Bachelorarbeit

GIS2Simulation

Fachgebiet „Computergestützte Modellierung und Simulation“ an der TU-München

Autor: Wildgruber Josef
Betreuerin: Dipl.-Inf. Angelika Kneidl

Thema:

GIS2Simulation

Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Gewinnung und der Verarbeitung von Geodaten, um sie als Grundlage für Fußgängersimulationen zu nutzen. Für die Verarbeitung wurde ein selbst entwickeltes Java-Programm verwendet. Die Daten werden in einer XML-Datei ausgegeben.
Am Anfang der Bachelorarbeit wird das Internetportal „Openstreetmap.org“ als Quelle für Geodaten vorgestellt und seine Funktionsweise erklärt. Die erhaltenen Daten können auf verschiedenen Arten in Java verarbeitet werden, die gängigsten werden vorgestellt und auf Vor- und Nachteile untersucht. Im Anschluss wird die im Programm verwendete Methode und ihre Funktionsweise detaillierter erklärt. Es folgen die entstandenen Probleme sowie der Aufbau und der Inhalt der erzeugten XML-Datei. Zum Abschluss wird gezeigt, wie ein Fußgängersimulator diese Datei verwendet und welche Schwächen das Programm nach wie vor besitzt.

Abstract

The following Bachelor Thesis deals with the recovery and converting of spatial data. They are the foundation to simulate the routes chosen by pedestrians in order to transit a predefined area. The converting is realised by a self created Java programm. The purpose is to create a XML-File which contains the inportend data. If it includes the required structure, a simulator will be able to use it.
In the beginning of the thesis openstreetmap is introduced as a source for spatial data. Further, the functions of this platform are explained. There are different ways to handle the exported data with Java. The most popular ones are named and analysed. After that the applied method and the way it works are explained more detailed. Subsequent the thesis contains a closer look at the created XML-file. In the end, the way a simulator uses these files and still unsolved problems are shown.

Gliederung

1. Bedeutung von Fußgängersimulationen

Fußgänger sind ein wichtiger Bestandteil des Verkehrsbildes einer Innenstadt. Allerdings verursachen sie auch eine Vielzahl an Problemen. Da jeder einzelne Fußgänger ständig subjektiv entscheidet, wie er seinen Weg fortsetzt, entsteht ein chaotisches System. In Extremfällen kann dies zu erheblichen Personenschäden führen, wie die Vorfälle auf der Loveparade 2010 in Duisburg zeigen. Eine Möglichkeit das Verhalten von Fußgängern im Voraus abzuschätzen bieten Fußgängersimulatoren. Durch die Entwicklung immer schnellerer Prozessoren ist es möglich sehr komplexe Situationen durchzurechnen und damit reale Gefahrenpotentiale vorzeitig zu erkennen und Unglücke zu verhindern.
Die Aufgabe meiner Bachelorarbeit „GIS2Simulation“ im Fachgebiet „Computergestützte Modellierung und Simulation“ an der TU-München bestand darin, für einen vom Lehrstuhl entwickelten Fußgängersimulator, begehbare Flächen, Straßen und Gebäude eines zuvor festgelegten Gebiets in Form einer XML-Datei zu liefern. Ist diese richtig aufgebaut, kann der Simulator daraus das Gebiet rekonstruieren und als Grundlagen für Fußgängersimulationen benutzen.
Die TU-München hatte anhand von Versuchen das Verhalten von Fußgängern beim durchqueren der münchener Innenstadt untersucht. Mit Hilfe des Simulators sollen die gewonnen Erkenntnisse nun vollautomatisch auf beliebige Gebiete angewendet werden. Die dafür benötigten Daten können aus verschiedenen Quellen stammen. Eine attraktive Möglichkeit Rohdaten über die vorhandene Bebauung eines beliebigen Gebiets zu gewinnen bot der kostenlose Exportservice der Interseite „openstreetmap.com“ (OSM-Dateien). Sie werden in dieser Arbeit verwendet und damit gleichzeitig auf ihre Tauglichkeit geprüft.
In der Programmiersprache Java entwickelte ich ein Programm, das die gewünschten Informationen aus einer OSM-Datei ausliest und daraus eine neue XML-Datei mit vorgegebener Form erzeugt. Diese XML-Datei kann abschließend vom Fußgängersimulator verwendet werden. Im Simulator werden zunächst die Gebäude und Wege angelegt. Die Gebietsgrenzen, sowie die Lage der Gebäude und Wege werden dafür aus der XML-Datei ausgelesen. Die Simulation kann nach Angabe einer Quelle und eines Ziels, welche beliebig im betrachteten Gebiet gewählt werden können, ablaufen.

2. OSM-Dateien

Anfragen bei Behörden nach Geodaten sind meist sehr zeitaufwändig und kostenpflichtig. Eine attraktive Alternative bietet das Internetportal „openstreetmap.org“, auf dem über eine einfache „Export“-Funktion Rohdaten eines beliebigen Flächenstücks als OpenStreetMap-XML-Datei (im Folgenden OSM-Datei), aber auch in anderen Formaten, heruntergeladen werden können. Die nachfolgende Abbildung zeigt wie das zu exportierende Gebiet auf dem Internetportal, durch die Angaben seiner Grenzen in geographischen Koordinaten, markiert wird.

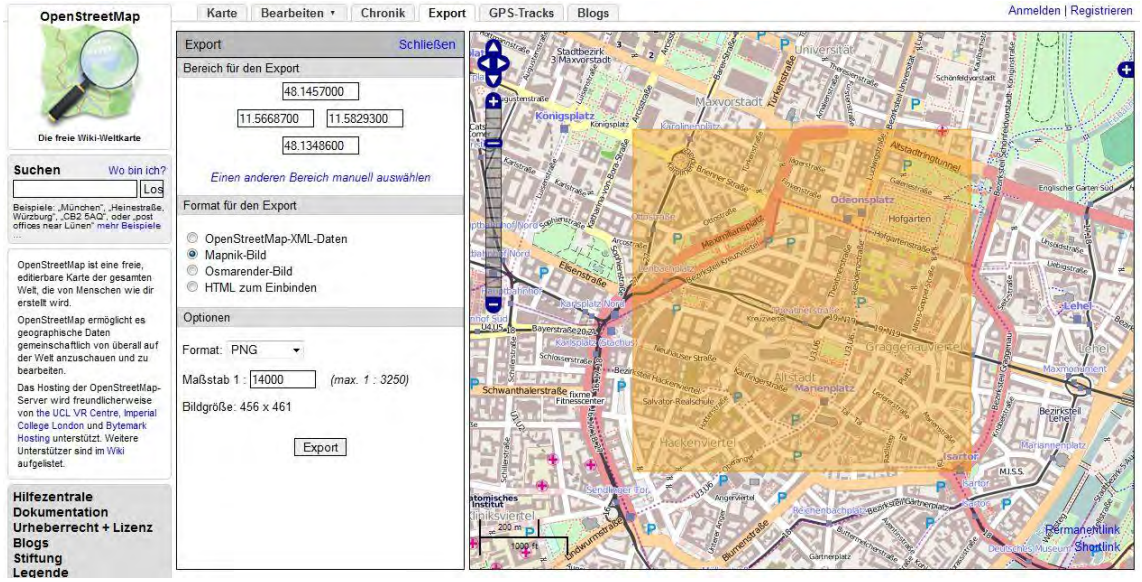

Abb1: Export-Service von „www.Openstreetmap.org“ (siehe Quelle 6)

Hinter dem OSM Format verbirgt sich wie auf „www.openstreetmap.org“ angegeben eine XML-Datei. Hierbei handelt es sich nicht um eine Bilddatei, sondern um eine schematische Auflistung aller von Mitgliedern erfassten Daten. Sie liefern die Grundlage für meine Arbeit. Da die OSM-Datei sehr umfangreich ist, befindet sich hier vorerst nur ein kleiner Ausschnitt.

Abb2: Anfang einer OSM-Datei (Exportfunktion von „http://www.openstreetmap.org“)

„Mapnik-Bild“, „Osmarender-Bild“ und „HTML zum einbinden“ stellen drei weitere Exportmöglichkeiten dar. Bei „Mapnik“ und „Osmarender“ handelt es sich um verschiedene Renderer, die das Bild auf unterschiedliche Art und Weise zeigen. Ein „Mapnik“-Bild kann als PNG, JPEG, SVG, PDF oder Postskript gespeichert werden. Zusätzlich hat man die Möglichkeit einen Maßstab für das Bild zu bestimmen.

Abb3: Mapnik-Bild (Exportfunktion von „http://www.openstreetmap.org“)

Ein „Osmarender“-Bild kann als PNG oder JPEG exportiert werden. Hier wird außerdem eine ZOOM-Funktion angeboten.

Abb4: Osmerander-Bild (Exportfunktion von „http://www.openstreetmap.org“)

Um das Bild in einen eigenen HTML-Code zu integrieren, liefert die Exportmöglichkeit „HTML zum einbinden“ die benötigte Textzeile.
Im Folgenden wird nur auf die OSM-Dateien eingegangen, da in dieser Arbeit nur diese verwendet wurden. Die Rohdaten umfassen Straßen, Eisenbahnen, Flüsse, Wälder und Häuser, aber auch Positionen kleinerer Objekte, wie Parkbanken, Briefkästen und Geldautomaten.
Die Vollständigkeit einer OSM-Datei ist in jedem Gebiet unterschiedlich, da freiwillige Mitglieder mit und ohne GPS-Empfänger, die Geodaten liefern. Jeder Mensch weltweit kann beliebige Objekte mit einem GPS-Empfänger vermessen und die Koordinaten mit Beschreibung des Objekts zu „Openstreetmap.org“ hinzufügen. Ohne GPS-Empfänger kann man „Openstreetmap.org“ verbessern, indem man falsche, veraltete oder ungenaue Inhalte meldet. Im Moment steht vor allem in dicht bebauten Gebieten eine große Menge an genauen Daten zur Verfügung. In ländlicheren Gebieten hingegen sind oft nur Straßen vorhanden, kleine Dörfer fehlen zum Teil vollständig. Die für meine Arbeit benötigen Daten befinden sich innerhalb des Gebiets mit den Koordinatengrenzen 48.13486° bis 48.1457° und 11.56687° bis 11.58293°.
Dieses Flächenstück liegt in der Münchner Innenstadt und erstreckt sich vom Odeonsplatz bis zum Isartor. Hier wurden die vorhandenen Gegebenheiten für meine Zwecke ausreichend gut aufgenommen. OSM-Daten stehen unter der Lizenz “Creative Commons Attribution-Share Alike 2.0”, welche auch eine gewerbliche Nutzung der Daten zulässt. Sämtliche daraus abgeleiteten Produkte müssen unter der CC-BY-SA-Lizenz stehen und sind deshalb ebenfalls frei zugänglich.[1]

[1] http://www.openstreetmap.de/faq.html (aufgerufen am 4.12.2011)

2.1. Allgemeine Inhalte

Die Grundelemente einer Karte sind Punkte, Wege und Flächen. Mit Hilfe einer sehr langen „Liste“[2] von anerkannten Kategorien, die sich ständig weiterentwickelt, werden sie näher beschrieben. Dabei gibt es immer einen „Schlüssel“ und einen „Wert“. In der OSM-Datei tauchen sie unter jedem Element mit der Bezeichnung „tag“ als Attribute „k“ und „v“ auf. „K“ ist dabei der „Schlüssel“ und „v“ der „Wert“. Der „Schlüssel“ ordnet das Objekt einer Kategorie zu. Das Grundelement Weg erhält zum Beispiel häufig den Schlüssel „highway“. Dieser steht für begehbare und/oder befahrbare Wege. Der „Wert“ liefert dann eine detailliertere Beschreibung, wie etwa „footway“. Somit ist ersichtlich, dass es sich um einen Weg, den nur Fußgänger betreten dürfen, handelt. Weitere „Schlüssel“ und „Werte“ können, wie auch der folgende Ausschnitt zeigt, beliebig hinzugefügt werden.

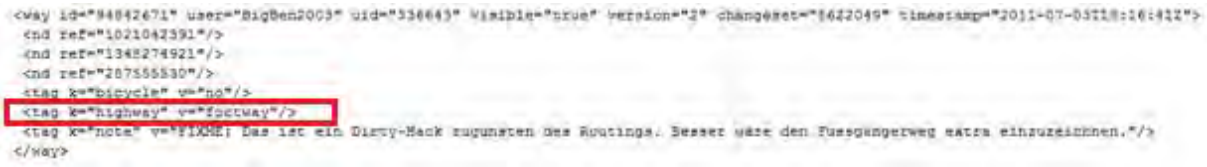

Abb5: Beispiel Weg-Element

Prinzipiell ist es auf „Openstreetmap.org“ jedem selbst überlassen auf welche Art und Weise er seine aufgenommenen Daten zur XML-Datei hinzufügt. Für eine einheitliche Nutzung der Daten ist es allerdings vorteilhaft, wenn bestimmte Grundregeln eingehalten werden. Aufgenommene Straßen sollten zuerst mit bereits vorhandenen Straßen auf „Openstreetmap.org“ verglichen werden, um herauszufinden welche Kategorie die eigene Straße am besten beschreibt. Bei Straßen, die unter keine bereits vorhandene Kategorie fallen, ist es ratsam sich in Foren mit anderen „Openstreetmap“-Mitgliedern, die vielleicht dasselbe Problem haben, zu unterhalten, eine allgemeine Lösung zu entwickeln und diese zu melden. Solche Diskussionen finden ständig statt und machen „Openstreetmap.org“ zu einer immer attraktiver werdenden Quelle für Karten aller Art.

2.2. OSM-Datei der Münchener Innenstadt

Die in dieser Arbeit verwendete OSM-Datei besteht aus drei Teilen. Zuerst werden alle Punkte, die im betrachteten Gebiet aufgenommen wurden, aufgelistet. Jedes Punkt-Element besitzt ein Attribut „id“, als Wert hat es die Identifikationsnummer des Punktes, sowie zwei weitere Attribute „lat“ und „lon“. Diese drei Informationen sind für die Arbeit von Bedeutung. Sie sind in der anschließenden Abbildung rot markiert.

[2] http://wiki.openstreetmap.org/wiki/DE:Map_Features (aufgerufen am 4.12.2011)

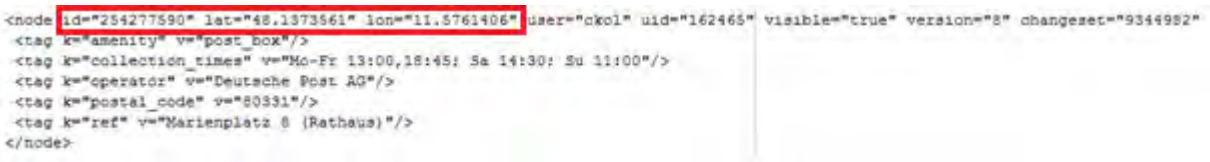

Abb6: Beispiel für ein Punkt-Element in der OSM-Datei

Über die Identifikationsnummer kann an jedem Ort der OSM-Datei eine Referenz zu einem beliebigen Punkt eingefügt werden. Dies wird bei den Weg-Elementen sichtbar. „Lat“ und „lon“ beinhalten die zwei geographischen Koordinaten des Punktes, die beispielweise mit GPS-Geräten aufgenommen wurden. Durch Kind-Elemente mit der Bezeichnung „tag“ können weitere Informationen zu jedem Punkt hinzugefügt werden. Dies geschieht, wie oben bereits erklärt, mit „Schlüsseln“ und „Werten“. Geldautomaten, Springbrunnen, Briefkästen und Rollstuhlrampen können so in der XML-Datei erkennbar dargestellt werden. Der zweite Teil der OSM-Datei enthält sämtliche Wege. In den Kind-Elementen „nd“, von denen jedes Weg-Element mehrere besitzt, steht jeweils ein Attribut mit der Bezeichnung „ref“. Dieses steht für eine Referenz und beinhaltet die Identifikationsnummer eines Punktes. Der Weg wird letztendlich als Polylinie durch alle diese Punkte beschrieben. Mit Hilfe der „tag“-Elemente wird nun, wie schon bei den Punkten, der Weg genauer beschrieben. Im Abschnitt mit den Weg -Elementen befindet sich ein Großteil der, für die Simulation letztendlich wichtigen, Informationen. Ziel ist es, die begehbaren Flächen, die Straßen und die Gebäude aus der OSM-Datei auszulesen. Alle drei sind in Form von Weg-Elementen gegeben. Wege mit dem Schlüssel „building“ und dem Wert „yes“ sind Gebäude, die als Polygone dargestellt werden. Sie sind daran zu erkennen, dass der erste zugehörige Punkt gleich dem letzten ist. Begehbare Wege besitzen den Schlüssel „highway“ und als Wert entweder „footway“, „steps“, „pedestrian“ oder „residential“. Das erste Element im Folgenden Ausschnitt einer OSM-Datei ist damit ein Gebäude, das zweite ein begehbarer Weg.

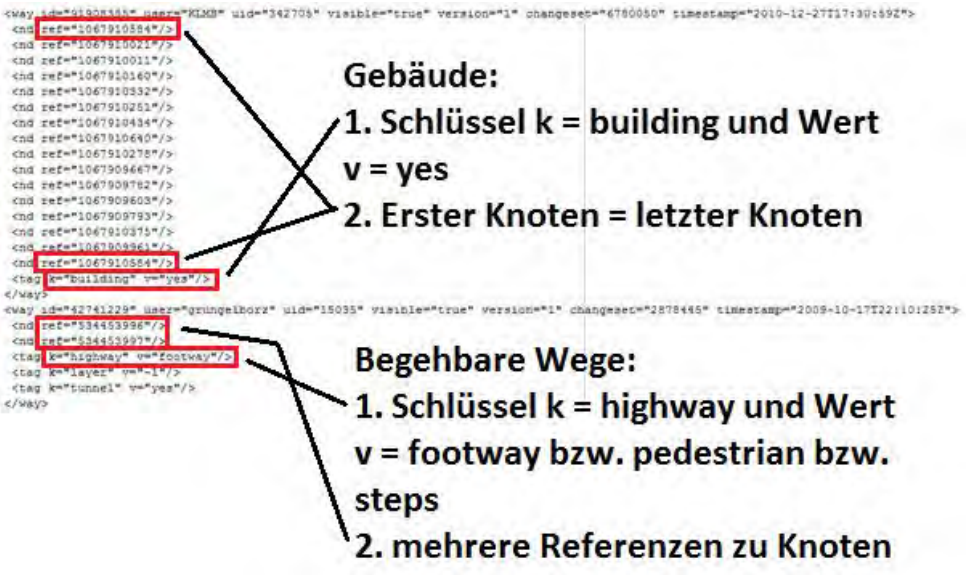

Abb7: Beispiele für Weg-Elemente in der OSM-Datei

Die begehbaren Wege setzen sich aus Bürgersteigen, Fußgängerzonen und Treppen zusammen. Der Wert „footway“ steht für Bürgersteig. Der Wert „pedestrian“ steht für Plätze und Wege, auf denen nur Fußgänger erlaubt sind. Einkaufspassagen wie die Kaufingerstraße sind ein typisches Beispiel dafür. Treppen, die Bestandteil eines Fußweges sind, haben den Wert „steps“. Straßen innerhalb von Wohngebieten haben als Wert „residential“.
Der dritte Teil der OSM-Datei enthält *relation*-Elemente. Jedes Element besitzt mehrere Mitglieder, die entweder für einen Weg oder einen Punkt stehen. Es gibt immer einen Zusammenhang zwischen allen Mitgliedern, was mit dem übergeordneten Element Relation schon angedeutet wird. Eine Relation steht dann beispielsweise für eine MVV Haltestelle, die sich aus mehreren Elementen zusammensetzt. Ein *relation*-Element ist im Folgenden dargestellt, für die vorliegende Arbeit sind sie allerdings nicht von Bedeutung.

```
<relation id="1582107" user="Michael Forster" uid="64536" visible="true" version="1" changeset="8117208" timestamp="2011-05-11T22:
  <member type="node" ref="307495749" role="stop"/>
  <member type="node" ref="928661369" role="stop"/>
  <member type="node" ref="735102760" role="stop"/>
  <member type="node" ref="486031943" role="platform"/>
  <member type="node" ref="735102749" role="stop"/>
  <member type="node" ref="735102758" role="platform"/>
  <member type="node" ref="656507719" role="entrance"/>
  <member type="node" ref="975224874" role="entrance"/>
  <member type="node" ref="656507696" role="entrance"/>
  <member type="node" ref="656507693" role="entrance"/>
  <member type="node" ref="682198273" role="entrance"/>
  <tag k="name" v="Odeonsplatz"/>
  <tag k="network" v="MVV"/>
  <tag k="operator" v="MVG"/>
  <tag k="public_transport" v="stop_area"/>
  <tag k="type" v="public_transport"/>
  <tag k="website" v="http://www.mvv-muenchen.de"/>
</relation>
```

Abb8: Beispiel für eine Haltestelle der MVV in der OSM-Datei

3. XML-Verarbeitung mit Java

Im nächsten Schritt werden die so gewonnenen Daten mit Hilfe eines selbst entwickelten Java-Programms weiter bearbeitet. Die für den Fußgängersimulator wichtigen Informationen sind die begehbaren Wege, die Straßenflächen und die Gebäude, welche vom Programm aus der OSM-Datei ausgelesen und in eine neue XML-Datei geschrieben werden sollen. Ein solches Programm wird als Parser bezeichnet. Es gibt mehrere Möglichkeiten einen solchen Parser mit Java zu schreiben. Die grundlegenden Methoden eine XML-Datei mit Java zu bearbeiten werden im Folgenden vorgestellt.

3.1. SAX-Parser

SAX steht für Simple API for XML und bietet schnelle und einfache Möglichkeit, um XML-Dateien zu bearbeiten. Wichtig ist zunächst, dass ein SAX-Parser ereignisorientiert handelt. Er liest die XML-Datei Zeichen für Zeichen von oben nach unten. Es ist also nicht möglich, Sprünge an verschiedene Stellen der XML-Datei zu machen. Während der Parser liest, erkennt er unterschiedliche Ereignisse, wie zum Beispiel ein neues Startelement. Bei jedem Ereignis kommt der sogenannte „Contenthandler“ ins Spiel. Der Parser übergibt jedes Ereignis, auf das er beim Lesen stößt, an den „Contenthandler“, welcher dann überprüft, ob eine Methode ausgeführt werden soll oder nicht. Eine der Hauptaufgaben bei der XML-Bearbeitung mit SAX ist es, diese Methoden oder Rückruffunktionen für verschiedene Ereignisse zu implementieren. Diese grundlegende Funktionsweise wird im folgenden Schema veranschaulicht.

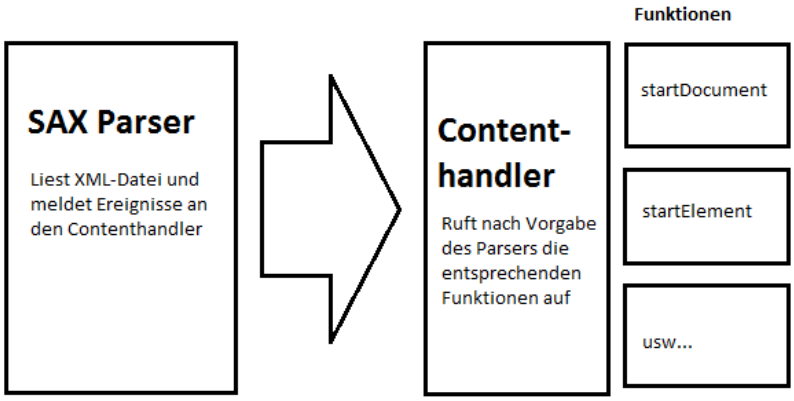

Abb9: Arbeitsschema eines SAX Parsers

Ein einfaches Beispiel ist eine Rückruffunktion, die bei jedem Startelement in der XML-Datei eine Zählvariable um die Zahl Eins erhöht. Am Ende des Dokuments, welches wiederum ein Ereignis darstellt, kann der „Contenthandler“ dann über eine weitere Methode ausgeben, wie viele Elemente in der XML-Datei vorhanden sind.[3]

[3] Siehe Quelle 1 S. 41f, S.49

3.1.1. Vorteile SAX-Parser

SAX-Parser arbeiten in der Regel schneller als DOM-Parser, da sie die XML-Datei direkt beim Einlesen gleichzeitig weiterverarbeiten. Außerdem ist es nicht nötig die komplette Baumstruktur zwischen zu speichern, was einen geringen Bedarf an Speicherplatz zur Folge hat. Damit ist ein SAX-Parser besonders dafür geeignet nach bestimmten Inhalten in XML-Dateien zu suchen.[4]

3.1.2. Nachteile SAX-Parser

Da der SAX-Parser die XML-Datei nur in eine Richtung liest, ist es nicht möglich in beliebiger Reihenfolge auf die Elemente zuzugreifen.[5] Aus demselben Grund werden auch nichtwohlgeformte XML-Dateien zunächst geparst, da erst am Schluss und nach mehreren Methoden die Gesamtstruktur feststeht. Ein validieren vor dem Parsen würde keinen Sinn ergeben, da der Vorteil von SAX darin besteht, dass man die XML-Datei vor der Bearbeitung nicht speichern muss.[6] Zudem ist weder ein Mehrfachzugriff mit unterschiedlichen Methoden auf einzelne Elemente, noch eine Ausgabe der veränderten Struktur in einer neuen XML-Datei möglich.

3.2. DOM-Parser

DOM steht für Document Object Model und ist eine Schnittstelle zum Zugriff auf HTML- und XML-Dateien. Es wurde 1998 von W3C entwickelt (DOM Level 0) und wird von vielen Programmiersprachen unterstützt. Ursprünglich versuchte man mit DOM die miteinander konkurrierenden Möglichkeiten der HTML-Bearbeitung, JavaSkript und dynamic HTML, zu standardisieren und letztendlich zu verdrängen, was schließlich erfolgreich war. Es ist Möglich sowohl XML-Dateien als auch HTML-Codes mit der Hilfe von DOM in Baumstrukturen umzuändern. Diese Darstellung als Baum wird als DOM-*Core* bezeichnet. Im Anschluss ist dieser Zusammenhang dargestellt.

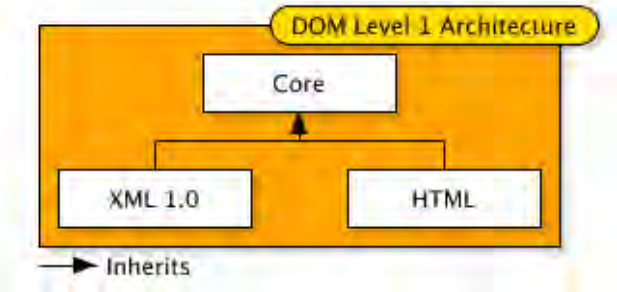

Abb10: DOM-Ebene 1 (U. Altmann, Einführung in das Document Object Model (DOM), Institut für Medizinische Informatik, Justus-Liebig-Universität Gießen www.med.uni-giessen.de/akkk/xml/DOM/)

[4] http://openbook.galileodesign.de/javainsel8/javainsel_15_003.htm#mj15294ee1d9a1aca900b941214c5e9e55 (aufgerufen am 4.12.2011)
[5] Siehe Quelle 1 S. 96
[6] http://de.wikipedia.org/wiki/Simple_API_for_XML (aufgerufen am 4.12.2011)

Über die Jahre wurde es ständig weiterentwickelt und mit neuen Modulen ergänzt. Heute ist er wichtiger Bestandteil der JavaSkript-Programmierung. Die aktuelle Version ist „DOM Ebene 3“. Die folgende Abbildung zeigt die Vielzahl an Ergänzungen die seit dem ersten DOM-Entwurf hinzugefügt wurden.

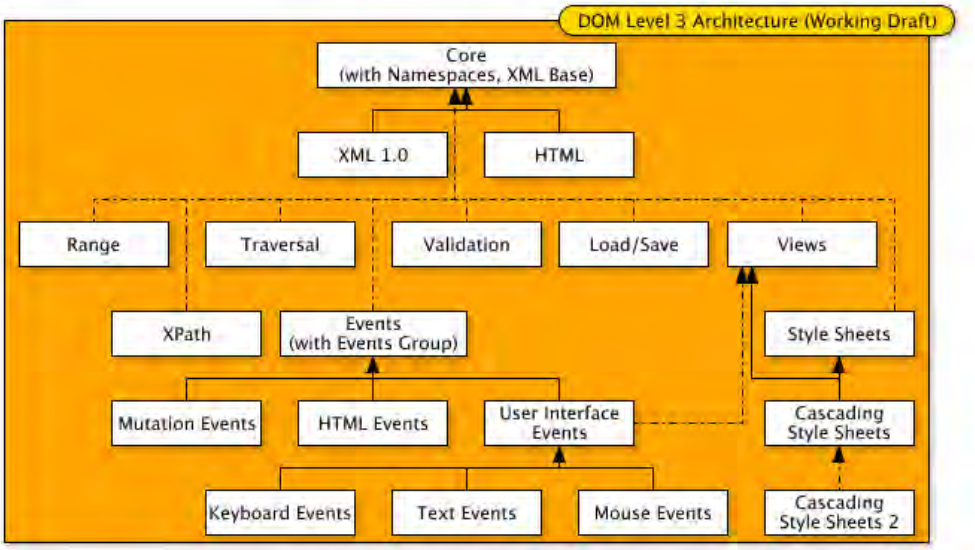

Abb11: DOM-Ebene 3 (U. Altmann, Einführung in das Document Object Model (DOM), Institut für Medizinische Informatik, Justus-Liebig-Universität Gießen www.med.uni-giessen.de/akkk/xml/DOM/)

Im Gegensatz zum SAX-Parser erstellt der DOM-Parser zunächst eine Kopie der kompletten Struktur einer XML-Datei. Anschließend wird sie als Baum mit Knoten für jedes Element abgespeichert. Das erste Element wird als Wurzelelement (*root*) bezeichnet. Elemente, die eine Stufe unter dem *root*-Element stehen, sind dessen Kinder (*children*) und das *root*-Element ist ihr Elternteil (*parent*). Zwei Elemente mit dem gleichen *parent*-Element bezeichnet man als Geschwister (*siblings*). Diese Struktur zieht sich durch alle Ebenen, sodass jedes Element außer dem *root*-Element, ein *parent*-Element besitzt. Nun ist es möglich, jedes Element von jedem anderen Element aus über *parent-child*-Beziehungen zu erreichen. Das bedeutet, dass alle Informationen zu jedem Zeitpunkt verwendet werden können.[7]

3.2.1. Vorteile DOM-Parser

Ein DOM-Parser eignet sich hervorragend um Informationen aus gegebenen XML-Dateien zu verändern, zu löschen und zu ergänzen, da man sich sehr einfach, schnell und in allen Richtungen durch die Datenstruktur bewegen kann. Es ist somit ein leichtes, XML-Strukturen zu sortieren und die Datei interaktiv zu bearbeiten.[8]

[7] http://de.wikipedia.org/wiki/Document_Object_Model (aufgerufen am 4.12.2011)
[8] http://de.wikipedia.org/wiki/Document_Object_Model (aufgerufen am 4.12.2011)

3.2.2. Nachteile DOM-Parser

Der DOM-Parser wurde als Standard entwickelt und muss in verschiedenen Sprachen anwendbar sein. Programmierer, die nur Java gewohnt sind, werden also unter Umständen Probleme haben, weil javaspezifische Strukturen nicht berücksichtigt werden und die Programmierung somit weniger intuitiv ist.[9]
Da die komplette XML-Datei zunächst gespeichert werden muss, ist diese Variante besonders bei großen Dateien sehr speicherintensiv, was bei der schnellen Entwicklung moderner Computer allerdings immer unwichtiger wird.

3.3. JDOM

Eine dritte Möglichkeit XML-Dateien mit Java zu bearbeiten bietet JDOM.
Hierbei handelt es sich um eine speziell für Java entwickelte Methode XML-Dateien zu bearbeiten. Das komplette Dokument, der JDOM-Baum, wird über das Schlüsselwort „new“ erzeugt. Es unterscheidet sich somit nicht von anderen JAVA-Klassen. Es ist nicht nötig zusätzliche Hilfsmittel zum Erstellen des Dokuments zu implementieren. Danach wird der JDOM-Baum mit Informationen aus der gegebenen XML-Datei gefüllt. Dies kann über zwei sogenannte *builder*-Klassen geschehen. Ist eine XML-Datei vorhanden, die weiter verarbeitet werden soll, bietet sich der „SAXBuilder“ an. Er konstruiert einen JDOM-Baum aus SAX Ereignissen, was sehr schnell und speichersparend abläuft. Der „DOMBuilder“ sollte verwendet werden, wenn bereits ein DOM-Baum vorliegt und dieser in einen JDOM-Baum transformiert werden soll. Der größte Vorteil von JDOM ist die intuitive Anwendung. Java Programmierer müssen daher beim Schreiben des Quellcodes nicht umdenken. Genau wie das komplette Dokument, werden daher auch die Elemente und Attribute wie normale Klassen behandelt.[10]
JDOM verbindet die Vorteile von SAX und DOM. Informationen können mit dem „SAXBuilder“ schnell eingelesen und gleichzeitig in einer übersichtlichen Baumstruktur gespeichert werden. Desweiteren ergänzt sie diese Vorzüge mit einer für Java zugeschnittenen Struktur.

3.4. Erstellen einer XML-Datei mit StAX

Zum Ausgeben der mit JDOM bearbeiteten Informationen in einer neuen XML-Datei wird hier StAX verwendet. Mit dieser Methode können XML-Dateien schnell und einfach erzeugt werden. Benötigt wird lediglich die sogenannte „XMLStreamWriter“-Klasse. Zunächst werden ein beliebiger Pfad und ein Dateiname auf der Festplatte gewählt. Diese geben an, wo die neue XML-Datei gespeichert werden und wie sie heißen soll. Zu Beginn wird eine komplett leere XML-Datei in diesem Verzeichnis erstellt. Über verschiedene „write“-

[9] Siehe Quelle 1 S.91
[10] s. Quelle 1 S.43-46

Methoden, wie zum Beispiel „writeStartElement“, kann dann der gewünschte Inhalt der neuen XML-Datei Zeile für Zeile in die leere, bereits erstellte Datei geschrieben werden. Auf die Wohlgeformtheit der XML-Datei ist dabei genauso zu achten wie auf Zeilenumbrüche, da sonst Probleme mit der Darstellung auftreten können.[11]

[11] s. Quelle 1 S.227ff

4. Umsetzung

Im Folgenden wird beschrieben, wie aus den Informationen der OSM-Datei eine für den Fußgängersimulator verwendbare XML-Datei erzeugt wird. Auf eine kurze Zusammenfassung der OSM-Daten folgt eine Erläuterung der Art und Weise mit der der Parser sie verarbeitet und letztendlich in eine neue XML-Datei schreibt.

4.1. Verarbeitung der OSM-Datei

Zuerst werden die Grenzen des betrachteten Gebiets ausgelesen und umgerechnet.
Hierbei ist zunächst zu berücksichtigen, dass die Koordinatengrenzen sowie die Koordinaten der einzelnen Punkte in „Openstreetmap.org“ als geographische Koordinaten angegeben werden.
Im vorliegenden Fall wird allerdings ein Koordinatensystem benötigt, in dessen Nullpunkt sich die linke untere Ecke des Untersuchungsgebiets befindet. Die Grenzen für den Hoch- und Rechtswert sowie die Punktkoordinaten müssen zusätzlich in Meter umgerechnet werden. Ein mögliches Vorgehen, um dies zu realisieren, wird im Folgenden erläutert.
Eine geographische Position kann, wie in der OSM-Datei, durch die Angabe von zwei Winkeln eindeutig bestimmt werden. Der Breitengrad (hier „lat“) gibt den Winkel zwischen dem Äquator und dem beobachtetem Punkt wieder, der Längengrad (hier „lon“) den Winkel zwischen dem Nullmeridian und dem betrachteten Punkt. Dies ist anschaulich in der nächsten Abbildung dargestellt.

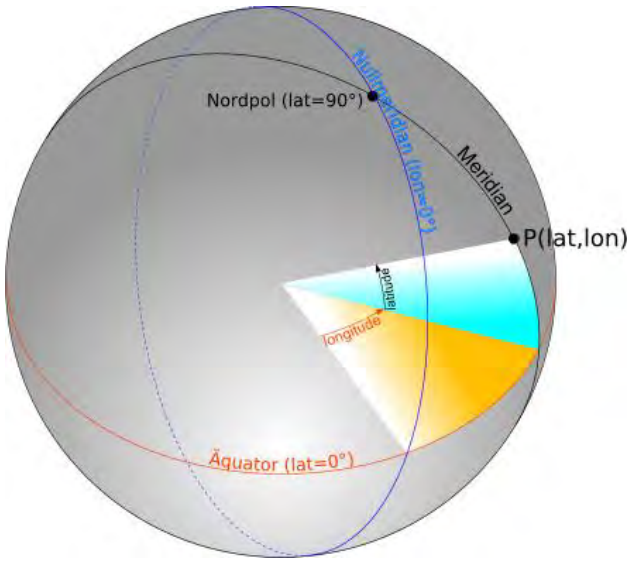

Abb12: geographische Koordinaten (http://www.kompf.de/gps/distcalc.html)

Das Untersuchungsgebiet wird durch Folgende geographische Koordinaten beschrieben.

Minlat=48.13486°, maxlat= 48.1457°, minlon=11.56687°, maxlon=11.58293°

Mit diesen Daten musste ein für den Simulator nutzbares System konstruiert werden. Seine grundlegende Beschaffenheit zeigt die nachfolgende Abbildung.

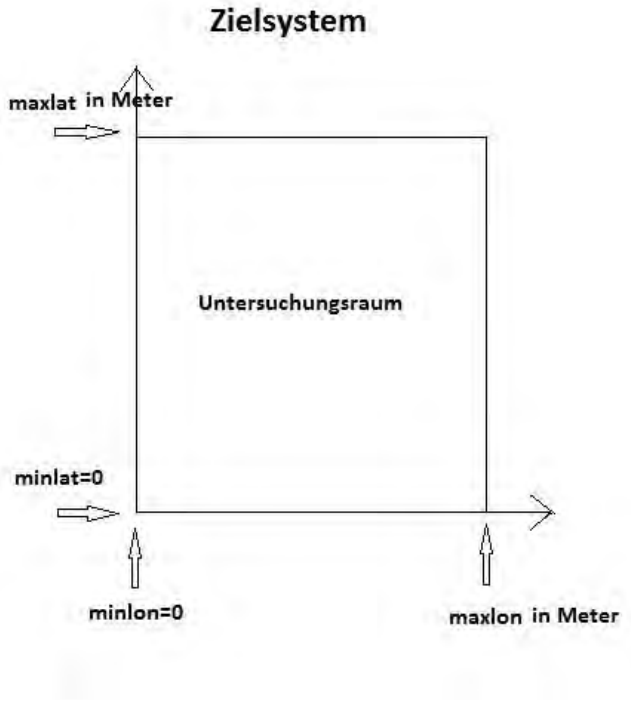

Abb13: Zielsystem

Ein Grad Breite steht immer für eine Distanz von 111300 Metern auf der Erdoberfläche. Am Äquator steht ein Grad Länge ebenfalls für eine Distanz von 111300 Metern auf der Erdoberfläche. Da sich aber alle Meridiane in den Polen schneiden, nimmt der Abstand zweier Längengrade zu den Polen hin ab. An den Polen ist er schließlich Null. Die Breitengrade zweier Punkte müssen subtrahiert und mit dem Wert 111300 multipliziert werden. Die Längengrade werden ebenfalls subtrahiert und anschließend mit dem durchschnittlichen Abstand zweier Längengrade multipliziert. Dieser beträgt 71500 Meter. Die Abmessungen meines Untersuchungsgebiets würden sich damit wie folgt ergeben.

Maxlon=71500*(maxlon-minlon)[12]
Maxlat=111300*(maxlat-minlat)

[12] Formeln zur Koordinatenumrechnung aus „http://www.kompf.de/gps/distcalc.html“ (aufgerufen am 4.12.2011)

Für nah aneinander liegende Punkte ist diese Lösung ausreichend, da der entstehende Fehler sehr klein ist. Für diese Arbeit wollte ich allerdings eine genauere Lösung. Diese berücksichtigt den Längengradabstand in Abhängigkeit des Breitengrads, auf dem der betrachtete Punkt liegt, in Form einer Kosinus-Funktion. Da diese im Allgemeinen einen Winkel im Bogenmaß (rad) erwartet müssen in Grad gegebene Winkel umgerechnet werden.[13]

$1° = 180/\pi \text{ rad} \approx 0{,}01745$

Die Abmessungen des Untersuchungsgebiets sind damit

Maxlon_in_m = 111300*cos((minlat+maxlat)/2*0,01745)*(maxlon-minlon)
Maxlat_in_m = 111300*(maxlat-minlat).

Im nächsten Schritt werden nun die Punkte aus der OSM-Datei ausgelesen. Die Koordinaten der einzelnen Punkte werden, wie die der Gebietsgrenzen, in Meter umgerechnet. „Maxlon“ und „maxlat“ müssen hierfür lediglich mit den Punktkoordinaten „lon“ und „lat“ ersetzt werden. Hier ist zu erwähnen, dass sich einige aufgenommene Wege über die Gebietsgrenzen hinweg erstrecken. Da sie immer komplett exportiert werden, befinden sich einige Punkte außerhalb des Untersuchungsgebiets. Das Problem wurde gelöst, indem alle Koordinaten die kleiner als Null waren gleich Null und alle die größer als der Maximalwert der x- oder y-Richtung waren gleich dem Maximalwert gesetzt wurden. Diese Lösung führte allerdings zu weiteren Problemen die im Kapitel 6 näher beschrieben werden.
Zum zwischenspeichern der ausgelesenen Punkte wird eine *Hashmap* verwendet.
Sie wird mit den Identifikationsnummern sämtlicher Punkte und den jeweils zugehörigen Koordinaten gefüllt. Im Programm wird sie als „NodeMap“ bezeichnet. Eine *Hashmap* besitzt einen Schlüssel- und einen Ausgabewert. Der erste Wert ist der Schlüssel, in diesem Fall die Identifikationsnummer, die als einfache Zeichenfolge abgespeichert wird. Der zweite Wert ist der Ausgabewert, der ausgegeben wird sobald über den Schlüssel auf ihn zugegriffen wird, hier die Koordinaten des Punktes. Zur Veranschaulichung ist die *Hashmap* im Anschluss skizziert.

Hashmap 1	
Identifikationsnummer =Schlüssel	**Koordinaten =Ausgabewert**
295117	x= 11.5759263 y=48.1345382
295122	x= 11.5816568 y= 48.1348224
361770	x= 11.5660751 y= 48.1397178
usw.	usw.

Abb14:Hashmap 1

[13] „http://www.kompf.de/gps/distcalc.html“ (aufgerufen am 4.12.2011)

Zu beachten ist, dass die *Hashmap* über eine Schleife gefüllt wird. Daher muss bei jedem Durchlaufen der Schleife ein neues Objekt erstellt werden, das dann in der *Hashmap* gespeichert werden kann. Wird das Objekt nur einmalig außerhalb der Schleife erstellt, wird der nur einmal reservierte Speicherplatz, bei jedem Durchlauf überschrieben. Die *Hashmap* besitzt dann nur einen Eintrag.
Alle Identifikationsnummern und Koordinaten wurden also aus der OSM-Datei ausgelesen und in einer *Hashmap* so zwischengespeichert, dass über die Identifikationsnummer eines Punktes die zugehörigen Koordinaten geliefert werden.
Als nächstes müssen die bedeutsamen Weg-Elemente ausgelesen und gespeichert werden. Jedes einzelne Weg-Element der OSM-Datei wird getestet. Wenn es ein „tag“-Element beinhaltet, das als Attribut „v“ den Wert „footway“ , „pedestrian“, „steps“ oder „residential“ hat, dann wird es in eine Liste, in dieser Arbeit „footwaysList“, geschrieben. Bei den Gebäuden muss geprüft werden, ob das „tag“-Element ein Attribut „k“ mit dem Schlüssel „building“ und ein Attribut „v“ mit dem Wert „yes“ besitzt. Sie werden dann in eine eigene Liste gespeichert, hier „obstacleList“.
Um die nun zwischengespeicherten Informationen im Simulator verwenden zu können ist allerdings noch folgendes zu beachten.
Da der Fußgängersimulator für jeden in der XML-Datei enthaltenen Punkt eine Markierung in Form eines Quadrates mit Abmessungen 1mx1m erstellt, bedurfte es einer kleinen Änderung, um das Programm verwendbar zu machen. Diese Markierung liegt mit ihrem Mittelpunkt auf den Koordinaten eines jeden Punktes. Hat ein Punkt nun eine Position die weniger als 0,5m von den Gebietsgrenzen entfernt ist, liegt ein Teil der Markierung nicht im Untersuchungsgebiet, was im Simulator zu einer Fehlermeldung führt. Die folgende Skizze stellt die Situation dar.

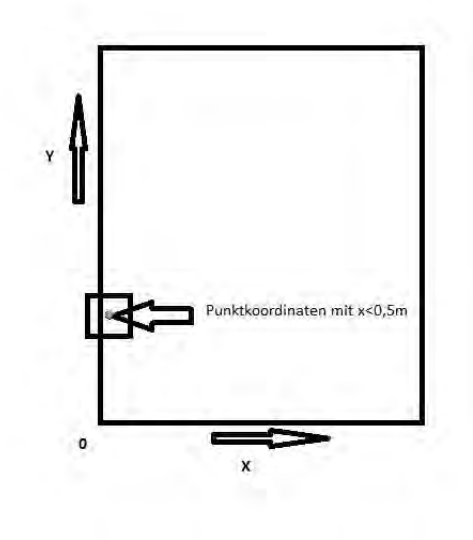

Abb15: Problematik mit simulatorinternen Punktmarkierungen

Um das zu verhindern wurden die maximalen Koordinatengrenzen um 0,5m vergrößert. Zu Punkten mit einer Koordinate unter 0,5m mussten ebenfalls 0,5m hinzuaddiert werden. Nun können die Informationen weiterverarbeitet werden.

4.2. Erstellung der neuen XML-Datei

Die gewünschte XML-Datei wird mit Hilfe von StAX kreiert und besteht aus vier Teilen. Einen für sämtliche *locations*, einen für die Wege, einen für die Gebäude und einen um die Datei für den Simulator nutzbar zu machen. Der letzte Teil ist der *traffic*-Teil und enthält keine Informationen aus der OSM-Datei. Er ist in folgender Abbildung dargestellt.

```
<traffic name ="default">
  <tqm name ="source">
    <timedata end="200" numberpersons="1200" start="0">
    </timedata>
  </tqm>
  <tqm name ="target">
    <timedata end="1000" numberpersons="12000" start="0">
    </timedata>
  </tqm>
  <velocityprofile name ="DEFAULT_GUI_VELOPROFILE">
    <meanvelocity>1.34</meanvelocity>
    <deviation>0.26</deviation>
  </velocityprofile>
  <velocityprofile name ="__DEFAULT_VELO_PROFILE__">
    <meanvelocity>1.34</meanvelocity>
    <deviation>0.26</deviation>
  </velocityprofile>
</traffic>
```

Abb16:*traffic*-Teil

Zu Beginn wird der Kopf der XML-Datei erstellt. Er beinhaltet bereits die oben berechneten Gebietsgrenzen in Metern (maxX, maxY). In der Abbildung sind sie rot markiert.

```
<?xml version="1.0"?>
  <simulation version="v3.0">
    <scenario coordsystem="this" dx="0.0" dy="0.0" rotation="0.0">
      <topology maxX="1193.5113777856902" maxY="1206.9919999994024">
        <level height="0.0" index="0" name="Level_0">
```

Abb17: Kopf der neuen XML-Datei

Innerhalb des Elements *level* befinden sich alle *location*-Elemente, sowie die Gebäude. Jede *location* hat ein Startelement. Dieses hat als Attribut immer einen Namen der sich aus den Buchstaben „loc“ und einer um den Wert Eins ansteigenden Zählvariablen zusammensetzt. Jede *location* steht für einen Punkt im betrachteten Gebiet, der Teil eines Weges ist. Sie waren ursprünglich Punkt-Elemente der OSM-Datei und wurden vom Programm in der oben beschriebenen *Hashmap* gespeichert. Mit Hilfe der „footwaysList“ können alle Identifikationsnummern der auszulesenden Wegpunkte, eine nach der anderen durchgegangen werden. Das muss über zwei Schleifen geschehen, da sowohl alle Weg-Elemente, als auch sämtliche in diesen enthaltenen Identifikationsnummern, durchlaufen werden müssen. Für jede dieser Nummern liefert die *Hashmap* ein Koordinatenpaar, welches in der neuen XML-Datei, nach folgendem Schema eingefügt wird.

```
<location name="loc1">
<point x ="734.3799203345283" y ="366.76688999975295">
</point>
</location>
<location name="loc2">
<point x ="701.7511143890954" y ="376.44998999982136">
</point>
</location>
```

Abb18: *location* Elemente in der neuen XML-Datei

Im Programm wird hier eine weitere *Hashmap* mit dem Namen „LocList“ gefüllt. Als Schlüssel werden die Koordinatenpaare der Wegpunkte gespeichert. Als Ausgabewert liefert diese zweite *Hashmap* dann den oben beschriebenen Namen der entsprechenden *location* in Form einer Zeichenfolge. Wie das Schema zeigt, funktioniert sie genauso wie die oben bereits erstellte, erste *Hashmap*. Lediglich die Art der „Schlüssel“ und die der „Werte“ sind unterschiedlich.

Hashmap 2	
Koordinaten =Schlüssel	**Name der location =Ausgabewert**
x= 11.5759263 y=48.1345382	loc1
x= 11.5816568 y= 48.1348224	loc2
x= 11.5660751 y= 48.1397178	loc3
usw.	usw.

Abb19: Hashmap 2

Dies ist nötig, da die Weg-Elemente im unteren Teil der XML-Datei nicht mehr die Koordinaten beinhalten sollen, sondern nur noch die Namen der ihnen zugehörigen *locations*. Die Abbildung Nummer 22 auf Seite 24 zeigt zwei solche Elemente.
Die Gebäude, die in der OSM-Datei noch als Weg-Elemente dargestellt wurden, tauchen in der neuen XML-Datei als Hindernis-Elemente auf. Für jedes Element der *obstacleList* wird ein neues Hindernis-Element angelegt. Sie besitzen einen Namen, der sich aus dem Buchstaben „b“ und einer, um Eins ansteigenden, Zählvariable zusammensetzt. Die Koordinaten der zugehörigen Punkte werden für jedes Hindernis über die Identifikationsnummern der Punkte aus der oben bereits verwendeten *Hashmap* ermittelt und unter das Hindernis-Element geschrieben. Die Identifikationsnummern erhält man über zwei Schleifen, die alle Elemente der *obstacleList* durchlaufen. Da es sich bei den Hindernissen um Polygone handelt, entspricht der letzte zu einem Hindernis gehörende Punkt dem ersten zu diesem Hindernis gehörenden Punkt. Jeweils zwei *locations* und Gebäude sind hier dargestellt.

```
<location name="loc2885">
<point x ="862.5822156253909" y ="663.9156299997097">
</point>
</location>
<location name="loc2886">
<point x ="858.9060247622721" y ="649.6247099995707">
</point>
</location>
<obstacle name="b1">
<point x ="978.9489739884112" y ="35.80520999928112"></point>
<point x ="982.8419557902555" y ="35.01497999953642"></point>
<point x ="984.3643156353379" y ="42.34964999984001"></point>
<point x ="995.7089362779764" y ="40.04573999980465"></point>
<point x ="991.171175839902" y ="22.805369999994696"></point>
<point x ="977.1743013123042" y ="25.131539999499353"></point>
<point x ="978.9489739884112" y ="35.80520999928112"></point>
</obstacle>
<obstacle name="b2">
<point x ="550.9124766848932" y ="379.51074000000204"></point>
<point x ="555.0127994022952" y ="389.52774000001824"></point>
<point x ="565.5844439376227" y ="385.7991899999419"></point>
<point x ="568.9195405465407" y ="396.31703999948445"></point>
<point x ="582.3439654649817" y ="391.6313099992941"></point>
<point x ="581.6383146586955" y ="389.59452000000783"></point>
<point x ="586.6603953016679" y ="388.0363199999877"></point>
<point x ="584.34284085926" y ="382.3154999995637"></point>
<point x ="580.7620346692185" y ="382.87199999974035"></point>
<point x ="577.6720212703472" y ="373.95686999994734"></point>
<point x ="576.7138193651906" y ="371.18549999998504"></point>
<point x ="550.9124766848932" y ="379.51074000000204"></point>
</obstacle>
```

Abb20: *locations* und Hindernisse in der neuen XML-Datei

Nachdem alle Gebäude aufgelistet sind, wird das Element *level* geschlossen und die begehbaren Wege und Straßen werden hinzugefügt. Sie werden entweder als *route* oder als *way* bezeichnet und besitzen einen Namen, der sich aus „way“ und einer um Eins ansteigenden Zählvariablen zusammensetzt. Desweiteren wird ihr Wert innerhalb eines Attributs namens „type“ angegeben. Sie sind Kinder des Elements „topology“ und besitzen mehrere Kind-Elemente, die alle als „intermediatetarget“ bezeichnet werden. In jedem dieser Elemente steht der Name einer *location*, die zu dem Weg gehört. Um die richtigen Namen dem richtigen Weg zuzuordnen benötigt man die *footwaysList* sowie die beiden *Hashmaps*. Die *footwaysList* liefert am Anfang für jeden Weg die Identifikationsnummern der auf ihm liegenden Punkte. Über die Nummer erhält man aus der ersten *Hashmap* die beiden Koordinaten des Punktes. Diese wiederum führen in der zweiten *Hashmap* zur Ausgabe des Namens der *location* mit diesen Koordinaten. Das nächste Bild veranschaulicht diesen Ablauf.

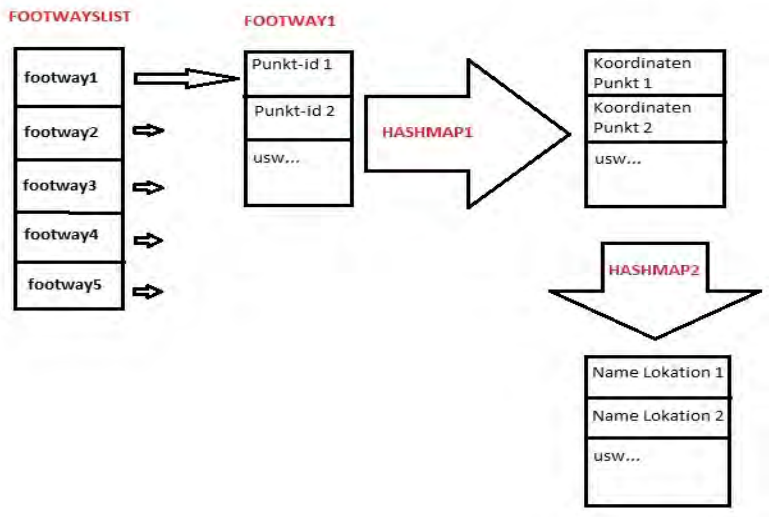

Abb21: Zuordnung von *location*-Namen zu Wegen

Jeder Name wird als Inhalt zu einem „intermediatetarget“-Element hinzugefügt. Die anschließende Abbildung zeigt unterschiedliche Darstellung von begehbaren Wegen und Straßen. Begehbare Wege, werden vom Programm anhand ihrer Werte, also „pedestrian“*,* „footway“ oder „steps“ erkannt und stehen innerhalb eines *way*-Elements. Bei Straßen wird als Kriterium der Wert „residential“ verwendet. Sie stehen innerhalb eines *route*-Elements.

```
<way name="way399" type="steps">
<intermediatetarget>loc2862</intermediatetarget>
<intermediatetarget>loc2583</intermediatetarget>
</way>
<route name="way400" type="residential">
<intermediatetarget>loc2353</intermediatetarget>
<intermediatetarget>loc2795</intermediatetarget>
</route>
```

Abb22: Begehbare Wege und Straßen in der neuen XML-Datei

Um die richtigen Attribute den richtigen Wegen zuzuordnen besitzt das Programm eine Methode namens „footwayInsert“, die für jedes Element der „footwaysList“ bis zu viermal aufgerufen wird. Bei jedem Aufruf wird geprüft, ob das Element den Wert „pedestrian“*,* „footway“, „ steps“ oder „residential“ hat. Erst wenn die Werte übereinstimmen wird ein neues Element angelegt. Hierfür ein kurzes Beispiel. Ein Element der „footwaysList“ hat den Wert „residential“ und ist somit eine Straße. Es soll daher ein Startelement namens *route* angelegt werden. Die Methode „footwayInsert“ wird aufgerufen und erhält als Übergabeparameter das aktuelle Element, sowie zwei Zeichenfolgen, „pedestrian“, „footway“ oder „step“ und „way“ bzw. „residental“ und „route“. Die ersten drei Aufrufe prüfen, ob es sich um einen begehbaren Weg handelt und somit das Startelement *way* angelegt werden soll. In unserem Fall würden diese drei Aufrufe zu keiner weiteren Prozedur führen, da die Werte nicht überein stimmen. Erst der vierte Aufruf hätte zur Folge, dass die Werte übereinstimmen und ein neues Element namens *route,* dessen Attribut „type“ den Inhalt „residential“ hat, erstellt wird.

5. Erzeugte XML-Datei

Die neue XML-Datei kann je nach Gebietsgröße eine sehr große Menge an Informationen beinhalten. Im Nachfolgenden wird auf den Umfang der erzeugten XML-Datei für das Gebiet in der Münchener Innenstadt eingegangen.

5.1. Inhalt

Die gesamte Datei hat 26141 Zeilen. Sie bilden die vier unter 4.3. aufgeführten Teile. Im Lokationen-Teil befinden sich 2886 Lokation-Elemente, die jeweils für einen Punkt auf einem der Wege stehen und durch zwei Koordinaten definiert werden. Desweiteren beinhaltet die Datei 1035 Gebäude. Jedes Gebäude ist in Form eines Hindernis-Elements gegeben. Zu jedem Gebäude sind mehrere Koordinatenpaare angegeben. Jedes Paar steht für eine Ecke des Gebäudes. Da es sich bei allen Gebäuden um Polygone handelt, ist die erste Koordinate immer gleich der letzten. Nur so ist das Polygon geschlossen. Nach den Gebäuden folgen 666 Wege. Wenn es sich bei einem Weg um einen Fußweg, eine Fußgängerzone oder eine Fußgängertreppe handelt, wird er als Weg-Element mit dem entsprechenden Wert angezeigt. Straßen hingegen sind Route-Elemente mit dem Wert „residential“. Sie werden in der gleichen Ebene wie die Wege angezeigt und erhalten genau wie sie einen Namen der sich aus dem Wort „way“ und einer um Eins ansteigenden Zählvariablen zusammensetzt. Straßen und Flächen für Fußgänger tauchen nicht in Blöcken auf sondern sind vermischt. Jedes Weg-Element beinhaltet die Namen aller Punkte aus denen sich der Weg zusammensetzt. Diese sind auch im Lokationen-Teil mit den zugehörigen Koordinaten zu finden. Der zum Schluss angehängte Verkehr-Teil besteht aus 18 Zeilen, die für den Ablauf der Simulation benötigt werden.
Die Anzahl an Elementen ist nicht konstant, sondern kann sich jederzeit ändern. Immer wenn ein neues Gebäude oder ein neuer Weg Vermessen und zu „Openstreetmap.org“ hinzugefügt wird, wird ein neues Weg-Element erstellt.

5.2. Verwendung

Die XML-Datei kann anschließend vom Simulator verwendet werden. Aus den Angaben der maximal zulässigen Koordinaten wird zunächst eine rechteckige Ebene erstellt. Die linke untere Ecke des Rechtecks ist dabei der Nullpunkt. Innerhalb dieses Szenarios werden dann zunächst die Gebäude konstruiert. Jedes Gebäude wird als graues Polygon dargestellt. Ein Hindernis-Element in der XML-Datei steht für ein Gebäude und enthält die Koordinaten sämtlicher Ecken des zu erzeugenden Polygons in Form von jeweils zwei Gleitkommazahlen. Im Anschluss daran werden alle *locations* markiert. Jede *location* wird, wie bereits beschrieben, durch ein Quadrat mit den Abmessungen 1mx1m dargestellt. Der Screenshot des Simulators zeigt beide Elemente.

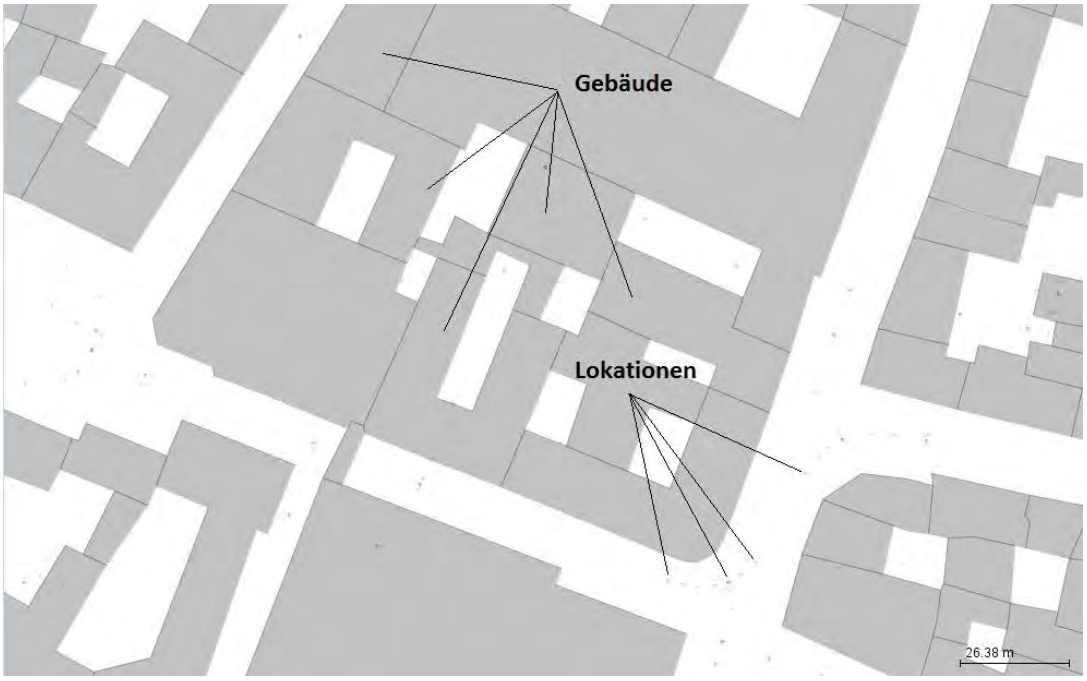

Abb23: Darstellung der Gebäude und Lokationen im Simulator

Zuletzt werden noch die Wege kreiert. Hierfür werden alle zu einem Weg gehörenden *locations* durch mehrere aufeinanderfolgende Pfeile miteinander verbunden. Welche *locations* zu welchem Weg gehören, wird über die Namen der *locations* bestimmt. Jedes Weg-Element in der XML-Datei beinhaltet die Namen aller *locations* aus denen sich der Weg zusammensetzt. Aus dem oben gezeigten Bild wird somit folgende Darstellung.

Abb24: Darstellung der Wege im Simulator

Nach Angabe einer Quelle, also eines Startpunkts von dem aus ein oder mehrere Fußgänger losgehen, und eines Ziels, können Fußgängerbewegungen zwischen diesen beiden Punkten simuliert werden.

6. Schwächen des Parsers

Teile des Quellcodes der Anwendung sind ungewöhnlich und haben daher negative Auswirkungen auf die Funktionalität. Der Parser besteht lediglich aus einer Klasse und besitzt Abschnitte die zu großen Teilen identisch sind. Der Code ist daher sehr lang und benötigt viel Festplattenspeicher. Ein weiteres Problem ergibt sich beim Einlesen der OSM-Datei und beim Ausgeben der XML-Datei. Für beide Aktionen benötigt man einen Pfad. Diese sind fest im Quellcode integriert und sagen dem Programm zu Beginn, wo sich die OSM-Datei auf der Festplatte befindet und abschließend an welchem Ort es die erzeugte XML-Datei auf der Festplatte speichern soll. Da die Pfade immer gleich sind, muss die OSM-Datei auf jedem Computer mit dem der Parser ausgeführt wird an genau dem gleichen Ort gespeichert sein. Auch der Dateiname muss identisch sein. Es ist also nötig, entweder im Vorhinein zu wissen wie der Pfad und der Dateiname lauten, oder wie der Pfad selbst im Quellcode zu ändern ist, was Grundkenntnisse im Programmieren und die nötige Software voraussetzt. Selbes gilt für die Ausgabe der neuen XML-Datei, da ansonsten nach Ablauf des Programms nicht bekannt ist, wo sich diese auf der Festplatte befinden. Der in dieser Arbeit gewählte Pfad zur OSM-Datei ist C:/map.xml und der Ausgabepfad für die neue XML-Datei D:/osm_neu.xml. Zweiter ist eher schlecht gewählt, da nicht alle Systeme über eine Partition mit der Bezeichnung D: verfügen, was dazu führen würde, dass die Datei nicht gespeichert werden kann. Letztendlich entstand eine weitere Schwäche, weil beim Programmieren und Testen lediglich eine OSM-Datei verwendet wurde. Wie bereits erwähnt, wurden Koordinaten außerhalb des Untersuchungsgebiets von mir auf 0, bzw. auf die Maximalwerte gesetzt. Dies führt dazu, dass unter Umständen mehreren Punkten die gleichen Koordinaten zugewiesen werden. Im Simulator erhält man dann eine Fehlermeldung. Im betrachteten Gebiet in der Münchener Innenstadt tritt dieser Fall zufälligerweise nicht auf. OSM-Dateien anderer Gebiete könnten allerdings aus diesem Grund vom Simulator nicht verwendet werden.

7. Zusammenfassung und Ausblick in die Zukunft

Die Arbeit zeigt zunächst wie im Internetportal „Openstreetmap.org“ Geodaten eingefügt und extrahiert werden können. Außerdem werden verschiedenen Möglichkeiten vorgestellt wie so gewonnene Dateien in der Programmiersprache Java weiterverarbeitet werden können. Der Hauptteil veranschaulicht die Bearbeitung einer OSM-Datei mit Hilfe von JDOM und die Ausgabe einer neuen XML-Datei mit Hilfe von StAX, wie sie von mir im Rahmen der Bachelorarbeit durchgeführt wurden. Folgendes Schema bietet hierzu eine abschließende Gesamtübersicht.

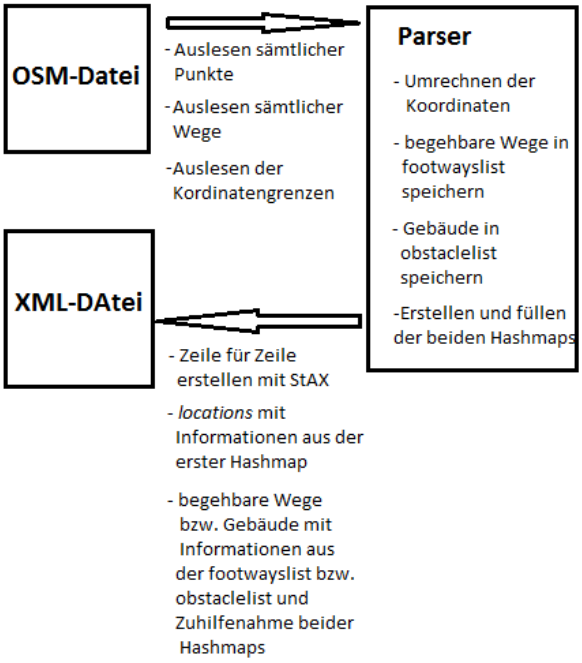

Abb25: Gesamtschema

Zum Schluss wird noch die Art und Weise erklärt, auf die die so erzeugte XML-Datei letztendlich vom Fußgängersimulator des Fachgebiets „Computergestützte Modellierung und Simulation“ an der TU-München verwendet wird.
Die verbleibenden Schwächen des Parsers bedürfen allerdings noch einiger Veränderungen. Zunächst wäre die Implementierung weiterer Funktionen und Prozeduren notwendig, um zu verhindern, dass mehrere Quellcodeabschnitte den gleichen Zweck erfüllen und den Code zu verkürzen. Besonders wichtig ist ebenfalls die Programmierung des Parsers, so dass sämtliche OSM-Dateien ein verwendbares Ergebnis liefern. Hierfür müssen die Punkte, die zwar Wegpunkte sind, aber nicht im Untersuchungsgebiet liegen, von vornherein aussortiert werden. Betroffene Wege enden dann an den Grenzen des Gebiets, obwohl sie sich außerhalb noch weiter erstrecken würden. So wird ausgeschlossen, dass zwei Punkte am Ende dieselben Koordinaten besitzen.

Ein solcher universell einsetzbarer Parser für OSM-Dateien stellt in vielerlei Hinsicht ein wichtiges Werkzeug, für das sich zukünftig viele Einsatzmöglichkeiten ergeben werden, dar. „Openstreetmap.org“ bietet eine sich ständig entwickelnde Quelle von Geodaten, deren Genauigkeit und Vollständigkeit immer größer wird. Da sie schnell und kostenlos erlangt werden können, ist „Openstreetmap.org“ ein attraktiver Datenlieferant für die Entwicklung zukünftiger, computergestützter Lösungen. Dabei sollte allerdings nie vergessen werden, dass jeder Daten zu „Openstreetmap.org“ hinzufügen kann und somit immer Fehler durch falsche Implementierung vorhanden sein werden.

8. Abbildungsverzeichnis

9. Quellenverzeichnis

1. Brett D. McLaughlin & Justin Edelson, Java & XML 3rd Edition 2007, O'Reilly Media

2. Christian Ullenboom, Java ist auch eine Insel, Galileo Computing, Kapitel 16
http://openbook.galileocomputing.de/javainsel

3. Martin Kompf, Kompf.de, Entfernungsberechnung
http://www.kompf.de/gps/distcalc.html

4. U. Altmann, Einführung in das Document Object Model (DOM),
Institut für Medizinische Informatik, Justus-Liebig-Universität Gießen
www.med.uni-giessen.de/akkk/xml/DOM/

5. http://wiki.openstreetmap.org/wiki/DE:Map_Features

6. http://www.openstreetmap.org