Open Historical Data Map
Systembeschreibung
Version 0.0.0

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

4. 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 1980/81	IHR NAME	text text text text text text

Tabelle 2.1: Dokumentengeschichte

2.2 Aufgabe der Komponente

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

(Ausfüllen in Prototyp-Phase)

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

Editoren-API

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

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

Linked Data Schnittstelle

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

SPARQL 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

GeoSPARQL 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

Data Provenance

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

CIDOC CRM Unterstützung

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

OHDM OfflineMaps mit Xamarin

13.1 Dokumentengeschichte

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

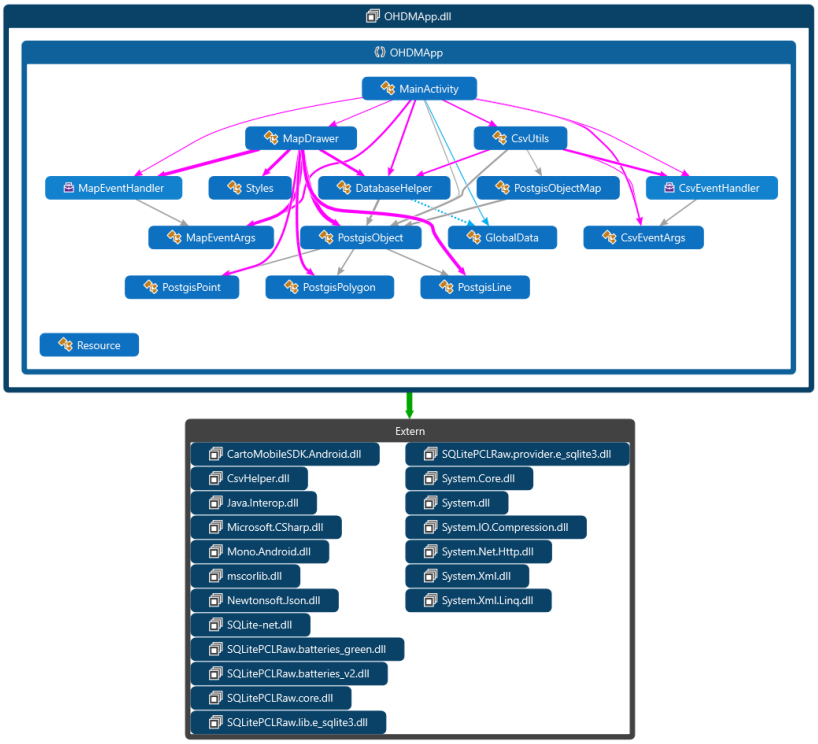
Tabelle 13.1: Dokumentengeschichte

13.2 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)

13.3 Architektur

13.3.1 Überlick



In der obenstehenden durch Visual Studio automatisiert generierten Code-Map lassen sich die einzelnen Bestandteile der Xamarin-Anwendung ablesen, sowie die extern eingebundenen Bibliotheken/NuGet-Pakete.

Nachfolgend werden alle Bestandteile sehr kurz erläutert:

- MainActivity: Diese Klasse ist für das UserInterface unter Android zuständig und definiert jegliche Interaktion mit dem Benutzer.
- MapDrawer: Diese Klasse definiert die grundlegende Darstellung von Kartenelementen, also zum Beispiel in welchen Zoomstufen sie gerendert werden und welche Styles sie nutzen.
- CsvUtils: Diese Klasse beinhaltet die Implementierung der Übertragung der Kartendaten aus einer CSV, welche zuvor aus der OHDM-Datenbank erzeugt wurde in eine neue interne SQLite-Datenbank für die Anwendung.
- MapEventHandler, MapEventArgs, CsvEventHandler, CsvEventArgs: Diese Klassen definieren Events, welche bei den verschiedenen Datenverarbeitungsschritten der App genutzt werden um dem Anwender den Fortschritt zu signalisieren.
- Styles: In dieser Klasse werden die Styles definiert, welche der MapDrawer später anwendet. Dies sind zum Beispiel die Farbe und Stärke von Linien, Punkten oder Polygonen.
- DatabaseHelper: Der DatabaseHelper definiert die Schnittstellen und Funktionen zur Anbindung an die lokale SQLite-Datenbank, welche zur Datenhaltung der eingelesenen Kartendaten genutzt wird.
- PostgisObjectMap: Diese Klasse wird genutzt um die Felder der Postgis-Objects auf die Spalten der Daten-CSV zu mappen.
- PostgisObject, PostgisPoint, PostgisPolygon, PostgisLine: Diese Klassen stellen die objektbasierte Darstellung der Daten aus der OHDM-Datenbank dar, welche vom Programm genutzt wird.
- GlobalData: Diese Klasse ist nur eine statische Klasse zur Zwischenspeicherung von Daten auf welche von mehreren Klassen zugegriffen werden muss von denen nicht alle den nötigen Kontext aufweisen.

Nachfolgend werden alle externen Libraries erläutert, welche nicht automatisiert von Xamarin geladen oder benötigt werden, sondern spezifisch für die Anwendung erforderlich sind:

- CartoMobileSDK: Hierbei handelt es sich um das Karten-Framework Carto, welches zur Darstellung der Offline-Karten genutzt wird.
  `https://carto.com/docs/carto-engine/mobile-sdk/`
- CsvHelper: Bei CsvHelper handelt es sich um ein .NET basiertes Framework zum automatisierten einlesen und mappen von Objekten aus einer CSV-Datei.
  `https://github.com/JoshClose/CsvHelper`
- Newtonsoft.JSON: Hierbei handelt es sich um ein Framework zur Serialisierung und Deserialisierung von Objekten in JSON-Objekte.
  `https://www.newtonsoft.com/json`
- SQLite-net: Hierbei handelt es sich um eine Multiplattform-Bibliothek, welche die Erzeugung und Anbindung von SQLite-Datenbanken ermöglicht.
  `https://github.com/praeclarum/sqlite-net`

13.3.2 Schnittstellendefinitionen

Die einzige Quasi-Schnittstelle, welche von den OfflineMaps genutzt bzw. benötigt wird ist ein Datenexport mit OHDM-Daten aus einer OHDM-Datenbank. Zum Einlesen wird ein CSV-File in der folgenden Struktur benötigt:

```
id;name;type_target;classification_id;classname;subclassname;valid_since;
valid_until;point;line;polygon
19;"S Tiergarten";1;396;"public_transport";"stop_position";"2015-12-22";
"2017-07-10";"{""type"":""Point"",""coordinates"":[13.3364598,52.5142085]}";"";""
20;"S Tiergarten";1;542;"railway";"station";"2015-12-22";
"2017-07-10";"{""type"":""Point"",""coordinates"":[13.3364598,52.5142085]}";"";""
65;"Jakob-Kaiser-Platz";1;589;"highway";"motorway_junction";"2017-01-01";
"2017-07-10";"{""type"":""Point"",""coordinates"":[13.2906741,52.533132]}";"";""
112;"S Pichelsberg";1;542;"railway";"station";"2016-03-04";
"2017-07-10";"{""type"":""Point"",""coordinates"":[13.2275633,52.5102387]}";"";""
136;"S Schoeneberg";1;583;"highway";"bus_stop";"2016-03-11";
"2017-07-10";"{""type"":""Point"",""coordinates"":[13.3529104,52.4797246]}";"";""
```

Aus der Berliner-OHDM-Datenbank, welche auf dem PostgreSQL-Server der HTW Berlin liegt können die Daten in diesem Format mit der folgenden Abfrage extrahiert werden:

```
SELECT gog.id, go.name, gog.type_target, gog.classification_id,
c.class as classname, c.subclassname, gog.valid_since, gog.valid_until,
ST_AsGeoJSON(p.point) as point, ST_AsGeoJSON(l.line) as line,
ST_AsGeoJSON(poly.polygon) as polygon

FROM berlin.geoobject as go, berlin.classification as c, berlin.geoobject_geometry as gog

LEFT OUTER JOIN berlin.points as p ON gog.type_target=1 AND gog.id_target=p.id

LEFT OUTER JOIN berlin.lines as l ON gog.type_target=2 AND gog.id_target=l.id

LEFT OUTER JOIN berlin.polygons as poly ON gog.type_target=3 AND gog.id_target=poly.id

WHERE go.id=gog.id_geoobject_source AND gog.classification_id=c.id AND
gog.type_target>0 AND gog.type_target<4 AND
(point IS NOT NULL OR line IS NOT NULL OR polygon IS NOT NULL);
```

Alternativ könnten die Daten auch aus den Rendering-Tabellen bezogen werden, hierzu müsste nur eine neue Abfrage entwickelt werden und das Programm von WGS84, welches bisher genutzt wird, auf Pseudo-Mercator umgestellt werden.

13.3.3 Genutzte Komponenten

In der Anwendung wurden die folgenden Komponenten, welche zuvor in Kapitel 13.3.1 detaillierter beschrieben wurden in den angegebenen Versionen benutzt:

- CartoMobileSDK: Version 4.1.0
- CsvHelper: Version 2.16.3.0
  (Achtung! Diese Version ist wichtig, da in der nachfolgenden Version die Nutzung komplett umgestellt wurde und ein komplettes Rewrite des CSV-Codes erfordern würde.)
- Newtonsoft.JSON: Version 10.0.3
- SQLite-net bzw. sqlite-net-pcl: Version 1.4.118

Alle Versionen können ganz normal über den in Visual Studio integrierten NuGet-Paketmanager installiert werden.

13.4 Nutzung

13.4.1 Code

Der aktuelle Code kann unter
`https://github.com/OpenHistoricalDataMap/OfflineMaps`
bezogen werden. Die Struktur des Codes wurde bereits in Kapitel 13.3.1 erläutert und grafisch dargestellt.

13.4.2 Deployment / Runtime

Als Vorbedingung für das Deployment müssen zuerst folgende Dinge auf dem PC, welcher eingesetzt werden soll vorliegen bzw. installiert sein:

Android ADB-Treiber von dem Mobilgerät, auf welchem später die Anwendung deployed werden soll.
Der Standard-Google-Treiber kann hier bezogen werden:
`https://developer.android.com/studio/run/win-usb.html`
Alternativ sind noch diese möglich: `https://adb.clockworkmod.com`
Hierbei ist zu beachten, dass jeder Gerätehersteller zum Teil eigene Treiber benötigt, vor allem Samsung, diese sind separat im Internet zu beziehen.

Zusätzlich muss das USB-Debugging auf Android aktiviert werden, wie hier beschrieben:
`https://www.howtogeek.com/129728/how-to-access-the-developer-options-menu-and-enable-usb-debugging-on-android-4.2/`

Es muss ein Datenextrakt aus der OHDM-Datenbank in Form einer CSV-Datei vorliegen. In Kapitel 13.3.2 wird erläutert, wie und in welchem Format dieser zu erzeugen ist.
Die Datei muss hinterher den Namen “outputnew.csv” erhalten.

Um das OfflineMaps-Projekt zu bearbeiten oder zu deployen ist als Entwicklungsumgebung Visual Studio 2017 mit Xamarin nötig.
Visual Studio 2017 kann theoretisch in der kostenlosen Community-Edition heruntergeladen und installiert werden, da aber für Hochschulangehöroge die Enterprise-Version über Microsoft Imagine/Dreamspark kostenlos bezogen werden kann, sollte diese genutzt werden und in der nachfolgenden Erläuterung wird auch nur auf diese eingegangen.
Informationen zum Download von Microsoft-Produkten für Hochschulangehörige der HTW Berlin finden sich hier:
`http://www.f4.htw-berlin.de/studieren/softwarelizenzen/`
Im nachfolgenden Link wird erläutert mit welchen Optionen Visual Studio 2017

installiert werden muss um Xamarin-Projekte zu unterstützen:
`https://developer.xamarin.com/guides/cross-platform/getting_started/installation/windows/`
Die Xamarin-Installation entsprechend dem Guide ist zwingend erforderlich! Zusätzlich müssen im Installationsbildschirm von Visual Studio 2017, in welchem auch die Xamarin-Pakete angehakt werden auch zwingend alle Git und GitHub-Pakete, sowie Pakete, welche das Android SDK betreffen angehakt werden, hierzu sollten alle Unterpunkte durchgeklickt und durchsucht werden.

Nachdem Visual Studio 2017 mit allen zuvor beschriebenen Komponenten installiert wurde, kann es gestartet werden.
Anschließend kann man dem Tutorial von Microsoft zum Klonen eines Git-Repos ab dem Punkt "Clone from another Git Provider" folgen (die Punkte darüber können ignoriert werden):
`https://docs.microsoft.com/en-us/vsts/git/tutorial/clone?tabs=visual-studio#clone-from-another-git-provider`
Der Git-Link des Repos der OfflineMaps, welcher zum klonen eingesetzt wird lautet: `https://github.com/OpenHistoricalDataMap/OfflineMaps.git`
Gegebenfalls fragt VisualStudio vor dem Klonen noch die eigenen GitHub-Zugangsdaten ab.

Nachdem die Solution entsprechend der Angaben des Microsoft-Tutorials geklont und geöffnet wurde ist das Projekt nahezu bereit zum Deployment und es müssen nur noch zwei individuelle Schritte ausgeführt werden:

Zuerst muss in der AndroidManifest.xml, welche sich unter Properties befindet ein eigener Name für das Package gesetzt werden, da es sonst später zu Fehlern mit dem API-Key kommt.
In der Klasse MainActivity muss die Stringkonstante LICENSE in welcher nur exemplarisch "XXX" steht gegen einen eigenen API-Key von Carto ausgetauscht werden. Dieser kann von Carto kostenlos erzeugt werden gemäß der nachfolgenden Anleitung unter dem Punkt "Registering your App":
`https://carto.com/docs/carto-engine/mobile-sdk/getting-started/#registering-your-app`

Als letzten Schritt muss nur noch die im Projektverzeichnis unter Assets liegende "outputnew.csv" durch die ausgetauscht werden, welche gemäß Kapitel 13.3.2 erzeugt wurde.

Danach kann die App durch einen Klick auf den grünen Pfeil im oberen Zentrum der Navigationsleisten oder einen Druck auf F5 auf dem (an den PC angeschlossenen) Android-Gerät deployed und ausgeführt werden.
Sollte es Fehlermeldungen wegen nicht korrekt geladener NuGet-Pakete geben, so können diese in der NuGet-Konsole (Extras -> NuGet-Paket-Manager -> Paket-Manager-Konsole) mit dem Befehl "update-package -reinstall" neu initialisiert werden.

Zusätzlicher Hinweis: Durch einen Bug im aktuellen Android-SDK verhält sich Android Oreo noch etwas seltsam, so schließt sich beim Ausführen über Visual Studio die Anwendung direkt wieder. Hier muss die Anwendung nachdem sie über Visual Studio installiert wurde einfach vom Handy aus gestartet werden, dann funktioniert alles. Für das Debugging sollte aus diesem Grund aktuell Android in der Version $\leq$ 7.1.2 verwendet werden. Außerdem ist dringend anzuraten, alle Visual Studio und Android SDK Updates zu installieren.

13.5 Qualitätssicherung

Die Qualitätssicherung der Darstellung erfolgt bisher durch akribische Abgleiche der OHDM-Darstellung mit der aktuellen deutschen OSM-Darstellung und die Angleichung an dieselbe. Für das Handling der Anwendung was Verständlichkeit und Performance angeht wurde die Qualität durch externe unbeteiligte Personen bewertet und dann verbessert.

Issue-Tracking: Bisher keines.

13.5.1 Test

Die Komponente kann momentan ausschließlich durch Anwender-Testgruppen getestet werden und nicht automatisiert, aufgrund der Komplexität der Darstellung. Es wäre allenfalls möglich die Datenimport und -Exportfunktionen um Unit-Tests zu erweitern, was aber aktuell noch nicht umgesetzt ist.

13.6 Vorschläge / Ausblick

In der Anwendung sind bisher nur exemplarisch die wichtigsten Typen an OSM-Objekten mit spezifischen Rendering-Merkmalen für eine spezifische Darstellung versehen. Hier bietet es sich an auch die übrigen zu implementieren. Außerdem wäre es sinnvoll und gewinnbringend die vorhandene Xamarin-Codebasis zu erweitern und um die entsprechenden iOS und UWP-Schnittstellen zu erweitern um die Anwendung auch auf iOS- und Windows-Geräten zum Einsatz bringen zu können.