Open Historical Data Map
Systembeschreibung
Version 0.0.0

Thomas Schwotzer
Mohamadbehzad Karimi Ahmadabadi
Daniel Schulz
nächste/r Projektleiter/in
(Herausgeber)

8. November 2017

Inhaltsverzeichnis

Kapitel 1

Überblick

1.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 1.1: Dokumentengeschichte

1.2 Ziel des Systems

1.3 Laufenden Arbeiten

1.4 Pläne

Kapitel 2

OHDM-Datenmodell

2.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 2017/18	Behzad Karimi	Bild eingefügt (Bilder ab jetzt mit 120mm einfügen) text text text text text

Tabelle 2.1: Dokumentengeschichte

2.2 Aufgabe der Komponente

Im Datenmodell von OHDM geht alles vom Geoobjekt (geoobject) aus. Dieses zentrale Object beschreibt alle Modelle auf der Karte, da alle Informationen

der Modelle auf das geobject verweisen. Das einzige Objekt worauf das geobject verweist, ist der Ersteller (external_users). Der Ersteller des Objekts greift durch ein externes System (external_systems) auf dern Server zu und gibt die Informationen über das Geobjekt weiter. Wie nun ein Geobjekt im Allgemeinen aussieht wird im nächsten Abschnitt beschrieben.

2.2.1 Geometrien in GIS

Da OHDM mit PostGIS arbeitet (für GIS siehe ...) werden die zweidimensionalen Objekte als Polygone repräsentiert. Polygone bestehen dabei aus Punkten (points) und Linien (lines). Die Punkte werden mit Linien verbunden, so dass am Ende ein Polygon entsteht. Der folgende Satz ist dabei eine Vorraussetzung für einen Polygon:

Ein Polygon, eine geoordete Menge von Strecken, mit der Eigenschaft, dass ein Punkt der letzten Strecke identisch zu einem Punkt der ersten Strecke ist.

Das heißt ein Polygon kann ohne Punkte und Linien nicht existieren. Ebenso kann eine Linie ohne zwei Punkte nicht existieren. Wie erstellt man nun ein Gebäudekomplex aus mehreren Gebäuden? Diese sogenannten *Multi-Polygone*, sind mehrere nicht überlappende Polygone. Dann gibt es noch die Möglichkeit Löcher in den Polygonen zu erstellen. Diese Löcher sind nichts weiter als ein Polygon in einem anderen Polygon. In GIS gibt es dabei folgende Einschränkungen:

- Polygone im inneren dürfen sich untereinander nicht überlappen. Falls doch, könnten sie auch als ein einzelnes Polygon dargestellt werden.
- Ein Rand eines inneren Polygons darf nicht Rand des äußeren Polygons sein. Falls dem nämlich so ist, wird das äußere Polygon nämlich anders dargestellt werden.
- Aus dem oberen Satz lässti sich auch folgende Eigenschaft erklären. Kein Punkt des inneren Polygons darf gleichzeitig dem äußerem Polygon gehören. Hier würden ebenfalls das äußere Polygon ansonsten anders dargestellt werden.

Wie genau nun Objekte enstehen und Polygone dargestellt werden, wird im Kapitel (...TODO...) genauer erläutert. Damit die Polygone bzw. Geobjekte ordentlich auf der Karte dargestellt werden können, teilem wir jedem Geoobjekt eine Klasse (class) zu.

2.2.2 Klassifikation von Geoobjekten

Die Entity classification weißt mit einer ID auf das Geoobjekt hin und teilt diesen in eine bestimmte Klasse ein. Jede Klasse hat wiederum nochmals Unterklassen. Dadurch können wir Geoobjekte genau beschreiben um diese auf der Karte dementsprechend anzuzeigen. Zu Klassen gehört zum Beispiel: Geschäft, Gebäude, Autobahn, Büro, historisch, Tourismus, etc.. Zu den Unterklassen

gehören Dinge wie: Bahnhof, Flugplatz, Fahrrad, Brücke, Busstaion, etc.. Nun können wir Geoobjekte darstellen und erklären zu welcher Klasse bzw. Unterklasse diese gehören. Wie nun genau ein Polygon dargestellt wird, erklären wir im nächsten Abschnitt.

2.2.3 Der Inhalt eines Geoobjekts

Die Entity geoobject_content verweist anhand einer ID auf die Instanz Inhalt (content). Dort wird beschrieben was genau das Geoobjekt ist. Wenn z.B. auf einer Karte die HTW zu erkennen ist und darauf geklickt wird, werden Informationen angezeigt die in dieser Instanz gespeichert sind. Zurück zur Instanz geoobject_content. Dort werden Zeitliche Informationen gespeichert, wie z.B. bis wann das Objekt existierte. Wir möchten nun in OHDM eine Möglichkeit haben, anhand einer URL auf spezifische Geoobjekte zuzugreifen. Damit das möglich ist gibt es zwei weitere Instanzen.

2.2.4 URLs für Geoobjekte

In der Instanz URL (url) existiert eine URL womit direkt auf das verwiesene Geobjekt zugegriffen werden kann. Diese Instanz weist aber erst auf die Instanz geoobject_url welche denselben Inhalte wie die Insatnz geoobject_content speichert.

2.3 Architektur

2.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

2.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

2.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

2.4 Nutzung

2.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

2.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

2.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

2.5.1 Test

Wie wird die Komponente getestet.

2.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 3

Kartenerzeugung und WMS/WFS

3.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 3.1: Dokumentengeschichte

3.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

3.3 Architektur

3.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

3.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

3.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

3.4 Nutzung

3.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

3.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

3.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

3.5.1 Test

Wie wird die Komponente getestet.

3.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 4

OSM-Archiv

4.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Wintersemester 17/18	Schwotzer, Thomas	OSM parsen und Füllen Intermediate DB
Wintersemester 17/18	IHR NAME	text text

Tabelle 4.1: Dokumentengeschichte

4.2 Aufgabe der Komponente

Eine, wenn nicht die, wesentliche Quelle für OHDM ist Open Street Map (OSM)[1].

In einem *initialen Upload* wurde OHDM im Sommer 2017 mit den Daten des Planet.osm Files vom Januar 2017 gefüllt.

Diese Komponenten realisiert daneben das jährlich Update der OHDM Datenbank basierend auf OSM-Planet-Files.

Der Update-Prozess wird im Detail weiter unten beschrieben.

4.3 Architektur

Die Komponente teilt sich in zwei Subkomponenten:

OSM2Intermediate parsed das OSM File und füllt die Intermediate Database.

[1] osm.org

Intermediate2OHDM füllt oder erneuert die OHDM Datenbank mit Daten aus OSM.

Diese Komponenten bietet keine Schnittstelle nach außen an.
Diese Komponente nutzt keine weiteren Komponenten des Systems.

4.4 OSM-to-Intermediate

Diese Teilkomponenten parsed die OSM Files und füllt die Intermediate Database. Die Struktur der Intermediate ist einfach. Sie enthält fünf Tabellen.

Die Tabelle `nodes`, `ways` und `relations` werden direkt aus den Einträgen im OSM-File gefüllt. Jede Tabelle enthält OSM-Nutzer und -ID. Die Nodes enthalten die Koordinaten. Ways und Relations enthalten die IDs der Nodes bzw. Ways, die den Way bzw. die Relation beschreiben.

Die IDs werden in diesen Tabelle als String gehalten. Dadurch wird die Reihenfolge der IDs gespeichert.

Es gibt zwei weitere Tabelle: `waynodes` repräsentiert die 1-n Beziehung zwischen ways und nodes. Die `relationsmember` speichert die 1-n-Beziehung zwischen Relation und ihren Membern (nodes bzw. ways.)

4.4.1 SQL_OSMImporter

Der `SQL_OSMImporter` implementiert `DefaultHandler` und arbeitet wie folgt:

Begin / Ende Dokument

Der Parser erkennt den Beginn und das Ende des XML Dokuments. Diese Events werden jeweils einmal am Anfang und am Ende des Parse-Prozesses geworfen. Die Methoden `startDocument()` und `endDocument()` werden dabei aufgerufen. Bei Beginn wird eine Statusmeldung erzeugt.

Am Ende werden die `SQLQueues` geschlossen - siehe dazu 4.5.1.

Start / End Element

Der Parser ruft die Methode `startElement()` auf, wenn er den Beginn eines XML Tags erkennt. Die Methode `endElement()` wird gerufen, wenn das Ende eines XML Elements entdeckt wird. XMl-Elemente können geschachtelt sein und sind es im OSM-XML-File auch. Einem Aufruf eines `startElement()` können weitere Aufrufe der gleichen Methode folgen, weshalb der Zustand relevant ist, in dem der Aufruf erfolgt.

Die beiden Methoden werden durch den Importer implementiert. Es gibt sechs verschiedene Tags im OSM File. Die Tags `node, way, relation` enthalten Beschreibungen von Punkten, Wegen oder Relationen. Die Tags `tag, nd, member` treten nur innerhalb der Tags auf.

Die ersten drei Tags dürfen nur als direkte Kindknoten der XML-Root auftauchen. Es wird deshalb geprüft, ob der Parser aktuell *außerhalb - OUTSIDE*

war, d.h. auf der Ebene der Root. Es ist ein Fehler, wenn das nicht der Fall ist. Der Fehler wird aber ignoriert, was nicht sauber programmiert ist (!).

Im Erfolgsfall wird für `node, way, relation` die Methode `newElement` aufgerufen. Im Fall von `tag, nd, member` wird jeweils `addAttributes, addND, und addMember` aufgerufen.

Das Tag `tag` enthält weitere Informationen zu dem Element - das sind Attribute, die später in die Intermediate DB eingetragen werden. Das Tag `nd` gibt es nur innerhalb von `way` Tags. Es folgt die ID eines Nodes, das Teil des Weges ist. Das Tag `member` gibt es nur innerhalb einer `relation`. Es folgen Beschreibungen (vor allem IDs) der Member einer Relation. Das können Nodes und Ways sein.

newElement

Mit jedem Aufruf von `newElement` wird ein `INSERT` Kommando erzeugt. Dieses Kommando wird nicht direkt an die Datenbank geschickt, sondern in einer `SQLStatementQueue` gepuffert, siehe 4.5.1. Das dient lediglich der Performance.

In dieser Implementierung werden parallel mehrere `SQLStatementQueues` gefüllt. Die Insert-Queue enthält `INSERT`-Statements, die die Tabellen `node, ways, relations` der Intermediate DB füllen. Die Member-Queue sammelt Statements, die in die `waynodes, relationmember` gespeichert werden.

Der Code mag anfangs etwas verwirrend sein. Es hilft, zu verfolgen, wie die verschiedenen Queues nacheinander gefüllt werden. Es ist auch zu beachten, dass die Statements erst mit dem Aufruf von `endElement` geschlossen werden.

Es gilt auch zu beachten, dass zwischen den Start und dem Ende eines Elements auch die anderen drei Methoden `addAttributes, addND, und addMember` aufgerufen werden können, die die `INSERT`-Statement im weitere Parameter ergänzen.

4.5 Utilities

4.5.1 SQLStatementQueue

Objekte von `SQLStatementQueue` sind ein Puffer zwischen dem Parser/Handler und der Datenbank. Objekte der `SQLStatementQueue` werden mit einem Parameterfile erzeugt. In dem File stehen die wesentlichen Informationen, um eine JDBC-Connection zu einer Datenbank zu erzeugen.

Danach arbeiten sie ähnlich einem `StringBuilder`. Es können schrittweise mit `append` String hinzugefügt werden. Die Objekte prüfen nicht, ob eine gültige SQL-Syntax entsteht. Die Objekte senden die Statement an die Datenbank, wenn ein definierbarer Schwellwert erreicht ist oder wenn explizit die Methode `force` (in Varianten) aufgerufen wird.

Eine Variante sind die FileSQLQueues. Diese erzeugen Files, in denen die Statements gespeichert werden. Die Managed-Queues sorgen außerdem dafür, dass diese Files nach einem gewissen Füllstand geschlossen werden und mittels psql ausgeführt werden.

Die Implementierung dieser Klassen ist sehr stabil. **Der Nutzung hat sich bewährt und ist in dieser Komponente Pflicht!**

4.6 Intermediate-to-OHDM

Der Quellcode dieser Teilkomponenten liegt im package `osm2inter`.

Das Package enthält nur wenige Klassen. `OSMImport` enthält die `main()` Funktion. Dort wird ein `SAXParser` erzeugt. Der Parser benötigt ein Objekt, das die Klasse `DefaultHandler` implementiert.

Der Parser parsed darauf das OSM-File. Sobald ein neues XML-Element gefunden wurde, wird eine entsprechende Methode auf dem DefaultHandler aufgerufen.

4.7 Nutzung

Der Code befindet sich im Repository `OSMUpdateInsert`[2]

4.7.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

4.7.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

4.8 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

4.8.1 Test

Wie wird die Komponente getestet.

[2] https://github.com/OpenHistoricalDataMap/OSMImportUpdate

4.9 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 5

Import

5.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 5.1: Dokumentengeschichte

5.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

5.3 Architektur

5.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

5.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

5.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

5.4 Nutzung

5.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

5.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

5.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

5.5.1 Test

Wie wird die Komponente getestet.

5.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 6

Importer 2

6.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 6.1: Dokumentengeschichte

6.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

6.3 Architektur

6.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

6.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

6.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

6.4 Nutzung

6.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

6.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

6.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

6.5.1 Test

Wie wird die Komponente getestet.

6.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 7

Editoren-API

7.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 7.1: Dokumentengeschichte

7.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

7.3 Architektur

7.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

7.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

7.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

7.4 Nutzung

7.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

7.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

7.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

7.5.1 Test

Wie wird die Komponente getestet.

7.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 8

Editoren

8.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 8.1: Dokumentengeschichte

8.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

8.3 Architektur

8.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

8.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

8.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

8.4 Nutzung

8.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

8.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

8.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

8.5.1 Test

Wie wird die Komponente getestet.

8.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 9

Linked Data Schnittstelle

9.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 9.1: Dokumentengeschichte

9.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

9.3 Architektur

9.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

9.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

9.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

9.4 Nutzung

9.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

9.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

9.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

9.5.1 Test

Wie wird die Komponente getestet.

9.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 10

SPARQL Schnittstelle

10.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 10.1: Dokumentengeschichte

10.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

10.3 Architektur

10.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

10.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

10.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

10.4 Nutzung

10.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

10.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

10.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

10.5.1 Test

Wie wird die Komponente getestet.

10.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 11

GeoSPARQL Schnittstelle

11.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 11.1: Dokumentengeschichte

11.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

11.3 Architektur

11.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

11.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

11.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

11.4 Nutzung

11.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

11.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

11.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

11.5.1 Test

Wie wird die Komponente getestet.

11.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 12

Data Provenance

12.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 12.1: Dokumentengeschichte

12.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

12.3 Architektur

12.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

12.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

12.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

12.4 Nutzung

12.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

12.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

12.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

12.5.1 Test

Wie wird die Komponente getestet.

12.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 13

CIDOC CRM Unterstützung

13.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 1980	IHR NAME	text text text text text text
Wintersemester 1980/81	IHR NAME	text text text text text text

Tabelle 13.1: Dokumentengeschichte

13.2 Aufgabe der Komponente

Verbale kurze prägnante Beschreibung, was die Komponente leisten soll. Das sind wenige Seiten.

(Ausfüllen in Prototyp-Phase)

13.3 Architektur

13.3.1 Überlick

Grafik der Teile der Komponente (wichtig: Benennung aller Schnittstellen). Anwendung der Komponente nennen (Use Case).

Übliche Interaktionen durch Interaktionsdiagramme.

(Ausfüllen in Prototyp-Phase)

13.3.2 Schnittstellendefinitionen

Beschreibung der angebotenen Schnittstellen. Benennung der Funktionen mit Vor- und Nachbedingungen. Beschreibuung des Protocol-Bindings.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

13.3.3 genutztes Komponenten

Beschreibung, welche weiteren Komponenten (in welchen Versionen, wo beziehbar) genutzt werden.

(Beginnen in Prototyp-Phase. Konkretisieren in der Alphaphase)

13.4 Nutzung

13.4.1 Code

Wo findet man den Code. Struktur des Codes. (In Prototyphase ausfüllen, kann dort sehr kurz sein. Ab Alpha-Phase konkret beschreiben.)

13.4.2 Deployment / Runtime

Beschreibung wie die Komponenten aus dem Quellcode erzeugt werden kann, wie sie installiert wird und wie man sie startet.

13.5 Qualitätssicherung

(Ausfüllen ab Alpha-Phase).

Wie erfolgt die Sicherung der Qualität? Keine Romane, sondern ehrlich notieren, was man tut. Wenn man nichts tut, dann steht hier: Wir sichern die Qualität der Komponente nicht.

Issue-Tracking: wie erfolgt das, interne Fehlermeldungen (ab Alpha), externe Fehlermeldungen ab Beta.

13.5.1 Test

Wie wird die Komponente getestet.

13.6 Vorschläge / Ausblick

Was ist aufgefallen, was sollte man ändern? Löschen Sie auch gern die Kommentare der Vorgänger, aber nur, wenn es wirklich nicht mehr relevant ist.

Kapitel 14

OHDM OfflineMaps mit Xamarin

14.1 Dokumentengeschichte

Zeitraum	PL/Autor(en)	Änderungen
Sommersemester 2017	Schulz, Daniel	Kapitel erstellt und Software dokumentiert

Tabelle 14.1: Dokumentengeschichte

14.2 Aufgabe der Komponente

Bei den OHDM OfflineMaps (oder auch der OHDMApp) mit Xamarin handelt es sich um eine mobile Anwendung, welche vorab einen Datenexport aus OHDM erhält und danach in der Lage ist auf dem mobilen Gerät offline die entsprechenden Karten zu einem selbst bestimmbaren Zeitpunkt zu rendern. So kann in der App beispielsweise der 01.02.1790 ausgewählt werden und es würden die zu diesem Datum gültigen Kartendaten dargestellt werden. Da die Anwendung auf Xamarin und C# basiert kann sie jederzeit mit geringem Aufwand auch auf iOS portiert und ausgerollt werden. (Bisher wird nur Android unterstützt)