufinstanz gehören.
Da diese Daten nicht flüchtig gehalten werden, sondern bei der Bearbeitung ausgelesen und erst beim Auslösen der entsprechenden Knöpfe geschrieben werden, sind die Daten vor Systemausfällen und Neustarts gesichert.

6.7.4. Formulartabellen

Grafik 6.13 zeigt die wichtigen Beziehungen der Formulartabellen. Zusätzlich zu den Nutzdaten der Formulare sind Referenzen auf einen Bürger, einen Beamten und eine

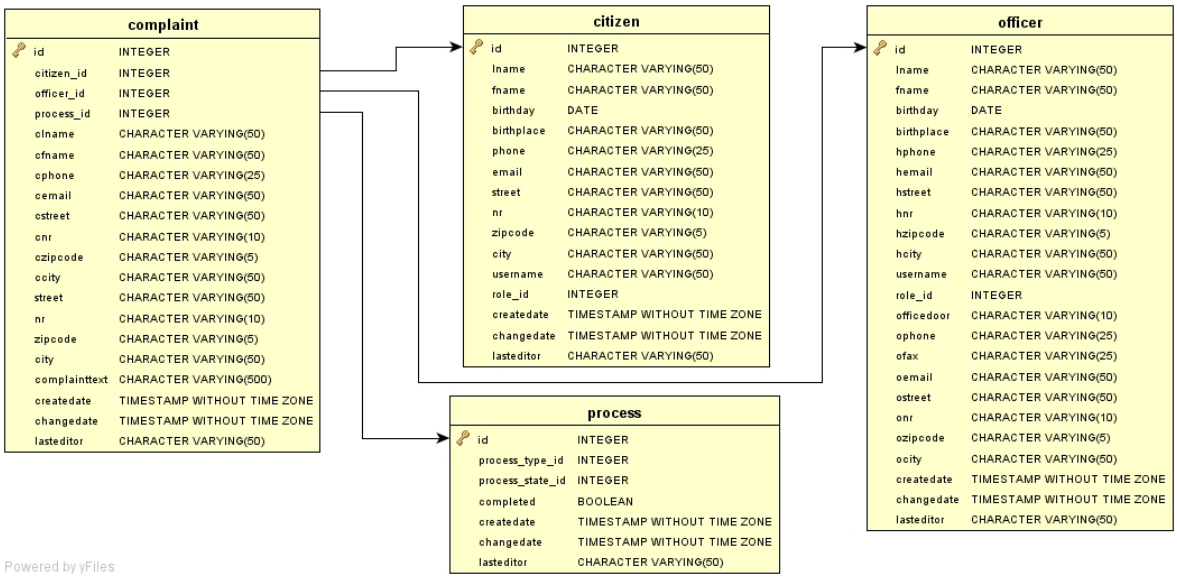


Abbildung 6.14.: Die Beschwerde in der Datenbank

Ablaufinstanz gesetzt. Der referenzierte Bürger ist der Antragsteller. Der referenzierte Beamte ist derjenige, der den Antrag entgegennimmt, also initial den Antrag als Beamter bearbeitet. Natürlich kommt es vor, dass ein Antrag durch mehrere Hände geht. Dazu gibt es eine Spalte **lasteditor**. Hier wird der Benutzer festgehalten, der diesen Antrag zuletzt bearbeitet hat. Über die Referenz auf die Ablaufinstanz kann festgestellt werden, zu welchem Ablauf das Formular gehört und somit der Ablauftyp und der aktuelle Zustand.

Beschwerde

Die Tabelle des Beschwerdeformulars (siehe Bild 6.14) enthält für jedes in unserem fachlichem Beschwerdeformular entworfenen Feld eine Spalte. Weiter Informationen waren in diesem Ablauf nicht nötig. Eine Referenz auf einen Bürger hätte statt der Antragstellerinformationen in unserem Beispiel nicht ausgereicht. Der Beschwerdeablauf soll auch anonym bzw. von nicht im Portal registrierten Benutzern ausgeführt werden können. In diesen Fällen gibt es keinen Benutzer im Datenbanksystem, auf den referenziert werden kann. Um die Fremdschlüsselbeziehung nicht zu verletzen wird in einem solchen Fall eine Referenz auf einen „Dummy“-Benutzer gesetzt. Dennoch sollte es möglich sein, Kontaktinformationen und Informationen zur eigenen Person anzugeben. Diese werden dann nur in der Formulartabelle gespeichert.
Anders als im Beschwerdeablauf genügt für den Ablauf **Hund anmelden** eine Referenz

auf einen bestehenden Bürger (siehe hierzu Bild A.7 im Anhang), da nur registrierte Benutzer über das Portal Hunde anmelden dürfen. Zusätzlich zu der Referenz auf den Bürger und die Daten der Hunde wird in dieser Tabelle ein Kommentar des Beamten gespeichert. Je nachdem in welchem Schritt der Ablauf sich befindet und welche Rolle den Schritt bearbeiten darf, müssen im Formular bestimmte Felder editierbar oder nicht editierbar sein. Dies kann natürlich nicht im Datenbanksystem modelliert werden, sondern muss bei der Erstellung der Formulare geschehen. Weiter Informationen zur Erstellung der elektronischen Formulare sind im Kapitel 5 zu finden.
Außerdem haben wir die Formulardaten in zwei Tabellen aufgeteilt. Da es dem Benutzer möglich sein sollte, beliebig viele Hunde anzumelden, war es nötig eine separate Tabelle mit den Hundedaten zu benutzen. Die Formulartabelle enthält als Nutzdaten nur noch eine Referenz auf einen Bürger, einen Beamten, eine Ablaufinstanz und die Anmerkungen eines Beamten. Die Tabelle **dog_data** enthält neben den Daten des Hundes eine Referenz auf einen Bürger, den Besitzer, und auf eine Formularinstanz, das entsprechende Anmeldeformular.

7 Portalgenerierung

Das folgende Kapitel beschäftigt sich im weitesten Sinne mit der Generierung des Portals. Hierzu gehören zusätzlich zu der Generierung der Portaldateien mit einer Template Engine (Velocity) noch die (grafische) Erstellung der Abläufe, ebenso wie das Analysieren der daraus erstellten Ablaufdatei. Schlussendlich wird auf das automatisierte Kompilieren und Deployen des Portals eingegangen.

7.1. Technologien für die Generierung

Der nachfolgende Abschnitt beschreibt die Vorgehensweise der Projektgruppe bei der Generierung des Portals. Hierfür wurden in einem ersten Schritt verschiedene Technologien evaluiert. Dabei fiel die Entscheidung auf eine templatebasierte Generierung mit Velocity. Nach einer kurzen Einarbeitungsphase wurden Templates entwickelt, mit deren Hilfe dann die zuvor manuell erstellten Portaldateien generiert werden konnten. Abschließend wurde mit Hilfe von ANT[1], einem Hilfswerkzeug mit dessen Unterstützung das automatische Kompilieren der Quellen und Deployen auf dem Portalserver erreicht wurde, ein automatisierter Ablauf erzeugt, der die manuellen Handgriffe zur Laufzeit auf ein Minimum reduziert.

7.1.1. Evaluierungsphase

In der Evaluierungsphase standen prinzipiell zwei Werkzeuge zur Auswahl. Auf der einen Seite Velocity[2]. Velocity wurde von der „Apache Software Foundation“ entwickelt und steht unter der „Apache Software License“. Es handelt sich dabei um eine Template Engine, mit deren Hilfe Lückentexte (die Templates) mit Inhalt gefüllt werden können.
Auf der anderen Seite wurde XSLT[3], eine vom „World Wide Web Consortium“ entwickelte Stylesheetprozessorsprache, in die nähere Betrachtung einbezogen. XSLT benutzt einen anderen Ansatz als Velocity. Während Velocity auf Templates operiert und diese zur Laufzeit wie einen Lückentext ausfülllt, transformiert XSLT ein eingegebenes XML Dokument nach vordefinierten Regeln in ein neues XML Dokument. Da es sich bei XSLT

[1] http://ant.apache.org/
[2] http://velocity.apache.org/
[3] www.w3.org/TR/xslt20/

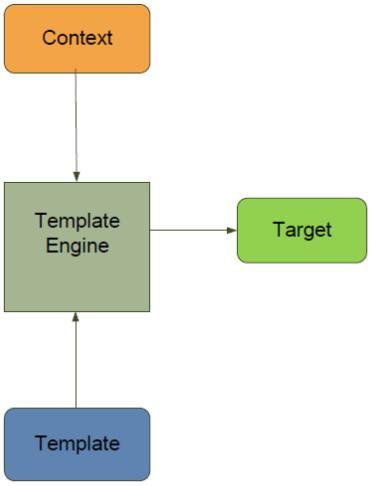


Abbildung 7.1.: Funktionsweise von Velocity

um eine Sprache zur Transformation von XML Dokumenten handelt und unsere Ablaufdatei in XPDL kodiert ist, hatte sich diese Lösung angeboten. XPDL selber wird durch XML dargestellt, so dass davon auszugehen war, dass XSLT direkt auf den XPDL Dateien arbeiten kann.
In der Evaluierungsphase hat sich gezeigt, dass XSLT für unsere Zwecke ein mächtiges Werkzeug darstellt. Diese Mächtigkeit bringt allerdings eine erheblich kompliziertere Einarbeitungsphase mit sich. Da der Faktor Zeit aber eines der ausschlaggebenden Kriterien in unserem Entscheidungsprozess darstellte, war XSLT für uns nicht attraktiv genug.
Die Entscheidung für Velocity fiel also primär wegen der besseren Zugänglichkeit und es hat sich im Laufe des Projekts herausgestellt, dass diese Entscheidung sinnvoll war. Nicht nur, dass durch die sehr flache Lernkurve die Einarbeitung dahingehend vereinfacht wurde, dass die Velocity Gruppe im Laufe des Semesters dynamisch durch neue Mitglieder der Projektgruppe verstärkt werden konnte. Als zweiter großer Pluspunkt ergab sich die leichte Wartbarkeit bzw. Erweiterbarkeit. Durch die ständigen Änderungen der Anforderungen an die Templates und die damit verbundene Umgestaltungen dieser, erwies sich die leicht zu verstehende Velocity Template Language als vorteilhaft XSLT.

7.1.2. Einführung in Velocity

Damit die Vorstellung der Templates im übernächsten Unterkapitel vereinfacht wird, soll in diesem Kapitel Velocity und im Besonderen die Velocity Template Language

vorgestellt werden. Die hier gegebene Übersicht ist nur von sehr grundlegender Natur und ist nur als Hilfestellung für die Darstellung der erzeugten Templates zu verstehen. Velocity wurde von der Apache Software Foundation entwickelt und steht somit unter einer freien Lizenz. Es wurde komplett in Java entwickelt und benutzt auch zum großen Teil dessen Syntax, was die oben beschriebene Vereinfachung der Einarbeitung mit sich bringt.

Funktionsweise

Velocity wurde entwickelt um eine strikte Trennung zwischen Geschäftslogik und Darstellung zu erzielen. Die Designer und Entwickler müssen nicht an denselben Dateien arbeiten. Sie müssen sich lediglich über die Schnittstelle im Klaren sein. Die Template-Sprache von Velocity beinhaltet mächtige, aber leicht verwendbare Script-Sprachen-Elemente. Der Einsatz von Velocity sollte produktiv sein, da es nämlich eine klare und einfache Syntax für den Template-Designer zu Verfügung stellt. Weiterhin bietet es ein einfaches Programmier-Modell für den Entwickler an. Es kann unabhängig entwickelt und gewartet werden, da Templates und Code separat gehalten werden. In eine Java-Anwendung lässt sich die Velocity Template Engine integrieren. Velocity wird in verschiedenen Bereichen eingesetzt, unter anderem in der Java und SQL Code Erzeugung, aber auch in der XML Verarbeitung/Transformation. Die von Velocity verwendeten Applikationen bestehen aus dem Template (Vorlage) und dem dazugehörigen Java-Programm. Es werden immer die gleichen Schritte beim Ausführen von Velocity erledigt. Zunächst wird Velocity initialisiert. Ein Context Objekt wird erzeugt, welchem dann Daten hinzugefügt werden. Danach wird ein Template ausgewählt und zuletzt entsteht der Output durch das Zusammenführen (merge) des Templates mit den Daten. Die verschiedenen Objekte, die der Progammierer im Context platziert, werden über Template-Elemente dem Designer zugänglich gemacht. Das Context-Konzept, welches auf einem Container basiert, kann zwischen verschiedenen Schichten Daten transferieren. Sie ist eine Förderanlage von Daten zwischen dem Java-Layer (Programmierer) und dem Template-Layer (Designer).
Die Abbildung 7.1 zeigt die vier Komponenten einer Template-basierten Transformation mit Velocity.
Im Einzelnen sind dies:

- Template: Stellt den Lückentext dar, welcher ausgefüllt werden soll
- Contex: Enthält die Daten zum Füllen der Templates

- Template-Engine: bekommt als Eingabe den Context und die Templates und ist dann für das Starten des Generierungsprozesses verantwortlich
- Target: Endergebnis des Transformationsprozess.

Velocity Template Language

Im einfachsten Fall erzeugt Velocity im Zusammenspiel mit den entsprechenden Java Anweisungen eine Datei, deren Inhalt bereits vor der Generierung feststeht.

```
1 VelocityEngine ve = new VelocityEngine();
2 ve.init();
3 Template t = ve.getTemplate("velocityTemplateeinfach.txt");
4 VelocityContext context = new VelocityContext();
5 BufferedWriter writer = new BufferedWriter(new FileWriter("outputDatei.txt"));
6 t.merge(context, writer);
7 writer.flush();
8 writer.close();
```

Quellcode 7.1: Ein Velocity Aufruf

Für diese Art der Generierung ist der in 7.1 dargestellte Java Code nötig. Nachdem ein neues Objekt vom Typ VelocityEngine erzeugt wird, wird diese mit dem Befehl `init` initialisiert. Über die Methode `getTemplate` wird das Template übergeben, welches für die Generierung benutzt werden soll. Als nächstes wird der Context initialisiert und ein Objekt vom Typ `BufferedWriter` erzeugt. Der Context ist in diesem Fall leer, da das Template komplett statisch ist und sowieso keine Änderungen zulässt. Das erzeugte Objekt `writer` nimmt dann in Zeile 6 den generierten Text auf und dieser wird in den letzten beiden Zeilen in die Datei „outputDatei.txt"geschrieben.

```
1 Dies ist ein einfaches Velocity Template. Da in diesem keine Variablen
2 oder Kontrollstrukturen mit Hilfe der VTL enthalten sind, ist das
3 Ergebnis der generierung sehr schlicht.
```

Quellcode 7.2: Ein Velocity Template

Ein entsprechendes Template ist eine einfache Datei mit dem in 7.2 dargestellten Inhalt. Die generierte Datei ist dann vom Inhalt her identisch mit dem Template.
Das Beispiel zeigt also, dass die Generierung im einfachsten Fall statischen Text in einer Datei erzeugen kann. Dieser Anwendungsfall ist natürlich auf der einen Seite sehr trivial und auf der anderen Seite wird sich wohl kein Anwendungsfall finden lassen, bei dem eine Datei dauernd mit demselben Inhalt neu generiert werden muss. Um die Stärken von Velocity zu nutzen wird die Velocity Template Language (kurz: VTL) benutzt. Mit

dieser Sprache kann die Generierung über Kontrollstrukturen und Variablen gesteuert werden. Grundsätzlich gilt, dass alle VTL Elemente in Velocity mit dem Symbol # eingeleitet werden müssen.

```
1 #set ($prcName= "beliebiger Text")
```

Quellcode 7.3: Einfache Variablendeklaration

Eine Variablendeklaration sieht beispielsweise wie in 7.3 aus: Hiermit wird eine Variable mit dem Namen `variable` erzeugt, die initial den String `beliebiger Text` speichert. Grundsätzlichen muss der Typ einer Variablen nicht definiert werden. Velocity ist an dieser Stelle ziemlich flexibel und interpretiert den Inhalt der Variablen immer als der Typ, der momentan benötigt wird. Dies bedeutet, wird einer Variablen der Wert 3 zugewiesen, dann kann dieser Wert als String zur Ausgabe in der generierten Datei benutzt werden, kann aber auch als Element in einer Zählschleife benutzt werden (also als tatsächliche Zahl interpretiert werden). Diese Flexibilität zieht sich durch das gesamte Konzept der VTL, bringt aber auch einige Probleme mit sich auf die später noch genauer eingegangen wird. Auf die Variable wird mit Hilfe des Zeichens $ zugegriffen werden. Auf die Variable `variable` aus dem Beispiel kann also mit `\$variable` zugegriffen werden. Hierbei gilt, dass zur besseren Lesbarkeit geschweifte Klammern um den Variablennamen gesetzt werden können.
Variablen müssen nicht erst im Template definiert werden, sondern können auch schon vor dem Aufruf in der entsprechenden Java Klasse definiert werden. Dies ist die einfachste Methode einen Context aufzubauen. Nachdem das Context Objekt vor dem Aufruf von Velocity definiert wird, können diesem Objekt dann verschiedene Variablendefinitionen übergeben werden.

```
1 VelocityEngine ve = new VelocityEngine();
2 ve.init();
3 Template t = ve.getTemplate("velocityTemplateeinfach.txt");
4 VelocityContext context = new VelocityContext();
5 context.put("variable", "ein einfacher Text");
6 BufferedWriter writer = new BufferedWriter(new FileWriter("outputDatei.txt"));
7 t.merge(context, writer);
8 writer.flush();
9 writer.close();
```

Quellcode 7.4: einfacher Velocity Context

In Listing 7.4 wird zusätzlich zu den bereits erklärten Elementen in Zeile 5 der Context um die Variable `variable` erweitert. Diese kann nun im Template benutzt werden. Mit

diesem Mechanismus ist es einfach möglich Informationen an das Template zu übergeben. Beispielsweise können in dem zugrunde liegenden Java Programm Informationen abgefragt werden (Vorname, Nachname), die dann wie bei einem Lückentext entsprechende Stellen im Template füllen. Eine Anwendung ist beispielsweise ein Serienbrief, bei dem sich immer nur der Empfänger ändert.
Weiterhin ist es in Velocity möglich über den Context nicht nur primitive Datentypen zu übergeben, sondern komplette Objekte. Somit ist es möglich aus dem Template heraus auf Methoden des Objekts zurückzugreifen und die Rückgabewerte einzubauen. Die Zuweisung in der Java Klasse ist identisch mit der Zuweisung eines primitiven Datentyps. Die Methoden werden mit `\$objektVariable.Methode(Übergabeparameter)` im Template aufgerufen. Der Rückgabewert dieser Methode kann dann direkt im Template ausgewertet werden. Neben der Deklaration von Variablen biete die VTL noch die Benutzung der meisten bekannten Kontrollstrukturen. Dazu gehören zum Beispiel `if-then-else`, `forEach` und `while`.
Hierbei ist es wichtig, dass jeder Block mit dem Wort `\#end` abgeschlossen werden muss. Die drei genannten Konzepte funktionieren prinzipiell genauso wie in Java, so dass diese an dieser Stelle nicht weiter erklärt werden müssen. Allerdings ist die Benutzung von `forEach` in der VTL besonders interessant. Mit diesem Konstrukt kann über alle Elemente in einer Liste iteriert werden.

```
1 #foreach ($nachname in $Vorfall.Name())
2 Text
3 Anweisung
4 Text
5 Text
6 #end
```

Quellcode 7.5: For Each Kontrollstruktur

Im Schleifenrumpf in Listing 7.5 kann jetzt direkt die Variable `nachname` benutzt werden, die bei jeder Iteration den Wert des nächsten Elements in der Liste annimmt. Der Inhalt dieser Variablen wird durch die Rückgabe der Methode `vorfall.Name()` bestimmt, welche beispielsweise eine Liste von Namen zurückgibt. Hier zeigt sich erneut die Flexibilität der VTL. Prinzipiell ist es egal, ob es sich bei der Rückgabe um ein Array, eine verkettete Liste, eine doppelte verkettete Liste, eine ArrayList, ... handelt. Velocity versucht jeweils zu erkennen, um was für einen Typ es sich handelt und stellt den Inhalt zur Verfügung. Dies sorgt wieder dafür, dass die Templates viel besser zu lesen sind und die Kontrollstrukturen einheitlich operieren können.
Natürlich können die Kontrollstrukturen auch beliebig ineinander geschachtelt werden,

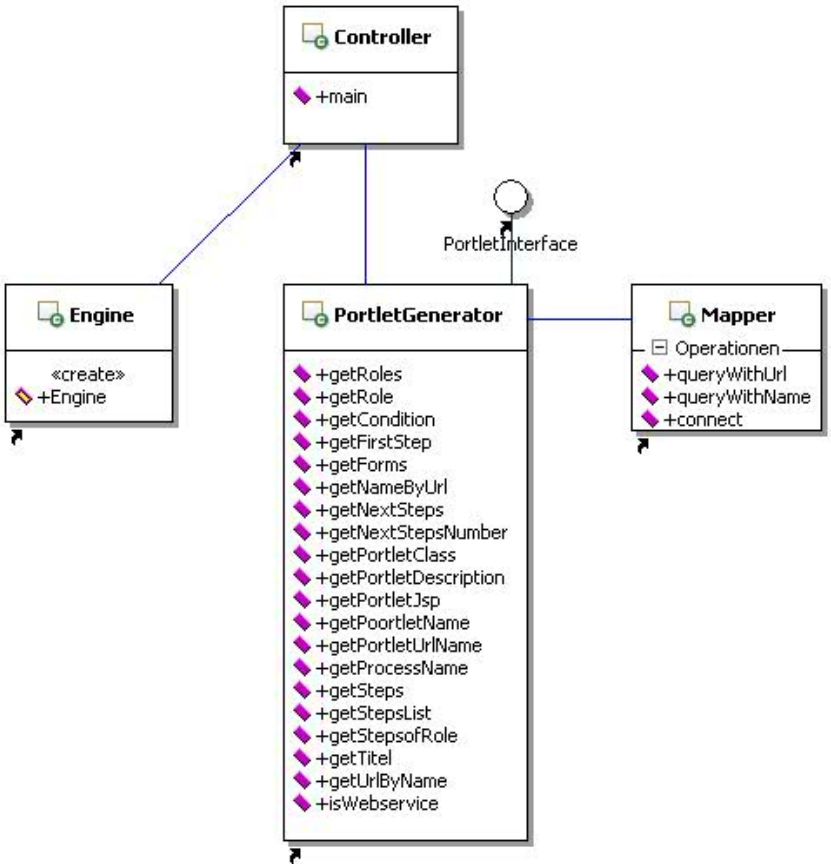


Abbildung 7.2.: Die Klassen des Generators

wobei hier für die Sichtbarkeit von Variablen diesselben Konventionen gelten, wie beispielsweise auch bei Java. Beispielsweise ist die im obigen Beispiel definierte Variable `nachname` ist lediglich innerhalb des ForEach Blocks benutzbar, in dem sie definiert wurde.

7.2. Der Generator

Als Basis zur Entwicklung des Java Programms wurde ein Interface benutzt. Dieses sollte als Schnittstelle zwischen den Templates und der Ablaufdatei dienen. Der Vorteil besteht darin, dass Velocity im Context vorerst nur ein Objekt übergeben werden muss, welches das Interface implementiert. Später kamen noch einige wenige Variablendeklarationen hinzu. Im Großen und Ganzen bestehen alle Informationen, die in den Templates verarbeitet werden aus Rückgabewerten von Methodenaufrufen des Interface. Zusätzlich gibt es eine Klasse Engine, in der der Velocity Aufruf gehandhabt wird - die Velocity Engine

wird erstellt, ebenso wie der Context und der Generierungsvorgang wird gestartet.
Das Klassendiagramm (Abbildung 7.2) zeigt die nötigen Klassen, die für das Projekt erstellt wurden. Diese sollen im Nachfolgenden genauer Beschrieben werden.

7.2.1. Die Klasse Controller

Die Klasse `Controller` übernimmt die Steuerungsfunktionalität. In dieser Klasse befindet sich die Main-Methode, mit der das Programm gestartet werden kann. Im Nachfolgenden wird der Ablauf einer Generierung dargestellt, um das Zusammenspiel der einzelnen Klassen und Methoden zu veranschaulichen.
Nachdem die main-Methode gestartet wird, wird der Ordner, der die Ablaufdateien beinhaltet, durchsucht und die gefundenen Dateien einer ArrayList mit dem Namen „pathNames“ hinzugefügt. Bei der Instanziierung eines Objekts vom Typ `PortletGenerator` (welches die Schnittstelle zum Interface darstellt), muss der Dateiname der zu untersuchenden XPDL-Datei angegeben werden.

```
1 ArrayList<PortletGenerator> portletGeneratorList = new ArrayList();
2 for (int i=0; i<pathNames.size();i++){
3     portletGeneratorList.add(new PortletGenerator(pathNames.get(i)));
4 }
```

Quellcode 7.6: portletGeneratorList

Als nächstes wird eine ArrayList eben dieser `PortletGenerator` Objekte erstellt - siehe Listing 7.6. Auf diese Elemente kann dann im weiteren Verlauf von den Templates aus zugegriffen werden. In einem nächsten Schritt wird die Generierung der drei Portaldeskriptoren gestartet. Hierbei wird der Konstruktor der Klasse `Engine` genutzt.

```
1 new Engine (portletGeneratorList, "templates/portlet-instancesTemplate.vm", "portlet-instances.xml" );
2 new Engine (portletGeneratorList, "templates/SportsPortal-objektTemplate.vm", "SportsPortal-object.xml" );
3 new Engine (portletGeneratorList, "templates/portletXMLTemplate.vm", "portlet.xml" );
```

Quellcode 7.7: Generierung der Deskriptoren

Beim Aufruf (dargestellt in Listing 7.7) wird jeweils die Liste der Objekte der Klasse `PortletGenerator`, der Pfad des Templates und der gewünschte Dateiname des erzeugten Deskriptors übergeben.
Die JAVA-Dateien der einzelnen Portlets werden zwar auf ähnlichem Weg generiert, allerdings ist an dieser Stelle eine etwas kompliziertere Vorgehensweise nötig. Eine äußere Schleife iteriert über die Anzahl der Elemente der innerhalb der `portletGeneratorList`

existierenden Elemente - also die Anzahl der XPDL-Dateien. Eine weitere Schleife iteriert dann über die Anzahl der Schritt innerhalb einer XPDL-Datei. Im Rumpf dieser Schleifen wird durch eine Abfrage unterschieden, welcher Rolle jeder Schritt zuzuordnen ist. In der aktuellen Fassung ist die Abfrage explizit nach der Rolle „Buerger"bzw. „Sachbearbeiter". Wünschenswert wäre eine etwas generischere Abfrage, in der die Namen der Rollen selber aus der XPDL-Datei ausgelesen werden. Dies wurde allerdings bis zur Abgabe des Projekts nicht mehr implementiert. Die Abfrage auf die Rollen ist deshalb nötig, weil für einen Sachbearbeiter andere Templates benutzt werden müssen als für einen Bürger. Der Engine-Aufruf funktioniert dann aber ähnlich wie schon bei den Deskriptoren, außer dass dem Konstruktor noch ein Integer übergeben wird, der angibt, um welchen Schritt innerhalb des Vorgangs es sich handelt. Bis zum diesem Zeitpunkt wird immer der Vorgang als ganzes Betrachtet. Der zusätzlich übergebene Wert dient der Unterscheidung, welcher Schritt im aktuellen Vorgang (beispielsweise „Ausfüllen eines Antrags“ im Prozess „Hund anmelden“) während der folgenden Generierung betrachtet werden soll.

7.2.2. Die Klasse Engine

Die `Engine`-Klasse stellt zwei Konstruktoren zur Verfügung.

```
1  public Engine (ArrayList<PortletGenerator> generate, String templ, String outputFile, int stepId) throws
       Exception {
2      VelocityEngine ve = new VelocityEngine();
3      Properties props = new Properties();
4      props.setProperty("output.encoding", "UTF-8");
5      ve.init(props);
6      Template t = ve.getTemplate(templ);
7      VelocityContext context = new VelocityContext();
8      context.put("gen", generate);
9      context.put("stepId", stepId);
10     BufferedWriter writer = new BufferedWriter(new FileWriter("output\\"+outputFile));
11     t.merge(context, writer);
12     writer.flush();
13     writer.close();
14 }
```

Quellcode 7.8: Konstruktor der Klasse Engine

Der Konstruktor für die Templates zur Generierung der JAVA-Dateien des Portlets wird in Listing 7.8 gezeigt.

Der Code ähnelt stark dem bereits weiter oben gezeigten Code zum Initialisieren und Starten von Velocity. In Zeile 2 wird die Velocity-Engine instanziiert. Eine Neuerung findet sich in Zeile 3. Hier wird ein Properties-Objekt erzeugt. Dieses wird dazu benutzt, um Velocity bestimmte Einstellungen zu übergeben. In Zeile 4 wird mit Hilfe dieses Objekts definiert, dass die Ausgabe eine in UTF-8 kodierte Datei sein soll. Diese Einstellungen werde bei der Initialisierung der Velocity-Engine (Zeile 5) übergeben. Die restlichen Zeilen wurden bereits oben erklärt. Anzumerken bleibt, dass die Variable `gen` ein Objekt vom Typ `PortletGenerator` übergeben bekommt.
Der zweite Konstruktor (Listing 7.9) ist etwas schlanker und anschaulicher.

```
1 public Engine (ArrayList<PortletGenerator> generate, String templ, String outputFile) throws Exception {
2     this (generate, templ, outputFile,0);
3 }
```

Quellcode 7.9: zweiter Konstruktor der Klasse Engine

Der Unterschied liegt, wie bereits oben beschrieben darin, dass der Konstruktor für die Portaldeskriptoren keine zusätzliche Zahl übergeben bekommen muss. Die Zahl beschreibt den aktuell zu betrachtenden Schritt innerhalb eines Ablaufs. Dies ist für die Deskriptoren unerheblich. Innerhalb des Konstruktors passiert auch nicht mehr, als dass der zuerst genannte Konstruktor aufgerufen wird und der benötigte Integer-Wert mit 0 gefüllt wird. Dieser Wert wird dann auch über die Variable `stepID` in das Template übergeben, dort allerdings nicht ausgewertet.

7.2.3. Die Klasse Mapper

Die Klasse `Mapper` hat die Aufgabe ein Mapping zwischen dem Namen des Forumlars eines Portlets und der entsprechenden URL herzustellen. Das Mapping selber geschieht in einer Datenbank, die von der Klasse Mapper mit Hilfe von JDBC verwaltet wird. Es werden zwei wichtigen Methoden zur Verfügung gestellt, die von der Klasse `PortletGenerator` verwendet werden. Mit der Methode `queryWithName` aus Listing 7.10 wird die entsprechende URL aus der Datenbank geholt, die zum übergebenen Namen passt.

```
1 public String queryWithName(String name){
2 try {
3     Statement stmt = c.createStatement ();
4     ResultSet rset = stmt.executeQuery("select url from Mapping where name='"+name+"';");
5     while (rset.next()) {
```

```
6         String url =rset.getString("url");
7         return url;
8     }
9 }
10 catch (Exception sqlexception){
11     System.out.println("Statement exception");
12 }
13 return "null";
14 }
```

Quellcode 7.10: Datenbankanfrage 1

Analog kann mit der Methode `queryWithUrl` aus Listing 7.11 der Name zu einer gegebene URL bestimmt werden.

```
1  public String queryWithUrl(String url){
2  try {
3      Statement stmt = c.createStatement ();
4      ResultSet rset = stmt.executeQuery("select name from Mapping where url='"+url+"';");
5      while (rset.next()) {
6          String name =rset.getString("name");
7          return name;
8      }
9  }
10 catch (Exception sqlexception){
11     System.out.println("Statement exception");
12 }
13 return "null";
14 }
```

Quellcode 7.11: Datenbankanfrage 2

Die Klasse Mapper stellt also nur zwei Methoden nach Außen zur Verfügung.Diese sind allerdings wichtig für das Template, da diese damit während der Generierung über das Interface die URL für das Orbeon Formular bestimmen können. Der Vorteil liegt aber nicht nur darin, sondern in der grundsätzlichen Überlegung, die dem Mapper zu Grunde liegt. Der Ersteller der Prozesse braucht sich während der Erstellung keine Gedanken darum machen, an welcher Stelle auf welchem Server die Formulare liegen. Prinzipiell müssten diese zum Zeitpunkt der Erstellung noch gar nicht existieren. Erst wenn der Prozess modelliert wurde und die Formulare erstellt worden sind, muss das Mapping in die entsprechende Tabelle in die Datenbank eingetragen werden. Dieses bietet dem Benutzer größtmögliche Flexibilität bei der Erstellung des Prozesses.
Das einzige Problem mit diesem momentan existierenden Nachteil liegt in der benötigten Eindeutigkeit der Namen. Sollten in der Tabelle mehrere Eintrage mit demselben Namen existieren, und eine Abfrage auf diesen Namen gestartet werden, werde alle

URLs zurückgegeben, die diesem Namen zugeordnet werden können. Trotzdem wertet das Template nur den ersten zurückgegebenen Namen aus. Dieser Stand, wie er zum Abschluss des Projekts herrscht, ist in gewisser Weise nachteilig, weil etwas Kreativität bei der Findung von Namen vom Benutzer verlangt wird. Trotzdem ist der Mechanismus so robust, dass eine einmal einem bestimmten Namen zugewiesene URL (als Eintrag in der Datenbank) niemals durch einen Eintrag mit identischem Namen verdrängt werden kann. Es wird immer der Eintrag zuerst zurückgegeben, der auch zuerst existierte. Somit ist sichergestellt, dass ein einmal existierender Eintrag und der zugehörige Prozess nicht in gewisser Weise zerstört werden können. Dies ist natürlich dahingehend unnötig, als dass ein einmal generierter Prozess erst neu generiert werden muss, wenn sich etwas am Portal ändert, wenn das Portal also neu generiert wird. Allerdings bedeutet dies auch, dass bei einer Neugenerierung keine Überraschungen auftreten (falsche Zuordnung Formulare - Prozessschritte)

7.2.4. Die Klasse PortletGenerator

Die Klasse `PortletGenerator` gilt als Bindeglied zwischen der Portalgenerierung und der Prozessgenerierung. Sie implementiert ein Interface, in welchem alle Methoden vereinbart wurden, die für die spätere Generierung der Portalelemente benötigt werden. Der Vorgang der Entwicklung dieser Klasse (Parsing der Ablaufdatei, Bereitstellen der gewonnenen Informationen, ...) wird im Nachfolgenden beschrieben. Der Vorteil in der Benutzung eines Interfaces liegt darin, dass nachdem sich über die gemeinsame Schnittstelle geeinigt wurde, jede Gruppe für sich arbeiten kann. Sollten sich die Anforderungen ändern, indem zum Beispiel neue Funktionen benötigt werden, kann dies durch Anpassen des Interfaces geschehen. Die einzelnen Methoden des Interface werden genauer bei der Vorstellung der Templates dargestellt.

7.3. XPDL-Parser

Die Festlegung auf *XPDL* als Zwischenmodell für die im *BizAgi Process Modeler* modellierten Abläufe führte zu der Anforderung, die darin enthaltenen Informationen für eine Weiterverarbeitung in eine andere Datenstruktur zu transformieren. Es musste also ein eigener Parser entwickelt bzw. ein bereits vorhandenes Produkt untersucht werden, das die einzelnen Elemente der *XPDL*-Datei in eine adäquate Datenstruktur überführt, die einen Zugriff durch andere Komponenten ermöglicht.
Dazu suchten wir zuerst nach bereits vorhandenen Lösungen, um diese idealerweise be-

nutzen bzw. Ideen davon in eigene Ansätze übernehmen zu können. Eine Möglichkeit stellte die *Open Business Engine (OBE)*[4] dar, die ein Interface zur Implementierung eines *XPDL*-Parsers bereitstellt. Die Klasse *Dom4jXPDLParser*, die ein Teil der *OBE* ist und dieses Interface implementiert, konnte allerdings nur *XPDL*-Dateien der Version 1.0 verarbeiten und war deshalb für unsere Anforderungen nicht ausreichend. Eine weitere Option stellte das Produkt *Bonita Open Solution*[5] dar. Der darin enthaltene Parser ließ sich zwar extrahieren, jedoch ist es uns nicht gelungen, ihn in eine lauffähige Testumgebung einzubinden. Aus diesem Grund trafen wir die Entscheidung, einen eigenen *XPDL*-Parser zu entwickeln, der *XPDL*-Dateien der Version 2.1 unterstützt, die für unsere Anforderungen notwendigen *XPDL*-Elemente extrahiert und in ein passendes *Java*-Objektmodell transformiert. Zur Erstellung des Objektmodells lehnten wir uns an die von der *Open Business Engine* benutzte Struktur an und passten diese auf unsere Anforderungen an. Zur Extrahierung der einzelnen *XPDL*-Elemente verwendeten wir die *API XOM*[6], die mittels der bereitgestellten Methoden einen leichten Zugriff auf die einzelnen Elemente ermöglicht.

7.3.1. Klassenmodell

Das Modell besteht dabei aus mehreren *Java*-Klassen, die die jeweiligen von uns benötigten *XPDL*-Elemente mit allen wichtigen Attributen und Subelementen abbilden. Die einzelnen Informationen werden dadurch gekapselt und bieten so einen leichten Zugriff auf die Informationen. Im Folgenden werden die einzelnen objektdefinierenden Klassen die in Abb. 7.3 dargestellt sind, detailliert beschrieben.

Participant: Die Klasse *Participant* verfügt über die Attribute *id*, *type*, *name* und *description*. Damit können alle am Prozess beteiligten Personen identifiziert werden.

DataObject: Die Klasse *DataObject* verfügt über die Attribute *id*, *name* und *state*. Damit es möglich ist, die verschiedenen Formulare mit ihren möglichen Zuständen zu identifizieren.

Artifact: Die Klasse *Artifact* verfügt über die Attribute *id*, *name*, *type* und *dataObject*

[4][15] Die *Open Business Engine* ist eine Open Source Workflow-Engine, die unter anderem einen *XPDL*-Parser beinhaltet.

[5][16] *Bonita Open Solution* ist eine Komplettlösung zur Erstellung prozessbasierter Anwendungen, die neben einer Workflow-Engine inklusive *XPDL*-Parser eine Modellierungsoberfläche beinhaltet.

[6][17] *XOM* ist ein Open Source *XML*-Objektmodell und bietet mit der *Java-API* durch die Baumstruktur einen einfachen Zugriff auf die *XML*-Elemente

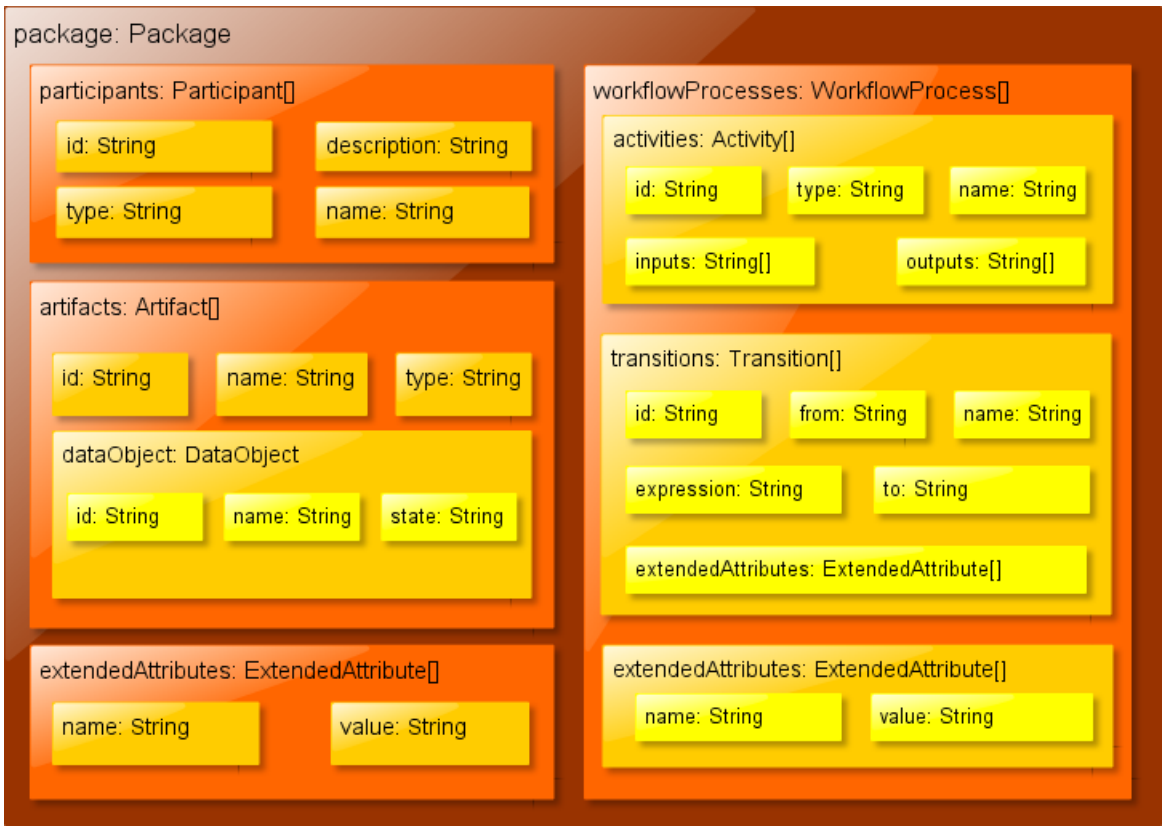


Abbildung 7.3.: Die Struktur des *XPDL*-Objektmodells

und identifiziert die in dem Prozess benötigten Formulare, die durch dieses Element beschrieben werden.

Transition: Die Klasse *Transition* verfügt über die Attribute *id*, *name*, *description*, *from*, *to*, *expression* und *extendedAttributes*. Damit kann die Verbindung zwischen den einzelnen Aktivitäten mittels der Attribute *from* und *to* nachvollzogen werden, um so Nachfolger bzw. Vorgänger bestimmen zu können. Dabei gibt das Attribut *expression* eine Bedingung an, die erfüllt sein muss, damit die jeweilige Transition aktiv ist.

Activitiy: Die Klasse *Activity* verfügt über die Attribute *id*, *name*, *type*, *description*, *performers*, *inputs*, *outputs* und *extendedAttributes*. Damit kann zu jeder modellierten Aktivität der jeweilige Typ und die Verantwortlichkeiten durch das Attribut *performers* mit einer Referenz auf eine *Id* eines oder mehrerer Elemente des Typs *Participant* identifiziert werden. Die Attribute *inputs* und *outputs* enthalten eine Referenz auf die *Ids* der zu einer Aktivität gehörenden Elemente des Typs *Arti-*

fact. Dadurch ist es möglich, für jede Aktivität die Ein- und Ausgaben in Form von Formularen mit einer eventuellen Änderung des Zustandes zu speichern.

WorkflowProcess: Die Klasse *WorkflowProcess* verfügt über die Attribute *id*, *name*, *activities*, *transitions* und *extendedAttributes*. Sie kapselt die wichtigsten Objekte des Prozesses und speichert den Prozessnamen im Attribut *name*.

Package: Die Klasse *Package* kapselt alle vorher geparsten Elemente. Sie besteht also aus dem Objekt *WorkflowProcess* und den weiteren Objekten *Participants*, *Artifacts* und *extendedAttributes*.

7.3.2. Implementierung

Mittels der Modellklassen lassen sich nun aus beliebigen *XPDL*-Dateien als Eingabe komfortabel zugehörige Java-Objekte als Ausgabe erzeugen. Dazu beginnt der Parser beim äußersten Element und arbeitet sich in mehreren Schritten bis zur untersten Elementebene durch das *XPDL*-Dokument. Für das Parsen bietet uns *XOM* hier die nötigen Hilfsmittel, um bequem an die einzelnen *XML*-Elemente und deren Attribute heranzukommen. Wir gehen in aller Kürze auf alle Methoden des Parsers ein und stellen das Vorgehen anhand der Aktivitäten einmal exemplarisch vor.

XPDLParser(...): Der Konstruktor ist dafür zuständig, die übergebene Datei in *XOM* einlesen zu lassen und das *root*-Objekt zu setzen. Damit hat diese Instanz Zugriff auf das gesamte *XPDL*-Dokument in einer Baumstruktur.

parsePackage(): Die einzige Methode, die von außen sichtbar ist und somit den Ausgangspunkt für die Benutzung des Parsers bildet. Sie löst beim Aufruf die mehrstufige Abarbeitung der Eingabe aus und liefert ein *Package*-Objekt, das unserer *XPDL*-Struktur entspricht. Alle anderen Methoden sind Hilfsmethoden, die nur intern verwendet werden und nach einem ähnlichen Schema verfahren.

parseParticipants(...): Sucht in dem übergebenen Element nach dem Unterknoten *Participants* und ruft *parseParticipant()* wiederholt auf, bis alle dazugehörigen Elemente in *Participant*-Objekte umgesetzt wurden. Diese Objekte werden in einem Feld zusammengefasst und zurückgegeben.

parseParticipant(...): Behandelt ein einzelnes *Participant*-Objekt, indem es die nötigen Attribute setzt und das Ergebnis zurückliefert.

parseWorkflowProcesses(...): Geht nach dem selben Prinzip vor, wie *parseParticipants()* oder *parseActivities()* und liefert ein Feld von *WorkflowProcess*-Objekten. Obwohl wir nur einen Prozess in einer Datei modellieren, exportiert der *BizAgi Process Modeler* zwei Prozesse, von denen für uns aber nur der Zweite relevant ist.

parseWorkflowProcess(...): Erzeugt ein *WorkflowProcess*-Objekt mit den passenden Attributen.

parseActivities(...): Anhand dieser und der zugehörigen nachfolgenden Methode erklären wir das generelle Vorgehen im Detail. Als Parameter wird das Element *e* erwartet, das die *Activities* als Unterknoten enthält. Sollte *e* auf *null* referenzieren oder der besagte Unterknoten nicht existieren, wird ein leeres Feld zurückgegeben. Ansonsten wird abhängig von der Anzahl der Unterelemente ein *Activity*-Feld erzeugt und dessen Einträge durch *parseActivity()* weiterverarbeitet.

```
1   private Activity[] parseActivities(Element e) {
2       e = e == null ? null : e.getFirstChildElement(E.Activities.name(), nsURI);
3       Elements l = e == null ? null : e.getChildElements();
4       Activity[] r = new Activity[l == null ? 0 : l.size()];
5       for (int i = 0; i < r.length; i++)
6           r[i] = parseActivity(l.get(i));
7       return r;
8   }
```

parseActivity(...): Als Eingabe wird bereits ein *Activity*-Element erwartet, das von der vorherigen Methode als Kindelement übergeben wird. Es wird daraufhin ein entsprechendes *Activity*-Objekt erstellt, anschließend dessen Attribute durch die entsprechenden Untermethoden gesetzt und an *parseActivities()* zurückgeliefert.

```
1   private Activity parseActivity(Element e) {
2       Activity r = new Activity();
3       r.setId(parseId(e));
4       r.setName(parseName(e));
5       r.setDescription(parseDescription(e));
6       r.setType(parseActivityType(e));
7       r.setPerformers(parsePerformers(e));
8       r.setInputs(parseInputs(e));
9       r.setOutputs(parseOutputs(e));
10      r.setExtendedAttributes(parseExtendedAttributes(e));
11      return r;
12  }
```

parseActivityType(...): Abhängig davon, ob das Element ein *Event*, eine *Implementation* oder eine *Route* ist, wird der Typ der Aktivität zurückgegeben.

parseEvent(...): Liefert den Typ des *Events* zurück, das in erster Linie ein *Start Event* oder *End Event* ist.

parseImplementation(...): Da wir nur eine Art von *Implementation* verwenden, wird der Aufruf nach *parseTask()* weitergeleitet.

parseTask(...): Liefert den genauen Typ des *Tasks*, falls dieser deklariert wurde.

parseRoute(...): Gibt den *Gateway*-Typ zurück. Meistens ist es einfach *Route*.

parsePerformers(...): Einer Aufgabe können mehrere *Performers* zugeordnet sein, die hier in einem Feld zurückgegeben werden. Die Feldeinträge stellen Referenzen auf die *Participants* dar.

parseInputs(...): Diese Methode ermöglicht es, die eingehenden Verbindungen mit *Artifacts* zu erfassen und in einem *String*-Array zurückzugeben.

parseOutputs(...): Funktioniert analog zu *parseInputs()* für ausgehende Verbindungen.

parseArtifacts(...): Schleife für den mehrfachen Aufruf von *parseArtifact()*, die die Ergebnisse in einem Array zusammenfasst.

parseArtifact(...): Hier werden die Attribute und ein mögliches Datenobjekt eines *Artifacts* gesetzt.

parseDataObject(...): Neben dem Namen enthält das zurückgegebene Objekt dessen Status, den wir ebenfalls benötigen.

parseTransitions(...): Das Rückgabefeld besteht aus Verbindungen, die durch mehrfache Aufrufe von *parseTransition()* ermittelt werden.

parseTransition(...): An dieser Stelle wird eine Verbindung zwischen zwei *Activities* erzeugt, die durch eine Quelle und ein Ziel angegeben werden. Außerdem werden auch der Bedingungsausdruck und die Beschreibung ausgelesen.

parseId(...): Elementare Methode, die an vielen Stellen verwendet wird, um das *Id*-Attribut zu erhalten.

parseName(...): Ähnlich wie *parseId()* liefert es nur den Wert eines Attributs zurück. In diesem Fall ist es das Attribut *Name*.

parseDescription(...): Einige andere Elemente können ein *Description*-Element enthalten, dessen Wert mit dieser Methode bestimmt werden kann.

parseExtendedAttributes(...): Auch hier wird abhängig von der Anzahl der Unterelemente die zuständige Methode mehrmals aufgerufen. Die Ergebnisse von *parseExtendedAttribute()* werden in einem Feld zurückgeliefert.

parseExtendedAttribute(...): Das Objekt *ExtendedAttribute* besteht lediglich aus einem Namen und einem beliebigen Wert.

Möchte man den Parser nun verwenden, so genügt es dem Konstruktor den Dateipfad zu übergeben, um den *XML*-Baum aufzustellen. Zugriff auf die *XPDL*-Elemente erhält man durch den Aufruf der Methode *parsePackage()*, die in mehreren Stufen ein *Package*-Objekt erstellt und zurückgibt. Zwar ist der Funktionsumfang im Vergleich zur *XPDL*-Referenz stark eingeschränkt, lässt sich aber auf dieser Grundlage nachträglich beliebig erweitern.

7.4. Implementierung der Klasse Portletgenerator

An dieser Stelle kommt der *XPDL*-Parser als Werkzeug zum Einsatz, um die Schnittstelle zur Portalgenerierung umzusetzen. Wir sind damit in der Lage, alle notwendigen Informationen aus unserem Zwischenmodell direkt oder indirekt herauszuholen. In manchen Fällen ist es erforderlich, mehrere kleinere Umwege über andere Elemente zu gehen, um an die gewünschten Daten zu kommen, in einfachen Fällen lassen sie sich auch unmittelbar aus den entsprechenden *XPDL*-Elementen herleiten. Wir gehen kurz auf die Umsetzung ein und erklären die zentralen Methoden ausführlicher.

PortletGenerator(...): Im Konstruktor werden der Parser mit dem übergebenen Dateipfad initialisiert und die wichtigsten Elemente in Hilfsvariablen abgespeichert.

getRoles(): Durchsucht alle im Prozess existierenden Aktivitäten. Falls diese Aktivität ein *User Task* ist, werden die der Aktivität zugeordneten Rollenreferenzen mit allen im Prozess vorkommenden Rollenreferenzen verglichen. Einer Aktivität kann mehr als eine Rolle zugeordnet sein, jedoch wird in unserem Fall diese Anzahl auf eins beschränkt. Bei einer Übereinstimmung wird der zu der Rollenreferenz gehörende

Rollenname in einem Feld gespeichert und zurückgegeben. Dadurch werden alle im Prozess vorkommenden Rollen identifiziert.

getRole(...): An dieser kurzen Methode stellen wir das prinzipielle Vorgehen beispielhaft vor. Für den Parameter *id*, der eine Aktivität im *XPDL*-Modell referenziert, soll die zugehörige Rolle ermittelt werden. Dazu muss zuerst das betreffende Aktivitätsobjekt im Feld *activities* bestimmt werden. Da die Generierung keine besonderen Anforderungen bezüglich der Performanz erfüllen soll, werden in diesen Fällen die Arrays ohne besondere Strategien einfach von Anfang bis Ende durchsucht. Stimmt die *id* der *Activity* mit dem aktuellen Feldelement überein, wird die Referenz auf den ersten[7] *Performer* festgehalten. Anschließend wird auch diese Referenz aufgelöst, indem das *Participant*-Feld nach dem passenden Eintrag durchsucht wird, und der Name der Rolle wird zurückgegeben.

```
1   @Override
2   public String getRole(String id) {
3       String p = "", r = "";
4       for (Activity activity : activities)
5           if (activity.getId().equals(id))
6               p = activity.getPerformers()[0];
7       for (Participant participant : participants)
8           if (p.equals(participant.getId()))
9               r = participant.getName();
10      return r;
11  }
```

getStepsOfRole(...): Liefert zu einem Rollennamen die Referenzen aller Aktivitäten, die diese Rolle als *Performer* aufführen.

getSteps(): Durchsucht alle im Prozess vorkommenden Aktivitäten, überprüft deren Typ und speichert alle Aktivitäten, die *User Tasks* sind in einem Feld und gibt diese zurück.

getFirstStep(): Identifiziert das *Start Event* durch eine Prüfung des Aktivitätstyps aller Aktivitäten. Die Aktivität vom Typ *Start Event* stellt zwar den ersten Knoten innerhalb des Ablaufs dar, jedoch ist sie nicht relevant für den eigentlichen Prozess. Daher muss nun mit der Hilfe der Transitionen die erste Aktivität bestimmt werden, indem man die Transition selektiert, die das *Start Event* als Quelle enthält. Der erste relevante Schritt ist nun das Ziel dieser Transition.

[7] Zwar können *Activities* mehreren *Participants* zugeordnet sein, aber nach unseren Konventionen sollte es nur einen *Performer* geben, da nur der Erste beachtet wird.

getNextSteps(...): Bestimmt den oder die nachfolgenden Schritte zu der Aktivität mit der zugehörigen übergebenen *Id*. Dafür wird zunächst die Transition selektiert durch deren Quelle die Aktivität mit der übergebenen *Id* referenziert wird. In Bezug auf das Ziel der Transition, die den oder die nachfolgenden Aktivitäten darstellen, sind nun zwei Fälle zu unterscheiden. Falls die als Ziel referenzierte Aktivität ein *Task* ist, so wird deren Id zu einem Feld hinzugefügt. Falls die referenzierte Aktivität jedoch eine Aktivität des Typs *Gateway* ist, so werden die Transitionen selektiert, deren Quelle eine Referenz auf dieses Gateway darstellt, die jeweiligen Zielreferenzen der selektierten Transitionen einem Feld hinzugefügt und dieses Feld zurückgegeben.

getCondition(...): Ziel dieser simplen Methode ist es, die Kantenbeschriftung der eingehenden Kante und somit die Vorbedingung einer Aktivität zurückzugeben.

isWebservice(...): Das Ergebnis dieser kurzen Methode gibt lediglich an, ob es sich bei der referenzierten Aktivität um einen *Send Task* handelt oder nicht.

getPortletName(...): Gibt den Namen der Aktivität zurück, deren *Id* mit der übergebenen *Id* übereinstimmt.

getPortletClass(...): Ruft die Methode *getPortletName()* mit der übergebenen *Id* und die Methode *getProcessName()* auf und konkateniert die beiden Ergebnis-Strings. Anschließend werden zur Erzeugung eines gültigen Klassennamens alle in diesem String vorkommenden Leerzeichen entfernt und Umlaute ersetzt.

getPortletDescription(...): Gibt die Beschreibung der Aktivität zurück, deren *Id* mit der übergebenen *Id* übereinstimmt.

getShortTitle(...): Der Einfachheit halber wird auf *getPortletName()* zurückgegriffen.

getTitle(...): Ruft die Methode *getPortletDescription()* mit der übergebenen *Id* auf.

getProcessName(): Gibt den Namen des ersten *WorkflowProcess*-Objekts zurück, das den modellierten Geschäftsprozess darstellt.

getForms(...): In dieser Methode wird die Verbindung zwischen den Aktivitäten und ihren zugehörigen Datenobjekten hergestellt und in Form eines String-Felds zurückgegeben. Dazu müssen zunächst die Eingabereferenzen der angegebenen Aktivität festgestellt und dann für jede Referenz im Feld *artifacts* der Name und Status

des entsprechenden Datenobjekts gesetzt werden. Das Ergebnis enthält also alle mit der Aktivität assoziierten Formularnamen und deren Zustände.

getPortletUrlName(...): Hier wird der Formularname mithilfe des *Mappers* auf die entsprechende Adresse des *Orbeon*-Formulars abgebildet.

7.5. Velocity Templates

Dieser Abschnitt beschäftigt sich mit den Velocity-Templates. Im Detail soll darauf eingegangen werden, welche Templates für welche Dateien verantwortlich sind und es sollen exemplarisch markante Stellen aufgezeigt werden. Es soll aber auch gezeigt werden, an welchen Stellen gewisse Einschränkungen getroffen werden mussten. Grundsätzlich gilt, dass jedes Template aus großen Bereichen besteht, in denen nichts generiert werden muss, da es sich um allgemeingültigen Code handelt. Diese Stellen werden in der nachfolgenden Darstellung zumeist übersprungen. Insgesamt gibt es 6 verschiedene Templates - für 3 Deskriptoren, das Portlet für den Bürger/Sachbearbeiter und den Posteingang des Sachbearbeiters.

7.5.1. Template: portletTemplate.vm

Das Template `portletTemplate.vm` generiert das Portlet (JAVA-Datei) für den Bürger. Um den nachfolgenden Code lesbarer zu machen werden zu Beginn zwei Variablen deklariert (Listing 7.12.

```
1 #set ($variable = $gen.getSteps().get($stepId))
2 #set ($prcName= $gen.getProcessName())
```

Quellcode 7.12: Variablendeklaration innerhalb des Templates

Die Variable `gen` ist eine Referenz auf ein Objekt der Klasse `PortletGenerator`. Über die Methode `getProcessName()` wird der Name des aktuell betrachteten Ablaufs bestimmt. Der Methodenaufruf für die Variable `variable` ist etwas verworrener. Die Methode `getSteps` gibt die ID aller Schritte im aktuell betrachteten Ablauf zurück. Dies geschieht in Form einer ArrayList. Die Variable `stepId` gibt die Position des aktuellen Schritts in dieser Liste an. Über `get($stepId)` wird so dann die tatsächlich ID des aktuellen Schritts bestimmt. Diese ID ist identisch mit der in der Ablaufdatei und wird für spätere Methodenaufrufe benötigt. Da die späteren Methodenaufrufe aber ziemlich lang werden und noch länger werden würden, wenn man anstatt der ID in der Variablen

`variable` zu übergeben, den Methodenaufruf zum Bestimmen der ID benutzen würde, ist die Deklaration an dieser Stelle durchaus sinnvoll und fördert die Lesbarkeit des Templates. Prinzipiell handelt es sich bei diesem Template um einen einfachen Lückentext, bei dem lediglich an die richtigen Stellen die entsprechenden Funktionsaufrufe gesetzt werden müssen. Listing 7.13 zeigt ein Beispiel.

```
1 response.setTitle("$gen.getTitle($variable)");
```

Quellcode 7.13: Methodenaufruf aus den Templates

Die Methode `response.setTitle` ist eine Methode aus der Portletdatei. Allerdings benötigt sie als Parameter den Titel des Portlets. Mit der Methode `getTitle(...)` wird der Titel an diese Stelle generiert. Diese Art von Lückenfüllen findet sich durchgängig in allen Portlets.
In der Methode `processAction` findet sich eine weitere prägnante Stelle dieses Templates. Hier soll ein SQL Statement generiert werden, welches ein Update in der „process“-Tabelle macht. Allerdings ändert sich ein Wert in dem Statement, abhängig davon um welchen Ablauf (Hund anmelden oder Beschwerde einreichen) es sich handelt. Prinzipiell wird nun so vorgegangen, dass alle Teile bis auf diesen einen Wert statisch hingeschrieben werden und der Wert selber aus der „process“-Tabelle bestimmt wird.

7.5.2. Template: portletXML.vm

Der erste Teil dieses Templates beschreibt wieder einen einfach zu füllenden Lückentext. Prinzipiell soll aber nicht unerwähnt bleiben, dass die zu leistende Vorarbeit bis zum Erreichen dieses Status nicht zu unterschätzen ist. Auch wenn das Ergebnis an einigen Stellen eventuell etwas minimalistisch aussieht, war doch eine Menge Einarbeitung in die Funktionsweise und in das Zusammenspiel der einzelnen Portletdateien untereinander nötig.
Der zweite Teil des Templates ist im Gegensatz zum ersten interessanter. Hier werden die einzelnen Portlets sortiert nach Benutzer eingetragen. Dies wird so realisiert, dass zwei `for-each`-Schleifen über die verschiedenen Abläufe und über die einzelnen Schritte in den entsprechenden Abläufen iterieren(Listing 7.14).

```
1 #foreach ($xx in $gen)
2 #foreach ($yy in $xx.getSteps())
3   #if($xx.getRole($yy)== 'Buerger' )
```

Quellcode 7.14: Geschachtelte For-Each-Schleifen

Die Variable `gen` ist vom Typ ArrayList, und stellt eine Liste der einzelnen Instanzen von `PortletGenerator` für die verschiedenen Abläufe dar. Somit iteriert die äußere `FOR-EACH`-Schleife genau über alle Abläufe (Hund anmelden und Beschwerde einreichen). Die innere Schleife ruft die Methode `getSteps()` jeweils auf einem der Abläufe auf. In dieser Methode werden die IDs aller Schritte zurückgegeben, über die dann iteriert wird. Für jeden Schritt muss nun ein Eintrag in der Datei gemacht werden. Die Einträge selber sind wieder reiner Lückentext, der mit den Methoden aus `PortletGenerator` gefüllt werden muss. Allerdings müssen die Einträge, wie bereits oben erwähnt, nach der zugehörigen Rolle sortiert werden, da sie abhängig von der Rolle eine andere Behandlung erfordnern. Dies wird durch IF-Abfragen realisiert. In dem Listing oben wird in Zeile drei geprüft, ob der aktuelle Schritt der Rolle „Buerger“ zuzuordnen ist. Ist dies der Fall wird der Lückentext für diesen Schritt gefüllt und der Eintrag wird an dieser Stelle generiert. Passt die Zuordnung nicht zur Rolle Bürger, wird weiter unten geprüft, ob der Schritt der Rolle Sachbearbeiter zuzuordnen ist. Ist dies der Fall, wird entsprechend an dieser Stelle generiert.
An dieser Stelle zeigt sich wieder, welche Arbeiten in „Zukunft“ anstehen würden. Es wäre natürlich schöner, wenn die einzelnen Rollen nicht direkt in die `if-Abfrage` geschrieben werden würden, sondern wenn alle verfügbaren Rollen über eine Methode aus PortletGenerator zu beziehen wären und dies an dieser Stelle ausgewertet werden würde.

7.5.3. Template: SportsPortalObject.vm

Dieses Template ist von den Anforderungen vergleichbar mit dem portletXML-Template. Wieder wird Lückentext gefüllt. Am Ende des Templates muss Text generiert werden, der explizit der Rolle „Sachbearbeiter“ zugewiesen wird. Die Abfrage die an dieser Stelle passiert, ist identisch mit der aus dem portletXML-Template. Es werden zwei geschachtelte `FOR-EACH-Schleifen` zum Durchlaufen aller Schritte aller Abläufe in Verbindung mit einer `IF-Abfrage` zur Zuordnung der Rolle benötigt.

7.5.4. Template: PosteingangsPortlet.vm

Dieses Template generiert den Posteingang für den Sachbearbeiter. Die Methodik die in diesem Template benutzt wird, ist identisch mit der aus portletTemplate.vm. Wie sich mittlerweile erkennen lässt, wiederholen sich die Techniken immer wieder. Entweder es muss lediglich ein Lückentext gefüllt werden, oder es muss über die Anzahl der Schritte, bzw. die einzelnen Rollen iteriert werden. Grundsätzlich ähneln sich die zu generierenden

Dateien bezüglich ihres zu generierenden Inhalts so stark, dass grundsätzliche Techniken einfach wiederverwendet werden können.

7.5.5. Template: portlet-instances.vm

Dieses Template generiert den „portlet-instances“-Deskriptor. Auch hier werden die Techniken aus den vorherigen Templates genutzt, so dass eine weitere Beschreibung eigentlich nicht nötig ist.

7.5.6. Template: SachbearbeiterPortlet.vm

Das Template für das Sachbearbeiter-Portlet lässt sich an den meisten Stellen mit den bereits mehrfach erwähnten Techniken generieren. Einziger Unterschied hier besteht im „Evaluator“ Es wurde während der Arbeit am Projekt die Annahme getroffen, dass eine Prozesssteuerung lediglich auf Sachbearbeiterseite zu erwarten sei. Der Bürger füllt lediglich Antragsformulare aus und schickt diese ab. Der Sachbearbeiter bearbeitet diese Formulare und muss Entscheidungen treffen. Diese Entscheidungen bestimmen den nächsten Schritt im Ablauf. Somit müssen diese Entscheidungen abgefangen und in der entsprechenden Datenbanktabelle festgehalten werden. Die Auswertung und die Ausführung entsprechender SQL Befehle zum Aktualisieren der Datenbank wird von diesem Portlet übernommen. Innerhalb des Templates muss also einmal der aktuelle Zustand/Schritt bekannt sein und die Entscheidung des Sachbearbeiters muss bekannt sein - also beispielsweise, ob der Antrag abgelehnt oder akzeptiert wurde in diesem Schritt. Entsprechend dieser Angaben muss nun ein SQL Statement generiert werden, welches die Updates in der Prozesstabelle vornimmt.

7.6. Packaging und Deployment mit Ant

Die auf Java basierende Software Ant des Apache Ant Projects [28] steuert den Zusammenbau eines Softwareprojekts anhand einer speziellen XML-Datei, dem sogenannten Build-Script. Beim SportsPortal wird sowohl das Generieren und Kompilieren der Dateien und deren Verpackung in die WAR-Datei als auch das anschließende Veröffentlichen auf dem Portalserver von einem solchem Build-Script übernommen. Im Folgenden wird der wesentliche Inhalt dieser Datei kurz beschrieben, der komplette Code befindet sich im Anhang F.1. Die einzelnen Aufgaben stehen in Blöcken, den sogenannten `targets`. Im ersten Block werden eventuell noch vorhandene Dateien aus früheren Durchläufen

gelöscht, dann legt der nächste Block die im Folgenden benötigte Verzeichnisstruktur an. Anschließend wird die Generierung der Portlets und Deskriptoren durch Aufruf der entsprechenden Generatordatei ausgelöst. Danach werden die generierten Dateien an die richtigen Orte in der Zielstruktur kopiert und dann kompiliert. Als Vorletztes werden alle nötigen Dateien zur Portalanwendung zusammengeführt und in die WAR-Datei verpackt. Zu guter Letzt wird diese nun per SCP-Protokoll (*Secure Copy*) auf den Server kopiert und dort vom Portalserver automatisch erkannt und deployt.

7.7. Anlegen der Formulartabellen

Wie in Kapitel 6.7 beschrieben, existieren neben den Tabellen zur Benutzer- und Rollenverwaltung (**roles**, **citizen**, **officer**) und Prozess(zustands)verwaltung (**process**, **process_state**, **process_type**) auch Tabellen zur Speicherung der in einem Prozess anfallenden Daten, die Formulartabellen. Dazu gehört etwa die Tabelle **complaints** im Beispielfall **Beschwerde einreichen**. Um die manuelle Arbeit bei der Einbindung eines Ablaufes in das Portal zu minimieren, können einfache Formulartabellen, die für jedes Formularfeld eine Spalte bereithalten, automatisch angelegt werden. Allerdings sind dieser Methode Grenzen auferlegt. So können aus den Formularen keine Fremdschlüsselbeziehungen herausgelesen werden. Auch die Sinnhaftigkeit einer Aufteilung der Daten eines Formulars auf zwei Tabellen ist aus der Kenntnis des Formulars alleine nicht ersichtlich. Ein Bespiel dafür ist der Beispielablauf „Hund anmelden“. Dort werden die Daten des Anmeldeformulars, wie in Kapitel 6.7 erklärt, in die zwei Tabellen **dog_data** und **register_dog** geschrieben.
Im Folgenden wird beschrieben, wie die relevanten Informationen aus dem Formular geholt werden, wie daraus die SQL-Queries zur Erstellung von Tabellen werden und wie diese im Generierungsprozess zur Ausführung kommen.

7.7.1. Erzeugung der Create-Table Befehle aus den Formularen

Wie bereits in Kapitel 5.1 erwähnt, helfen XML-Schema-Dateien bei der Typisierung von Formulardaten. Solche xsd-Dateien haben den weiteren Vorteil, dass sie mit Java-SE Bordmitteln wie der DOM-API ausgelesen werden können.
Die Methode `parseTable` der Klasse `FormParser` liest mit diesen Mitteln die mit den Formularfeldern korrespondierenden XML-Schema-Elemente, welche jeweils typisiert sind. Der Aufbau einer zu einem XForms-Formular gehörenden Schema-Datei ist vom Prinzip wie in Listing 7.15, einem gekürzten Beispiel aus dem Fall **Beschwerde einreichen**.

```
<xs:element name="complaint">
  <xs:complexType>
    <xs:sequence minOccurs="0" maxOccurs="1">
      <xs:element name="id" type="xs:int"/>
      <xs:element name="citizen_id" type="xs:int"/>
      <xs:element name="process_id" type="xs:int"/>
      <xs:element name="cfname" type="varchar50"/>
      <xs:element name="cphone" type="varchar25"/>
      <xs:element name="cemail" type="email"/>
      <xs:element name="czipcode" type="zipcode"/>
      <xs:element name="zipcode" type="zipcode"/>
      <xs:element name="complainttext" type="varchar500"/>
      <xs:element name="createdate" type="xs:dateTime"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Quellcode 7.15: Auszug aus der XML-Schema Datei für das Formular **Beschwerde einreichen**

Die einzelnen Formularfelder tauchen hier als `<xs:element>` auf und werden von einer Sequenz geklammert. Diese Sequenz ist das einzige Element eines `<xs:complexType>`. Das Element mit dem Namensattribut **complaint** kann als Tabellenname interpretiert werden. Diese Informationen werden in einem Objekt der Klasse `Table` abgelegt, welche für die Spalten wiederum eine `ArrayList` des Typs `Column` besitzt. Die Klasse `Column` besteht im Wesentlichen aus dem Namen und dem Typ der Spalte.
Die verwendeten Datentypen (`varchar50`, `varchar25`, `varchar500`, `email` und `zipcode`) werden im weiteren Verlauf des XML-Schemas definiert und fehlen in diesem Listing aus Gründen der Lesbarkeit. Die `varchar` Datentypen finden ihre Entsprechung in den SQL-Datentypen `VARCHAR(X)`, wobei X der Zahl aus dem Typnamen entspricht. Diese ist lediglich durch die maximale Spaltengröße der verwendeten Datenbank beschränkt. Die Typen `email` und `zipcode` werden auf `VARCHAR(50)` bzw. `VARCHAR(5)` abgebildet. `xs:int` entspricht `INT` in der Datenbank. Der Datentyp `xs:dateTime` wird schließlich zu `TIMESTAMP`. Mit den Informationen aus dem von `parseTable` aus der XML-Schema Datei erzeugten `Table`-Objekt lässt sich nun eine Create-Table SQL-Abfrage erstellen. Beispielhaft ist dieses in der Klasse `CreateTable` implementiert.

7.7.2. Einbindung der Funktionalität in den Generator

Die automatische Ausführung des Create-Table-Befehls erfordert Vorsicht. Es muss darauf geachtet werden, ob bereits eine Tabelle gleichen Namens in der Datenbank existiert. Ist dies der Fall, muss der Name der zu erstellenden Tabelle geändert werden.

8 Fazit und Zusammenfassung

Die Rahmenbedingung für das Projekt stellte ein rollenbasiertes Web-Portal im Rahmen des E-Government dar. Die Erstellung der Portalinhalte sollte durch Codegenerierung gestützt werden, um die technischen Aspekte der Entwicklung in den Hintergrund zu schieben. Dieser Endbericht sollte zeigen, wie die ausgewählten Werkzeuge und Technologien zusammenarbeiten, um eine automatische Generierung zu ermöglichen.
Am Ende der Projektarbeit kommt der Zeitpunkt, an dem man auf die geleistete Arbeit zurückschaut. Innerhalb eines auf zwei Semester begrenzten Zeitraumes ist es eher unrealistisch alle Visionen und Vorstellungen jedes Teilnehmers bzw. Betreuers von der Projektgruppe umzusetzen. Daher musste wir uns im Laufe der Zeit immer weiter einschränken und klarere Grenzen für unsere Projektarbeit setzen. Die bisherigen Ergebnisse zeigen uns jedoch, dass viel der gewünschten Ziele erreicht haben.
Es ist uns gelungen, einen großen Teil des Entwicklungsvorgangs[1] zu automatisieren. Wie zu erwarten war, sind immer noch manuelle Schritte in diesem Vorgang nötig. So müssen die Erstellung der Formulare und Ablaufdateien noch von Hand geschehen. Doch auch diese Schritte konnten durch Generierung einzelner Dateien unterstützt werden. Der Generator wurde zum Beispiel so erweitert, dass er Dateien generiert, für die es zu Beginn nicht vorgesehen war.

8.1. Ergebnisse der Projektarbeit

Wir haben uns im ersten Semester zunächst in verschiedenen Bereichen orientiert, eingearbeitet und das Gerüst unserer Arbeit aufgestellt. Auf dieser Basis sollten weiterführende Anforderungen und Ziele aufgebaut werden. Hauptsächlich wurden der Bereich „Planung und Konzeption“ und „Produktauswahl“ für unterschiedliche Entwicklungsumgebungen während der Anfangsphase abgeschlossen. Es wurden Portlets manuell erstellt und auf einem Portalserver deployt. Dafür haben wir zwei Prototypen implementiert. Der Zweck der Umsetzung von Prototypen war es, sich besser mit den technischen Umgebungen vertraut zu machen.
Die Arbeit hat sich auf das zweite Semester verlagert, da wir einige Konzepte anpassen mussten. Wir haben uns für einen neuen Prozess-Editor entschieden und den Einsatz

[1] Von der Erstellung der fachlichen Abläufe bis hin zur Integrierung der ausgewählten Abläufe in das Portal

von Formulartechnologien zusätzlich mitaufgenommen, was uns neue Vorgehensweisen und Konzepte ermöglicht hat.
Wir haben stets verschiedene Konzepte bzw. Problemstellungen in kleine Teilaufgaben aufgeteilt, was für eine effektive Gruppenarbeit empfehlenswert ist. Dann haben wir Gruppen gebildet, die sich je mit diesen Teilaufgaben auseinandergesetzt haben. Das war von Vorteil, da Gruppen bzw. Personen nach einer gewissen Einarbeitungszeit gut mit den entsprechenden Werkzeugen bzw. Technologien umgehen konnten. Deswegen gab es Personen, die innerhalb der ganzen Laufzeit für bestimmte Themengebiete zuständig waren.
Die Teilaufgaben konzentrierten sich auf drei große Bereiche. Zum einen war dies die Modellierung der Abläufe in BPMN. Dieser Bereich umfasste auch die in XPDL transformierte Form der BPMN-Modelle. Dies galt für uns als Zwischenmodell. Außerdem gehörte die Erstellung von elektronischen Formularen in XForms mit Hilfe von Orbeon Forms zum ersten Bereich.
Der zweite Bereich war die Erstellung des Portals mit allen dazugehörigen Teilen. Hierzu gehörte sowohl die Erstellung der Portlets, der Web-Service- Aufrufe und eines Datenbankschemas, als auch das Implementieren der Web-Service-Aufrufe und des Zugriffs auf die Datenbank.
Die dritte große Teilaufgabe war der Generator, der die Komponenten der zwei anderen Teilaufgaben miteinander verbinden sollte. Der Generator sollte die Modelle als Eingabe aufnehmen und sie auf das Portal abbilden. Es wurde eine Template-basierte Generierung mittels des Softwareprodukts Velocity eingesetzt. Der Generator benötigte für die Generierung einen Parser namens XOM, der XPDL-Dateien auslesen konnte. Der Generator ist das Kernstück unseres Projektes, da er alle benötigten Teilkomponenten miteinander verbindet und eine erfolgreiche Abbildung der erstellten Modelle und dazugehörigen Formulare auf dem Portal ermöglicht.
Wir haben erfolgreich die beiden Prozesse *Beschwerde einreichen* und *Hund anmelden* generiert und auf das Portal abgebildet.

Im Folgenden werden einige Vorteile bzw. Nachteile von uns eingesetzten Werkzeugen und Technologien näher beschrieben:
Der BizAgi Process Modeler ermöglichte uns die Annotation vieler unterschiedlicher Informationen mit Hilfe von BPMN. Dadurch bot er eine gute Modellierungsplattform für die verschiedenen Prozesse. Außerdem konnte BizAgi die erstellten Prozesse in XPDL-Datein exportieren, die als Eingabe für unseren Parser dienten. Der andere Grund für

die Auswahl von BizAgi war, dass dieser zu diesem Zeitpunkt im Gegensatz zu anderen Tools die aktuellste Version von XPDL unterstützte.
Mit Hilfe von BPMN konnten die Geschäftsprozesse in einer einfachen Darstellung modelliert werden. Für die Rollenzuweisungen, Typisierung und Aufgabenteilung bietet BPMN zahlreiche Task-Typen und Funktionalitäten sowie Pools und Lanes. Service Tasks und Send Tasks unter BPMN ermöglichten uns Darstellung eines externen Webservices. Verzweigungen, Entscheidungen und Bedingungen in unserer Szenarien, konnten mit Hilfe von Gateways modelliert werden.
Die Transformation von BPMN-Dateien in XPDL-Dateien mittels BizAgi benötigte keine weitere Anpassung und Bearbeitung, da die exportierten XPDL-Dateien alle modellierten Aktivitäten und annotierten Zusatzinformationen von BPMN übernahmen. Dadurch konnten wir uns zusätzlichen Arbeitsaufwand ersparen.
Der Einsatz von XForms brachte uns einige Vorteile. So war die Typisierung der verwendeten Daten mittels eines XML-Schemas und die Kompatibilität mit XML gegeben. Die Benutzung von unterschiedlichen Aktionen innerhalb XForms vereinfachte die Darstellung und Realisierung der Abläufe.
Das für XForms eingesetzte Tool war Orbeon Forms als XForms-Engine. Die mit Orbeon erstellten formularbasierten Anwendungen können in einem einfachen Browserfenster geöffnet werden und erfordern daher keine weitere Software. Die Entwicklung der elektronischen Formulare musste mangels eines geeigneten Editors manuell geschehen. Dies stellte sich als sehr mühsam heraus, da die Fehlersuche in den Dateien der Formulare mit einem einfachen Editor und den Serverlogdateien durchgeführt wurde. Es gab keine Debugging-Hilfe, wie sie in aktuellen Entwicklungsumgebungen zu finden sind.
Es wurden an dieser Stelle einige Überlegungen angestellt, um die Handarbeit zu minimieren. Durch die sehr spezielle Natur der einzelnen Dateien, war dies jedoch nur eingeschränkt möglich. Zum Beispiel sind bei automatischem Anlegen von Formulartabellen Probleme aufgetreten, so dass aus den Formularen keine Fremdschlüsselbeziehung herausgelesen werden konnten. Die Verwendung von XForms und Orbeon war dennoch eine gute Wahl. Sie bieten viele Features, die für die Gestaltung eines übersichtlichen und benutzerfreundlichen Formulars verwendet werden konnten.
Der Einsatz von Velocity für unsere Zwecke war eine optimale Entscheidung, da die Software einerseits leicht zu warten und erweitern ist und andererseits relativ flexibel bzgl. neuer Anforderungen und Änderungen von Templates ist.

8.2. Ausblick

Hier werden einige Optimierungen bzw. Erweiterungen bzgl. unserer Projektarbeit, die an manchen Stellen vorgenommen werden können, aufgelistet.
Anhand unserer Vorgehensweise kann man z. B. komplexere Geschäftsprozesse modellieren. Man könnte eine bessere und benutzerfreundlichere Oberfläche für die Portalseite erstellen. Eine Entwicklung eines Editors für Orbeon-konforme Formulare, die universeller eingesetzt werden können, als die des Orbeon Form Builders, wäre denkbar. Vollständige Automatisierung im Generierungsprozess ist ein guter Ansatz zur Erweiterung der Arbeit. Die vorhandene Semantik, die in den Ablaufdateien vorhanden ist, kann in einer komplexeren Form erweitert werden. Umsetzung und Entwurf eines Editors, der bzgl. unserer Anforderungen angepasst wurde, um einfacher und schneller Prozesse zu modellieren, wäre eine vorteilhafte Herangehensweise, um den Entwicklungsprozess zu beschleunigen. Damit sollten auch semantische Beschreibungen und interaktive Operationen unterstützt werden.

A Datenbankschema

Hier sind die Grafiken des Datenbankschemas noch einmal in etwas größerer Auflösung.

Tabellenübersicht In Abbildung A.1 ist eine Übersicht über alle Tabellen zu sehen.

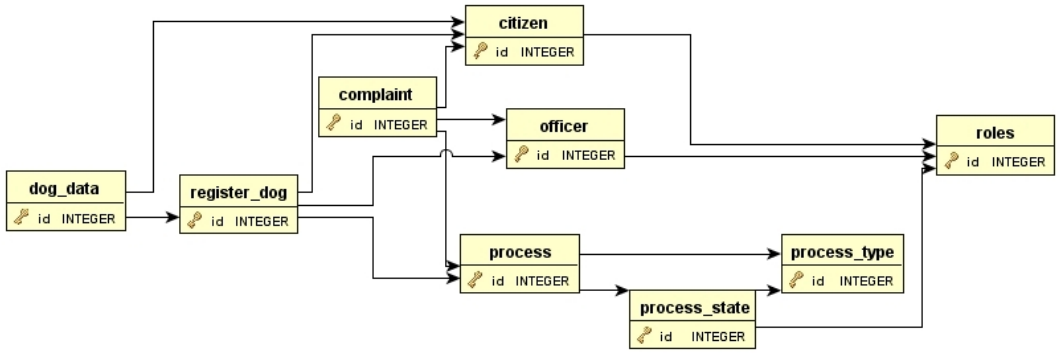


Abbildung A.1.: Übersicht über das Datenbankschema

Gruppeneinteilung Hier (Abbildung A.2) ist die Einteilung der Tabellen in drei Gruppen abgebildet.

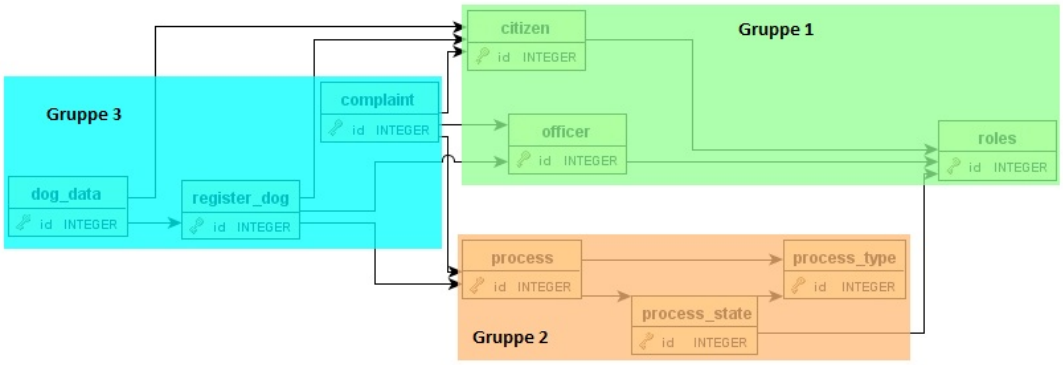


Abbildung A.2.: Einteilung der Tabellen in Gruppen

Benutzer In der hier gezeigten Abbildung A.3 ist erkennbar, wie die Benutzer in der Datenbank repräsentiert werden.

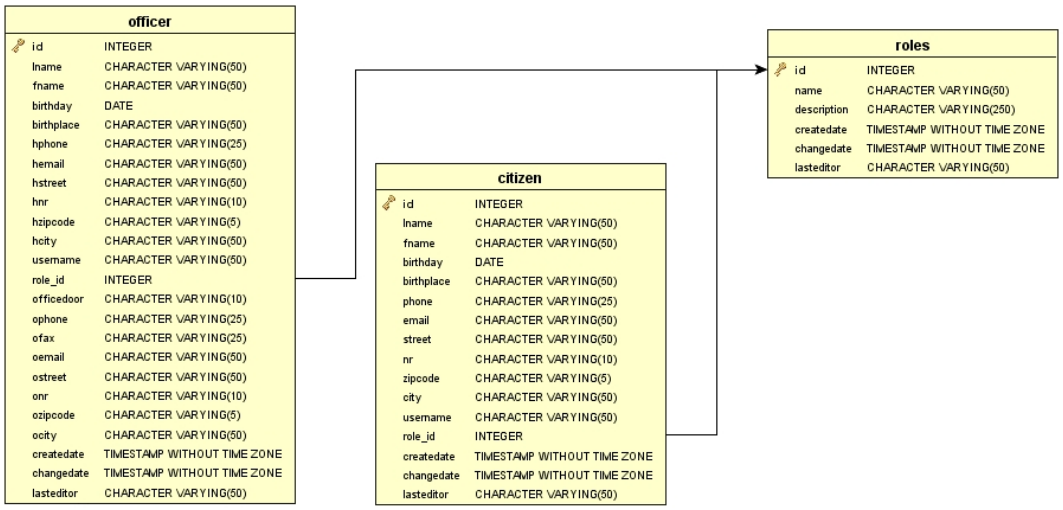


Abbildung A.3.: Die Benutzer in der Datenbank

Ablaufverwaltung Abbildung A.4 zeigt die Tabellen für die Ablaufverwaltung.

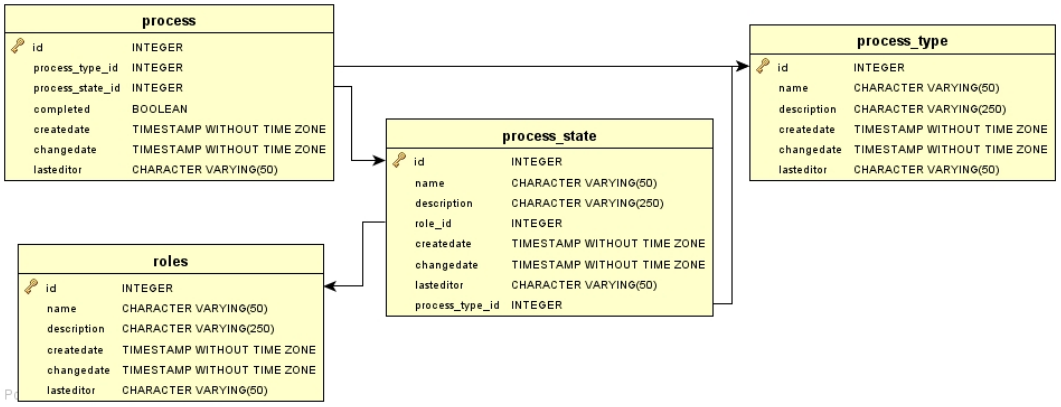


Abbildung A.4.: Aufbau der Abläufe in der Datenbank

Abläufe Die Tabellen für die umgesetzten fachlichen Abläufe werden in Abbildung A.5 gezeigt.

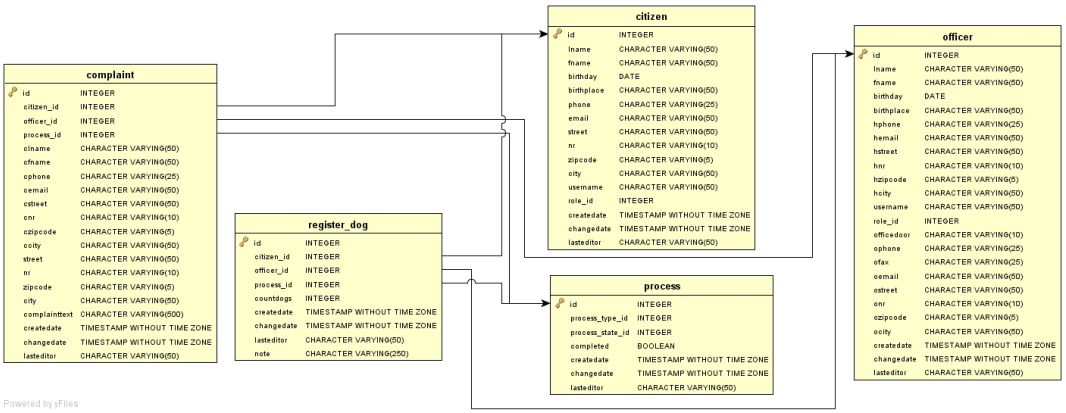


Abbildung A.5.: Die fachlichen Abläufe in der Datenbank

Beschwerdeablauf Hier (Abbildung A.6) ist die Tabelle für den Beschwerdeablauf inklusive der Tabellen, die direkt mit ihr in Beziehung stehen, abgebildet.

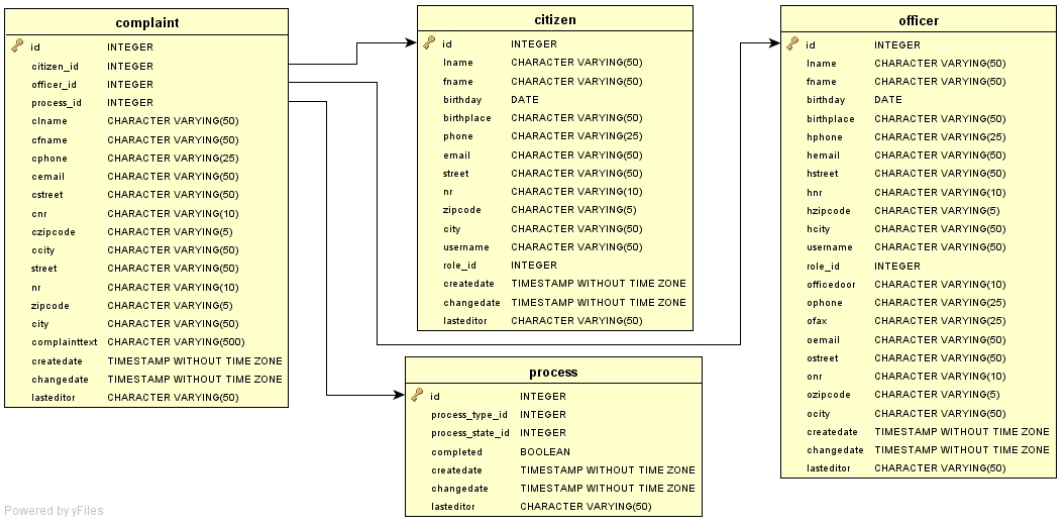


Abbildung A.6.: Die Beschwerde in der Datenbank

„Hund anmelden“-Ablauf Hier (Abbildung A.7) ist die Tabelle für den Beschwerdeablauf inklusive der Tabellen, die direkt mit ihr in Beziehung stehen, abgebildet.

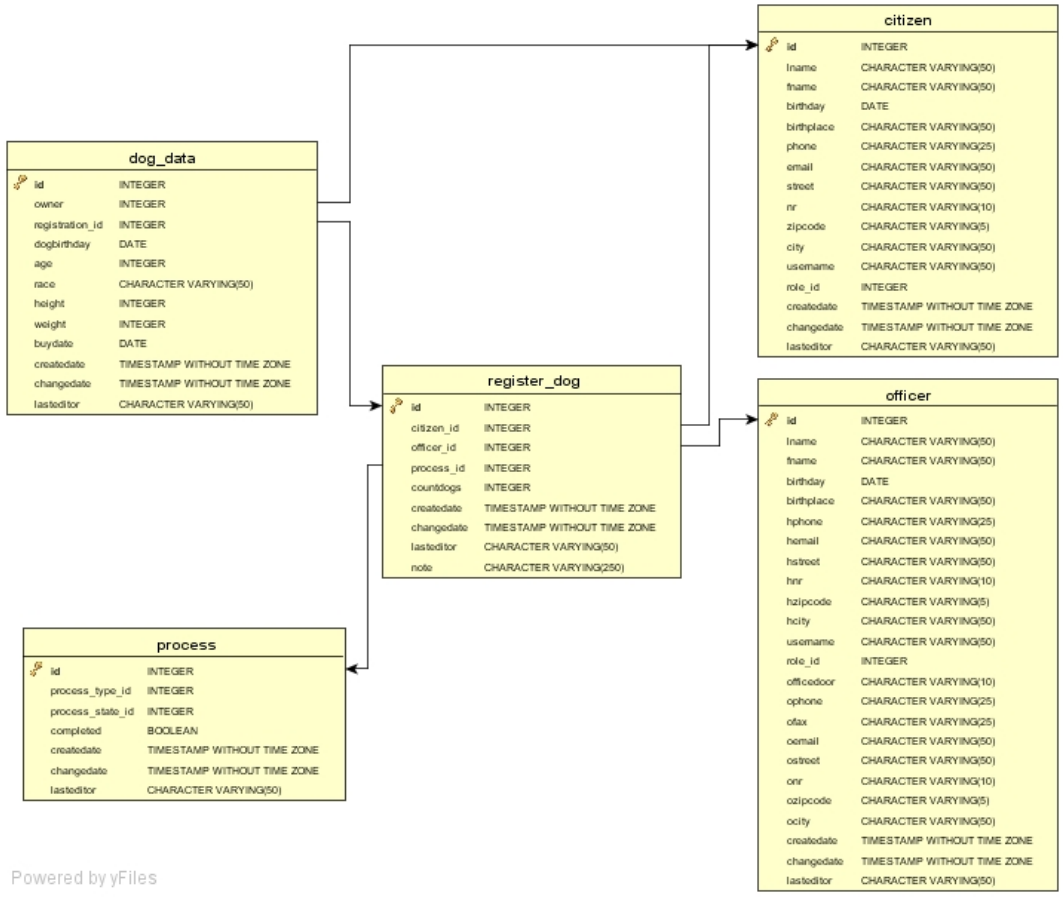


Abbildung A.7.: Der Hundanmeldenablauf in der Datenbank

B Beispiele für Orbeon Dateien

Listing B.1 zeigt den Aufbau einer XML-Pipeline-Datei am Beispiel einer Select-Datenbankabfrage. Dieses Listing dient nur als Strukturbeispiel und wird nicht weiter beschrieben.

```
<p:config xmlns:p="http://www.orbeon.com/oxf/pipeline"
          xmlns:sql="http://orbeon.org/oxf/xml/sql"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:oxf="http://www.orbeon.com/oxf/processors"
          xmlns:xi="http://www.w3.org/2001/XInclude">

<p:param type="input" name="instance"/>
  <p:param type="output" name="data"/>

    <p:processor name="oxf:sql">
        <p:input name="data" href="#instance"/>
        <p:input name="config">
            <sql:config>
              <sql:connection>
                <sql:datasource>PostgresDS</sql:datasource>
                  <sql:execute>
                    <sql:query>
                      select * from citizen where id=<sql:param type="xs:string" select="/dummy/user"/>
                    </sql:query>
                    <sql:result-set>
                      <sql:row-iterator>
                        <datensatz>
                          <id><sql:get-column-value type="xs:int" column="id"/></id>
                          <lname><sql:get-column-value type="xs:string" column="lname"/></lname>
                          ...
                          <changedate><sql:get-column-value type="xs:dateTime"
                                column="changedate"/></changedate>
                          <lasteditor><sql:get-column-value type="xs:string"
                                column="lasteditor"/></lasteditor>
                        </datensatz>
                      </sql:row-iterator>
                    </sql:result-set>
                  </sql:execute>
              </sql:connection>
            </sql:config>
          </p:input>
          <p:output name="data" ref="data"/>
    </p:processor>

</p:config>
```

Quellcode B.1: Aufbau einer XML Pipeline am Beispiel eines SQL-Select-Statements

C Fachliche Formulare

Die im Kapitel 2.2.1 referenzierte Abbildung des Beschwerdeformulars befindet sich hier.

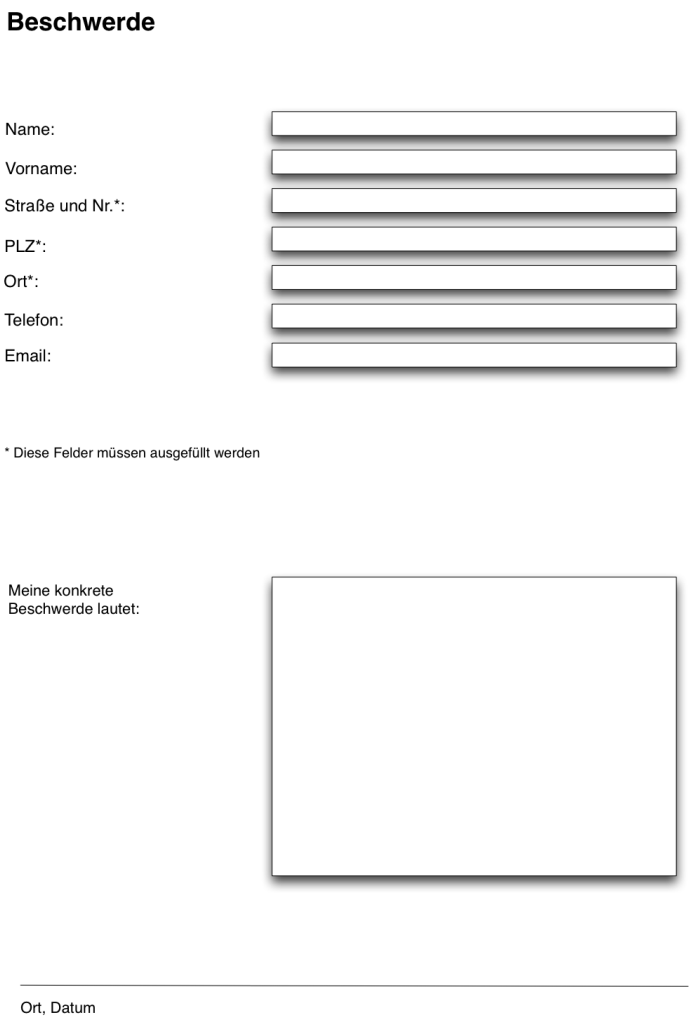
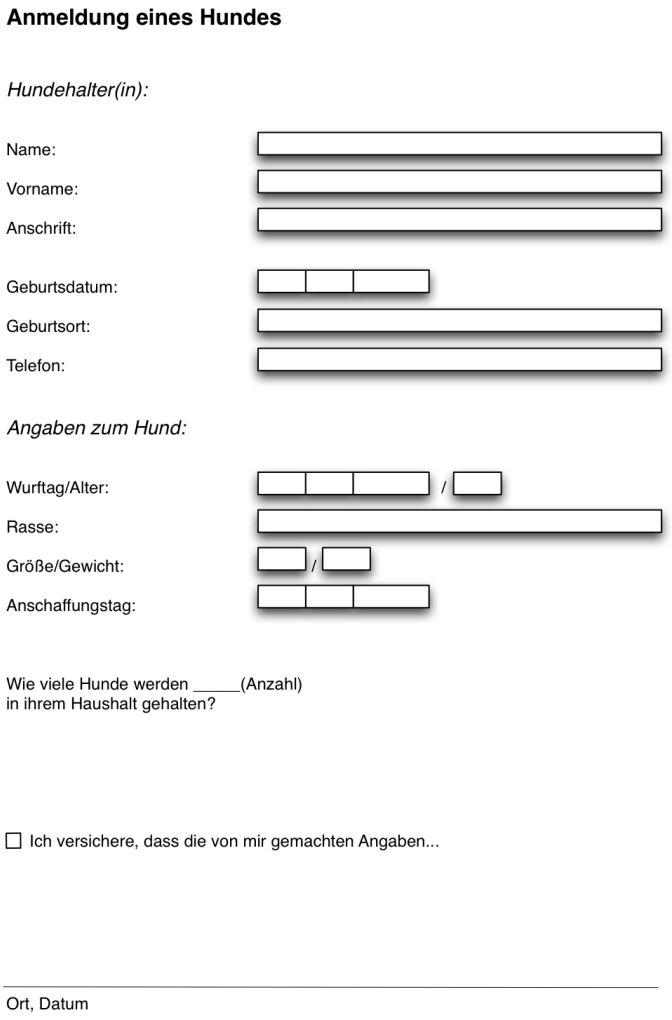


Abbildung C.1.: Formular für „Beschwerde“

Hier findet sich die im Kapitel 2.2.2 referenzierte HundAnmeldenFormular



Anmeldung eines Hundes

Hundehalter(in):

Name:

Vorname:

Anschrift:

Geburtsdatum:

Geburtsort:

Telefon:

Angaben zum Hund:

Wurftag/Alter: /

Rasse:

Größe/Gewicht: /

Anschaffungstag:

Wie viele Hunde werden _____(Anzahl)
in ihrem Haushalt gehalten?

☐ Ich versichere, dass die von mir gemachten Angaben...

Ort, Datum

Abbildung C.2.: Formular für „Hund Anmelden“

D Beispielszenarien

Der Vorgang Hund anmelden, das im Kapitel 2.1.3 referenziert ist, befindet sich hier.

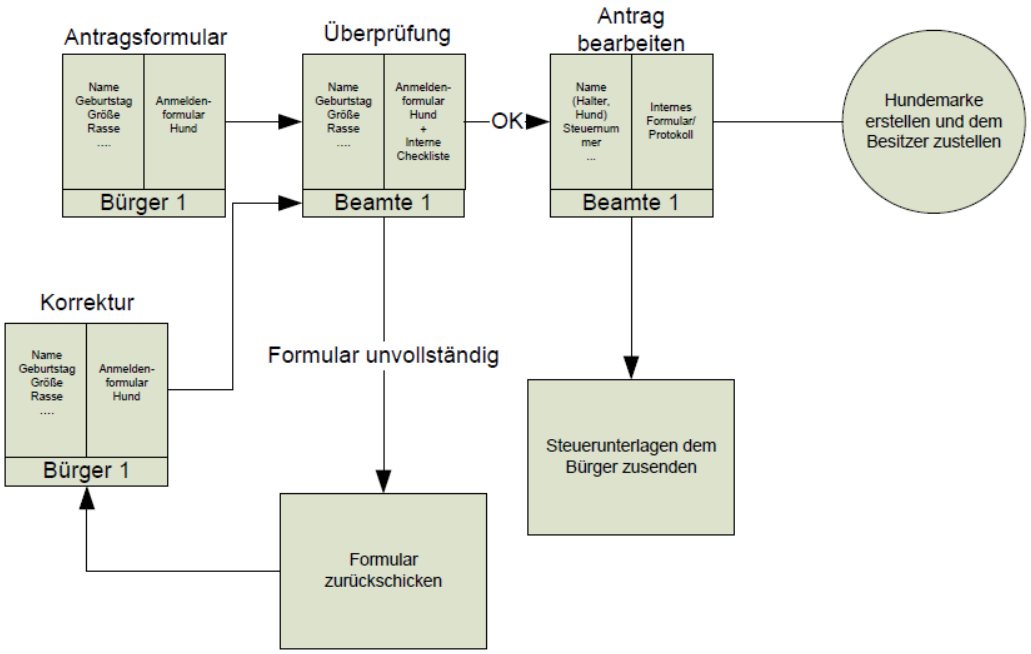


Abbildung D.1.: Hund anmelden

Hier findet sich die im Kapitel 2.1.2 referenzierte Abbildung eines Beschwerdevorgangs.

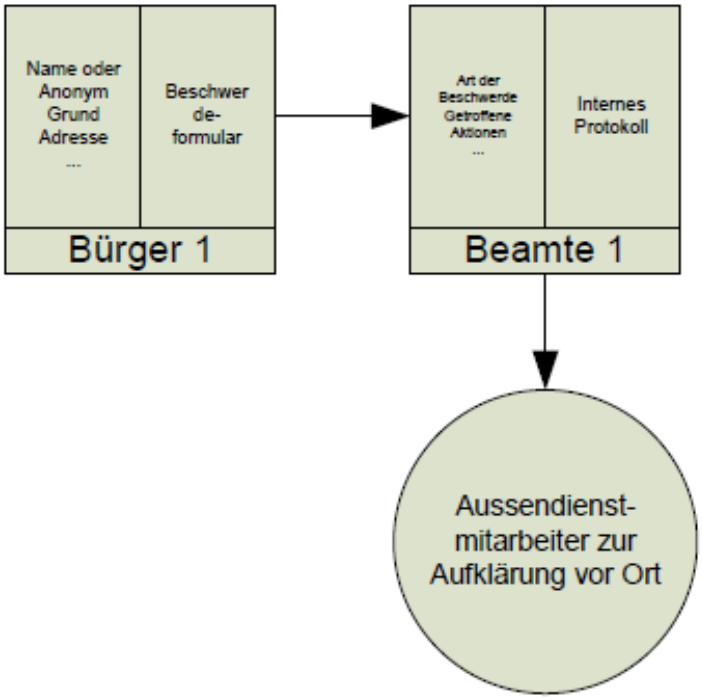


Abbildung D.2.: Beschwerde einreichen

Hier findet sich die im Kapitel 2.1 referenzierte Abbildung eines Briefwahl-Ablaufes.

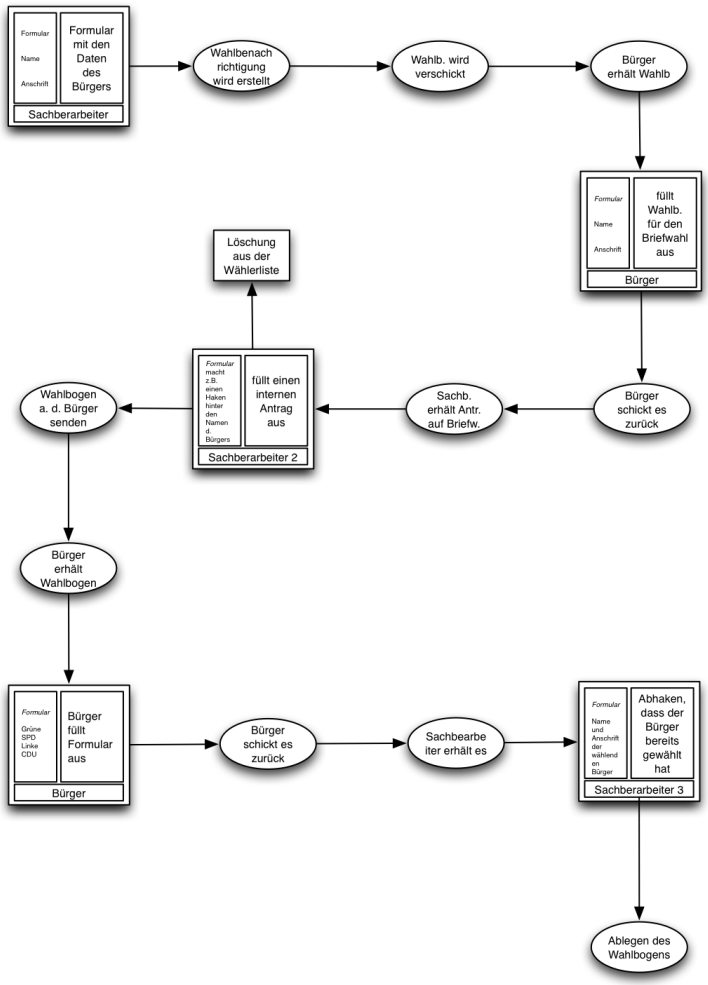


Abbildung D.3.: Briefwahl

E Portal-Dateien

Hier finden sich die im Kapitel 6 referenzierten ausführlichen Code-Listings.
Zu Kapitel 6.3.2:

```
<portlet-app xmlns='http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:schemaLocation='http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd' version='2.0'>
  <portlet>
    <description>Portlet zur Bearbeitung der Hundanmeldung </description>
    <portlet-name>HundAnmeldenSachbearbeiterPortlet </portlet-name>
    <display-name>Bearbeitung der Hundanmeldung </display-name>
    <portlet-class>sports.portlets.HundAnmeldenSachbearbeiterPortlet </portlet-class>
    <expiration-cache>0 </expiration-cache>
    <supports>
      <mime-type>text/html </mime-type>
      <portlet-mode>VIEW </portlet-mode>
    </supports>
    <resource-bundle>sports.portlets.messages </resource-bundle>
    <portlet-info>
      <title>Bearbeitung der Hundanmeldung durch einen Sachbearbeiter </title>
      <short-title>HundAnmeldenSachbearbeiterPortlet </short-title>
    </portlet-info>
    <supported-processing-event>
      <qname>sports.hundanmeldenauswahlevent </qname>
    </supported-processing-event>
  </portlet>

  <event-definition>
    <qname>sports.hundanmeldenauswahlevent</qname>
  </event-definition>
</portlet-app>
```

Quellcode E.1: portlet-Deskriptor

Zu Kapitel 6.3.4:

```
<!--Unterseite StammdatenBuerger-->
    <deployment>
        <if-exists>overwrite</if-exists>
        <instance>
            <instance-id>StammdatenBuergerPortletInstance</instance-id>
            <portlet-ref>StammdatenBuergerPortlet</portlet-ref>
        </instance>
    </deployment>
    <deployment>
        <if-exists>overwrite</if-exists>
        <instance>
            <instance-id>StammdatenBuergerHilfePortletInstance</instance-id>
            <portlet-ref>HilfePortlet</portlet-ref>
            <preferences>
                <preference>
                    <name>page</name>
                    <value>StammdatenBuerger</value>
                </preference>
            </preferences>
        </instance>
    </deployment>
```

Quellcode E.2: portlet-instances-Deskriptor

Zu Kapitel 6.3.5:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE deployments PUBLIC
"-//JBoss Portal//DTD Portal Object 2.6//EN"
"http://www.jboss.org/portal/dtd/portal-object_2_6.dtd">
<deployments>
    <deployment>
        <parent-ref/>
        <if-exists>overwrite</if-exists>
        <portal>
            <portal-name>default</portal-name>
            <supported-modes>
                <mode>view</mode>
            </supported-modes>
            <properties>
                <property>
                    <name>layout.id</name>
                    <value>3columns</value>
                </property>
                <property>
                    <name>theme.id</name>
                    <value>sports</value>
                </property>
                <property>
                    <name>portal.defaultObjectName</name>
                    <value>Alle</value>
                </property>
            </properties>
            <security-constraint>
                <policy-permission>
                   <action-name>view</action-name>
                   <unchecked/>
                </policy-permission>
            </security-constraint>
```

Quellcode E.3: SportsPortal-object-Deskriptor Teil 1

```
<page>
  <page-name>Bürgerportal</page-name>
    <security-constraint>
        <policy-permission>
           <action-name>viewrecursive</action-name>
           <role-name>buerger</role-name>
        </policy-permission>
    </security-constraint>
    <window>
        <window-name>Willkommen</window-name>
        <instance-ref>WelcomePortletInstance</instance-ref>
        <region>center</region>
        <height>1</height>
    </window>
    <page>
        <page-name>Stammdaten</page-name>
        <properties>
            <property>
                <name>order</name>
                <value>1</value>
            </property>
```

```
        </properties>
        <window>
            <window-name>StammdatenBuergerPortletWindow</window-name>
            <instance-ref>StammdatenBuergerPortletInstance</instance-ref>
            <region>center</region>
            <height>1</height>
        </window>
        <window>
            <window-name>StammdatenBuergerHilfePortletWindow</window-name>
            <instance-ref>StammdatenBuergerHilfePortletInstance</instance-ref>
            <region>right</region>
            <height>1</height>
        </window>

    </page>
    <page>
        <page-name>BuergerPosteingang</page-name>
        <properties>
            <property>
                <name>order</name>
                <value>2</value>
            </property>
        </properties>
        <window>
            <window-name>BuergerPosteingangsPortletWindow</window-name>
            <instance-ref>BuergerPosteingangsPortletInstance</instance-ref>
            <region>center</region>
            <height>1</height>
        </window>
        <window>
            <window-name>HundAnmeldenEditierenPortletWindow</window-name>
            <instance-ref>HundAnmeldenEditierenPortletInstance</instance-ref>
            <region>center</region>
            <height>2</height>
        </window>
        <window>
            <window-name>BuergerPosteingangsHilfePortletWindow</window-name>
            <instance-ref>BuergerPosteingangsHilfePortletInstance</instance-ref>
            <region>right</region>
            <height>1</height>
        </window>
    </page>
    <page>
        <page-name>Beschwerde</page-name>
        <window>
            <window-name>BeschwerdePortletWindow</window-name>
            <instance-ref>BeschwerdePortletInstance</instance-ref>
            <region>center</region>
            <height>2</height>
        </window>
        <window>
            <window-name>BeschwerdeHilfePortletWindow</window-name>
            <instance-ref>BeschwerdeHilfePortletInstance</instance-ref>
            <region>right</region>
            <height>1</height>
        </window>
        <window>
            <window-name>BeschwerdeHinweisPortletWindow</window-name>
            <instance-ref>BeschwerdeHinweisPortletInstance</instance-ref>
            <region>right</region>
            <height>2</height>
        </window>
    </page>
</page>
```

Quellcode E.4: SportsPortal-object-Deskriptor Teil 2

Zu Kapitel 6.4.2:

```
public void doView(RenderRequest request,RenderResponse response) throws PortletException,IOException {
    response.setTitle("Anmeldung eines Hundes");
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String submitted = request.getParameter("submitted");
    if (submitted != null && submitted.equals("true")){
        out.write("Ihre Hundeanmeldung wurde übermittelt und befindet sich " +
                "nun in Bearbeitung.<br>\n" +
                "Vielen Dank für die Nutzung unseres Online-Portals.");
        out.write("<hr><br>\n");
        out.write("<form method=\"POST\" action=\""+response.createActionURL()+"\">\n");
        out.write("<input type=\"hidden\" name=\"new\" value=\"new\">\n");
        out.write("<input type=\"submit\" value=\"Eine weitere Hundeanmeldung einreichen\">\n");
    }else{
        HashMap properties = new HashMap();
        if (request.getRemoteUser() == null) {
            properties.put("user", "anonym");
        } else {
            properties.put("user", request.getRemoteUser());
        }
        properties.put("role", "buerger");
        properties.put("type", "1");

        String antwort = UrlRequest.urlGETRequest("http://ralph.cs.tu-dortmund.de:8080/
~~~~~~~~~~~~~~~~hundanmelden_buerger/?orbeon-embeddable=true", properties);
        out.write(antwort);
    }
    out.close();
}
```

Quellcode E.5: doView()-Methode eines Formularportlets für Bürger

Zu Kapitel 6.4.4:

```
public void processAction(ActionRequest request, ActionResponse response) throws PortletException,IOException {
    try {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(request.getPortletInputStream());
        NodeList ids = doc.getElementsByTagName("process_id");
        NodeList result = doc.getElementsByTagName("result");
        if(ids.getLength() > 0) {
            if(result.getLength() > 0){
                if (result.item(0).getTextContent().equals("korrektur")){
                    DataBase.dbUpdate("UPDATE process SET process_state_id=5 WHERE
                            id="+ids.item(0).getTextContent());
                }
                if (result.item(0).getTextContent().equals("angenommen")){
                    DataBase.dbUpdate("UPDATE process SET process_state_id=9 WHERE
                            id="+ids.item(0).getTextContent());
                }
                if (result.item(0).getTextContent().equals("abgelehnt")){
                    DataBase.dbUpdate("UPDATE process SET process_state_id=10, completed=true
                            WHERE id="+ids.item(0).getTextContent());
                }
            }
            String temp = null;
            response.setRenderParameter("HundAnmeldenevent", temp);
```

```
22            }
23        } catch(Exception e) {
24            e.printStackTrace();
25        }
26    }
```

Quellcode E.6: processAction()-Methode eines Formularportlets für Sachbearbeiter

F Buildscript für Ant

Hier befindet sich das in Kapitel 7.6 beschriebene Ant-Skript.

```
<project name="SportsPortal">
    <property name="lib.dir" value="lib"/>
    <path id="classpath">
        <fileset dir="\${lib.dir}" includes="**\*.jar"/>
    </path>

    <target name="clean">
        <delete dir="build"/>
        <delete dir="..\output"/>
        <delete file="SportsPortal.war"/>
    </target>

    <target name="init" depends="clean">
        <mkdir dir="build"/>
        <mkdir dir="..\output"/>
    </target>

    <target name="generate" depends="init">
      <java jar="..\generieren.jar" fork="true"/>
    </target>

    <target name="move" depends="generate">
      <copy todir="java\sports\portlets">
        <fileset dir="..\output">
          <exclude name="**\*.xml"/>
        </fileset>
      </copy>
      <copy todir="web\WEB-INF">
        <fileset dir="..\output">
          <exclude name="**\*.java"/>
        </fileset>
      </copy>
    </target>

    <target name="compile" depends="move">
        <javac srcdir="java\" destdir="build" classpathref="classpath" target="1.5"/>
    </target>

    <target name="war" depends="compile">
        <war destfile="SportsPortal.war" webxml="web\WEB-INF\web.xml">
            <zipfileset dir="web\images" prefix="images"/>
            <fileset file="web\portal_style.css"/>
            <classes dir="build"/>
            <webinf dir="web/WEB-INF"/>
        </war>
    </target>

    <target name="deploy" depends="war">
      <scp file="SportsPortal.war" todir="root:XXX@ralph.cs.uni-dortmund.de:/usr/local/
          jboss/server/default/deploy" trust="yes"/>
    </target>
</project>
```

Quellcode F.1: Build-Script für das SportsPortal

Literaturverzeichnis

[1] Vorstellung der PG SPorTs auf den Seiten des LS 14
URL: http://ls14-www.cs.tu-dortmund.de/main/index.php?option=com_content&view=article&id=29%3Aprojektgruppe-sports&catid=3%3Aam-ls-14&Itemid=10 (zuletzt abgerufen: 27. April 2010)

[2] XMPP Standards Foundation Homepage
URL: http://xmpp.org/ (zuletzt abgerufen: 27. April 2010)

[3] E-Government 2.0 - Das Programm des Bundes
URL:http://www.cio.bund.de/cae/servlet/contentblob/63262/publicationFile/6989/egov2_programm_des_bundes_download.pdf
(zuletzt abgerufen: 27. April 2010)

[4] Der Beauftragte der Bundesregierung für Informationstechnik - Glossar
URL:http://www.cio.bund.de/cln_093/DE/Service/Glossar/glossar_node.html
(zuletzt abgerufen: 27. April 2010)

[5] Speyere Definition von Electronic Governmena
URL:http://foev.dhv-speyer.de/ruvii/Sp-EGov.pdf
(zuletzt abgerufen: 27. April 2010)

[6] Die Rolle elektronischer Behördendienste für die Zukunft Europas
URL:http://ec.europa.eu/information_society/eeurope/2005/doc/all_about/egov_communication_de.pdf
(zuletzt abgerufen: 27. April 2010)

[7] *Eclipse Graphical Modeling Framework*
http://www.eclipse.org/modeling/gmf/
(zuletzt abgerufen: 27. April 2010)

[8] *ORYX Editor*
http://bpt.hpi.uni-potsdam.de/Oryx
(zuletzt abgerufen: 27. April 2010)

[9] *BizAgi Process Modeler*
http://www.bizagi.com/
(zuletzt abgerufen: 27. April 2010)

[10] *yEd Graph Editor*
http://www.yworks.com/en/products_yed_about.html
(zuletzt abgerufen: 27. April 2010)

[11] *XPDL* Spezifikation
http://www.wfmc.org/xpdl.html
(zuletzt abgerufen: 27. April 2010)

[12] *Enhydra JaWE*
http://www.enhydra.org/workflow/jawe/index.htmll
(zuletzt abgerufen: 27. April 2010)

[13] „Introduction to BPMN“, Stephen A. White, *IBM Corporation*
http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf
(zuletzt abgerufen: 27. April 2010)

[14] *Object Management Group*
http://www.omg.org/
(zuletzt abgerufen: 27. April 2010)

[15] *Open Business Engine*
http://obe.sourceforge.net/
(zuletzt abgerufen: 27. April 2010)

[16] *Bonita Open Solution*
http://www.bonitasoft.com/
(zuletzt abgerufen: 27. April 2010)

[17] *XML Object Model*
http://www.xom.nu/
(zuletzt abgerufen: 27. April 2010)

[18] IBM WebSphere Portal Server
URL: http://www-01.ibm.com/software/de/websphere/portal/
Abrufdatum: 15. März 2010

[19] Exoplatform
URL: http://www.exoplatform.com
Abrufdatem: August 2009

[20] JBoss Portal Server
URL: http://www.jboss.de/products/jbossportal
Abrufdatem: August 2009

[21] IBM Rational Application Developer
URL: http://www-01.ibm.com/software/de/rational/design.html

[22] Eclipse Portalpack Plugin
URL: https://eclipse-portalpack.dev.java.net/

[23] Netbeans Portal Pack
URL: http://portalpack.netbeans.org/

[24] Sun Microsystems: *JSR 168: Java Portlet Specifikation*, 2003
URL: http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html
Abrufdatum: 11. Februar 2009

[25] Sun Microsystems: *JSR 286: Java Portlet Specifikation 2.0*, 2008
URL: http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html
Abrufdatum: 5. März 2009

[26] IETF RFC 2616 - HTTP 1.1 Spezifikation
URL: http://tools.ietf.org/html/rfc2616
Abrufdatum: 9. April 2010

[27] Sun Microsystems: *JSR 315: Java Servlet 3.0 Specifikation*, 2009
URL: http://jcp.org/aboutJava/communityprocess/final/jsr315/index.html
Abrufdatum: 30. Februar 2010

[28] Apache Ant Project
URL: http://ant.apache.org/
Abrufdatum: 15. März 2010

[29] Favicon HowTo des W3C URL: http://www.w3.org/2005/10/howto-favicon
Abrufdatum: 9. April 2010

[30] Orbeon Forms Homepage
URL: http://www.orbeon.com/
(zuletzt abgerufen: 27. April 2010)

[31] Orbeon Forms Wiki
URL: http://wiki.orbeon.com/forms/welcome
(zuletzt abgerufen: 27. April 2010)

[32] Homepage von PostgreSQL: The world's most advanced open source database
URL: http://www.postgresql.org/
(zuletzt abgerufen: 27. April 2010)

[33] Homepage von MySQL: The world's most popular open source database
URL: http://www.mysql.com/?bydis_dis_index=1
(zuletzt abgerufen: 27. April 2010)

[34] Englischer Wikipedia Artikel zu AJAX
URL: http://en.wikipedia.org/wiki/Ajax_%28programming%29
(zuletzt abgerufen: 27. April 2010)

[35] Deutscher Wikipedia Artikel zu AJAX
URL: http://de.wikipedia.org/wiki/Ajax_%28Programmierung%29
(zuletzt abgerufen: 27. April 2010)

[36] Homepage der AJAX-Community
URL: http://www.ajax-community.de/
(zuletzt abgerufen: 27. April 2010)

[37] Entwurf der Spezifikation für XPL (XML Pipeline Language)
URL: http://www.w3.org/Submission/xpl/
(zuletzt abgerufen: 27. April 2010)

[38] XForms-Spezifikation
URL: http://www.w3.org/MarkUp/Forms/,
zuletzt abgerufen: 27. April 2010

[39] XForms 1.1 - W3C Recommendation 20 October 2009
URL: http://www.w3.org/TR/2009/REC-xforms-20091020/,
zuletzt abgerufen: 27. April 2010

[40] XForms-Unterstützung durch OpenOffice.org
URL: http://wiki.services.openoffice.org/wiki/Documentation/OOoAuthors_User_Manual/Writer_Guide/XForms,
zuletzt abgerufen: 27. April 2010

[41] ubiquity-xforms - XForms in Web Browsers and presentational Ajax libraries
URL: http://code.google.com/p/ubiquity-xforms/,
zuletzt abgerufen: 27. April 2010

[42] XForms Accessibility - Mozilla Developer Center
URL: https://developer.mozilla.org/en/Accessibility/XForms,
zuletzt abgerufen: 27. April 2010

[43] Orbeon Forms User Guide
URL: http://www.orbeon.com/orbeon/doc/,
zuletzt abgerufen: 27. April 2010

[44] Orbeon Form Builder
URL: http://orbeon.com/orbeon/fr/orbeon/builder/summary,
zuletzt abgerufen: 27. April 2010

[45] L. Srinivasan and J. Treadwell: An Overview of Service-oriented Architecture, Web Services and Grid Computing

[46] Michael Papazoglou: An Overview of Service-oriented Architecture, Web Services and Grid Computing

[47] Anura Guruge : Web Services: Theory and Practice

[48] JBoss-WS
URL: http://www.jboss.org/jbossws

[49] JBossWS - JAX-WS Tools
URL: http://community.jboss.org/wiki/JBossWS-JAX-WSTools

[50] SOAP Version 1.2
URL: http://www.w3.org/TR/soap12-part1/

[51] WSDL Version 2.0
URL: http://www.w3.org/TR/wsdl20/

[52] UDDI Version 3.0.2
URL: http://www.uddi.org/pubs/uddi-v3.htm/

[53] Apache Axis2/Java Version 1.5.1
URL: http://ws.apache.org/axis2/

[54] JAX-WS 2.0/2.1 (JSR 224)
URL: https://jax-ws.dev.java.net/

[55] Martin Kalin : Java Web Services: Up and Running

[56] Spezifikation: WS-BOEL Extension for People (BPEL4People)
URL: http://tinyurl.com/6sacto