XUBetrieb

ekomax - elektronische Übertragung von kommunalen Abwasserdaten mit XML

0.9.9 Ausgabe

Veröffentlicht 2011-11-30

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Beispiele

Kapitel 1. Einleitung

1.1. XUBetrieb

Der Standard XUBetrieb wird im Rahmen des Projekts "XML Repository Betriebliche Stamm- und Berichtsdaten als Teil des XÖV des Bundes" entwickelt.
Dieses Projekt sieht vor, dass die Tauglichkeit des erstellten Metamodells durch die Anwendung auf eine konkrete Umweltberichtspflicht unter Beweis gestellt wird, indem eine Referenzimplementierung vorgenommen wird. Die ausgewählte Umweltberichtspflicht sollte bereits durch eine möglichst frei verfügbare, schreibende und eine frei verfügbare, lesende Anwendung unterstützt werden.
In Abstimmung mit dem Auftraggeber wurde die Umweltberichtspflicht nach EU Kommunalabwasser-Richtlinie 91/271/EWG ausgewählt. Das existierende, frei verfügbare Software-System e-Kommunalabwasser (https://apps.enda.eu/e-kommu) unterstützt die Erfüllung dieser Umweltberichtspflicht lesend und schreibend.

1.2. ekomax

Das Akronym ekomax steht für elektronische Übertragung von kommunalen Abwasserdaten mit XML. Das Software-System ekomax ist ein von e-Kommunalabwasser unabhängiges System, das e-Kommunalabwasser die notwendigen Funktionen zum Lesen und Schreiben von XUBetrieb über eine REST API zur Verfügung stellt.
Zur Entwicklung des Software-Systems ekomax wurden Apache Maven als Konfigurationsmanagementsystem und JBoss Application Server als Laufzeitumgebung verwendet.

Voraussetzungen für den Betrieb

- PostGIS Datenbank e-kommu
  http://enda.eu/e-kommu
- JBoss Application Server 5.1
  http://www.jboss.org/jbossas/docs/5-x

Kapitel 2. Analyse

In der folgenden Abbildung sind die Anwendungsfälle dargestellt, die das Software-System ekomax erfüllen soll.

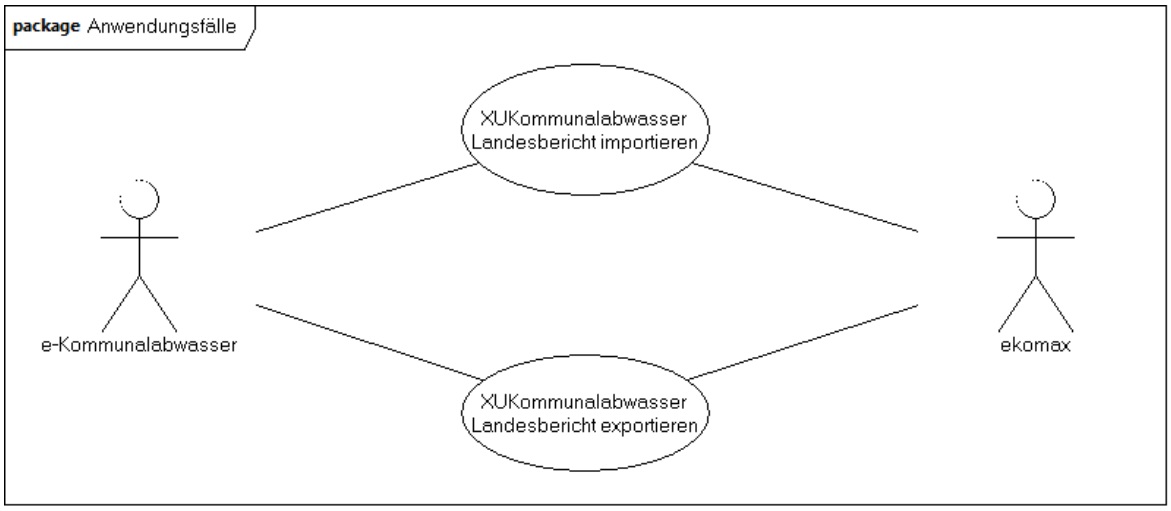

Abbildung 2.1. UML-Diagramm ekomax Anwendungsfälle

2.1. Anwendungsfall - XUKommunalabwasser Landebericht importieren

Das Software-System e-Kommunalabwasser erhält eine Nachricht im Format XUKommunalabwasser. Diese Nachricht leitet es an ekomax weiter. Dort wird die Nachricht empfangen, in das Datenmodell von e-Kommunalabwasser transformiert und anschließend in die Datenbank e_kommu geschrieben.

2.2. Anwendungsfall - XUKommunalabwasser Landebericht exportieren

Es besteht der Bedarf, für einen festgelegten Berichtszeitraum die Kommunalabwasserdaten eines Bundeslandes im Format XUKommunalabwasser bereitzustellen. Das Software-System ekomax nimmt die Anfrage entgegen, lädt die notwendigen Daten aus der Datenbank e_kommu, übersetzt sie in das Format XUBetrieb und übermittelt diese Nachricht im Anschluß an den Anfragenden.

Kapitel 3. Entwurf

Die Übertragung der Daten zwischen den beiden Systemen soll über ein Netzwerk erfolgen. Die beiden Systeme sollen über wenige Schnittstellen interoperieren. In der Folge sind die beiden Systeme lose gekoppelt, so dass sich Änderungen an den Komponenten des einen Systems nicht auf das andere System auswirken. Änderungen des einen Systems haben nur dann Auswirkungen auf das andere System, wenn die Änderungen die gemeinsame Schnittstelle betreffen. In der folgenden Abbildung ist dieser Lösungsansatz skizziert.

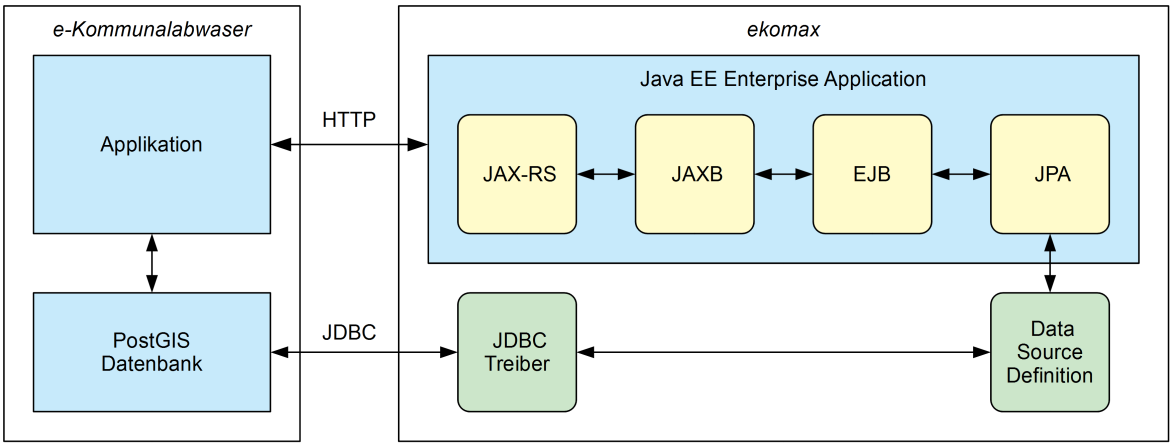

Abbildung 3.1. Entwurfsskizze

3.1. Message Translator

Das Software-Entwurfsmuster Message Translator definiert eine Lösung für die Kommunikation zwischen Software-Systemen, die unterschiedliche Datenformate verwenden. Den Software-Systemen liegt ein individuelles Datenmodell zu Grunde, das beispielsweise die Gestaltung der Datenbank beschreibt. Zur Interaktion mit anderen Systemen werden häufig Nachrichtenformate verwendendet, die unabhängig von spezifischen Datenformaten sind (z. B. EDIFACT oder UBL). Die Software-Systeme benötigen zur Verarbeitung solcher Nachrichten eine Repräsentation der Nachrichten, die mit ihrem internen Datenformat kompatibel ist. Damit die Software-Systeme Nachrichten miteinander austauschen können, verwenden sie einen Message Translator, der aus dem eigenen Datenformat in das Zieldatenformat sowie aus einem Quelldatenformat in das eigenen Datenformat übersetzt. Diese Übersetzung wird als Transformation und mit Mappings realisiert, die jeweils die Abbildung der Daten aus dem Quell- in das Zieldatenformat festlegen.

3.2. REST

Das Akronym REST (Representational State Transfer) bezeichnet ein Programmierkonzept, bei dem die Ergebnisse eines von einem Server angebotenen Dienstes als Resourcen unter einer eindeutigen Adresse zur Verfügung gestellt werden. Die eindeutigen Adressen der Ressourcen können nach den Vorgaben des Standards Uniform Resource Identifier (URI) gebildet werden. Die unter einer URI zugänglichen Ressourcen können in Abhängigkeit von der Anfrage des Clients unterschiedliche Repräsentationen (z. B. HTML oder XML) haben. Weiterhin sieht das REST Konzept vor, dass für jede Ressource eine klar definierte Menge von Operationen definiert wird. Für das Protokoll HTTP sind diese Aktionen in der nachfolgenden Liste dargestellt.

Aktionen bei REST über HTTP

- GET - die Ressource wird übermittelt.
- POST - der Server erstellt eine neue Ressource. Da noch keine URI für zu erstellende Ressource existiert, wird die URI die übergeordnete Ressource adressiert. Als Ergebnis wird die URI der neuen Ressource an den Client übermittelt.
- PUT - die Ressource wird angelegt oder modifiziert.
- DELETE - die Ressource wird gelöscht.
- HEAD - die Metadaten der Ressource werden übermittelt.
- OPTIONS - die Operation, die für die Ressource verfügbar werden übermittelt.

Das REST Konzept sieht ein zustandsloses Protokoll vor. Jede Nachricht enthält alle Informationen, die notwendig sind, um diese Nachricht richtig zu interpretieren. Daher muss weder der Cleint noch der Server Zustandsinformationen zwischen zwei Nachrichten speichern. Bei der Verwendung des HTTP-Protokolls können Informationen über Zugriffsart (Operation), Repräsentation (Format) und Authentifizierung im Header des Request übermittelt werden.

Kapitel 4. Architektur

4.1. Software-System-Architektur

Die Struktur eines Software-Systems kann mit einer Architektur beschrieben werden. Software-System-Architekturen verwenden häufig ein Schichtenmodell als Strukturierungsprinzip. In einem Schichtenmodell werden die Funktionen eines Systems isolierten Bereichen (Schichten) zugeordnet.
Die Client-Server-Architektur besteht aus zwei Schichten, die ein verteiltes Software-System beschreiben. Das System besteht aus einem Netzwerk von Clients und Servern. Ein Server ist ein Programm, das einen Dienst anbietet. Ein Client ist ein Programm, das einen von einem Server angebotenen Dienst nutzt. Der Client fordert einen Dienst aktiv beim Server an, der Server ist passiv und reagiert auf die Kontaktaufnahme von Clients. Die Regeln der Kommunikation zwischen Client und Server für einen Dienst werden durch ein für jeden Dienst spezifisches Protokoll (z. B. HTTP) festgelegt.
Webanwendungen sind Software-Systeme, bei denen als Client ein Webbrowser benutzt wird. Webanwendungen können mit einem Vier-Schichten-Modell, bestehend aus Präsentations-, Web-, Verarbeitungs- und Datenhaltungsschicht beschrieben werden. In der folgenden Abbildung ist diese Architektur dargestellt.

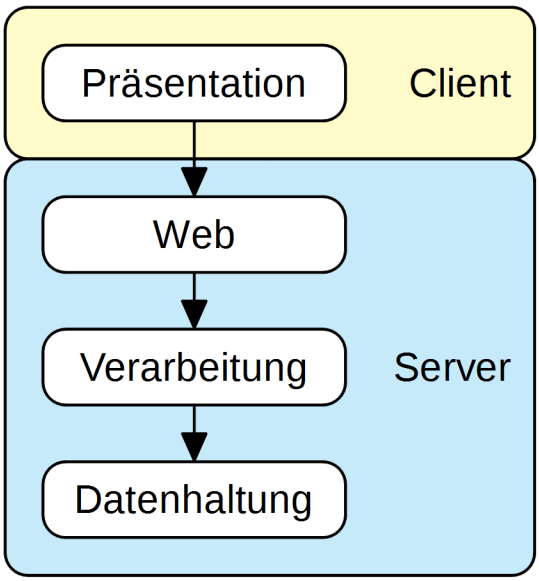

Abbildung 4.1. Vier-Schichten-Architektur einer Webanwendung

4.1.1. Präsentationsschicht

Die Präsentationsschicht dient der Interaktion mit dem Benutzer. Die Benutzerinteraktion erfolgt mittels eines Webbrowsers unter Verwendung des HTTP-Protokolls. Über das Protokoll werden Anfragen gestellt und Antworten empfangen. Der Webbrowser stellt die Anworten dar. Diese Antworten werden von der Web-Schicht des Servers dynamisch generiert. Da kein direkter Zugriff auf die Verbeitungs- oder Datenhaltungsschicht möglich ist, wird ein solcher Client als "thin client" bezeichnet. Die Funktionen von "thin clients" sind auf die Benutzerinteraktion, das Aufrufen bestimmter Inhalte und die Übermittlung bestimmter Aufgaben an den Server beschränkt. Die Ausführung der Aufgaben gemäß der Verarbeitungslogik ist dem Server vorbehalten.

4.1.2. Webschicht

Die Webschicht besteht aus Komponenten, die der Interaktion mit der Präsentations- und der Verarbeitungsschicht dienen. Die Webschicht nimmt Anfragen der Präsentationsschicht entgegen, leitet sie an die Verarbeitungsschicht weiter, generiert Antworten nach Vorgabe der Verarbeitungsschicht und übermittelt diese an die Präsentationsschicht.

4.1.3. Verarbeitungsschicht

Die Verarbeitungsschicht führt die vom Benutzer des System angestoßenen Aktivitäten gemäß den Prozessen einer fachlichen Domäne aus. Den Komponenten der Verarbeitungsschicht stehen Systemdienste zur Verfügung, die z. B. die notwendigen Zugriffe auf die Datenhaltungsschicht ermöglichen.

4.1.4. Datenhaltungsschicht

Die Datenhaltungsschicht sorgt für die Verwaltung und dauerhafte (persistente) Speicherung von Daten.

4.2. Java Platform, Enterprise Edition (Java EE)

Die Java Technologie umfasst die objektorientierte Programmiersprache Java und mehrere Java-Plattformen, die Java-Programme ausführen können. Jede Java-Plattform besteht aus einer Java Virtual Machine (JVM) und einer Application Programming Interface (API). Eine API ist eine Sammlung von Schnittstellen, die verwendet werden können, um Software-Komponenten zu erstellen. Eine JVM ist ein Programm, das in Java geschriebene Programme auf einer festgelegten

Kombination von Hardware- und Software auführen kann. Java-Programme können auf jeder Plattform (Kombination von Hardware- und Software-Systemen) ausgeführt werden, für die eine JVM verfügbar ist. Die JVM stellt als "Betriebsystem" für Java-Programme die plattformübergreifende Portabilität (write once - run anywhere) sicher.
Die Java Platform, Standard Edition (Java SE) besteht im wesentlichen aus der Programmiersprache Java, der JAVA SE API und einer JVM.
Die Java Platform, Enterprise Edition (Java EE) basiert auf der Java SE und umfasst zusätzlich die Java EE API und eine Java EE Laufzeitumgebung. Java EE wurde mit dem Ziel entworfen, sichere, zuverlässige, skalierbare und verteilte Programme (Enterprise Applications) zu entwickeln und zu betreiben. Enterprise Applications sind dazu bestimmt, informationstechnische Probleme großer Institutionen zu lösen. Die Spezifikation stellt allgemein akzeptiert Problemlösungen zur Verfügung, um modulare, mehrschichtige und verteilte Software-Systeme mit der Programmiersprache Java entwickeln zu können.
Die Java EE Spezifikation wird innerhalb des Java Community Process (JCP) erarbeitet und der Öffentlichkeit frei verfügbar in Form eines Dokuments zusammen mit einer Referenzimplementierung zur Verfügung gestellt. Ziel ist es, klare Schnittstellen zu schaffen, die dafür sorgen, dass die Implementierungen verschiedener Hersteller interoperabel sind. Der JCP ist sowohl für die Weiterentwicklung der Programmiersprache Java als auch für die Weiterentwicklung der Java Plattformen (Java SE, Java EE, Java ME) verantwortlich. Innerhalb des JCP werden die Weiterentwicklungen in Form von Java Specification Requests (JSR) ausgearbeitet. Dabei wird der größte gemeinsame Konsens aller beteiligten Partner angestrebt.
Der JSR 244: Java Platform Enterprise Edition 5 (Java EE 5) definiert selbst keine APIs, vielmehr ist er eine abgestimmte Sammlung von APIs, die in anderen JSRs festgelegt sind.

Java EE 5 JSRs

- JSR 176: Java 2 Platform Standard Edition 5.0 (Java SE 5)
- JSR 77: J2EE Management
- JSR 88: J2EE Application Deployment
- JSR 112: J2EE Connector Architecture (JCA) 1.5
- JSR 154: Java Servlet 2.5
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0
- JSR 220: Java Persistence API (JPA)
- JSR 220: Enterprise JavaBeans (EJB) 3.0

- JSR 245: JavaServer Pages (JSP) 2.1
- JSR 250: Common Annotations for the Java Platform
- JSR 252: JavaServer Faces (JSF) 1.2
- JSR 907: Java Transaction API (JTA)
- JSR 914: Java Message Service API (JMS)
- JSR 925: JavaBeans Activation Framework (JAF) 1.1

4.2.1. API und SPI

Damit Java-Programme von einer Java EE Laufzeitumgebung auf eine andere überführt werden können, wurde eine strikte Trennung zwischen den Schnittstellen für die Java-Programme (Application Programming Interface, API) und den Schnittstellen für die Anbieter von Laufzeitumgebungen (Service Provider Interface, SPI) etabliert. Die Definition von zwei unterschiedlichen Sammlungen von Schnittstellen führt dazu, dass Java-Programme nur auf Grundlage der API und nicht gegen eine tatsächliche Implementierung entwickelt werden können. Die Anbieter von Implementierungen entwickeln ihre Laufzeitumgebungen auf der Grundlage des SPI. Dieses Konzept stellt sicher, dass Java-Programme ohne Änderungen am Quellcode in unterschiedlichen Laufzeitumgebungen verschiedener Anbieter ausgeführt werden können.

4.2.2. Laufzeitumgebung

Die Laufzeitumgebung für Java EE 5 ist ein Programm, das in einer JVM ausgeführt wird und den Zugriff auf die Ressourcen des zugrundeliegenden Betriebssystems (Dateisystem, Netzwerk etc.) kapselt. Weiterhin stellt die Laufzeitumgebung alle im JSR 244 festgelegten Dienste und Schnittstellen zur Verfügung. Häufig wird eine solche Laufzeitumgebung "Java EE Application Server" genannt. Die Konformität der Laufzeitumgebung zur Spezifikation der Plattform wird zertifiziert, so dass Java EE Komponenten auf Java EE Application Server unterschiedlicher Hersteller ausführbar sind.

Java EE 5 konforme Laufzeitumgebungen

- JBoss Application Server
- Oracle GlassFish Application Server
- IBM WebSphere Application Server
- SAP NetWeaver Application Server

4.2.3. Java EE Komponenten Container

Java EE basiert auf einem Komponenten-Container-Modell. Daher setzten sich Java EE Programme aus abgeschlossenen funktionalen Einheiten (Komponenten) zusammen. Jede dieser Komponenten wird in einem bestimmten Containertyp ausgeführt. Die Container stellt der Java EE Application Server bereit. Die Container stellen die von der Spezifikation festgelegten Schnitstellen und Dienste für die Komponenten zur Verfügung. Die Komponenten kommunizieren nicht direkt miteinander sondern über die Container, die Anfragen an andere Komponenten vermitteln. Darüber hinaus verwalten die Container die Komponenten während ihres gesamten Lebenszyklus und gewähren Ihnen Zugang zu anderen Diensten der Laufzeitumgebung. Zu den Diensten auf die die Komponenten zugreifen können zählen Autorisierung und Authentifizierung, Transaktionsmanagement, Threadmanagement sowie Zugriffe auf Ressourcen (Netzwerk- und Datenbankverbindungen). Die beiden wichtigsten Container sind der EJB Container und der Web Container.

4.2.4. Webanwendungen

In der folgenden Abbildung ist eine vierschichtige Java EE Webanwendung dargestellt, in der sowohl die Web- als auch die Verarbeitungsschicht von einem Java EE Application Server zur Verfügung gestellt wird.

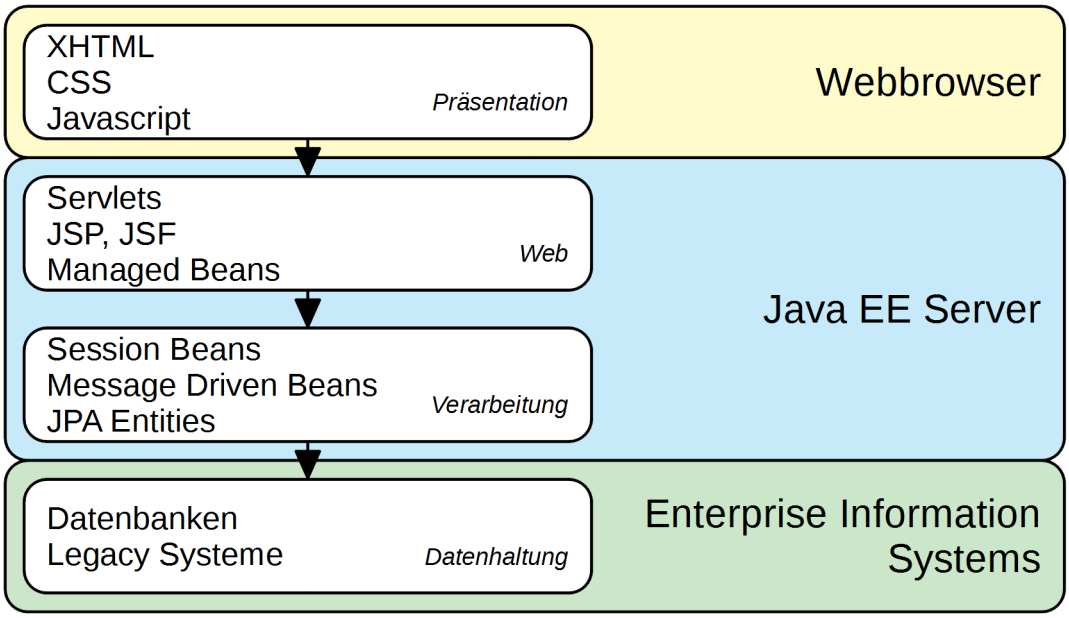

Abbildung 4.2. Vier-Schichten-Architektur Java EE

4.2.4.1. Präsentationsschicht

Der Anwender arbeitet mit einem Web-Client, der die von der Präsentationsschicht übermittelten Daten aufbereitet und so die graphische Benutzeroberfläche zur Verfügung stellt. In der Präsentationsschicht werden die folgenden Technologien verwendet: XHTML, CSS und JavaScript.

4.2.4.2. Webschicht

Die Komponenten der Webschicht steuern die Präsentationsschicht. Hierzu verarbeiten Servlets die Anfragen (Requests) der Präsentationschicht und generieren die Anworten (Responses) unter Zuhilfnahme eines View-Handlers und den Managed Beans genanten Komponenten, die temporäre Daten sowie den Zustand der Sitzung des Anwenders verwalten. Der View-Handler kann auf den Technologien JavaServer Pages (JSP) oder JavaServer Faces (JSF) basieren. Servlets, Managed Beans, JSP und JSF werden im Webcontainer des Java EE Application Server ausgeführt.

4.2.4.3. Verarbeitungsschicht

Die Komponenten der Verarbeitungsschicht sind Enterprise JavaBeans (EJBs). Eine EJB erhält Daten von der Webschicht, verarbeitet diese gemäß den Prozessen der fachlichen Domäne und greift hierzu ggf. auf Daten zu, die von der Datenhaltungschicht bereitgestellt werden. Die folgenden Technologien kommen in der Verarbeitungsschicht zum Einsatz: Enterprise JavaBeans (EJBs) und Java Persistence API (JPA) Entities.

4.2.4.4. Datenhaltungsschicht

Die Datenhaltungsschicht wird bei Java EE als Enterprise Information System (EIS) bezeichnet. Diese Schicht dient der Anbindung von externen Systemen, mit denen das Java EE Programm zusammenarbeiten soll oder zu deren Daten und Funktionen Abhängigkeiten bestehen. Neben Datenbanken können dies beispielsweise Legacy Systeme sein, die auf Mainframes ausgeführt werden. In der Datenhaltungsschicht werden die folgenden Technologien verwendet: Java Database Connectivity API (JDBC), Java Persistence API (JPA), J2EE Connector Architecture (JCA) und Java Transaction API (JTA).

4.2.5. Komponenten

4.2.5.1. Java Naming and Directory Service (JNDI)

Das Java Naming and Directory Interface (JNDI) definiert eine einheitliche Schnittstelle für den Zugriff auf Namens- und Verzeichnisdienste. Namens- und Verzeichnisdienste machen Dienste und Informationen auffindbar, die von unabhängigen Komponenten bereitgestellt werden. Werden von einer Java EE Komponente Dienste oder Daten benötigt, die von einer anderen Komponente bereitgestellt werden, so können diese über JNDI identifiziert werden. Namens- und Verzeichnisdienste ermöglichen es Objekte in Kontexthierarchien einzuordnen. Der Kontext wird dabei durch eine Anzahl von Bindungen zwischen Namen und Objekten gebildet und ermöglicht die Navigation durch die Hierarchie der Kontexte, die Suche und Auflistung von Objekten sowie das Erzeugen und das Löschen von Einträgen. Statt des Objekts kann auch eine Referenz auf dieses Objekt zurückliefert werden.
Ein Namensdienst bildet einen Namen auf ein damit assoziiertes Objekt ab, so dass eine Menge von Namen mit Objekten eines bestimmten Typs verbunden ist. Die Namen innerhalb eines Namensdienst folgen definierten Syntaxregeln (Naming Conventions). Namensdienste ermöglichen es, Objekte in komplexen, hierarchischen Strukturen abzulegen und über verständliche Namen wieder zu extrahieren.
Ein Verzeichnisdienst ist die Erweiterung eines Namensdienstes, die neben der Zuordnung von Namen zu Objekten, auch die Angabe von Attributen zum Objekt erlaubt. Jedem der Attribute können wiederum mehrere Werte zugeordnet sein können.
JNDI erfüllt eine zentrale Aufgabe innerhalb der Java EE Plattform, da erst durch die Auffindbarkeit von Daten und Diensten die Interoperation lose gekoppelter Java Programme und Komponenten möglich ist.

4.2.5.2. Java Database Connectivity API (JDBC)

Die Java Database Connectivity API (JDBC) ermöglicht die Verbindung mit Datenbanken. Mit JDBC als Transportprotokoll können Java Programme Daten in relationalen Datenbank erzeugen, lesen, aktualisieren und löschen. Durch die Verwendung von JDBC ist es möglich in der Verarbeitungsschicht SQL Befehle auszuführen. JDBC defniert nicht, welche SQL Kommandos übertragen werden dürfen und welche nicht, fordert aber, dass mindestens der SQL-2 Entry-Level-Standard von 1992 erfüllt wird. Da fast jede relationale Datenbank einen eigenen SQL Dialekt definiert, ist der Wechsel auf eine andere Datenbank häufig mit großem Aufwand verbunden.

4.2.5.3. Java Transaction API (JTA)

Die Java Transaction API (JTA) ermöglicht transaktionsorientierte Interaktionen mit Ressourcen. Sie besteht im Kern aus einem Transaktionsmanager, der von der Laufzeitumgebung zur Verwaltung von Transaktionen verwendet wird. Zum Beispiel handelt es sich um Dienste für Transaktionsabgrenzung und Synchronisierung. Mit JTA können Methodenaufrufe als Transaktion definiert und es kann festlegt werden, wann bestimmte Ereignisse (z. B. Commit und Rollback) stattfinden sollen. EJB Container nutzen die JTA zur Transaktionsverwaltung und ermöglichen es das Transaktionen in Komponenten der Verarbeitungsschicht verwendet werden können.

4.2.5.4. Java Persistence API (JPA)

Die Java Persistence API (JPA) definiert einen Satz von Schnittstellen, deren Signatur sich auch bei einer Weiterentwicklung oder dem Wechsel der SPI nicht ändert. JPA ermöglicht das Überführen von Java Objekten auf die Modelle relationaler Datenbanken (Object-Relational Mapping, ORM). Die Beschreibung der Tabellen und Relationen eines relationalen Modells erfolgt bei JPA mit Metadaten, die mithilfe von Annotationen beschrieben werden. Kern der JPA ist der EntityManager, der Objekte erzeugen, lesen, aktualisieren, löschen und suchen kann. Diese Objekte müssen nicht in einer relationalen Datenbank gespeichert werden, da die jeweilige SPI Befehle des EntityManager auf beliebige Speichermedien umsetzen kann. Als Abfragespreache wird die Java Query Language (JQL) verwendet, daher sind keine SQL-Befehle im Programmcode notwendig. Da JPQL den spezifischen SQL Dialekt von Datenbanken kapselt, ist die Portierung auf eine andere Datenbank mit geringem Aufwand verbunden als bei JDBC.

4.2.5.5. Java Message Service (JMS)

Der Java Message Service (JMS) unterstützt die nachrichtenbasierte Kommunikation, die den Prinzipien der Message Oriented Middleware (MOM) folgt. Den Kern der nachrichtenbasierten Kommunikation bildet der so genannte Message Broker. Der Message Broker nimmt Nachrichten des Senders entgegen und ist für die richtige Zustellung verantwortlich. MOM unterscheidet die beiden Kommunikationsmodelle "point to point" und "publish and subscribe". Bei point to point Kommunikation wird jede Nachricht einem festgelegten Empfänger zugestellt. Da der Empfänger häufig nur eine Nachricht auf einmal verarbeiten kann, bilden die Nachrichten eine Warteschlange (Queue). Bei publish and subscribe werden die Nachrichtem einer Gruppe von Empfängern zugestellt. Jede Empfängergruppe beschäftigt sich in der Regel mit genau

einem Thema (Topic). JMS unterstützt beide Kommunikationsmodelle, da jedoch nicht alle Implementierungen Queues und Topics unterstützen enthält die API jeweils zwei Ausprägungen aller Klassen und Schnittstellen, die von einer gemeinsamen Klasse abgeleitet wurden.

4.2.5.6. Java API for XML Processing (JAXP)

Mit der Java API for XML Processing (JAXP) wird die Verarbeitung von XML beschrieben. JAXP ermöglicht die Einbindung von unterschiedlicher XML Parsern (DOM, SAX und StAX). Darüber hinaus integriert JAXP die Transformation von XML Dokumenten mit XSLT.

4.2.5.7. Enterprise Java Beans (EJB)

Enterprise Java Beans (EJBs) sind Komponenten, die Funktionen zur verteilten, transaktionsorientierten Verarbeitung von Aufgaben, die sich aus den Anforderungen der Prozesse einer fachlichen Domäne ergeben, bereitstellen.
EJBs werden von einem EJB Container verwaltet, der auch ihren Lebenszyklus steuert. Der EJB Container bietet die Dienste der von ihm verwalteten EJBs über einen Namendienst an. Die Zugriffe auf die EJBs erfolgen nie direkt, sondern immer über den EJB Container. Das Zusammenspiel zwischen Container und EJB wird über einen Component Contract geregelt. Zur Anbindung der Datenhaltungsschicht werden EJB Connectoren verwendet, die ebenfalls innerhalb des EJB-Containers ausgeführt werden. So können Daten mittels JPA Transaktionen in Datenbanken gespeichert werden.
EJBs lassen sich folgenden Typen zuordnen: Stateless Session Beans, Stateful Session Beans und Message Driven Beans.
Stateless Session Beans arbeiten zustandslos, dass heißt nach Beendigung einer Transaktion wird ihr Status verworfen. Für den jeweiligen Kommunikationspartner sind Stateless Session Beans nicht voneinander unterscheidbar. Da auch für den EJB-Container alle Instanzen dieses Objektes identisch sind, werden sie von ihm über einen Pool verwaltet und für jede Transaktion eine beliebige Instanz der EJB bereitgestellt.
Stateful Session Beans sind in der Lage, ihren Zustand zu speichern. Der jeweilige Kommunikationspartner arbeitet stets mit der gleichen Instanz zusammen. Da sich der Zustand von Stateful Session Beans von Instanz zu Instanz unterscheidet, können diese bei einer Transaktion nicht mehr beliebig gegeneinander ausgetauscht werden.
Message Driven Beans basieren auf dem asynchronen Nachrichtenaustausch via JMS. Sie werden mit Hilfe einer ConnectionFactory über den Namensdienst zur Verfügung gestellt und empfangen Anfragen in Form von Nachrichten. Der EJB

Container ermöglicht die parallele Nachrichtenverarbeitung, indem er mehrere Instanzen einer Message Driven Bean einsetzt.
Die Steuerung und die Konfiguration von EJBs durch deklarative Anweisungen ermöglicht es, eine EJB einer bestehenden Anwendung hinzuzufügen, ohne den Quelltext zu ändern oder sie erneut zu kompilieren.

4.2.5.8. Managed Beans

Managed Beans verbinden Komponenten in der Webschicht mit Komponenten in der Verarbeitungsschicht (Backing Beans). Sie verwalten die Daten des Komponentenmodells der zu generierenden Benutzeroberfläche und stellen Methoden zur Verfügung, mit denen Daten und Ereignisse der Interaktion mit dem Benutzer verarbeitet werden können.

4.2.5.9. JavaServer Pages (JSP)

JavaServer Pages (JSP) sind textbasierte Dokumente, die als Servlets ausgeführt werden. JSP erlaubt es Java Code und spezielle JSP Aktionen in HTML- oder XML-Seiten einzubetten. Diese JSP Aktionen werden in so genannten Tag Bibliotheken als Erweiterung von HTML bzw. XML definiert. JSP wird unter Verwendung eines speziellen Compilers in Java Bytecode umgewandelt und können anschließend in der Websschicht ausgeführt, um die Benutzeroberfläche für die Präsentationsschicht zu erzeugen.

4.2.5.10. JavaServer Faces (JSF)

Mit JavaServer Faces (JSF) können Benutzeroberflächen definiert werden. JSF stellt eine API zur Erzeugung und Verwaltung von Komponenten der Benutzeroberflächen und einer Bibliothek vordefinierter Komponenten zur Verfügung. Die Benutzeroberfläche wird abstrakt aus einzelnen Komponenten in Form einer Baumstruktur durch JSF Tags deklariert. Der Komponentenbaum wird von einem Render Kit in die gewünschte Darstellungsform (z. B. HTML) überführt. Wobei für jede Komponente eine spezielles Renderobjekt verwendet wird, in dem die Darstellung der Komponente festgelgt ist.
Neben den Komponenten zur Deklaration der Benutzeroberfläche stellt JSF Funktionen zur Navigation, Zustandsverwaltung der Komponenten, Verarbeitung von Ereignissen, Validierung von Benutzereingaben, Konvertierung von Daten sowie der Einbindung der Verarbeitungsschicht bereit.
JSF verarbeitet Anfragen der Benutzer nach dem in der folgenden Liste dargestellten Lebenszyklus, der durch eine initiale Anfrage des Benutzers gestartet wird.

Request Processing Lifecycle (RPL)

- Restore View
- Apply Request Values
- Process Validations
- Update Model Values
- Invoke Application
- Render Response

In den Phasen werden durch Benutzereingaben verursachte Ereignisse verarbeitet, Daten validiert, Funktionen der Verarbeitungsschicht aufgerufen, der Komponentenbaum aktualisiert, die Benutzeroberfläche für die Präsentationsschicht generiert, die Antwort an den Client gesendet und schließlich der neue Zustand des Komponentenbaums für die nächste Ausführung des Lebenszyklus gespeichert.
Die Vorteile von JSF liegen primär in der komponentenorientierten Definition von Benutzeroberflächen für Webanwendungen. Die Komponenten sind konfigurierbar, erweiterbar und wiederverwendbar. Ein weiterer Vorteil von JSF ist die Unabhänigkeit von einem bestimmten Render Kit, da der Komponentenbaums unabhänig von einer Darstellungsform oder von einem bestimmten View Handler ist.

4.2.5.11. Unified Expression Language (EL)

Die Unified Expression Language (EL) ermöglicht es, Eigenschaften und Methoden der Managed Beans mit den Definitionen der von der Webschicht zu generierenden Päsentation zu verbinden. Hierzu stellt die EL Value Expressions und Method Expressions zur Verfügung. Die EL kann sowohl mit JSP als auch mit JSF verwendet werden.

4.3. JBoss Seam Application Stack

Der JBoss Seam Application Stack ist ein Framework für die Entwicklung von Webanwendungen auf der Basis der Java EE 5 Spezifikation. Die bestehenden Java EE 5 Spezifikation werden durch Seam zu einer in sich homogenen Sammlung von ineinandergreifenden Konzepten und Technologien zusammengefügt. Die Integration von JPA, EJB und JSF wird von Seam durch die Erweiterung der Spezifikationen von Java EE 5 verbessert. Seam greift in den JSF Lebenszyklus ein und nutzt die Erweiterungsfähigkeit von EL um neue Möglichkeiten der Kommunikation zwischen Komponenten einzuführen. Die Verwendung von Metadaten (Annotationen)

wird auf Plattformebene ausgeweitet und primär das Prinzip Konfiguration als Ausnahme (configuration by exception) verfolgt. Darüber hinaus integriert Seam weitere Technologien, wie zum Beispiel jBPM für das Ausführen und Verwalten von Geschäftsprozessen, Drools für das Ausführen und Verwalten von Geschäftsregeln sowie Richfaces für die AJAX Funktionalitäten in der Benutzeroberfläche.
Webanwendungen, die auf Seam basieren, implementieren das Entwurfsmuster Model-View-Controller (MVC). Dieses Entwurfsmuster verfolgt das Ziel, Präsentations-, Verarbeitungs- und Datenhaltungschicht zu trennen. Hierdurch werden die Wartbarkeit und die Flexibilität des Software-Systems gesteigert, da die verschiedenen Sichten unabhängig voneinander austauschbar sind. MVC organiseirt ein Programm unter den drei Aspekten Model (Datenhaltung), View (Präsentation) und Controller (Verarbeitung). Der Controller steht als Vermittler zwischen dem Model und der View. Das Model repräsentiert die eigentliche Kernfunktionalität sowie den internen Zustand der Anwendung. Der Controller nimmt die im View generierten Anforderungen des Benutzers entgegen und übersetzt sie in Nachrichten an das Model. Die View generiert die Benutzeroberfläche der Anwendung und greift auf die vom Model verwalteten Daten zu.

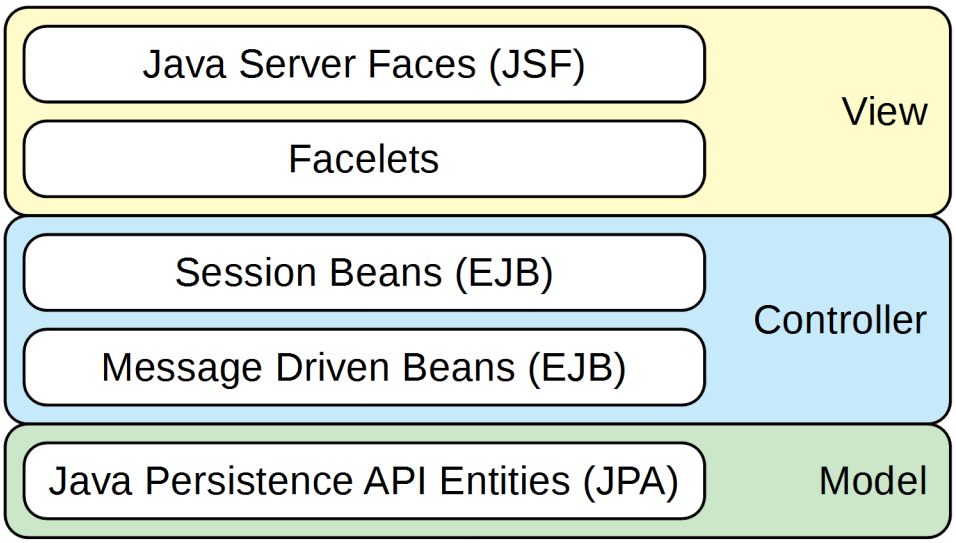

Abbildung 4.3. Model-View-Controller Pattern

4.3.1. Facelets

Seam verwendet Facelets als View Handler. Facelets ermöglicht es, die Benutzeroberfläche mit XHTML zu deklarieren und stellt die Komponenten der Benutzeroberfläche als Baumstruktur aus Tags dar. Facelets wurde direkt für JSF entwickelt, unterstützt alle JSF Komoponenten und ist vollständig in den

JSF Lebenszyklus integriert. Durch diese Integration in den JSF Lebenszyklus ergibt sich eine schnellere Generierung der Benutzeroberfläche als bei der Verwendung von JSP, da diese nur in der entsprechenden Phase des JSF Lebenszyklus aus dem Komponentenbaum generiert wird. Facelets ermöglicht das Layout der Benutzeroberfläche mit Templates und macht es möglich Teile der Benutzeroberflächen einfach wiederzuverwenden.

4.3.2. Seam Komponenten

Seam basiert auf einem kontextbehafteten Komponentenmodell. Die folgenden Java Komponenten können als Seam Komponenten verwendet werden: JavaBeans, EJBs und JPA Entities. Jeder Seam Komponenten wird einer der folgenden Kontexte zugeordnet.

Kontexte

- Stateless - Der Zustand der Komponente wird nicht gespeichert, es wird jeweils eine neue Instanz erzeugt.
- Event - Die erzeugte Komponente wird nach dem Verarbeiten eines Ereignisses zerstört.
- Page - Die erzeugte Komponente existiert bis eine Antwort auf die Anfrage des Benutzers gesendet wurde oder dieser eine andere Anfrage sendet.
- Conversation - Die Komponente existiert bis ein Anwendungsfall aus Benutzersicht vollständig abgearbeitet wurde. Dieser kann aus mehreren Anfrage Antwort Zyklen bestehen.
- Session - Die Komponente existiert über die gesamte Zeitdauer einer Session eines Benutzers.
- Application - Die Komponente existiert während der gesamten Laufzeit eines Programmes.
- Business Process - Die Komponente existiert über die Laufzeit eines Programmes hinaus. Ihr Zustand wird gespeichert und ist wiederherstellbar, bis der Prozess beendet wird.

Der Kern eines Programmes auf der Basis von Seam ist der Seam Container. Alle Komponenten und Kontexte werden durch den Seam Container erzeugt und verwaltet. Darüberhinaus sorgt er dafür, dass jeder Komponente die von ihr benötigten Dienste zur Verfügung stehen. Jede Komponente kann gewünschte Kommunikationspartner mit Hilfe des Seam Containers finden, unabhängig davon

in welchem Kontext, welcher Laufzeitumgebung oder welcher Technologie dieser existiert. Die Interoperation und der Lebenszyklus der Komponenten wird durch die Angabe von Metadaten in Form von Annotation festgelegt. Anhand der mit den Annotationen vergebenen Variablennamen der Komponenten steuert der Seam Container Transaktionen, Einhaltung von Sicherheitsdefinitionen und die Erzeugung von benötigten Komponenten. Jede erzeugte Instanz einer Komponente wird mit ihrem Variablennamen vom Seam Container verwaltet. Wenn zu einem angeforderten Variablennamen keine Instanz gefunden wird, agiert der Seam Container als Factory und erzeugt eine neue Instanz der angeforderten Komponente.
Darüber hinaus erweitert Seam das Konzept der Dependency Injection durch Dependency Outjection zur Bijection. Die Abhängigkeiten zwischen Komponenten werden anhand von Referenzen durch den Container verwaltet. Das Konzept der Bijection erlaubt es, Komponenten zu erzeugen deren Eigenschaften aus den verschiedenen Kontexten injiziert werden. Die Injektion findet dabei dynamisch bei einem Zustandswechsel statt, also während des gesamten Lebenszyklus einer Komponente. Darüber hinaus besteht die Möglichkeit mittels Outjection, von einer Komponenten in eine andere Komponente zu injizieren.
Seam eliminiert die Managed Beans als Vermittler zwischen der Web- und der Verarbeitungsschicht. Hierzu wird die Möglichkeit der Erweiterung der EL genutzt. Seam stellt einen Variablen Resolver zur Verfügung, der auf den Seam Container zugreifen kann. So können EL Ausdrücke applikationsweit verwendet werden. Jede Komponente kann z. B. in die JSF Webseite durch EL-Ausdrücke eingebunden werden.

4.4. Java EE 6

Mit dem JSR 313: Java Platform, Enterprise Edition 6 (Java EE 6) liegt eine aktuellere Version der Java EE Plattform vor, als die in dem Projekt verwendete. Bei der Ausarbeitung von Java EE 6 lagen die Schwerpunkte auf den Spezifikationen Persistenz (JPA 2.0), Enterprise JavaBeans (EJB 3.1) JavaServer Faces (JSF 2.0) und RESTful Web Services (JAX-RS 1.0). In die Weiterentwicklung von Java EE 5 zu Java EE 6 sind viele der Konzepte von Seam eingeflossen. Wesentliche Teile von Seam sind jetzt Bestandteil von Weld, der Referenzimplementierung für die neue Spezifikation JSR 299: Contexts and Dependency Injection for the Java EE platform. Facelets wurde als primäre View-Handler-Technologie in Java EE 6 Spezifikation integriert.
Die Entwicklung von Seam 3 hat zum Ziel, die nicht in Weld enthaltenen Komponenten von Seam 2 in andere Projekte einzugliedern. Seam Persistence wird unter dem Namen Hibernate CDI in das Hibernate-Core-Projekt überführt. Seam Validation

wird Teil des Projekts Hibernate Validators. Seam REST in das RESTEasy-Projekt einfließen. Seam Faces wird eine eigenständige Komponente in Form der Faces-CDI-Bibliothek in RichFaces eingehen. Seam selbst soll im Rahmen des CDI-Ökosystems als Inkubator für neue Ideen im CDI-Kontext verstanden werden. Dieser Prozess ist jedoch noch nicht abgeschlossen, so dass Java EE 6 zwar viele Konzepte von Seam zur Verfügung stellt, aber die Integration von z. B. jBPM (Prozessmanagement) und Drools (Regelmanagement) auf der Basis von Java EE 6 noch nicht verfügbar sind.

Kapitel 5. Implementierung

5.1. Konfigurationsmanagement

Für das Konfigurationsmanagement des Projekts ekomax wurde Apache Maven verwendet.
Das Konzept von Maven sieht vor, dass die Prozesse Kompilierung, Test, Paketierung etc. in den meisten Softwareentwicklungsprojekten Projekten ähnlich sind. Diese Prozesse werden durch Plug-ins unterstützt, die jeweils die benötigten Funktionen bereitstellen. Die Prozesse und Plug-ins werden über eine Projektkonfiguration (Project Object Model, POM) gesteuert. Das POM wird in einer XML-Datei definiert und enthält alle Informationen, welche die Plug-ins zur Laufzeit benötigen. Zudem ist es möglich mit Maven die Abhängigkeiten eines Projekts von externen Bibliotheken (Dependencies) in einem Repository zu verwalten.
Maven ermöglicht es, ein Projekt in kleinere Einheiten zu zerlegen. Ein Mehrmodul-Projekt definiert sich durch ein Elternprojekt, das beliebig viele Subprojekte referenziert und das seine Konfiguration an seine Subprojekte vererbt. Das Elternprojekt kann genutzt werden, um Einstellungen und Versionen von externen Bibliotheken zentral zu verwalten.
Das Projekt ekomax besteht aus den in der folgenden Tabelle beschriebenen Maven Projekten.

Projekt	Aufgabe	Artefakt
ekomax	Verwaltung der globalen Projekteigenschaften Verwaltung der Maven Plugins Verwaltung der externen Bibliotheken	pom.xml
ekomax-distribution	Zusammenstellung des JBoss AS Server Configuration Fileset	Dateissystem
ekomax-ear	Zusammenstellung des Enterprise Application Archive Generierung von Deployment Descriptors	ekomax.ear
ekomax-war	Zusammenstellung des Web Application Archive Generierung von Deployment Descriptors	ekomax.war
ekomax-jar	Zusammenstellung des Java Application Archive Kompilierung der Java Komponenten	ekomax.jar

Tabelle 5.1. Maven Projekte

5.2. Datenhaltung

Für die Anbindung der Datenbank e_kommu wurde die JPA Implementierung Hibernate verwendet. Hibernate bietet Mechanismen die zur Kompatibilität mit

verschiedenen Datenbanken führen. Die zum Datenbankzugriff erforderlichen SQL Statements werden nicht explizit in SQL programmiert, sondern von Hibernate in Abhängigkeit vom SQL Dialekt der verwendeten Datenbank generiert.
Die Abfrage der persistierten Objekte erfolgt wahlweise über die SQL ähnliche Abfragesprache Hibernate Query Language (HQL), mittels SQL Statements oder objektorientiert mittels der Hibernate Criteria API. Die Abfragen werden je nach verwendeter Datenbank mittels JDBC auf den entsprechenden SQL-Dialekt übersetzt. Das Mapping von Java Objekten auf Einträge in der Datenbank wird durch die Verwendung von Java Annotationen festgelegt.
Die wichtigsten Java Objekte in Hibernate sind die SessionFactory, Session und Transaction. Die SessionFactory lädt die Konfiguration und die Mappings, und wird normalerweise nur einmal pro Anwendung erzeugt. Session ist das Bindeglied zwischen den Java Komponenten und den Hibernate Diensten, und bietet Methoden für Insert-, Update-, Delete- und Query-Operationen. Transaction bildet Transkationen über JDBC und JTA ab, wobei geschachtelte Transaktionen nicht unterstützt werden. Die Verbindung zu der Datenbank e_kommu wird über einen JCA Ressource Adapter hergestellt. Hierzu muss der JCA Connector Provider des JBoss Apllication Servers mit Hilfe eines JDBC DataSource Deployment Descriptors konfiguriert werden. Dieser Deployment Descriptor wird in Form einer XML-Datei mit dem Suffix –ds.xml erstellt. Zur Laufzeit stellt der Application Server anhand des Deployment Descriptors die benötigten Verbindungen zur Datenbank her und verwaltet diese.

5.3. Datentransformation

Zur Transformation der Daten eingehender und ausgehender Nachrichten wird der "Structured Data Event Stream Processor" Smooks verwendet. Smooks stellt eine umfangreiche Visitor API bereit, mit der Daten aus beliebigen Quelle eingelesen und als XML Dokument dargestellt werden können. Die Implementierung der Visitor API für die Quelldatenformate XML, CSV, JSON, Java und EDI sind Bestandteil von Smooks. Wird ein benötigtes Format nicht unterstützt, kann Smooks durch die Implementierung der Visitor API erweitert werden. Basierend auf dem aus der Quelle erzeugten XML Dokument führt Smooks die Transformation in das Zielformat durch. Die Transformation wird mit einer XML Datei konfiguriert. Die Möglichkeiten Transformationen zu konfigurieren (z. B. CSV zu XML, XML zu Java, Java zu Java und EDI zu Java) sind Bestandteil von Smooks. Neben der Transformation von Daten beherrscht Smooks auch das Splitting und Routing von Nachrichten via JMS. Transformation, Splitting oder Routing sind typische Aspekte eines Enterprise Service Bus (ESB), daher wird Smooks hierfür z. B. in Apache Mule und JBoss ESB eingesetzt.

5.4. REST API

Die REST API wurde unter Verwendung des JBoss RESTeasy implementiert. RESTeasy implementiert die Java API for RESTful Web Services (JAX-RS), welche im Rahmen des JSR 311 entwickelt wird. Der JSR 311 beschreibt, wie Java Programme entwicklet werden können, die das REST Konzept umsetzen. RESTEasy nutzt die Java Architecture for XML Binding (JAXB, JSR 222), um Ressourcen in den Fromaten XML, JSON, YAML, oder Atom auszuliefern. RESTeasy lässt sich in EJBs integrieren.

5.5. Export

Der Export ist über HTTP mit den in der folgenden Liste aufgeführten Parametern erreichbar.

Export

- Adresse: `http://${SERVER_ADDRESS}/ekomax/resources/${REPORTING_PERIOD}/${FEDERAL_STATE}`
- Methode: `GET`

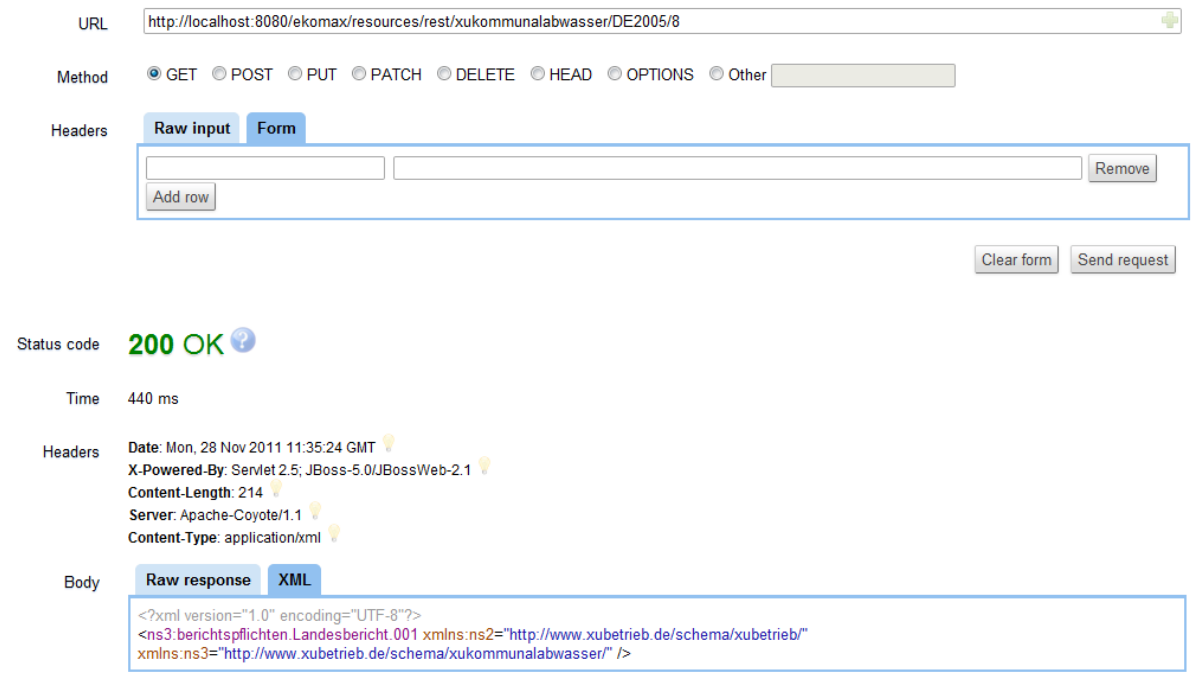

Abbildung 5.1. Export REST request

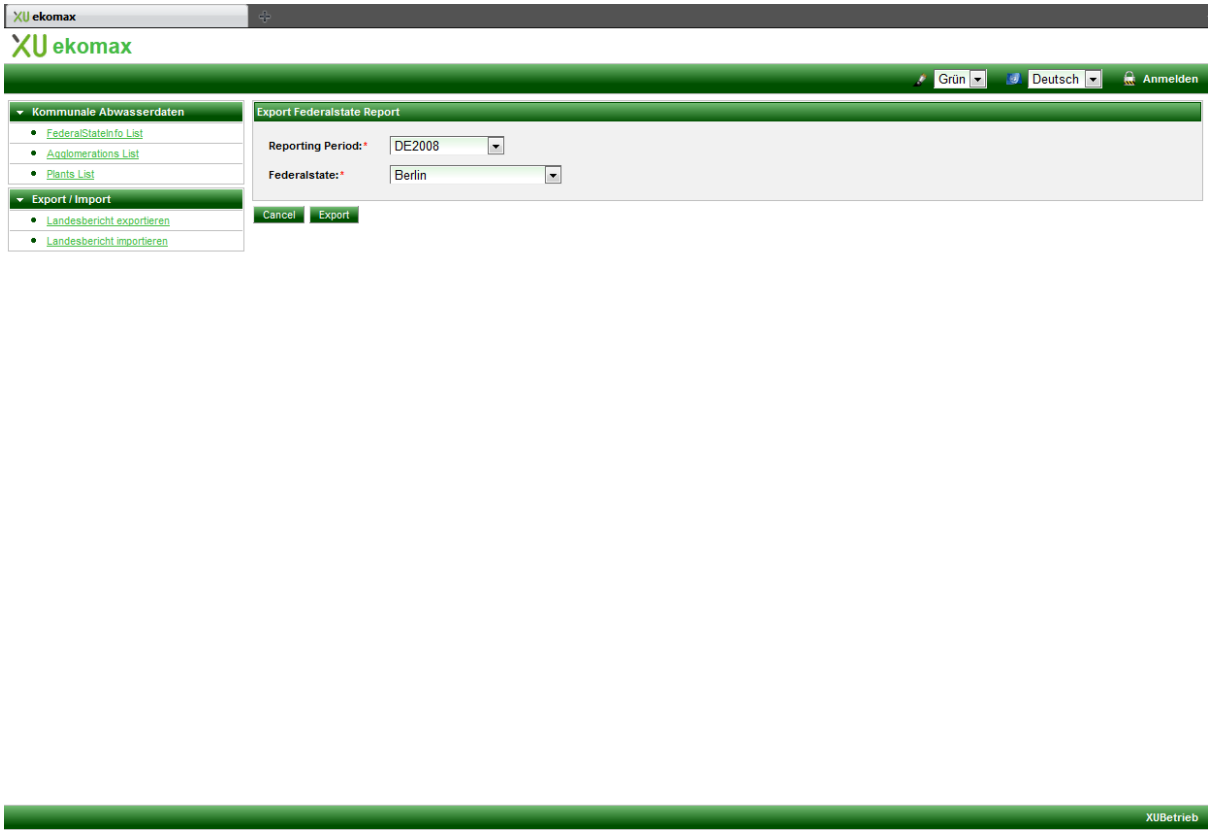

Abbildung 5.2. Export Web Benutzeroberfläche

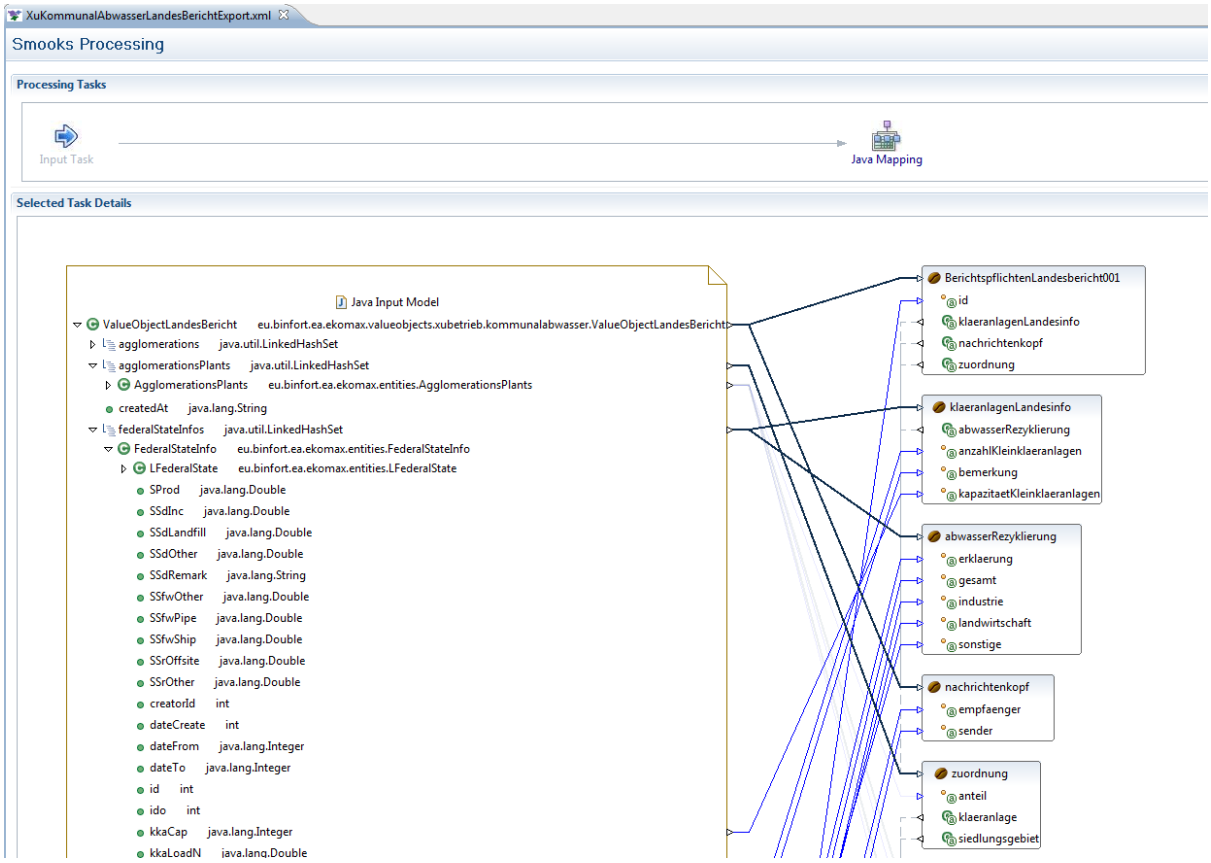

Abbildung 5.3. Export Smooks Konfiguration

5.6. Import

Der Import ist über HTTP mit den in der folgenden Liste aufgeführten Parametern erreichbar.

Import

- Adresse: `http://${SERVER_ADDRESS}/ekomax/resources/`
- Methode: `POST`
- Request Header, Authorization: `Basic ${CREDENTIALS_BASE64_ENCODED}`
- Request Header, Content-Type: `application/xml`
- Request Header, Body: `XML Infoset: berichtspflichten.Landesbericht.001`

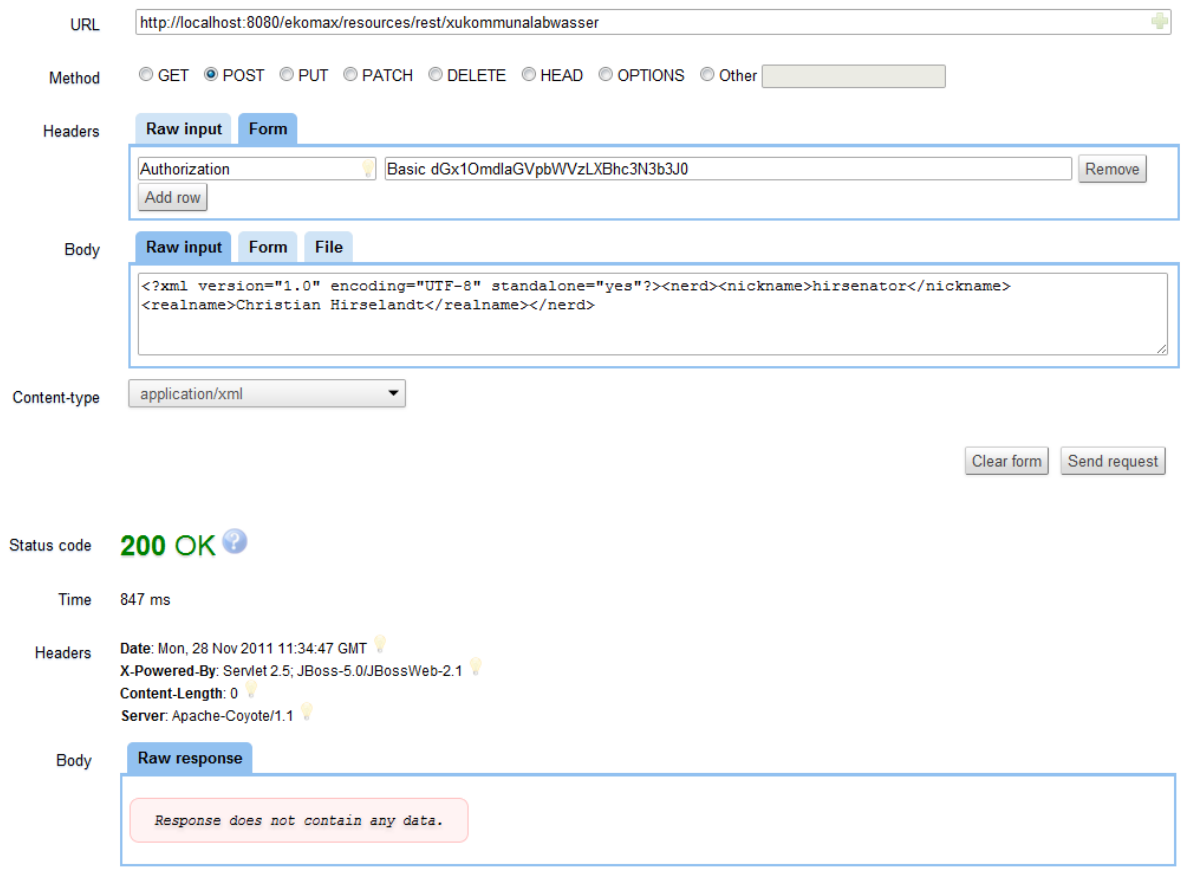

Abbildung 5.4. Import REST request

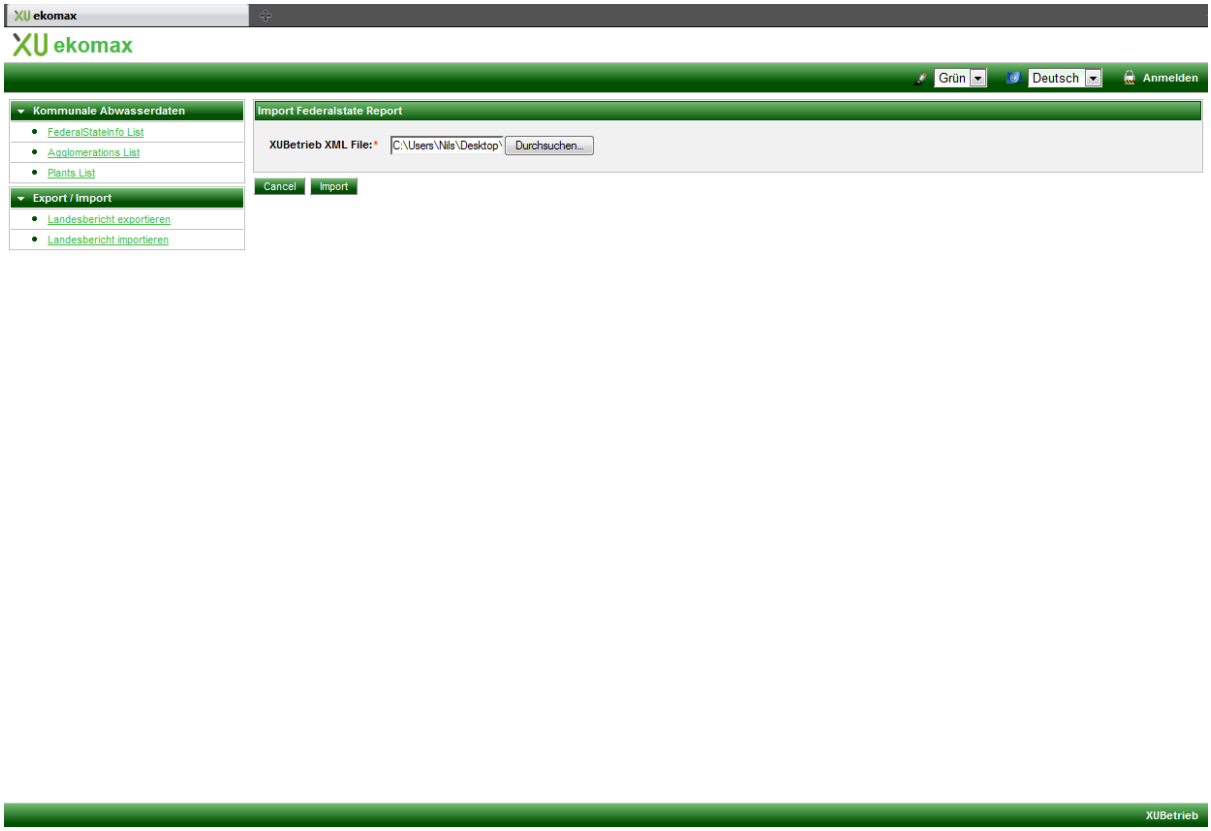

Abbildung 5.5. Import Web Benutzeroberfläche

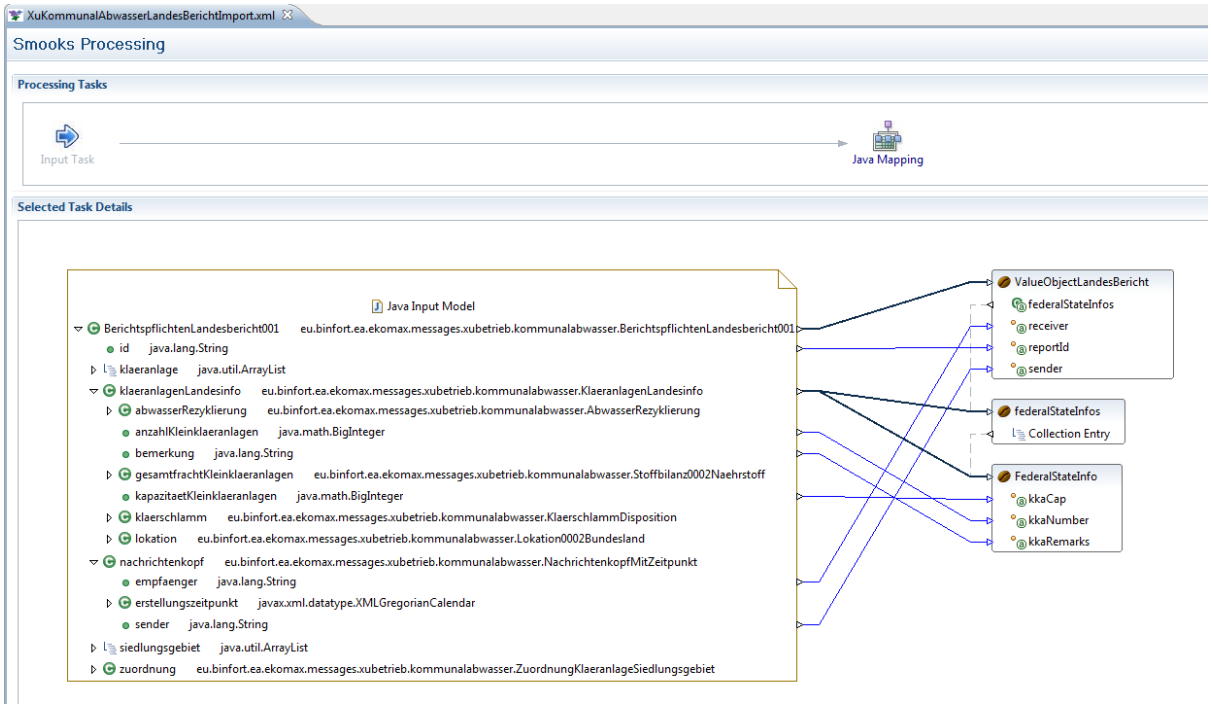

Abbildung 5.6. Import Smooks Konfiguration

Kapitel 6. Konfiguration

Die Enterprise Application ekomax kann auf einem JBoss Application Server 5.1 betrieben werden. Dazu ist es notwendig das in der folgenden Liste aufgeführte Dateisystem bereitzustellen.

Dateisystem ekomax

- deploy/ekomax-ds.xml
- deploy/ekomax.ear
- lib/postgis-jdbc-1.3.3.jar
- lib/postgresql-9.0-801.jdbc4.jar

Dieses Dateisystem befindet sich nach dem Build des Maven Projekts ekomax unter dem Dateisystempfad `${WORKSPACE}/ekomax-distribution/target/distribution/distribution-jbossas/server` und muss in das Dateisystem `${JBOSS_HOME}/server/${CONFIGURATION}` integriert werden.
Weiterhin ist es notwendig, die DataSource Definition in der Datei `ekomax-ds.xml` anzupassen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE datasources PUBLIC "-//JBoss//DTD JBOSS JCA Config 1.5//EN" "http://
www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd">
<datasources>
 <local-tx-datasource>
  <jndi-name>ekomaxDatasource</jndi-name>
  <use-java-context>false</use-java-context>
  <connection-url>jdbc:postgresql://${SERVER_ADDRESS}/e_kommu</connection-url>
  <driver-class>org.postgis.DriverWrapper</driver-class>
  <user-name>${USERS_NAME}</user-name>
  <password>${USERS_PASSWORD}</password>
  <new-connection-sql>SELECT CURRENT_TIMESTAMP</new-connection-sql>
  <check-valid-connection-sql>SELECT CURRENT_TIMESTAMP</check-valid-connection-
sql>
  <metadata>
   <type-mapping>PostgreSQL 8.0</type-mapping>
  </metadata>
 </local-tx-datasource>
</datasources>
```

Beispiel 6.1. PostGIS DataSource Definition

Die Elemente `connection-url`, `user-name` und `password` müssen so aktualisiert werden, dass sie mit den entsprechenden Parametern der zur Verfügung stehenden Datenbank e_kommu übereinstimmen.

Kapitel 7. Fazit

Die Anwendbarkeit von XUBetrieb als Metamodell für betriebliche Umweltberichtspflichten konnte am Beispiel XUKommunalabwasser mit ekomax als Referenzimplementierung gezeigt werden.
Die Referenzimplementierung ekomax verfügt funktional nicht über den Umfang des existierenden Software Systems e-Kommunalabwasser, zeigt jedoch auf, wie die typischen Anwendungsfälle bei der Implementierung von XÖV Standards unter der Verwendung von aktuellen Open Source Lösungen, die grundsätzlich SAGA konform sind, realisiert werden können.
Der Import funktioniert bis zum dem Punkt an dem die Daten in die existierenden Datenbank e_kommu geschrieben werden, da aus bisher vorhandenen XML Infosets keine Daten extrahiert werden können, die von den programmierten Einschränkungen der Datenbank e_kommu akzeptiert werden.
Der Export liefert ein XML Infoset, das Daten aus der existierende Datenbank e_kommu beinhaltet, jedoch ist dieses weder vollständig noch valide zum XML Schema.
Im Folgendem werden Herausforderungen erläutert, die zu diesem Ergebnis beigetragen haben.
Durch die Implementierung sollten und konnten im verwendeten XML Schema Fehler identifiziert und in XUKommunalabwasser Version 1.0.0 korrigiert werden. Jede Iteration des XML Schemas machte es erforderlich ein neues Inkrement der Komponenten JAXB Klassen und Smooks Konfiguration zu erstellen. Daher steht der Funktionsumfang von ekomax stets dem XML Schema nach.
Eine häufige Fehlerart war die inkorrekte Multiplizität an verschiedenen Elementen. Bei nicht explizit angegebenen Grenzen wird vom Modellierungstool MagicDraw standardmäßig die Multiplizität [0..1] verwendet und fehlerhafte Grenzen sind nicht während der Validierung, sondern erst in der praktischen Nutzung feststellbar.
Nicht alle in der Datenbank vorhandenen Attribute entsprechen einem Element im XML-Schema. Umgekehrt sind nicht alle im neu entwickelten XML-Schema enthaltenen Elemente in der Datenbank existent. So wurde unter anderem das Element Vorgehensweise bei der Gesamtfracht der Kleinkläranlagen auf XML-Schemaebene hinzugefügt und auch die Einheiten für eine Messgröße sind bisher nicht immer in der Datenbank berücksichtigt.
Strukturelle Unterschiede zwischen Datenbank und Schema, wie die XOR Unterscheidung einer exterritoriale Lokation einer Klaeranlage, sind eine weitere Herausforderung. Die praktische und pragmatische Auswertung von "umfunktionierten" Datenbankeinträgen, wie sie in e-Kommunalabwasser vorgenommen wird, ist bei der Verwendung von XML Schema nicht möglich.

Zusätzlich wurden in XUBetrieb verschiedene Parameter in Codelisten zusammengefasst. Ein Bespiel hierfür sind die Anforderungen an eine Kläranlage hinsichtlich der Erlaubniswerte. Da sie aus einzelnen Datenbankeinträgen in eine Aufzählung auf Schemaebene überführt werden müssen, ist kein direktes Mapping möglich.
Unterschiede der Datentypen von Datenbankeinträgen und XML-Schemaelementen müssen zusätzlich berücksichtigt werden. Sie treten vor allem an den Stellen auf, an denen in der Datenbank Wahrheitswerte verwendet werden. Diese sind in der Regel als Integer realisiert, während im XML-Schema entsprechend der sonst üblichen Praxis entsprechend der Datentyp Boolean genutzt wird.
In der zur Verfügung stehenden Datenbank sind nicht alle Werte in den entsprechenden Berichtspflichten für die Schemaerstellung vorhanden. Für die Berichtsperiode 2005 sind keine Landesinformationen gespeichert.
Die in XUBetrieb bereitgestellten und von XUKommunalabwasser verwendeten Codes weichen von den aktuell genutzten, historisch gewachsenen Codes mit mehreren tausend Einträgen ab. Ursache dafür ist, dass XUBetrieb auf bereits standardisierte Codelisten zurückgreift oder sich an ihnen orientiert. So verwendet die Liste der Großstädte über 100.000 Einwohner, abweichend von den vorher festgelegten Codes, den jeweiligen amtlichen Gemeindeschlüssel als eindeutigen Identifikator. Die Diskrepanz der unterschiedlichen Codes in den Codelisten führt zu invaliden Nachrichten.
Für verschiedene Elemente sind bisher je nach Anwendungsfall unterschiedliche Schlüssel verwendet worden. So existiert für die Kläranlage ein nationaler Schlüssel und ein Schlüssel für die Kommunikation mit der EU. Für XUKommunalabwasser in der Verwendung eindeutig geregelt, stellt diese Gegebenheit bei der Integration von Altdatenbeständen eine Herausforderung dar.
Bei der Datenbank handelt es sich um eine PostgreSQL Datenbank mit PostGIS Erweiterung. Dies macht die zusätzliche Verwendung einer Hibernate Extension notwendig.
Die Datenbank stellt zwei Schemata zur Verfügung. Auch Teile der für die zu erfüllende Berichtspflicht notwendigen Daten liegen in der Datenbank in zwei Schemata vor, was einen erhöhten Aufwand bei der Schreib- und Leseoperationen zur Folge hat.
Eine Normalisierung der Datenbank würde die jeder Tabelle zugeordneten Gültigkeitszeiträume auflösen und die Berichtszusammenstellung vereinfachen. Dies gilt auch für die Auflösung der Vorgänger-Nachfolger-Tabellen.
Die aus der Struktur der Datenbank resultierenden JPA Datenbankklassen enthalten rekursive Bezüge. Ob dieses Problem aus der Struktur der Datenbank oder generell

aus einer Inkompatibilität des verwendeten Java Frameworks mit der verwendeten Datenbank resultiert, ist vorerst nicht geklärt.
Die angewendeten Tabellen- und Spaltennamen entsprechen nicht den Konventionen der Hibernate Java Klassen. Die unterschiedlichen Schreibweisen führen zu Konflikten bei den Akzessoren (setter und getter).
XUBetrieb stellt die Klassen und deren Attribute in deutscher und englischer Sprache zur Verfügung. Da die Datenbankbezeichner in englischen Akronymen vorliegen und das XML-Schema deutsche Bezeichner nutzt, sollten die spezifischen Elemente in XUKommunalabwasser analog zu XUBetrieb zweisprachig bereitgestellt werden.
Eine zukünftige Weiterentwicklung der beschrieben Implementierung sollte die folgenden Punkte umsetzen oder beachten.
Für die oben beschriebenen Punkte von Datentyp- und Strukturdiskrepanz sind die notwendigen spezifischen Mappings zu entwicklen. Dies ist unter Verwendung von Custom Decodern für das Mapping Tool Smooks möglich und wird unter anderem für die folgenden Punkte benötigt: Kläranlage, Siedlungsgebiet, Datums- und Zeitinformationen (XMLGregorianKalendar).
Als Codelisten sollten die neu definierten Codelisten aus XUBetrieb verwendet werden, was aufgrund ihrer Kompatibilität zu bereits veröffentlichten Codelisten angeraten ist. Alternativ ist bei Verwendung alter Codelisten ein Mapping bereitzustellen.
Als genereller Aspekt ist abschliessend die Dokumentation der fachlichen Domäne wünschenswert. Eine detaillierte Erklärung der existierenden komplexen Abläufe und des speziellen fachlichen Vokabulars ist eine Grundvoraussetzung für eine schnelle und korrekte Umsetzung der fachlichen Anforderungen.