Konzeption und Implementierung von OpenInfRA in der Version 2.0

Lehrstuhl Datenbank- und Informationssysteme

Konzeptversion: 1.4

21. Oktober 2015

Inhaltsverzeichnis

1 Einleitung

Das *Open Information System for Research in Archaeology (OpenInfRA)*[1] ist als ein webbasiertes Informationssystem zur Dokumentation archäologischer Forschungsprojekte geplant. Ein umfangreiches *Grobkonzept* [Ope14] ist die Basis für eine weiterführende Entwicklung von OpenInfRA. Das Grobkonzept beschreibt grundlegende Funktionalitäten sowie Anwendungsfälle aus der Sicht eines Fachanwenders. Dazu gehören u.a. ein struktureller Überblick über die gewünschten Systemkomponenten, eine Beschreibung der gewünschten Schnittstellen zu anderen Systemen, eine Darstellung der Synchronisation zwischen Online- und Offline-Systemen und die Beschreibung einer gewünschten Mehrsprachigkeit.

Abbildung 1 stellt die im Grobkonzept geplanten Systemkomponenten grafisch dar. Den geplanten Systemkomponenten sind drei Schichten zugeordnet. Die erste Schicht wird *Datenhaltungsschicht* genannt und bezieht sich auf die persistente Speicherung von Fachdaten und Metadaten. Je nach Art der zu speichernden Daten können diese entweder in einer Datenbank oder im Dateisystem gespeichert werden. Für die Speicherung der Fachdaten in einer Datenbank existiert bereits ein konkretes Datenbankschema, welches im Datenbankmanagementsystem (DBMS) PostgreSQL[2] implementiert ist. Die Speicherung im Dateisystem ist für Binärdateien (z.B.: PDF-Dokumente und Bilddateien) vorgesehen. Die zweite Schicht wird *Anwendungsschicht* genannt und umfasst den Zugriff auf die Datenbank bzw. das Dateisystem, Administrationsfunktionalitäten (z.B.: Benutzer oder Projekte anlegen), Benutzerfunktionalitäten (z.B.: Themeninstanzen anlegen und bearbeiten), Schnittstellen (z.B.: JavaScript object notation (JSON) oder Extensible Markup Language (XML) Export bzw. Import) sowie ein Benutzer-, Rollen- und Rechtesystem, welches die Identifizierung und Authentifizierung der Benutzer von OpenInfRA umsetzten soll.

Wie in Abbildung 1 dargestellt ist, wird OpenInfRA zusätzlich in ein *Kernsystem*, ein *Basissystem* und ein *erweiteretes Basisystem* unterteilt.

- **Kernsystem**: Das Kernsystem umfasst die Kernfunktionalitäten, die zum minimalen Betrieb von OpenInfRA bereitgestellt werden sollen. Der Entwurf und die Implementierung weiter Teile des Kernsystems, wie z.B. die Weboberfläche

[1] http://www.tu-cottbus.de/projekte/de/openinfra/
[2] http://www.postgresql.org/

und das Benutzer, Rechte und Rollen-System, waren ursprünglich durch eine externe Entwicklung geplant.

- **Basisystem**: Das Basissystem soll das Kernsystem um weitere intern Entwickelte Funktionalitäten ergänzen, wie z.B. ein 2D-WegGIS und Schnittstellen zu externen Systemen.
- **erweitertes Basissystem** : Das erweiterte Basissystem soll das Basisystem um weitere Funktionalitäten ergänzen, die Forschungsaspekte umfassen. Dazu gehören z.B. ein 3D-WebGIS sowie eine Expertensuche.

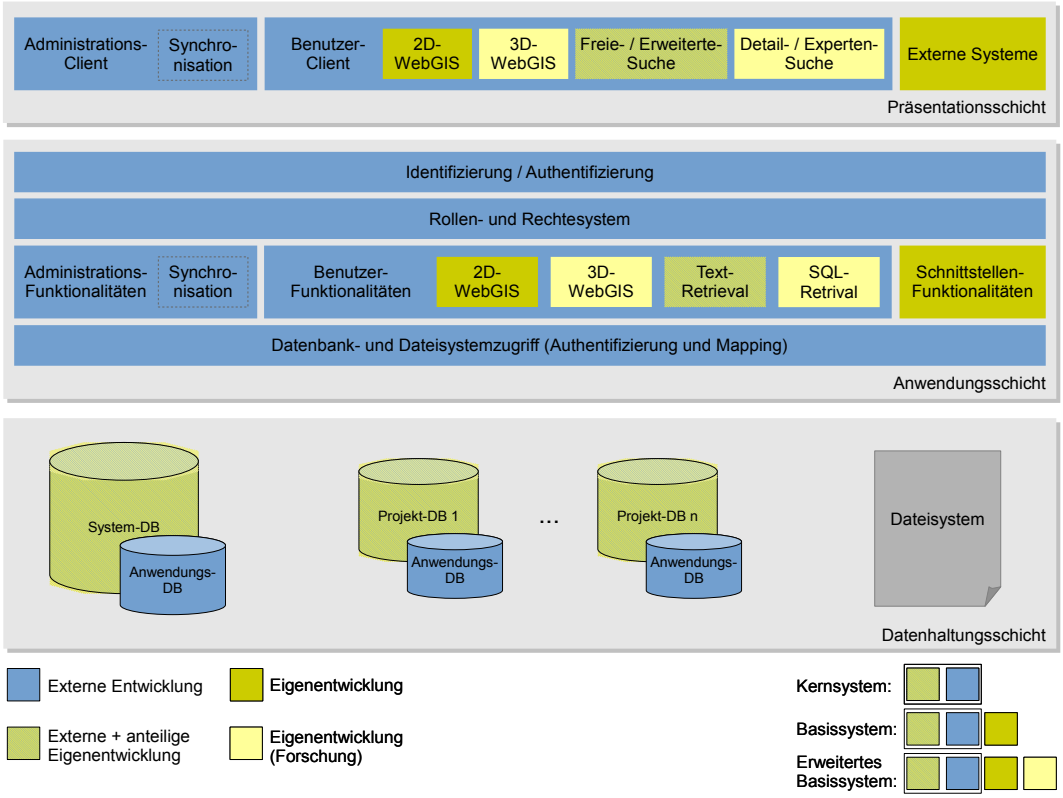


Abbildung 1: Systemkomponenten (vgl. [Ope14])

1.1 Ziele

Dieses Dokument umfasst zwei Ziele. Erstens soll hier das OpenInfRA-Feinkonzept detailliert dargestellt werden. Das Feinkonzept soll konkrete Design-Entscheidungen

für eine Weiterentwicklung von OpenInfRA nachvollziehbar machen. Zweitens soll hier die konkrete Umsetzung von OpenInfRA in der Version 2.0, die auf dem Feinkonzept basiert, beschrieben werden. Diese Beschreibung soll als Entwicklerdokumentation dienen und eine zukünftige Weiterentwicklung von OpenInfRA vereinfachen.

Das Feinkonzept und die anschließende konkrete Umsetzung sollen die vollständige Datenhaltungsschicht sowie Teile der Anwendungsschicht umfassen. Folgenden Teile der Anwendungsschicht gehören dazu (vgl. Abbildung 1):

- *Datenbank- und Dateisystemzugriff*
- *Administrationsfunktionalitäten*
- *Synchronisation*
- *Benutzerfunktionalitäten*
- *Schnittstellenfunktionalitäten*
- *Rollen- und Rechtesystem*
- *Identifizierung / Authentifizierung*

1.2 Anforderungskatalog

Dem Grobkonzept ist ein detaillierter *Anforderungskatalog* beigefügt, der sowohl funktionale als auch nicht funktionale Anforderungen beschreibt. Zu diesem Anforderungskatalog gehören Themenbereiche, die im folgenden aufgezählt werden.

1. *WebGIS* – WGC_WebGIS
2. *Benutzung* – BN_GUI
3. *Administration* – AC_Administration
4. *Geo Document Service* – GDS_GeoDocumentService
5. *Konzepte* – KT_Konzepte
6. *System* – SW_System
7. *Schnittstellen* – IFS_SchnittstellenDienste
8. *Sicherheit* – SEC_Sicherheit
9. *Performance* – PZ_Performanz
10. *Suche* – SEA_Suche

11. *Benutzbarkeit* – UAB_Usability

Folgende Themenbereiche des Anforderungskatalogs werden im vorliegenden Dokument nicht betrachtet. Für jeden der aufgeführten Themenbereiche wird diese Entscheidung kurz begründet.

- *WebGIS* (WGC_WebGIS): Dieser Themenbereich wird vom Kernsystem getrennt entwickelt und ist aus diesem Grund nicht Bestandteil dieses Konzeptes.
- *Benutzbarkeit* (UAB_Usability): Dieser Themenbereich bezieht sich ausschließlich auf Funktionalitäten in der Benutzeroberfläche und wird aus diesem Grund in diesem Konzept nicht weiter betrachtet.

1.3 Gliederung und Aufbau

Im Kapitel 2 werden zunächst die Grundlagen erläutert, die für das Verständnis der eingesetzten Technologien benötigt werden. Weiterhin werden im Kapitel 3 die für OpenInfRA geplanten Komponenten beschrieben. Diese Beschreibung der Komponenten bildet die Basis für das OpenInfRA-Konzept und bezieht sich auf die im Grobkonzept eingeteilten Systemkomponenten (vgl. 1). Aufbauend darauf wird im Kapitel 7 das objektrelationale Mapping erklärt, welches die Zugriffe auf die Datenbank umsetzen wird. Weiterhin wird im Kapitel 8 das URI-Mapping erläutert, welches die Schnittstellen für den Zugriff externer Systeme auf die Daten und Informationen von OpenInfRA beschreibt. Die zuvor beschriebenen Kapitel beziehen sich ausschließlich auf die Beschreibungen der konzeptionellen Planung von OpenInfRA. Anschließend wird im Kapitel 14 die konkrete Implementierung erläutert.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die zum Verständnis der folgenden Kapitel notwendig sind. Zuerst wird das Programmierparadigma: *Representational State Transfer (REST)*[1] vorgestellt. Aufbauend auf dieser Vorstellung werden Methoden, die zum *Hypertext Transfer Protocol (HTTP)* gehören, erklärt.

2.1 OpenInfRA – Client-Server-Modell und Schnittstellen

Im Grobkonzept [Ope14] wird für OpenInfRA ein *Client-Server-Modell*[2] geplant. Bei einem 'Client-Server-Modell' erbringt der Server Dienste, die in Form von *Schnittstellen*[3] nach außen bereitgestellt werden. Eine 'Schnittstelle' definiert in diesem Kontext die Regeln, die für die für Kommunikation mit dem Server gelten. Ein beliebiger Client (unabhängig von der Programmiersprache) kann diese Schnittstellen nutzen, um die vom Server bereitgestellten Dienste zu konsumieren.

In der Abbildung 2 ist die OpenInfRA-Server-Komponente zentral in der Mitte dargestellt. Diese Server-Komponente soll die Datenhaltungsschicht sowie die Anwendungsschicht (vgl. Abbildung 1) enthalten. Wie in der Abbildung 2 dargestellt ist, sind für diese zentrale Server-Komponente weitreichende Schnittstellenfunktionalitäten geplant, die z.B. eine Kommunikation über Benutzerschnittstellen, einen Datenaustausch über Web-Services und die Benutzung externer Web-Applikationen mit einschließen. Diese Schnittstellenfunktionalitäten beziehen sich auf die Präsentationsschicht (vgl. Abbildung 1). Zusätzlich sollen diese Schnittstellen unterschiedliche Formate wie z.B. XML, *Portable Document Format (PDF)*[4] oder *Comma-separated Values (CSV)*[5] für den Austausch von Daten und Informationen bereitstellen.

[1] http://de.wikipedia.org/wiki/Representational_State_Transfer
[2] http://de.wikipedia.org/wiki/Client-Server-Modell
[3] http://de.wikipedia.org/wiki/Schnittstelle
[4] http://de.wikipedia.org/wiki/Portable_Document_Format
[5] http://de.wikipedia.org/wiki/CSV_(Dateiformat)

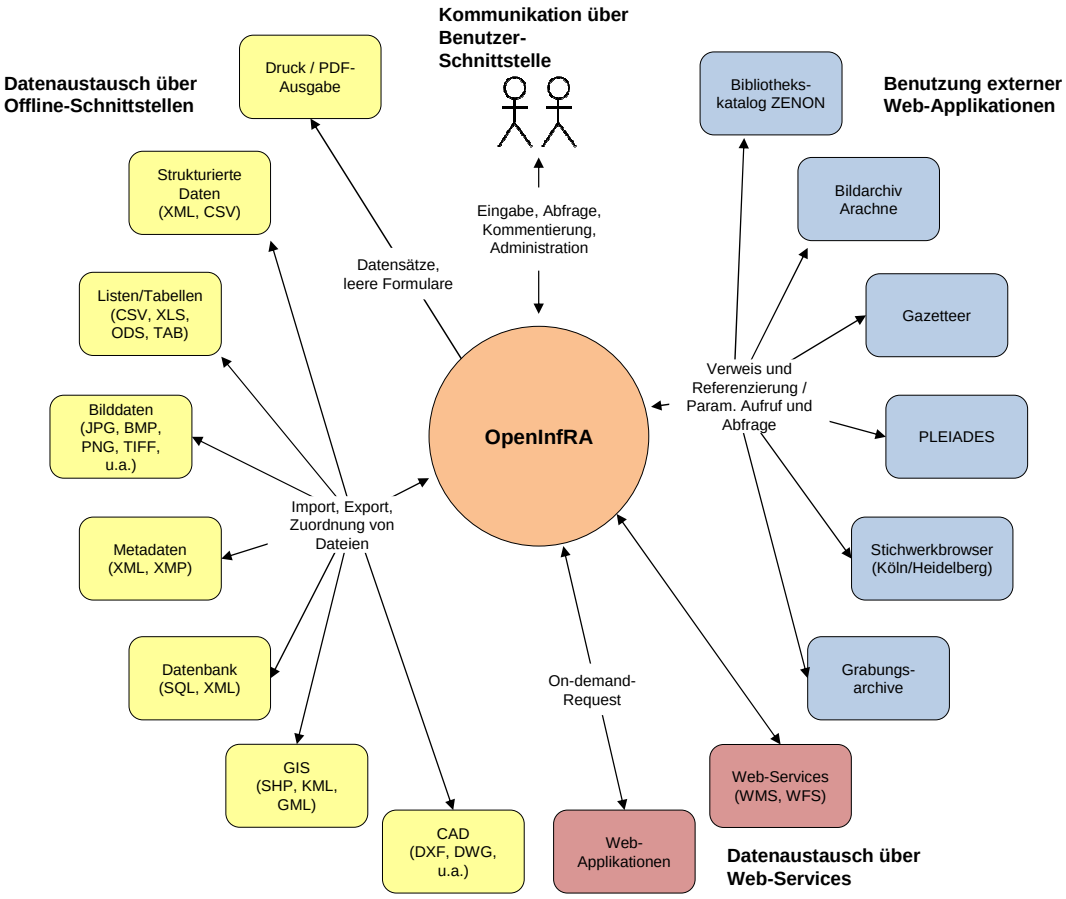


Abbildung 2: Schnittstellen (vgl. [Ope14])

2.2 Representational State Transfer

REST[6] [Fie00] ist ein Programmierparadigma, welches im Bereich von Webapplikationen eingesetzt wird und eine Reihe von Bedingungen für die Erstellung einer Systemarchitektur dieser Webapplikationen definiert. Zu diesen Bedingungen gehören: *Ressourcen*, *Schnittstellen*, *Repräsentation*, *Zustandslosigkeit* und *Hypermedia As The Engine Of Application State (HATEOAS)*, die anschließend näher erläutert werden. Sind diese Bedingungen erfüllt, dann können häufig gewünschte Eigenschaften, wie z.B. Skalierbarkeit des Systems sowie eine lose Kopplung zwischen den in Beziehung stehenden Systemen, leicht umgesetzt werden. Webdienste,

[6] http://de.wikipedia.org/wiki/Representational_State_Transfer

die diese Bedingungen einhalten, werden als *RESTful-Web-Services* bezeichnet und wenden für das Web definierte Standards, wie z.B. *Uniform Resource Identifiers (URIs)*, HTTP, XML, und JSON, konsequent an [Bur10].

- **Ressourcen**: Der Begriff der Ressource ist im Kontext von REST eine Schlüsselkomponente zur Abstraktion von Daten und Informationen. Jede Ressource muss durch eine URI eindeutig adressierbar sein.
- **Schnittstellen**: Für die Bereitstellung der Schnittstellen wird gefordert, dass ausschließlich eine kleine Menge an wohl definierten Methoden zur Manipulation der Ressourcen zur Verfügung stehen. Die einzelnen Methoden werden im folgenden Kapitel 2.4 näher beschrieben.
- **Repräsentation**: Wie bereits beschrieben, adressiert eine URI eine Ressource, wobei die Ressource die zugehörigen Daten bzw. Informationen abstrahiert. Durch diese Abstraktion ist die Repräsentation der Daten bzw. Informationen unabhängig und kann in unterschiedlichen Formaten ausgeliefert werden. Unterschiedliche Kommunikationssysteme, die auf diese Ressourcen zugreifen, haben jeweils unterschiedliche Anforderungen an das zurück zu liefernde Format: z.B. wird eine JavaScript-Anwendung JSON oder XML konsumieren, eine Druckeinheit könnte z.B. ein PDF-Dokument verlangen, ein OpenOffice-Dokument könnte eine CSV-Datei verarbeiten und ein Browser könnte eine *Hypertext Markup Language (HTML)*-Seite[7] darstellen.
- **Zustandslosigkeit**: Mit der Zustandslosigkeit ist gemeint, dass Anfragen keinen Bezug zu früheren Anfragen haben, mehrere Anfragen als voneinander unabhängige Transaktionen behandelt werden und keine Sitzungsinformationen ausgetauscht werden.
- **HATEOAS**: Die Idee von HATEOAS ist, dass die ausgelieferten Daten zusätzliche URIs zu weiteren Ressourcen enthalten. Dafür ist im Standard [NRA05] eine Liste von Beziehungen definiert.

2.3 Verwendung von REST für OpenInfRA

Für OpenInfRA können die folgenden Kriterien identifiziert werden:

- **Webbasiertes Informationssystem**: Wie bereits beschrieben, ist OpenInfRA als ein webbasiertes Informationssystem geplant (siehe Kapitel 1).

[7] http://de.wikipedia.org/wiki/Hypertext_Markup_Language

- **Weitreichende Schnittstellen**: OpenInfRA soll weitreichende Schnittstellenfunktionalitäten anbieten (siehe Kapitel 2.1).
- **Existierende IT-Landschaften**: OpenInfRA ist als Infrastrukturkomponente geplant, welche sich in eine bereits existierende IT-Landschaft integrieren soll (siehe Kapitel 2.1).
- **Existierendes Datenbankschema**: Weiterhin bildet ein bereits existierendes Datenbankschema die Fachanwenderdaten so ab, dass projektspezifische Bedürfnisse berücksichtigt werden.

Für die Implementierung von OpenInfRA müssen Technologien angewählt werden, die die genannten Kriterien angemessenen berücksichtigen. Im Folgenden wird erläutert, warum durch die Verwendung von REST als Programmierparadigma die genannten Kriterien von OpenInfRA in einem angemessenen Maße berücksichtigt werden.

- **Webbasiertes Informationssystem**: REST ist ein modernes Programmierparadigma, welches webbasierte Technologien einsetzt. Dabei schließt die Nutzung von REST die Verwendung weit verbreiteter webbasierter Technologien wie z.B. das Erzeugen von HTML-Inhalten und Formularen auf dem Server nicht aus.
- **Weitreichende Schnittstellen**: Das Programmierparadigma REST verfolgt zwei Ziele: Erstens dient es zur Vereinheitlichung von Schnittstellen zwischen Systemen. Zweitens zur Reduzierung der Menge an Aktionen auf eine überschaubare Menge[8]. Daher ist die Verwendung von REST ideal dazu geeignet, um weitreichende Schnittstellenfunktionalitäten anzubieten.
- **Existierende IT-Landschaften**: Im Gegensatz zu anderen Technologien, wie z.B. das *Simple Object Access Protocol (SOAP)*[9] oder *Common Object Request Broker Architecture (CORBA)*[10] ist REST viel leichtgewichtiger und kommt, weil es auf dem HTTP-Standard basiert, ohne die Definition neuer Protokolle aus. Aus diesem Grund können auf REST basierende Webdienste von sehr vielen Anwendungen sehr einfach konsumiert werden. Daher ist das Programmierparadigma REST ideal dafür geeignet, damit sich OpenInfRA in bereits existierende Infrastrukturen, wie z.B. vom Deutschen Archäologischen Institut[11], passgenau integrieren kann.

[8] http://de.wikipedia.org/wiki/Representational_State_Transfer
[9] http://de.wikipedia.org/wiki/SOAP
[10] http://de.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
[11] www.dainst.org

- **Existierendes Datenbankschema**: Durch das Programmierparadigma REST werden Daten bzw. Informationen eindeutig durch Ressourcen adressierbar. Durch die Verwendung von REST wird es möglich, die Fachanwenderdaten feingranular auf die nötigen Schnittstellenfunktionalitäten abzubilden.

Zusätzlich wird darauf hingewiesen, dass bei der Entwicklung von OpenInfRA nicht zwingend alle durch REST definierten Bedingungen (vgl. Kapitel 2.2) erfüllt werden müssen.

2.4 HTTP-Methoden

Für den Betrieb einer Webapplikationen wird das HTTP-Protokoll verwendet. Bei Webapplikationen, die nicht auf dem Programmierparadigma REST basieren und eine herkömmliche Client-Server-Architektur verwenden, bei der der auszuliefernde HTML-Text auf dem Server erzeugt wird, wird die Mächtigkeit des HTTP-Protokolls zumeist verschleiert z.B. durch den Browser.

Um eine auf dem Programmierparadigma REST basierende Webapplikationen zu entwickeln, ist die richtige Verwendung der durch das HTTP-Protokoll bereitgestellten Funktionalitäten von großer Bedeutung. Aus diesem Grund werden hier die für OpenInfRA verwendeten HTTP-Methoden kurz erläutert. Das HTTP-Protokoll hat eine kleine Menge an Methoden, die alle eine spezielle Definition haben und einer definierten Funktionalität entsprechen. Diese Methoden können zur Abbildung der Schnittstellenfunktionalitäten auf die Funktionen: Create, Read, Update und Delete (CRUD) verwendet werden. Im Folgenden werden die HTTP-Methoden *GET*, *PUT*, *DELETE* und *POST* erläutert [Bur10]. Anschließend wird zusätzlich die HTTP-Methode *PATCH* erläutert, die partielle Updates auf einer Ressource ermöglicht.

- **GET**: Die GET-Methode ist eine ausschließlich lesende Methode und kann dazu genutzt werden, um Daten, die durch eine URI referenziert werden, von einem Server abzurufen. Somit entspricht diese Methode hier der Read-Methode der CRUD-Operationen. Diese Methode ist idempotent. Die Idempotenz besagt, dass diese Funktion, egal wie oft sie aufgerufen wird, immer das gleiche Ergebnis zur Folge hat.

- **PUT**: Die PUT-Methode wird hier dazu genutzt, um eine existierende Ressource zu verändern. Somit verweist die konkrete URI auf ein konkretes Objekt, welches verändert werden soll und setzt hier die Update-Methode der CRUD-Operationen um. Auch diese Methode ist idempotent.

- **DELETE**: Die DELETE-Methode wird verwendet, um eine Ressource zu löschen und entspricht der Delete-Methode der CRUD-Operationen. Auch diese Methode ist idempotent.

- **POST**: Die POST-Methode wird hier verwendet, um eine neue Ressource zu erstellen. Damit entspricht diese Methode der Create-Methode der CRUD-Operationen. Diese Methode ist nicht idempotent.

Wie in [Bur10] beschrieben, kann die PUT-Methode zum Erstellen oder zum Aktualisieren einer Ressource genutzt werden. Der Unterschied zwischen der PUT- und der POST-Methode liegt darin, dass die PUT-Methode nur auf Ressourcen ausgeführt werden kann, die dem Client bekannt sind. Somit müssen die Ressourcen bereits existieren oder der Server muss das Erstellen von Ressourcen bzw. der URI durch den Client erlauben. Dem entgegen wird die POST-Methode dazu genutzt, um neue Ressourcen anzulegen, wobei der Server die URI der Ressource bestimmt.

Die beschriebene PUT-Methode erlaubt nur das aktualisieren einer Ressource, wenn alle Attribute des entsprechenden Objektes über das HTTP-Protokoll übertragen werden. Dies kann u.U. dazu führen, dass bei relativ großen Objekten sehr viele Daten übertragen werden müssen. Aus diesem Grund wird hier eine weitere HTTP-Methode eingeführt, die *PATCH-Methode*, welche zusätzlich partielle Updates von Ressourcen ermöglicht. Die folgende Beschreibung bezieht sich auf das Dokumentenformat *JSON Merge Patch*[12], wobei zusätzlich zur PATCH-Methode der Content-Type: *application/merge-patch+json* angeben werden sollte.

- **PATCH**: Im Zusammenhang mit dem Dokumentenformat 'JSON Merge Patch' kann ein partielles Update auf einer Ressource durchgeführt werden. Somit entspricht die PATCH-Methode der Update-Methode der CRUD-Operationen um. Ein konkretes Beispiel findet sich unter dem angegebenen Link[12].

2.5 Zeitzonen

OpenInfRA ist dafür vorgesehen, in unterschiedlichen Teilen der Welt eingesetzt zu werden. Damit einhergehend können Daten in unterschiedlichen Zeitzonen erstellt werden. Um eine einheitliche Speicherung zu ermöglichen, müssen alle Zeitstempel in der koordinierten Weltzeit (UTC) hinterlegt werden. Dafür muss der PostgreSQL Server in seiner Konfiguration entsprechend angepasst werden.

[12] https://tools.ietf.org/html/rfc7386

Für die einheitliche Darstellung des Datums- und Zeitformates wird ISO 8601[13] angewandt. Dabei wird die Datumsangabe in der Form *JJJJ-MM-TT* und die Zeitangabe in der Form *hh:mm:ss* umgesetzt. Der Unterschied der verwendeten Zonenzeit zur koordinierten Weltzeit (UTC) wird in der Form *+/-hhmm* angegeben. Ein Beispiel für eine deutsche Zeitangabe wäre 2015-03-10 13:26:41+0100.

2.6 Universally Unique Identifier

Eine *Universally Unique Identifier (UUID)* ist ein spezieller Identifikator, der nach einem Standard definiert und in Bezug auf die Menge aller UUIDs global eindeutig ist[14]. UUIDs werden in OpenInfRA genutzt, um Objekte, die in den Verteilten Datenbanken verwaltet werden, eindeutig identifizieren zu können.

2.7 Sprachumgebung

Für OpenInfRA ist ein umfangreiches Konzept zur Umsetzung einer Mehrsprachigkeit vorgesehen. Ziel dieses Kapitels ist eine kurze Erläuterung von zugehörigen Begrifflichkeiten, die im Folgenden erläutert werden. Eine *Sprachumgebung* ist eine Kombination aus einer Sprachkodierung und einer Zeichenkodierung. Zusätzlich kann eine Länderkodierung definiert werden [DIN10].

2.7.1 Sprachkodierung

Eine Sprachkodierung ist ein nach *ISO 639* definierter Wert[15], welcher durch zwei Kleinbuchstaben eine konkrete Sprache referenziert (bspw.: de – Deutsch; en – Englisch).

Zusätzlich zum aufgeführten Standard (ISO 639)[15] wurde hier für unbekannte Sprachen der Sprachcode *xx* zusätzlich für OpenInfRA hinzugefügt. Dieser Sprachcode beschreibt eine nicht definierte Sprache und bezieht sich auf den Sprachcode, der im ISO-Standard *ISO 639-2* mit *zxx* definiert ist[16].

[13] http://de.wikipedia.org/wiki/ISO_8601
[14] http://tools.ietf.org/html/rfc4122
[15] http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
[16] http://en.wikipedia.org/wiki/List_of_ISO_639-2_codes

2.7.2 Länderkodierung

Eine Länderkodierung ist ein nach *ISO 3166 Alpha 2* definierter Wert[17], welcher durch zwei Großbuchstaben ein konkretes Land referenziert (bspw.: DE – Deutschland; GB – Großbritannien).

2.7.3 Zeichenkodierung

Die Zeichenkodierung definiert, mit welcher Methode die Unicode-Zeichen[18] abgebildet werden. Als Standardformat sollte hier immer UTF-8 verwendet werden[19].

[17] http://de.wikipedia.org/wiki/ISO-3166-1-Kodierliste
[18] http://de.wikipedia.org/wiki/Unicode_Transformation_Format
[19] http://de.wikipedia.org/wiki/UTF-8

3 Komponenten

Die Abbildung 3 stellt die einzelnen geplanten Komponenten von OpenInfRA dar. Zunächst soll ein ORM für den Zugriff auf Objekte aus der Datenbank und dem Dateisystem eingesetzt werden. Mit Zugriff ist das Lesen, Speichern, Aktualisieren und Löschen von Objekten gemeint. Darauf aufbauend wird es einen Controller geben, der Anfragen an OpenInfRA in viele kleine Komponenten aufteilt. Diese stellen einzelne Funktionen zur Verfügung und bilden dadurch das Gesamtsystem. Mittels einer REST-Schnittstelle kann mit den, durch den Controller verwalteten, Objekten interagiert werden. Diese Interaktion ist auf der HTTP-Ebene notwendig, um Daten aus der Datenbank oder dem Dateisystem zur Verfügung zustellen oder diese zu manipulieren. Für die Verwendung von Elementen in der GUI, wird eine direkte Auslieferung von HTML-Fragmenten über den Controller ermöglicht, wodurch sich bestimmte Abläufe aus der GUI in das Backend verlagern lassen.

Die folgende Abbildung 4 liefert eine grobe Übersicht über die geplanten Software-Komponenten und deren Abhängigkeiten.

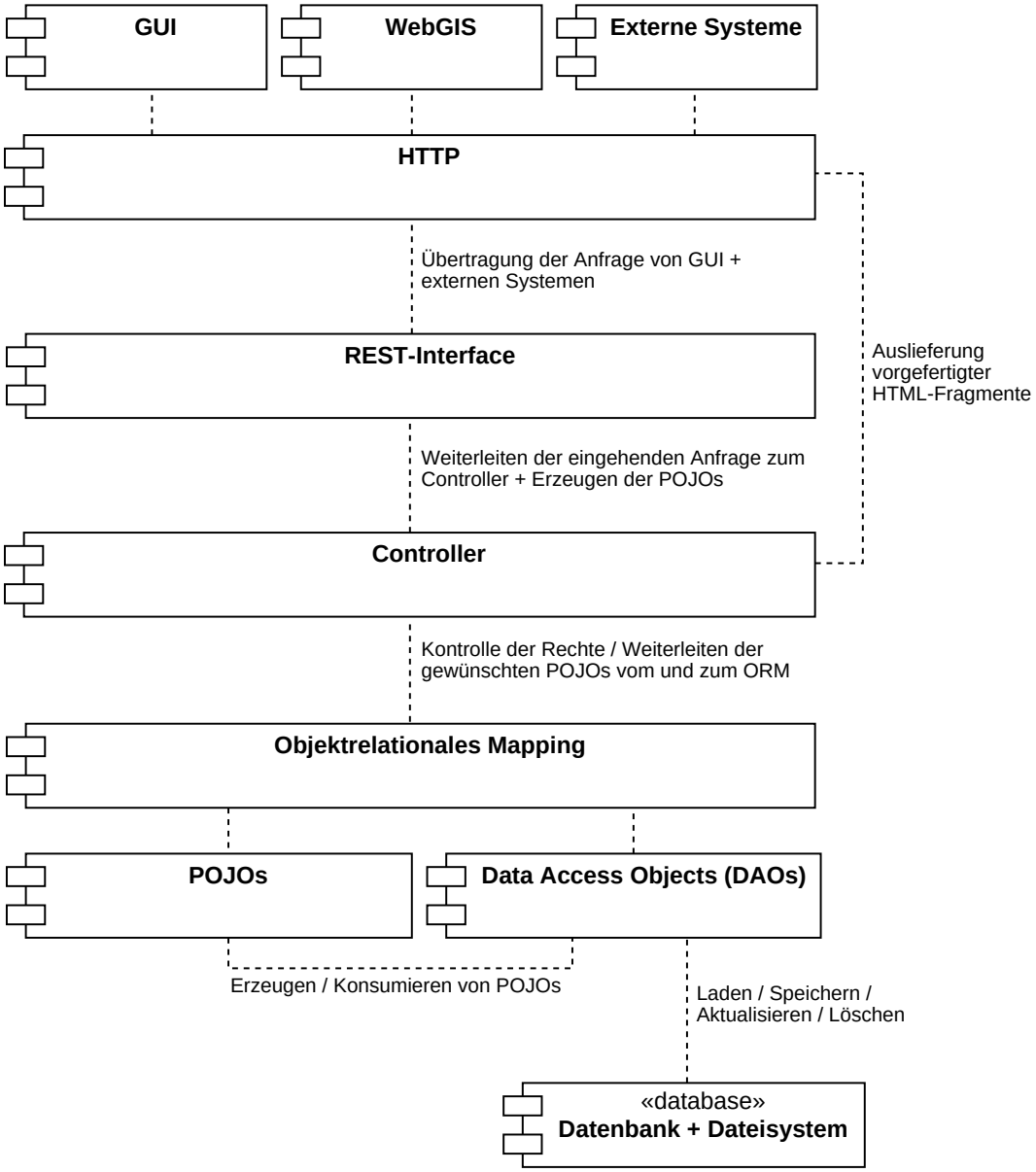


Abbildung 3: Logische Komponenten von OpenInfRA

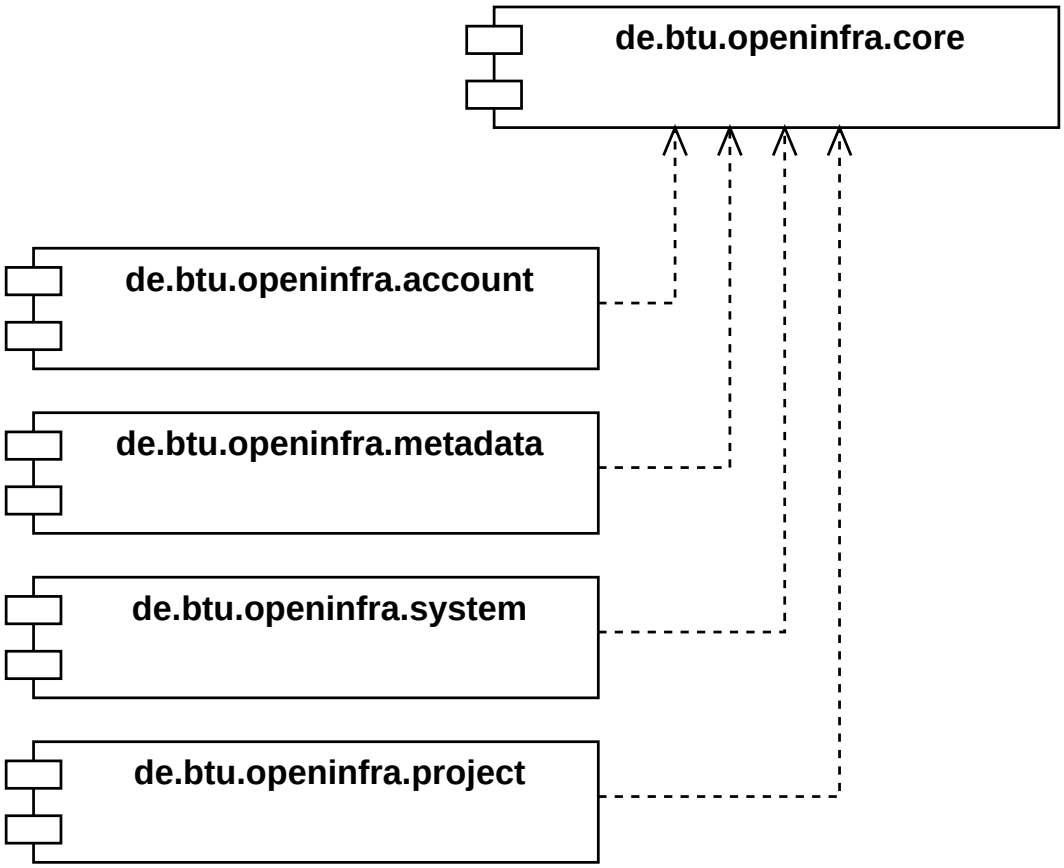


Abbildung 4: Software-Komponentendiagramm

4 Datenbank

Zur Haltung der Daten in OpenInfRA werden verschiedene Schemata benötigt. PostgreSQL erlaubt es, innerhalb einer Datenbank, mehrere benannte Schemata zu verwenden [Pos15b]. Das ermöglicht die strikte Trennung von unterschiedlichen Bereichen, ohne datenbankübergreifende Anfragen nutzen zu müssen. Neben den unter 4.2 erwähnten System- und Projektschematas, werden weitere für die Integritätsbedingungen (4.4), die Metainformationen (5.1) und die Benutzer- / Rollen- / Rechteverwaltung (9) benötigt.

4.1 Übersicht der verwendeten Datenbank-Schemata

In der Tabelle 1 sollen die geplanten Schemata und ihre inhaltlichen Bedeutungen dargestellt werden.

4.2 Fachdatenbank

Die Fachdatenbank bildet die zentrale Einheit zur Haltung von Fachanwenderdaten in OpenInfRA. Das Datenbankschema wurde so entwickelt, dass die Inhalte auf projektspezifische Bedürfnisse angepasst werden können, ohne das Schema selbst zu ändern. In OpenInfRA existieren zwei Arten von Schemata für Fachdatenbanken: die Projekt- und die Systemdatenbank. Die Systemdatenbank (siehe Abbildung 5) stellt dabei eine Teilmenge der Projektdatenbank (siehe Abbildung 6) dar. Darüber hinaus zeigen die Abbildungen 7, 8 und 9 weitere Konzepte der Fachdatenbank. An dieser Stelle soll nicht weiter auf die Details der Fachdatenbank eingegangen, sondern auf die Beschreibung im Grobkonzept [Ope14, S. 66 ff.] verwiesen werden.

Schemata	Bedeutung
account	Das Account-Schema umfasst sämtliche für die Benutzer- / Rollen- / Rechteverwaltung benötigten Tabellen.
constraints	Das Constraints-Schema umfasst alle Integritätsbedingungen, welche innerhalb der System- und Projektdatenbanken verwendet werden.
meta_data	Das MetaData-Schema umfasst u. a. die Resolver-, Protokollierungs-, Konfigurations- und Metadatentabellen.
system	Das System-Schema umfasst die System-Fachdatenbank und ihre entsprechenden Tabellen.
project_[id]	Das Projekt-Schema umfasst die Projekt-Fachdatenbank und ihre entsprechenden Tabellen. Jedes Projekt-Schema wird dabei durch eine Id gekennzeichnet. Diese Id ist eine UUID des Projektes in der dafür vorgesehenen Projekt-Tabelle (siehe 5.5). Damit die Zuordnung des Schemas zu Projekten besser lesbar ist, wird der konkrete Name des Projektes als Kommentar zum Schema gespeichert. Dieser kann z. B. innerhalb von *psql* über den Befehl \dn+ abgerufen werden.

Tabelle 1: Übersicht der verwendeten Datenbank-Schemata

4.3 Fachdatenspezifische Metadaten

Metadaten spielen eine wichtige Rolle in OpenInfRA. Sie werden für die Anreicherung der Fachdaten durch zusätzliche Informationen genutzt. Da diese Daten projektspezifisch sind, wird die in Kapitel 4.2 beschriebene Fachdatenbank um die in Abbildung 10 dargestellte Relation erweitert.

Die Speicherung der eigentlichen Informationen erfolgt in einem JSON-Objekt innerhalb der Datenbank. Die Nutzung von Metadaten ist optional und nicht funktionsrelevant. Damit das Backend die Metadaten nutzen kann, müssen verschiedene Schlüsselnamen fest definiert werden. Im Folgenden sollen diese Schlüssel benannt und deren Nutzen erklärt werden.

- *identifying_name*: Die angegeben *pt_free_text_id* und der dahinter stehende Name identifiziert ein Objekt namentlich. Das ist besonders für Objekte nötig, die keinen eigenen Namen oder Beschreibung besitzen, wie z.B. Themeninstanzen.

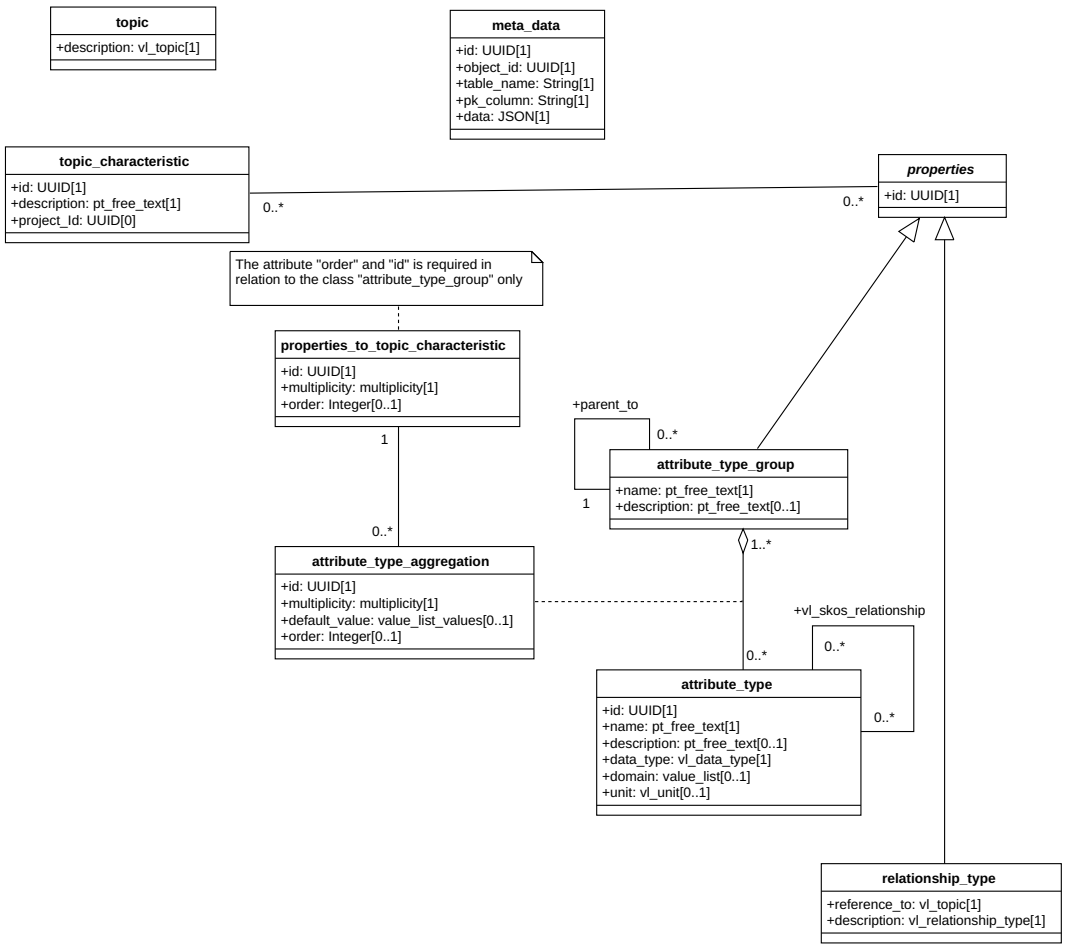


Abbildung 5: Systemdatenbank

- *last_modification*: Hier wird der Zeitstempel (siehe 2.5) der letzten Bearbeitung des Objektes gespeichert.

Neben diesen fest definierten Schlüsseln, können auch eigenständige Schlüssel angegeben werden. Diese können dann an geeigneter Stelle zu den jeweiligen Objekten angezeigt werden. Die Metadaten werden nicht durch Integritätsbedingungen überwacht.

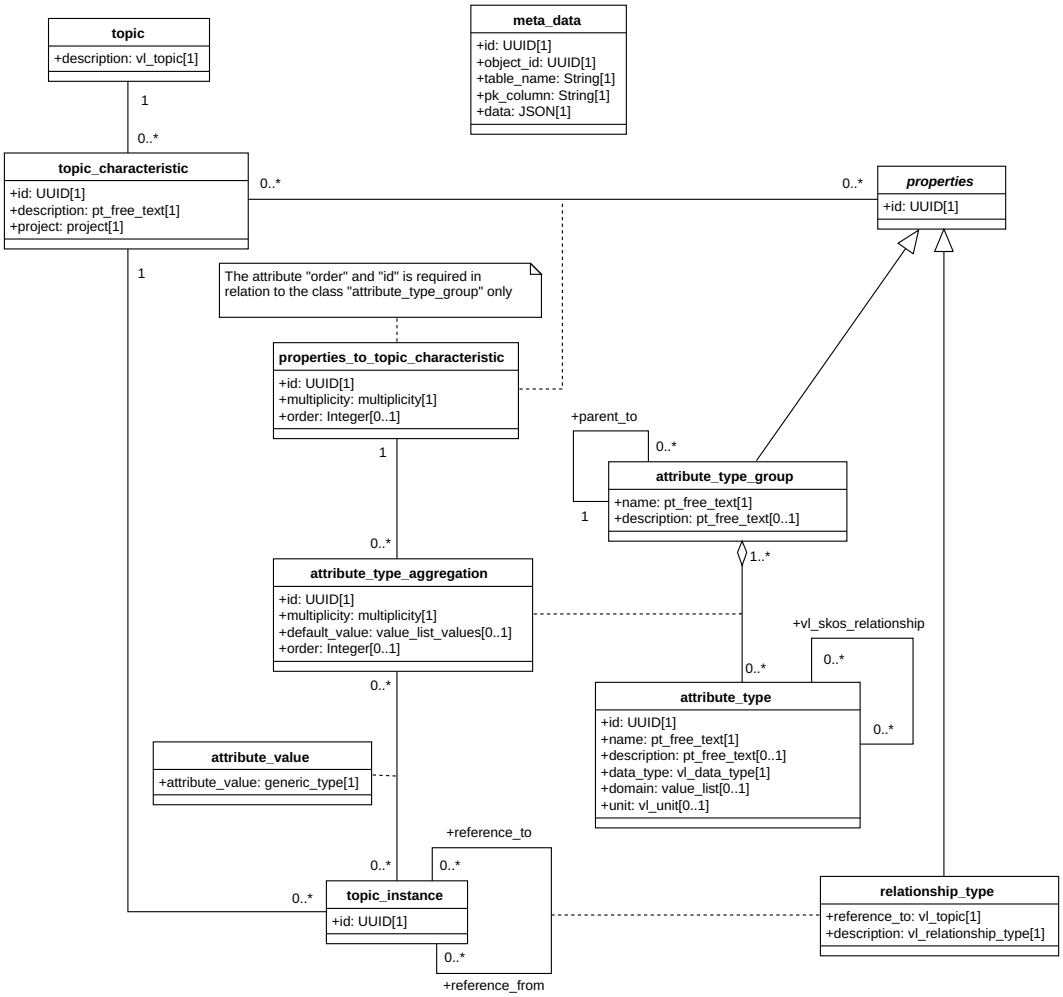


Abbildung 6: Projektdatenbank

4.4 Integritätsbedingungen

Das flexible Datenbankschema von OpenInfRA erfordert gewisse Vorkehrungen zur Sicherstellung der Integrität der enthaltenen Daten. Dafür wurden textuell Integritätsbedingungen (siehe [Ope15]) beschrieben. Diese Integritätsbedingungen teilen sich in einfache, durch das Datenbanksystem bereitgestellte (z. B. Primärschlüssel, NOT NULL, etc.) und komplexe Bedingungen. Die komplexen Integritätsbedingungen beziehen sich auf Synergien zwischen verschiedenen Tabellen. Die Operationen *INSERT*, *UPDATE* und *DELETE* müssen überwacht werden. Dies erfolgt durch die

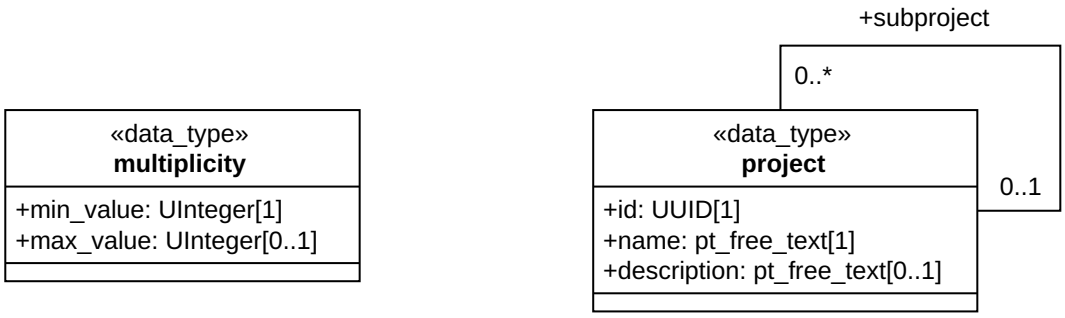


Abbildung 7: Datentypen Multiplicity und Project

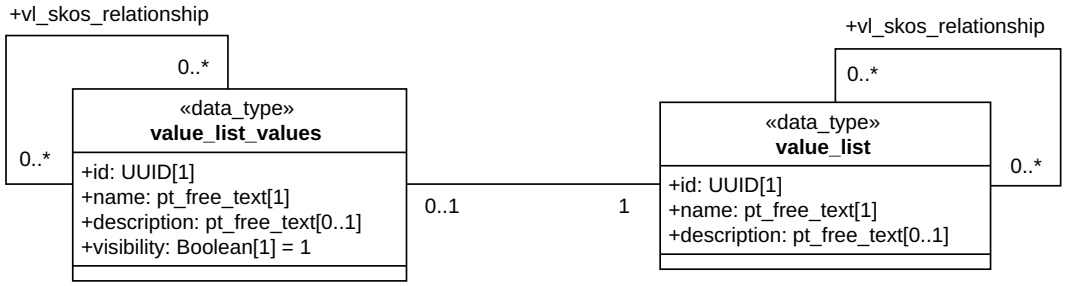


Abbildung 8: Datentypen ValueList und ValueListValues

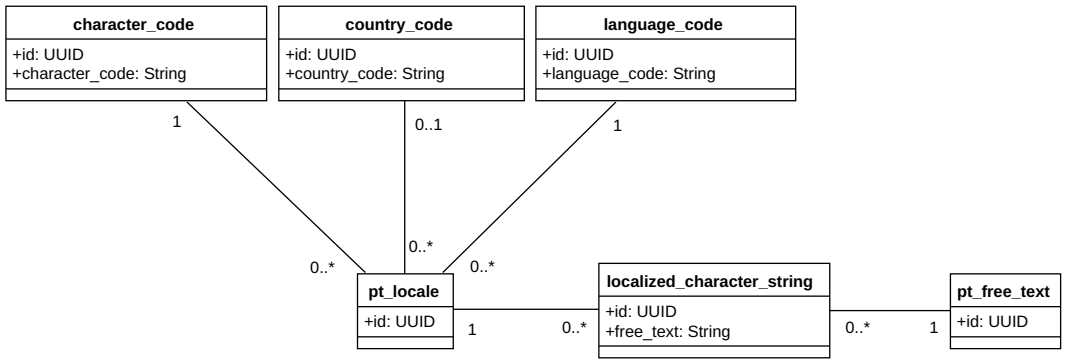


Abbildung 9: Mehrsprachigkeit

Montage von Triggern auf den entsprechenden Tabellen, welche vor dem Ausführen der genannten Operationen ausgeführt werden. Diese Trigger wiederum starten die o. g. komplexen Integritätsbedingungen, welche in Form von plpgsql-Funktionen hinterlegt sind. Wenn die Integritätsprüfung keine Fehler aufweist, werden die Operationen auf den Tabellen ausgeführt.

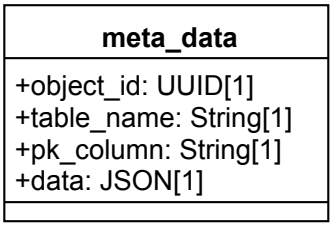


Abbildung 10: Projektspezifische Metadaten

5 Kern-System

5.1 Metadatenbank

Neben der Fach- (4.2) und Account-Datenbank (9.1), müssen in OpenInfRA noch weitere systemrelevante Informationen gespeichert werden. Für diese Aufgabe steht die Metadatenbank zur Verfügung. In ihr werden Protokolle, Systemkonfigurationen und die Verbindungen zu den Projekt-Schematas gehalten. Diese Metadaten sind unabhängig von den in Kaptitel 4.3 beschriebenen fachdatenspezifischen Metadaten.

5.2 Protokolle

Die Klasse *log* speichert Protokollierungen, die durch das System erstellt werden.

- *id (PK)*: Primärschlüssel für den Protokolleintrag
- *user_id*: Id des eingeloggten Benutzers, der den Protokolleintrag ausgelöst hat (kein Fremdschlüssel)
- *user_name*: Name des eingeloggten Benutzers, der den Protokolleintrag ausgelöst hat
- *created_on*: Zeitpunkt zu dem der Protokolleintrag erstellt wurde
- *logger*: Klasse die den Protokolleintrag ausgelöst hat
- *level*: Einstufung des Protokolleintrages (z. Bsp.: Debug, Info, Error, etc.)
- *message*: Nachricht die den Protokolleintrag beschreibt

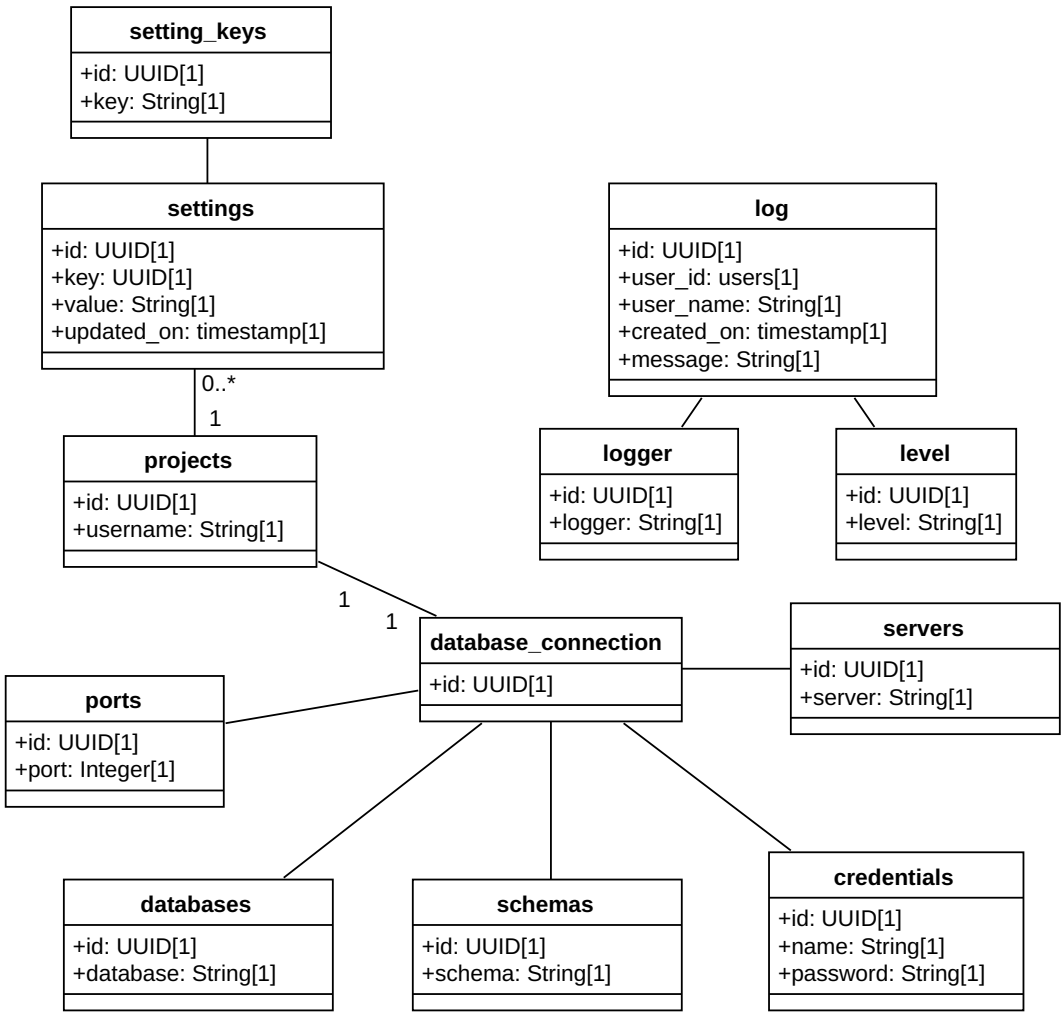


Abbildung 11: Metadatenbank

5.3 Systemkonfiguration

Die Klasse *settings* speichert sämtliche systemrelevante Einstellungen.

- *id (PK)*: Primärschlüssel
- *key*: Schlüsselname für die Konfiguration
- *value*: konkreter Konfigurationsparameter des Schlüssels
- *updated_on*: Datum, an dem die Konfiguration zuletzt angepasst wurde

5.4 Verbindungsdaten

Die Klasse *connection_string* beinhaltet die Datenbank Verbindungsdaten, welche sich auf die einzelnen Objekte der Resolver-Klasse beziehen.

- *id (PK)*: Primärschlüssel
- *server*: IP-Adresse oder DNS Name des Datenbankservers
- *port*: Port des Datenbankservers
- *database*: Name der Datenbank
- *schema*: Name des PostgreSQL Schema in dem sich das Objekt befindet
- *credentials*: Zugangsdaten für die Datenbankverbindung

5.5 Projekte

Die Klasse *projects* definiert alle Projekte, welche ein eigenes Fachdaten-Schema besitzen und somit ein Hauptprojekt sind.

- *id (PK)*: Primärschlüssel
- *name*: Name des Projektes ohne Bezug zu einer Sprache (derselbe Name wird als Kommentar zum jeweiligen Projekt-Schema gespeichert)

6 Dateisystem

6.1 Dateistruktur

Das Dateisystem stellt, neben der Datenbank, die Basis für OpenInfRA dar. Der dargestellte Verzeichnisbaum verdeutlicht den strukturellen Aufbau des Systems. Hinter den Verzeichnisnamen befinden sich Kommentare zu dessen Bedeutung.

```
openinfra ..... root
├── config ..... globale Konfigurationen für OpenInfRA
├── plugins ..... alle Plug-Ins
│   └── [plugin name] ..... konkretes Plug-In
│       └── config ..... Konfigurationen für Plug-Ins
├── tmp ..... temporäre Dateien
├── projectdata ..... Daten für Projekte
│   └── project_[id] ..... konkretes Projekt
│       ├── config ..... projektspezifische Konfigurationen, notwendig?
│       ├── upload ..... hochgeladene Dateien
│       │   ├── files ..... Dokumente (z. Bsp. pdf, doc, etc.)
│       │   └── images ..... Bilder
│       │       ├── lowres ..... Bilder in verringerter Auflösung
│       │       ├── original ..... Bilder in Originalgröße
│       │       └── thumb ..... Thumbnails der Bilder
│       └── xslt ..... projektspezifische XSLT für Export
│           ├── csv ..... projektspezifische XSLT für CSV-Export
│           └── pdf ..... projektspezifische XSLT für PDF-Export
└── xslt ..... globale XSLT für Export
    ├── csv ..... globale XSL für CSV-Export
    └── pdf ..... globale XSLT für PDF-Export
```

Das Verzeichnis *config* taucht in unterschiedlichen Hierarchiestufen auf. Die dort enthaltenen Dateien ergänzen bzw. erweitern sich einander.

6.2 Benutzerdateien

Jeder Benutzer mit geeigneter Berechtigung, kann innerhalb eines Projektes Dateien hochladen. Dabei wird zwischen Rastergrafiken (z. Bsp. JPG, TIFF, etc.) und anderen Formaten (DOC, PDF, SHP, SVG etc.) unterschieden. Rastergrafiken erhalten, im Vergleich zu den restlichen Formaten eine gesonderte Behandlung. Dort werden nach dem übertragen der Rastergrafiken an den Server, zwei unterschiedliche Auflösungsstufen erzeugt. Zum einen ein Vorschaubild, welches der verkleinerten Anzeige z. Bsp. innerhalb einer Themeninstanz dient, und zum anderen ein Bild in verringerter Auflösung.

Damit die Benutzerdateien mit den Fachdaten verknüpft werden können, werden neue Datentypen eingeführt und in die statische Werteliste aufgenommen. Dabei handelt es sich um die folgenden Datentypen:

- image: Attributtypen mit diesem Datentyp speichern Dateinamen von Rastergrafiken
- file: Attributtypen mit diesem Datentyp speichern Dateinamen anderer Formate
- url: Attributtypen mit diesem Datentyp speichern URLs

7 Objektrelationales Mapping

Das objektrelationale Mapping bzw. die objektrelationale Abbildung bezieht sich auf die unterste Anwendungsschicht: Datenbank- und Dateisystemzugriff (vgl. Abbildung 1). Das objektrelationale Mapping hat zur Aufgabe, die Daten bzw. Objekte, die in der Anwendungsschicht verwendet werden, mit einem relationalen DBMS austauschen zu können. Dazu wird hier zwischen dem *Datenmodell* und dem *Datenzugriff* unterschieden. Das Datenmodell stellt die konkreten Datenobjekte, die hier als *Plain Old Java Objects (POJOs)* bezeichnet werden, in der Anwendungsschicht bereit. Der Datenzugriff stellt Zugriffsobjekte bereit, die hier als *Data Access Objects (DAOs)* bezeichnet und zum Erzeugen bzw. Speichern der Datenobjekte verwendet werden.

Durch die Bereitstellung der POJOs wird ein sehr feingranularer Zugriff auf die in der Datenbank verwalteten Objekte ermöglicht. Durch diesen feingranularen Zugriff können die Objekte je nach Bedarf individuell zusammengestellt werden. Zusätzlich wird aber auch gefordert diesen feingranularen Zugriff für ausgewählte Objekte, die sich aus mehreren POJOs zusammensetzen, zu vereinfachen. Diese zusammengesetzten Objekte werden durch eine *Komposition*[1] beschrieben, im Weiteren als *Business Objekte* bezeichnet und bilden neben den POJOs eine ganz eigene Klasse von Objekten. Die Erzeugung dieser 'Business Objekte' wird durch eine spezielle DAOs bereitgestellt.

Das für OpenInfRA entworfene Datenbankschema ist aufgrund der Konzepte für die Mehrsprachigkeit und wegen der Adaption der nötigen Abläufe der archäologischen Forschung im Feld bzw. der Veröffentlichungsmöglichkeiten der resultierenden Forschungsergebnisse sehr speziell und komplex. Das Ziel des vorgeschlagenen objektrelationalen Abbildung ist der Entwurf einer maßgeschneiderten Lösung, die in der Lage ist, dieses spezielle Datenbankschema für die Verwendung mittels Web-basierter Technologien vorzubereiten.

7.1 Datenmodell (POJO)

Die folgende Abbildung 12 zeigt eine Übersicht über das Datenmodell. Üblicherweise werden Interfaces für die Implementierung von Datentypen modelliert. In diesem Fall

[1] http://de.wikipedia.org/wiki/Assoziation_(UML)#Komposition

wird davon abgesehen, weil es sich hier um POJOs handelt, die ausschließlich Getter- und Setter-Methoden bereitstellen. Weitere Funktionalitäten, wie z.B. das Ändern konkreter Attributwerte sind für diese POJOs nicht geplant.

Wie in der Abbildung 12 zu erkennen ist, gibt es im Datenmodell drei abstrakte Klassen, die nicht von anderen Klassen erben: *Name*, *Description* und *OpeninfraDataObject*. Die abstrakte Klasse 'Name' stellt einen Datencontainer dar, der eine Menge von Namen sowie die jeweils zugehörigen Sprachumgebungen bereitstellt. Die abstrakte Klasse 'Description' stellt einen Datencontainer dar, der eine Menge von Beschreibungen sowie die jeweils zugehörigen Sprachumgebungen bereitstellt. Alle Klassen, die einen Namen bzw. eine Beschreibung enthalten, erben von diesen beiden Klassen. Die abstrakte Klasse 'OpeninfraDataObject' ist ein Datencontainer, der eine UUID bereitstellt. Alle OpenInfRA-Klassen, außer den Klassen 'Name' und 'Description', erben von dieser Klasse.

Wie in der Abbildung 12 dargestellt ist, erbt auch die Klasse *PtLocale* von der Klasse *OpeninfraDataObject* und enthält somit eine UUID. Die Klasse 'PtLocale' bildet die Sprachumgebung (vgl. Kapitel 2.7) ab, die sich aus einer Sprachkodierung (*countryCode*), einer Länderkodierung (*languageCode*) und einer Zeichenkodierung (*characterCode*) zusammensetzt.

Weiterhin erbt auch die Klasse *Multiplicity* von der Klasse *OpeninfraDataObject* und erhält somit ebenfalls eine UUID. Die Klasse 'Multiplicity' beschreibt die Multiplizität einer Beziehung zwischen zwei Objekten mittels eines Minimums (*min*) und eines Maximums (*max*).

Aufgrund der Konzepte für die Mehrsprachigkeit im Datenbankschema von OpenInfRA, kann ein Objekt in der Datenbank mehrere Sprachumgebungen besitzen. Wird die Datenbank angefragt, könnten für jedes Objekt alle Sprachumgebungen inklusive der konkreten Textobjekte aus der Datenbank geladen werden. Um dies zu vermeiden ist das Datenmodell so entworfen, dass ein konkretes Objekt immer auf eine bestimmte Menge an Sprachumgebungen plus die in den jeweiligen Sprachen zugehörigen Textobjekte bezieht. Dies wurde so modelliert, weil angenommen wird, dass ein Bearbeiter immer nur mit einer vorher definierten Menge von Sprachen mit dem System arbeitet. Zusätzlich wird somit der Aufwand reduziert, weil nicht immer alle Sprachumgebungen mit den zugehörigen Textobjekten aus der Datenbank geladen werden müssen. Die Anfrage an die Datenbank definiert feingranular, welche Sprachen geladen werden sollen. Aufgrund der Konzepte für die Mehrsprachigkeit kann es sein, dass eine angefragte Sprache nicht zur Verfügung steht. In diesem Fall soll eine andere Sprache als die angefragte geliefert werden. Dafür wurde die abstrakte Klasse *OpeninfraRequest* eingefügt, welche die ursprünglich angefragten Sprachen enthält, die bei der Datenbank abgefragt wurden. Somit ist der Unterschied zwischen

den Sprachen, die in der Anfrage konkret spezifiziert wurden, und den Sprachen der zurückgelieferten Textobjekten eindeutig nachvollziehbar.

Im Folgenden werden die Klassen: *Project*, *TopicCharacteristic*, *RelationshipType*, *AttributeType*, *AttributeTypeGroup*, *AttributeTypeGroupToTopicCharacteristic*, *TopicInstance*, *AttributeValue* und *ValueList* detailliert beschrieben.

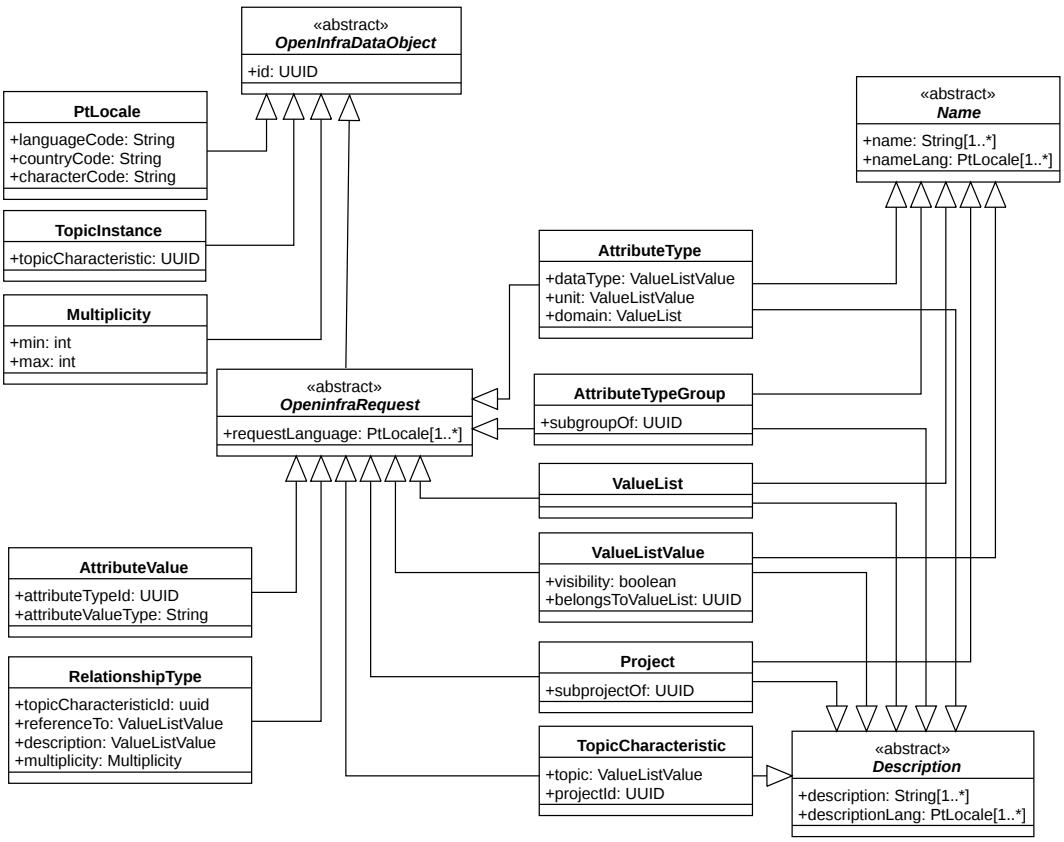


Abbildung 12: Übersicht über das Datenmodell

7.1.1 Die Klasse Project

Die folgende Abbildung 13 zeigt die Klasse *Project*.

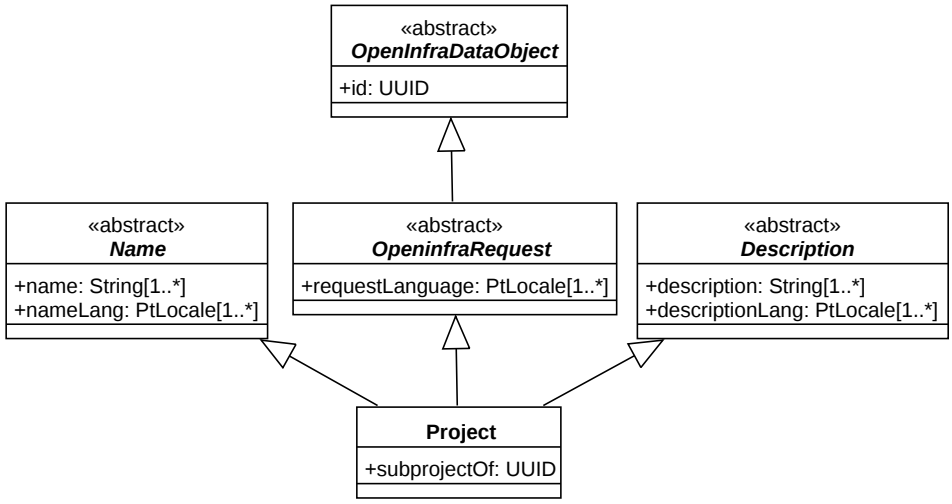


Abbildung 13: Die Klasse Project

7.1.2 Die Klasse TopicCharacteristic

Die folgende Abbildung 14 zeigt die Klasse *TopicCharacteristic*.

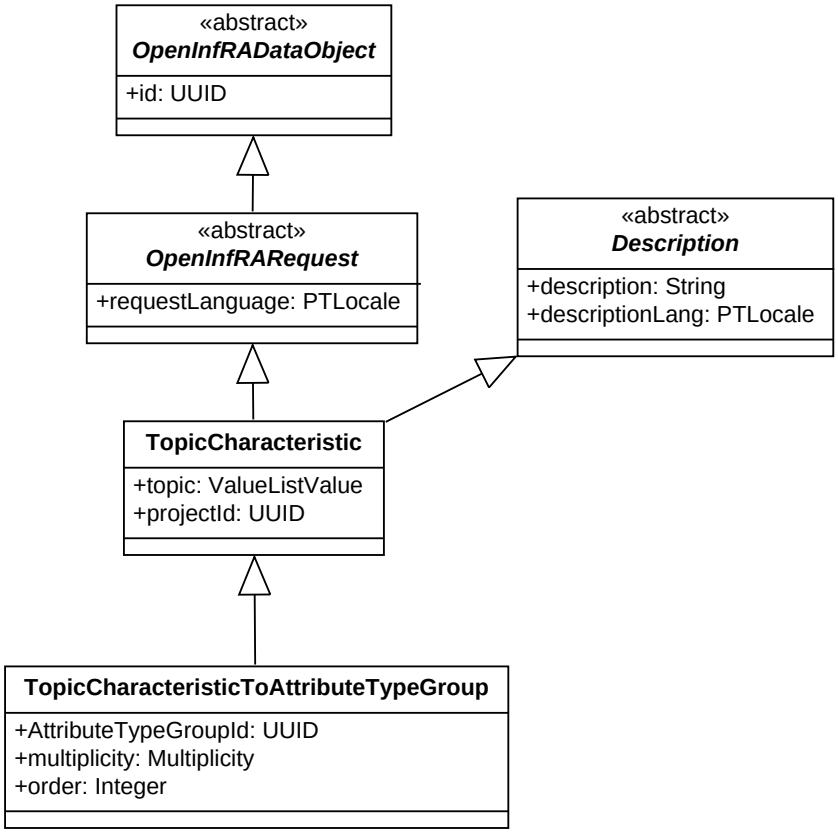


Abbildung 14: Die Klasse TopicCharacteristic

7.1.3 Die Klasse RelationshipType

Die folgende Abbildung 15 zeigt die Klasse *RelationshipType*.

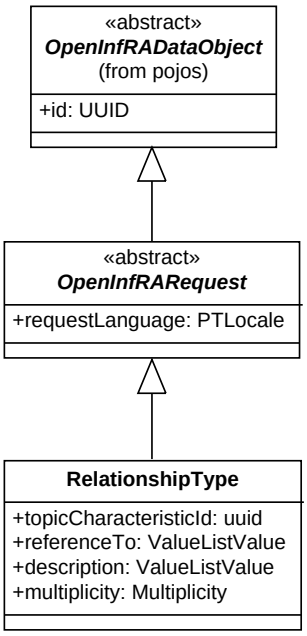


Abbildung 15: Die Klasse RelationshipType

7.1.4 Die Klasse RelationshipTypeToTopicCharacteristic

Die folgende Abbildung 16 zeigt die Klasse *RelationshipTypeToTopicCharacteristic*.

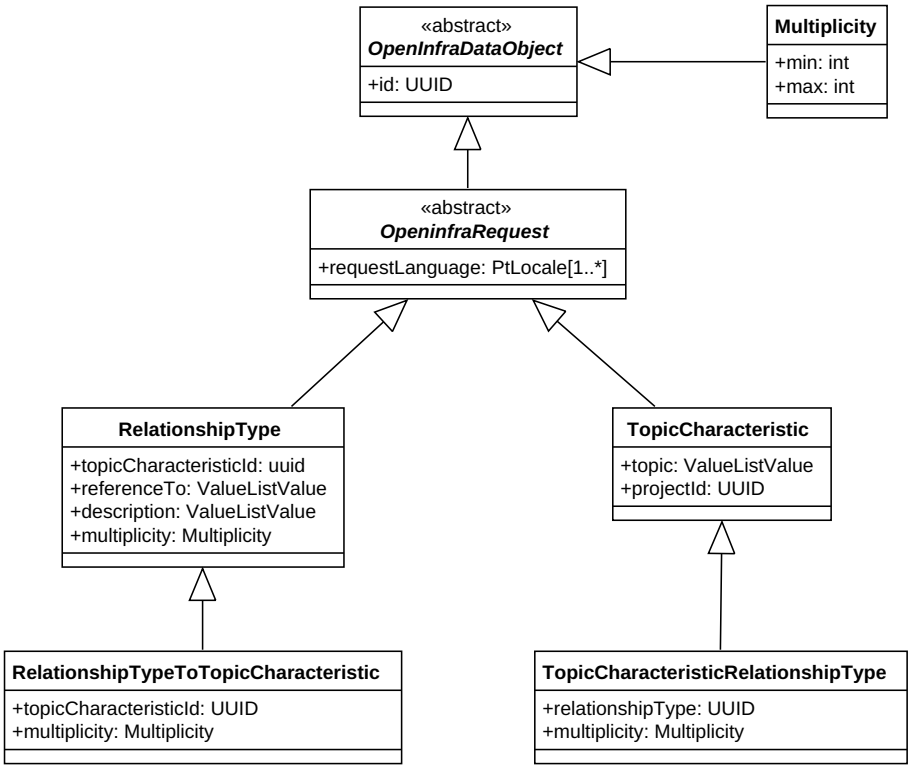


Abbildung 16: Die Klasse RelationshipTypeToTopicCharacteristic

7.1.5 Die Klasse AttributeType

Die folgende Abbildung 17 zeigt die Klasse *AttributeType*.

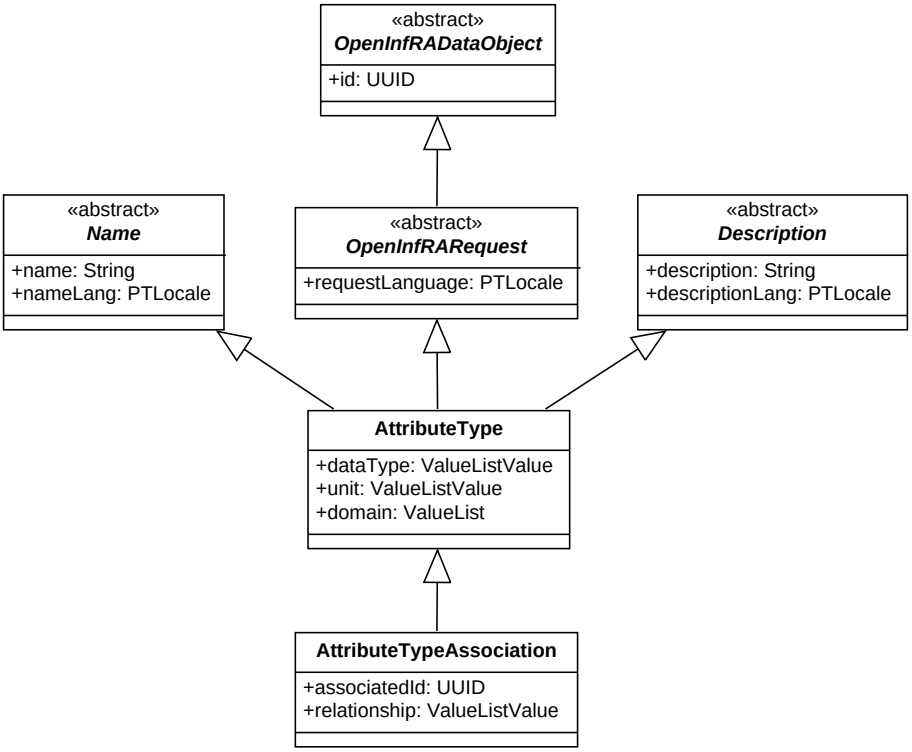


Abbildung 17: Die Klasse AttributeType

7.1.6 Die Klasse AttributeTypeGroup

Die folgende Abbildung 18 zeigt die Klasse *AttributeTypeGroup*.

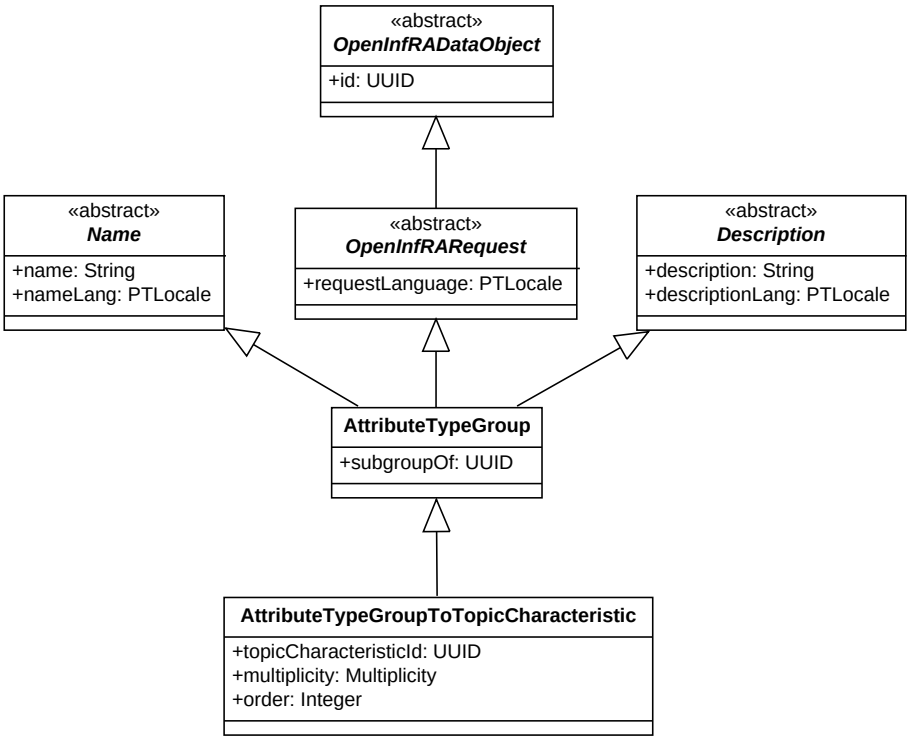


Abbildung 18: Die Klasse AttributeTypeGroup

7.1.7 Die Klasse AttributeTypeToAttributeTypeGroup

Die folgende Abbildung 19 zeigt die Klasse *AttributeTypeToAttributeTypeGroup*.

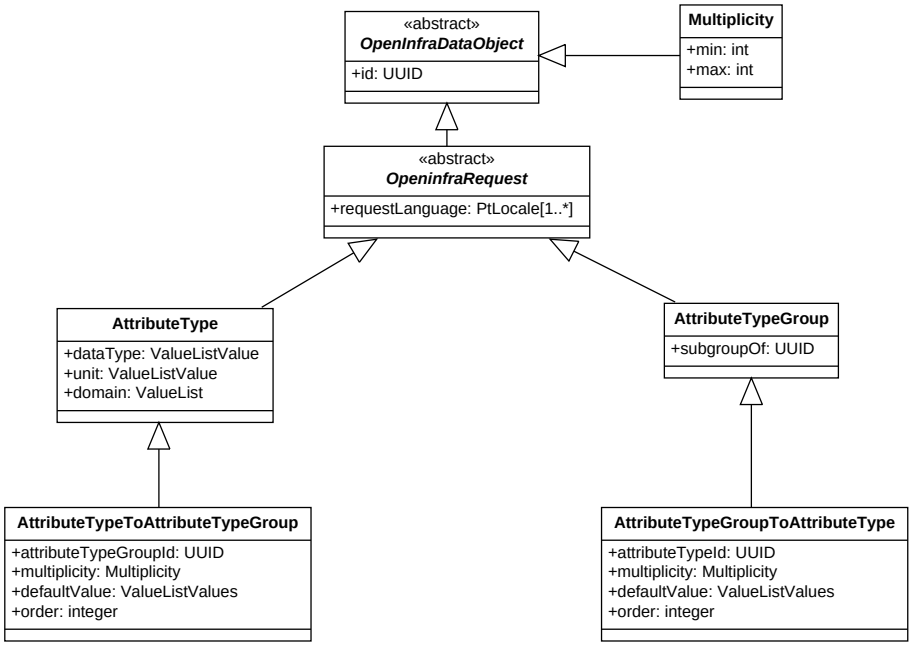


Abbildung 19: Die Klasse AttributeTypeToAttributeTypeGroup

7.1.8 Die Klasse AttributeTypeGroupToTopicCharacteristic

Die folgende Abbildung 20 zeigt die Klasse *AttributeTypeGroupToTopicCharacteristic*.

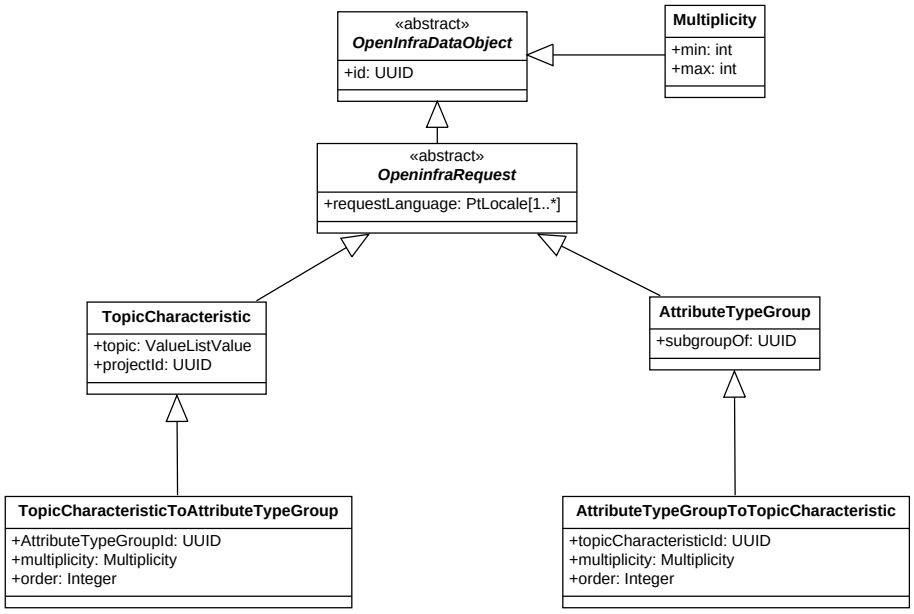


Abbildung 20: Die Klasse AttributeTypeGroupToTopicCharacteristic

7.1.9 Die Klasse TopicInstance

Die folgende Abbildung 21 zeigt die Klasse *TopicInstance*.

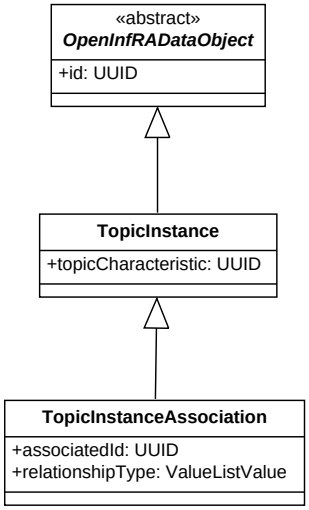


Abbildung 21: Die Klasse TopicInstance

7.1.10 Die Klasse AttributeValue

Die folgende Abbildung 22 zeigt die Klasse *AttributeValue*. Die Klasse 'AttributeValue' enthält eine Variable *attributeValueType*, welchen den konkreten Typen des Attributwertes in Textform enthält.

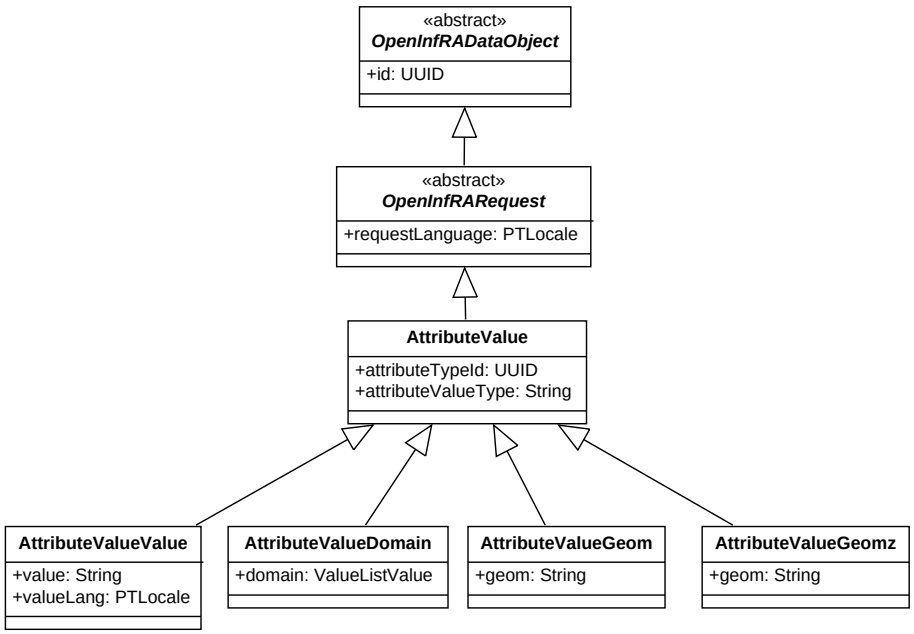


Abbildung 22: Die Klasse AttributeValue

7.1.11 Die Klasse ValueLists

Die folgende Abbildung 23 zeigt die Klassen *ValueList* und *ValueListValue*.

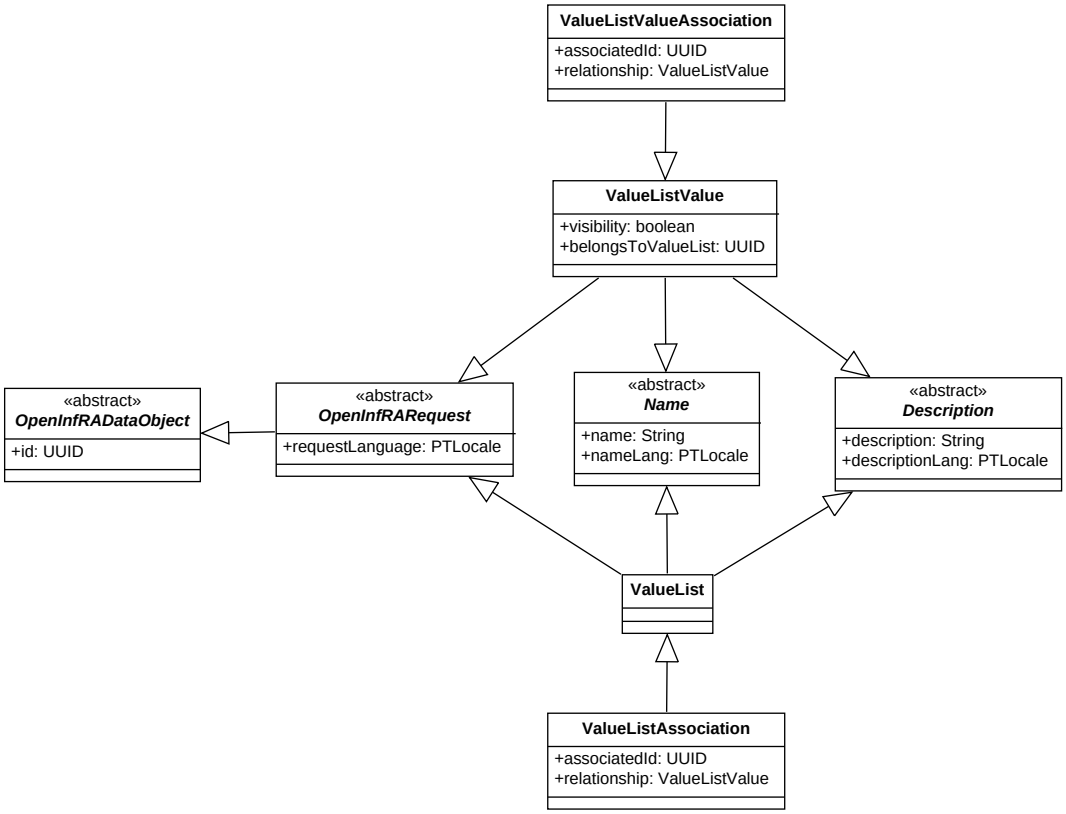


Abbildung 23: Die Klassen ValueList und ValueListValue

7.2 Datenzugriff (DAO)

7.2.1 Die Klasse ProjectDao

Die folgende Abbildung 24 zeigt die Klasse *ProjectDao*.

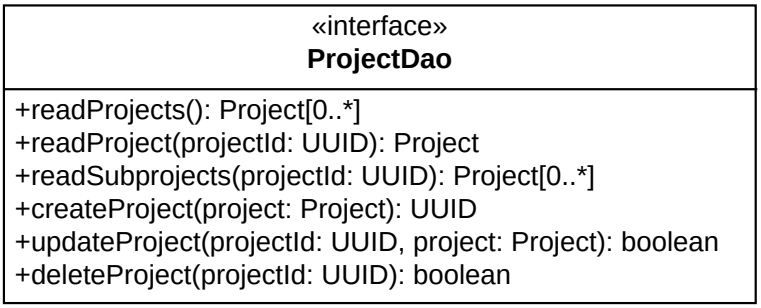


Abbildung 24: Die Klasse für ProjectDao

7.2.2 Die Klasse TopicCharacteristic

Die folgende Abbildung 25 zeigt die Klasse *TopicCharacteristic*.

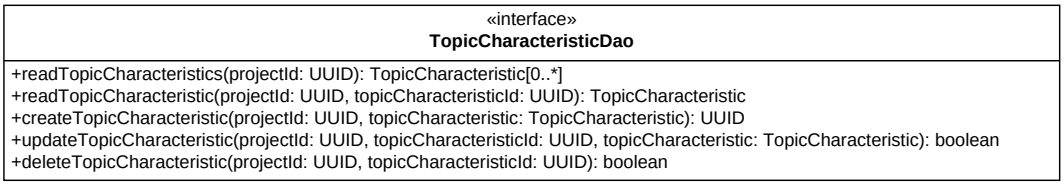


Abbildung 25: Die Klasse für TopicCharacteristic

7.2.3 Die Klasse RelationshipType

Die folgende Abbildung 26 zeigt die Klasse *RelationshipType*.

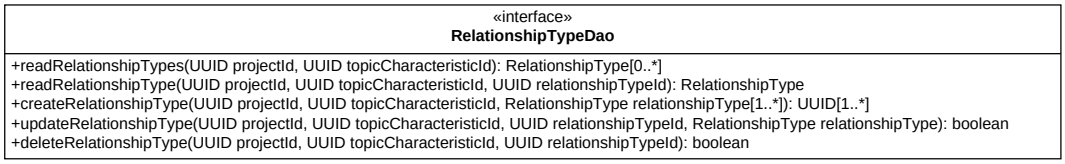
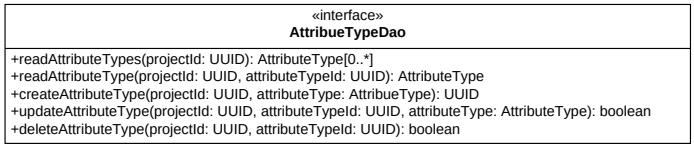


Abbildung 26: Die Klasse für RelationshipType

7.2.4 Die Klasse AttributeType

Die folgende Abbildung 27 zeigt die Klassen für *AttributeType*.



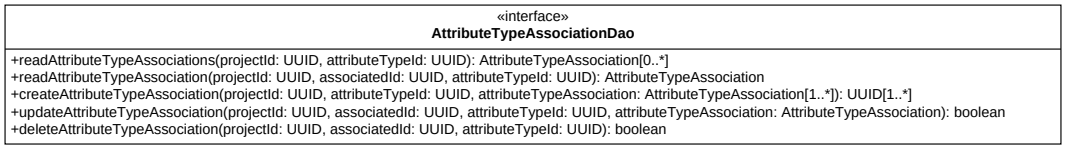


Abbildung 27: Die Klassen für AttributeType

7.2.5 Die Klasse AttributeTypeGroup

Die folgende Abbildung 28 zeigt die Klassen für *AttributeTypeGroup*.

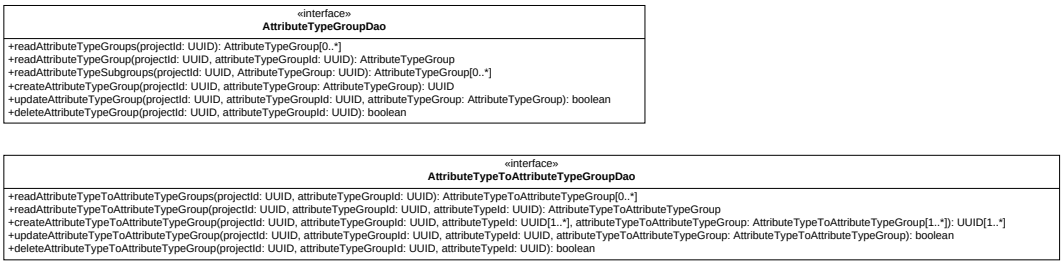


Abbildung 28: Die Klassen für AttributeTypeGroup

7.2.6 Die Klasse AttributeTypeToAttributeTypeGroup

Die folgende Abbildung 29 zeigt die Klassen für *AttributeTypeToAttributeTypeGroup*.

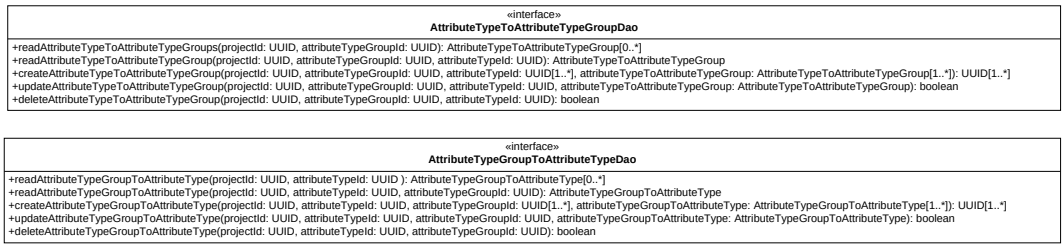


Abbildung 29: Die Klassen für AttributeTypeToAttributeTypeGroup

7.2.7 Die Klasse AttributeTypeToTopicCharacteristic

Die folgende Abbildung 30 zeigt die Klassen für *AttributeTypeToTopicCharacteristic*.

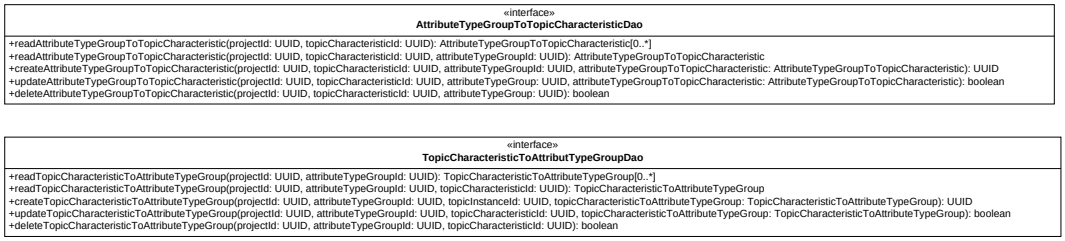


Abbildung 30: Die Klassen für AttributeTypeToTopicCharacteristic

7.2.8 Die Klasse TopicInstance

Die folgende Abbildung 31 zeigt die Klassen für *TopicInstance*.

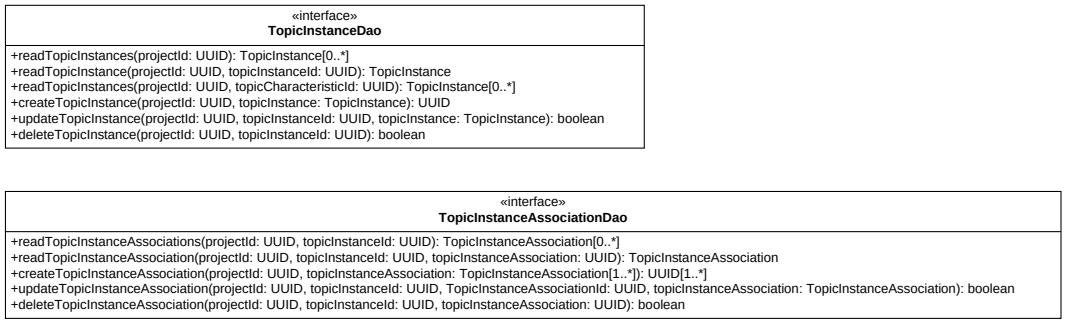


Abbildung 31: Die Klassen für TopicInstance

7.2.9 Die Klasse AttributeValue

Die folgende Abbildung 32 zeigt die Klasse *AttributeValue*.

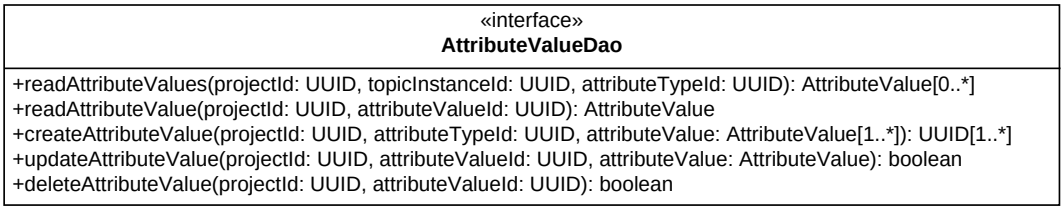


Abbildung 32: Die Klasse für AttributeValue

7.2.10 Die Klasse ValueList

Die folgende Abbildung 33 zeigt die Klassen für *ValueList*.

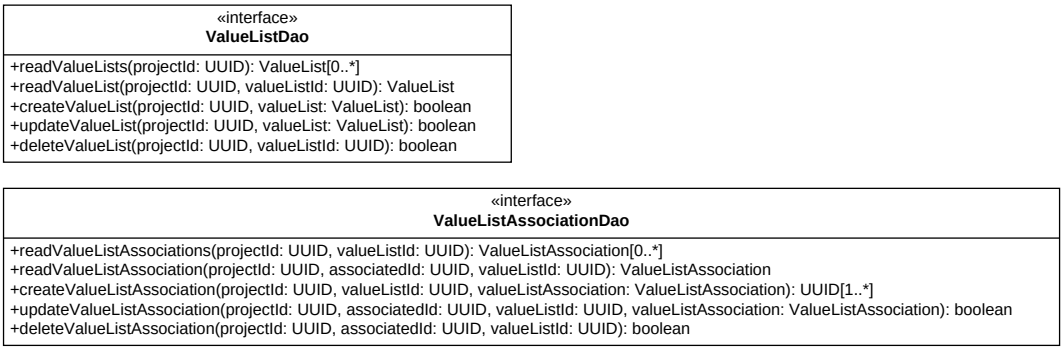


Abbildung 33: Die Klassen für ValueList

7.2.11 Die Klasse ValueListValues

Die folgende Abbildung 34 zeigt die Klassen für *ValueListValues*.

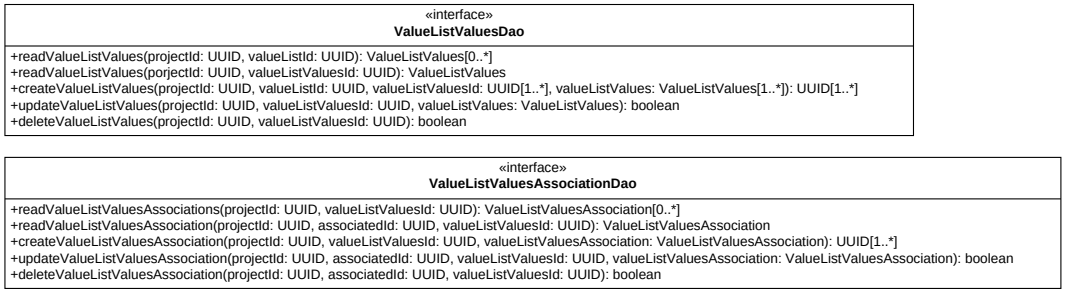


Abbildung 34: Die Klassen für ValueListValues

7.3 Business Objekte

Wie bereits beschrieben stellen die Business Objekte eine Komposition der POJOs dar.

7.3.1 Die Klasse TopicInstanceBo

Die folgende Abbildung 35 zeigt die Klasse *TopicInstanceBo*.

7.3.2 Die Klasse AttributeTypeGroupBo

Die folgende Abbildung 36 zeigt die Klasse *AttributeTypeGroupBo*.

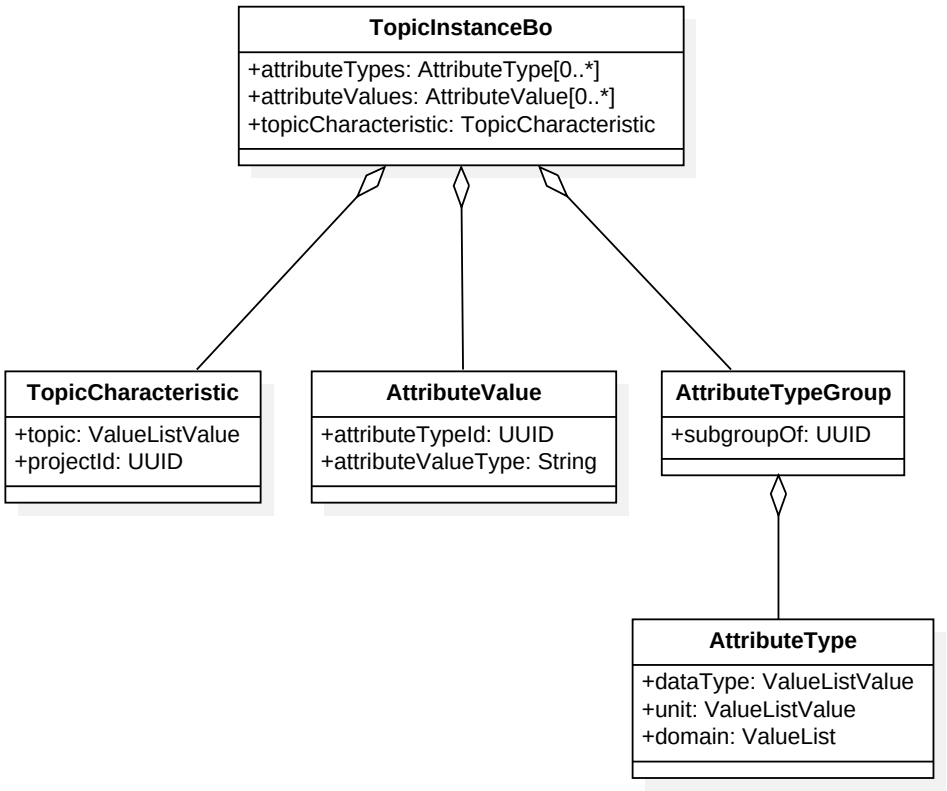


Abbildung 35: Die Klasse TopicInstanceBo

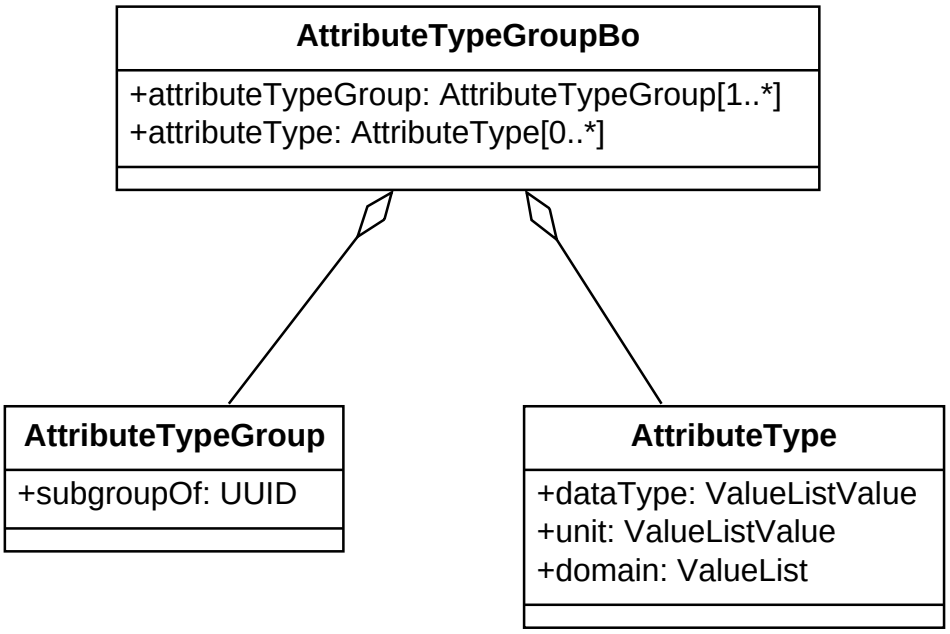


Abbildung 36: Die Klasse AttributeTypeGroupBo

8 URI-Mapping

Wie im aktuellen Standard [BLFM05] definiert, besteht eine URI aus fünf Teilen: dem *Schema* (scheme), dem *Anbieter* (authority), einem *Pfad* (path), einer *Abfrage* (query) und einem *Teil* (fragment). Das folgende Beispiel veranschaulicht den Aufbau einer URI:

foo://	example.com:8042	/over/there	?name=ferret	#nose
Schema	Anbieter	Pfad	Abfrage	Teil

Der Pfad wird hier wie folgt eingeteilt: *System* (system) und *Projekte* (projects). Der Pfad 'System' soll Ressourcen zur Verfügung stellen, die sich auf die Systemdatenbank beziehen. Der Pfad 'Projekte' soll Ressourcen zur Verfügung stellen, die sich auf die Projektdatenbanken beziehen.

- /system
- /projects

Für den Zugriff auf einzelne Tupel in der Datenbank müssen z.T. Identifikatoren übergeben werden. Zu diesen Identifikatoren gehören UUIDs und benutzerdefinierte Ids. Eine benutzerdefinierte Id ist ein durch den Benutzer frei wählbarer Identifikator, welcher ausschließlich für Projekte und Themeninstanzen vergeben werden kann. Zur besseren Übersichtlichkeit wird bei der Beschreibung der URIs im Folgenden das Akronym *id* verwendet, wenn sowohl eine UUID als auch eine benutzerdefinierte Id verwendet werden kann:

z.B. wird */projects/*{*uuid* | *benutzerdefinierte Id*}
zu
/projects/{*id*}.

Sonst, wenn nur eine UUID erlaubt ist, wird das Akronym *uuid* verwendet:

z.B. */system/languageCodes/*{*uuid*}.

8.1 Allgemeine Hinweise

In diesem Kapitel werden allgemeine Hinweise zur Verwendung der URIs und die zugehörigen HTTP-Methoden gegeben.

8.1.1 Groß- und Kleinschreibung von URIs

Bei den angegebenen URIs ist nach dem Standard (vgl. [BLFM05]) die Groß- und Kleinschreibung von Bedeutung. Mit anderen Worten eine URI ist case-sensitive.

8.1.2 Anwendung der HTTP-Methoden PUT und POST

Bei Objekten, welche andere Objekte in ihren Eigenschaften vereinen (z.B.: enthält das Objekt *AttributeTypeGroupToTopicCharacteristic* ein Objekt *Multiplicity* als Eigenschaft) beziehen sich die HTTP-Methoden: POST und PUT ausschließlich auf Eigenschaften, die selbst keine eigenständigen Objekte sind. Für die Referenzierung zugehöriger Objekte, welche bereits in der Datenbank existieren müssen, zählt ausschließlich die Angabe der UUID. Sollte das Objekt in der Datenbank nicht existieren, wird eine Fehlermeldung zurückgeliefert. Zum Anlegen oder Ändern dieser Objekte muss zunächst eine separate Funktion aufgerufen werden.

8.1.3 Löschen von Inhalten mit Fremdschlüsselbeziehung

Das Löschen von Inhalten, welches sich auf die HTTP-Methode DELETE bezieht, wird nur eingeschränkt erlaubt, solange eine Fremdschlüsselbeziehung zu anderen Datenbankinhalten beseht oder Integritätsbedingungen 4.4 verletzt werden. Bspw. kann eine Werteliste nicht als ganzes gelöscht werden, solange Werte aus dieser Werteliste in Attributtypen verwendet werden. Sollten jedoch alle Werte aus einer Werteliste nicht verwendet werden, kann die Werteliste direkt gelöscht werden, wobei alle zugeordneten Werte mit gelöscht werden. Die folgende Auflistung soll die Möglichkeiten zum Löschen aufzeigen:

- **Attributtyp**, insofern er nicht von Attributwerten verwendet, nicht mit Attributtypen verknüpft oder in Attributtypgruppen enthalten ist
- **Attributtypgruppe** (Attributtypen werden nicht gelöscht), insofern sie nicht mit Themenausprägungen verknüpft ist
- **Multiplizität**, insofern sie nicht verwendet wird

- **Projekt** inkl. Themenausprägungen, Themeninstanzen und Attributwerte (Attributtypen und -gruppen werden nicht gelöscht)
- **Themenausprägung**, insofern sie nicht verwendet wird
- **Themeninstanz** inkl. Attributwerte (Attributtypen werden nicht gelöscht), insofern sie nicht mit einer anderen Themeninstanz verknüpft ist
- **Werteliste** inkl. Wertelistenwerte, insofern Wertelistenwerte nicht verwendet werden
- **Wertelistenwert**, insofern er nicht verwendet wird

8.1.4 Versionisierung

Für eine Gewährleistung der Verwendbarkeit der REST-Schnittstellen im Zuge von zukünftigen Änderungen sowie Erweiterungen ist eine Versionisierung notwendig. Damit ist gemeint, dass z.B. durch die Weiterentwicklung einzelne URIs wegfallen können oder Objekte, die eindeutig durch URIs referenziert werden, geändert werden. Dann würde der Fall eintreten, dass eine Anwendung, die diese REST-Schnittstellen konsumiert plötzlich nicht mehr funktioniert. Auf Dauer kann so kein zuverlässiger Betrieb gewährleistet werden und Anwendungen, die diese REST-Schnittstellen verwenden, müssten ständig gewartet und angepasst werden. Dies kann insbesondere zu Problemen führen wenn genau zu diesem Zeitpunkt keine Ressourcen für die Wartung zur Verfügung stehen.

Wie im Folgenden gezeigt, wird zur Versionisierung jede URI mit einer Versionsnummer versehen:

- /v1/system
- /v1/projects

Sollten Änderungen an der REST-Schnittstelle vorgenommen werden, dann bleiben die alten URIs bestehen und es können neue URIs erstellt oder bestehende für die neue Version angepasst bzw. geändert werden. Damit ist gewährleistet, dass Änderungen an den REST-Schnittstellen keine Seiteneffekte auf Anwendungen haben, die diese Schnittstellen konsumieren.

- /v2/system
- /v2/projects

Die folgenden URIs sind von dieser Versionisierung ausgenommen:

- /login

- /logout
- /application.{wadl|html}

Der Einfachheit halber wird im Folgenden auf die Auflistung bzw. Darstellung der Versionsnummern in URIs verzichtet.

8.2 System

Die in diesem Kapitel beschriebenen URIs beziehen sich auf Daten, die in der Systemdatenbank von OpenInfRA gespeichert sind. Ein zentraler Zugriffspunkt für die Systemdatenbank wird in der folgenden Tabelle 2 definiert.

HTTP	Beschreibung
/system	
GET	liefert einen View für die Systemdatenbank

Tabelle 2: URI-Mapping für die Systemdatenbank

8.2.1 Sprach-, Länder- und Zeichenkodierungen

Die Sprach-, Länder- und Zeichenkodierungen (vgl. Kapitel 2.7) werden in der folgenden Tabelle 3 definiert. Dabei ist darauf zu achten, dass ausschließlich HTTP-GET-Methoden zur Verfügung stehen. Funktionalitäten zum Hinzufügen neuer Sprach-, Länder- bzw. Zeichenkodierungen werden nicht bereitgestellt. Dies muss direkt über die Datenbank erfolgen (z.B. mittels eines SQL-Befehls).

HTTP	Beschreibung
/system/languagecodes	
GET	liefert Liste aller Sprachkodierungen aus der Systemdatenbank
/system/languagecodes/count	
GET	liefert die Anzahl aller Sprachkodierungen aus der Systemdatenbank
/system/languagecodes/{uuid}	
GET	liefert eine Sprachkodierung aus der Systemdatenbank
/system/countrycodes	
GET	liefert Liste aller Länderkodierungen aus der Systemdatenbank
/system/countrycodes/count	
GET	liefert die Anzahl aller Länderkodierungen aus der Systemdatenbank
/system/countrycodes/{uuid}	
GET	liefert eine Länderkodierung aus der Systemdatenbank
/system/charactercodes	
GET	liefert Liste aller Zeichenkodierungen aus der Systemdatenbank
/system/charactercodes/count	
GET	liefert die Anzahl aller Zeichenkodierungen aus der Systemdatenbank
/system/charactercodes/{uuid}	
GET	liefert eine Zeichenkodierung aus der Systemdatenbank

Tabelle 3: URI-Mapping für die Sprach-, Länder- und Zeichenkodierung

8.2.2 Sprachumgebung

In der folgenden Tabelle 4 wird das URI-Mapping für Sprachumgebungen der Systemdatenbank dargestellt. Eine Sprachumgebung wird hier mit der Bezeichnung *ptlocales* definiert. Änderung von Sprachumgebungen sind nicht möglich, da dies

direkte Auswirkungen auf beteiligte Projekte hätte, da diese die Sprachumgebung aus der Systemdatenbank kopieren (siehe 8.3.1). Dadurch soll verhindert werden, dass Sprachumgebungen mit derselben UUID, jedoch unterschiedlichen Inhalten, in verschiedenen Projekten existieren können.

HTTP	Beschreibung
/system/ptlocales	
GET	liefert Liste aller Sprachumgebungen aus der Systemdatenbank
POST	erstellt eine neue Sprachumgebung in der Systemdatenbank
/system/ptlocales/count	
GET	liefert die Anzahl aller Sprachumgebungen aus der Systemdatenbank
/system/ptlocales/new	
GET	liefert ein leeres Sprachumgebungs-Objekt
/system/ptlocales/{uuid}	
GET	liefert eine Sprachumgebung aus der Systemdatenbank
DELETE	löscht eine Sprachumgebung aus der Systemdatenbank

Tabelle 4: URI-Mapping für System-Sprachumgebungen

8.2.3 Themenausprägungen

In der folgenden Tabelle 5 wird das URI-Mapping für Themenausprägungen in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/topiccharacteristics	
GET	liefert Liste aller Themenausprägungen aus der Systemdatenbank
POST	erstellt eine neue Themenausprägung in der Systemdatenbank
/system/topiccharacteristics/count	
GET	liefert die Anzahl der Themenausprägungen aus der Systemdatenbank
/system/topiccharacteristics/{uuid}	
GET	liefert eine spezifische Themenausprägung aus der Systemdatenbank
PUT	ändert eine spezifische Themenausprägung in der Systemdatenbank
DELETE	löscht eine spezifische Themenausprägung in der Systemdatenbank

Tabelle 5: URI-Mapping für System-Themenausprägungen

8.2.4 Beziehungstypen

In der folgenden Tabelle 6 wird das URI-Mapping die Beziehungstypen in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/relationshiptypes	
GET	liefert Liste aller Beziehungstypen aus der Systemdatenbank
POST	erstellt einen neuen Beziehungstypen in der Systemdatenbank
/system/relationshiptypes/count	
GET	liefert die Anzahl aller Beziehungstypen aus der Systemdatenbank
/system/relationshiptypes/new	
GET	liefert einen leeren Beziehungstypen
/system/relationshiptypes/{uuid}	
GET	liefert einen Beziehungstypen aus der Systemdatenbank
PUT	ändert einen spezifischen Beziehungstypen in der Systemdatenbank
DELETE	löscht einen Beziehungstypen aus der Systemdatenbank

Tabelle 6: URI-Mapping für System-Sprachumgebungen

8.2.5 Beziehungstypen von Themenausprägungen

In der folgenden Tabelle 7 wird die Zuordnung von Beziehungstypen zu Themenausprägungen in in der Systemdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einer Themenausprägung, als auch von einem Beziehungstypen aus betrachtet werden.

HTTP	Beschreibung
/system/topiccharacteristics/{uuid}/relationshiptypes	
GET	liefert Liste von Beziehungen vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4), die zu einer spezifischen Themenausprägung aus der Systemdatenbank zugeordnet sind
POST	erstellt eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in der Systemdatenbank
/system/topiccharacteristics/{uuid}/relationshiptypes/count	
GET	liefert die Anzahl der Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einer spezifischen Themenausprägung in der Systemdatenbank zugeordnet sind
/system/topiccharacteristics/{uuid}/relationshiptypes/{uuid}	
GET	liefert eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in der Systemdatenbank
PUT	ändert eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in der Systemdatenbank
DELETE	löscht eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in der Systemdatenbank

Tabelle 7: URI-Mapping für System-Beziehungstypen von Themenausprägungen

In der folgenden Tabelle 8 wird die Zuordnung von Themenausprägungen zu Beziehungstypen in der Systemdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Themenausprägungen und Beziehungstypen sowohl von einer Themenausprägung, als auch von einem Beziehungstypen aus betrachtet werden.

HTTP	Beschreibung
/system/relationshiptypes/{uuid}/topiccharacteristics	
GET	liefert Liste aller Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einem spezifischen Beziehungstypen in der Systemdatenbank zugeordnet sind
POST	erstellt eine Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in der Systemdatenbank
/system/relationshiptypes/{uuid}/topiccharacteristics/count	
GET	liefert die Anzahl der Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einem spezifischen Beziehungstypen in der Systemdatenbank zugeordnet sind
/system/relationshiptypes/{uuid}/topiccharacteristics/{uuid}	
GET	liefert eine Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in der Systemdatenbank
PUT	ändert eine Beziehung Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in der Systemdatenbank
DELETE	löscht eine Beziehung Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in der Systemdatenbank

Tabelle 8: URI-Mapping für System-Themenausprägungen zu Beziehungstypen

8.2.6 Attributtypen

In der folgenden Tabelle 9 wird das URI-Mapping für die Attributtypen in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/attributetypes	
GET	liefert Liste aller Attributtypen aus der Systemdatenbank
POST	erstellt einen neuen Attributtypen in der Systemdatenbank
/system/attributetypes/count	
GET	liefert die Anzahl von Attributtypen aus der Systemdatenbank
/system/attributetypes/new	
GET	liefert ein leeres Attributtyp-Objekt
/system/attributetypes/{uuid}	
GET	liefert einen spezifischen Attributtypen aus der Systemdatenbank
PUT	ändert einen spezifischen Attributtypen in der Systemdatenbank
DELETE	löscht einen spezifischen Attributtypen in der Systemdatenbank

Tabelle 9: URI-Mapping für System-Attributtypen

8.2.7 Beziehungen von Attributtypen

In der folgenden Tabelle 10 werden die Beziehungen von Attributtypen zu Attributtypen in der Systemdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 17). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRADataObjects*.

HTTP	Beschreibung
/system/attributetypes/{uuid}/associations	
GET	liefert Liste aller Attributtypbeziehungen eines spezifischen Attributtypen aus der Systemdatenbank
POST	erstellt eine oder mehrere neue Beziehungen für einen spezifischen Attributtypen in der Systemdatenbank
/system/attributetypes/{uuid}/associations/count	
GET	liefert die Anzahl aller Attributtypbeziehungen eines spezifischen Attributtypen aus der Systemdatenbank
/system/attributetypes/{uuid}/associations/{uuid}	
GET	liefert eine spezifische Attributtypbeziehung aus der Systemdatenbank
PUT	ändert eine spezifische Attributtypbeziehung in der Systemdatenbank
DELETE	löscht eine spezifische Attributtypbeziehung in der Systemdatenbank

Tabelle 10: URI-Mapping für Beziehungen von System-Attributtypen

8.2.8 Attributtypgruppen

In der folgenden Tabelle 11 wird das URI-Mapping für die Attributtypgruppen in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/attributetypegroups	
GET	liefert Liste aller Attributtypgruppen aus der Systemdatenbank
POST	erstellt eine neue Attributtypgruppe in der Systemdatenbank
/system/attributetypegroups/count	
GET	liefert die Anzahl aller Attributtypgruppen aus der Systemdatenbank
/system/attributetypegroups/{uuid}	
GET	liefert eine spezifische Attributtypgruppe aus der Systemdatenbank
PUT	ändert eine spezifische Attributtypgruppe in der Systemdatenbank
DELETE	löscht eine spezifische Attributtypgruppe in der Systemdatenbank
/system/attributetypegroups/{uuid}/subgroups	
GET	liefert Liste aller Untergruppen für eine Hierarchiestufe aus der Systemdatenbank

Tabelle 11: URI-Mapping für System-Attributtypgruppen

8.2.9 Attributtypgruppen von Attributtypen

In der folgenden Tabelle 12 wird die Zuordnung von Attributtypgruppen zu Attributtypen in der Systemdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einem Attributtypen, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/system/attributetypes/{uuid}/attributetypegroups	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7), die zu einem spezifischen Attributtypen in der Systemdatenbank zugeordnet sind
POST	erstellt eine oder mehrere Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) zu einem spezifischen Attributtypen in der Systemdatenbank
/system/attributetypes/{uuid}/attributetypegroups/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7), die zu einem spezifischen Attributtypen in der Systemdatenbank zugeordnet sind
/system/attributetypes/{uuid}/attributetypegroups/{uuid}	
GET	liefert eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe aus der Systemdatenbank
PUT	ändert eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe in der Systemdatenbank
DELETE	löscht eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe in der Systemdatenbank

Tabelle 12: URI-Mapping für System-Attributtypgruppen von System-Attributtypen

In der folgenden Tabelle 13 wird die Zuordnung von Attributtypen zu Attributtypgruppen in der Systemdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Attributtypen und Attributtypgruppen sowohl von einem Attributtypen, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/system/attributetypegroups/{uuid}/attributetypes	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7), die zu einer spezifischen Attributtypgruppen in der Systemdatenbank zugeordnet sind
POST	erstellt eine oder mehrere Beziehungen vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) zu einer spezifischen Attributtypgruppe in der Systemdatenbank
/system/attributetypegroups/{uuid}/attributetypes/count	
GET	liefert die Anzahl von Attributtypen, die zu einer spezifischen Attributtypgruppe in der Systemdatenbank zugeordnet sind
/system/attributetypegroups/{uuid}/attributetypes/{uuid}	
GET	liefert eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen aus der Systemdatenbank
PUT	ändert eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen in der Systemdatenbank
DELETE	löscht eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen in der Systemdatenbank

Tabelle 13: URI-Mapping für System-Attributtypen von System-Attributtypgruppen

8.2.10 Attributtypgruppen zu Themenausprägungen

In der folgenden Tabelle 14 wird die Zuordnung von Attributtypgruppen zu Themenausprägungen in der Systemdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einer Themenausprägung, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/system/topiccharacteristics/{uuid}/attributetypgroups	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Themenausprägung in der Systemdatenbank zugeordnet sind
POST	erstellt eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in der Systemdatenbank
/system/topiccharacteristics/{uuid}/attributetypgroups/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Themenausprägung in der Systemdatenbank zugeordnet sind
/system/topiccharacteristics/{uuid}/attributetypgroups/{uuid}	
GET	liefert eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in der Systemdatenbank zugeordnet ist
PUT	ändert eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in der Systemdatenbank
DELETE	löscht eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in der Systemdatenbank

Tabelle 14: URI-Mapping für System-Attributtypgruppen zu System-Themenausprägungen

In der folgenden Tabelle 15 wird die Zuordnung von Themenausprägungen zu Attributtypgruppen in der Systemdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Themenausprägungen und Attributtypgruppen sowohl von einer Themenausprägung, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/system/attributetypegroups/{uuid}/topiccharacteristics	
GET	liefert Liste aller Beziehungen vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8), die zu einer spezifischen Attributtypgruppe in der Systemdatenbank zugeordnet sind
POST	erstellt eine Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in der Systemdatenbank
/system/attributetypegroups/{uuid}/topiccharacteristics/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Attributtypgruppe in der Systemdatenbank zugeordnet sind
/system/attributetypegroups/{uuid}/topiccharacteristics/{uuid}	
GET	liefert eine Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in der Systemdatenbank zugeordnet ist
PUT	ändert eine Beziehung Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in der Systemdatenbank
DELETE	löscht eine Beziehung Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in der Systemdatenbank

Tabelle 15: URI-Mapping für System-Themenausprägungen zu System-Attributtypgruppen

8.2.11 Multiplizität

In der folgenden Tabelle 16 wird das URI-Mapping für Multiplizitäten in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/multiplicities	
GET	liefert Liste aller Multiplizitäten aus der Systemdatenbank
POST	erstellt eine neue Multiplizität in der Systemdatenbank
/system/multiplicities/count	
GET	liefert die Anzahl aller Multiplizitäten aus der Systemdatenbank
/system/multiplicities/new	
GET	liefert ein leeres Multiplizität-Objekt
/system/multiplicities/{uuid}	
GET	liefert eine spezifische Multiplizität aus der Systemdatenbank
PUT	ändert eine spezifische Multiplizität in der Systemdatenbank
DELETE	löscht eine spezifische Multiplizität in der Systemdatenbank

Tabelle 16: URI-Mapping für System-Multiplizität

8.2.12 Wertelisten

In der folgenden Tabelle 17 wird das URI-Mapping für die Wertelisten in der Systemdatenbank dargestellt.

8.2.13 Wertelistenwerte

In der folgenden Tabelle 18 wird das URI-Mapping für Wertelistenwerte in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/valuelists	
GET	liefert eine Liste aller Wertelisten aus der Systemdatenbank
POST	erstellt eine neue Werteliste in der Systemdatenbank
/system/valuelists/count	
GET	liefert die Anzahl aller Wertelisten aus der Systemdatenbank
/system/valuelists/new	
GET	liefert ein leeres Wertelisten-Objekt
/system/valuelists/{uuid}	
GET	liefert eine spezifische Werteliste aus der Systemdatenbank
PUT	ändert eine spezifische Werteliste in der Systemdatenbank
DELETE	löscht eine spezifische Werteliste in der Systemdatenbank

Tabelle 17: URI-Mapping für System-Wertelisten

8.2.14 Beziehungen von Wertelisten

In der folgenden Tabelle 19 werden die Beziehungen von Wertelisten zu Wertelisten in der Systemdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 23). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRADataObjects*.

HTTP	Beschreibung
/system/valuelists/{uuid}/valuelistvalues	
GET	liefert eine Liste aller Wertelistenwerte einer spezifischen Werteliste aus der Systemdatenbank
POST	erstellt einen oder mehrere neue Wertelistenwerte für eine spezifische Werteliste in der Systemdatenbank
/system/valuelists/{uuid}/valuelistvalues/count	
GET	liefert die Anzahl aller Wertelistenwerte einer spezifischen Werteliste aus der Systemdatenbank
/system/valueslistvalues/new	
GET	liefert ein leeres Wertelistenwert-Objekt
/system/valuelistvalues/{uuid}	
GET	liefert einen spezifische Wertelistenwert aus der Systemdatenbank
PUT	ändert einen spezifische Wertelistenwert in der Systemdatenbank
DELETE	löscht einen spezifische Wertelistenwert in der Systemdatenbank

Tabelle 18: URI-Mapping für System-Wertelistenwerte

8.2.15 Beziehungen von Wertelistenwerte

In der folgenden Tabelle 20 werden die Beziehungen von Wertelistenwerten zu Wertelistenwerten in der Systemdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 23). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRADataObjects*.

8.2.16 System-Metadaten

In der folgenden Tabelle 21 wird das URI-Mapping für die Metadaten in der Systemdatenbank dargestellt.

HTTP	Beschreibung
/system/valuelists/{uuid}/associations	
GET	liefert Liste aller Wertelistenbeziehungen einer spezifischen Werteliste aus der Systemdatenbank
POST	erstellt eine oder mehrere neue Wertelistenbeziehungen in der Systemdatenbank
/system/valuelists/{uuid}/associations/count	
GET	liefert die Anzahl aller Wertelistenbeziehungen einer spezifischen Werteliste aus der Systemdatenbank
/system/valuelists/{uuid}/associations/{uuid}	
GET	liefert eine Wertelistenbeziehung für eine spezifische Werteliste aus der Systemdatenbank
PUT	ändert eine Wertelistenbeziehung für eine spezifische Werteliste in der Systemdatenbank
DELETE	löscht eine Wertelistenbeziehung für eine spezifische Werteliste in der Systemdatenbank

Tabelle 19: URI-Mapping für Beziehungen von System-Wertelisten

8.3 Projekte

Dieses Kapitel beschreibt das URI-Mapping für Daten, welche sich in den Projektdatenbanken befinden. In der folgenden Tabelle 22 werden die HTTP-Methoden beschrieben, welche die Funktionen für Projekte bereitstellen.

HTTP	Beschreibung
/projects	
GET	liefert Liste aller Projekte (ohne Unterprojekte) – der Zugriff erfolgt über die Metadatenbank
POST	erstellt ein neues Projekt
/projects/count	
GET	liefert die Anzahl aller Haupt-Projekte (welche ein eigenständiges Schema besitzen)

/projects/{id}	
GET	liefert ein spezifisches Projekt
PUT	ändert ein spezifisches Projekt
DELETE	löscht ein spezifisches Projekt und dessen Unterprojekte
/projects/{id}/new	
GET	liefert ein leeres Projekt-Objekt welches als Unterprojekt für ein spezifisches Projekt dient
/projects/{id}/subprojects	
GET	liefert Liste aller Unterprojekte für eine Hierarchiestufe
/projects/{id}/subprojects/count	
GET	liefert die Anzahl der Unterprojekte des entsprechenden Projektes
/projects/{id}/parents	
GET	liefert das Elternprojekt eines spezifischen Projektes
/projects/{id}/metadata/{uuid}	
GET	liefert eine spezifische Metainformation für ein spezifisches Projekt
/projects/3d	
GET	liefert einen View für den 3D WebGIS-Client
/projects/maps	
GET	liefert einen View für den 2D WebGIS-Client

Tabelle 22: URI-Mapping für Projekte

HTTP	Beschreibung
/system/valuelistvalues/{uuid}/associations	
GET	liefert Liste aller Wertelistenwertebeziehungen eines spezifischen Wertelistenwertes aus der Systemdatenbank
POST	erstellt eine oder mehrere neue Wertelistenwertebeziehungen in der Systemdatenbank
/system/valuelistvalues/{uuid}/associations/count	
GET	liefert die Anzahl aller Wertelistenwertebeziehungen eines spezifischen Wertelistenwertes aus der Systemdatenbank
/system/valuelistvalues/{uuid}/associations/{uuid}	
GET	liefert eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert aus der Systemdatenbank
PUT	ändert eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert in der Systemdatenbank
DELETE	löscht eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert in der Systemdatenbank

Tabelle 20: URI-Mapping für Beziehungen von System-Wertelistenwerte

8.3.1 Sprachumgebung

In der folgenden Tabelle 23 wird das URI-Mapping für Sprachumgebungen in den Projektdatenbanken dargestellt. Wie bereits beschrieben, wird die Sprachumgebung hier mit *ptlocales* definiert (vgl. 8.2.2). Die Sprach-, Länder- und Zeichenkodierungen werden direkt aus der Systemdatenbank in eine spezifische Projektdatenbank übernommen. Da die Sprachumgebung die konkrete Sprache eines Datensatzes in der Fachdatenbank beschreibt, wird eine Änderung der Sprachumgebung nicht mit angeboten. Eine Sprachumgebung kann somit nur in der Systemdatenbank erstellt und aus dieser in eine Projektdatenbank kopiert werden. Weiterhin ist das Anzeigen von Sprachumgebungen aus einer Projektdatenbank möglich. Beim Löschen einer Sprachumgebung wird diese nur aus der Projektdatenbank entfernt bleibt aber in der Systemdatenbank erhalten.

HTTP	Beschreibung
/system/metadata	
GET	liefert Liste aller Metadaten aus der Systemdatenbank
POST	erstellt eine neue Metainformation in der Systemdatenbank
/system/metadata/count	
GET	liefert die Anzahl aller Metadaten aus der Systemdatenbank
/system/metadata/new	
GET	liefert ein leeres Metadaten-Objekt
/system/metadata/{uuid}	
GET	liefert eine spezifische Metainformation aus der Systemdatenbank
PUT	ändert eine spezifische Metainformation in der Systemdatenbank
DELETE	löscht eine spezifische Metainformation in der Systemdatenbank

Tabelle 21: URI-Mapping Projekt-Metadaten

8.3.2 Themenausprägungen

In der folgenden Tabelle 24 wird das URI-Mapping für Themenausprägungen in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/ptlocales	
GET	liefert Liste aller Sprachumgebungen aus einer spezifischen Projektdatenbank
POST	kopiert eine bestehende Sprachumgebung aus der Systemdatenbank in eine spezifische Projektdatenbank
/projects/{id}/ptlocales/count	
GET	liefert die Anzahl aller Sprachumgebungen aus einer spezifischen Projektdatenbank
/projects/{id}/ptlocales/{uuid}	
GET	liefert eine spezifische Sprachumgebung aus einer spezifischen Projektdatenbank
DELETE	löscht eine spezifische Sprachumgebung aus einer spezifischen Projektdatenbank

Tabelle 23: URI-Mapping für Projekt-Sprachumgebungen

8.3.3 Beziehungstypen

In der folgenden Tabelle 25 wird das URI-Mapping die Beziehungstypen in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/topiccharacteristics	
GET	liefert Liste aller Themenausprägungen für ein spezifisches Projekt
POST	erstellt eine neue Themenausprägung für ein spezifisches Projekt
/projects/{id}/topiccharacteristics/count	
GET	liefert die Anzahl der Themenausprägungen für ein spezifisches Projekt
/projects/{id}/topiccharacteristics/{uuid}	
GET	liefert eine spezifische Themenausprägung für ein spezifisches Projekt
PUT	ändert eine spezifische Themenausprägung für ein spezifisches Projekt
DELETE	löscht eine spezifische Themenausprägung für ein spezifisches Projekt

Tabelle 24: URI-Mapping für Projekt-Themenausprägungen

8.3.4 Beziehungstypen von Themenausprägungen

In der folgenden Tabelle 26 wird die Zuordnung von Beziehungstypen zu Themenausprägungen in einer spezifischen Projektdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einer Themenausprägung, als auch von einem Beziehungstypen aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/relationshiptypes	
GET	liefert Liste aller Beziehungstypen für ein spezifisches Projekt
POST	erstellt einen neuen Beziehungstypen für ein spezifisches Projekt
/projects/{uuid}/relationshiptypes/count	
GET	liefert die Anzahl aller Beziehungstypen für ein spezifisches Projekt
/projects/{id}/relationshiptypes/{uuid}	
GET	liefert einen Beziehungstypen für ein spezifisches Projekt
PUT	ändert einen spezifischen Beziehungstypen für ein spezifisches Projekt
DELETE	löscht einen Beziehungstypen für ein spezifisches Projekt

Tabelle 25: URI-Mapping für System-Sprachumgebungen

In der folgenden Tabelle 27 wird die Zuordnung von Themenausprägungen zu Beziehungstypen in einer spezifischen Projektdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Themenausprägungen und Beziehungstypen sowohl von einer Themenausprägung, als auch von einem Beziehungstypen aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/topiccharacteristics/{uuid}/relationshiptypes	
GET	liefert Liste von Beziehungen vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4), die zu einer spezifischen Themenausprägung und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen und einem spezifischen Projekt zugeordnet sind
/projects/{id}/topiccharacteristics/{uuid}/relationshiptypes/count	
GET	liefert die Anzahl der Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einer spezifischen Themenausprägung und einem spezifischen Projekt zugeordnet sind
/projects/{id}/topiccharacteristics/{uuid}/relationshiptypes/{uuid}	
GET	liefert eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in einem spezifischen Projekt
PUT	ändert eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in einem spezifischen Projekt
DELETE	löscht eine Beziehung vom Typ *RelationshipTypeToTopicCharacteristic* (vgl. Kapitel 7.1.4) für eine spezifische Themenausprägung zu einem spezifischen Beziehungstypen in einem spezifischen Projekt

Tabelle 26: URI-Mapping für Projekt-Beziehungstypen von Themenausprägungen

8.3.5 Attributtypen

In der folgenden Tabelle 28 wird das URI-Mapping für die Attributtypen in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/relationshiptypes/{uuid}/topiccharacteristics	
GET	liefert Liste aller Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einem spezifischen Beziehungstypen und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in einem spezifischen Projekt
/projects/{id}/relationshiptypes/{uuid}/topiccharacteristics/count	
GET	liefert die Anzahl der Beziehungen vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4), die zu einem spezifischen Beziehungstypen und einem spezifischen Projekt zugeordnet sind
/projects/{id}/relationshiptypes/{uuid}/topiccharacteristics/{uuid}	
GET	liefert eine Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in einem spezifischen Projekt
PUT	ändert eine Beziehung Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in einem spezifischen Projekt
DELETE	löscht eine Beziehung Beziehung vom Typ *TopicCharacteristicToRelationshipType* (vgl. Kapitel 7.1.4) für einen spezifische Beziehungstypen zu einer spezifischen Themenausprägung in einem spezifischen Projekt

Tabelle 27: URI-Mapping für Themenausprägungen zu Beziehungstypen

8.3.6 Beziehungen von Attributtypen

In der folgenden Tabelle 29 werden die Beziehungen von Attributtypen zu Attributtypen in einer spezifischen Projektdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 17). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRADataObjects*.

HTTP	Beschreibung
/projects/{id}/attributetypes	
GET	liefert Liste aller Attributtypen für ein spezifisches Projekt
POST	erstellt einen neuen Attributtypen für ein spezifisches Projekt
/projects/{id}/attributetypes/count	
GET	liefert die Anzahl von Attributtypen für ein spezifisches Projekt
/projects/{id}/attributetypes/new	
GET	liefert ein leeres Attributtyp-Objekt
/projects/{id}/attributetypes/{uuid}	
GET	liefert einen spezifischen Attributtypen für ein spezifisches Projekt
PUT	ändert einen spezifischen Attributtypen für ein spezifisches Projekt
DELETE	löscht einen spezifischen Attributtypen für ein spezifisches Projekt

Tabelle 28: URI-Mapping für Projekt-Attributtypen

8.3.7 Attributtypgruppen

In der folgenden Tabelle 30 wird das URI-Mapping für die Attributtypgruppen in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/attributetypes/{uuid}/associations	
GET	liefert Liste aller Attributtypbeziehungen eines spezifischen Attributtypen und eines spezifischen Projekts
POST	erstellt eine oder mehrere neue Beziehungen für einen spezifischen Attributtypen in einem spezifischen Projekt
/projects/{id}/attributetypes/{uuid}/associations/count	
GET	liefert die Anzahl aller Attributtypbeziehungen eines spezifischen Attributtypen und eines spezifischen Projekts
/projects/{id}/attributetypes/{uuid}/associations/new	
GET	liefert ein leeres Attributtypbeziehung-Objekt
/projects/{id}/attributetypes/{uuid}/associations/{uuid}	
GET	liefert eine spezifische Attributtypbeziehung für ein spezifisches Projekt
PUT	ändert eine spezifische Attributtypbeziehung für ein spezifisches Projekt
DELETE	löscht eine spezifische Attributtypbeziehung für ein spezifisches Projekt

Tabelle 29: URI-Mapping für Beziehungen von Attributtypen

8.3.8 Attributtypgruppen von Attributtypen

In der folgenden Tabelle 31 wird die Zuordnung von Attributtypgruppen zu Attributtypen in einer spezifischen Projektdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einem Attributtypen, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/attributetypegroups	
GET	liefert Liste aller Attributtypgruppen für ein spezifisches Projekt
POST	erstellt eine neue Attributtypgruppe für ein spezifisches Projekt
/projects/{id}/attributetypegroups/count	
GET	liefert die Anzahl aller Attributtypgruppen für ein spezifisches Projekt
/projects/{id}/attributetypegroups/new	
GET	liefert ein leeres Attributtypgruppen-Objekt
/projects/{id}/attributetypegroups/{uuid}	
GET	liefert eine spezifische Attributtypgruppe für ein spezifisches Projekt
PUT	ändert eine spezifische Attributtypgruppe für ein spezifisches Projekt
DELETE	löscht eine spezifische Attributtypgruppe für ein spezifisches Projekt
/projects/{id}/attributetypegroups/{uuid}/subgroups	
GET	liefert Liste aller Untergruppen für eine Hierarchiestufe für ein spezifisches Projekt

Tabelle 30: URI-Mapping für Projekt-Attributtypgruppen

In der folgenden Tabelle 32 wird die Zuordnung von Attributtypen zu Attributtypgruppen in einer spezifischen Projektdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Attributtypen und Attributtypgruppen sowohl von einem Attributtypen, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/attributetypes/{uuid}/attributetypegroups	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7), die zu einem spezifischen Attributtypen und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine oder mehrere Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) zu einem spezifischen Attributtypen für ein spezifisches Projekt
/projects/{id}/attributetypes/{uuid}/attributetypegroups/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7), die zu einem spezifischen Attributtypen und einem spezifischen Projekt zugeordnet sind
/projects/{id}/attributetypes/{uuid}/attributetypegroups/{uuid}	
GET	liefert eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe für ein spezifisches Projekt
PUT	ändert eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe für ein spezifisches Projekt
DELETE	löscht eine spezifische Beziehung vom Typ *AttributeTypeGroupToAttributeType* (vgl. Kapitel 7.1.7) eines spezifischen Attributtypen zu einer spezifischen Attributtypgruppe für ein spezifisches Projekt

Tabelle 31: URI-Mapping für Projekt-Attributtypgruppen von Projekt-Attributtypen

8.3.9 Attributtypgruppen zu Themenausprägungen

In der folgenden Tabelle 33 wird die Zuordnung von Attributtypgruppen zu Themenausprägungen in einer spezifischen Projektdatenbank dargestellt. Dabei kann die Zuordnung sowohl von einer Themenausprägung, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/attributetypegroups/{uuid}/attributetypes	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7), die zu einer spezifischen Attributtypgruppen und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine oder mehrere Beziehungen vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) zu einer spezifischen Attributtypgruppe für ein spezifisches Projekt
/projects/{id}/attributetypegroups/{uuid}/attributetypes/count	
GET	liefert die Anzahl von Attributtypen, die zu einer spezifischen Attributtypgruppe und einem spezifischen Projekt zugeordnet sind
/projects/{id}/attributetypegroups/{uuid}/attributetypes/{uuid}	
GET	liefert eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen für ein spezifisches Projekt
PUT	ändert eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen für ein spezifisches Projekt
DELETE	löscht eine spezifische Beziehung vom Typ *AttributeTypeToAttributeTypeGroup* (vgl. Kapitel 7.1.7) einer spezifischen Attributtypgruppe zu einem spezifischen Attributtypen für ein spezifisches Projekt

Tabelle 32: URI-Mapping für Projekt-Attributtypen von Projekt-Attributtypgruppen

In der folgenden Tabelle 34 wird die Zuordnung von Themenausprägungen zu Attributtypgruppen in einer spezifischen Projektdatenbank dargestellt. Wie bereits geschildert, kann die Zuordnung zwischen Themenausprägungen und Attributtypgruppen sowohl von einer Themenausprägung, als auch von einer Attributtypgruppe aus betrachtet werden.

HTTP	Beschreibung
/projects/{id}/topiccharacteristics/{uuid}/attributetypegroups	
GET	liefert Liste von Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Themenausprägung und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in einem spezifischen Projekt
/projects/{id}/topiccharacteristics/{uuid}/attributetypgroups/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Attributtypgruppe in der einem spezifischen Projekt zugeordnet sind
/projects/{id}/topiccharacteristics/{uuid}/attributetypegroups/{uuid}	
GET	liefert eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in einem spezifischen Projekt
PUT	ändert eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in einem spezifischen Projekt
DELETE	löscht eine Beziehung vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8) für eine spezifische Themenausprägung zu einer spezifischen Attributtypgruppe in einem spezifischen Projekt

Tabelle 33: URI-Mapping für Attributtypgruppen zu Themenausprägungen

8.3.10 Multiplizität

In der folgenden Tabelle 35 wird das URI-Mapping für Multiplizitäten in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/attributetypegroups/{uuid}/topiccharacteristics	
GET	liefert Liste aller Beziehungen vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8), die zu einer spezifischen Attributtypgruppe und einem spezifischen Projekt zugeordnet sind
POST	erstellt eine Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in einem spezifischen Projekt
/projects/{id}/attributetypegroups/{uuid}/topiccharacteristics/count	
GET	liefert die Anzahl aller Beziehungen vom Typ *AttributeTypeGroupToTopicCharacteristic* (vgl. Kapitel 7.1.8), die zu einer spezifischen Attributtypgruppe in einem spezifischen Projekt zugeordnet sind
/projects/{id}/attributetypegroups/{uuid}/topiccharacteristics/{uuid}	
GET	liefert eine Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in einem spezifischen Projekt
PUT	ändert eine Beziehung Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in einem spezifischen Projekt
DELETE	löscht eine Beziehung Beziehung vom Typ *TopicCharacteristicToAttributeTypeGroup* (vgl. Kapitel 7.1.8) für eine spezifische Attributtypgruppe zu einer spezifischen Themenausprägung in einem spezifischen Projekt

Tabelle 34: URI-Mapping für Themenausprägungen zu Attributtypgruppen

8.3.11 Themeninstanz

In der folgenden Tabelle 36 wird das URI-Mapping für Themeninstanzen in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/multiplicities	
GET	liefert Liste aller Multiplizitäten zu einem spezifischen Projekt
POST	erstellt eine neue Multiplizität für ein spezifisches Projekt
/projects/{id}/multiplicities/count	
GET	liefert die Anzahl aller Multiplizitäten für ein spezifisches Projekt
/projects/{id}/multiplicities/new	
GET	liefert ein leeres Multiplizität-Objekt
/projects/{id}/multiplicities/{uuid}	
GET	liefert eine spezifische Multiplizität für ein spezifisches Projekt
PUT	ändert eine spezifische Multiplizität für ein spezifisches Projekt
DELETE	löscht eine spezifische Multiplizität für ein spezifisches Projekt

Tabelle 35: URI-Mapping für Multiplizität

HTTP	Beschreibung
/projects/{id}/topicinstances/{uuid}	
GET	liefert eine spezifische Themeninstanz für ein spezifisches Projekt
PUT	ändert eine spezifische Themeninstanz für ein spezifisches Projekt
DELETE	löscht eine spezifische Themeninstanz für ein spezifisches Projekt
/projects/{id}/topicinstances/{uuid}/parents	
GET	liefert eine Liste von Elterninstanzen bis zur Wurzelinstanz, ausgehend von einer spezifischen Themeninstanz
/projects/{id}/topicinstances/{uuid}/topic	
GET	liefert eine spezifische Themeninstanz für ein spezifisches Projekt
/projects/{id}/topicinstances/{uuid}/topic.csv	
GET	liefert eine spezifische Themeninstanz für ein spezifisches Projekt als CSV-Datei
/projects/{id}/topicinstances/{uuid}/topic.pdf	

GET	liefert eine spezifische Themeninstanz für ein spezifisches Projekt als PDF-Datei
/projects/{id}/topiccharacteristics/{uuid}/topicinstances	
GET	liefert Liste aller Themeninstanzen für eine spezifische Themenausprägung und ein spezifisches Projekt
/projects/{id}/topiccharacteristics/{uuid}/topicinstances/count	
GET	liefert die Anzahl der Themeninstanzen für eine spezifische Themenausprägung und ein spezifisches Projekt
/projects/{id}/topiccharacteristics/{uuid}/topicinstances/geomz	
GET	liefert Liste von Themeninstanzen, die 3D Geometrien besitzen, und deren zugehörige 3D Geometrien für eine spezifische Themenausprägung und ein spezifisches Projekt
/projects/{id}/topiccharacteristics/{uuid}/topicinstances/geomz/count	
GET	liefert die Anzahl der Themeninstanzen, die 3D Geometrien besitzen, für eine spezifische Themenausprägung und ein spezifisches Projekt

Tabelle 36: URI-Mapping für Themeninstanzen

8.3.12 Beziehungen von Themeninstanzen

In der folgenden Tabelle 37 werden die Beziehungen von Themeninstanzen zu Themeninstanzen in einer spezifischen Projektdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 21). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRAData-Objects*.

HTTP	Beschreibung
/projects/{id}/topicinstances/{id}/associations	
GET	liefert Liste aller Themeninstanzbeziehungen einer spezifischen Themeninstanz und eines spezifischen Projekts
POST	erstellt eine oder mehrere neue Themeninstanzbeziehungen in einem spezifischen Projekt
/projects/{id}/topicinstances/{id}/associations/count	
GET	liefert die Anzahl aller Themeninstanzbeziehungen einer spezifischen Themeninstanz und eines spezifischen Projekts
/projects/{id}/topicinstances/{uuid}/associations/{uuid}	
GET	liefert eine spezifische Themeninstanzbeziehung für ein spezifisches Projekt
PUT	ändert eine spezifische Themeninstanzbeziehung für ein spezifisches Projekt
DELETE	löscht eine spezifische Themeninstanzbeziehung für ein spezifisches Projekt

Tabelle 37: URI-Mapping für Beziehungen von Themeninstanzen

8.3.13 Attributwerte

In der folgenden Tabelle 38 wird das URI-Mapping für Attributwerte in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/topicinstances/{uuid}/attributetypes/{uuid}/attributevalues	
GET	liefert eine Liste von Attributwerten mit einem spezifischen Attributtypen zu einer spezifischen Themeninstanz in einem spezifischen Projekt
/projects/{id}/topicinstances/{uuid}/attributetypes/{uuid}/attributevalues/new	
GET	liefert ein leeres Attributwert-Objekt dessen Inhalt von der übergebenen Themeninstanz Id und Attributtyp Id abhängig ist
/projects/{id}/attributevalues	
POST	erstellt einen neue Attributwert mit einem spezifischen Attributtypen zu einer spezifischen Themeninstanz in einem spezifischen Projekt
/projects/{id}/attributevalues/{uuid}	
GET	liefert einen spezifischen Attributwert für ein spezifisches Projekt
PUT	ändert einen spezifischen Attributwert für ein spezifisches Projekt
DELETE	löscht einen spezifische Attributwert für ein spezifisches Projekt
/projects/{id}/attributevalues/geomtypes	
GET	liefert eine Liste der zur Verfügung stehenden Geometrietypen ein spezifisches Projekt

Tabelle 38: URI-Mapping für Attributwerte

8.3.14 Wertelisten

In der folgenden Tabelle 39 wird das URI-Mapping für die Wertelisten in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/valuelists	
GET	liefert eine Liste aller Wertelisten eines spezifisches Projekts
POST	erstellt eine neue Werteliste für ein spezifisches Projekt
/projects/{id}/valuelists/count	
GET	liefert die Anzahl aller Wertelisten eines spezifisches Projekts
/projects/{id}/valuelists/new	
GET	liefert ein leeres Wertelisten-Objekt
/projects/{id}/valuelists/{uuid}	
GET	liefert eine spezifische Werteliste für ein spezifisches Projekt
PUT	ändert eine spezifische Werteliste für ein spezifisches Projekt
DELETE	löscht eine spezifische Werteliste für ein spezifisches Projekt

Tabelle 39: URI-Mapping für Wertelisten

8.3.15 Wertelistenwerte

In der folgenden Tabelle 40 wird das URI-Mapping für Wertelistenwerte in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/valuelists/{uuid}/valuelistvalues	
GET	liefert eine Liste aller Wertelistenwerte einer spezifischen Werteliste für ein spezifisches Projekt
POST	erstellt einen oder mehrere neue Wertelistenwerte für eine spezifische Werteliste in einem spezifischen Projekt
/projects/{id}/valuelists/{uuid}/valuelistvalues/count	
GET	liefert die Anzahl aller Wertelistenwerte einer spezifischen Werteliste für ein spezifisches Projekt
/projects/{id}/valueslistvalues/new	
GET	liefert ein leeres Wertelistenwert-Objekt
/projects/{id}/valuelistvalues/{uuid}	
GET	liefert einen spezifische Wertelistenwert für ein spezifisches Projekt
PUT	ändert einen spezifische Wertelistenwert für ein spezifisches Projekt
DELETE	löscht einen spezifische Wertelistenwert für ein spezifisches Projekt

Tabelle 40: URI-Mapping für Wertelistenwerte

8.3.16 Beziehungen von Wertelisten

In der folgenden Tabelle 41 werden die Beziehungen von Wertelisten zu Wertelisten in einer spezifischen Projektdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 23). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRADataObjects*.

HTTP	Beschreibung
/projects/{id}/valuelists/{uuid}/associations	
GET	liefert Liste aller Wertelistenbeziehungen einer spezifischen Werteliste und eines spezifischen Projekts
POST	erstellt eine oder mehrere neue Wertelistenbeziehungen in einem spezifischen Projekt
/projects/{id}/valuelists/{uuid}/associations/count	
GET	liefert die Anzahl aller Wertelistenbeziehungen einer spezifischen Werteliste und eines spezifischen Projekts
/projects/{id}/valuelists/{uuid}/associations/{uuid}	
GET	liefert eine Wertelistenbeziehung für eine spezifische Werteliste in einem spezifischen Projekt
PUT	ändert eine Wertelistenbeziehung für eine spezifische Werteliste in einem spezifischen Projekt
DELETE	löscht eine Wertelistenbeziehung für eine spezifische Werteliste in einem spezifischen Projekt

Tabelle 41: URI-Mapping für Beziehungen von Wertelisten

8.3.17 Beziehungen von Wertelistenwerte

In der folgenden Tabelle 42 werden die Beziehungen von Wertelistenwerten zu Wertelistenwerten in einer spezifischen Projektdatenbank dargestellt. Dabei bezieht sich jeweils die erste UUID auf die *associatedId* (vgl. Abbildung 23). Im zweiten URI-Mapping bezieht sich die zweite UUID auf den Identifikator des *OpenInfRAData-Objects*.

HTTP	Beschreibung
/projects/{id}/valuelistvalues/{uuid}/associations	
GET	liefert Liste aller Wertelistenwertebeziehungen eines spezifischen Wertelistenwertes und eines spezifischen Projekts
POST	erstellt eine oder mehrere neue Wertelistenwertebeziehung in einem spezifischen Projekt
/projects/{id}/valuelistvalues/{uuid}/associations/count	
GET	liefert die Anzahl aller Wertelistenwertebeziehungen eines spezifischen Wertelistenwertes und eines spezifischen Projekts
/projects/{id}/valuelistvalues/{uuid}/associations/{uuid}	
GET	liefert eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert in einem spezifischen Projekt
PUT	ändert eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert in einem spezifischen Projekt
DELETE	löscht eine Wertelistenwertebeziehung für einen spezifischen Wertelistenwert in einem spezifischen Projekt

Tabelle 42: URI-Mapping für Beziehungen von Wertelistenwerte

8.3.18 Projekt-Metadaten

In der folgenden Tabelle 43 wird das URI-Mapping für die Metadaten in einer spezifischen Projektdatenbank dargestellt.

HTTP	Beschreibung
/projects/{id}/metadata	
GET	liefert Liste aller Metadaten eines spezifischen Projekts
POST	erstellt eine neue Metainformation in einem spezifischen Projekt
/project/{id}/metadata/count	
GET	liefert die Anzahl aller Metadaten aus einem spezifischen Projekt
/projects/{id}/metadata/new	
GET	liefert ein leeres Metadaten-Objekt
/projects/{id}/metadata/{uuid}	
GET	liefert eine spezifische Metainformation für ein spezifisches Projekt
PUT	ändert eine spezifische Metainformation für ein spezifisches Projekt
DELETE	löscht eine spezifische Metainformation für ein spezifisches Projekt

Tabelle 43: URI-Mapping für Projekt-Metadaten

8.4 Metadaten

Dieses Kapitel beschreibt das URI-Mapping für Daten, welche sich in der Metadatenbank befinden.

8.4.1 Konfiguration

In der folgenden Tabelle 44 wird das URI-Mapping für die Konfiguration der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/settings	
GET	liefert Liste aller Konfigurationen der Anwendung
POST	erstellt eine neue Konfiguration für die Anwendung
/metadata/settings/count	
GET	liefert die Anzahl der Konfigurationen der Anwendung
/metadata/settings/new	
GET	liefert ein leeres Konfigurationen-Objekt
/metadata/settings/{uuid}	
GET	liefert eine spezifische Konfiguration der Anwendung
PUT	ändert eine spezifische Konfiguration der Anwendung
DELETE	löscht eine spezifische Konfiguration der Anwendung

Tabelle 44: URI-Mapping für Konfiguration der Anwendung

8.4.2 Protokolleinträge

In der folgenden Tabelle 45 wird das URI-Mapping für die Protokollierungseinträge der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/logs	
GET	liefert Liste aller Protokolleinträge der Anwendung
POST	erstellt einen neuen Protokolleintrag für die Anwendung
/metadata/logs/count	
GET	liefert die Anzahl der Protokolleinträge der Anwendung
/metadata/logs/new	
GET	liefert ein leeres Protokolleinträge-Objekt
/metadata/logs/{uuid}	
GET	liefert einen spezifischen Protokolleintrag der Anwendung
DELETE	löscht einen spezifische Protokolleintrag der Anwendung

Tabelle 45: URI-Mapping für Protokolleinträge der Anwendung

8.4.3 Projekte

In der folgenden Tabelle 46 wird das URI-Mapping für alle Projekte der Anwendung dargestellt. Das beinhaltet nicht nur die Hauptprojekte, welche ein eigenes Datenbankschema besitzen, sondern auch für die darin enthaltenen Unterprojekte.

HTTP	Beschreibung
/metadata/projects	
GET	liefert Liste aller Projekte der Anwendung
POST	erstellt ein neues Hauptprojekt in der Anwendung
/metadata/projects/count	
GET	liefert die Anzahl der Projekte der Anwendung
/metadata/projects/new	
GET	liefert ein leeres Projekte-Objekt
/metadata/projects/{uuid}	
GET	liefert ein spezifisches Projekt der Anwendung
PUT	ändert ein spezifisches Projekt (bezieht sich auf die Verbindungsdaten) der Anwendung
DELETE	löscht ein spezifisches Projekt der Anwendung

Tabelle 46: URI-Mapping für Projekte der Anwendung

8.4.4 Datenbankverbindungen der Projekte

In der folgenden Tabelle 47 wird das URI-Mapping für die Datenbankverbindungen der Projekte in der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/dbconnections	
GET	liefert Liste aller Datenbankverbindungen der Anwendung
POST	erstellt eine neue Datenbankverbindung in der Anwendung
/metadata/dbconnections/count	
GET	liefert die Anzahl der Datenbankverbindungen der Anwendung
/metadata/dbconnections/new	
GET	liefert ein leeres Datenbankverbindungen-Objekt
/metadata/dbconnections/{uuid}	
GET	liefert eine spezifische Datenbankverbindung der Anwendung
PUT	ändert eine spezifische Datenbankverbindung der Anwendung
DELETE	löscht eine spezifische Datenbankverbindung der Anwendung

Tabelle 47: URI-Mapping für die Datenbankverbindungen der Projekte in der Anwendung

8.4.5 Zugangsdaten

In der folgenden Tabelle 48 wird das URI-Mapping für die Zugangsdaten der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/credentials	
GET	liefert Liste aller Zugangsdaten der Anwendung
POST	erstellt ein neues Zugangsdatum in der Anwendung
/metadata/credentials/count	
GET	liefert die Anzahl der Zugangsdaten der Anwendung
/metadata/credentials/new	
GET	liefert eine leeres Zugangsdaten-Objekt
/metadata/credentials/{uuid}	
GET	liefert ein spezifisches Zugangsdatum der Anwendung
PUT	ändert ein spezifisches Zugangsdatum der Anwendung
DELETE	löscht ein spezifisches Zugangsdatum der Anwendung

Tabelle 48: URI-Mapping für die Zugangsdaten der Anwendung

8.4.6 Datenbanken

In der folgenden Tabelle 49 wird das URI-Mapping für die Datenbanken der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/databases	
GET	liefert Liste aller Datenbanken der Anwendung
POST	erstellt eine neue Datenbank in der Anwendung
/metadata/databases/count	
GET	liefert die Anzahl der Datenbanken der Anwendung
/metadata/databases/new	
GET	liefert eine leeres Datenbanken-Objekt
/metadata/databases/{uuid}	
GET	liefert eine spezifische Datenbank der Anwendung
PUT	ändert eine spezifische Datenbank der Anwendung
DELETE	löscht eine spezifische Datenbank der Anwendung

Tabelle 49: URI-Mapping für die Datenbanken der Anwendung

8.4.7 Einstufungen der Protokolleinträge

In der folgenden Tabelle 50 wird das URI-Mapping für die Einstufungen der Protokolleinträge der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/level	
GET	liefert Liste aller Einstufungen der Protokolleinträge der Anwendung
POST	erstellt eine neue Einstufung der Protokolleinträge in der Anwendung
/metadata/level/count	
GET	liefert die Anzahl der Einstufungen der Protokolleinträge der Anwendung
/metadata/level/new	
GET	liefert eine leeres Protokolleintrageinstufungen-Objekt
/metadata/level/{uuid}	
GET	liefert eine spezifische Einstufung der Protokolleinträge der Anwendung
PUT	ändert eine spezifische Einstufung der Protokolleinträge der Anwendung
DELETE	löscht eine spezifische Einstufung der Protokolleinträge der Anwendung

Tabelle 50: URI-Mapping für die Einstufungen der Protokolleinträge der Anwendung

8.4.8 Auslöser der Protokolleinträge

In der folgenden Tabelle 51 wird das URI-Mapping für die Auslöser der Protokolleinträge der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/logger	
GET	liefert Liste aller Auslöser der Protokolleinträge der Anwendung
POST	erstellt einen neuen Auslöser der Protokolleinträge in der Anwendung
/metadata/logger/count	
GET	liefert die Anzahl der Auslöser der Protokolleinträge der Anwendung
/metadata/logger/new	
GET	liefert eine leeres Protokolleintrageauslöser-Objekt
/metadata/logger/{uuid}	
GET	liefert einen spezifischen Auslöser der Protokolleinträge der Anwendung
PUT	ändert einen spezifischen Auslöser der Protokolleinträge der Anwendung
DELETE	löscht einen spezifischen Auslöser der Protokolleinträge der Anwendung

Tabelle 51: URI-Mapping für die Auslöser der Protokolleinträge der Anwendung

8.4.9 Datenbankserverports

In der folgenden Tabelle 52 wird das URI-Mapping für die Datenbankserverports der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/ports	
GET	liefert Liste aller Datenbankserverports der Anwendung
POST	erstellt einen neuen Datenbankserverport in der Anwendung
/metadata/ports/count	
GET	liefert die Anzahl der Datenbankserverports der Anwendung
/metadata/ports/new	
GET	liefert eine leeres Datenbankserverports-Objekt
/metadata/ports/{uuid}	
GET	liefert einen spezifischen Datenbankserverport der Anwendung
PUT	ändert einen spezifischen Datenbankserverport der Anwendung
DELETE	löscht einen spezifischen Datenbankserverport der Anwendung

Tabelle 52: URI-Mapping für die Datenbankserverports der Anwendung

8.4.10 PostgreSQL-Schemata

In der folgenden Tabelle 53 wird das URI-Mapping für die PostgreSQL-Schemata der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/schemas	
GET	liefert Liste aller PostgreSQL-Schemata der Anwendung
POST	erstellt ein neues PostgreSQL-Schema in der Anwendung
/metadata/schemas/count	
GET	liefert die Anzahl der PostgreSQL-Schemata der Anwendung
/metadata/schemas/new	
GET	liefert eine leeres PostgreSQL-Schemata-Objekt
/metadata/schemas/{uuid}	
GET	liefert ein spezifisches PostgreSQL-Schema der Anwendung
PUT	ändert ein spezifisches PostgreSQL-Schema der Anwendung
DELETE	löscht ein spezifisches PostgreSQL-Schema der Anwendung

Tabelle 53: URI-Mapping für die PostgreSQL-Schemata der Anwendung

8.4.11 Datenbankserver

In der folgenden Tabelle 54 wird das URI-Mapping für die Datenbankserver der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/servers	
GET	liefert Liste aller Datenbankserver der Anwendung
POST	erstellt einen neuen Datenbankserver in der Anwendung
/metadata/servers/count	
GET	liefert die Anzahl der Datenbankserver der Anwendung
/metadata/servers/new	
GET	liefert eine leeres Datenbankserver-Objekt
/metadata/servers/{uuid}	
GET	liefert einen spezifischen Datenbankserver der Anwendung
PUT	ändert einen spezifischen Datenbankserver der Anwendung
DELETE	löscht einen spezifischen Datenbankserver der Anwendung

Tabelle 54: URI-Mapping für die Datenbankserver der Anwendung

8.4.12 Konfigurationsschlüssel

In der folgenden Tabelle 55 wird das URI-Mapping für die Konfigurationsschlüssel der Anwendung dargestellt.

HTTP	Beschreibung
/metadata/settingkeys	
GET	liefert Liste aller Konfigurationsschlüssel der Anwendung
POST	erstellt einen neuen Konfigurationsschlüssel in der Anwendung
/metadata/settingkeys/count	
GET	liefert die Anzahl der Konfigurationsschlüssel der Anwendung
/metadata/settingkeys/new	
GET	liefert eine leeres Konfigurationsschlüssel-Objekt
/metadata/settingkeys/{uuid}	
GET	liefert einen spezifischen Konfigurationsschlüssel der Anwendung
PUT	ändert einen spezifischen Konfigurationsschlüssel der Anwendung
DELETE	löscht einen spezifischen Konfigurationsschlüssel der Anwendung

Tabelle 55: URI-Mapping für die Konfigurationsschlüssel der Anwendung

8.5 Suche

Dieses Kapitel beschreibt, in der folgenden Tabelle 56, das URI-Mapping für die Suche.

HTTP	Beschreibung
/search	
GET	liefert das Suchergebnis zu einer im Query Parameter (siehe 8.7.2) angegebenen Anfrage
/search/index	
GET	Platzhalter für die Initialisierung der Indexierung des Datenbestandes

Tabelle 56: URI-Mapping für Konfiguration der Anwendung

8.6 sonstige Informationen

Dieses Kapitel beschreibt, in der folgenden Tabelle 57, das URI-Mapping für weitere OpenInfRA relevante Informationen.

HTTP	Beschreibung
/version	
GET	liefert die aktuelle Versionsnummer von OpenInfRA
/application.html	
GET	liefert Liste mit allen zur Verfügung stehenden REST-Schnittstellen

Tabelle 57: URI-Mapping sonstige OpenInfRA relevante Informationen

8.7 Abfrageparameter

In diesem Kapitel werden alle zusätzlichen URI-Parameter erläutert.

8.7.1 Pagination

Für die Pagination werden drei zusätzliche Parameter benötigt. Dabei ist Anzumerken, dass es für die Sortierung innerhalb der Fachdaten, bereits ein entsprechendes Attribut *Order* gibt (siehe Abbildung 6).

1. *Sort*: Sortierung der Objekte
2. *Offset*: Startnummer der zurückzuliefernden Objekte
3. *Count*: Anzahl der zurückzuliefernden Objekte

8.7.2 Suchanfrage

Für die Übermittlung einer Suchanfrage, steht der Parameter *query* zur Verfügung. Dieser wird ausschließlich in der unter 56 aufgeführten Suchanfrage verwendet.

8.7.3 Kaskadierendes Löschen

Dieser Parameter hat ausschließlich Auswirkungen auf die HTTP-DELETE-Methode. Der Default-Wert ist *false*.

- {domain}/{ressource}?cascade={true | false}

Damit kaskadierendes Löschen auf der Datenbank möglich wird, müsste bereits die Tabellendefinition mit *CASCADE* versehen werden. Da dies jedoch Primärschlüssel-Constraints aushebelt (siehe Listing 8.1), wird von dieser Variante abgesehen.

Wenn nun bei einer HTTP-DELETE-Methode der Parameter *cascade* mit dem Wert *true* übergeben wird, dann wird der konkrete *DELETE*-Befehl, wie im Listing 8.1 dargestellt, kaskadierend durchgeführt. Wird der Parameter *cascade* gar nicht übergeben oder mit dem Wert false, dann wird das Delete-Statement nicht kaskadierend ausgeführt.

Listing 8.1: Kaskadierendes Löschen durch Ändern der Fremdschlüsselbeziehung

```
1  BEGIN;
2
3  ALTER TABLE {table_name_1}
4  DROP CONSTRAINT {foreign_key_1},
5  ADD CONSTRAINT {foreign_key_1} FOREIGN KEY (id_2)
6  REFERENCES {table_name_2} (id) ON DELETE CASCADE;
7
8  {delete_statement}
9
10 ALTER TABLE {table_name_1}
11 DROP CONSTRAINT {foreign_key_1},
12 ADD CONSTRAINT {foreign_key_1} FOREIGN KEY (id_2)
13 REFERENCES {table_name_2} (id);
14
15 COMMIT;
```

8.8 Mehrsprachigkeit – HTTP-Header

Für die Mehrsprachigkeit wird der HTTP-Standard verwendet!

Für die Mehrsprachigkeit wird an der URI jeweils ein Parameter *lang* angegeben, der sich aus einer Sprachcodierung und einer Länderbezeichnung zusammensetzt. Die Sprachcodierung enthält nach dem ISO-Standard *ISO 639-1* zwei Kleinbuch-

staben. Die Länderbezeichnung enthält nach dem ISO-Standard *ISO-3166-1* zwei Großbuchstaben. Die Sprachcodierung und die Länderbezeichnung werden durch einen Unterstrich voneinander getrennt:

lang=Sprachcodierung_Länderbezeichnung.

Die folgenden drei Beispiele zeigen den Parameter *lang* für die Sprachen und Gebiete: Deutsch/Deutschland, Deutsch/Österreich und Französisch/Schweiz.

- {domain}/projects?lang=de_DE
- {domain}/projects?lang=de_AT
- {domain}/projects?lang=fr_CH

Die folgende Auflistung beschreibt die Auswahl der Sprache anhand des Parameters *lang*, der Default-Sprache für den Benutzer und die Default-Sprache für das Projekt. Die Reihenfolge definiert die Priorität der Sprachauswahl (Bezug auf AC_1022 - Priorisierung der Sprache).

1. **Parameter *lang***: Jede Anfrage sollte mit dem Parameter *lang* übermittelt werden. Ist diese Sprache für den gewünschten Datensatz verfügbar, dann wird diese Sprache zurückgeliefert.
2. Die Default-Sprache wird separat für jeden Benutzer in der Account Datenbank gespeichert. Sollte für den gewünschten Datensatz
3. Es gibt für jedes Projekt eine Default-Sprache in der Metadatenbank (Default-Sprache des Benutzers ist höherwertig)

9 Role-based Access Control (Benutzer-Rollen-Rechte)

In diesem Kapitel wird das Benutzer-Rollen-Rechte-System, welches im Folgenden als Role-based Access Control (RBAC) bezeichnet wird, detailliert beschrieben.

9.1 RBAC-Datenbankschema

Das RBAC-Datenbankschema (Abbildung 37) stellt die Basis für das RBAC-System dar. Allgemein gesagt sollen die folgenden Informationen in der abgeleiteten RBAC-Datenbank gespeichert werden:

- Benutzer mit sämtlichen Informationen
- Rollen, welche einem Benutzer zugewiesen werden können
- Rechte, die an Rollen und damit implizit an Benutzer vergeben werden können

Ein wesentlicher Bestandteil ist die Definition von Rollen und deren Projektbezug. Diese Definition wird im Folgenden erläutert.

9.2 Rollen und deren Projektbezug

Zunächst wird zwischen *allgemeinen Rollen* und *projektbezogene Rollen* unterschieden. Allgemeine Rollen zeichnen sich dadurch aus, dass die universell und projektübergreifend gelten und können frei definiert werden. Projektbezogene Rollen sind Rollen, die nur in Verbindung mit einem konkreten Projekt gelten und im Voraus fest definiert sind. Die folgenden allgemeinen Rollen können für OpenInfRA identifiziert werden:

- **Systemadministrator (sysadmin)**: Der Systemadministrator soll das Nutzermanagement und die Projektverwaltung durchführen können. Das schließt die Zuordnung von Nutzern zu Projekten und die allgemeine Konfiguration von OpenInfRA mit ein.
- **Systembearbeiter (syseditor)**: Der Systembearbeiter soll Inhalte der Systemdatenbank lesen und bearbeiten können. Auf eine detaillierte Zugriffskontrolle der einzelnen Objekte in der Systemdatenbank wird hier verzichtet.

- **Systemgast (sysguest)**: Der Systemgast soll ausschließlich lesenden Zugriff auf die Inhalte der Systemdatenbank haben.

Zusätzlich können die folgenden projektbezogenen Rollen identifiziert werden:

- **Projektadministrator (projectadmin)**: Der Projektadministrator soll das Nutzermanagement für das Projekt verwalten und entsprechende Konfiguration durchführen können. Zusätzlich kann der Projektadministrator das Projekt umbenennen und den Beschreibungstext ändern.
- **Projektbearbeiter (projecteditor)**: Der Projektbearbeiter soll Inhalte der Projektdatenbank lesen und bearbeiten können. Im Gegensatz zum Systembearbeiter soll hier eine detaillierte Zugriffskontrolle der einzelnen Objekte umgesetzt werden. Zunächst werden die Editierfunktion und Sichtbarkeit auf Projektebene kontrolliert. Weiterhin wird eine Zugriffskontrolle auf Ebene der Themenausprägungen umgesetzt.
- **Projektgast (projektguest)**: Der Projektgast soll ausschließlich lesenden Zugriff auf die Inhalte der Systemdatenbank haben. Wie beim Projektbearbeiter wird eine detaillierte Zugriffskontrolle der einzelnen Objekte umgesetzt

9.3 Genehmigungen

Genehmigungen (*permissions*) werden für jede Ressource bzw. URI einzeln vergeben. Folgendes Beispiel soll das verdeutlichen. Es gibt bspw. die URI */system*. Die entsprechende Genehmigungen hat die folgende Form:

- '/system:[r|w|r,w]'.

Dabei deutet der Buchstabe 'r' an, dass ein Leserecht eingeräumt wird. Der Buchstabe 'w' räumt das Recht zum ändern der Ressource ein. Zusätzlich wird für ein Projekt mit der UUID 'fd27a347-4e33-4ed7-aebc-eeff6dbf1054' wie folgt angegeben:

- '/projects/{id}:[r|w|r,w]:fd27a347-4e33-4ed7-aebc-eeff6dbf1054'.

Sollte der Zugriff auf mehrere Projekte gleichzeitig eingeräumt werden, können nach dem zweiten Doppelpunkt die konkreten UUIDs mithilfe von Kommata getrennt angegeben werden. Dabei ist zu beachten, dass die folgenden Notationen zum Zugriff auf sämtliche Projekte führen:

- '/projects/{id}:[r|w|r,w]' ist gleich '/projects/{id}:[r|w|r,w]:*'.

Aktuell ist geplant das RBAC-System bis auf die Ebene der Themenausprägungen auszubauen. Die Genehmigung für den Zugriff auf alle Themenausprägungen eines Projektes ist wie folgt:

- '/projects/fd27a347-4e33-4ed7-aebc-eeff6dbf1054/topiccharacteristics/{id}:[r|w|r,w]:UUIDs'.

Im Folgenden wird der UML-Entwurf des RBAC-Datenbankschemas näher erläutert.

9.3.1 Benutzer (Subject)

Der Begriff 'Benutzer' wird im Schema als *subject* übersetzt. Die Klasse *subject* enthält sämtliche Information über einen Benutzer, der im System registriert ist.

- *id (PK)*: Primärschlüssel für den Benutzer
- *login*: Benutzername der zum Anmelden am System verwendet wird
- *password*: Passwort zur Authentifizierung des Benutzers
- *password_created_on*: Erstellungsdatum des Passwortes
- *salt*: Salt der zur Verschlüsselung des Passwortes genutzt wird (bei jedem Passwort neu generierte UUID)
- *mail*: Email Adresse des Benutzers
- *name*: Name des Benutzers
- *description*: Beschreibung des Benutzers
- *status*: Der Status des Benutzer Accounts, z. Bsp. Inaktiv (0), Aktiv (1), Gesperrt (-1)
- *default_language*: Standard Sprache des Benutzers
- *created_on*: Datum, an dem der Benutzer angelegt wurde
- *updated_on*: Datum, an dem der Benutzer editiert wurde
- *last_login_on*: Datum, an dem sich der Benutzer das letzte mal angemeldet hat

9.3.2 Allgemeine Rollen

Die Klasse *role* enthält alle allgemeinen Rollen ohne Projektbezug. Diese Rollen können einem oder mehreren Benutzern zugeordnet werden und enthalten konkrete Genehmigungen, die anschließend beschrieben werden. Allgemeine Rollen und die dazugehörigen Genehmigungen können frei vergeben werden.

- *id (PK)*: Primärschlüssel für die Rolle
- *name*: Name der Rolle
- *description*: Beschreibung der Rolle

9.3.3 Genehmigungen

Die Klasse *permission* beinhaltet alle vergebenen Genehmigungen die sich auf konkrete Ressourcen bzw. URIs beziehen. Dazu wurde der Ansatz *domain:action:object* gewählt. Allgemeine Rollen und die dazugehörigen Genehmigungen können frei vergeben werden.

- *id (PK)*: Primärschlüssel für die Berechtigung
- *description*: Beschreibung der Berechtigung
- *permission*: Shiro-Berechtigungszeichenkette

9.3.4 Passwort Historie

Die Klasse *Passwort-Historie* verwaltet die bereits vorher durch Benutzer verwendete Passwörter.

- *id (PK)*: Primärschlüssel eines Passwortes aus der Historie
- *password*: Verschlüsseltes Passwort
- *salt*: Salt der zur Verschlüsselung des Passwort genutzt wurde
- *password_created_on*: Erstellungsdatum des Passwortes

9.3.5 Passwort-Blacklist

Die Klasse *Passwort-Blacklist* beinhaltet Passwörter im Klartext, welche nicht als Passwort verwendet werden dürfen.

- *id (PK)*: Primärschlüssel eines Passwortes auf der Blacklist
- *password*: Passwort, dass nicht verwendet werden darf

9.3.6 Projektbezogene Genehmigungen

Für projektbezogene Genehmigungen gibt es zwei weitere Tabellen: 'subject_project' und 'subject_tc'. Alle sich aus dieser Tabellenstruktur ergebenen Genehmigungen werden automatisch/algorithmisch generiert bzw. erzeugt. Die Tabelle 'subject_project' stellt einen Projektbezug von einem Benutzer mit einer vorher definierten projektbezogenen Rolle her. Um die Genehmigung zu erzeugen wird die projektbezogene Rolle interpretiert und je nach Rolle Lese- oder Schreibrecht eingeräumt: '/projects/{id}:[r|w|r,w]:UUIDs'. Diese Tabelle ist wie folgt definiert:

- *id (PK)*: Primärschlüssel
- *subject_id*: Benutzer, auf den sich der Eintrag bezieht
- *project_id*: Projekt, auf das sich der Eintrag bezieht
- *project_related_role*: Projektbezogene Rolle

Die Tabelle 'subject_tc' stellt einen Projektbezug von einem Benutzer und einer Menge von Themenausprägungen her. In diesem Fall muss das Lese- bzw. Schreibrecht explizit angegeben werden:

- *id (PK)*: Primärschlüssel
- *subject_id*: Benutzer, auf den sich der Eintrag bezieht
- *project_id*: Projekt, auf das sich der Eintrag bezieht
- *tc_id*: Themenausprägungen, auf die sich der Eintrag bezieht
- *read*: Leserecht
- *write*: Schreibrecht

9.3.7 RBAC für GeoServer

Der GeoServer, der für das WebGIS benötigt wird, nutzt ein eigenes Benutzer- / Rollen- / Rechtesystem. Dabei werden Benutzer und Rollen in einer Datenbank und Berechtigungen in *Java Propertie Files* abgelegt. Um die Verbindung zu OpenInfRA herzustellen, soll der GeoServer auf die OpenInfRA Klassen zugreifen können. Dadurch sind die die Benutzer und Rollen sowohl in OpenInfRA, als auch im GeoServer identisch. Das hat allerdings zur Folge, dass die im GeoServer bearbeiteten Benutzer und Rollen direkte Auswirkungen auf OpenInfRA haben. Um hier ein Trennung vorzunehmen, soll es für den GeoServer eine eigenständige Rollen Klasse geben, welche die Nutzer von OpenInfRA verwendet.

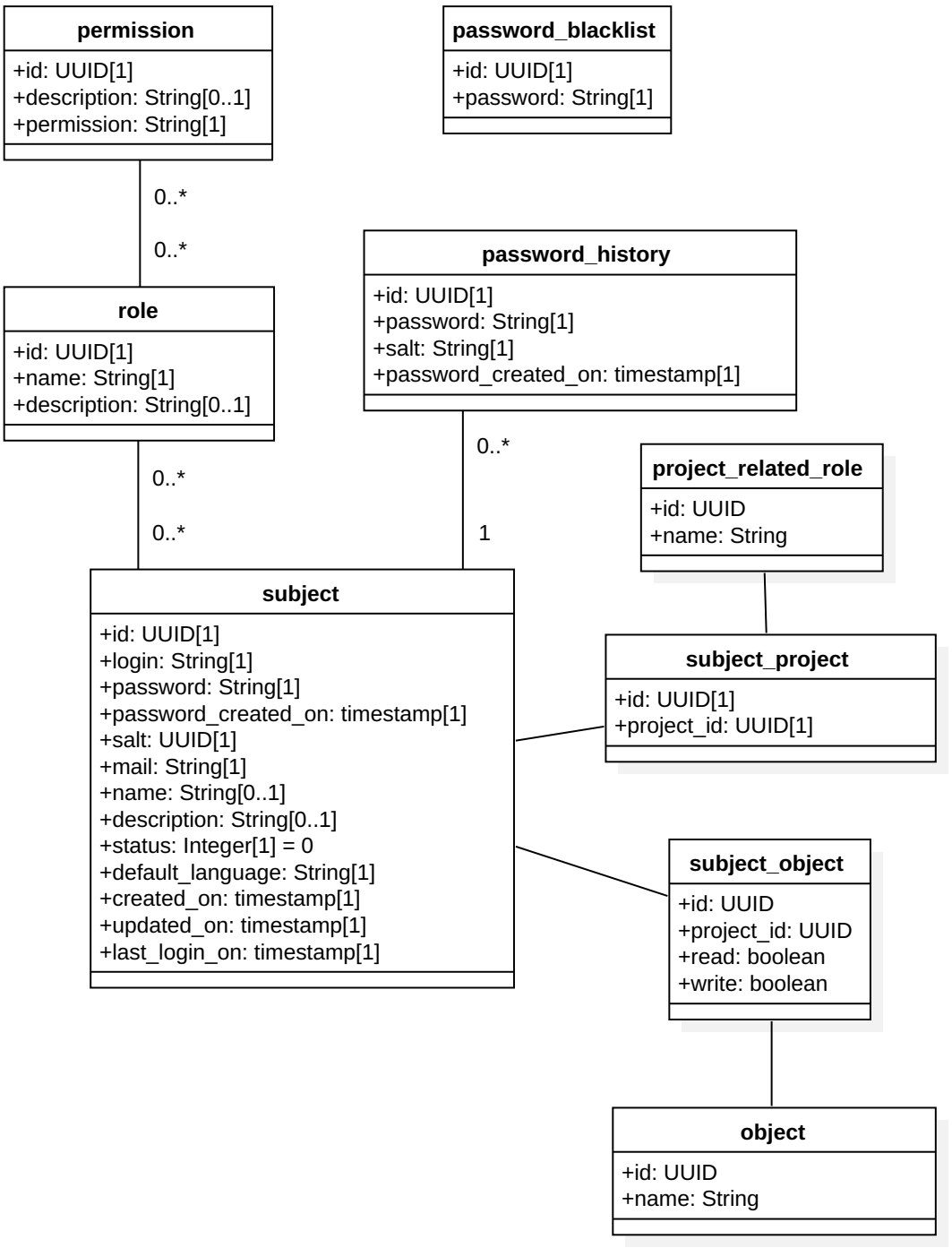


Abbildung 37: RBAC-Datenbankschema

10 Konfiguration

Für den Betrieb von OpenInfRA sind Konfigurationen zwingend notwendig. Mit ihnen werden verschiedene Systemparameter für die korrekte Funktionsweise bereit gestellt. Die Konfigurationen liegen dabei in zwei Varianten vor und decken unterschiedliche Bereiche ab.

1. Konfigurationsdatei (Kapitel 10.1)
2. Konfigurationstabelle (Kapitel 10.2)

10.1 Konfigurationsdatei

Die Konfigurationsdatei beinhaltet ausschließlich die Verbindungsdaten zur OpenInfRA-Datenbank. Dabei sollen die in Tabelle 58 aufgelisteten Einträge den Bestandteil der Konfigurationsdatei bilden.

Konfiguration	Beschreibung
server	Hostname oder IP-Adresse des Datenbankservers auf dem die OpenInfRA-Datenbank läuft
port	Port für den Datenbankserver
dbname	Datenbank für die OpenInfRA Schemata
user	Name des Datenbankbenutzers für den Zugriff auf die OpenInfRA-Datenbank
password	Passwort für den Datenbankbenutzer

Tabelle 58: Datenbank-Konfigurationsdatei

Für eine vollständige Liste der möglichen Konfigurationseinträge wird auf Kapitel 14.19 verwiesen.

10.2 Konfigurationsdatenbank

Sämtliche weitere, die OpenInfRA-Anwendung betreffende, Konfigurationen werden in der Datenbank gespeichert. Dazu enthält die Meta-Datenbank die Tabelle *settings* (siehe Kapitel 5.3). Innerhalb dieser Tabelle sollen alle System-, Projekt- und Plug-In- Konfigurationen abgelegt werden. Beispiele für Systemeinstellungen werden in der Tabelle 59 dargestellt.

key	**value**
de.btu.openinfra.backend.rest.defaultOffset	0
de.btu.openinfra.backend.defaultSize	20
de.btu.openinfra.plugins.solr.server	localhost
de.btu.openinfra.plugins.solr.port	8983

Tabelle 59: Beispielhafte Darstellung von Systemeinstellungen

11 WebGIS

Das WebGIS stellt eine Erweiterung zum Basissystem OpenInfRA dar. Es zielt darauf ab die Arbeit mit Geodaten, die mit dem Kernsystem aufgenommen und verwaltet werden sollen, zu erleichtern. Die wichtigste Komponente des WebGIS stellt der als JavaScript-Anwendung konzipiert WebGIS-Client dar. D.h., dass der überwiegende Teil der Logik auf Seiten des Clients im Sinne einer Rich Internet Application implementiert ist. Konsumiert werden dabei in erster Linie standardisierte Schnittstellen der Geodatenwelt wie Web Mapping Service (WMS) für Raster- und Web Feature Service (WFS) für Vektordaten. So ist es möglich über die JavaScript-Anwendung neben den eigentlich OpenInfRA-Geodaten eben auch auf beliebige Geodatenservices Dritter zuzugreifen und Informationen sinnvoll zu verknüpfen. Des weiteren können so Anforderungen mit Hinblick auf die Integration des WebGIS in Webangebote Dritter erfüllt werden, indem die benötigten Skripte direkt in anderen Webseiten eingebunden werden können.

Als grundlegende Serverkomponente des WebGIS kann der OSGeo GeoServer angesehen werden der eben jene geforderten OWS-Schnittstellen zur Verfügung stellt. Allgemein folgt die Wahl des OWS Servers an dieser Stelle der konkreten Nennung im Anforderungskatalogs. Durch die Nutzung der wohldefinierten OGC Schnittstellen ist aber auch die Nutzung anderer OWS Server denkbar.

11.1 Technische Anforderungen

Grundvoraussetzung zum Ausführen der Clientanwendung im Browser ist die Unterstützung von JavaScript und HTML5 (bspw. Fullscreen API, File API). Im wesentlichen sind diese Vorraussetzungen durch die im Moment am verbreitetsten Browser in ihrer aktuellen Version erfüllt (s.a. [1] bzw. [2]). Die hier vorgeschlagenen Serverkomponenten bauen auf den durch das Kernsystem OpenInfRA genutzten Komponenten auf. Im folgenden Kapitel sind darüber hinaus beteiligte Komponenten beschrieben.

[1] http://caniuse.com/#search=fullscreen
[2] http://caniuse.com/#search=file

11.1.1 Server

PostGIS 2.1.5 Stellt geometrische und geographische Datentypen und Funktionen für PostgreSQL zur Verfügung die die Nutzung mit komplexen Geodaten ermöglichen. Die Erweiterung ist bereits Teil der Installation des Kernsystems OpenInfRA.[3]

Apache Tomcat 7 Ein OpenSource Servlet Container der zum Ausführen des GeoServer Servlets dient. Im Allgemeinen ist die Nutzung des selben Containers möglich in dem die OpenInfRA-Kernanwendung ausgeführt wird. Alternativ ist aber auch die verteilte Installation über mehrere Tomcat-Instanzen denkbar und unter Umständen zur Performanzoptimierung sinnvoll.[4]

OSGeo GeoServer 2.7.1.1 Ein Open Source Server der OpenGIS Web Services (OWS) zur Verfügung stellen kann. Als Datengrundlage können eine Vielzahl von Quellen, unter anderem ESRI Shapefiles oder GeoTiffs genutzt werden. Im Kontext OpenInfRA wird eine Anbindung via PostgreSQL/PostGIS genutzt um Geodaten über den Server und dessen standardisierte Schnittstellen zur Verfügung zu stellen.

(OSGeo GeoServer Print Extension) Eine optionale Komponente die dem Kartendruck bzw. als Geo Document Service (GDS) dient und als Erweiterung zum GeoServer installiert werden kann. Alternativ ist die Nutzung des eigenständigen MapFish Print Servlets möglich, dass auch für die GeoServer-Erweiterung als Grundlage dient. Über eine Konfigurationsdatei im YAML-Format können Druck-Templates erstellt werden, die über eine einfache REST-Schnittstelle abgerufen werden. Diese Komponente ist im Kapitel 12 näher beschrieben. (Download unter `http://geoserver.org/release/stable/` Extensions printing)

(HTTP Proxy) Aktuelle Browser setzen die Same-Origin-Policy (SOP) [5] strikt um. Das führt zu Problemen beim Konsum externer Web Services die Cross-Origin Ressource Sharing (CORS) nur unzureichend unterstützen. Historisch gewachsen ist dieser Umstand von OWS-Diensten welche zumeist über Desktop-GIS Systeme problemlos genutzt werden können, bei einem Zugriff über AJAX-Requests im Browser aber die Zusammenarbeit verweigern. Der WebGIS-Client erlaubt die Konfiguration einer Proxy-URL die genutzt werden kann um AJAX-Anfragen über die aktuelle Domaine umzuleiten und die Server-

[3] `http://postgis.net/`
[4] `http://tomcat.apache.org/`
[5] `https://de.wikipedia.org/wiki/Same-Origin-Policy`

Client-Kommunikation so zu gewährleisten. (Beispiel https://github.com/terrestris/JSP-Whitelist-Proxy)

Im Folgenden sind Hinweise zur Installation der einzelnen Komponenten zusammengefasst.

11.1.2 Apache Tomcat

Für die Handhabe großer Datenmengen und die parallele Ausführung der OpenInfRA-Kernanwendung und GeoServer kann es nötig sein den Speicherbedarf des Servlet Containers anzupassen. Als Vorschlag können die folgenden Werte genutzt werden:

```
JAVA_OPTS="-server -Djava.awt.headless=true -Xms128m -Xmx512m
                    -XX:MaxPermSize=256m"
```

Insbesondere die Erhöhung des Permananent generation size[6] ist beim Einsatz von GeoServer nötig.[7]

Die Konfiguration ist für jedes Betriebssystem individuell unterschiedlich. Unter der Linux Distribution Ubunutu erfolgt sie bei Installationen von Apache Tomcat via `apt-get`-Befehl in der Regel über die Datei `/etc/default/tomcat7`.

Optional ist es für einen optimalen Betrieb der OWS-Dienste sinnvoll die Cross-Origin Policy des Servlet Containers anzupassen. Zumindest der Zugriff auf den dem GeoServer zugeordneten Kontext (i.d.R. `/geoserver`) erscheint sinnvoll. Die Konfiguration erfolgt über die Datei `$CATALINA_HOME/conf/web.xml` der Tomcat-Installation. Entsprechend könnte die Filterkonfiguration folgendermaßen aussehen (s.a. [8]):

```
<filter>
  <filter-name>CorsFilter</filter-name>
  <filter-class>
        org.apache.catalina.filters.CorsFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

[6] http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
[7] http://docs.geoserver.org/stable/en/user/production/container.html
[8] http://enable-cors.org/server_tomcat.html

11.1.3 OSGeo GeoServer

Die Installation des GeoServers beschränkt sich auf das Ablegen des aktuell frei verfügbaren WAR-Archivs im `$CATALINA_HOME/webapps` Verzeichnis des Servlet Containers. Nach kurzer Zeit sollte das Archiv entpackt und mit dem entsprechenden Kontext geladen werden (i.d.R. `/geoserver`). Anschließend ist die Web-Interface unter dieser Adresse aufrufbar (Standardlogin: `admin:geoserver`. Bei dieser Standardroutine wird wird ein im WAR mitgeliefertes Datenverzeichnis initialisiert und im Verzeichnis des entpackten GeoServer-Servlets abgelegt (`$GEOSERVER_DATA_DIR/data`). Diese Konfiguration wird auch beim ersten Starten des GeoServer geladen. Es enthält Konfigurationsdateien und externe Demo-Geodaten. Die konkrete Gestalt des Verzeichnisses und die Bewandtnis der einzelnen Unterverzeichnisse ist in der Dokumentation[9] des Produkts ausführlich beschrieben.

11.1.3.1 Konfiguration Datenverzeichnis

Während diese erste Demo-Umgebung beliebig gehandhabt werden kann, ist zu gewährleistet, dass für ein produktives System beim Upgrade der GeoServer-Installation (durch Austausch des Web-Archivs durch eine neuere Version) das Datenverzeichnis erhalten bleibt. So ist es sinnvoll per Umgebungsvariable auf ein fest definiertes Datenverzeichnis außerhalb des Servlet Kontexts zu verweisen auf der der ausführende Nutzer des Servlet Containers lesend und schreibend zugreifen kann. Die Konfiguration könnte folgendermaßen aussehen:

```
1 export GEOSERVER_DATA_DIR=/home/users/tomcat/geoserver-data
```

Üblicherweise bietet es sich an diesen Export direkt als Umgebungsvariable der Tomcat Installation zu setzen (siehe auch Kapitel 11.1.2). Die Einstellung wird so beim Start des Servlets anstatt des Demo-Verzeichnis gelesen und interpretiert. Über das Web-Interface können Administratoren unter dem Menüpunkt `Serverstatus` überprüfen ob die Konfiguration erfolgreich war. Alternativ ist es möglich das Verzeichnis über die `web.xml` des GeoServer-Servlets zu setzen[10]. Zu beachten ist in diesem Fall, dass die Einstellung nach einem Upgrade wiederholt werden muss.

[9] http://docs.geoserver.org/stable/en/user/datadirectory/data-dir-structure.html

[10] http://docs.geoserver.org/latest/en/user/datadirectory/data-dir-setting.html#servlet-context-parameter

11.1.3.2 Verwalten von Objekten und Diensten

GeoServer ermöglicht das Verwalten von OWS-Diensten über die entsprechende Geodaten zur Verfügung gestellt werden können. Im einzelnen sind dies:

Web Map Service (WMS) Schnittstelle zum Abruf von Karten[11].

Web Feature Service (WFS) Schnittstelle zum Abruf von Geodaten im Vektorformat[12].

Web Coverage Service (WCS) Schnittstelle zum Abruf von Geodaten zur Darstellung raum- und zeitvariianter Phänomene[13].

Die Anforderungserhebung OpenInfRAs umfasst lediglich die Umsetzung eines WMS und WFS-Dienstes. Als Grundlage dieser Dienste dienen die folgenden durch GeoServer zu verwaltenden Objekte:

Arbeitsbereich Die oberste Struktureinheit der Datenspeicher und Layer, aber auch Stile und Gruppenlayer zugeordnet werden können. Objekte die einem Arbeitsbereich zugeordnet sind, sind nicht in anderen Arbeitsbereichen verfügbar.

Datenspeicher Ein konkreter Zeiger auf eine Datenquelle, beispielsweise ein Verzeichnis von ESRI Shapefiles, einer GeoTIFF Rasterdatei oder aber einer Datenbankverbindung.

Layer Feature Types die über die WMS/WFS-Schnittstellen angesprochen werden können. Als Datengrundlage eines Layers dienen die konfigurierten Datenspeicher.

Stile Unter Stile versteht sich die optische Ausgestaltung der als Layer definierten Feature Types. Sie erfolgt im Styled Layer Descriptor-Format (SLD [14]). Üblicherweise wird beim Anlegen eines Layers für diesen ein Standardstil festgelegt. Dieser kann durch neu definierte Stile beliebig geändert werden. Einem Layer können in der Regel auch mehrere Stile zugeordnet sein, die über einen Client zur Auswahl stehen.

Das Einrichten der beschrieben Objekte ist über das GeoServer-Webinterface vorzunehmen und sollte ohne weitere Beschreibung auskommen. Weitere Informationen können der Dokumentation[15] des Produkts entnommen werden. Eigenheiten bei der

[11] https://de.wikipedia.org/wiki/Web_Map_Service
[12] https://de.wikipedia.org/wiki/Web_Feature_Service
[13] https://de.wikipedia.org/wiki/Web_Coverage_Service
[14] http://docs.geoserver.org/stable/en/user/styling/sld-introduction.html
[15] http://docs.geoserver.org/stable/en/user/

Einrichtung einer OpenInfRA-Projektdatenbank als Datenquelle werden in Kapitel 11.2.2 näher beschrieben.

11.1.4 Definition nutzerspezifischer Projektionen

Sowohl die PostgreSQL-Erweiterung PostGIS als auch der OpenGeo-GeoServer unterstützen eine Vielzahl bekannter Projektionen zur Darstellung und Transformation von Geodaten. Sollte es jedoch nötig sein, eigenen Projektionen zu definieren um beispielsweise lokal genutzte Systeme zu unterstützen, so muss die Konfiguration für beide Komponenten vorgenommen werden um die Kompatibilität zwischen ihnen zu sichern. Wichtigste Merkmale ist dabei ein eindeutiger Identifikator der Projektion der Form `Autorität:SRID`. Die Website http://epsg.io/ kann unter Umständen als Datenquelle herangezogen werden.

11.1.4.1 Konfiguration nutzerspezifischer Projektionen PostGIS

Die Tabelle `spatial_ref_sys` (i.d.R. im Datenbankschema `public`) enthält alle durch PostGIS-Funktionen nutzbare Projektionen. Eine Abfrage dieser Tabelle kann dazu dienen auf das Vorhandensein einer bestimmten Projektion zu schließen. Um eigene Definitionen hinzuzufügen ist es nötig hier einen neuen Eintrag zu tätigen. Für die Transformation von Koordinaten wird das Format Proj.4[16] (Tabellen-Spalte `proj4text`) genutzt.

Listing 11.1: Proj.4 Definition (Zeilenumbruch dient der Darstellung)

```
1 +proj=sterea
2   +lat_0=34.2 +lon_0=39.15 +k=0.999534
3   +x_0=281768.0448 +y_0=28076.0311 +a=6378249.2
4   +b=6356515 +units=m +no_defs
5   +towgs84=591.8,897.7,841.3,-10.9,-14.9,-22.6,-201.1
```

11.1.4.2 Konfiguration nutzerspezifischer Projektionen GeoServer

Die Definition von eigenen Projektionen erfolgt für GeoServer über die Datei `$GEOSERVER_DATA_DIR/user_projections/epsg.properies` im Well-known Text-Format (WKT)[17]. Bsp.:

[16] https://trac.osgeo.org/proj/wiki/GenParms

[17] http://webhelp.esri.com/arcgisserver/9.3/java/index.htm#geodatabases/the_ogc-607957855.htm

Listing 11.2: "Projektion in WKT-Schreibweise

```
PROJCS["DAI Libanon (Korrektion)",
  GEOGCS["unnamed ellipse",
    DATUM["unknown",
          SPHEROID["unnamed",
        6378249.2,
        293.4660212936265
      ],
      TOWGS84[
        591.8, 897.7, 841.3,
        -10.9, -14.9, -22.6, -201.1
      ]
    ],
    PRIMEM["Greenwich", 0],
    UNIT["degree", 0.0174532925199433]
  ],
  PROJECTION["Oblique_Stereographic"],
  PARAMETER["latitude_of_origin", 34.2],
  PARAMETER["central_meridian", 39.15],
  PARAMETER["scale_factor", 0.9995341],
  PARAMETER["false_easting", 281768.0448],
  PARAMETER["false_northing", 28076.0311],
  UNIT["Meter", 1]
]
```

11.1.5 OSGeo GeoServer Print Extension

Die Installation der Extensions zu GeoServer erfolgt durch ablegen der im Download verpackten JAR-Dateien im Verzeichnis `WEB-INF/lib` der GeoServer-Installation. Nach einem Neuladen des GeoServer Servlets stehen die entsprechenden Funktionen zur Verfügung. Die erfolgreiche Installation kann geprüft werden, indem der Dienst im Browser angesprochen wird. die Adresse `/geoserver/pdf/info.json` muss eine Konfiguration in JSON-erwidern.[18]

[18] http://gorse.informatik.tu-cottbus.de:8080/geoserver/pdf/info.json

11.2 Integration OpenInfRA

11.2.1 Geo-View

Eine wesentliche Herausforderung bei der Veröffentlichung von Projektdaten, insbesondere Themenausprägungen bzw. derer Themeninstanzen stellt die Diskrepanz zwischen den geforderten standardisierten Schnittstellen WMS/WFS und der eigentlichen Datenhaltung nach dem OpenInfRA-Datenmodell dar. Die OWS-Schnittstellen konsumieren sogenannte Simple Feature Types wie sie durch die OGC Simple Feature Access Spezifikation[19] definiert sind. Durch diese Spezifikation ist das Mapping von einfachen Datenstrukturen wie Tabellen o.ä. auf eine flache XML-Struktur vorgesehen. Insbesondere verschachtelte Attribute und komplexe Datentypen wie Wertelisten, wie sie im Datenbankschema von OpenInfRA definiert sind, werden nicht unterstützt.

Der durch den Anforderungskatalog geforderte OWS-Server GeoServer implementiert auch den Datenzugriff nach eben dieser Spezifikation und setzt dabei das einfachste der möglichen Simple Feature Profile (SF-0) um. [20]. Es besteht die Möglichkeit diese Beschränkung durch die Erweiterung `app-schema` [21] auszugleichen und insbesondere für die Vektordatenverarbeitung via WFS sogenannte komplexe Feature zu definieren. Dies hat allerdings erheblichen Einfluss auf die Nutzbarkeit der Schnittstelle mit externen Desktop-GIS-Systemen. Die Unterstützung dieser komplexen Feature ist derzeit insbesondere im OpenSource Umfeld und damit in der Zielgruppe des Informationssystems OpenInfRA nicht gegeben. Infolgedessen ist es notwendig die normalisierte Tabellenstruktur einer Themenausprägung nach dem im Grobkonzept beschriebenen generalisierten Datenbankschema auf eine simple Struktur zu überführen.

Ein weiteres Teilproblem stellt die fachliche Anforderung dar, beliebige Geometrietypen in beliebigen Projektionen zu speichern. Während dies über das generalisierte Datenbankschema und nicht zuletzt über den generischen PostGIS-Geometrietyp `geometry(GEOMETRY)` sehr gut umsetzbar ist, wird durch die Zugriffsschicht des GeoServers ein konkreter Geometrietyp und eine konkrete Projektion gefordert in der die Ein- und Ausgabe zu erfolgen hat. Dies macht es nötig schon auf Seite der Datenbank eine Transformation vorzunehmen. In der Regel muss hier davon ausgegangen werden, dass Themenausprägungen eines Projekts im selben Koordinatensystem aufgemessen wurden und den selben Geometrietyp darstellen.

[19] http://www.opengeospatial.org/standards/sfa
[20] http://docs.geoserver.org/stable/en/user/data/app-schema/complex-features.html
[21] http://docs.geoserver.org/stable/en/user/data/app-schema/index.html

In der aktuellen Implementierung wurden Datenbankfunktionen (`public.create_view`, `public.init_geom_views`) bereitgestellt, die entsprechende PostgreSQL-Views generieren, auf die GeoServer über einen PostGIS-Datenquelle als Grundlage eines Layers zugreifen kann. Hierbei wird die normalisierte und generalisierte Tabellenstruktur des OpenInfRA-Datenbankschemas auf einfache Struktur überführt. Einer jeden Themenausprägung die ein Geometrie-Attribut führt, wird dabei ein eigener View zugeordnet der als eigener *Feature Type* im Sinne der zuvor beschrieben Spezifikation begriffen werden kann.

```
1  SELECT
2    ti.id::text AS "id",
3    avv_1.value_1 AS "Beschreibung"
4    st_transform(st_multi(avg.geom), 42499) AS "the_geom"
5  FROM topic_instance ti
6  JOIN (
7    SELECT
8          attribute_value_geom.topic_instance_id,
9      attribute_value_geom.geom
10   FROM attribute_value_geom
11 ) avg ON avg.topic_instance_id = ti.id
12 LEFT JOIN (
13   SELECT
14     avv.value AS value_1,
15     avv.topic_instance_id
16   FROM attribute_value_value avv
17   WHERE avv.attribute_type_to_attribute_type_group_id = 'ti1'
18 ) avv_1 ON avv_1.topic_instance_id = ti.id
19 WHERE ti.topic_characteristic_id = 'tc1';
```

Um Schreiboperationen via WFS-T auf Themeninstanzen zuzulassen, müssen die bestehenden Datenbankviews um Trigger ergänzt werden, die entsprechende INSERT-, UPDATE- und DELETE-Queries umleiten.

11.2.2 Konzeptuelle Zuordnung zu Themenausprägungen

Für den Zugriff auf den Projektdatenbestand via WMS/WFS ist ein Mapping zwischen den Akteueren GeoServer und OpenInfRA-Kernsystem von Nöten. Die Semantik lässt sich dabei nach Tabelle 60 leicht übertragen.

Entsprechend der Definition in 60, wird für Projekte welche Themenausprägungen als WMS/WFS-Layer zur Verfügung stellen sollen, über die Admin-Oberfläche des

GeoServer	OpenInfRA
Arbeitsbereich	Projekt/Teilprojekt
Datenspeicher	Projektdatenbank
Layer	Themenausprägung
(Feature)	Themeninstanz

Tabelle 60: Konzeptuelles Mapping GeoServer-OpenInfRA

GeoServers ein Arbeitsbereich angelegt. Ein Beispiel hierfür ist in Abbildung 38 gegeben.

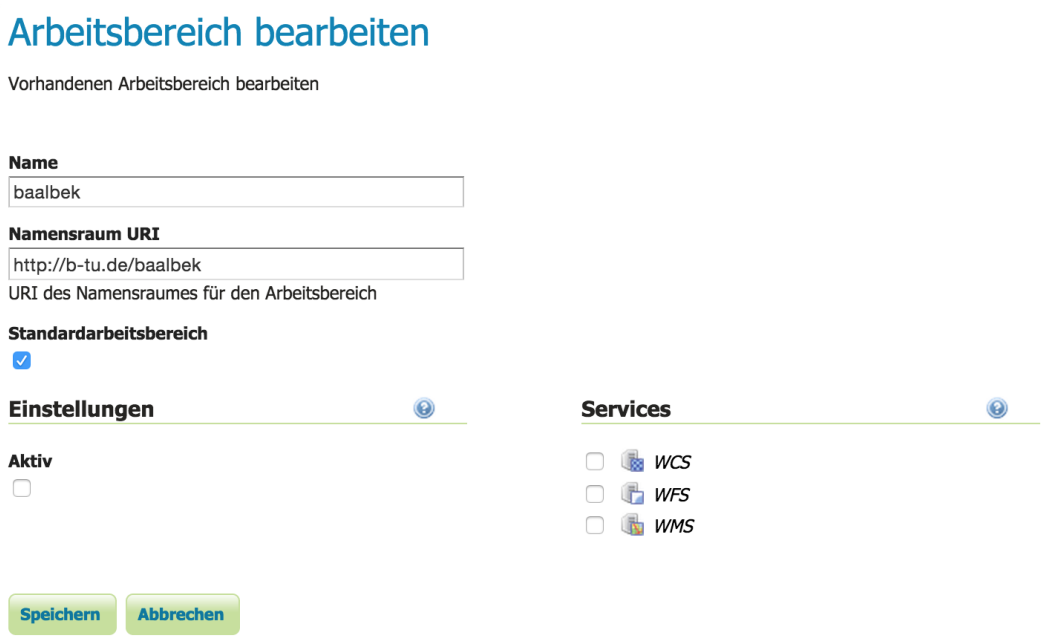
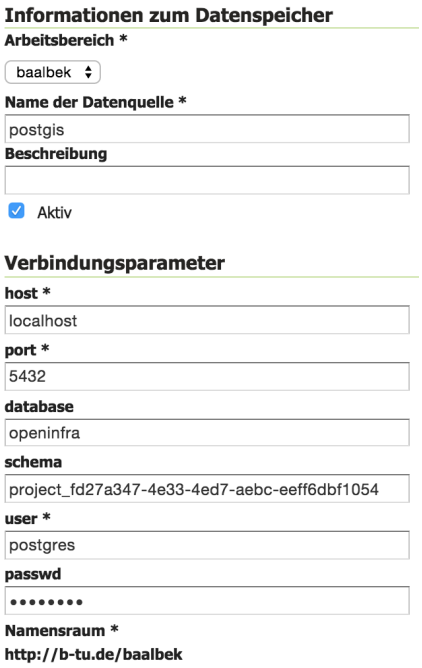


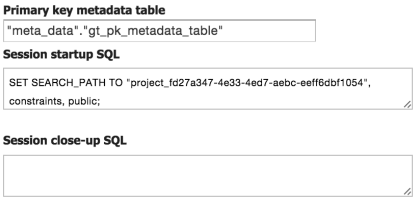
Abbildung 38: Anlegen eines Arbeitsbereichs im GeoServer für das Projekt Baalbek

Für diesen Arbeitsbereich wird anschließend eine Datenbankverbindung bzw. eine PostGIS-Datenquelle definiert, welche auf die PostgreSQL-Projektdaten zeigt (s. Abbildung 39a).

Bei erfolgreicher DB-Verbindung wird GeoServer alle lesbaren Datenbankobjekte auflisten, darunter auch die zuvor angelegten Geo-Views der Themenausprägungen. Diese können als Layer veröffentlich werden.



(a) Verbindungsparameter



(b) Parameter zur Metadatentabelle und zum setzen des korrekten `SEARCH_PATH` für DB-Verbindungen

Abbildung 39: Einrichten einer GeoServer-Vektordatenquelle für den Zugriff auf den OpenInfRA-Projektdatenbestand

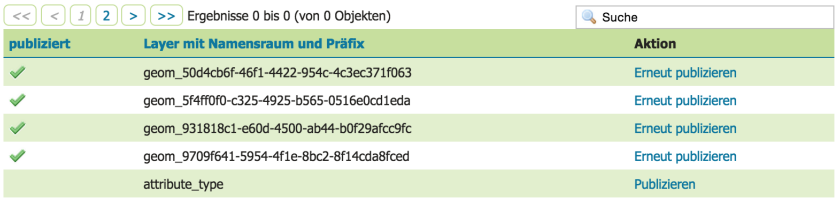


Abbildung 40: Liste der verfügbaren Datenbankobjekte (Tabellen, Views) einer PostGIS-Verbindung

Aus dieser Auflistung heraus können schließlich die verschiedenen Themenausprägungen auf Grundlage der zuvor erstellten Geo-Views veröffentlicht werden. Wichtig ist hier die Zuordnung eines maschinenlesbaren Namens (primär Name des Geo-Views), sowie der Vergabe eines nutzerfreundlichen Titels (i.d.R. die Bezeichnung der Themenausprägung) (s. Abbildung 41a). Darüber hnaus muss zwingend die räumliche Ausdehnung der Themenausprägung in der Projektion des Geo-Views erfolgen (s. Abbildung 41b). I.d.R. sollte dabei die Projektion des zugrundeliegenden Datenbankviews erkannt werden. Die konkrete Ausdehnung kann über die GeoServer-Oberfläche aus den bestehenden Themeninstanzen bzw. den bestehenden Geometrien des Views berechnet werden. Die Zuordnung erfolgt über den Reiter Publizieren (s. 41c).

11.2.3 Authentifizierung

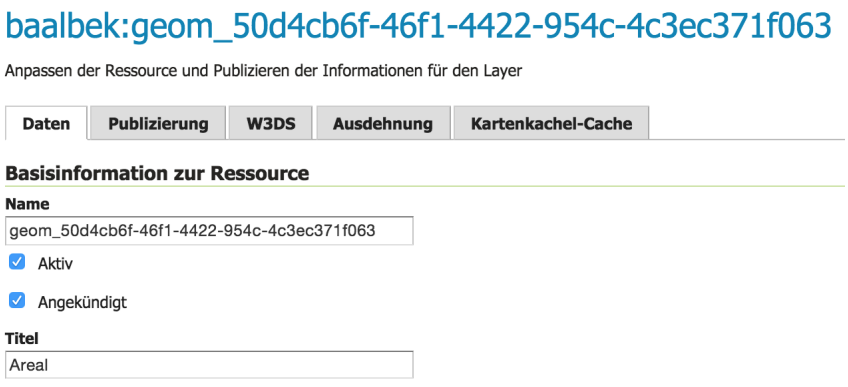
GeoServer nutzt ein eigenes Benutzer/Rollen/Rechte-System [22] um die angebotenen Ressourcen Web-Interface, OWS-Services und Rest-Interface zu sichern. Dabei wird zwischen den Subkomponenten *User/Group-Service* [23] und *Role-Service* [24] zur Authentifizierung und den einzelnen Security-Services zur Autorisierung für Ressourcen unterschieden. Alle Komponenten können dabei bei der Auswertung von Zugriffsrechten auf eine heterogene Menge an Datenquellen zugreifen. In der Standardkonfiguration kommen lediglich XML-Datein zum Einsatz, die im angegeben Datenverzeichnis hinterlegt sind. Alternativ ist aber auch die Nutzung einer Datenbankverbindung zur Auflösung von Zugriffsrechten möglich.

Benutzer stellen die unterste Ordnungseinheit des Kontrollsystems dar und beschreiben Menschen, Computer oder Software Systeme die einen Zugriff auf GeoServer-Dienste erhalten sollen. Ihnen zugeordnet sind in erster Linie Benutzername, Pass-

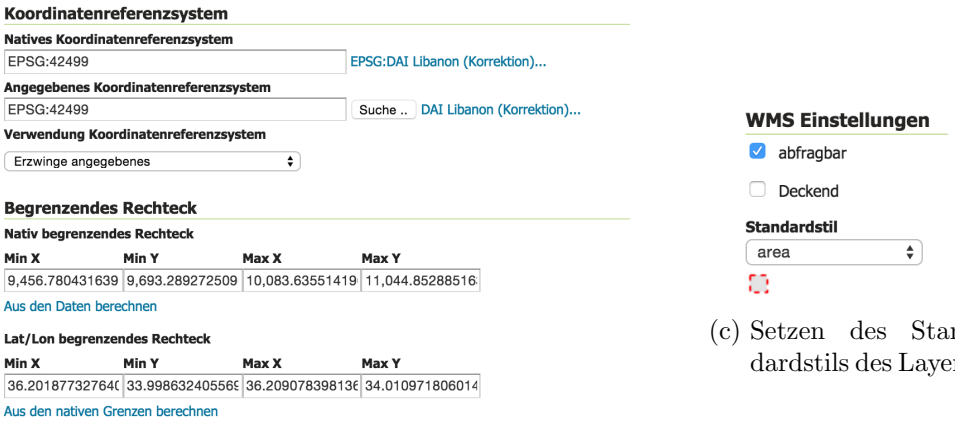
[22] http://docs.geoserver.org/latest/en/user/security/usergrouprole/index.html

[23] http://docs.geoserver.org/latest/en/user/security/usergrouprole/usergroupservices.html

[24] http://docs.geoserver.org/latest/en/user/security/usergrouprole/roleservices.html



(a) Name und Title festlegen



(c) Setzen des Standardstils des Layers

(b) Festlegen der räumlichen Ausdehnung der Themenausprägung

Abbildung 41: Einrichten einer Themenausprägung als WMS/WFS-Layer

wort und ein Statusflag ob der Benutzer für das System aktiviert ist. Benutzer können darüber hinaus in *Gruppen* organisiert werden. Die Zuordnung konkreter Rechte erfolgt über sogenannte *Rollen*. GeoServer-Rollen unterstützen Vererbung, d.h. Zugriffsrechte die einer Eltern-Rolle zugeordnet sind werden durch Kind-Rollen geerbt. Folgende Standardrollen sind reserviert:

ROLE_ADMINISTRATOR Zugriff auf alle Operationen und Ressourcen

ROLE_GROUP_ADMIN Administration von Benutzer-Gruppen

ROLE_AUTHENTICATED Standardrolle aller authentifizierter Nutzer

ROLE_ANONYMOUS Standardrolle aller nicht authentifizierter Nutzer bei aktivierter Anonymer Authentifizierung

Bis auf die optionale Ordnungseinheit der Benutzer-Gruppen entspricht dieser Authentifizierungsmechanismus der konzeptuellen Beschreibung für OpenInfRA. D.h., dass insbesondere über das OpenInfRA-Kernsystem definierte Benutzer direkt über eine Datenbankanbindung zum Einsatz kommen können. Das impliziert die Einrichtung eines JDBC-User/Group-Services[25] über GeoServer der auf die internen Datenstrukturen des OpenInfRA-Rollen/Rechte-Systems zugreift. Der Zugriff kann dabei auf lesende Operationen beschränkt sein, um die komplexe Administration über das OpenInfRA-Frontend zu gewährleisten. Das bedeutet auch, dass der User/Group-Services nicht direkt auf die entsprechenden Tabellen des OpenInfRA-Datenbankschemas zugreifen muss, sondern über entsprechend angepasste Datenbank-Views Zugriff erhalten kann. Dies ist insbesondere wichtig, wenn das Datenbankschema zum Persistieren von OpenInfRA-Nutzern sich vom durch GeoServer erwarteten User/Group-Service-Schema unterscheidet. In diesem Fall muss das folgende Schema nach Tabelle 61-64 zur Verfügung stehen. Analog gilt dies für den Role-Service für den auf die Tabellen 65-68 gemappt werden sollte. Die Auflösung von Nutzer und Rollen erfolgt schließlich nach dem in 42 dargestellten Schema (s.a. [26]). Neben der Möglichkeit Nutzer so bei Interaktion mit den einzelnen Layer (Themenausprägungen) zu authentifizieren, können so auch OpenInfRA-Administratoren Zugriff auf die GeoServer-Oberfläche erhalten.

11.2.4 Autorisierung

GeoServer unterstützt zwei Ansätze bei der Autorisierung von Zugriffen die sich grundlegend in der möglichen Granularität unterscheiden. Zum einen ist es möglich

[25] http://docs.geoserver.org/stable/en/user/webadmin/security/ugr.html#webadmin-sec-usergroupservices

[26] http://docs.geoserver.org/latest/en/user/security/usergrouprole/interaction.html

Feld	Typ	Null	Key
name	varchar(128)	NO	PRI
password	varchar(254)	YES	
enabled	char(1)	NO	

Tabelle 61: Tabelle: users

Feld	Typ	Null	Key
username	varchar(128)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Tabelle 62: Tabelle: user_props

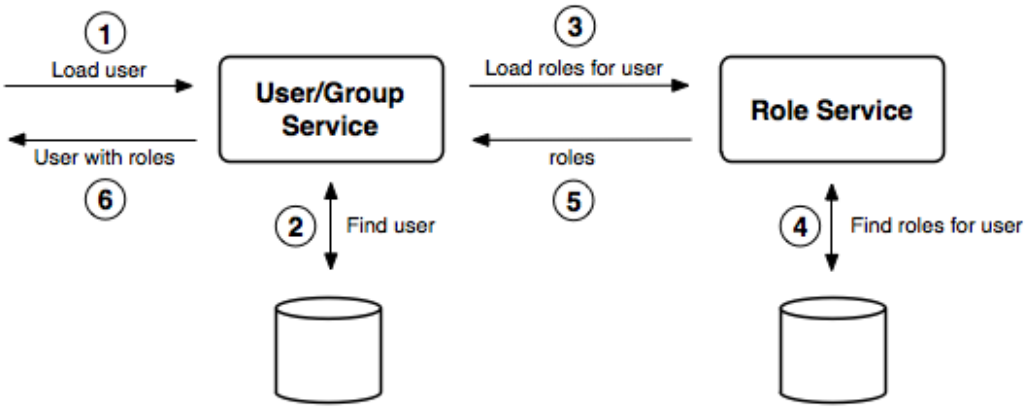


Abbildung 42: Interaktion zwischen User/Group- und Role-Service

anhand der nachgewiesenen Rolle den Zugriff auf ganze Dienste (OWS, REST) einzuschränken (*Service Security* [27]), zum anderen kann eine feinere Unterscheidung auf höhe konkreter Ressourcen umgesetzt werden (*Layer-Security* [28]). Letztere ist im Kontext des in 60 beschriebenen Mappings von OpenInfRA-Objekten auf GeoServer-Ressourcen besser geeignet, da so beispielsweise der Zugriff auf einen Layer (≙ OpenInfRA-Themenausprägung) für bestimmte Rollen eingeschränkt werden kann.

[27] http://docs.geoserver.org/latest/en/user/security/service.html
[28] http://docs.geoserver.org/latest/en/user/security/layer.html#sec-layer

Feld	Typ	Null	Key
name	varchar(128)	NO	PRI
enabled	char(1)	NO	

Tabelle 63: Tabelle: groups

Feld	Typ	Null	Key
groupname	varchar(128)	NO	PRI
username	varchar(128)	NO	PRI

Tabelle 64: Tabelle: group_members

Die Festlegung der Zugriffsrechte erfolgt im Gegensatz zur Definition von Nutzer und Rollen über Java-Properties-Dateien die im GeoServer-Datenverzeichnis hinterlegt sind. Über die Datei `$GEOSERVER_DATA_DIR/security/layers.properties` erfolgt so die Zuordnung von Ressourcen (Workspaces & Layer) mit bestimmten Freigaben (*a*dministrate, *w*rite, *r*ead) zu den in Kapitel 11.2.3 beschriebenen Rollen. Die hierfür anzuwendende Syntax ist der Dokumentation [29] zu entnehmen und ist im folgenden zur Veranschaulichung dargestellt:

```
1 # workspace.layer.permission=role[,role2,...]
2
3 *.*.r=*
4 *.*.w=NO_ONE
5 private.*.r=TRUSTED_ROLE
6 private.*.w=TRUSTED_ROLE
7 topp.congress_district.w=STATE_LEGISLATORS
```

Alternativ und bisweilen einfacher kann die Konfiguration durch Administratoren über das Web-Interface des GeoServers erfolgen.

11.3 WebGIS-Client

Der WebGIS-Client ist eine eigenständige JavaScript-Anwendung die in beliebige Webanwendungen eingebunden werden kann. Die Gestalt und der Funktionsumfang kann dabei zur Laufzeit bestimmt werden. Die Konfiguration erfolgt im JSON-Format die leicht durch Anwendungslogik generiert werden kann.

[29] http://docs.geoserver.org/latest/en/user/security/layer.html#rules

Feld	Typ	Null	Key
name	varchar(64)	NO	PRI
parent	varchar(64)	YES	

Tabelle 65: Tabelle: roles

Feld	Typ	Null	Key
rolename	varchar(64)	NO	PRI
propname	varchar(64)	NO	PRI
propvalue	varchar(2048)	YES	

Tabelle 66: Tabelle: role_props

Für die Umsetzung des Clients kommen dabei die folgenden Komponenten zum Einsatz:

Sencha ExtJS Ein weit verbreitetes JavaScript-Framework welches UI-Komponenten und eine Datenabstraktionsschicht zur Verfügung stellt um so die Entwicklung komplexer Oberflächen zu vereinfachen. Das Framework ist dual lizensiert und kann im Rahmen der Entwicklung für OpenInfRA unter der GPL3-Lizenz genutzt werden. Zum Einsatz kommt Version 4.2.1 [30].

DeftJS Eine leichtgewichtige Erweiterung zu ExtJS die es ermöglicht nach dem Dependency Injection-Entwurfsmuster (DI) mit ExtJS-Klassen zu interagieren und so hilft die Abhängigkeiten zwischen den Klassen zu minimieren. Das Paket ist unter der MIT-Lizenz nutzbar. Zum Einsatz kommt Version 0.9 [31].

GeoExt Eine Erweiterung die basierend auf dem Mapping Framework OpenLayers ExtJS um räumliche Komponenten erweitert und so die Verbindung zwischen klassischen GUI-Elementen wie Tabellen, Listen und Popups und der OpenLayers Karte herstellt. Die Erweiterung ist unter der BSD-Lizenz nutzbar. Zum Einsatz kommt Version 2.0.3[32].

OpenLayers OpenLayers ist ein JavaScript-Mapping Framework zur Darstellung interaktiver Karten im Browser. Als Datengrundlage sind eine Vielzahl von Diensten geeignet. Das Framework ist unter der BSD-Lizenz nutzbar. Zum Einsatz kommt Version 2.13.1[33].

[30] http://docs.sencha.com/extjs/4.2.1/
[31] http://deftjs.org/
[32] http://geoext.github.io/geoext2/
[33] http://openlayers.org/two/

Feld	Typ	Null	Key
username	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

Tabelle 67: Tabelle: user_roles

Feld	Typ	Null	Key
groupname	varchar(128)	NO	PRI
rolename	varchar(64)	NO	PRI

Tabelle 68: Tabelle: group_roles

Proj4js Ermöglicht die Transformation von Geometrie zwischen beliebigen Kartenprojektionen. OpenLayers unterstützt dies lediglich für die Projektionen EPSG:4326 (WGS84) und EPSG:900913 (Mercator), sollen weitere unterstützt werden ist der Einsatz von Proj4js zwingend erforderlich.[34]

Q Eine kleine Bibliothek welche die Nutzung der für ECMAScript 2015 vorgesehen JavaScript-Promises[35] umsetzt, und es so ermöglicht asynchrone Anfragen leichter zu strukturieren.[36]

11.3.1 Einbinden der Anwendung

Im Grunde stellt die Entwicklung eine weitere Bibliothek dar, die es erlaubt einen Client dynamisch zur Laufzeit zu generieren. Dafür wird eine Konfiguration im JSON-Format interpretiert welche neben den initial zur Verfügung stehenden Layer und Services eben auch Informationen zu den zur Verfügung stehenden Oberflächenelementen enthält. Der Umfang der Anwendung und ihre Gestalt kann so bei jedem Aufruf, etwa für jeden Nutzer individuell angepasst werden. Die Darstellung 11.3 gibt eine minimale Konfiguration vor, welche initial lediglich einen Layer auf der Karte anzeigt und wenige Tools in einer Toolbar auflistet. Details zur Konfiguration können der Online-Dokumentation entnommen werden[37].

[34] http://proj4js.org/
[35] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Promise
[36] http://documentup.com/kriskowal/q/
[37] https://github.com/OpenInfRA/GXC

Listing 11.3: Dynamisch geladene WebGIS-Konfiguration

```
/**
 * Globale Variable die beim Laden der Anwendung
 * als Konfiguration der Kartenanwendung interpretiert wird.
 */
var GXC_ENV = {
  proxy: {
    host: 'http://localhost:8081/proxy/whitelist.jsp?'
  },
  geoserver: {
    host: 'http://localhost:8081/geoserver'
  },
  targetId: 'gxc-container',
  viewportItems: [{
    region: 'center',
    xtype: 'gxc_panel_map',
    tbar: [{
      xtype: 'gxc_button_zoomin'
    }, {
      xtype: 'gxc_button_zoomout'
    }]
  }],
  projections: {
    'EPSG:4326': true,
  },
  mapOptions: {
    projection: 'EPSG:4326',
    center: [13.75, 51.05],
    minScale: 10,
    maxScale: 150000000,
    zoom: 6
  },
  layers: [{
    url: 'http://ows.terrestris.de/osm/service?',
    type: 'WMS',
    version: '1.1.1',
    layer: 'OSM-WMS'
  }],
  services: []
};
```

Das Einbinden der Anwendung erfolgt über das Setting `targetId` welches einen DOM-Knoten des HTML-Dokuments angibt, in die die Anwendung geladen werden soll. Alle dynamisch generierten Elemente werden in diesen Container eingesetzt. Um die Größe des der Anwendung zu bestimmen, kann dieses DOM-Element mit einfachen CSS-Regeln gestylt werden. Bei der Nutzung des Fullscreen-Tools zum Anzeigen der Kartenanwendung im Vollbildmodus müssen entsprechende CSS-Regeln ergänzt werden. Listing 11.4 gibt ein einfaches Beispiel für ein Stylesheet vor, bei dem die texttttargetId gxc-container genutzt wird.

Listing 11.4: WebGIS Stylesheet

```
1  /* Groesze der Kartenanwendung */
2  #gxc-container {
3      height: 600px;
4      width: 960px;
5  }
6
7  /* Skalieren der Anwendung im Vollbildmodus */
8  #gxc-container:-webkit-full-screen {
9          width: 100%;
10         height: 100%;
11 }
12 #gxc-container:-moz-full-screen {
13         width: 100%;
14         height: 100%;
15 }
16 #gxc-container:-ms-fullscreen {
17         width: 100%;
18         height: 100%;
19 }
20 #gxc-container:fullscreen {
21         width: 100%;
22         height: 100%;
23 }
```

Zur Verfügung stehende viewportItems sind der Dokumentation[38] der Entwicklung zu entnehmen.

[38] https://github.com/OpenInfRA/GXC#viewportitems

12 Geo Document Service (GDS)

Der Geo Document Service ist durch die Erweiterung MapFish Print umgesetzt. Die Installation ist in . Alternativ ist es möglich das MapFish Print Servlet als eigenständiges WAR-Archiv im Servlet Container zu laden (s.a. [1]).

12.0.2 Konfiguration Druckvorlagen

Die Konfiguration erfolgt serverseitig im `YAML`-Format. Diese wird im Verzeichnis `$GEOSERVER-DATA-DIR/printing/config.yaml` hinterlegt. Eine genaue Beschreibung der möglichen Parameter kann der Dokumentation[2] zu MapFish-Print entnommen werden.

Ein grundlegendes Beispiel für den Inhalt der Konfigurationsdatei ist in 12.1 gegeben. Im bereitgestellten $GEOSERVER-DATA-DIR ist darüber hinaus eine Konfiguration mit zwei Layouts enthalten.

Listing 12.1: Generalle Struktur MapFish-Print-Konfiguration

```
1  # Aufloesungen
2  dpis: [75, 150, 300]
3
4  # Unterstuetzte Ausgabeformate
5  formats:
6    - pdf
7    - png
8
9  # Kartenskalen
10 scales:
11   - 50
12   - 500
13   - 1500
14
15 # Erlaubte Host-Adressen
16 hosts:
```

[1] http://www.mapfish.org/doc/print/installation.html
[2] http://www.mapfish.org/doc/print/configuration.html

```
17    - !localMatch
18      dummy: true
19    - !dnsMatch
20      host: terraservice.net
21      port: 80
22  TestLayout:
23    mainPage:
24      pageSize: A4
25      rotation: true
26      landscape: false
27      items:
28          # Karte
29          - !map
30            width: 600
31            height: 400
```

13 Synchronisation

Die durch OpenInfRA unterstützte Offline Nutzung erfordert eine Synchronisation zwischen verschiedenen Instanzen. Diese Synchronisation betrifft sowohl Datenbankinhalte, als auch Dateien im Dateisystem. Ein geeignetes Werkzeug dazu ist die Software SymmetricDS, deren Einsatzmöglichkeit für OpenInfRA bereits Prototypisch nachgewiesen wurde ([Ben13]).

Die folgende Auflistung soll eine Übersicht über zu synchronisierende Daten liefern.

- Fachdatenbank
- Dateien und Bilder
- Benutzer- / Rollen- / Rechte für synchronisierte Fachdaten
- Einstellungen
- Protokolle?

Anmerkungen zu SymmetricDS

- Kann sowohl als WAR bereitgestellt, oder direkt eingebettet verwendet werden, wobei die Einbettung sinnvoller erscheint.
- Die Konfiguration erfolgt über SymmetricDS interne Datenbanktabellen. Diese würde, wie alle anderen Ressourcen auch, per REST ansprechbar sein. Somit kann die Synchronisationskonfiguration direkt in die GUI eingebettet werden.
- Da die Synchronisation auf unterschiedlichen Betriebssystemen funktionieren soll, muss besonders auf die Pfadangaben für das Dateisystem geachtet werden!
- Seiteneffekte sind noch nicht abzusehen und könnten u. a. durch die Synchronisationstrigger und die Integritätsbedingungstrigger entstehen.

Eine sehr umfangreiche Erläuterung zu diesem Thema findet sich unter: https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

14 Implementierung

In diesem Kapitel wird die konkrete Umsetzung von OpenInfRA beschrieben.

14.1 Programmiersprache

Als konkrete Programmiersprache wird Java verwendet und als Java-Compiler ist die Version 1.7 vorgesehen[1]. Das JDK 7 wird hier verwendet, weil die Installation neuerer Versionen von Java unter Ubuntu noch nicht standardisiert über die Repositories möglich ist.

14.2 Entwicklungsumgebung

Als Entwicklungsumgebung wird *Eclipse IDE for Java EE Developers* verwendet[2]. Prinzipiell kann aber auch jede andere Entwicklungsumgebung verwendet werden.

14.3 Versionskontrolle

Für die Versionskontrolle wird aktuell Apache Subversion[3] eingesetzt und durch Lehrstuhl DBIS an der BTU Cottbus – Senftenberg unter dem folgenden Link bereitgestellt:

- https://saffron.informatik.tu-cottbus.de/repos/OpenInfRA_Backend/

Dieses Repository enthält die drei Ordner: *trunk*, *branches* und *tags*[4]. Im Ordner 'trunk' wird der Hauptzweig der Implementierung geführt. Der Ordner 'branches' soll Kopien vom Trunk aufnehmen können. Letztlich soll der Ordner 'tags' die einzelnen Releases aufnehmen.

[1] http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

[2] https://eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/lunar

[3] https://subversion.apache.org/

[4] http://svnbook.red-bean.com/de/1.6/svn.branchmerge.maint.html

Als SVN-Software für die Arbeit im Windows Explorer eignet sich TortoiseSVN[5] sehr gut. Als SVN-Software in Eclipse eignet sich Subclipse[6] sehr gut.

14.4 Dependency-Management

Für das Dependency-Management eignen sich Apache Ant[7], Apache Maven[8] und Gradle[9]. Im Internet[10,11] finden sich einige Diskussionen über den Einsatz dieser Dependency-Management-Systeme.

Für OpenInfRA wird zunächst Apache Maven als Dependency-Management-System eingesetzt, weil es weit verbreitet ist, in den ersten Tests die notwendigen Funktionalitäten bereit gestellt hat und sich direkt in die gewählte Entwicklungsumgebung integriert. Dennoch kann es sinnvoll sein, dass im Zuge der weiteren Entwicklung von OpenInfRA ein anderes Dependency-Management-System (z.B. Gradle) verwendet wird.

14.5 Plug-in-Konzepte

Für die Umsetzung der Plug-in-Konzepte bieten sich folgende Frameworks an: *Open Services Gateway initiative (OSGi)*[12], das Java Simple Plugin Framework (JSPF)[13] und Service Provider Interfaces (SPIs)[14].

OSGi ist eine dynamische Softwareplattform, die standardisiert und sehr umfangreich ist. OSGi geht über den allgemeinen Anwendungsfall eines Plug-in-Frameworks weit hinaus. Daher kann die Anwendung dieser Softwareplattform in Bezug auf bestimmte Funktionalitäten sehr komplex sein und ein zielorientiertes Arbeiten in einigen Situationen verhindern. Zusätzlich müssen alle notwendigen Komponenten diese Softwareplattform unterstützen, was z.B. durch die geplante Datenbank-

[5] http://tortoisesvn.net/index.de.html
[6] http://subclipse.tigris.org/servlets/ProjectProcess?pageID=p4wYuA
[7] http://ant.apache.org/
[8] https://maven.apache.org/
[9] https://gradle.org/
[10] http://www.drdobbs.com/jvm/why-build-your-java-projects-with-gradle/240168608?pgno=1
[11] http://technologyconversations.com/2014/06/18/build-tools/
[12] http://www.osgi.org/Main/HomePage
[13] https://code.google.com/p/jspf/
[14] https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html

Replikationssoftware SymmetricDS[15] nicht direkt gegeben ist. Aus den genannten Gründen wird OSGi für OpenInfRA nicht genutzt.

Das JSPF ist ein kleines eigenständiges Projekt, welches Plug-in-Funktionalitäten für kleinere bis mittlere Projekte verfügbar machen will. Dieses Projekt wird nicht mehr aktiv weiter entwickelt und wird daher für OpenInfRA nicht verwendet.

Das SPI ist ein sehr leichtgewichtiger Bestandteil des Java-Standards. Für die Umsetzung eines Plug-in-Konzepts in OpenInfRA bietet das SPI alle notwendigen Voraussetzungen und ermöglicht in Zukunft, auf Grund der Einfachheit, eine zielorientierte Weiterentwicklung. Aus dem genannten Grund wird es als Plug-in-Konzept für OpenInfRA verwendet.

14.6 Objekt-Relationales-Mapping

Für das Objekt-Relationales-Mapping (ORM) können unter anderem folgende Techniken für die Umsetzung von OpenInfRA eingesetzt werden: SQL-Ansatz, JOOQ[16] und Java Persistence API (JPA)[17]. Diese Techniken werden im Folgenden näher erläutert.

Beim reinen SQL-Ansatz werden die SQL-Anfragen direkt ohne die Verwendung eines Frameworks implementiert und an die Datenbank geschickt. Dieser Ansatz hat einige Nachteile: zunächst ergibt sich ein sehr hoher Aufwand für die Formulierung der SQL-Anfragen, weiterhin muss mit einem erhöhten Aufwand für die Wartung der SQL-Anfragen gerechnet werden (z.B. bei Änderungen am Datenbankschema) und letztlich ist es notwendig, Caching-Mechanismen selbstständig zu implementieren. Aus den genannten Gründen wird dieser Ansatz für OpenInfRA nicht gewählt.

Durch den Einsatz von JOOQ wird ein objektorientierter Dialekt generiert, welcher die Formulierung von Anfragen an die Datenbank anhand der in Java üblichen Notation erlaubt. Somit müssen die Datenbankanfragen trotzdem konkret formuliert und an die Datenbank gesendet werden. Zusätzlich ist nicht genau klar, wie Caching-Strategien und Connection-Pooling im Umgang mit JOOQ sinnvoll umgesetzt werden können. Außerdem wurde bei den durchgeführten Geschwindigkeitstests festgestellt, dass selbst einfachste Datenbankanfragen durch den Einsatz von JOOQ äußerst langsam sind. Auch in der Dokumentation[18] wird bereits erläutert, dass JOOQ kein leichtgewichtiges Werkzeug ist.

[15] http://www.symmetricds.org/
[16] http://www.jooq.org/
[17] http://de.wikipedia.org/wiki/Java_Persistence_API
[18] http://www.jooq.org/doc/3.5/manual/sql-execution/performance-considerations/

Die JPA ist Bestandteil des Java-Standards und definiert die Rahmenbedingungen für die Umsetzung einer vollständigen Datenbankzugriffsschicht. Für die Verwendung in OpenInfRA wurden drei Frameworks untersucht, die JPA implementieren: Spring Hibernate[19], Apache OpenJPA[20] und EclipseLink[21]. Für die Implementierung von OpenInfRA wird EclipseLink ausgewählt, weil es die Referenzimplementierung von JPA ist und im Gegensatz zu Spring Hibernate deutlich schneller ist und im Gegensatz zu OpenJPA einer größeren Beliebtheit in der Community erfreut. Außerdem gliedert sich die Verwendung von EclipseLink direkt in die gewählte Entwicklungsumgebung ein und das Erzeugen der Datenbankzugriffsschicht ist direkt aus Eclipse möglich. Diese erzeugten Quellen können direkt durch EclipseLink verwendet werden.

14.7 REST-API

Für die Umsetzung der REST-API kann hier Jersey[22] eingesetzt werden. Jersey ist die Referenzimplementierung des Java-REST-Standards (JAX-RS[23]), ist sehr performant, wird aktiv weiterentwickelt, hat eine sehr umfangreiche Dokumentation und erfreut sich einer sehr großen Beliebtheit in der Community. Aus den genannten Gründen wurde für die Umsetzung der REST-API keine weiteren Frameworks verglichen.

14.8 Webserver

Als Webserver kann prinzipiell jeder Webserver eingesetzt werden, der einen Java-Servelet-Container besitzt wie z.B. Jetty oder Apache Tomcat. Als Webserver wird hier Apache Tomcat in der Version 7.0[24] eingesetzt. Diese Version wird hier verwendet, weil die Installation neuerer Versionen unter Ubuntu noch nicht standardisiert über die Repositories möglich ist.

[19] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/orm.html
[20] http://openjpa.apache.org/
[21] http://eclipse.org/eclipselink/
[22] https://jersey.java.net/
[23] https://jax-rs-spec.java.net/
[24] https://tomcat.apache.org/download-70.cgi

14.9 Connection-Pooling

Für das Connection-Pooling werden hier die Funktionalitäten von EclipseLink verwendet[25]. Dies ist aber nur sinnvoll solange die OpenInfRA-Anwendung alleine im Tomcat läuft. Im Zuge der Weiterentwicklung kann es aber auch notwendig werden, dass der Tomcat für das Connection-Pooling eingesetzt werden muss (z.B. Anwendungsübergreifendes Connection-Pooling bzw. wenn mehr als eine Webanwendung im Tomcat läuft).

14.10 Caching-Strategien

Die Umsetzung der Caching-Strategien muss auf unterschiedlichen Ebenen betrachtet werden. Zunächst muss die Datenbankzugriffsschicht entsprechende Mechanismen bereitstellen. Weiterhin müssen entsprechende Mechanismen auch für die REST-Schnittstelle zur Verfügung stehen. Durch die Verwendung von EclipseLink enthält die Datenbankzugriffsschicht bereits entsprechende Caching-Strategien[26], die nach den entsprechenden Bedürfnissen individuell angepasst werden können. Durch die Verwendung von Jersey werden bereits ausreichende Caching-Strategien[27] für die REST-Schnittstelle angeboten.

14.11 Übersicht der eingesetzten Techniken

Die in der Tabelle 69 aufgelisteten Techniken sollen in OpenInfRA eingesetzt werden.

14.12 Modifikation der Migrationdaten

Durch die Implementierung der REST-Schnittstelle, fiel eine Problematik bezüglich der Vergabe von *pt_free_text_ids* auf, welche vorher nicht im Detail durchdacht wurde. Zunächst wurde angenommen, dass eine Zeichenkette, welche in der Relation *localized_character_string* gespeichert wird, in unterschiedlichen Zusammenhängen verwendet werden kann. So könnte die Zeichenkette als Name eines Projektes auftreten, als auch als Name eines Attributtypen. In beiden Fällen, würde auf diesel-

[25] http://eclipse.org/eclipselink/documentation/2.4/jpa/extensions/p_connection_pool.htm

[26] http://eclipse.org/eclipselink/documentation/2.5/jpa/extensions/a_cache.htm

[27] https://devcenter.heroku.com/articles/jax-rs-http-caching

Bereich	Technik	Version
Programmiersprache	Java	1.7 (JDK 7)
Entwicklungsumgebung	Eclipse EE	4.4 (Luna)
Dependency-Management	Maven	3.0.4 (Eclipse embedded)
Plug-in-Konzepte	SPI	- (Java Bestandteil)
Objekt-Relationales-Mapping	JPA/EclipseLink	2.6.0
REST-API	Jersey	2.17
Webserver	Apache Tomcat	7.0

Tabelle 69: Übersicht der eingesetzten Techniken

be *pt_free_text_id* verwiesen werden. Der Ursprüngliche Gedanke war, falls in einer der beiden Verwendungszwecke der Name geändert wird, sollte er sich nicht auf den jeweils anderen Verwendungszweck auswirken. Dabei hätte also geprüft werden müssen, ob der entsprechende *pt_free_text_id* mehrfach verwendet wird. Wäre das der Fall gewesen, hätte ein neuer Eintrag in *localized_character_string* erzeugt werden müssen, wobei die neu erstellte *pt_free_text_id* in das zu ändernde Feld hätte eingetragen werden müssen. Selbiges hätte für den umgekehrten Weg gegolten. Wenn ein *localized_character_string* verwendet werden sollte, der bereits vorhanden war, sollte eben dieser genutzt werden, anstatt ihn erneut mit einer anderen *pt_free_text_id* zu erzeugen.

Nun trat folgendes Problem auf. Angenommen es existieren zwei Projekte A und B mit dem deutschen Namen *Bank*. Nun soll der Name von Projekt A auf Englisch übersetzt werden. Die Übersetzung fällt auf den Begriff *bank*, da es sich um ein Geldinstitut handelt. Damit erhält automatisch Projekt B ebenfalls diese Übersetzung, obwohl es sich bei Projekt B um die Sitzgelegenheit handelt, die korrekte Übersetzung also *bench* wäre. Fortan ist es nicht mehr möglich, die Übersetzung von Projekt A oder B unabhängig voneinander zu ändern. Einzige Möglichkeit wäre, wenn eines der Projekte einen anderen deutschen Namen erhält. Um dieses Problem zu umgehen, werden Zeichenketten und deren Übersetzungen zu logischen Einheiten zusammengefasst. Im o.g. Fall würde der Begriff *Bank* mit einem deutschen *pt_locale* zweimal in *localized_character_string* vorkommen, einer für Projekt A und einer für Projekt B. Somit wird eine unabhängige Lokalisierung gewährleistet. Durch die redundante Datenhaltung werden wesentlich mehr Einträge generiert (Baalbek: von 32.687 auf 129.035 Einträge). Andererseits werden dadurch auch Vergleichsoperationen vermieden, welche ausgeführt werden müssten um die oben angesprochene Mehrfachverwendung zu ermitteln.

Für die Anpassung der bereits vorhandenen Migrationsdaten wurde ein Skript erstellt, welches auf dem Datenbestand ausgeführt wird und die Redundanzen erzeugt. Dieses Skript benötigt eine relativ lange Ausführungszeit und ist unter der folgenden URL zu finden:

https://saffron.informatik.tu-cottbus.de/repos/OpenInfRA_Datenbank/schema/trunk/public_functions.sql

14.13 Installation der Datenbank

Für eine korrekte Arbeitsweise über verschiedene Zeitzonen hinaus wird dringend empfohlen, die Standard-Zeitzone der Datenbank auf die koordinierten Weltzeit (UTC) einzustellen. Dies muss über die PostgreSQL Konfigurationsdatei erfolgen.

Für die Installation der Fachdatenbank stehen SQL-Skripte zur Verfügung. Diese können entweder einzeln, oder mit Hilfe von Batch-Dateien installiert werden. Hier soll ausschließlich auf die Installation mit den vorhandenen Batch-Dateien eingegangen werden. Für Informationen zur manuellen Installation, sollte direkt in die Batch-Dateien geschaut werden.

Für die automatische Installation der Datenbank stehen Skripte für Windows und Linux bereit. Initial werden bei beiden Skripten der Benutzername *postgres*, das Passwort *postgres*, die Datenbank *openinfra* und unter Windows eine Pfadvariable für die Ausführung von *psql* als Standard festgelegt. Durch die Angabe von Parametern können die Datenbank und der Benutzername verändert werden. Diese Skripte umfassen derzeit das erstellen der Datenbank mit den notwendigen Erweiterungen, die Integritätsbedingungen, das Systemschema mit Daten, das Schema für die Metadaten und die Projektdatenbanken Baalbek, Palatin und ein Test-Projekt. Weitere Informationen dazu finden sich in den entsprechenden Skripten.

- Windows: https://saffron.informatik.tu-cottbus.de/repos/OpenInfRA_Datenbank/installAll.bat
- Linux: https://saffron.informatik.tu-cottbus.de/repos/OpenInfRA_Datenbank/installAll.sh

14.14 Erzeugen der UUIDs

Die notwendigen UUIDs werden ausschließlich durch die Anwendung, d.h. in der Datenbank oder durch die Anwendungsschicht, generiert. Die Erzeugung und Übergabe

von UUIDs von außerhalb ist nicht möglich. Im Falle des Aufrufs einer HTTP-POST-Methode wird immer die durch die Anwendung erzeugte UUID zurückgeliefert.

14.15 Zeitzonen

Da OpenInfRA auf verschiedenen Betriebssystemen bereitgestellt werden muss, liegt die Nutzung des, mit der Java SE Plattform mitgelieferten, Zeitzonen-Repository ([Ora15]) nahe.

In der Datenbank werden Zeitstempel stets mit koordinierten Weltzeit (UTC) hinterlegt. Dieser Zeitstempel wird auch stets über die REST-Schnittstellen ausgeliefert. Für die korrekte Darstellung in der Client-Zeitzone ist der Client selbst verantwortlich.

14.16 Vererbung der Attributwerte

Die Implementierung der Fachdatenbank beinhaltet eine Vererbungshierarchie bei *attribute_value*. Der Grund, warum diese Vererbung genutzt wird und nicht einfach nur die vier Tabellen *attribute_value_value*, *attribute_value_domain*, *attribute_value_geom* und *attribute_value_geomz* einzeln verwendet werden, liegt an den Integritätsbedingungen. Es gibt z. Bsp. die Bedingung, dass bestimmte Teile eines Attributtyps nicht mehr verändert werden dürfen, wenn dieser von einem Attributwert genutzt wird. Wenn es die Vererbung nicht geben würde, müsste in jeder der 4 o. g. Tabellen einzeln nachgeschaut werden, ob der Attributtyp verwendet wird. Im ungünstigsten Fall, müssen alle 4 Tabellen durchsucht werden, um festzustellen, dass der gesuchte Attributtyp nicht verwendet wird. Andersherum ist es ohne viel Aufwand möglich herauszufinden, in welcher erbenden Tabelle eine Id vorkommt, wenn sie in *attribute_value* gefunden wurde:

```
1 SELECT p.relname, a.id
2 FROM attribute\_value a, pg\_class p
3 WHERE a.id = 3 AND a.tableoid = p.oid;
```

Die Art und Weise, wie die 4 erbenden Tabellen definiert sind, ist dem Umstand geschuldet, dass Constraints ([Pos15a]) nicht mit vererbt werden. Das bedeutet, in den erbenden Klassen müssen der Primärschlüssel und die Fremdschlüssel neu definiert werden.

Anmerkung: Wenn die Vererbung aufgelöst, also *attribute_value* entfernt wird, hat das keine Auswirkungen auf die darin enthaltenen Daten. Es hätte lediglich Auswirkungen auf die programmierten Integritätsbedingungen.

14.17 Gültigkeitsdauer von Passwörtern

Der Gültigkeitsablauf eines Passwortes in der Passworthistorie muss vor dem Ändern eines Passwortes geprüft werden. Sollten Passwörter vorhanden sein, welche außerhalb des Wiederholungszeitraums liegen, können diese aus der Passworthistorie gelöscht werden.

Das Testen der History könnte wie folgt realisiert werden.

1. Einträge in der Passworthistorie ermitteln
2. Salt des alten Passwortes mit dem neuen Passwort hashen
3. Vergleich des eben gehashten Passwortes mit dem Hash-Wert aus der Passworthistorie
4. Vorgang für alle Passwörter des Benutzers aus der Passworthistorie wiederholen

14.18 Konfigurationsdatei

Die Konfigurationsdatei mit dem Namen *OpenInfRA.properties* befindet sich im WebApp-Ordner unter *openinfra_core/WEB-INF/classes/de/btu/openinfra/backend/properties*

14.19 Konfigurationseinträge der Konfigurationstabelle

Hier sollen alle vom System verwendeten Konfigurationseinträge aufgelistet und mit einer kurzen Beschreibung erklärt werden. Konfigurationseinträge von Plugins werden hier nicht beachtet.

14.20 Konfiguration des Session-Timers

Für die Umsetzung des RBAC-Systems wird Apache Shiro verwendet. Der Session-Timer ist derzeit global für alle Nutzer in der *shiro.ini* auf eine Stunde (3,600,000

milliseconds) definiert und kann dort geändert werden. Die Datei *shiro.ini* befindet sich im resources-Ordner (openinfra_core/src/main/resources).

Alle ausgehenden Antworten (Responses) werden durch einen Filter geleitet, der in der Datei *web.xml* registriert ist:

Listing 14.1: web.xml Registrierung der Filter

```
<init-param>
  <param-name>
    jersey.config.server.provider.classnames
  </param-name>
  <param-value>
    de.btu.openinfra.backend.filter.OpenInfraRequestFilter,
    de.btu.openinfra.backend.filter.OpenInfraResponseFilter
  </param-value>
</init-param>
```

Der *OpenInfraResponseFilter* hinterlegt in jeder ausgehende Nachricht den Wert des Session-Timers im Header. Dazu wird das Feld *Expires* verwendet.

14.21 Erstellen des Projektes mit Eclipse

14.21.1 Jersey

Wenn Maven als Dependency-Management-System eingesetzt wird, dann eignen sich die Archtypes am besten, um eine lauffähige Version von Jersey zu erstellen. Das folgende Archtype wird hier vorgeschlagen: *jersey-quickstart-webapp* (Version 2.17). Um dieses Archtype in Eclipse verwenden zu können, kann folgender Katalog verwendet werden:

- http://mirrors.ibiblio.org/maven2/archetype-catalog.xml

14.21.2 JPA/Datenbankzugriffsschicht

Folgenden Schritte sind zum Erstellen der Datenbankzugriffsschicht in Eclipse notwendig:

1. Erzeugen einer Datenbankverbindung[28,29]
2. Das Facet JPA-Tools für das Projekt freischalten[30] (Eclipse IDE for Java EE Developers)

14.22 Dateisystem

Zum Speichern von Dateien im Dateisystem wird in der Konfigurationsdatei ein Pfad für Windows und Unix-basierte Betriebssysteme angegeben. Wenn ein anderer Pfad gewünscht ist, sollte dieser geändert werden. Das OpenInfRA-System ermittelt automatisch, welchen der beiden Pfade genutzt werden soll.

[28] https://wiki.eclipse.org/Scout/Tutorial/3.8/Database_Development_Perspective

[29] http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.datatools.connectivity.doc.user%2Fdoc%2Fhtml%2Fasc1229700328274.html

[30] http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jst.j2ee.doc.user%2Ftopics%2Ftchangefacet.html

Anhang

A Literaturverzeichnis

[Ben13] Benjamin Thurm: *Entwurf und Implementierung einer Offline-Replikation unter PostgreSQL*, Hochschule für Technik und Wirtschaft Dresden, Diplomarbeit, 2013. http://geoinformatik.htw-dresden.de/openinfra/masterarbeit_thurm/Masterarbeit_Benjamin_Thurm.pdf

[BLFM05] Berners-Lee, T. ; Fielding, R. ; Masinter, L.: *RFC 3986*. http://tools.ietf.org/html/rfc3986. Version: 2005

[Bur10] Burke, B. ; Loukides, M. (Hrsg.): *RESTful Java with JAX-RS*. O'Reilly, 2010

[DIN10] DIN: *CEN ISO/TS 19139:2009*. 2010

[Fie00] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Diss., 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[ISO14a] ISO: *ISO-3166-1-Kodierliste*. http://de.wikipedia.org/wiki/ISO-3166-1-Kodierliste. Version: 2014. – 20.01.2015

[ISO14b] ISO: *ISO 639.2*. http://www.loc.gov/standards/iso639-2/php/code_list.php. Version: 2014. – 20.01.2015

[NRA05] Nottingham, M. ; Reschke, J. ; Algermissen, J.: *Link Relations*. http://www.iana.org/assignments/link-relations/link-relations.xhtml. Version: 2005. – 30.01.2015

[Ope14] OpenInfRA Team: OpenInfRA – Grobkonzept für ein webbasiertes Informationssystem zur Dokumentation archäologischer Forschungsprojekte / Brandenburgische Technische Universität Cottbus - Senftenberg, Hochschule für Technik und Wirtschaft Dresden, Deutsches Archäologisches Institut Berlin. 2014 (2.3). – Forschungsbericht. – 22.01.2014

[Ope15] OpenInfRA Team: *Integritätsbedingungen*. 2015. – 19.02.2015

[Ora15] Oracle: *Timezones and the Java Runtime Environment: Frequently Asked Questions*. http://www.oracle.com/technetwork/java/javase/dst-faq-138158.html#patches. Version: 2015. – 18.02.2015

[Pos15a] PostgreSQL Global Development Group: *Inheritance*. http://www.postgresql.org/docs/9.4/static/ddl-inherit.html. Version: 2015. – 17.02.2015

[Pos15b] PostgreSQL Global Development Group: *Schemas*. http://www.postgresql.org/docs/9.4/static/ddl-schemas.html. Version: 2015. – 05.02.2015

[Sta13] State Geodetic Administration: *MD_CharacterSetCode*. http://listovi.dgu.hr/nippmetadata/mdcharactersetcode_engl.html. Version: 2013. – 20.01.2015

B Akronyme

CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update und Delete
CSV	Comma-separated Values
DAO	Data Access Object
DBMS	Datenbankmanagementsystem
HATEOAS	Hypermedia As The Engine Of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JPA	Java Persistence API
JSON	JavaScript object notation
JSPF	Java Simple Plugin Framework

OpenInfRA	Open Information System for Research in Archaeology
ORM	Objekt-Relationales-Mapping
OSGi	Open Services Gateway initiative
PDF	Portable Document Format
POJO	Plain Old Java Object
RBAC	Role-based Access Control
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XML	Extensible Markup Language

C Glossar

benutzerdefinierte Id

Eine benutzerdefinierte Id ist ein durch den Benutzer frei wählbarer Identifikator, welcher ausschließlich für Projekte und Themeninstanzen vergeben werden kann.

Länderkodierung

Eine Länderkodierung ist ein nach ISO 3166 Alpha 2 [ISO14a] definierter Wert.

OpenInfRA

Ein webbasiertes Informationssystem zur Dokumentation archäologischer Forschungsprojekte.

Sprachkodierung

Eine Sprachkodierung ist ein nach ISO 639-2 Alpha 3 [ISO14b] definierter Wert.

Sprachumgebung

Eine Sprachumgebung ist eine Kombination aus einer Sprachkodierung und einer Zeichenkodierung. Zusätzlich kann eine Länderkodierung definiert werden [DIN10].

Zeichenkodierung

Eine Zeichenkodierung ist ein nach MD_CharacterSetCode<<CodeList>> [Sta13] definierter Wert.

D Abbildungsverzeichnis

E Tabellenverzeichnis