Technische Universität München

Chair of Geoinformatics

Context dependent multimodal routing in indoor/outdoor environments based on IndoorGML and OpenStreetMap

Master Thesis

Author:	George Mouratidis
Study Field:	Environmental Engineering
1st Supervisor:	Thomas H. Kolbe
2nd Supervisor:	Andreas Donaubauer
3rd Supervisor:	Aftab Khan

October 2015

Lehrstuhl für Geoinformatik · Technische Universität München · Arcisstraße 21 · 80333 München

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht.

München, …./…./2015

George Mouratidis

Acknowledgements

First and foremost, I would like to thank my family, Maria and all friends from the TUM and back home for their continued and unending supply of support, as well as George and Mandana for all the tech support and all around greatness.

I would also like to thank the extremely supportive Dr. Donaubauer, for constantly being available with great feedback, recommendations and ideas on the project. I could not have hoped for a better experience with a supervisor, and this project would not have moved forward as it has otherwise. The same goes for Aftab Khan, who has provided very useful feedback and support during both the initial stages of the thesis, as well as later on, assisting greatly even after leaving Munich. I would be remiss not to thank Prof. Kolbe, Mr. Dietrich, Mr. Yao, Mr. Sindram and all other members of the chair of Geoinformatics that have assisted with their feedback and help during my work on the Master's Thesis.

Additionally, I would like to thank Prof. Tsakarestos and Mrs. Bayer from the Environmental Engineering Master's team, for their availability and support with any questions and issues that arose during my study at the TUM.

Finally, I would like to thank the wonderful people of the Greek State Scholarship Foundation / IKY, for giving me the chance to learn at an institute such as the TUM and for being patient and supportive during the difficulties that arose with the completion of the Master's program.

The completion of this project was made in the framework of the implementation of the postgraduate program (MSc), that has been co-financed through the Action "State Scholarships Foundation's Grants Programme following a procedure of individualized evaluation for the academic year 2012-2013", from resources of the operational program "Education and Lifelong Learning" of the European Social Fund and the National Strategic Reference Framework 2007-2013".

Table of Contents

Table of Figures

Table of Tables

Table of Appendix Figures

Table of Abbreviations

0D	zero dimensional
1D	one dimensional
2D	two dimensional
3D	three dimensional
CAD	Computer-aided Design
CityGML	City Geography Markup Language
COLLADA	Collaborative Design Activity
CRS	Coordinate Reference System
DB	Database
ESRI	Environmental Systems Research Institute
FME	Feature Manipulation Engine
GIS	Geographic Information System
GML	Geography Markup Language
GPS	Global Positioning Systems
GTFS	Global Transit Feed Specification
IFC	Industry Foundation Classes
IndoorGML	Indoor Geography Markup Language
ISO	International Organization for Standardization
LBS	Location-Based Services
LOD	Level of Detail
MLSEM	Multilayered Space-Event Model
NRG	Node-Relation Graph
OGC	Open Geospatial Consortium
OSM	Open Street Map
OTP	Open Trip Planner
SQL	Structured Query Language
UML	Uniform Modeling Language
WGS 84	World Geodetic System 1984

1 Introduction

In this chapter, some introductory information regarding the motivation for the research into our thesis topic will be provided. Additionally, some of the challenges and current status of related work & research will be briefly discussed.

1.1 Motivation

With the increasing focus on location aware technology, indoor navigation is one field that still remains quite open for development, more so when compared to research and applications of outdoor navigation. While there has been a significant surge in development, this has been mostly aimed towards applications with specific use cases. Two factors that appear to be key for the further improvement of indoor navigation is expanding the context awareness of route planning applications (to a level comparable with outdoor navigation solutions) and achieving better interoperability of indoor/outdoor navigation, with the goal of reaching a point of seamless navigation from outdoor to indoor space.

However, there are several issues with the current state of interoperability between the available data models for outdoor and indoor space, as well as constraints that have to be taken into consideration or overcome.

It is thus the purpose of this thesis to examine in depth the architecture of IndoorGML and compare it to data models and routing implementations of outdoor dataset providers, with a special focus on OpenStreetMap. The goal is the combined use of the two models for the development of a single system capable of multi-modal indoor/outdoor routing. Several alternative processes for the achievement of this goal will be examined.

1.2 Challenges/ Current Status

One of the main challenges in developing this conceptual system is achieving a good level of interoperability. While this is an issue even when comparing data models with the same purpose, it becomes much more of an issue when trying to combine a model describing indoor spaces and their relations (such as IndoorGML) with another model that is more oriented towards mapping and routing, without so much of a focus on buildings and indoor space (such as OpenStreetMap).

Luckily, with the added traction that location aware services and open platforms like OSM have been gaining in the past years, an increasing amount of development is taking place with the goal of extending the OpenStreetMap model in order to accommodate more precise and complex data, such as building information.

While not offering the same complexity and flexibility as dedicated indoor mapping data models, these extensions provide a good starting point for enabling some level of connectivity between OSM and indoor models.

Another important challenge is the many different formats and data models being used for indoor data. From multi-purpose formats such as AutoCAD DWG to information models such as IFC to CityGML, creating a conceptual model that would allow for interoperability with all the possible data structures is practically unfeasible. As a solution, the IndoorGML model is to be used as the data model to describe indoor data. IndoorGML is a great fit as it is focused towards navigation. Additionally, it provides great flexibility by allowing for external referencing e.g. to IFC or CityGML objects, while data stored in other models such as IFC and CityGML can also be converted into IndoorGML with very good accuracy, enabling us to work with a varied set of source data.

The addition of support for context aware routing [e.g., change in form of transport from walking to Public Transit (outdoor segment) and then back to walking (outdoor and indoor segment)] is one further goal, as a multi-modal routing system without such capabilities would offer little practical use. Naturally, context awareness extends to conditions beyond accessibility and means of transport, such as time of day and opening hours (e.g. for commercial buildings), weather conditions (preference for indoor/ outdoor navigation), or access controlled areas (e.g., employee only entrances & areas). As both IndoorGML and OSM include support for restricting navigation by specific constraints and conditions, we will investigate which constraints and to what extent can be covered via the combined use of the two models regarding context-awareness.

Finally, if interoperability between the two data models is achieved, the next challenge would be the development of a prototype implementation providing multi-modal routing data. This could work as proof of concept and provide valuable feedback regarding the usability, as well as the limits of the above conceptual work.

The research approach, examined process concepts and expected outcomes will be examined in further detail in the following section.

2 Expected outcomes

This section will list the expected outcomes originating from the assorted work on the thesis topic. As the stages defined in the beginning of the proposal also define the objectives to a certain extent, the same differentiation between objectives is followed here:

The research & conceptual development should provide a solid analysis of the IndoorGML and OpenStreetMap Data models & the routing implementations available for the two data models.

Based on the above analysis, the level of conflation between the two models will be examined, as a necessary factor in achieving the interoperability necessary for combining an indoor and outdoor dataset. In the context of this thesis, conflation is used as the term to indicate the necessary process to merge different datasets, with the desired goal being the extraction and addition of indoor data from our indoor model to the outdoor dataset and its model.

Regardless of the process variant decided during the research stage, a conceptual data model will be defined, as a container of the common elements linking outdoor with indoor data.

In addition, in case Variants 1, 2 will be selected, a process will be defined for the integration of data under one of the two initial data models.

In the case of Variant 3, a wrapper schema will be defined, under which the combined indoor/outdoor data will be contained.

For Variant 4, the initial data model linking the data will be used as the basis for the development of an application concept, managing the outdoor and indoor routing segments and connecting them accordingly.

Finally, depending on the variant decided, an implementation will be created for testing purposes using available test datasets. The main objective of the implementation is a routing engine capable of handling combined indoor/outdoor datasets and providing correct routing results.

Additionally, contextual constraints (time, access, etc.) should be definable and taken into account by the routing implementation. The third objective of the implementation is multi-modal routing capability, providing the optimal combination of vehicle and on-foot routing for the outdoor & indoor segments. Additional support for other modes of transport (e.g., bicycle, transit) is a desired objective, however it is not part of the scope of this thesis.

The representation of the generated routes in a graphical user environment is also part of the thesis work objectives. However, navigation instructions or the 3D display of the route, while particularly useful for the indoor segment, are not part of the current thesis scope.

3 Research approach

This chapter will offer an overview of the research approach driving this Master's thesis. At first, a work outline is provided, describing the stages in which the work is separated. This is followed by a more detailed definition of four conceptual processes that were considered for the achievement of the thesis objectives. Finally, the expected outcomes from the thesis work are outlined.

3.1 Research Stages

As mentioned above, the work related to this Master Thesis can be broken down in the following three stages:

3.1.1 Stage 1 – Research & Conceptual Development

This stage will consist of the following objectives:
Analysis of related work.
Analysis of the IndoorGML Data Model and current indoor routing implementations.
Analysis of Outdoor data provider (OpenStreetMap) and comparative analysis of the possible routing implementations.
Examination of a conceptual system for a combined use of OSM and IndoorGML data.
Examination of an application concept enabling routing for indoor and outdoor segments, via separate handling of the indoor and outdoor segments.

3.1.2 Stage 2 – Indoor/Outdoor data integration

This stage will consist of the following objectives:
Identification of possible processes to achieve data interoperability enabling the conflation of IndoorGML and OpenStreetMap data, either by utilizing one of the two initial data models or under a combined "wrapper" schema.
Alternatively, examination of the data merging/linkage requirements of an application enabling routing for indoor and outdoor segments, via separate handling of the indoor and outdoor segments.
Identification of the optimal process/variant for context aware multi-modal routing between outdoor/indoor spaces.

3.1.3 Stage 3 – Implementation & Testing

This stage will consist of the following objectives:
Development of a prototype implementation of the selected process, allowing for routing between outdoor/indoor spaces, supporting multi-modal routing as well as context awareness (semantic driven constraints).
Assessment of the implementation regarding routing accuracy as well as context awareness.

3.2 Reviewed process concepts/variants

The below section will examine four approaches/ process concepts that were considered during development of the thesis proposal, in order to achieve the thesis objective for the development of a multi-modal, context aware routing implementation, utilizing IndoorGML and OpenStreetMap data.

3.2.1 Variant 1 – Integration of IndoorGML and OpenStreetMap data models (IndoorGML → OSM)

In this approach, the main focus is the conflation of the IndoorGML and OpenStreetMap data models. The purpose is the migration of IndoorGML indoor data to OpenStreetMap data, in a manner that maintains as much of the information contained in IndoorGML as possible, including not only the network information (edges and nodes of the building) but also any semantic information derived from the MLSEM that is necessary for contextual constraints.

The reason for attempting this approach is that OpenStreetMap allows for immediate access to outdoor spatial information (maps, road networks, etc.) which can be used for the outdoor navigation segment.

Additionally, OpenStreetMap already offers a host of options in regards to online and offline routing engines, which can be easily set-up and hosted for the creation of a web interface allowing navigation in the available datasets. Apart from the OpenStreetMap network, many of the routing engines support easy integration of GTFS (General Transit Feed Specification) data, enabling calculation of transit routes as part of the multi-modal navigation options.

Migrating the IndoorGML data to OpenStreetMap would allow us to treat indoor and outdoor space as one seamless dataset, enabling routing via one of the available OSM routing engines. However, one important issue that may arise with this conflation is the loss of data when migrating from the combined network and space cell based approach of IndoorGML to the node-edge network structures typical in OpenStreetMap.

The other challenges with this approach appear mainly in the conflation process, since IndoorGML and the MLSEM are focused on the multi-layer description of indoor spaces to represent context (e.g., traversability in different modes of movement (flying, driving, walking), sensor spaces, etc.). On the other hand, OpenStreetMap does implement constraints on the edge/node network via the use of specific restriction tags, allowing for the definition of unconditional (e.g., wheelchair access) and conditional (e.g., opening hours) restrictions. However, as there is no multi-layer representation of the same topographical space, the amount of information that can be directly migrated is limited.

3.2.2 Variant 2 – Integration of IndoorGML and OpenStreetMap data models (OSM →IndoorGML)

In this approach, the conflation of the IndoorGML and OpenStreetMap data models would be again attempted. The difference to the previous process is the direction of data migration, as we would attempt to represent OpenStreetMap data in the IndoorGML data model.

The reason for attempting this approach is that the IndoorGML is generally more complex, allowing for a richer semantic description of data sets via the MLSEM. If the data is successfully migrated, a routing implementation for IndoorGML could be implemented. As with the previous variant, the main benefit of such an approach is using a single routing solution based on one data model which would include the combined indoor/outdoor data.

However, it quickly becomes apparent that there are multiple challenges to such an approach, since IndoorGML has been very specifically developed for the description and navigation of indoor spaces. As such, integrating outdoor data from OpenStreetMap would require extensive work in adapting and extending the IndoorGML data model, since the most applicable solution in the current IndoorGML implementation appears to be the use of Anchor nodes and external referencing of OSM objects.

Additionally, there is no ready-to-use routing implementation for IndoorGML datasets, which means that additional work would be needed in order to implement one, capable not only of handling the indoor dataset, but also at least providing rudimentary navigation for the externally referenced outdoor OSM data. However, as IndoorGML conforms to the ISO 19100 family of standards, it should meet all the necessary requirements for use with other ISO compliant GIS tools & platforms, such as Oracle Spatial, facilitating the implementation of a routing engine.

With the above in mind, this variant is only mentioned for the sake of completeness and will not be further investigated.

3.2.3 Variant 3 – Integration of IndoorGML and OpenStreetMap data models (OSM & IndoorGML → “Wrapper” schema)

In this approach, we attempt to combine the IndoorGML and OpenStreetMap data models under a single wrapper schema that would allow for the merging of indoor (initially in the IndoorGML format) and outdoor (derived from OpenStreetMap) data. A wrapper schema concept would then be developed, allowing for the seamless import and integration of both IndoorGML and OpenStreetMap data.

It then should be feasible to implement the conceptual wrapper model through a spatial-enabled database (in this approach, Oracle Spatial). Once the wrapper schema containing both data models has been successfully developed, a routing solution would be implemented that would need to utilize the complete wrapper data model, so as to accommodate the combined indoor/outdoor, multi-modal and context-aware routing requirements of the thesis objectives.

The benefit of such an approach is that since both IndoorGML and OpenStreetMap data can be represented and stored in an Oracle database, developing a wrapper schema would enable the creation of a seamless indoor/outdoor dataset, without requiring any modification or simplification of data that takes place as part of the integration processes in Variants 1 and 2. This is a very important benefit, as it allows for the import and export of data from the database wrapper schema without any loss of information compared to the initial IndoorGML/OSM dataset. Additionally, Oracle Spatial has built-in support for routing, which could be possibly also utilized for a navigation implementation based on the wrapped indoor/outdoor data.

The challenges in such an approach mostly focus around the implementation of the wrapper schema, which would have to conform to the ISO 19100 family of standards for geographic information systems, including the semantic and spatial representation of data, as well as Location-based services (e.g., routing). An important element is managing the correct merging of IndoorGML and OSM data, including accounting for any errors or mismatches in the description of the same elements in the initial datasets.

An additional challenge is the implementation of a routing application, possibly driven by the built-in functionality of Oracle Spatial, which would need to meet all the requirements of the thesis objectives, namely providing multi-modal navigation for combined indoor and outdoor datasets, while taking into account any context/semantics driven constraints.

3.2.4 Variant 4 – Application-based integration of navigation results from OSM & IndoorGML

This approach differs from all the previously examined variants, as there is no attempt to merge the IndoorGML and OSM data in a single dataset via conversion or a wrapper. According to this approach, the indoor and outdoor datasets would be examined separately within an application, using an OSM routing solution to calculate the outdoor segment of a route, an IndoorGML routing solution for the indoor segment, and an interface displaying the complete solution.

One benefit of such an approach is that there is no need for modification of the initial datasets, as they are each processed separately by their respective routing engines. Additionally there is increased flexibility in the multi-modal and context awareness aspects of the routing solution. Vehicle/mobility type as well as constraints for context-aware routing can be defined separately for each of the segments, allowing for more fine-tuned control, or provided once, based on the subset of constraints that can be included/ considered in both the IndoorGML and OpenStreetMap models.

The routing aspects for the outdoor segment can be handled by one of the available routing implementations for OpenStreetMap, while a routing solution would need to be developed/implemented for the IndoorGML data.

The main focus and challenges of such an approach is the development of the main application, which would need to provide an interface for the input of data necessary for both route segments (start & endpoint, mode of transport & other constraints), the functional display of both route segments and navigation instructions and most importantly, the correct linkage between the two segments. Such an approach would possibly require some further input from the user, such as definition of preferred building entrance for the indoor segment apart from the origin/destination point, or calculation of the complete indoor/outdoor routing request for all possible building entrances and selection of the optimal route.

This is extremely crucial, as correct routing needs to take into account information that needs to be a part of both data models and correctly matched between them. An easy example is the TUM building, which has multiple entrance points. These need to be matched between the IndoorGML model of the building and its representation in the OpenStreetMap model. Apart from matching, the multiple entrance points need to be accounted for during route calculation, along with their semantic information, in order to provide an optimal solution (e.g., entrance from the closest entrance when using transit, or only through the main entrance during late hours).

While the indoor & outdoor datasets could be used in their original formats in such an implementation, additional linkage would need to be defined between the datasets, in order to correctly connect the external and internal datasets. In order to achieve this, the IndoorGML and OSM data could be stored under a single database allowing for the definition of relations between objects from the two datasets.

4 Analysis of Related Work

The purpose of the following chapter is to provide a detailed look into research and work relevant to the objectives of this thesis. Initially, the selected data models for indoor & outdoor data, namely IndoorGML and OpenStreetMap, will be examined in more detail. A preliminary examination of the capacity for conflation of the two data models will follow, as a necessary step in deciding which of the previously described conceptual processes would be best suited for development of a working implementation. Finally, some of the platforms to be used as part of the conceptual system implementation will be examined in more detail.

4.1 Data models – Introduction

Data models are an essential tool in defining a structure in which sets of data can be organized, in order to facilitate tasks involving said data. These tasks can range from simple storage and exchange of data to complex processes involving the manipulation and/or processing of data that is part of the data model, including the addition and processing of semantic information.

The structure that is defined within a data model enables the storage of data in a consistent manner, allowing for the development of processes and tools which are compatible with the data model and utilize its structure efficiently.

One approach in categorizing data models is the three-level architecture model presented in 1975 by ANSI-SPARC. According to this approach, data models can be categorized as one of three types, External (User), Conceptual and Physical.

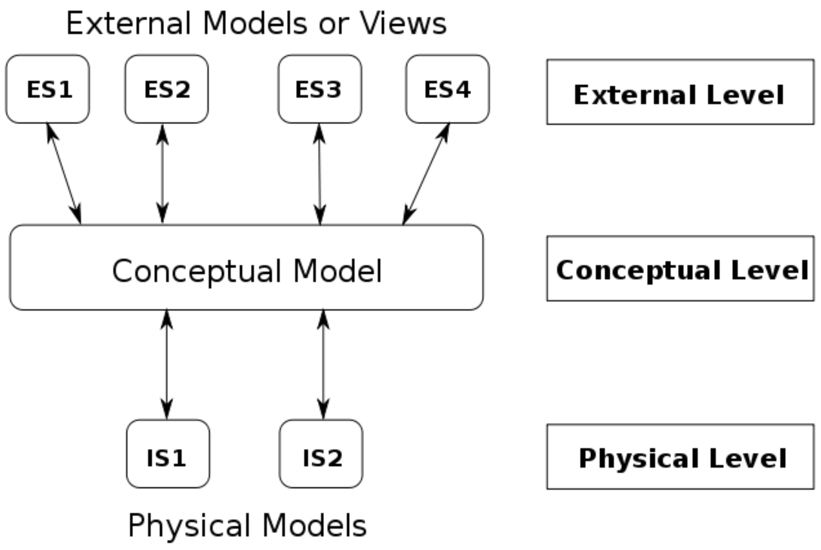


Figure 1 - The ANSI/SPARC three level architecture (West, M., 2011)

The external model defines the data that is available at the end user level. This can cover both the data structure requirements for user input as well as system output towards the user and helps better define the needs and requirements for the complete system, or simply the next level model, which is the conceptual model. According to the three-level architecture, the conceptual model level offers a high level view of the structure, requirements and interaction of data stored within the model. The types of data that are stored are defined in this level, along with their relations and interactions, hierarchical or other. This provides a global view of the model's functionality and capabilities, without providing specifics for its implementation within a database or file system. The later information about the actual storage of data within a database or file is covered by the physical level model, which clearly defines how the conceptual model is implemented in a system, i.e., how data is stored and processed in both the database/file system and related hardware.

Naturally, since 1975, many other approaches for the description of data and data models have been proposed, including additional steps or layers between the end user's needs and the physical implementation of those. One important addition that is common across data modeling architectures is the concept of the logical schema, a level between conceptual and physical models which covers the definition of specific structures within the model, such as tables and columns, classes, functions and even tags. While still platform agnostic, this level allows for a better definition of semantic information and its representation within the data model. It is however often the case that the above logical level is also defined within the conceptual model. **(S. Hoberman, 2014)**

The work within this thesis will initially deal mostly with conceptual level modeling, as this level is the most suited for ascertaining the feasibility of data integration between indoor and outdoor datasets stored in different data models. Naturally, both the external and physical model levels will also be examined, as part of the design of an implementation that will utilize the integrated indoor/outdoor data set for context-aware multi-modal routing purposes. A UML diagram could in turn be produced in order to better map and describe the conceptual model.

One of the challenges of implementing a seamless indoor/outdoor navigation system is the interaction between the available data models for outdoor and indoor maps.

Considering the scope of this thesis, it was decided to use a controlled and verified dataset for indoor data, based on IndoorGML data for a TUM building that was created based on IFC data from the TUM, in **(Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014)**. Further details on the indoor dataset will be provided in the next section, covering the IndoorGML data model.

For the outdoor segment, the use of OpenStreetMap was decided, due to the great availability of data, as well as the extensive documentation and community support, both in regards to the data model itself, as well as the multiple routing implementations that are available. More details on the datasets used will be provided in the relevant section regarding the OSM data model.

4.2 Indoor data model - IndoorGML

IndoorGML is an OGC standard, specifying “an open data model and XML schema for indoor spatial information. IndoorGML is an application schema of OGC® GML 3.2.1. While there are several 3D building modelling standards such as CityGML, KML, and IFC, which deal with interior space of buildings from geometric, cartographic, and semantic viewpoints, IndoorGML intentionally focuses on modelling indoor spaces for navigation purposes” (**OGC, 2014**)

IndoorGML was created based on the need for a standard for the accurate definition of indoor spaces, considering the needs of indoor location based services and differences between an outdoor and indoor dataset. One example is location acquisition, which is nowadays easy for outdoor spaces with the prevalence of GPS on mobile devices.

However, in indoor space, where GPS coverage is not possible, alternatives such as combined sensor measurements (RFID, Bluetooth, Wi-Fi) and sensor space/strength mapping are used. Additionally, while outdoor navigation is relatively straightforward, utilizing road networks with limited constraints, indoor navigation is much more complex due to the complexity of indoor layouts, with building elements like doors, corridors and rooms, in addition to elements for vertical movement such as stairs, elevators and escalators.

IndoorGML manages the above complexities by providing a data model which contains the following elements in its effort to accurately cover the constraints specific to indoor navigation **(OGC, 2014)**:

- Cellular space
- Semantic representation
- Geometric representation
- Topological representation
- Multi-Layered representation
- Subspacing
- Anchor nodes
- External referencing

The following segments will briefly examine each of the above elements separately. One additional point of note is that, as an OGC standard, IndoorGML is also compliant with the ISO 19100 family of standards, enabling its use for multiple GIS applications. One such example is the confirmed capability to transfer data from other typical Building Information Model formats such as IFC or from other OGC formats such as CityGML LoD4 to the IndoorGML data model, limiting the loss of semantic information available in the input models **(Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014).**

4.2.1 IndoorGML - Cellular space

Cellular space is a key feature differentiating IndoorGML from other standards. Instead of a thorough description of typical building elements (walls, floors, roofs), IndoorGML focuses on the indoor spaces enclosed and defined by these elements.

These spaces are defined via the use of Cells, derived from the decomposition of the indoor space to its smallest organizational or structural units. With that in mind, each cell is unique, allowing for common boundaries but no overlap. A unique identifier allows for the referencing and identification of cells.

The information of each cell can be complemented by the addition of further information, such as coordinates, semantic information, a geometric representation and the relevant relations, be it topological relations to other cells, or relations of cell to elements in different layers (a concept further examined alongside the Multiple Space-Event Layer Model) **(OGC,2014)**

4.2.2 IndoorGML - Semantic representation

In the scope of this master thesis, we focus on examining the feasibility and effectiveness of a context aware system. However, there are certain prerequisites for effective context awareness. More specifically, "Context-aware navigation systems require the use and integration of an appropriate indoor spatial model that satisfies application and structural constraints" (**Afyouni, I., et al., 2014**)

In order to define and utilize such constraints in our data model for indoor space, the addition and use of semantic information is required. As we mentioned IndoorGML has a cell-based approach for the subdivision of indoor space. However, semantics are also a driver of the definition of cells. For example, when defining semantics for topographical space, cells may be defined by the subdivision of a larger area into separate rooms. For a sensor space, each cell could represent the coverage area of an RFID card reader or a Wi-Fi network. Semantic elements such as function (e.g. living room, dining area, etc.) or access constraints (public, employee only, maintenance only) can be used for the definition of cells.

From the above approach, we can see that semantics play a significant role in IndoorGML, as they are an intrinsic driver of space subdivision and definition. Additionally, this allows for their use in defining the relations between elements, be it connectivity between cells or interrelations with sub or parent classes of elements. This is particularly useful for the quick identification and grouping of cells by their semantic properties, when examining a complete and complex indoor space dataset.

Navigation within IndoorGML is also driven by the inclusion of rich semantic information, as it is precisely the distinction between navigable and non-navigable spaces, along with the connections between the cell elements that allow for the definition of a route. Any additional semantic information, such as sensor spaces, locomotion or access constraints, can be also defined within the model and assist in creating a more semantically rich system, capable of providing precise and context-aware routing information.

The above defined semantic enrichment of the data model is also achieved via the use of the Multi-Layered Space-Event Model (MLSEM).

4.2.3 IndoorGML - Geometric representation

As we have previously mentioned, the thorough representation of geographical space and its elements is not one of the main focus areas of IndoorGML. Based on this, an IndoorGML dataset may contain no geometry information and still offer a usable dataset for navigation, semantic and topological classification of indoor spaces. Alternatively, the geometry of cells can be described via external references to elements in other datasets, such as CityGML LoD4 building models. Finally, geographic representation is also possible within the IndoorGML data model, covering both 2D and 3D spaces, as per the ISO19107 Spatial Schema. Usage of the GM_Solid or GM_Surface, for 2D and 3D spaces respectively, can provide geometry information as part of an IndoorGML dataset.

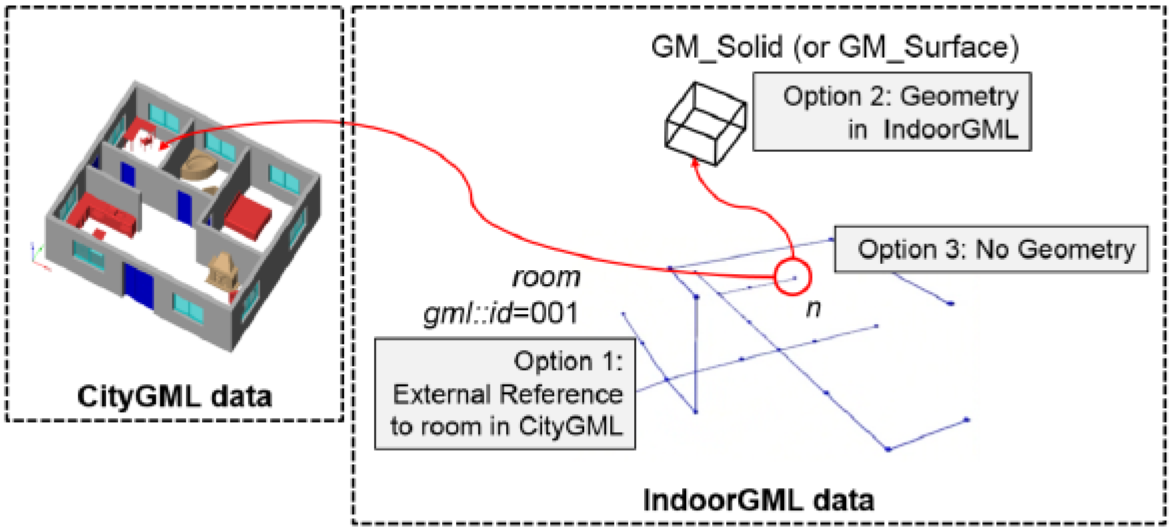


Figure 2 - Three options to represent geometry in IndoorGML (OGC, 2014)

One final point of note is that external referencing can be used alongside the built-in basic geometry definition capabilities of IndoorGML when a complete and matching external dataset is not available or preferable.

4.2.4 IndoorGML - Topological representation

Topological representation is essential in defining connectivity of spaces, however it is not implicitly defined in cellular space. As the geometrical representation of spaces (which could be leveraged in order to establish basic relations between elements) is not a requirement for IndoorGML datasets, the topology and relations between cells must be defined separately, in order to have a usable dataset. This establishment and representation of relations between cells is achieved by use of a relational graph called Node-Relation Graph (NRG).

NRG are created from indoor space through the application of the Poincaré duality. According to this principle: "a k-dimensional object in N-dimensional primal space is mapped to (N-k) dimensional object in dual space. Thus solid 3D objects in 3D primal space, e.g., rooms within a building, are mapped to nodes (0D object) in dual space. 2D surfaces shared by two solid objects is transformed into an edge (1D) linking two nodes in dual space." **(OGC, 2014)**

As a result, 3D objects in indoor space such as rooms are transformed to 0D objects (point nodes) in the dual space of the NRG, while 2D surfaces, boundaries between 3D spaces are transformed to 1D edges, connecting the nodes. This allows for the generation of an NRG indicating the adjacency relations of nodes (which also coincide with cells in the cellular space), called the adjacency graph.

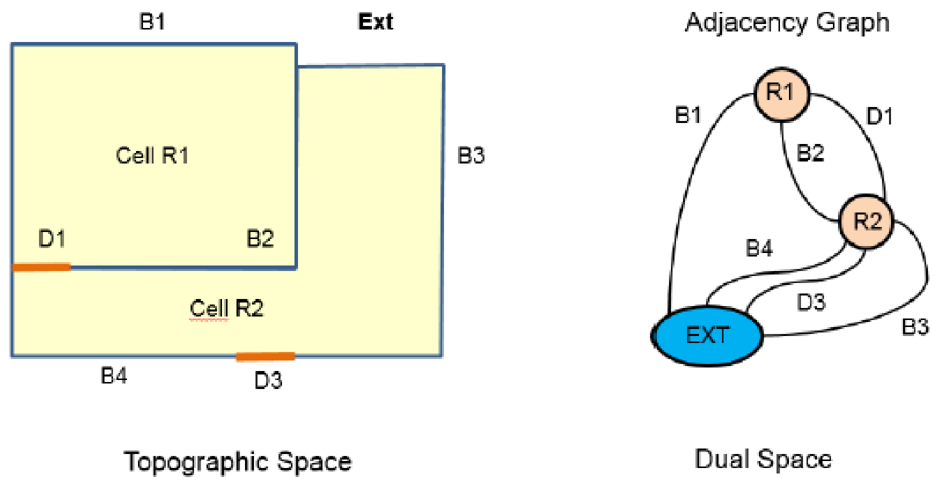


Figure 3 - Topographic and Indoor Space

One important note is that basic topological relations such as adjacency require no semantic information for their definition. However, the addition of semantics and further definition of more specific graphs allows for a very useful tool for routing and navigation implementations. As a first example, the addition of semantic information differentiating the surfaces of the topographic space between navigable (doors, e.g., D1,D3) and non-navigable (walls, e.g., B1,B2,B3,B4), can lead to the derivation of a connectivity graph, indicating only the edges (surfaces) which allow navigation through the nodes (rooms/spaces).

With the addition of further semantic information, e.g. wheelchair accessibility based on the dimensions of the transfer surfaces (doors), we could derive an accessibility graph. If we assume that based on the added semantic info door D1 is too small to fit a wheelchair through, our accessibility graph would look as follows, indicating that cell/room R1 is not accessible via wheelchair:

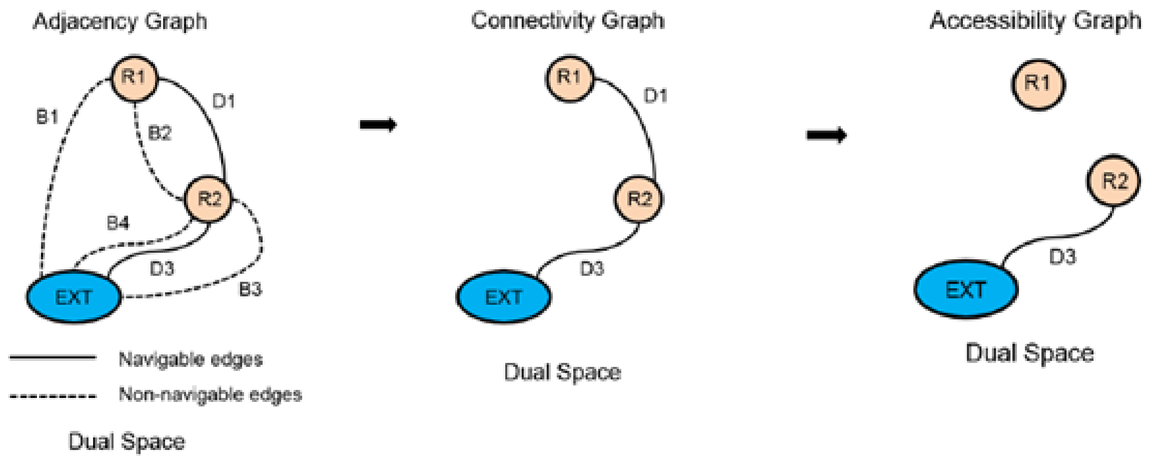


Figure 4 - Derivation of connectivity and accessibility graphs from adjacency graph (OGC, 2014)

As discussed, addition of geometry is not obligatory in IndoorGML. This is easier to accept when considering the capabilities enabled simply through the use of topological graphs, which suffice for the purposes of navigation, including constraints. In such cases, the NRG is called a logical NRG. However, basic geometrical properties can be added to elements of the NRG via GM_Point (nodes) and GM_Curve (edges), as per ISO 19107. NRGs that include geometric data are called geometric NRGs. **(OGC, 2014)**

One final note regarding the topological / network representation is that boundaries such as walls, windows and doors can also be depicted as cells, in what is called the thick wall model. Naturally, this changes the adjacency graph, as the doors and walls now also are 3D cell space objects which transform into nodes in the dual space of the NRG.

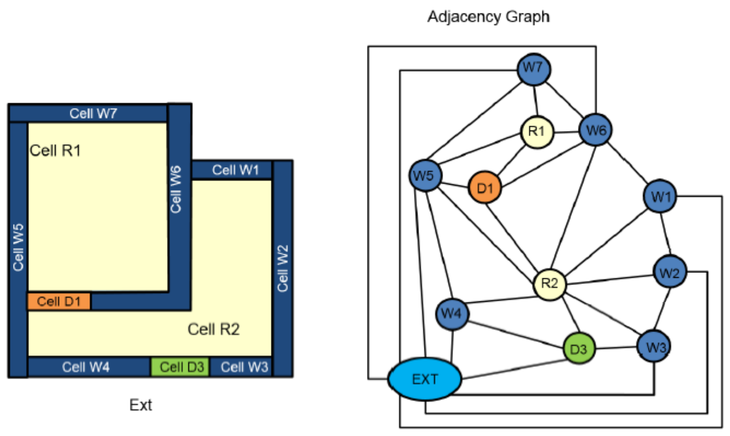


Figure 5 - Thick wall model and corresponding adjacency graph (OGC, 2014)

4.2.5 IndoorGML - Multi-Layered Representation

IndoorGML is based on a flexible and extendable model, called "Multi-Layered Space-Event Model (MLSEM)" (**Nagel, C., 2014**). The MLSEM allows for multiple layers of data, not only providing topographical information and connections between spaces, but allowing for the overlay and connection of multiple thematic layers, be it sensor spaces (e.g. Wi-Fi or BT coverage) or other layers, providing useful semantic information and connecting it to the actual topography of the building. This semantic information can be utilized in order to provide the constraints necessary for a context-aware system.

The MLSEM is an extension of the Structured Space Model: The concept is based on the subdivision of a space by Geometrical and Topological space, alongside primal & dual space. Each layer can be represented geometrically or topologically in an ISO 19107 compliant definition of primal space. Via the Poincaré duality, the ISO 19107 primal space is transformed to dual space Node Relation Graphs (NRGs) which can be geometric (containing coordinate information for the states/points and transitions/curves) or logical (only topology info, no coordinates).

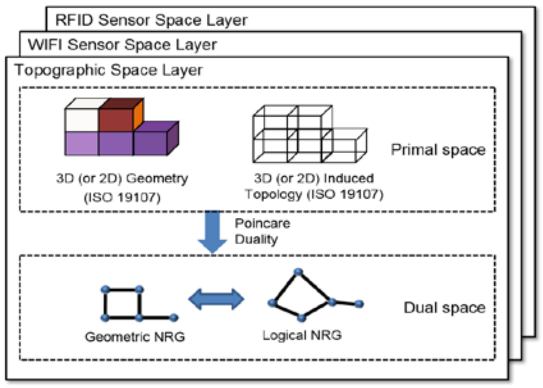


Figure 6 - Multiple Space-Event Layer Model (OGC, 2014)

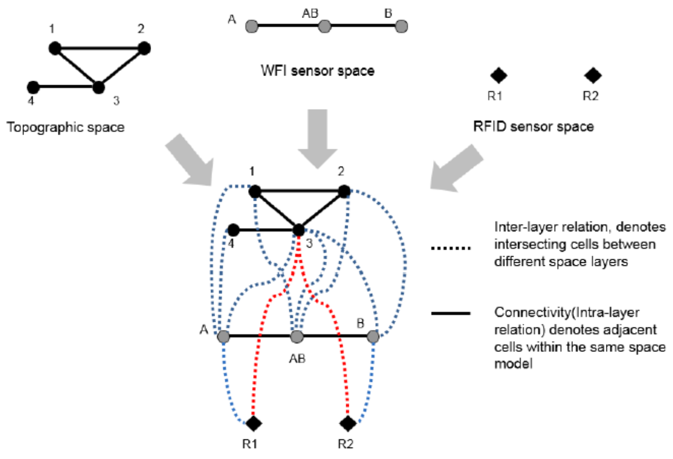
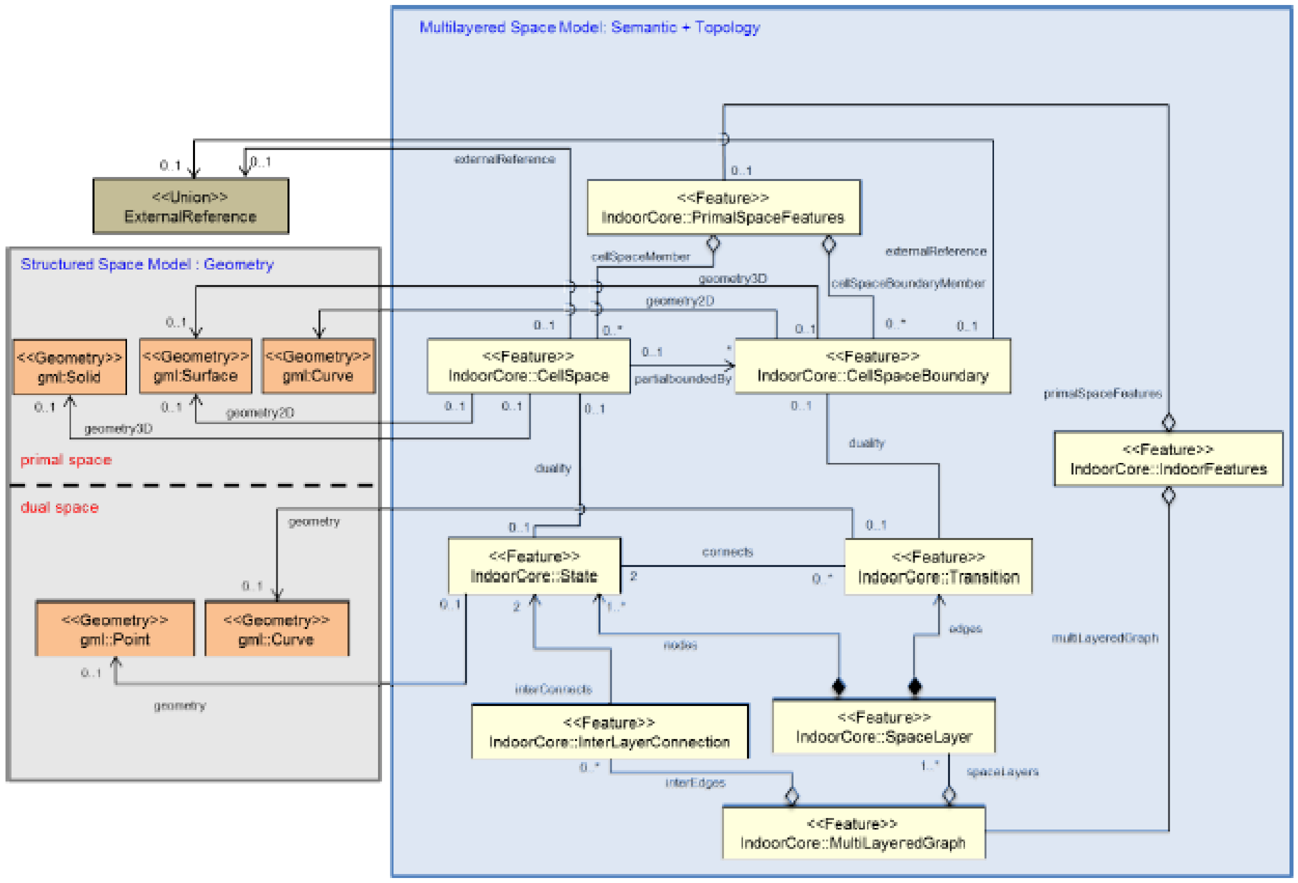

1
2
4
3
Topographic space
A
AB
B
WFI sensor space
R1
R2
RFID sensor space
Inter-layer relation, denotes intersecting cells between different space layers
Connectivity(Intra-layer relation) denotes adjacent cells within the same space model

Figure 7 - Multiple layer relations (OGC, 2014)

The MLSEM allows not only the addition of multiple layers of semantic information to the existing cellular space, but also the establishment of inter-layer relations between multiple layers, allowing for extensive fine-tuning of queries for data within the model, and the parallel examination of multiple constraints (e.g., a Wi-Fi coverage map for IT combining topographic and Wi-Fi sensor maps, or identification of a location in the topographic space by the combination of returned values from the Wi-Fi and RFID sensor space).

As an OGC standard, IndoorGML follows the concept of modularization, utilizing a base Core Module which is further extensible by additional thematic modules. The MLSEM, being the foundation of IndoorGML, is defined via the IndoorGML Core Module, which also provides the Structured Space Model for ISO 19107 compliant geometric representation (utilizing GML 3.2.1 geometry classes). A UML diagram of the Core module is depicted below, structured in order to highlight the SSM (geometry) and MLSEM (topology) segments of the module:



Multilayered Space Model: Semantic + Topology
<<Union>> ExternalReference
Structured Space Model : Geometry
<<Geometry>> gml:Solid
<<Geometry>> gml:Surface
<<Geometry>> gml:Curve
primal space
dual space
<<Geometry>> gml:Point
<<Geometry>> gml:Curve
geometry3D
geometry2D
geometry
externalReference
<<Feature>> IndoorCore::PrimalSpaceFeatures
cellSpaceMember
cellSpaceBoundaryMember
<<Feature>> IndoorCore::CellSpace
partialboundedBy
<<Feature>> IndoorCore::CellSpaceBoundary
primalSpaceFeatures
<<Feature>> IndoorCore::IndoorFeatures
duality
<<Feature>> IndoorCore::State
connects
<<Feature>> IndoorCore::Transition
nodes
edges
multiLayeredGraph
interConnects
<<Feature>> IndoorCore::InterLayerConnection
<<Feature>> IndoorCore::SpaceLayer
interEdges
spaceLayers
<<Feature>> IndoorCore::MultiLayeredGraph

Figure 8 - UML diagram of IndoorGML's core module (Multi-Layered Space Model) (OGC, 2014)

Currently, the only additional thematic module for IndoorGML is the IndoorNavigation Module, which will be examined further below, as it is essential for the use of IndoorGML in a routing implementation as per the scope of this thesis.

The IndoorNavigation Module extends the Core IndoorGML module by providing additional classes that allow for the characterization of space (State/Cellspace features in dual and primal space respectively) and surface (Transition/CellSpaceBoundary features) features into non-navigable and navigable features. The Navigable features are further defined depending on their function within a routing context. Additional Feature classes are defined for the description of a Route, along with the Route Segments and Route Nodes the route is comprised of.

An example of how space is mapped to the appropriate classes of the IndoorNavigation module is provided below **(OGC 2014)**:

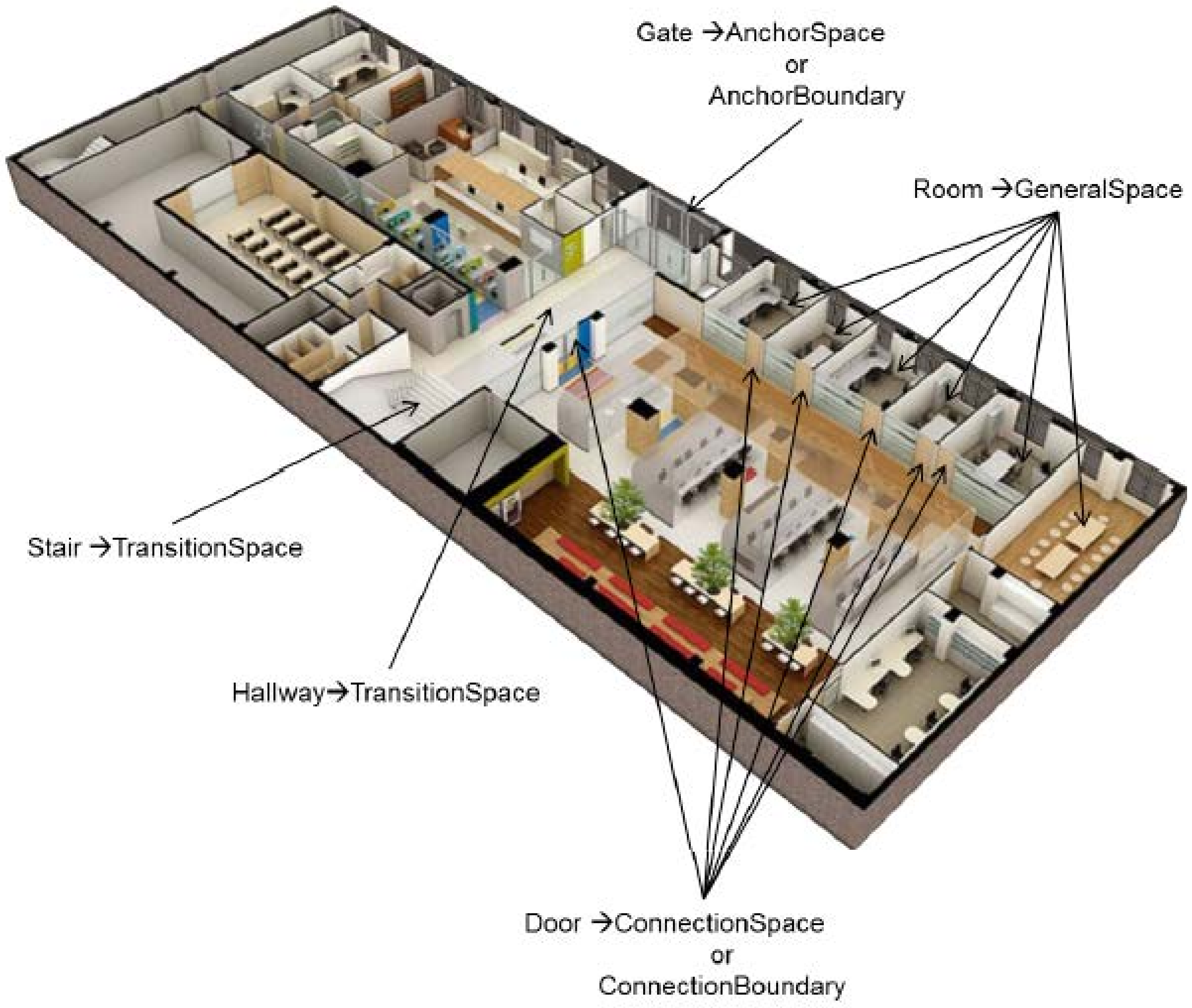
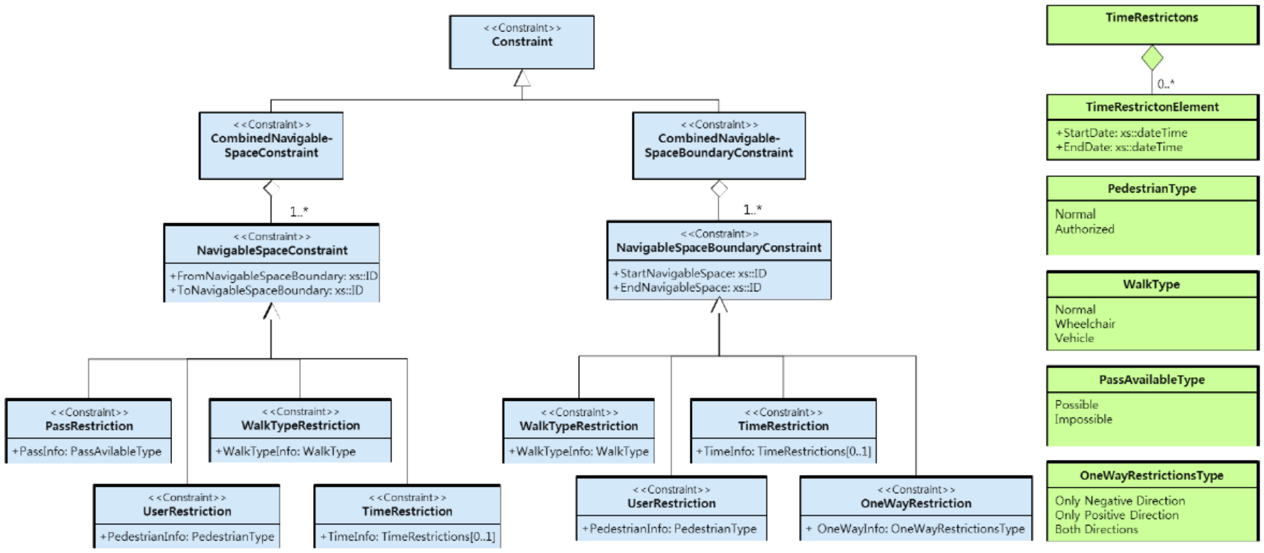


Figure 9 - Indoor space mapped to IndoorGML Navigation module classes (OGC, 2014)

One point of note is that the use of the Space/Boundary variants of the IndoorNavigation module feature classes is also dependent on the selection of model for the definition of indoor spaces. As a reminder, two approaches exist, with the thin wall model considering walls and transitions (doors/windows/etc.) as 2D Surfaces, while the thick wall models considers all elements (indoor spaces and walls/ doors alike) as 3D solids.

In addition to the above feature classes, the IndoorNavigation module provides a conceptual model for the addition of constraints to NavigableSpace and NavigableBoundary features. The constraints currently defined in the conceptual model are time-based constraints, access/ authorization constraints, traversability/ passability constraints, direction constraints (e.g., one way doors as in airports) and movement type constraints (e.g. vehicle, walking, wheelchair).

The UML diagram of the conceptual model for constraints is depicted below:



<<Constraint>> Constraint
<<Constraint>> CombinedNavigable-SpaceConstraint
<<Constraint>> CombinedNavigable-SpaceBoundaryConstraint
1..*
<<Constraint>> NavigableSpaceConstraint
+FromNavigableSpaceBoundary: xs::ID
+ToNavigableSpaceBoundary: xs::ID
<<Constraint>> NavigableSpaceBoundaryConstraint
+StartNavigableSpace: xs::ID
+EndNavigableSpace: xs::ID
<<Constraint>> PassRestriction
+PassInfo: PassAvilableType
<<Constraint>> WalkTypeRestriction
+WalkTypeInfo: WalkType
<<Constraint>> UserRestriction
+PedestrianInfo: PedestrianType
<<Constraint>> TimeRestriction
+TimeInfo: TimeRestrictions[0..1]
<<Constraint>> OneWayRestriction
+ OneWayInfo: OneWayRestrictionsType
TimeRestrictons
0..*
TimeRestrictonElement
+StartDate: xs::dateTime
+EndDate: xs::dateTime
PedestrianType
Normal
Authorized
WalkType
Normal
Wheelchair
Vehicle
PassAvailableType
Possible
Impossible
OneWayRestrictionsType
Only Negative Direction
Only Positive Direction
Both Directions

Figure 10 - Conceptual model of indoor navigation constraints (OGC, 2014)

The navigation constraints model will also be examined further later in this document, as one of the main points for data integration between IndoorGML and OSM data and a requirement for the implementation of a context aware routing system, as per the thesis objectives.

4.2.6 IndoorGML - Subspacing

IndoorGML allows the subspacing of indoor spaces via decomposition of a cellular space and its node representation in a dual space NRG to multiple sub-nodes, in order to better reflect hierarchical structures (e.g., dividing a corridor by segments better matching the connected door & room nodes. This is achieved in the MLSEM by the creation of a new subspacing NRG which contains the new NRG created by the subspacing, along with specific inter-layer connections showing the relations between the initial and new nodes, while all other nodes retain a 1 to 1 inter-layer connection. Below is an example of subspacing of a corridor (node n_6 to $n_{6\text{-}1}$ & $n_{6\text{-}2}$):

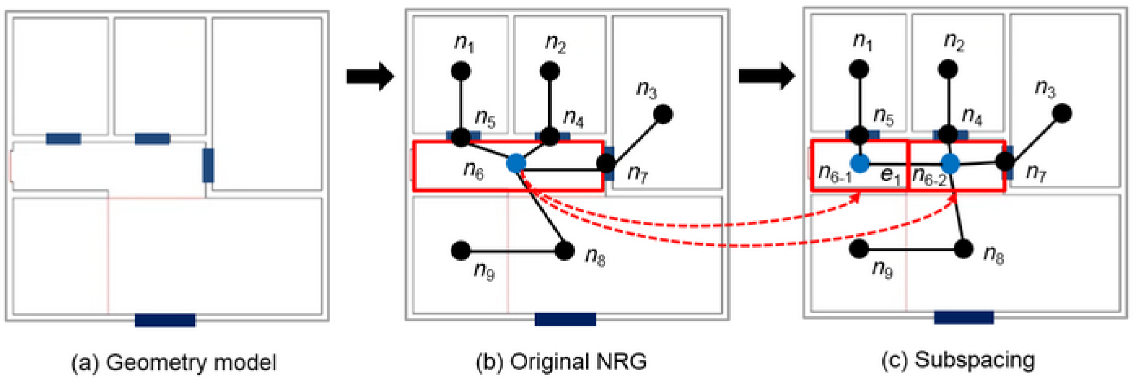


Figure 11 - Example of subspacing (OGC, 2014)

4.2.7 IndoorGML - Anchor nodes

One of the features of IndoorGML that greatly enable interoperability of IndoorGML with outdoor datasets is its support for bidirectional external references via the Anchor Node (usable via the AnchorSpace and AnchorBoundary feature classes of the Indoor Navigation module).

The Anchor node serves multiple purposes, not only allowing for an external reference to outdoor datasets (e.g., via mapping of one or many anchor nodes to the corresponding "building:entrance" tags of a building in OSM), but also by providing the necessary transformation parameters in case a different coordinate system is used between the indoor and outdoor data sets. These parameters are the following: i) rotation origin point (x0, y0, z0) ii) rotation angles (α, β, γ, along x, y, and z-axis), iii) rescaling factor (sx, sy, sz) and iv) translation vector (tx, ty, tz). **(OGC, 2014)**

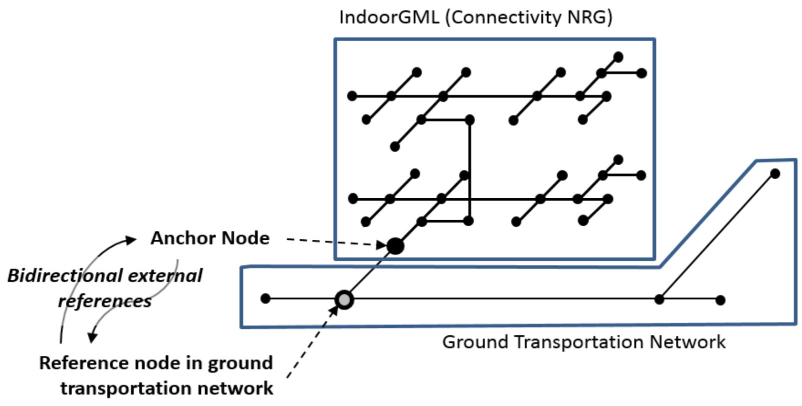

IndoorGML (Connectivity NRG)
Anchor Node
Bidirectional external references
Reference node in ground transportation network
Ground Transportation Network

Figure 12 - Indoor GML Anchor Node (OGC, 2014)

4.2.8 IndoorGML - External referencing

As mentioned previously, IndoorGML has extensive support for external referencing. Considering the scope and focus of the IndoorGML data towards topological data and network graphs with the purpose of navigation, this capability allows users to quickly create an IndoorGML dataset that is data-rich by supplementing it with external references to industry specific formats such as IFC or OGC standards such as CityGML. One point of note is that the use of external referencing is optional and dependent on the availability of matching datasets. Additionally, only one external reference is currently supported per IndoorGML feature.

The above section covers the initial review of the IndoorGML data model and its features. Some further analysis of certain features will follow while reviewing IndoorGML's potential for conflation with OpenStreetMap data. While we will review parts of the IndoorGML and OSM data model as necessary in the following sections, complete UML diagrams of the IndoorGML modules and sample OSM XML code are also included in Appendix I of this document.

4.3 Outdoor data model - OpenStreetMap (OSM)

OpenStreetMap is a free and editable global map, built from crowd sourced and public domain data, available to any user based on an open content license. Since its foundation in 2004, OpenStreetMap has become a huge resource for openly available mapping data by utilizing crowd-sourced and public-domain data. With more than 1.9 million registered users **(http://wiki.openstreetmap.org/wiki/Stats, Accessed February 2015)**, OSM has in many cases become an alternative to commercial providers of mapping data.

This section will provide an overview of the OpenStreetMap data model, starting with the base elements, schema and data sources for OSM, an overview of tagging in outdoor datasets, a brief look into the advances in indoor mapping with OSM and a review of some of the available routing solutions and their pros and cons. The selected tools for our implementation will be further examined in section 6.

Despite no official conceptual model being available, the basis of the OSM data model is fairly simple, utilizing the following base elements (also referred to as data primitives):

- **nodes** (point elements)
- **ways** (line elements defining linear features or boundary edges)
- **relations** (relational elements connecting all element types)
- **tags** (free tagging system defining attributes of elements and their values)

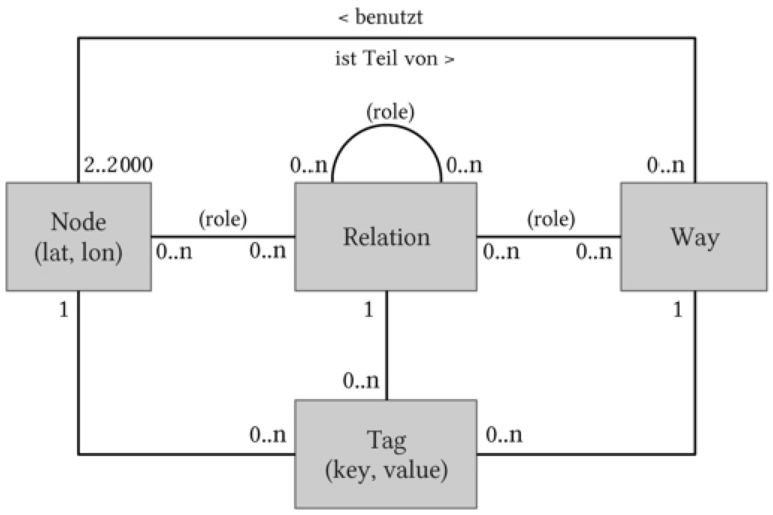


Figure 13 - Simplified OSM data primitives class diagram (F. Ramm, 2010)

Nodes in OSM are used to represent point features. These can be stand-alone features or part of a line-edge (node-way in OSM elements) network. Nodes include an id and coordinate values and are supplemented with information via the use of tags (further describing the exact nature of the point feature and providing attribute values) and relations, connecting the node to other elements.

Ways represent linear features (edges) in OSM. Ways may represent a single line feature connecting two nodes or be part of a more complex feature connecting up to 2000 nodes, e.g., providing the boundaries to a polygon, defining a subway line (by connecting the nodes representing the line's stations), etc. It is important to note that the use of ways to represent a boundary is the same for an area (e.g. the polygon outline of a square) and a loop (e.g. a walking path circling said square). Such cases, where the first and last node in the way are the same, are called closed ways and can be differentiated by the tags and relations used to further describe them. Multiple way features can be combined via the use of relations in order to describe complex polygon features (e.g., more than 2000 nodes or areas with "holes").

Relations are a data element providing relationship info between other elements (nodes, ways or other relations). There are multiple types of relations, defined via the use of tags and their content. The "type" tag usually describes the nature of the relation, with other associated tags providing further data or attributes. Relations may be collections of elements, e.g. a list of nodes and ways forming a bus route, along with the route's information (bus number, line name, etc.). Relations can also cover restrictions, for example in the case of turn restrictions between ways. Additionally, as we mentioned before, relations can express combinations of multiple ways for the description of complex polygon features. These are only some examples, as essentially most complex data structures containing multiple elements are defined via relations. **(OSM Wiki, 2015)**

Along these basic elements exists a very extensive set of tags which can further define all aspects of an element. Tags consist of a key, describing the category of information or feature to be defined, and values, providing the specific options available under the defined key. While this does allow for great flexibility and easy extension of the data model, it also has some concrete drawbacks considering the lack of standardization and arbitrary use of tags. Taginfo, a project tracking the use of tags in the main OpenStreetMap database, lists over 53 thousand distinct keys, with over 74 million distinct values (https://taginfo.openstreetmap.org/, figures from May 2015).

As an example, the path on the right can be tagged as follows (**OpenStreetMap, 2015**):
highway=cycleway
foot=yes
tracktype=grade2 (unpaved track)
horse=no
motor_vehicle=no

or, alternatively:
highway=path
foot=yes
bicycle=yes
horse=no

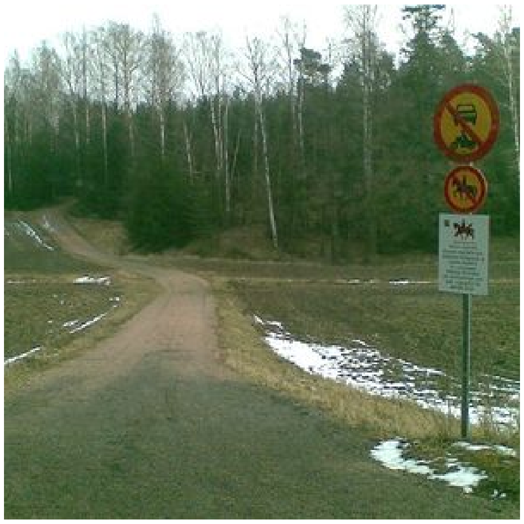


Figure 14 - OSM Highway example image (http://wiki.openstreetmap.org/wiki/Highway_examples)

Considering the above figures, as well as the arbitrarily defined elements, it is apparent that while OSM allows for the easy definition of new key and value pairs for e.g., the development of a custom applications, it is often hard to define a uniform approach for handling larger datasets and creating applications that will correctly parse larger or more complex OpenStreetMap datasets. However, certain tag assignments are more established, and there are certain efforts for the codification/ establishment of best practices, which also include tagging for indoor mapping. The current approaches for indoor & outdoor tagging standardization will be reviewed in more depth later in this document.

4.3.1 OSM Schema

OSM data is typically available in XML file format. Typically, XML files are accompanied by an XSD (XML Schema Definition) file, outlining the expected content of an XML file in regards to elements, attributes, parent/child relationships between elements, data types and default or fixed values for elements and attributes **(W3C, 2015)**

However, unlike other XML-based data models (e.g. IndoorGML, CityGML, etc.), OpenStreetMap has no official XSD Schema, due to the ever changing content of OSM elements and the issues arising when setting pre-requisites for expected element types or their attributes, as these vary depending on the selected data-set, its source and how it was created or extracted from the main OSM database. An OSM XML example is provided in Appendix I.

As some applications require an XSD Schema in order to process OSM data, some example XSD files are in circulation, but they are usable under specific circumstances (e.g., only for parsing data retrieved from a specific OSM API version) and may not always be compatible with the XML datasets examined. **(OGC, 2015)**

4.3.2 OSM Data Sources

OSM data in the standard XML format is directly retrievable from the main OSM database via the OSM API. Apart from direct usage of the API, users can also export data from the Openstreetmap.org site, via the export tab. A bounding area is set based on the current map view, which is further modifiable by the user. However, this method is only available for smaller areas / datasets. OSM editing clients such as JOSM also allow for direct download of data from the main OSM database, with similar limitations regarding the size of the area that can be downloaded.

In addition to the standard OpenStreetMap API, the Overpass API also allows read-only access and querying to the main OSM database, enabling advanced queries to the database and export of queried data via a comprehensive query language (Overpass QL). There are several public instances serving data via the Overpass API. However, limitations regarding size of the dataset still exist, along with limited support for querying and retrieval of historical data.

For larger datasets, a snapshot of the entire main OSM database, also known as planet.osm is available on a weekly basis via several mirror sites. These planet.osm XML files can be extremely large (0.5TB) and are rarely used. Instead, two typical formats of compressed OSM data are preferred, namely PBF files or BZ2 compressed OSM XML. Most OSM tools directly support these formats, which offer a dramatic reduction of size compared to the original XML data (42GB for BZ2 and 28.8GB for PBF, compared to 576GB for an uncompressed XML planet.osm file). **(OSM Wiki 2015)**

Additionally, there are several providers of extracts, which are smaller datasets, ranging from country to city levels. Extracts are available both freely or paid, depending on the provider, and in some cases, the extract format. The update frequency of extracts also varies, with several providers offering daily updates of their extract datasets. Typically, extracts are also available in XML (compressed & uncompressed) and PBF, however some providers have options for ESRI Shapefiles, navigator vendor specific files (Garmin, Navit, etc.), and other formats. Finally, apart from location based extracts, some providers also have the option of thematical extracts (e.g. WeoGeo Market). Two sources of OSM extracts data for Bavaria & Munich are Geofabrik and Mapzen (http://download.geofabrik.de/europe/germany.html and https://mapzen.com/metro-extracts/ respectively).

4.3.3 Tagging of features in OSM

As mentioned previously, OSM data is based on the combined use of simple elements (points, ways and relations) and tags, key-value pairs which help define these elements, so that they represent features in the real world.

While OSM has a free tagging system, allowing for the definition and use of arbitrary tags, the need to establish a commonly accepted and consistent manner of tagging features has led to the creation of an informal but comprehensive list of tags used for various map features. This list is maintained within the OpenStreetMap Wiki, offering a solid basis for defining real world features via applicable tags.

The OSM communities from various countries have also established some country-specific tagging rules which help improve consistency, while also allowing for the addition of information that makes sense within the regional/national context.

Each one of the commonly accepted feature tags has further documentation in the OSM Wiki, allowing for the better understanding and use of the relevant tags. Of particular interest in our case are the tags/features for Buildings, Restrictions and Indoor mapping.

4.3.4 Buildings in OSM

Buildings are described in OSM typically via use of a building footprint (closed way) or even a point node, in case an outline is not available due to poor building information availability. The “building” key is used, followed by the type of the building. In its simplest form, “building=yes” can be used, when the function of the building is unknown. The “building=yes” key/value combination actually accounts for 84.16% percent of the buildings defined in the main OSM database **(Taginfo, retrieved May 2015)**. Otherwise, there is an extensive list of commonly used values, also available via the Taginfo tool, as well as in the OSM Wiki.

However, with a free tagging system as in OSM, a user is not limited to certain values for the building type and can use values based on industry standardized typologies, if that facilitates interoperability of the defined OSM data with a specific application. In any case, the best practices defined for building tagging should still be considered, in order to at least ensure a certain level of consistency with global OSM data.

Other than the building type, important elements for the tagging of buildings are the “addr” (address) and “entrance” keys:
The addr:* keys (typical keys in the addr:* namespace being “addr:street”, “addr:housenumber”, “addr:postcode”, “addr:city” and “addr:country”) help better identify the exact location of a building and enable querying the dataset for specific address values. While the addresses of OSM features can be resolved from longitude & latitude using geocoding services, the addition of address information remains a best practice, as it provides an official value for the address of building features, compared to the estimates often provided by geocoding services.
The “entrance” key is used to identify the entrance of a building. This is an important key as it simultaneously provides the location of the building entrance (multiple entrances are also supported) allows for the definition of further parameters, such as access or accessibility constraints and entrance type (main entrance, service access, etc.). One important note is that the entrance key was proposed as a replacement to the deprecated “building=entrance” tag. However, the later tag still remains in use, with 52 and 823 thousand entries with “building=entrance” and “entrance=*” respectively **(Taginfo, retrieved May 2015)**. With that in mind, when considering querying for building entrances within a dataset, both tags should be examined.

For the 3D representation of buildings, an initial approach is through the combined used of the building footprint, as defined by the initial closed way using the “building” key, and the use of the “height” key and “building:levels” tag:
The “height” key is used to define the height of any feature, by entering the actual height value of the feature (by default in meters, although other units can be also used). For building features, it’s important to remember that the “height” key refers to the top edge of the building, i.e., its maximum height.
This is the main difference to the “building:levels” tag, which simply specifies the number of levels a building has above ground, without accounting for the building roof, which can be

described separately via the “roof” key. Another important note is that “building:levels” does not consider differences in height between levels, so it is better used alongside the “height” key.

An approach for the simple mapping and representation of 3D buildings was proposed in the 2012 2nd 3D Workshop Garching, named Simple 3D Buildings (S3DB). The S3DB proposal provides a tagging method with the purpose of representing more complex 3d geometries. The concept is based on the provision of a building outline via the “building” key, and its division into building parts, in order to represent parts of the building with different attributes (e.g., different number of levels/floors, height, different roof types and materials). As per the S3DB proposal, apart from the closed ways defining the “building” and “building:part” elements, it is also helpful to create a relation grouping the outline and part elements together, in order to facilitate queries and better indicate the hierarchical structure of the building and its parts.

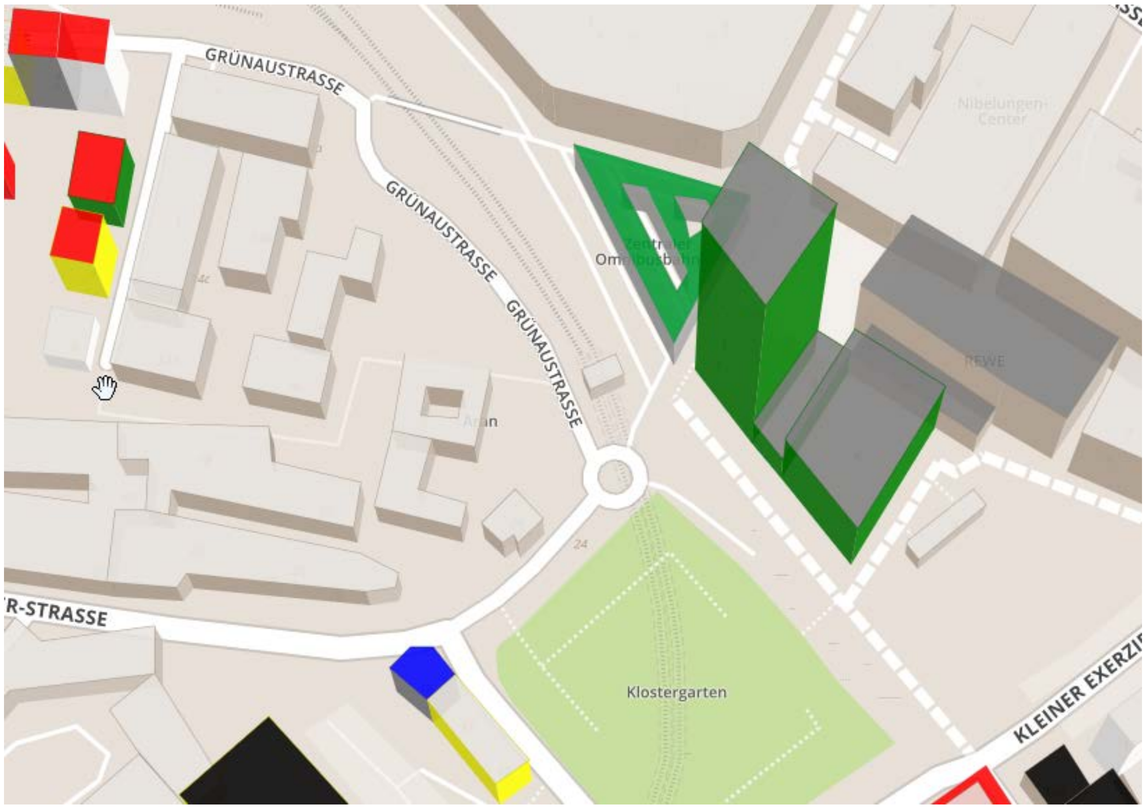


Figure 15 - Representation of buildings in Passau as per the S3DB proposal (OSMBuildings engine) (http://osmbuildings.org/?zoom=17&lat=48.57259&lon=13.45641)

4.3.5 Indoor mapping

There is an increased interest in indoor mapping within the OSM community in the past years, leading to the creation of a separate wiki section and forum, in an effort to propose and establish a tagging approach for indoor spaces.

One of the first proposals that gained significant traction was IndoorOSM. Proposed by the GIScience Group of the University of Heidelberg, IndoorOSM provided a model for indoor tagging, which could be used for mapping, as well as indoor navigation & routing purposes. The concept of IndoorOSM is based on the definition of a 3D Building Ontology covering most aspects necessary for the representation of outdoor and indoor features of a building. Based on this ontology, an extension of the OSM tagging schema is proposed, defining the use of primitives (nodes, ways & closed ways, organized hierarchically under a rigid relations structure) and tags, introducing new keys as well as existing ones where applicable, for a model that could be used as a consistent approach to building tagging. (**Goetz & Zipf**, **2011**) The 3D Building ontology proposed by IndoorOSM, as well as some example OSM XML of a building defined according to the model are available in Appendix I.

However, due to technical issues (e.g., reference of OSM IDs causing data persistence issues, heavy use of relations) IndoorOSM is now considered defunct as a schema **(http://wiki.openstreetmap.org/wiki/IndoorOSM, Accessed February 2015)**, with other proposals based on it such as the Simple Indoor Tagging and Full 3D Buildings (F3DB) schema being reviewed in order to accommodate the increasing request for better indoor mapping capabilities with OSM. **(OGC, 2015)**

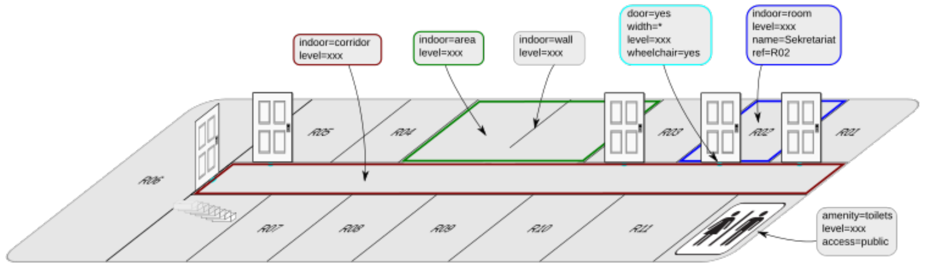


Figure 16 - Sketch showing the tagging of indoor elements in Simple Indoor Tagging schema / IndoorOSM_2.0 (OGC, 2015)

There are already some demo implementations of indoor routing which support partial outdoor navigation (i.e. by utilizing a predefined graph with walking directions to another nearby building entrance node). Such examples show that by using a graph derived from a building (defined as OSM data) and linking it to an existing outdoor graph, we can attempt a seamless indoor/ outdoor navigation implementation. (**Hubel A, 2011**)

All the above approaches are considered alongside the conflation process with IndoorGML, as well as the routing and representation implementation, in order to select the tagging model that is most appropriate for context-aware indoor/outdoor routing, as per the thesis' goals. The tagging model in use will be examined further as part of the implementation, in section 6 of this document.

4.3.6 Restrictions / Constraints in OSM

The OSM free tagging system allows for the definition of restrictions via the use of various key/value pairs supplementing the data primitives' descriptions. Usually, restrictions apply to node and way elements, however relations can also be used for restrictions, either indirectly, i.e., to specify a restriction interaction between elements of the relation (e.g. turn restrictions via used of "relation type=restriction"), or directly, by applying a restriction to all members of the relation.

The most used tag in regards to restrictions is the "access" tag, which can be used in multiple ways to define the legal access for elements such as highways, building entrances, areas, etc.
The access key can be used to define specific transport modes that are allowed or restricted (if that is not implicitly defined by the tags already used to define the described element), right of access (e.g., staff only, private, public, etc.), routing restrictions (e.g., one way streets or even emergency exits/doors), size and speed restrictions and even time restrictions.

There is a multitude of other key/value pairs that are used for constraints, many of which are more specialized/ stand-alone variations of the access tag. One important note is that the same restrictions can be (where it makes sense) applied for on foot or other navigation within indoor datasets.

Additionally, OSM supports the definition of complex conditional restrictions, which are restrictions that apply when a specific criteria is met. Access limits based on time and day of the week are a prime example, while others include speed limits depending on the vehicle type or weight, lane access depending on vehicle usage, etc. The AND logical operator can be used to define even more complex restrictions, while conflicting restrictions are also determined based on a specific algorithm, allowing for the definition of complex restrictions via the use of a specifically ordered or structured approach in defining them. The last point is particularly useful in setting "default" values that apply generally, as the implicit default values when defining highways or other elements may not accurately represent the real world constraints or restrictions.

One example of a conditional restriction is given below **(OSM Wiki, 2015)**

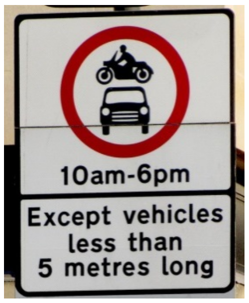


Figure 17 - Conditional restriction example (http://wiki.openstreetmap.org/wiki/Conditional_restrictions)

OSM Tagging:
motor_vehicle:conditional=no @ (10:00-18:00 AND length>5)

Resulting constraint:
Motor vehicles longer than 5 meters are not allowed between 10am - 6pm.

The above free tagging mechanism for restrictions allows for a very targeted and accurate definition of restrictions. One important parameter however, as with other aspects of the free tagging system, is that while free tagging is good for the accurate representation of elements, at least depending on the specific needs of the user doing the tagging, there is not one globally consistent approach in the use of tags for the definition of restrictions. This means that correct parsing of the data can either be limited to the most commonly used forms and values, in order to achieve a relatively generic, but globally operational implementation, or, alternatively, extensively specialize or customize the application, in order to accommodate for the added element definitions and their correct interpretation and use.

This is an issue particularly apparent with routing engines, which rely heavily on the tagging of elements to provide an accurate routing solution and can be problematic, if the tagging is not implemented in the manner that the routing engine is configured to parse.

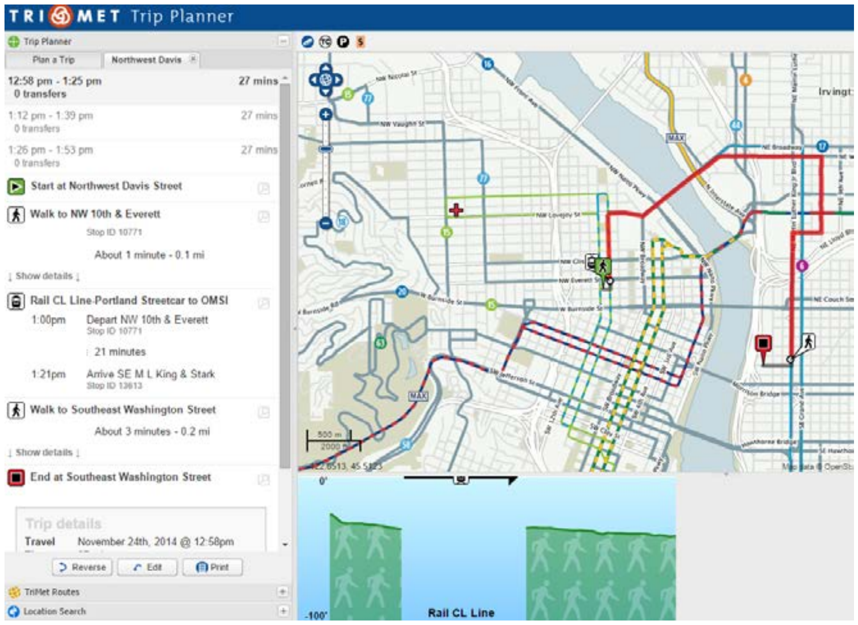
4.3.7 Routing and data representation in OSM

OpenStreetMap data, apart from the typical mapping applications, can additionally be used for navigation from one point to another. There are many options for routing applications utilizing OSM data, available in online and offline configurations for desktops, as well as mobile devices.

There are several key factors differentiating the many available engines, such as supported modes of transport (an important factor considering the multi-modal scope of this thesis), advanced routing options such as turn restrictions, vehicle details & speed, traffic & route avoidance (features that are important when considering context awareness of the routing implementation) and existing/ available user interfaces, which are integral in the ease of use of any navigation system implementation.

A rather extensive list & comparison of online routing engines is available in the OpenStreetMap Wiki pages (http://wiki.openstreetmap.org/wiki/Routing/online_routers, 2015).

One example of an OpenStreetMap routing implementation meeting several of the requirements is OpenTripPlanner (http://www.opentripplanner.org/), a multi-modal trip planner which has built-in support for on-foot movement, cycling (including definition of constraints such as maximum cycling distance & route steepness), transit (with the use of General Transit Feed Specification – GTFS data) and limited support for cars. Additionally, support for indoor features such as elevators is currently implemented, hinting at functionality with indoor datasets. The OpenTripPlanner platform also offers a customizable user interface for navigation including instructions and a 2D visual representation based on the Leaflet library and OSM data, while offering advanced features such as support for geocoding and integration with Google Street View. There are several high-profile OTP deployments that are currently online (http://docs.opentripplanner.org/en/latest/Deployments/).



12:58 pm - 1:25 pm 27 mins
0 transfers

1:12 pm - 1:39 pm 27 mins
0 transfers

1:26 pm - 1:53 pm 27 mins
0 transfers

Start at Northwest Davis Street

Walk to NW 10th & Everett
Stop ID 10771
About 1 minute - 0.1 mi

↓ Show details ↓

Rail CL Line-Portland Streetcar to OMSI
1:00pm Depart NW 10th & Everett
Stop ID 10771
21 minutes
1:21pm Arrive SE M L King & Stark
Stop ID 13613

Walk to Southeast Washington Street
About 3 minutes - 0.2 mi

↓ Show details ↓

End at Southeast Washington Street

Trip details
Travel November 24th, 2014 @ 12:58pm

Figure 18 - TRIMET - Multi-Modal Routing implementation based on OpenTripPlanner (http://trimet.org/)

While OTP offers a great solution for quick deployment of an outdoor routing solution, further customization is necessary for the correct processing of indoor graphs and their connection to the outdoor graph for routing and navigation purposes. The extent to which this is possible will be investigated via the manipulation of the available options for the built-in graph builder, as well as the definition of the indoor dataset.

Additionally, the representation of data based on the default configuration of Leaflet is aimed towards 2D outdoor routing. Features like display of floor overlays or 3D buildings can be added by customizing the viewer, with examples being available from several projects outside OTP (e.g., OSMTools, OpenLevelUp, OSMBuildings) which use the same Leaflet library as a basis for their Map viewers.

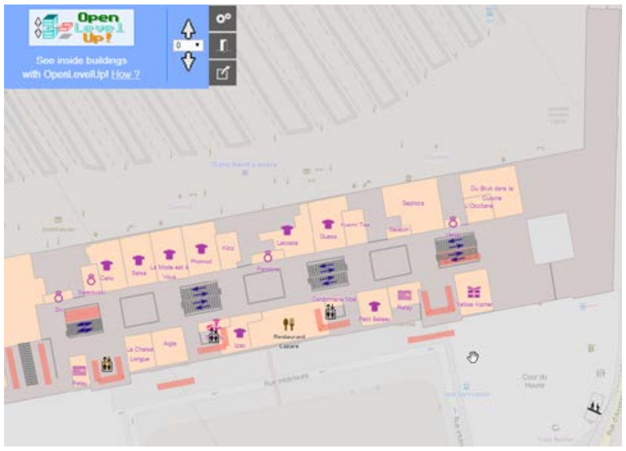


Figure 19 - OpenLevelUp indoor representation example
(http://github.pavie.info/openlevelup/)

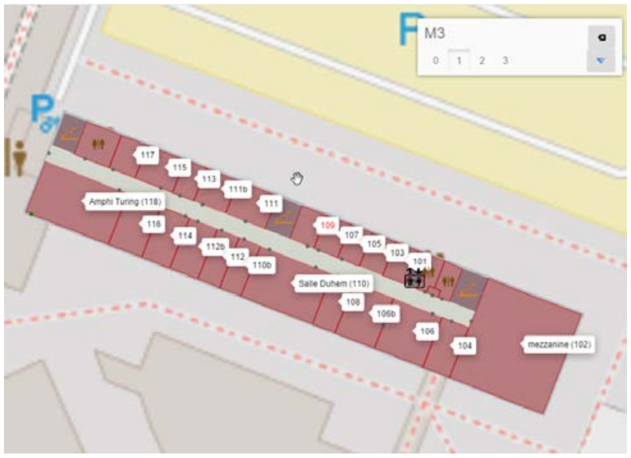


Figure 20 - OSMTools indoor representation example
(http://clement-lagrange.github.io/osmtools-indoor/)

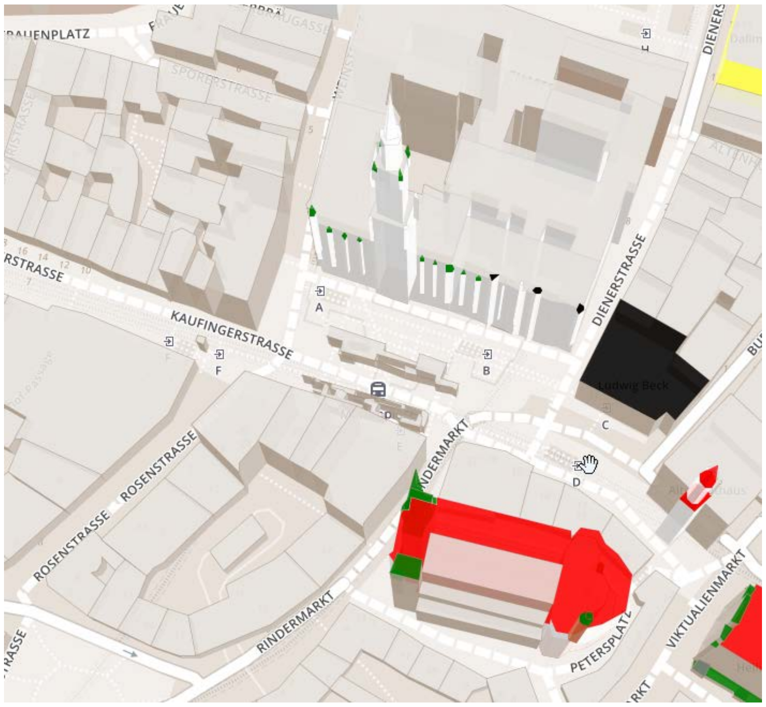


Figure 21 - OSMBuildings 3D Building representation example – Leaflet based implementation (http://osmbuildings.org/)

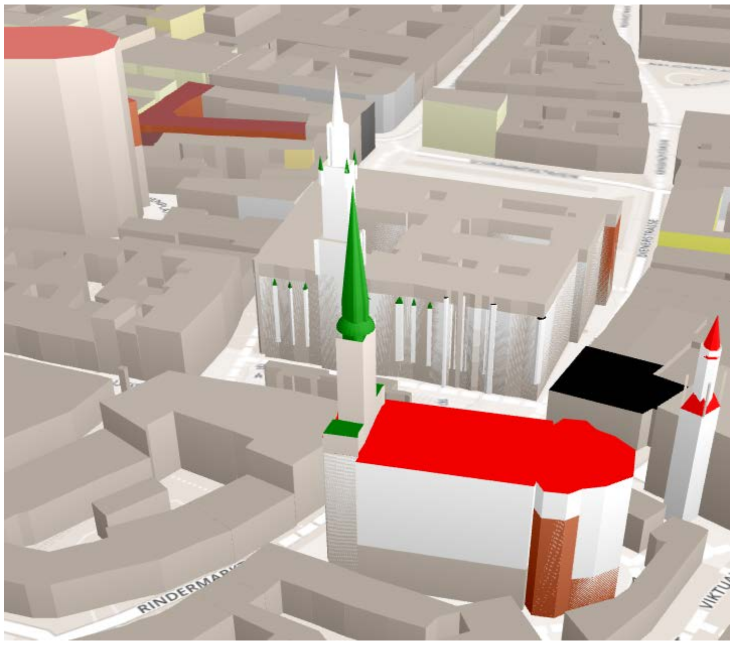


Figure 22 - OSMBuildings 3D Building representation example - GLMap based implementation (http://osmbuildings.org/gl/)

A full 3D virtual globe implementation (such as Cesium or other WebGL-based globes) could be investigated, but is currently outside the scope of this Master thesis, and is only discussed as part of potential improvements in the proof of concept results in subsection 6.11.

5 Conflation Conceptual Model - Indoor GML & OSM

One of the essential tasks encountered in all variants is the conflation or integration of data. Considering the different focus of the two data models as well as the objectives of the thesis, identifying the common features (as well as differences) between IndoorGML and the OpenStreetMap data model is crucial to identifying and enabling interoperability, in order to implement a routing solution capable of dealing with the combined indoor/outdoor space.

In the previous section, we examined the concepts and structure of IndoorGML and OpenStreetMap, the two data models that were considered suitable for achieving the goals of this Master Thesis. In the current section, we will examine how to best achieve the goal outlined above, namely identifying conflation points between the two data models that would ideally enable transferring data accurately, completely and consistently from the complex and specialized IndoorGML model to the more widespread and simple, yet fragmented, OSM model.

As we have mentioned, the following concepts form the core of IndoorGML:

- Cellular space
- Geometric representation
- Topological representation
- Semantic representation
- Multi-Layered representation
- Subspacing
- Anchor nodes
- External referencing

Respectively, OSM is based on a drastically simpler concept, using a flat, single layer approach based on the combined use of basic data primaries. These are the following:

- nodes (point elements)
- ways (line elements defining linear features or boundary edges)
- relations (relational elements connecting all element types)
- Tags (free tagging system defining attributes of elements and their values)

If we attempt an initial identification of conflation points, we would start with how the basic OSM elements are best matched to and covered by the concepts of IndoorGML:

	IndoorGML Concepts							
OSM Elements	**Cellular Space**	**Geometric Representation**	**Topological Representation**	**Semantic Representation**	**Multi-Layered Representation**	**Subspacing**	**Anchor Nodes**	**External Referencing**
Nodes	?	X	X	-	X	?	X	-
Ways	?	X	X	-	X	?	X	-
Relations	-	-	X	X	X	?	-	-
Tags	?	?	?	X	X	?	X	X

Table 1 - IndoorGML Concepts / OSM Elements - initial mapping

We can already see from the table above that by correctly utilizing OSM's basic blocks, a good level of coverage can be attempted for most main elements of the IndoorGML core concept. In addition, the extra data covered by the IndoorGML Navigation Module and Constraints concept can also be covered via the use of tags in OSM, which serve as the main extension method for OSM data.

The transference of these concepts will be first examined in greater detail below, while an actual mapping of IndoorGML Core and Navigation Module / Constraint Concept Features to OSM equivalents will be presented at the end of the section.

5.1 Conflation - Cellular Space

Cellular space, the concept of breaking down an indoor space to small, non-overlapping or intersecting space cells (which may however share boundaries) is a very important concept in IndoorGML, driving the structure of the data model itself. The use of cellular space ensures that data accuracy is high, with no overlapping elements and clearly defined spaces and boundaries. Naturally, this also benefits the geometric, topological and semantic representation accuracy, as they are structured on top of the cellular space model, preventing conflicts between indoor features.

One key difference between IndoorGML and OSM is that the second does not rely on a similar cellular space based approach. Node and Way Features in OSM may intersect and/or overlap without issue, as the model places no restrictions such as those in IndoorGML. Possible reasons for this behavior are the all-purpose use and more generalized content of OSM data, alongside the higher error margin that is generally considered acceptable in volunteered geographic information. As such, no direct implementation of cellular space topology is currently used or readily feasible within OSM.

In order to overcome the possible issues that arise from lack of such a structure in the OSM data model, there is a series of best practice guidelines in regards to uploaded data quality in the OSM Wiki. Additionally, some error checking functionality is available in most OSM editors (JOSM, Potlatch, etc.), as well as via Keep Right, a quality assurance tool for OSM (**http://keepright.at/**). There is also research in attempting to define further means of error checking for the logical consistency of data when working with OSM **(Hashemi, P. and R. Ali Abbaspour, 2015)**.

With the above in mind, the concept of Cellular Space is not transferable to OSM. However, the relevant CellSpace and CellSpaceBoundary features will be examined further when reviewing the topological / geometrical representation concepts and their conflation with OSM.

5.2 Conflation - Geometric / Topological representation

As we have mentioned in the relevant section, IndoorGML offers three options for geometric representation within the data model: i) No Geometry, ii) Externally referenced geometry (e.g. via CityGML dataset references) and iii) ISO 19107 compliant definitions of solid, surface and curve geometries within the IndoorGML dataset. In comparison, OSM, being a general use mapping platform, requires a geometrical representation in its datasets, removing option i). Regarding option ii), as we will see later in the section, support for external referencing is limited compared to its IndoorGML counterpart, not allowing for direct use of geometries stored in IndoorGML as external references (at least not without a separate process for storing the external geometries to IndoorGML's geometry as a preparation stage, as per **(Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014)**. This essentially leaves us only with direct support of the third IndoorGML option for geometric representation (inclusion of ISO 19107 geometry). However, we must also take the limitations of OSM into consideration, as it is mainly a platform geared towards the 2D representation of data.

As a first step, we need to consider the two models for the representation of boundary features in IndoorGML. One concept is the thin wall model, were elements such as doors and walls are considered as linear features, and the other option is the thick wall model, were each wall, door, or similar element is treated as a separate space cell.

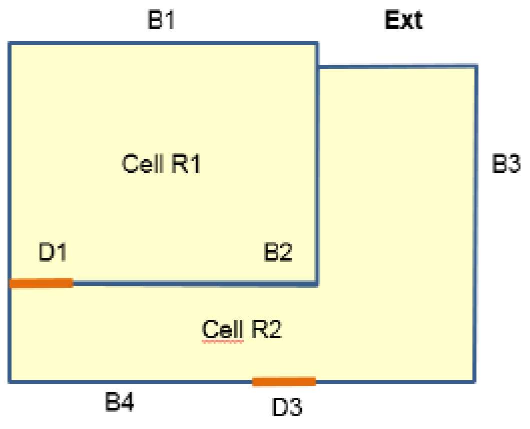


Figure 23- Thin Wall Model (OGC, 2014)

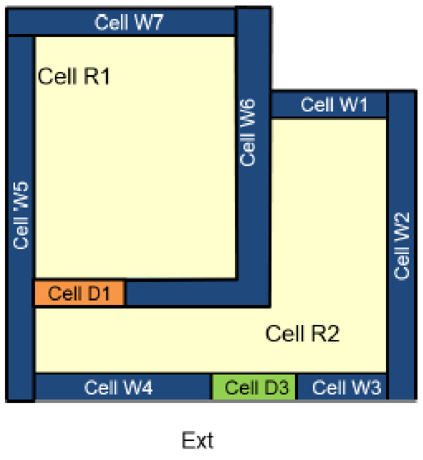
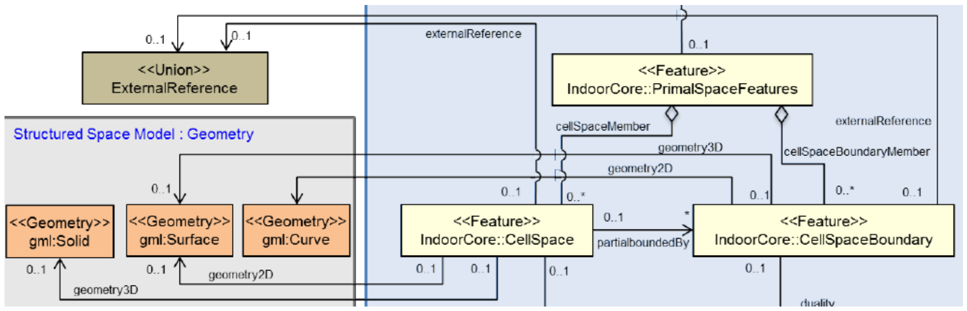


Figure 24 - Thick Wall Model (OGC, 2014)

Considering the limitations of OSM feature representation (which will also be examined further in the current subsection), the Thin Wall Model is better suited for OSM conflation and our conceptual model, due to the greatly increased complexity of representation with use of the Thick Wall model.

In such datasets, only use of Navigable Space features (from the Indoor Navigation module, to be examined later in this section) or State / Transition features (Network Graph without Space / Boundary features) is recommended for migration to a navigable OSM dataset.

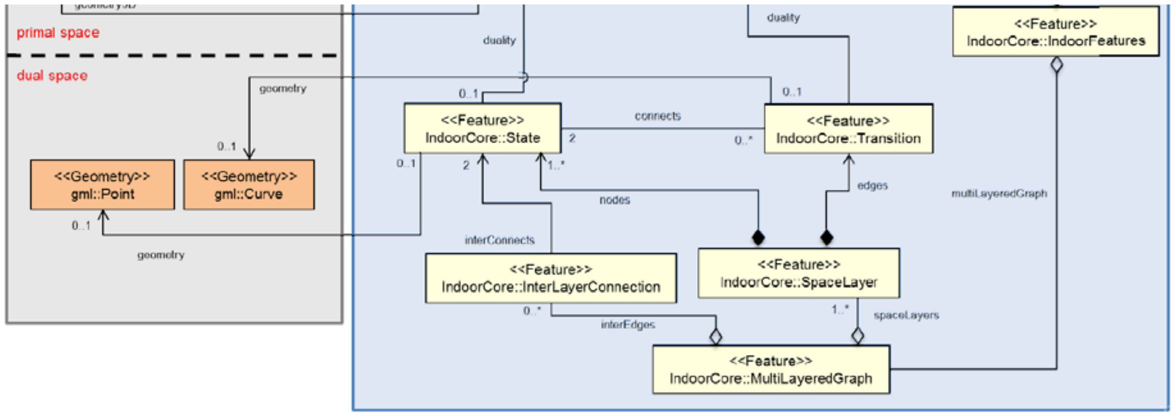
With the above in mind, we can continue with quickly re-examining the cell space features and the relevant geometry elements from the IndoorGML UML Diagram:



IndoorGML cell spaces and their boundaries (2D or 3D) can only be defined as a 2D way element in OSM (with the possible addition of a "height" tag for the 3D elements). Essentially, this can be considered close to only maintaining a single OSM feature equivalent of the two Primal Space feature classes, consisting of an OSM Way which simultaneously provides the boundary equivalent of CellSpaceBoundary and surface area of the contained CellSpace equivalent. Inclusion of geometry is obligatory for the OSM Way feature, but limited to 2D. Calculation of height information from the IndoorGML Solid 3D geometry needs to be done by extracting the feature boundaries to provide an appropriate "min_height" ("floor" level height / lower boundary Z value) or "height" (maximum height, e.g. room ceiling / upper boundary Z value) tag to the OSM Way.

It should be noted that this simplification of the stored geometrical representation also has an impact on the external reference conflation concept, as it has to now refer to the OSM object containing both the space and boundary elements, compared to the level of fine-grained referencing that is normally possible with IndoorGML. This point is further examined in subsection 5.7.

Regarding the Topological representation of data, OSM is well equipped to cover IndoorGML's representation concept. Again, we can first review the Dual Space representation from the IndoorGML Core Module UML Diagram
:



Since OSM features essentially consist of point and linear elements, generation of a Node/State Way/Transition network that replicates the dual space concept, is directly feasible. State IndoorGML features can be covered by OSM Node features, while Transitions are covered by Ways. It should be noted that OSM Ways are actually more complex than the typical IndoorGML Transition feature, as they can consist of multiple edges connecting several nodes and still be defined as a single Way element. As with primal space elements, inclusion of the dual space features' geometry is obligatory in OSM and limited to 2D coordinates. However, Z values of points can be added via the "min_height" key, and Transitions should contain both a "min_height" and "height" tag in order to indicate non-horizontal transitions (e.g. steps).

5.3 Conflation – Semantic Representation

Semantic representation in IndoorGML consists of multiple elements. For a mapping of the IndoorGML feature classes to OSM Elements, a separate subsection is available with a complete overview of the mapping. The current subsection covers the mapping of attributes, its handling in IndoorGML and how they can be migrated to OSM.

As per the initial conceptual model proposed by Nagel (**Nagel, C., 2014**), semantic attributes and their values are stored for each indoor feature, via a GenericAttributeType data type. Further semantics are added via the use of the MLSEM, in the sense that layers are used for semantically similar elements. States can belong to multiple or a single layer, depending on their characteristics (e.g. a corridor that belongs to both a layer covering a walking navigable space and a layer representing wheelchair accessible space, or a topographic space layer and a WLAN coverage layer). Attributes can also be inherited from the layer the element belongs to.

The complete diagram of the semantic concept is available under Appendix I, Figure IV, however, small segments are available below:

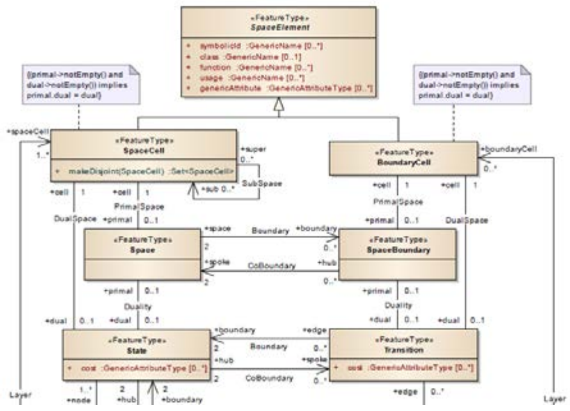

«FeatureType»
SpaceElement
+ symbolicId :GenericName [0..*]
+ class :GenericName [0..1]
+ function :GenericName [0..*]
+ usage :GenericName [0..*]
+ genericAttribute :GenericAttributeType [0..*]
{(primal->notEmpty() and dual->notEmpty()) implies primal.dual = dual}
{(primal->notEmpty() and dual->notEmpty()) implies primal.dual = dual}
«FeatureType»
SpaceCell
+ makeDisjoint(SpaceCell) :Set<SpaceCell>
+spaceCell
1..*
+super
0..*
SubSpace
+sub 0..*
«FeatureType»
BoundaryCell
+boundaryCell
0..*
+cell 1
+cell 1
PrimalSpace
DualSpace
+primal 0..1
«FeatureType»
Space
«FeatureType»
SpaceBoundary
+space
Boundary
+boundary
+spoke
CoBoundary
+hub
Duality
+dual 0..1
«FeatureType»
State
+ cost :GenericAttributeType [0..*]
«FeatureType»
Transition
+ cost :GenericAttributeType [0..*]
+edge
+node
+hub
+boundary
Layer
Layer

Figure 25A - Partial diagram of the semantic concepts of the Space Representation package (Nagel, C., 2014)

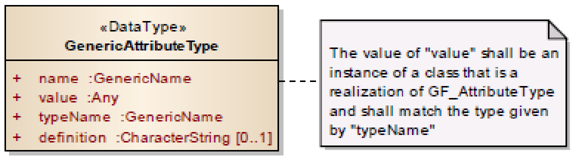
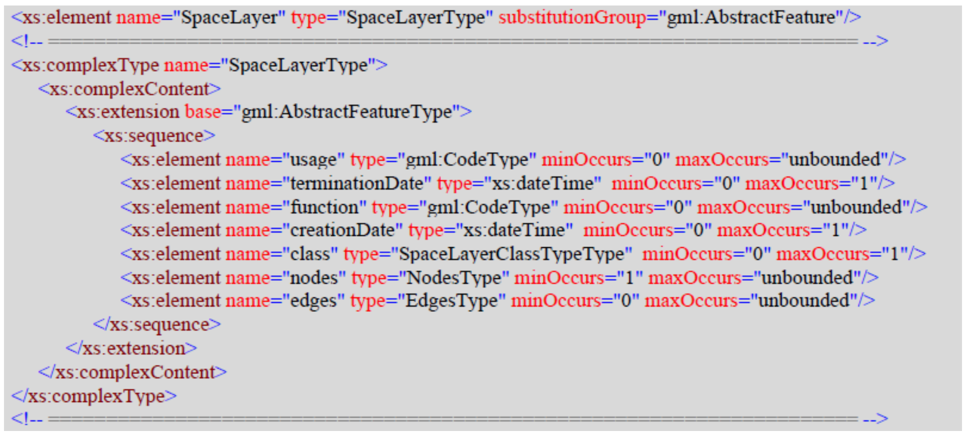

«DataType»
GenericAttributeType
+ name :GenericName
+ value :Any
+ typeName :GenericName
+ definition :CharacterString [0..1]
The value of "value" shall be an instance of a class that is a realization of GF_AttributeType and shall match the type given by "typeName"

Figure 17B - Partial diagram of the semantic concepts of the Space Representation package (Nagel, C., 2014)

In the IndoorGML v1 OGC reference **(OGC, 2014)**, semantic properties are stored in the SpaceLayer Features, which are also inherited by the elements (States, Transitions, Space Cells and Space Cell Boundaries) belonging to each layer. Other than providing additional data, this semantic information is also useful in mapping elements of the Core Module to other thematic modules, such as the Navigation module. It should be noted that these additional attributes have values based on predefined codelists to ensure interoperability. Codelist values are available as part of the IndoorGML v1 OGC reference and are defined based on the OmniClass classification, (Tables 13 and 14, available at http://www.omniclass.org/), which is extensively used in the US industry. The Omniclass & CityGML based codelists used in IndoorGML v1 are also available for review in this document, in Appendix III.

These values are defined as per the following excerpt from the SpaceLayer feature definition in the IndoorGML v1 XSD schema (OGC, 2014):



```
<xs:element name="SpaceLayer" type="SpaceLayerType" substitutionGroup="gml:AbstractFeature"/>
<!-- ================================================================ -->
<xs:complexType name="SpaceLayerType">
    <xs:complexContent>
        <xs:extension base="gml:AbstractFeatureType">
            <xs:sequence>
                <xs:element name="usage" type="gml:CodeType" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="terminationDate" type="xs:dateTime"  minOccurs="0" maxOccurs="1"/>
                <xs:element name="function" type="gml:CodeType" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="creationDate" type="xs:dateTime"  minOccurs="0" maxOccurs="1"/>
                <xs:element name="class" type="SpaceLayerClassTypeType"  minOccurs="0" maxOccurs="1"/>
                <xs:element name="nodes" type="NodesType" minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="edges" type="EdgesType" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- ================================================================ -->
```

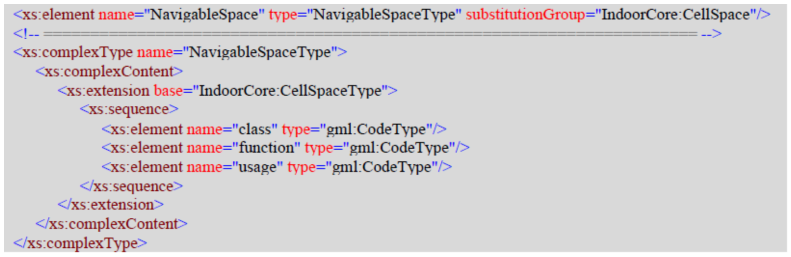
Figure 26 - Semantic attributes stored in SpaceLayer features as per IndoorGML v1 XSD Schema (OGC, 2014)

The above covers the semantic information stored via the IndoorGML Core Module. Thematic extensions such as the IndoorGML Navigation module and Constraints concept are defined to offer additional semantic attributes that are aimed towards specific use cases such as indoor navigation or constraint definition / context awareness support.

The IndoorGML Navigation Module indirectly provides semantic information by defining new feature types that are derived from the Core Module features. Core States and Transitions are mapped to Navigation Route Nodes and Route Segments which comprise a Route feature, while Core Cell Spaces and Cell Space Boundaries are mapped to Navigation Navigable and Non-Navigable Space and Boundary features. These are further broken down into other subcategories, depending on their function (General, Transition, Connection or Anchor Spaces).

The NavigableSpace elements are the only ones including direct storage of semantic information based on the class, function and usage attributes which utilize the same Codelists as Space

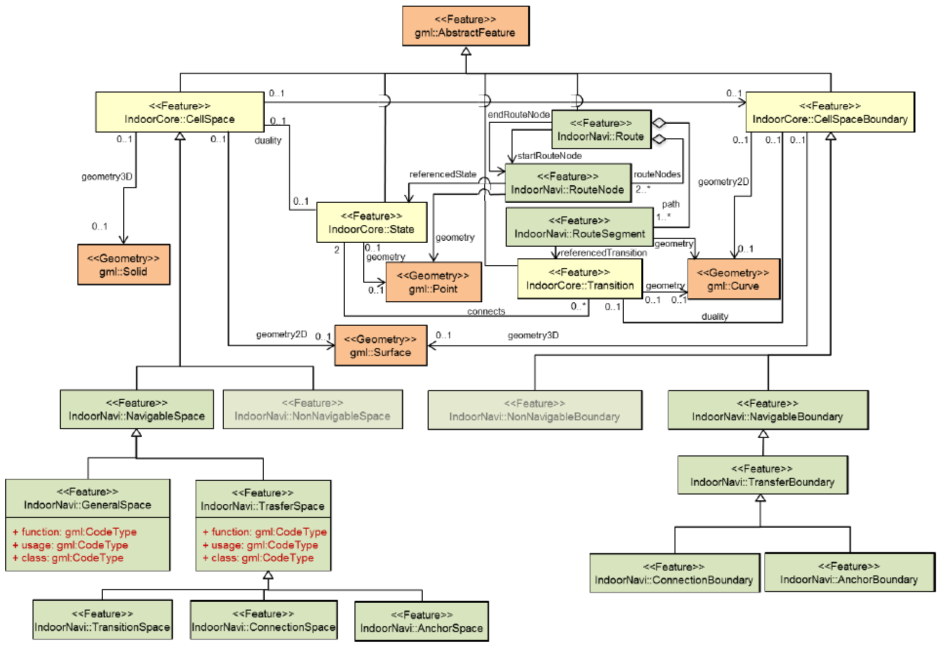
Layers, according to the NavigableSpace feature definition found in the Indoor Navigation Module description from the IndoorGML v1 XSD Schema (OGC, 2014)



```
<xs:element name="NavigableSpace" type="NavigableSpaceType" substitutionGroup="IndoorCore:CellSpace"/>
<!-- ================================================================== -->
<xs:complexType name="NavigableSpaceType">
    <xs:complexContent>
        <xs:extension base="IndoorCore:CellSpaceType">
            <xs:sequence>
                <xs:element name="class" type="gml:CodeType"/>
                <xs:element name="function" type="gml:CodeType"/>
                <xs:element name="usage" type="gml:CodeType"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Figure 27 – Navigation Module - NavigableSpace feature definition - IndoorGML v1 XSD Schema (OGC, 2014)

Semantic information is additionally provided indirectly from the IndoorGML Navigation Module via the child elements of Navigable Space and Navigable Boundary features, as the specific feature type used allows us to infer more information about the Space Cell or Boundary feature it's mapped to. A complete overview via the UML diagram of the Indoor Navigation Module is available below (OGC, 2014):



<<Feature>> gml::AbstractFeature
<<Feature>> IndoorCore::CellSpace
<<Feature>> IndoorCore::CellSpaceBoundary
<<Feature>> IndoorNavi::Route
<<Feature>> IndoorNavi::RouteNode
<<Feature>> IndoorNavi::RouteSegment
<<Feature>> IndoorCore::State
<<Feature>> IndoorCore::Transition
<<Geometry>> gml::Solid
<<Geometry>> gml::Point
<<Geometry>> gml::Curve
<<Geometry>> gml::Surface
endRouteNode
startRouteNode
routeNodes
path
referencedState
referencedTransition
duality
geometry3D
geometry2D
geometry
connects
<<Feature>> IndoorNavi::NavigableSpace
<<Feature>> IndoorNavi::NonNavigableSpace
<<Feature>> IndoorNavi::NonNavigableBoundary
<<Feature>> IndoorNavi::NavigableBoundary
<<Feature>> IndoorNavi::TransferBoundary
<<Feature>> IndoorNavi::GeneralSpace
+ function: gml:CodeType
+ usage: gml:CodeType
+ class: gml:CodeType
<<Feature>> IndoorNavi::TrasferSpace
+ function: gml:CodeType
+ usage: gml:CodeType
+ class: gml:CodeType
<<Feature>> IndoorNavi::ConnectionBoundary
<<Feature>> IndoorNavi::AnchorBoundary
<<Feature>> IndoorNavi::TransitionSpace
<<Feature>> IndoorNavi::ConnectionSpace
<<Feature>> IndoorNavi::AnchorSpace

Figure 28 - IndoorGML Navigation Module UML Diagram

Another important element of semantic information is constraints. These can range from time and access constraints which are function / rule driven to mode of transport and directionality constraints which have more to do with the actual physical space. The IndoorGML v1 OGC Reference document defines a concept for the definition of constraints, which follows a similar logic to the IndoorGML Navigation thematic module. More specifically, the Indoor Constraints concept defines Constraint features that can be mapped to Space Cell or Space Cell Boundary elements.

As it is still a concept, IndoorGML v1 does not include a specific definition of the constraints model in the current XSD schema. However, a quick overview of the constraint coverage included in the concept can be seen in the following conceptual model diagram **(OGC, 2014)**:

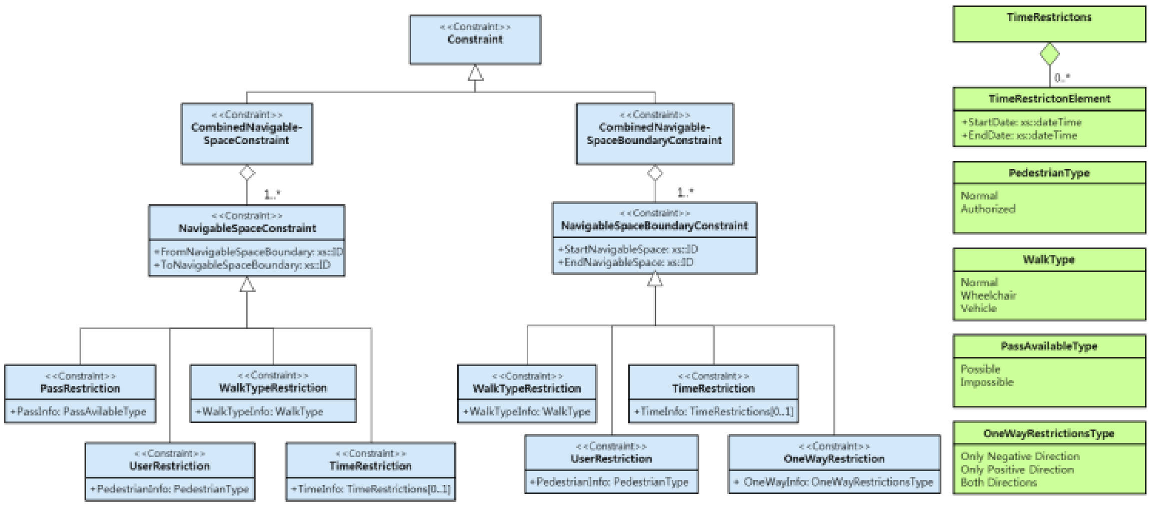


Figure 29 - Conceptual model of indoor navigation constraints (OGC, 2014)

With the above information, we have covered the semantic representation in IndoorGML. However, there are significant differences in the treatment of semantic information in IndoorGML and OpenStreetMap. As we have mentioned in section 4.3.3, OpenStreetMap employs a vast but arbitrary set of key & value pairs via its free tagging system, which can be used to define attributes of any node, way or relation element in an OpenStreetMap data set.

The free tagging system in OSM is not only used to provide attributes to features, but to further define complex features out of the node / way elements. As such, closed ways can be further defined as outlines or areas, including 2D representations of features such as buildings, building parts, rooms, etc. Nodes can be defined as point feature indicators for simplified representation of complex elements or direct representation of features such as building entrances, doors, etc. Relations often derive functionality from the addition of a "type" key which more explicitly defines the content and purpose of the relation.

It should be noted again, that with such a free-tagging schema, the extension possibilities of OSM are practically limitless; however this comes at the cost of consistency and interoperability. With that in mind, several proposals currently exist regarding the mapping of indoor spaces in OSM, as discussed in Section 4.3.5, however, no single model / proposal is currently considered a standard or at least a best practice. With that in mind, we will attempt to provide a proposal for the tagging of features that covers as much of the standard semantic information available in IndoorGML Core, Navigation and Constraint Modules via the use of well-established OSM tags, while still allowing for use of the OSM free tagging system in order to accommodate further generic attributes defined as part of an IndoorGML dataset via the use of new tag key namespaces.

Considering the above, we can already provide some general rules regarding tagging based on the MLSEM and Thematic modules. One common element for the MLSEM and Navigation modules are the class, function and usage Omniclass based attributes. These can be stored to OSM elements directly or as part of the layer element containing them by using the keys “IndoorGML:Core:Class”, “IndoorGML:Core:Function” and “IndoorGML:Core:Usage”. The values used should be based on the Omniclass values listed in the IndoorGML v1 OGC reference document (OGC, 2014) and in Appendix III. It is important to note that the meaning of the codelist values depends on the type of *Space element they are describing. With that in mind, the “IndoorGML:Navi” tag (further described in subsection 5.8.2) must be reviewed alongside the Omniclass based attributes.

The elements belonging to the IndoorGML Navigation module feature classes are normally mapped to features from the topographic space layers from the IndoorGML Core module. As part of our conflation process, Navigation features are treated simply as a source for additional semantic information for the topographic space features they are connected to. As a result, we use the tag key “IndoorGML:Navi:Navigable”, with “yes/no” values depending on whether the feature is mapped to **<Navigable*>** or **<NonNavigable*>** IndoorGML Navigation features. When mapped to the subclasses of the above IndoorGML Navigation classes, features are tagged with the “IndoorGML:Navi” key, with the values “Anchor/Transfer/Connection/Transition”, depending on the “source” IndoorGML Navigation class elements belong to.

The Constraint Concept is again intended to be used as a thematic extension for adding semantics to Core Module features. However, as there is no concrete implementation on the classes to be included as of v1 of the IndoorGML OGC reference document, we will only provide some recommendations for tagging based on the current concept classes. Again, depending on the source Constraints Module class, we would tag the corresponding Core Module derived feature, as with the IndoorGML Navigation Module. If using standard OSM tags, the “access” key can cover almost all of the Constraints module classes via the definition of a complex conditional value, with the exception of the **<OneWayRestriction>** class which can be covered via the “oneway” key. However, it would be recommended to also follow the tagging logic used for the Navigation Module, with use of an “IndoorGML:Constraints:*” key namespace, which can be the basis for specific keys covering the scope of the Constraints Module. Applicable values should then be defined depending on the type of constraint represented.

The above covers the general rules and recommendations for semantic information and the use of tagging to represent it. The exact tagging schema is covered more in section 5.8, as it is further defined on a feature by feature basis, in order to avoid adding unnecessary tags and increasing the complexity of elements.

5.4 Conflation - Multi-Layered representation

As we have mentioned in section 4.2, IndoorGML is based around the concept of the MLSEM, allowing for the simultaneous representation of multiple information layers with discreet or shared elements (space cells / states) in a single dataset.

The main benefit of this approach is that it allows for both the definition of semantic layers as well as the distinction between them and their elements. An additional benefit is the capacity for establishing inter-layer connections linking elements that are shared across layers or connected (e.g. a topographic space cell and a WLAN coverage cell or spaces and access restrictions).

One important note is that while semantic information can be provided via use of multiple layers, for example separate layers per mode of transport, as examined in combination with subspacing in **(Khan, A. A., & Kolbe, T. H., 2012)**, the Thematic extension modules of IndoorGML may be covering the same semantic data in a more structured manner that better ensures interoperability, despite being more constrained / limited compared to the free definition of additional layers.

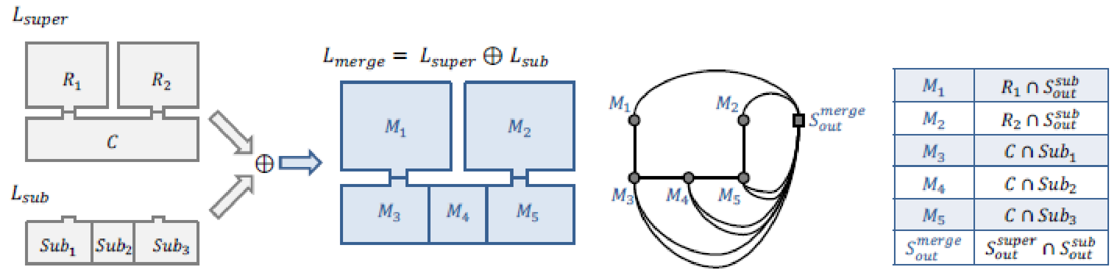
This comes to contrast with OSM, where a singular data layer exists, covering the complete representation of all features. As mentioned previously within this section, considering the mapping oriented nature of the OSM project explains the lack of support for features without a matching topographic space representation.

When attempting to migrate data from an MLSEM based IndoorGML dataset to OSM, only layers and elements with a topographic space representation (or elements linked to a matching topographic space via an inter-layer relation) can be defined in OSM.

Two possible approaches have been identified in order to represent a multiple layer based data model in the single data layer representation of OSM.

The optimal approach is merging elements of all semantically different MLSEM Layers to a single layer, including all the elements that are present in all layers (and have a topographic space representation), along with storing any attributes and semantic information that is available from the source layers on the respective merged element.

This concept was also investigated in **(Nagel, C., 2014)**, from where we have the following representation and example of the merging process:



M_1	$R_1 \cap S_{out}^{sub}$
M_2	$R_2 \cap S_{out}^{sub}$
M_3	$C \cap Sub_1$
M_4	$C \cap Sub_2$
M_5	$C \cap Sub_3$
S_{out}^{merge}	$S_{out}^{super} \cap S_{out}^{sub}$

Figure 30 - Recommended Layer merging process for MLSEM data, including table with input space cells and applied operations for merged data (Nagel, C., 2014)

According to the above approach, a single OSM data "layer" would be created, containing all elements (nodes, ways) from all IndoorGML MLSEM layers, with tagging being composed from the combination of layer attributes each element is a member of, along with any attributes derived from the mapping of IndoorGML Core Features to the thematic extension modules (IndoorGML Navigation Module and Constraints concept). Inter-layer connections definition in OSM is unnecessary in this approach, as the MLSEM is essentially "flattened" to a single layer and elements' merged attributes are derived precisely based on those inter-layer connections.

The second approach would be based on the creation of multiple OSM elements based on all the available layers and their elements. As a result, we would have multiple overlapping way and node OSM elements (where the elements share topographic space across layers in MLSEM), which would utilize different, mutually exclusive tagging, in order to accurately represent all layers separately. Inter-layer connections can be then defined via the use of OSM relations which would establish the base topographic space layer and link it with the matching elements. The concept can be graphically demonstrated as follows:

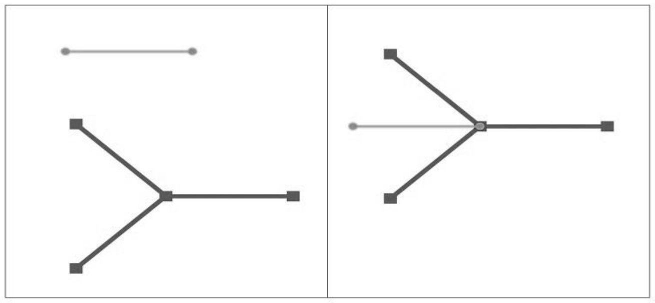


Figure 31 - Concept for superimposed OSM elements derived from multiple MLSEM layers

While one could argue that the latter approach better maintains the integrity of the MLSEM data in OSM and could be used to reverse engineer the source IndoorGML dataset, the drawbacks are significant. There is a large amount of overhead in the repeated representation of overlapping data, while the derived OSM dataset will be rather hard to work with in the most OSM editors, which are not optimized for overlapping data without heavy use of filtering based on properties. Additionally, the complete semantic attributes of a feature (the derived OSM equivalent of the initial IndoorGML State or Space Cell) are only retrievable via the relation, which would cause issues with most OSM-based applications (geocoders, routing engines, etc.).

However, there is one use case where such an approach makes sense, specifically when an IndoorGML dataset only includes a small number of layers, which are used to define separate layers per mode of transport (instead of using the Navigation thematic extension module).

As this was the case with the sample IndoorGML dataset that was used for the proof of concept implementation, this approach was followed, generating a network graph that was used for walking instructions based on the complete topographic space MLSEM layer, while a secondary layer which represented wheelchair accessibility was recreated in OSM as a second node/way network which overlapped the main network. The two were tagged with the "wheelchair" OSM key, with "no" and "yes" values for the topographic and wheelchair layers. As there were no other semantic based layers, there was no significant overhead in data and this allowed for the quick creation of a navigable dataset which could consider wheelchair accessibility constraints apart from typical walking instructions.

Considering the above, while there are some possible use cases where representing each MLSEM layer as separate overlapping networks in OSM looks acceptable, the merging/flattening of MLSEM layers in a single layer containing the complete semantic information seems like an overall preferable approach, providing a cleaner and easier to process dataset, both by tools as well as users.

5.5 Conflation - Subspacing

As we have previously mentioned, the notion of cellular space, as it is defined in IndoorGML, cannot be properly migrated to OSM. Another problematic concept in that aspect is IndoorGML's Subspacing. Breaking down features into smaller subsets is not a function that can be directly done in OSM.

As a result, the recommended workaround is the manual definition of additional equivalent features that are based on the "parent" primal space feature and within its boundaries, which can then be connected to the parent feature via an OSM relation. Indicating that these features were created as a result of subspacing is recommended, via the use of the "IndoorGML:Core:Subspace:Child=yes" key/value pair and the "IndoorGML:Core:Subspace:Parent" tag, with the value here referencing the OSM ID of the parent feature. This allows for easier selection and grouping of elements and creation of the relation indicating the subspace.

The relation should consist of the subspace features and its boundary should match the outline of the “parent” feature. It is not obligatory to define a type for OSM relations. However, for consistency reasons, the new “subspace” relation type could be entered. Existing relation types such as “multipolygon” should not be used, as their official semantics does not match the subspacing concept semantics.

When creating subspacing relations, it is important to indicate the parent/source feature. This feature should not be entered as a member of the relation but can be defined via the use of the “IndoorGML:Core:Subspace:Parent” tag, with the value again referencing the OSM ID of the parent feature.

5.6 Conflation - Anchor nodes

Anchor Nodes are a very important concept of IndoorGML, as they serve the role of connecting the indoor data defined in the dataset with their external environment, a function that is directly linked with the thesis goal of conflating the indoor and outdoor datasets.

The main functions of the Anchor node are defining the CRS used within the IndoorGML dataset, the transformation parameters necessary for reprojecting it to other CRSs and providing a connection point that is “shared” between the indoor and outdoor datasets.

In the context of conflation, the first function is handled as follows: OSM Data is based on the WGS84 / EPSG 4326 CRS. IndoorGML data may utilize a different absolute (e.g. EPSG:31468) or a relative CRS. The transformation parameters that are stored in the Anchor Node are examined at the beginning of the IndoorGML derived OSM dataset creation process and the needed transformation is applied to all the IndoorGML dataset features. As the features are transformed, there is no need for further storage of the transformation matrix or source CRS via tags or otherwise.

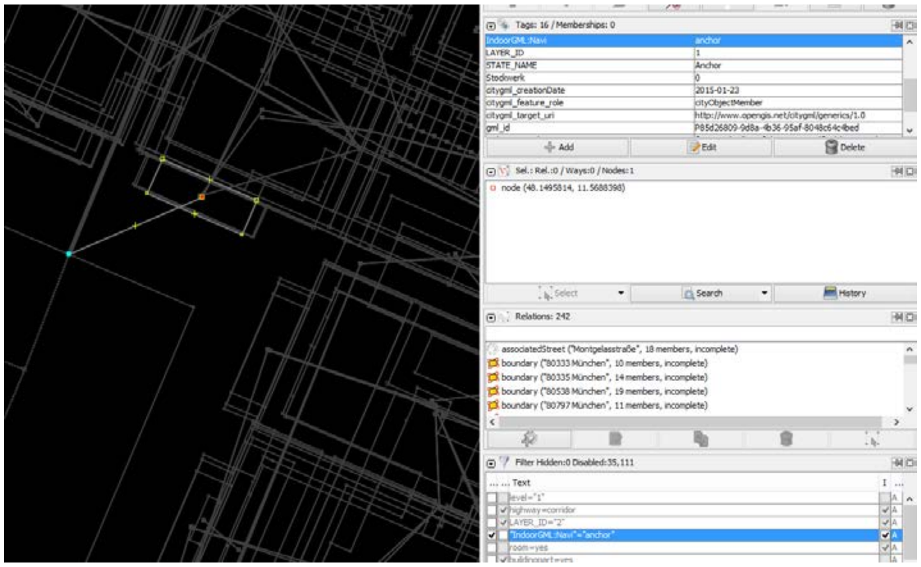
The second and more important function is addressed via the use of tagging. As Anchor Spaces / Nodes are a feature of the IndoorGML Navigation thematic module, they are treated as a source for additional semantic attributes for the core features defined from the topographic layer to which they are mapped. As a result, Anchor Spaces / Nodes use typical “building:part=yes” or “room=yes” tagging, with an additional custom “IndoorGML:Navi=Anchor” tag.

In order to achieve the goal of connectivity, the Anchor features need to be matched to a typical OSM feature that defines the same transition point between outdoor and indoor as the Anchor Space / Node. From a semantic perspective, the “entrance=*” OSM tag is the best match. However, it is important to note that the “entrance” tag has some limitations, which also prevent us from directly applying it to the Anchor elements. The “entrance” tag can only be used to describe a node feature and must be part of the “building” feature outline way.

This differentiation in representation leads to the need for a process to identify and match the Anchor Space / Node features to their “entrance” equivalents. This can be achieved via use of the Anchor features as base points and identification of the nearest “entrance” features within a reasonable distance threshold. Additional checks can be implemented to ensure that the matched features are also sharing the same direction and avoid wrong matches. One way to implement this is to confirm the orientation (determined for example via azimuth angle) of the broad side of Anchor Space (covering the vast majority of entrance types) matches that of the wall / “building” outline segment the matched OSM “entrance” feature belongs to.

Once matched, the features can be connected via the creation of a way element with “highway=corridor” tagging and additional semantic tags copied from the origin Anchor Node feature, so as to ensure that attribute/tag based semantics such as wheelchair accessibility are not lost in the transition.

As this is an automated process, it is recommended that the matches between elements and generated transitions are also reviewed manually, to ensure that no mismatches are present. One additional point is that it is often the case that no or wrong entrance features are defined in the main OSM database (and the resulting “outdoor” OSM dataset). With that in mind, it is recommended to always review the “outdoor” OSM dataset of the area surrounding the IndoorGML dataset defined buildings and correct or add the necessary entrance tags where applicable.



Tags: 16 / Memberships: 0

IndoorGML:Navi	anchor
LAYER_ID	1
STATE_NAME	Anchor
Stockwerk	0
citygml_creationDate	2015-01-23
citygml_feature_role	cityObjectMember
citygml_target_uri	http://www.opengis.net/citygml/generics/1.0
gml_id	P85d26809-9d8a-4b36-95af-8048c64c4bed

Sel.: Rel.:0 / Ways:0 / Nodes:1

node (48.1495814, 11.5688398)

Relations: 242

associatedStreet ("Montgelasstraße", 18 members, incomplete)
boundary ("80333 München", 10 members, incomplete)
boundary ("80335 München", 14 members, incomplete)
boundary ("80538 München", 19 members, incomplete)
boundary ("80797 München", 11 members, incomplete)

Filter Hidden:0 Disabled:35,111

		Text	I	...
		level="1"		A
	✓	highway=corridor	✓	A
	✓	LAYER_ID="2"	✓	A
✓		"IndoorGML:Navi"="anchor"	✓	A
		room=yes	✓	A
	✓	buildingpart=yes		A

Figure 32 - Anchor Node / Space equivalent OSM feature, Nearest Neighbor “entrance” Outdoor feature and generated Way / Transition

5.7 Conflation - External referencing

External Referencing in IndoorGML is used to directly link features from the IndoorGML dataset to other datasets such as a CityGML or IFC file.
To avoid confusion with the typical OSM use of the “ref” tag to indicate references, and as the IndoorGML reference concept serves a different purpose, a new “IndoorGML:Core:Ext_Ref:*” tag namespace is defined.

Since the main information stored in IndoorGML as per the OGC v1 schema is the information system, name of the object and URI, all these values can be defined as part of the same tag namespace, via “IndoorGML:Core:Ext_Ref:Inf_Sys”, “IndoorGML:Core:Ext_Ref:Obj_name” and “IndoorGML:Core:Ext_Ref:Obj_URI” tags. The tags are directly assigned to the OSM equivalents of the IndoorGML features that contained the external reference.

5.8 Conflation – Feature by Feature Mapping

So far, the Conflation section has been covering the higher-level mapping process for the IndoorGML core concepts. In the current subsection, we will examine the mapping from IndoorGML to OSM on a feature by feature level, covering all classes of the IndoorGML Core and Navigation Modules.

5.8.1 IndoorGML Core Feature Classes

Before delving into each of the classes of the IndoorGML Core Module, the table below provides an overview of the feature by feature mapping, including some notes regarding the necessary process steps:

IndoorGML Class	OSM Element	Process / Notes
<State>	Node	3D to 2D with min_height / height tag
<Transition>	Way	3D to 2D with min_height / height tag
<CellSpace>	Closed Way	3D to 2D with min_height / height tag - Surface projection for Solids
<CellSpaceBoundary>	Closed Way	3D to 2D with min_height / height tag
<SpaceLayer>	Relation	Contains all State / Transition elements / Tag source for layer members
<InterLayerConnection>	Relation	Tag source for connected elements
<MultilayeredGraph>	Relation	Contains all SpaceLayer elements
<PrimalSpaceFeatures>	Relation	Contains all CellSpace & CellspaceBoundary elements
<IndoorFeatures>	Relation	Contains all PrimalSpaceFeature & MultilayeredGraph elements

Table 2 - IndoorGML Core Feature Class to OSM Element mapping

Starting with our feature by feature analysis, we examine the Dual Space representation elements:

The **<State>** IndoorGML class can be represented in OSM via the use of a node element. Geometry is defined via extraction of the features' XYZ geometry and use of X,Y values for positioning the element, while the Z value is saved via use of the "min_height" tag. As part of the IndoorGML dataset, the "indoor=yes" tag is also added, along with a State specific tag of "room=yes". Further attributes are assigned to the element as per the recommendations outlined in subsections 5.3 and 5.4.

The **<Transition>** class can be reflected in OSM via use of OSM way features. The geometry is provided as usual with the retrieval of X,Y coordinates directly from the IndoorGML geometry, while Z values are provided via the "min_height" and "height" tags. The use of both tags is recommended in order to better cover any non-horizontal transitions, (e.g. stairs). The "indoor" tag is again applied to the OSM features, and they are further defined via use of the highway tag.

Depending on the type of source feature, it is recommended to use the values "corridor" for typical transitions such as corridors or ramps, "steps" for any stair elements and "elevator" to define vertical transitions, typically used for elevators. Further attributes are again defined as per the recommendations outlined in subsections 5.3 and 5.4.

The **<CellSpace>** and **<CellSpaceBoundary>** classes can both be mapped to closed way elements in OSM, however it is important to note that as a mainly 2D representation, the current OSM conflations concept does not require both feature classes. A surface projection of the solid space is the basis for the OSM feature definition. The surface projection can either be derived from the solid or the boundary geometry, but as only one projection is needed to define the feature, "duplicate" projections are not necessary.

In the proof of concept implementation, only solid geometries (no boundaries) were defined, so extraction was straightforward, however datasets with both classes need to be handled carefully to avoid the generation of duplicate features.

Again, for the geometry, the X,Y coordinates are stored in the feature geometry, while lower and upper boundary Z values are stored via the use of "min_height" and "height" tags. All features derived from the two classes are additionally tagged with "indoor=yes" and "building:part=yes" tags. Further semantic information and attributes are defined as per subsections 5.3 and 5.4.

The **<SpaceLayer>** class can be defined in OSM as a relation of type "IndoorGML:Core:MLSEM:Layer" that contains all member elements of a layer. The class, function and usage Omniclass-based attributes can also be stored here using the keys "IndoorGML:Core:Class", "IndoorGML:Core:Function" and "IndoorGML:Core:Usage".

The **<InterLayerConnection>** class is also defined as a relation in OSM, with the type "IndoorGML:Core:MLSEM:InterLayerConnection" where a pair of member elements can be stored. This allows for the identification of connections between layers and the transference of attributes from the secondary layers to the connected features defined via the main topography layer.

Additional tags are used to further define the relation as per the IndoorGML class. These are "IndoorGML:Core:MLSEM:InterLayerConnection:ConnectedLayers" which defines the parent layer of the relation members, via semicolon separated values, and "IndoorGML:Core:MLSEM:InterLayerConnection:typeOfTopoExpression" which can have the values "contains/overlaps/equals".

The **<MultilayeredGraph>** class is represented by a super-relation in OSM, of the type "IndoorGML:Core:MultilayeredGraph" which contains all the relations created from the **<SpaceLayer>** and **<InterLayerConnection>** IndoorGML classes.
The **<PrimalSpaceFeatures>** class can be again defined via the use of a relationship of the type "IndoorGML:Core:PrimalSpaceFeatures", containing all features generated from the **<CellSpace>** and/or **<CellSpaceBoundary>** classes as members.

Finally, the **<IndoorFeatures>** class is also defined via an OSM super-relation with the "IndoorGML:Core:IndoorFeatures" type, containing the relations derived from the **<PrimalSpaceFeatures>** and **<MultilayeredGraph>** classes.

The above overview covers the mapping of all Core Module feature types to OSM elements, including recommendations for the correct tagging of features. As a reminder, for the complete semantic data, it is important to also review the tagging recommendations from the Semantic Representation conflation overview in section 5.3.

5.8.2 IndoorGML Navigation Feature Classes

As with the IndoorGML Core Module, the table below provides an overview of the feature by feature mapping of the IndoorGML Navigation Module classes, again including notes about the process steps:

IndoorGML Class	OSM Element	Process / Notes
<NavigableSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<NonNavigableSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<GeneralSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<TransferSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<ConnectionSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<AnchorSpace>	Closed Way/None	CRS Source for necessary reprojection – Connection to Outdoor OSM dataset – 2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<TransitionSpace>	Closed Way/None	2D Closed way defined from equivalent CellSpace elements – used as Tag source only
<NavigableBoundary>	Closed Way/None	2D Closed way defined from equivalent CellSpaceBoundary elements – used as Tag source only
<TransferBoundary>	Closed Way/None	2D Closed way defined from equivalent CellSpaceBoundary elements – used as Tag source only
<ConnectionBoundary>	Closed Way/None	2D Closed way defined from equivalent CellSpaceBoundary elements – used as Tag source only
<AnchorBoundary>	Closed Way/None	CRS Source for necessary reprojection – Connection to Outdoor OSM dataset – 2D Closed way defined from equivalent CellSpaceBoundary elements – used as Tag source only
<RouteNode>	Node	Not necessary for OSM Routing, can be used to create predefined Route Relations
<RouteSegment>	Way	Not necessary for OSM Routing, can be used to create predefined Route Relations
<Route>	Relation	Not necessary for OSM Routing, can be used to represent predefined routes

Table 3 - IndoorGML Navigation Module Feature Class to OSM Element mapping

As we can see from the table, the IndoorGML Navigation module is mostly used (within the conflation concept) in order to derive further semantic attributes for the features defined via the Core Module classes.

More specifically, with the exception of the final three classes, there is no need to define additional OSM features, as information such as geometry is directly represented by the Core module features that the Navigation module class features are mapped to. Based on the above, we will be providing below a mapping of the appropriate tags to be added to the Core class features, depending on the Indoor Navigation module class that is mapped to them.

The **<NavigableSpace>** and **<NonNavigableSpace>** classes, as well as **<NavigableBoundary>** And **<NonNavigableBoundary>**, can be migrated to the OSM dataset via the use of a new tag, “IndoorGML:Navi:Navigable”, with “yes/no” values. As mentioned, the tags should be applied to the corresponding features defined from the Core module.

The classes **<GeneralSpace>**, **<TransferSpace>**, **<TransferBoundary>**, **<ConnectionSpace>**, **<ConnectionBoundary>**, **<AnchorSpace>**, **<AnchorBoundary>,** and **<TransitionSpace>** are handled in the same logic, via use of the “IndoorGML:Navi” tag, with values “Anchor/Transfer/Connection/Transition”. This tighter grouping is selected as the exact original class can be identified depending on the tag value and the OSM feature type it is attributed to, with node features (States) indicating a ***Space** origin class, while way features (Transitions) indicate a ***Boundary** origin.

The final three classes of the IndoorGML Navigation Module, **<RouteNode>**, **<RouteSegment>** and **<Route>** are not necessary for routing applications based on OSM, however, they can be used to create predefined routes through an IndoorGML dataset. With that in mind, **<RouteNode>** features can be created in the form of OSM Nodes, while **<RouteSegment>** features can be defined as ways. If new features are defined based on these classes, it is recommended to provide any additional tags necessary, such as “indoor=yes” “highway=corridor/steps/elevator” or “room=yes” and any additional semantics/ attributes that are necessary for correct navigation.

The **<Route>** class can then be defined in the OSM dataset as a relation which can utilize the existing OSM “route” type, and have as members elements from either the **<RouteNode>** and **<RouteSegment>** Navigation Module classes, or **<State>** and **<Transition>** elements from the Core classes. The relation can be defined in order to present typical predefined routes through a building, for example a tour guide route through a museum. The **<RouteNode>** and **<RouteSegment>** remain available to define and use as separate elements from the existing State nodes and Transition ways in our conflation concept, in order to allow for route node and segment definition that includes locations outside the IndoorGML derived dataset (e.g. the “outdoor” OSM data.

The above overview covers the mapping of all Navigation Module feature types to OSM semantic elements / tags. As the Constraints Concept has no concrete class definition, this subsection concludes the feature by feature mapping of IndoorGML features.

5.9 Conflation – OSM/IndoorGML Conflation Conceptual Model – Workflow

Based on the conflation analysis of the core IndoorGML concepts, along with the feature to feature mapping, we can define a workflow for generating an OSM dataset which manages to adequately migrate IndoorGML data to the OSM data model and provide a navigable OSM dataset that can be utilized by existing or new OSM-based applications and tools.

The workflow is presented below in graphical form:

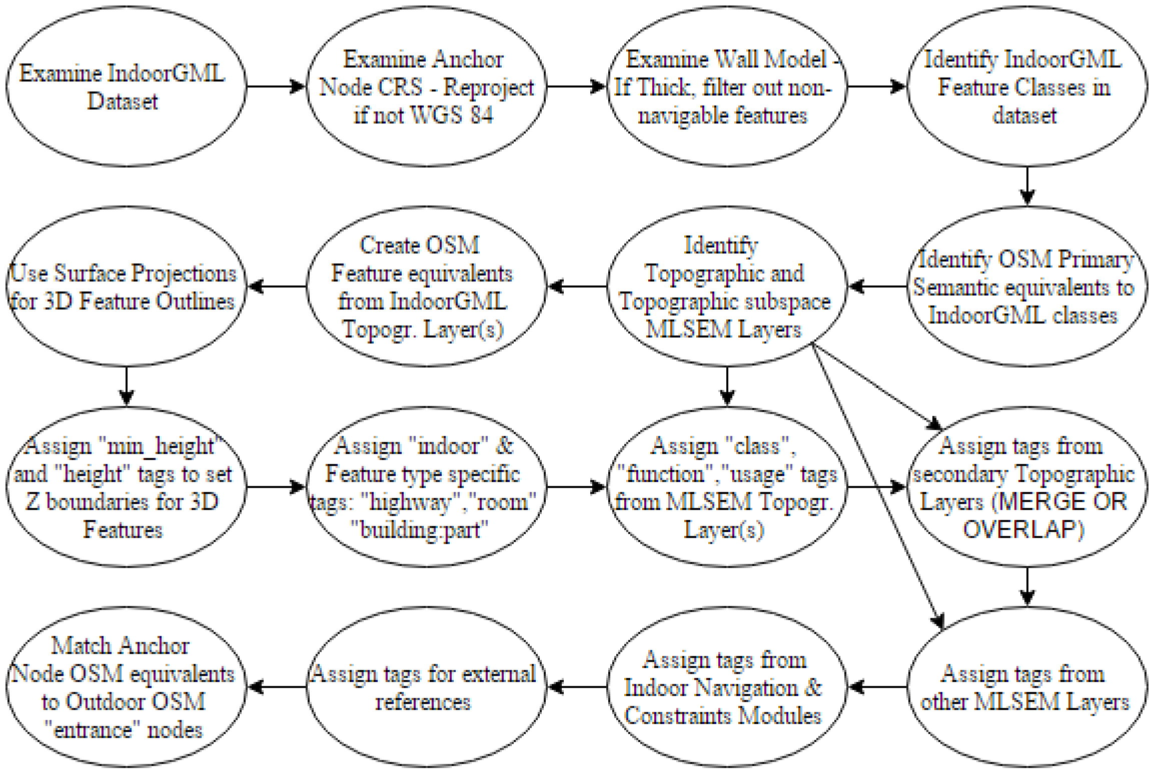

Examine IndoorGML Dataset
Examine Anchor Node CRS - Reproject if not WGS 84
Examine Wall Model - If Thick, filter out non-navigable features
Identify IndoorGML Feature Classes in dataset
Identify OSM Primary Semantic equivalents to IndoorGML classes
Identify Topographic and Topographic subspace MLSEM Layers
Create OSM Feature equivalents from IndoorGML Topogr. Layer(s)
Use Surface Projections for 3D Feature Outlines
Assign "min_height" and "height" tags to set Z boundaries for 3D Features
Assign "indoor" & Feature type specific tags: "highway","room" "building:part"
Assign "class", "function","usage" tags from MLSEM Topogr. Layer(s)
Assign tags from secondary Topographic Layers (MERGE OR OVERLAP)
Assign tags from other MLSEM Layers
Assign tags from Indoor Navigation & Constraints Modules
Assign tags for external references
Match Anchor Node OSM equivalents to Outdoor OSM "entrance" nodes

Figure 33 - Conceptual Conflation Model Workflow

Based on the above flowchart, a step by step overview of the creation of data based on the conceptual conflation model would be as follows:

- Analysis of the IndoorGML dataset.
- Detection and parsing of the Anchor Node / Space elements, including reprojection of the data (if needed) to the WGS 84 CRS.
- Identification of the wall model in use in the dataset, and retention of NavigableSpace (as per IndoorGML Navigation module) features or Primal Space (State / Transition) based representation in the case of Thick Wall Model datasets.
- Identification of IndoorGML Core and Navigation feature classes in dataset and their OSM semantic equivalents (as per Feature by Feature mapping from section 5.8).

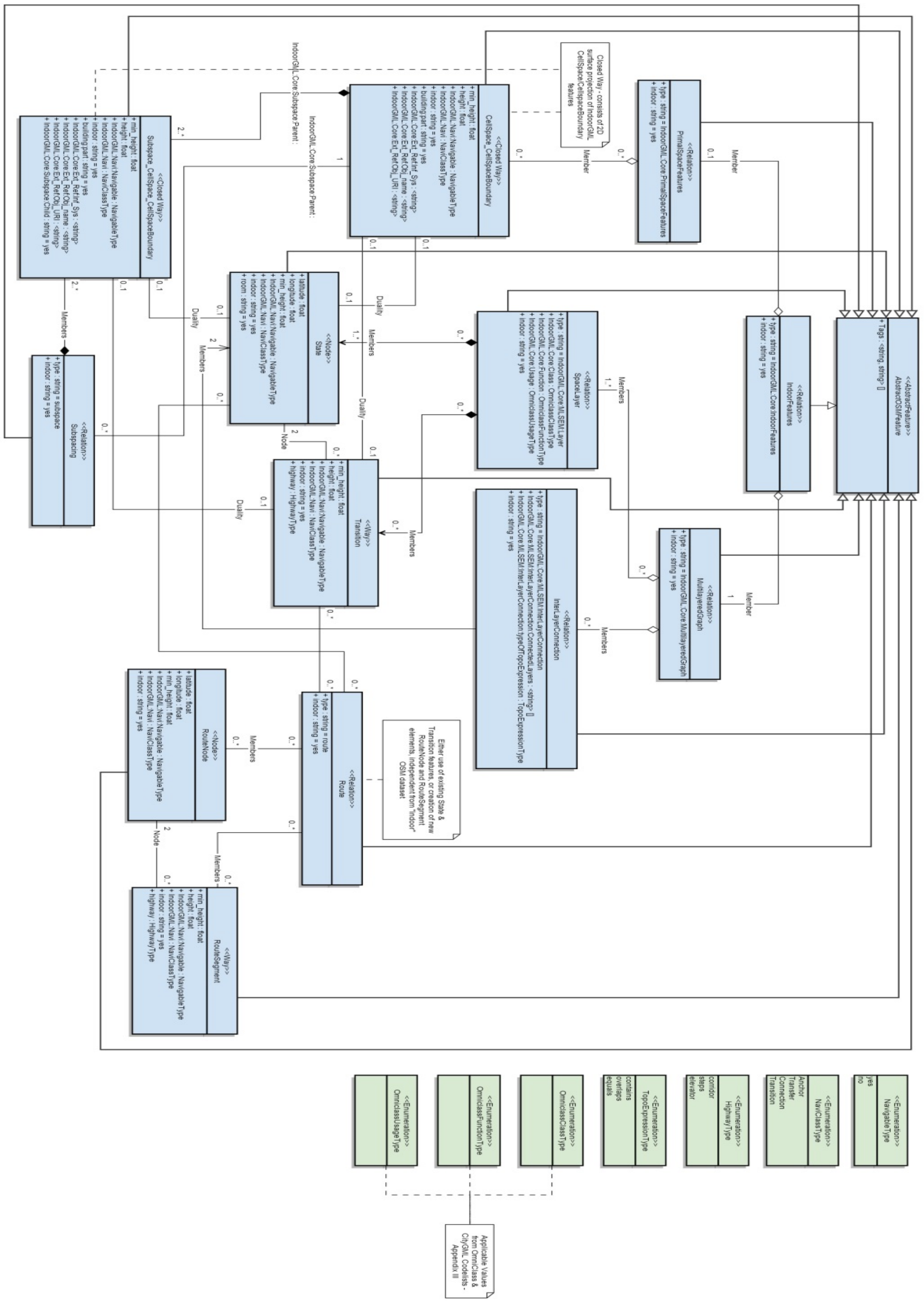
- Identification of base Topographic layers and secondary topographic layers (e.g. subspacing derived layers) and generation of OSM features based on the base topography layer.
- For feature generation, Geometry is represented by 2D (X,Y) surface projection of 3D features (CellSpace, CellSpaceBoundary) and "min_height" / "height" tagging for the lower/upper boundary Z values.
- The OSM features are additionally tagged with an "indoor=yes" tag and room=yes, "building:part=yes" or "highway=corridor/steps/elevator" according to the IndoorGML feature class they were derived from.
- The Omniclass codelist values defined for the features or from their parent MLSEM Layer and added as "IndoorGML:Core:Class", "IndoorGML:Core:Function" and "IndoorGML:Core:Usage" tags to the features.
- Any additional attributes that are directly derived from feature attributes or specified in the parent MLSEM Layers are added as tags. Care should be given regarding correct tag use, where it is best recommended to try and identify a correct OSM equivalent tag (e.g. "wheelchair=yes/no") and failing that, create a tag based on the "IndoorGML:Attribute:*" namespace. Using existing OSM tags ensures some interoperability
- For attributes added to features that are also mapped to classes from the thematic modules (IndoorGML Navigation Module and possibly in future versions Constraints), it is recommended to use a namespace defining the thematic module used for the creation of the tag, e.g. "IndoorGML:Navi:*".
- External references are defined via the "IndoorGML:Ext_Ref:*" tag namespace, where, as per the conceptual model proposal, the values are plain text strings defining the referenced dataset, object, and source or URL.
- Finally, the IndoorGML derived OSM dataset needs to be reviewed alongside the existing main OSM DB dataset which is used for "outdoor" data, and connections need to be established between the Anchor space/node "indoor" OSM equivalents and "outdoor" OSM "entrance" tagged features.

5.10 Conflation – Conceptual Model UML Diagram

In order to better represent the structure of the Conceptual Conflation Model that was outlined throughout this section, it was deemed helpful to create a UML diagram representation of the complete data model.

In order to best represent all elements, their attributes are also defined in the diagram, in the sense that the tags which are explicitly specified during the previous subsections are added as attributes to their respective features. However, an additional "open" Tags attribute is added to a common super class of OSM features, which can accommodate for any further semantic information added via use of tags, for example regarding wheelchair accessibility, or other additional information.

Based on the above, the UML diagram was designed as follows:



<<AbstractFeature>> AbstractOSMFeature
+ Tags : <string,string> []
<<Closed Way>> CellSpace_CellSpaceBoundary
<<Closed Way>> Subspace_CellSpace_CellSpaceBoundary
<<Relation>> PrimalSpaceFeatures
<<Relation>> IndoorFeatures
<<Relation>> SpaceLayer
<<Relation>> MultiLayeredGraph
<<Relation>> InterLayerConnection
<<Relation>> Subspacing
<<Node>> State
<<Way>> Transition
<<Relation>> Route
<<Node>> RouteNode
<<Way>> RouteSegment
Closed Way - consists of 2D surface projection of IndoorGML CellSpace/CellspaceBoundary features
Either use of existing State & Transition features, or creation of new RouteNode and RouteSegment elements, independent from "indoor" OSM dataset
<<Enumeration>> NavigableType
<<Enumeration>> NaviClassType
<<Enumeration>> HighwayType
<<Enumeration>> TopoExpressionType
<<Enumeration>> OmniclassClassType
<<Enumeration>> OmniclassFunctionType
<<Enumeration>> OmniclassUsageType
Applicable Values from OmniClass & CityGML Codelists - Appendix III

Figure 34 - Conceptual Conflation Model (OSM) - UML Diagram

As a note, a UML diagram for the OSM Data Model, available in the OSMSharp wiki (an OSM routing and optimization tool), was used as a reference for the basic element UML representation:

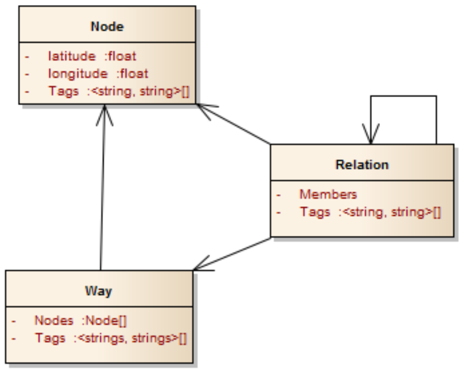

Node
- latitude :float
- longitude :float
- Tags :<string, string>[]
Relation
- Members
- Tags :<string, string>[]
Way
- Nodes :Node[]
- Tags :<strings, strings>[]

Figure 35 - OSM Data Model (Primaries) - OSMSharp Github Wiki Page - https://github.com/OsmSharp/ui/wiki/OpenStreetMap-data-model)

5.11 Conflation – Conclusion

Reviewing our initial IndoorGML Concept to OSM Element mapping table, we can update its contents as follows:

	IndoorGML Concepts							
OSM Elements	**Cellular Space**	**Geometric Representation**	**Topological Representation**	**Semantic Representation**	**MLSEM Representation**	**Subspacing**	**Anchor Nodes**	**External Referencing**
Nodes	-	X	X	-	-	-	X	-
Ways	-	X	X	-	-	-	X	-
Relations	-	X	X	X	X	X	-	-
Tags	-	X	X	X	X	-	X	X

Table 4 - IndoorGML Concepts / OSM Elements - revised / final mapping

Based on the above table, as well as the Feature to Feature mapping and workflow of our conflation data model concept, we can see that a fairly good representation of IndoorGML data can be achieved within OSM, at the very least in regards to obtaining a useful dataset for indoor navigation.

There are naturally some caveats, for example the lack of support for true cellular space representation and consequently subspacing, as well as the limited options for conveying the MLSEM based multiple layer approach to the single data “layer” representation of OSM.

Regardless, as experience has shown with other proposals for an OSM Indoor / Building mapping data model, the main challenge is acceptance and widespread use of any proposed schema by the extended OSM community, which is capable of more accurate review, particularly in regard to potential technical issues that might arise by our data model / schema definition which are not immediately apparent during the conceptual development phase.

6 Proof of Concept Implementation

As mentioned in Section 3 the 1st process concept was chosen during the early stages of the thesis work as the most suitable for achieving the thesis goals of developing a solution for combined indoor outdoor routing.

A conceptual process for the migration of IndoorGML data to the OSM data model was defined in Section 5. However, as the available IndoorGML source data is not defined as per the OGC standard, a different process was used for the creation of OSM data that will work as part of our proof of concept implementation. As explained, external factors also played a role in our definition of the target OSM "schema", such as compatibility of the data with the OTP routing engine and Nominatim Geocoder.

Additionally, certain modifications in the source code of the OSM tools (OTP and Nominatim, original source code available at https://github.com/opentripplanner/OpenTripPlanner and https://github.com/twain47/Nominatim) were made, with the purpose of making the tools more suitable to our goal of combined indoor/ outdoor routing. These modifications will also be examined in this section.

The current section will begin by comparing the conceptual conflation model to the data model used for the proof of concept implementation, in order to identify the reasoning behind differences in the conceptual and implementation process workflows.

We then continue by examining the platforms and tools that have shown potential and were used as parts of the implementation of a multi-modal, context-aware routing engine for combined indoor/outdoor datasets, as per the scope of this master thesis.

Following this, we will provide an overview of the proof of concept implementation setup, including a description and specifications of the Linux Virtual Machine that was used to host the local instances of the tools used in the implementation.

An analysis of the IndoorGML dataset that was used for the implementation is covered afterwards, followed by a quick overview of a complementary CityGML LoD4 dataset that was used for additional semantics and the mock GTFS Transit dataset which enabled multimodal Transit routing.

Finally, a complete overview of the data migration process via FME and JOSM is provided, alongside an overview of data processing and modifications for Nominatim and OTP. A graphical representation of the workflow is also provided, along with a visual guide through the workflow and testing process steps. The sections closes with a review of the results produced from the proof of concept implementation, and the conclusions derived and experience gathered from the complete process.

6.1 Conflation Conceptual Model and Proof of Concept Implementation comparison

In the previous section, a conceptual model for storage of IndoorGML data in the OSM XML format was defined. As the current section will cover the Proof of Concept Implementation that was created in order to test the feasibility of using IndoorGML data in an OSM-based routing solution with multi-modal capabilities and context awareness, we must note certain differences in our approach.

These are covered briefly in the table below, followed by a more detailed analysis:

Conceptual Model	**Proof of Concept Implementation**
Source data: Idealized complete dataset based on IndoorGML OGC standard schema	Source data: Custom MLSEM driven data model implementation, subset of complete IndoorGML data model
Target data: Idealized OSM representation with coverage of as much of the IndoorGML data model as possible, usage of all OSM data primaries	Target data: Routing capable OSM representation, usage of OSM nodes & ways + tagging for network creation
Constraints: Driven by IndoorGML conceptual model	Constraints: Driven by routing engine constraint support
No external restrictions considered	External restrictions considered (FME Reader/Writer capabilities, Routing engine feature & tag support)

Table 5 - Proof of Concept Implementation and Conceptual Model comparison

These differences stem initially from the available IndoorGML source data. The IndoorGML dataset available from the chair of Geoinformatics **(Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014)**, does not constitute a model IndoorGML dataset, in the sense that it does not utilize all available modules or contains data for all available feature types. More importantly, the structure used (include feature types, names, etc.) is not the same as the one defined in the OGC Standard for IndoorGML. It is instead a somewhat custom implementation based on the MLSEM definition (**Nagel, C., 2014**). A more complete analysis of the source dataset is provided in the next section.

Regarding the target OSM data, there are again differences between the implementation and the conceptual model described previously. The main reason is the aim towards an implementation that meets our criteria for routing capable OSM data, the main pre-requisite for which is the definition of a node & way network that can be used for routing. Semantics are stored where necessary to enable a certain level of context-awareness and include support for multiple modes of transport. The implementation data model does not attempt to fully capture the complete data stored within an OGC standard compliant IndoorGML dataset, but to derive a working network with certain semantics out of the available custom MLSEM implementation.
Additional limitations to the complexity of the utilized OSM data come from the specific functionality constraints of the OSM tools used for the implementation, namely the OTP routing engine and Nominatim geocoder. The use of tags and relations was limited to what the tools support and require in order to achieve the desired functionality, instead of attempting to completely rework the tools, so as to enable full support of the proposed conceptual model. Naturally, this has a direct influence on how context awareness and semantic data are handled, as our limitations are mostly derived by the limitations of the tools used. However, the most capable tools (based on the thesis requirements) were selected, in order to ameliorate this issue as much as possible.

6.2 Relevant platforms for the Proof of Concept Implementation

The platforms are presented in no specific order, as, depending on the platform, they may cover multiple different steps of the implementation (e.g., data storage, routing, visual representation, etc.).

6.2.1 Oracle Spatial & Graph platform

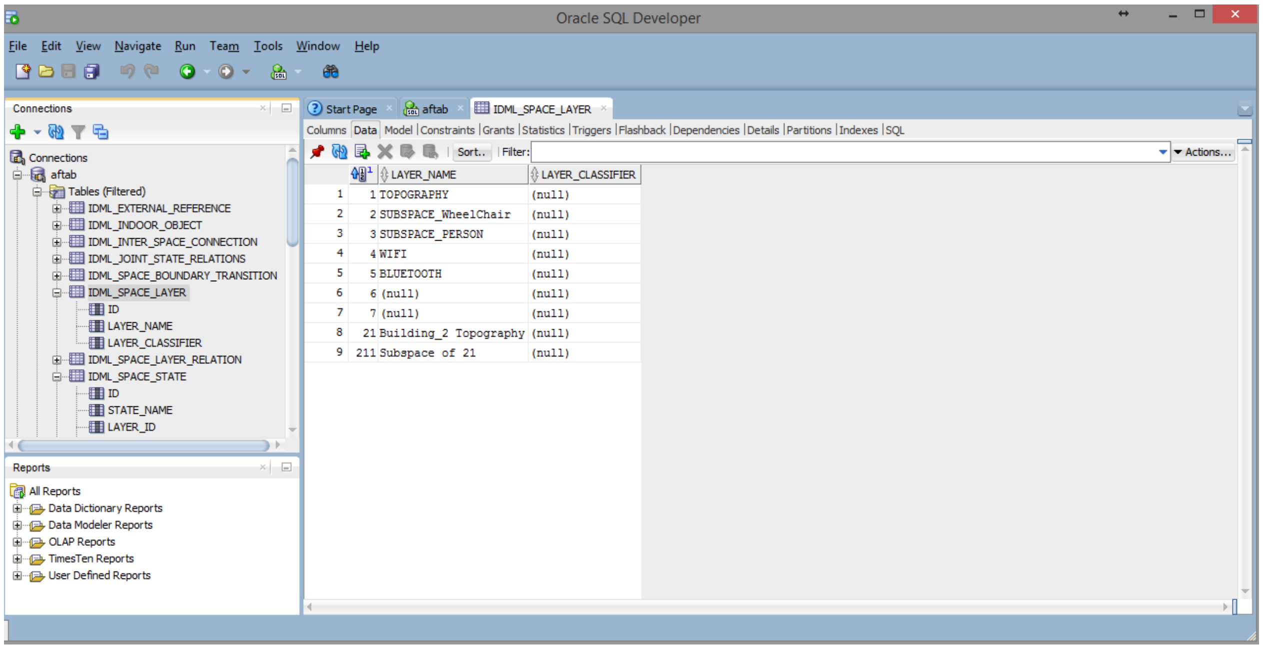
Oracle Spatial & Graph is an additional component for the Oracle Enterprise database platform. Oracle Spatial provides additional functionality to the Oracle object-relational database, enabling its use for storage and management of location and geographic data. This allows for the implementation of GIS applications which take advantage of the performance and support of the well-established Oracle database management system. The Graph components of Oracle Spatial & Graph allow for use of Resource Description Framework (RDF) semantics graphs as well as network graphs based on the Oracle Network Data Model (NDM)(Oracle, 2014).

In addition to a host of geospatial data features, the Oracle Spatial & Graph platform also has a built-in geocoding and routing engine, allowing for the quick implementation of customized solutions in the area of routing and navigation. It is worth noting that Oracle Spatial and the built-in routing engine has been successfully utilized for routing within an IndoorGML (MLSEM) dataset in related research (**Donaubauer, A., Straub, F., Panchaud, N., & Vessaz, C., 2013**).

One point of note is that the test TUM building IndoorGML dataset that is to be used in this thesis was stored and available in an Oracle Spatial database. The IndoorGML dataset was based on IFC data which was converted to IndoorGML (with an interim conversion to CityGML LoD4), as part of research in the creation of a transformation process for the generation of indoor routing graphs from an existing semantic 3D building model **(Khan, A. A., Donaubauer, A., Kolbe, T. H., 2014)**.

The dataset consists of the 3D representation of the building states/cell spaces, the point nodes for the spaces as well as the edges network connecting the point nodes and signifying the boundary transitions between states. Additionally, tables containing the different available layers for the MLSEM, list of indoor objects and a generic attribute container for the additional semantic information of each state/cell are included in the dataset. The complete dataset structure is available under Appendix I and examined in the corresponding part of the proof of concept workflow, namely in section 6.4. A partial dataset was additionally provided as an ESRI file geodatabase, and was initially used in the development of the proof of concept.

Oracle developed tools that were used to review the data were the Oracle Instant Client, Oracle SQL Developer tool and the Oracle SQL Developer Data Modeler, all available from Oracle's website (Oracle Instant Client and Oracle SQL Developer / SQL Developer Data Modeler were downloaded from http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html and http://www.oracle.com/technetwork/developer-tools/index.html respectively).



		LAYER_NAME	LAYER_CLASSIFIER
1	1	TOPOGRAPHY	(null)
2	2	SUBSPACE_WheelChair	(null)
3	3	SUBSPACE_PERSON	(null)
4	4	WIFI	(null)
5	5	BLUETOOTH	(null)
6	6	(null)	(null)
7	7	(null)	(null)
8	21	Building_2 Topography	(null)
9	211	Subspace of 21	(null)

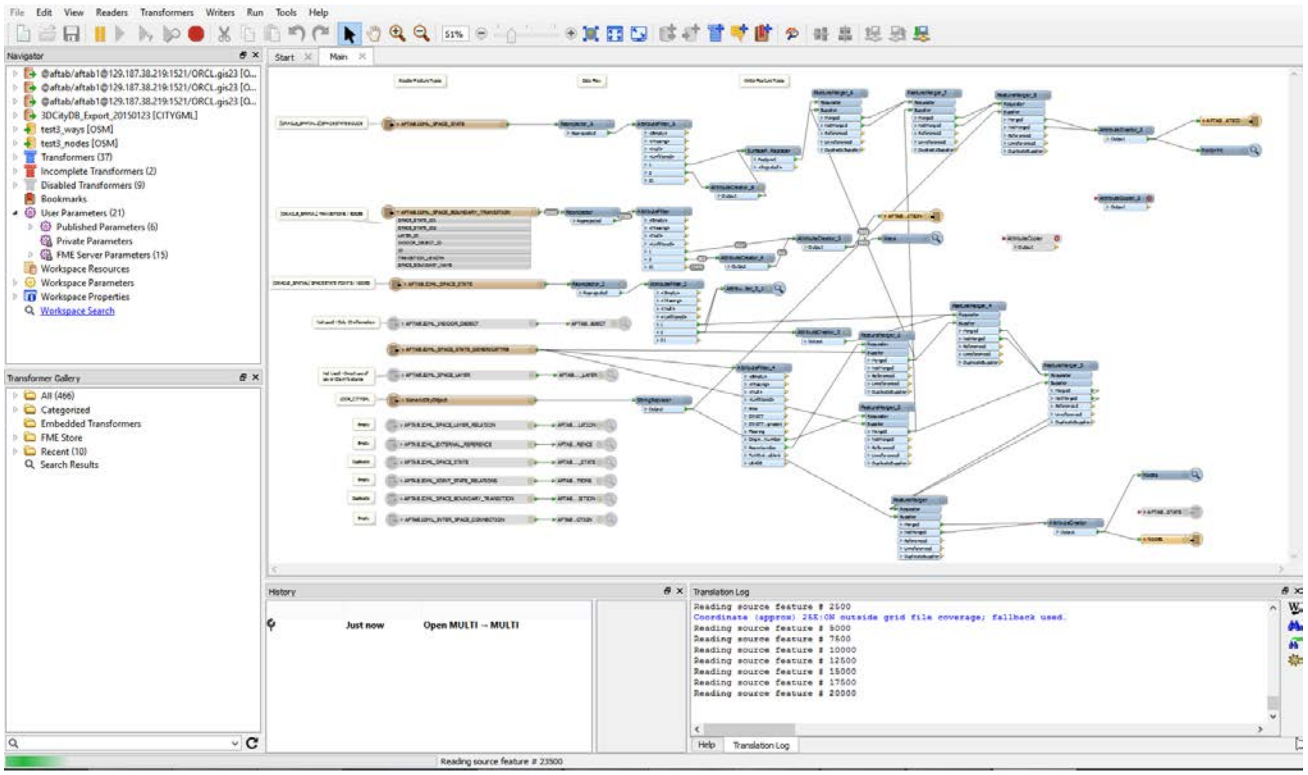
Figure 36 - Oracle SQL Developer environment - connected to IndoorGML source data Oracle Spatial DB

6.2.2 Safe Software FME

Safe Software's FME (Feature Manipulation Engine - http://www.safe.com/fme/) is a Spatial ETL (Extract, Transform, Load) suite which allows for manipulation and conversion to and from over 325 formats (**Safe Software, 2015**), via a graphical interface. Workflows are defined via a drag & drop interface that allows for the addition of data readers, transformers and writers, allowing for a quick setup of complex data manipulation, without any coding knowledge requirements, as is often the case for such processes in other tools.

The main workload in FME for the implementation is done via the FME Workbench application, through which the data manipulation tasks are defined and managed. The FME Data Inspector application was also used for testing of the workbench output and resulting data.

Screenshots of the specific workbenches defined and used will be provided in Appendix I, while the actual process will be further explained in section 6.7.


File Edit View Readers Transformers Writers Run Tools Help
Navigator
Transformer Gallery
History
Translation Log
Just now Open MULTI → MULTI
Reading source feature # 23500

```
Reading source feature # 2500
Coordinate (approx) 25E:0N outside grid file coverage; fallback used.
Reading source feature # 5000
Reading source feature # 7500
Reading source feature # 10000
Reading source feature # 12500
Reading source feature # 15000
Reading source feature # 17500
Reading source feature # 20000
```

Figure 37 - FME Workbench

6.2.3 Routing Engine – OpenTripPlanner (OTP)

OpenTripPlanner (OTP) is the OSM routing engine selected for use with our proof of concept implementation. The project launched in 2009, starting as a trip planner implementation for Portland, Oregon's TriMet agency. It has since advanced to a full featured multimodal trip planner (OTP) and transportation network analysis platform (OTP Analyst). As an open-source project, OTP is oriented towards open data standards, such as OpenStreetMap (OSM) and General Transit Feed Specification (GTFS). It additionally supports some other industry standard formats, such as Shapefiles, GeoTIFF.

Being Java based, OTP can be run on any OS with a working Java Virtual Machine installed. For the purposes of the thesis project, it was installed on a Virtual Machine running Ubuntu Server 14.04. OTP is available either as a prebuilt stand-alone JAR package that can be used directly to build a network graph, run the necessary routing engine it and serve it as web service via a built-in Java Grizzly server. Apart from the server, the JAR also includes a preconfigured web client and GUI based on Leaflet, an open source Javascript library for interactive map representation, as well as a REST-based API that can be used for external querying by third party applications.

In the case that modifications are necessary, the source of the entire project is available on Github, so that users or developers can clone the repository and build their own JAR files, including any possible customizations or modifications to the source code. As part of the thesis work, the stand-alone JAR packages were initially used for testing, before moving to a modified version of OTP built from source (Master branch, version 0.19).

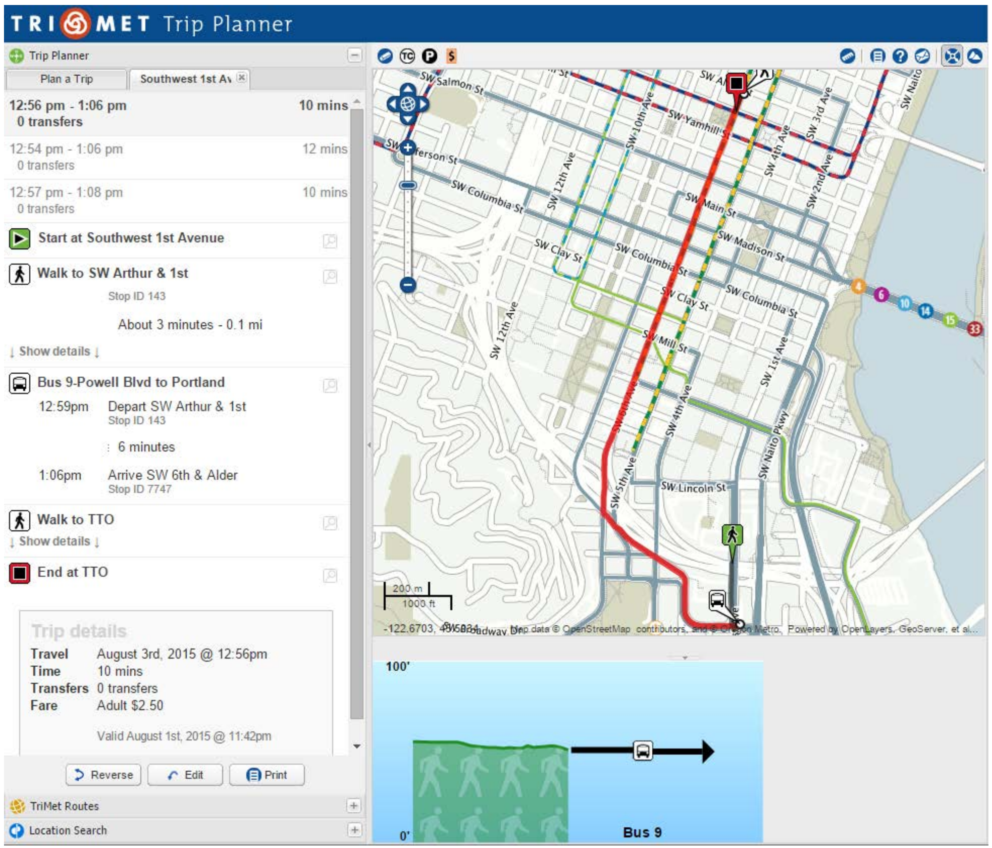
Apart from the support for the desired formats (OSM), ease of initial setup and installation and ready to use web interface, OTP has several other features which made it the preferred routing engine for the proof of concept implementation: OTP has very advanced multi-modal planning support, including **Transit, Bus Only, Rail Only, Bicycle Only, Bicycle & Transit, Walk Only, Drive Only, Park and Ride, Kiss and Ride, Bike and Ride** as possible modes of transport. When combined with an accurate and complete OSM and GTFS dataset, OTP is an extremely powerful tool for multi-modal trip planning. As per the goals of this thesis, multimodal capabilities are rather important for a seamless indoor/outdoor routing solution, as it is most likely that at least a segment of the outdoor navigation will utilize a mode of locomotion other than walking.

In regards to its suitability for indoor navigation, OTP is one of few routing engines which directly support vertical movement via elevators, including detailed route instructions (e.g. boarding and alighting). In addition, OTP developers are examining support for open area navigation, when no specific routes are defined (e.g., a floor space or foyer).

Context awareness in the routing implementation is another important goal of the thesis. OTP shows promising capabilities for context awareness via out of the box support for constraints such as wheelchair accessibility, time of travel selection so that the appropriate route options are displayed when using transit / public transportation, terrain steepness awareness for walking directions, as well as a preference selector for bicycle routes based on three factors (Quickness, Flatness, Bike Friendliness). If the appropriate information / datasets are available (e.g., a local DEM and accurate OSM street tagging), the OTP routing engine is immediately capable of meeting several of our criteria for context awareness, while also providing the option for further adaptation or modification, in order to further expand context awareness.

One other important aspect is the active user and developer community of the project, which gave it a significant advantage over other smaller or discontinued projects, as support was needed and received, especially when examining modifications to OTPs base functionality in order to achieve the goals set for the proof of concept implementation and thesis work.

Overall, OTP offers a well-rounded, highly customizable and capable solution for OSM and open data based routing, which is also evidenced by the use of it in several multimodal trip planner implementations by various state agencies, such as New York State's Department of Transportation and Portland, Oregon's TriMet agency in the US, as well as agencies in France, Spain, Poland and Italy (an up-to date list is available under the OTP documentation's Deployments section).



12:56 pm - 1:06 pm 10 mins
0 transfers

12:54 pm - 1:06 pm 12 mins
0 transfers

12:57 pm - 1:08 pm 10 mins
0 transfers

Start at Southwest 1st Avenue

Walk to SW Arthur & 1st
Stop ID 143
About 3 minutes - 0.1 mi
Show details

Bus 9-Powell Blvd to Portland
12:59pm Depart SW Arthur & 1st
Stop ID 143
6 minutes
1:06pm Arrive SW 6th & Alder
Stop ID 7747

Walk to TTO
Show details

End at TTO

Trip details

Travel	August 3rd, 2015 @ 12:56pm
Time	10 mins
Transfers	0 transfers
Fare	Adult $2.50
	Valid August 1st, 2015 @ 11:42pm

Figure 38 - Portland Tri-metropolitan area OTP powered multimodal route planner (http://ride.trimet.org/#/ , 2015)

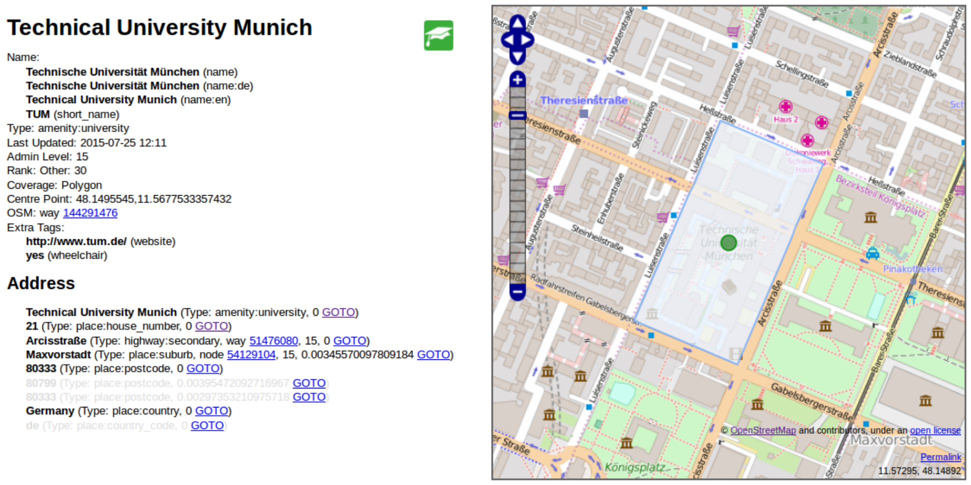
6.2.4 Geocoder – Nominatim

Nominatim is an open-source geocoding engine that is used alongside OTP to complement the routing engine with geocoding and facilitate the selection of trip start and end locations, both in the indoor and outdoor segment.

Geocoders in general allow for the identification of a point based on text input. More precisely, they provide x,y coordinates or other information (e.g. OSM node/way ids) by name or address input. Nominatim also supports reverse geocoding, returning the name or address of a location based on the input coordinates. Nominatim is based around the osm2pgsql utility, creating a PostgreSQL database with an index of all searchable locations, using a combination of C, plpgsql and php for communication with clients, such as OTP's Leaflet client.

As it is mostly aimed for outdoor datasets, Nominatim has built-in support for building and highway tagged features. However, its source code can be modified to add other feature types (tagged elements), such as buildingpart, building:part and room features in the search index, in order to be used for indoor geocoding. When it comes to indoor environments, use of a geocoder is important for the correct selection of trip start and end points, as the 2D representation of most OSM clients (including OTP's Leaflet client) does not directly allow for the selection of a specific room or floor, due to the general lack of z coordinates in OSM features, as well as height or level based filtering of features in the clients' 2D data representations.

Nominatim was selected as the geocoder for our proof of concept implementation due to its popularity (Nominatim is used as the geocoding provider for the main OSM webpage), extensive documentation and active community support, which was particularly helpful when it came to extending the base geocoder functionality with the inclusion of further tags / features for use in indoor geocoding.



Technical University Munich

Name:
- **Technische Universität München** (name)
- **Technische Universität München** (name:de)
- **Technical University Munich** (name:en)
- **TUM** (short_name)

Type: amenity:university
Last Updated: 2015-07-25 12:11
Admin Level: 15
Rank: Other: 30
Coverage: Polygon
Centre Point: 48.1495545,11.5677533357432
OSM: way 144291476
Extra Tags:
- **http://www.tum.de/** (website)
- **yes** (wheelchair)

Address

- **Technical University Munich** (Type: amenity:university, 0 GOTO)
- **21** (Type: place:house_number, 0 GOTO)
- **Arcisstraße** (Type: highway:secondary, way 51476080, 15, 0 GOTO)
- **Maxvorstadt** (Type: place:suburb, node 54129104, 15, 0.00345570097809184 GOTO)
- **80333** (Type: place:postcode, 0 GOTO)
- 80799 (Type: place:postcode, 0.00395472092716967 GOTO)
- 80333 (Type: place:postcode, 0.00297353210975718 GOTO)
- **Germany** (Type: place:country, 0 GOTO)
- de (Type: place:country_code, 0 GOTO)

Figure 39 - Nominatim instance created for thesis work – Search result "Details" view – built-in client

6.3 Implementation Setup

In order to create the proof of concept implementation, multiple systems were used. The source IndoorGML data was retrieved from an Oracle spatial database hosted by the TUM Chair of Geoinformatics. Retrieval and processing of the data was done via Safe Software's FME on a separate personal workstation. OSM Data for the "outdoor" dataset was also retrieved via the JOSM OSM editor on the workstation, and processed alongside the indoor data via FME. The complete target dataset was then merged via JOSM in a single file, in preparation for final processing by the geocoding and routing engines.

For the purpose of hosting a local instance of the routing and geocoding engines, a Virtualbox Virtual Machine was setup on the personal workstation. Due to the java based OpenTripPlanner routing engine and the inclusion of a Desktop GUI in the VM, the VM was created with above average specs, listed below:

OS: Ubuntu Server 14.04 64bit Virtual Machine + Unity Desktop
CPU: Intel i7-4770K (1 core allocated)
RAM: 16GB RAM allocated
HDD: 100 GB fixed-size VHD Virtual drive

The large HDD size allowed for the storage and testing of multiple OSM datasets, ranging from a few MB (Localized TUM extract) to Country-level extracts (Germany). The high amount of RAM was added to ensure smooth operation of the Java based OTP routing engine and client. In retrospect, a 4GB RAM allocation would likely suffice for a single user demonstration with a small dataset and corresponding graph, alongside the Ubuntu Unity Desktop.
For OTP, the v 0.19.0 Snapshot was used, initially by using the stand-alone jar release for v0.19. However, due to the need for more in-depth understanding and customization of the client and engine, a local build of OTP from the source (available on Github) was preferred, as per the instructions from the OpenTripPlanner documentation pages.

The Nominatim Geocoder was likewise built from source (again available via Github), as per the instructions available via the Nominatim wiki. The creation of the local instance allowed for some necessary customizations to the source code and core functionality of the geocoder, to better accommodate for its use for indoor geocoding.

A personal Dropbox account was additionally used to facilitate data exchange between the VM host workstation and the VM itself. Processing of the OSM data by the routing graph and geocoder indexing engines was done entirely in the VM. Both processes are relatively resource intensive with typical datasets. Luckily, the small dataset used for the purposes of this demonstration (merged Indoor/ Outdoor OSM dataset – 7.72MB) was processed relatively quickly, at half a minute and three minutes for the OTP Graph and Nominatim Index respectively.

It should be noted that the time required changes drastically with larger and/or more complex datasets. For comparison purposes, an OSM extract of the city of Munich required 5 minutes for the OTP Graph and 30 minutes for the Nominatim Index.

After the OTP graph and Nominatim Index are successfully created, the built-in OTP client can be started from the VM. It is then accessible via a browser on the VM, as well as any clients within the same network, including mobile devices. A visual representation of the implementation can be seen below:

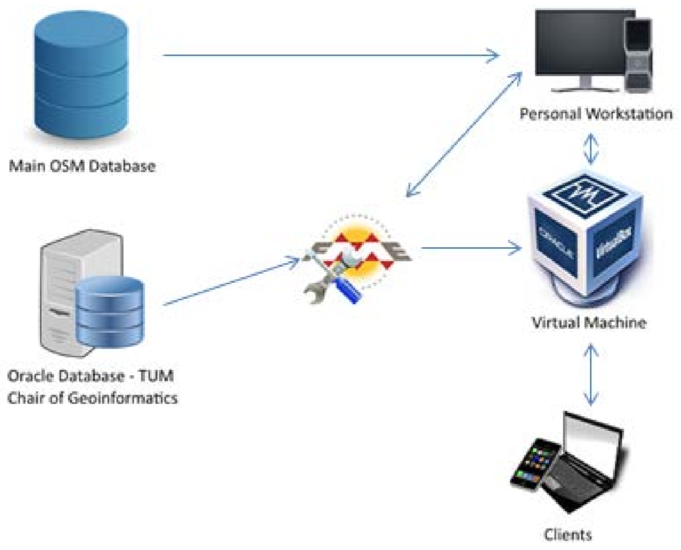


Figure 40 - Proof of Concept Implementation

6.4 IndoorGML Source data

The IndoorGML source data was initially received in an ESRI file Geodatabase, which was used for initial testing and setup of the necessary workspaces. However, we were made aware of a more complete IndoorGML dataset, available via an Oracle database. The process examined below is based on the Oracle IndoorGML dataset.

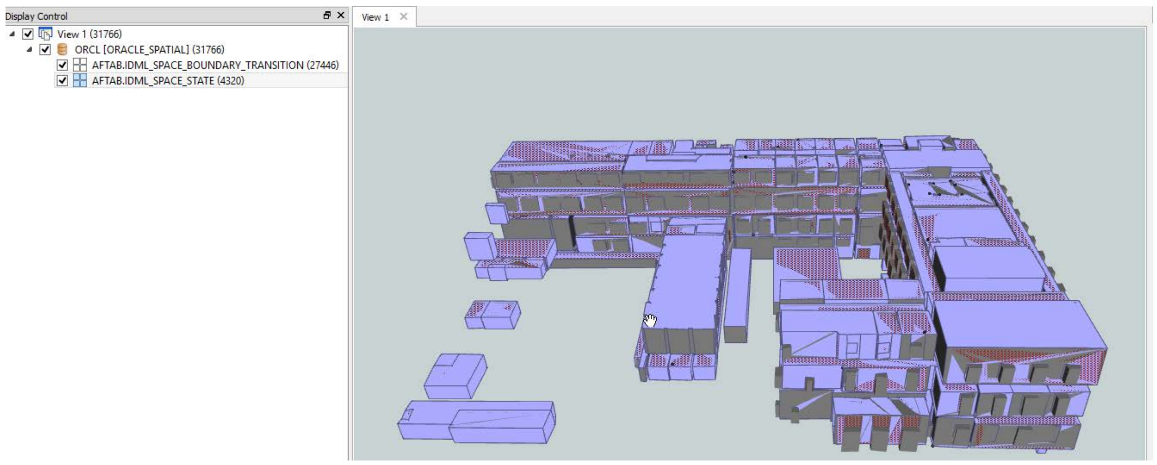

Display Control
View 1 (31766)
ORCL [ORACLE_SPATIAL] (31766)
AFTAB.IDML_SPACE_BOUNDARY_TRANSITION (27446)
AFTAB.IDML_SPACE_STATE (4320)
View 1

Figure 41 - FME Data Inspector view of the IndoorGML dataset spatial elements

The source data used a series of tables, including Oracle Spatial and non-spatial information. The main differentiator between the two types of data is the existence of data with the SDO_Geometry type, used for storing geometry information. An overview of the database structure is available in Appendix I, Figure VII. The dataset was created by Aftab Khan, as part of his work with the TUM chair of GeoInformatics in (**Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014)** and represents Building 7 of the Main TUM Building in Arcisstrasse.

The main elements used for the storage of IndoorGML data are the following tables (in alphabetical order):

IndoorGML dataset Table Name	Has Geometry	Has Data
IDML_EXTERNAL_REFERENCE	-	-
IDML_INDOOR_OBJECT	-	X (ID only)
IDML_JOINT_STATE_RELATIONS	X (x2)	-
IDML_SPACE_BOUNDARY_TRANSITION	X (x2)	X
IDML_SPACE_LAYER	-	X
IDML_SPACE_STATE	X (x2)	X
IDML_SPACE_STATE_GENERICATTRB	-	X

Table 6 - ORACLE IndoorGML dataset content

Reviewing the structure of the database, we can see that the database is built around the IDML_INDOOR_OBJECT table. All indoor elements are assigned with an id serving as the Primary key (and used in most of the other tables to connect elements to this table), along with additional columns for storage of semantic data (GML ID & Namespace, creation and termination date and author), derived from the initial CityGML dataset which was used to generate the IndoorGML data. It is worth noting that no geometry is stored in this table, serving only as an index for all the elements of the dataset. Additionally, in the provided dataset, the columns for the GML derived data are empty, as a result, only the ID value is populated.

IDML_SPACE_LAYER is the table defining IndoorGML's multiple layers. Each layer has an entry in the ID column, which is also the table's primary key. A LAYER_NAME column allows for the definition of the name, allowing for the easy identification of specific layers, while an additional LAYER_CLASSIFIER column exists for further classification. In the provided dataset, the last column was not populated with any values, and was not used further as a result.

The IDML_SPACE_STATE table is used for the definition of the possible states (nodes or spaces in dual and primal space) within the IndoorGML dataset. An ID column is also present here, functioning as the table's primary key. The LAYER_ID and INDOOR_OBJECT_ID columns are the two foreign keys connecting the states with their respective layers and indoor objects from the previous two tables. A STATE_NAME column allows for the definition of a name, while there are also two geometry columns, SPACE_GEOMETRY and STATE_GEOMETRY, defining the primal and dual space geometries of the State objects. It should be noted that not all layers from the IDML_SPACE_LAYER table were utilized, as only the topography and wheelchair subspace were considered relevant for the proof of concept implementation objectives. Additional layers that had elements were a person subspace layer (ID=3) which had a too small number of elements to generate any navigable network (29 States, no transitions), a Bluetooth layer (ID=5, 5 States, no transitions), a subspacing layer based on the topography which was not using the same projection as the rest of the data, likely resulting from subspacing tests (ID=21) and an unnamed layer (null value in the name column, ID = 6) which also only had few elements (61 States, no transitions). As the combined number of objects for Layer IDs 3,5 & 6 was 95 states, compared to 838 and 242 state objects from layers 1 and 2 (topography & wheelchair, respectively), it was deemed ok to not utilize them further in the implementation.

The IDML_SPACE_BOUNDARY_TRANSITION table provides the transitions between states within the dataset (boundaries and edges in primal and dual space). Two columns, SPACE_STATE_ID1 and SPACE_STATE_ID2, provide the start and end states, connecting the table as foreign keys with the IDML_SPACE_STATE table. The LAYER_ID column provides an additional connection via foreign key to the layer the transitions belong to and the IDML_SPACE_LAYER table. A primary key ID column also exists in this table. An INDOOR_OBJECT_ID column also exists; however it is not populated with data, or linked with the IDML_INDOOR_OBJECT via a foreign key. 2 columns contain the geometry of the boundary transitions, SPACE_BOUNDARY_GEOMETRY and TRANSITION_GEOMETRY for the primal and dual space, respectively. A SPACE_BOUNDARY_NAME column contains the names of the boundary transition objects, while their respective lengths are stored in the TRANSITION_LENGTH column. As with the IDML_SPACE_STATE, not all layers were available, and the same two layers were utilized for the proof of concept source data.

The IDML_SPACE_STATE_GENERICATTRB table was defined in order to provide additional semantic information for the various indoor objects. It is important to note that no Primary or Foreign keys were defined in this table, making the connection of the relevant info to the corresponding objects a manual process. An ID column is again included, possibly intended for a connection to the IDML_INDOOR_OBJECT table. However, while working with the data, the SPACE_STATE_ID column that is also included provided better results when attempting to match the attributes to the features. The attributes themselves are entered via use of ATTRNAME and DATATYPE columns, which specify the name and a codified value type of the attribute. The values are entered via separate columns for each data type, namely INTVAL, REALVAL, RUIVAL, DATEVAL and STRVAL. Out of these types, only the STRVAL column is used for string attribute values, even in the case of attributes like “Area”. The corresponding DATATYPE column value is also 1 for all table rows.

The IDML_EXTERNAL_REFERENCE table allows for assignment of external reference information to any indoor object ID (Primary + Foreign key matched to elements of the IDML_INDOOR_OBJECT table). The Information System referenced can be specified, along with a Name and URI. In the available dataset, this particular table did not include any information and was subsequently not used further.

The IDML_JOINT_STATE_RELATIONS table allows for the definition of joint state relations of specific elements. An ID column exists as a Primary key but was not directly connected to the Indoor Object table. In addition, columns exist to for the entry of Space State IDs of up to 5 layers, along with two separate geometry columns, for intersecting nodes and solids respectively. This table also does not include any data and was not used for the proof of concept implementation of this thesis. It is possible that the lack of data is also the reason no connection between this table and IDML_INDOOR_OBJECT or other tables was specified.

The above completes our overview of the available source data in the IndoorGML dataset. An additional CityGML dataset was provided by the chair which will be examined in the next subsection.

6.5 Complementary CityGML LoD4 data

In addition to the IndoorGML dataset outlined in the previous section, a CityGML LoD4 dataset was provided, in order to enhance the existing data with further semantic information, where possible. The data was created as part of the process converting IFC and CityGML LoD4 data to IndoorGML (**Khan, A. A., Donaubauer, A., & Kolbe, T. H., 2014)** and was supplemented with additional semantic info as part of a project work on automatic creation of 3d points of interest via CityGML LoD4 files (**S. Bauer, T. Vogl, 2014**).

The CityGML dataset is an LoD4 representation of Building 7 of the Main TUM Building in Arcisstrasse, same as the IndoorGML dataset. The Core and Building modules of CityGML were used for the initial dataset, which was expanded via use of the Generic module, in order to add GenericCityObject POI features. In order to accommodate two types of features, they used CityGML's capacity for generalization and specialization, creating a GenericCityObject feature class named POI_Room, along with a specialized child class named POI_Door. Under those two, various semantic information was stored. Specifically, the following attributes were added:

- gml_id
- Raumnummer
- Gebäudenummer
- Raumname
- Lehrstuhl
- Stockwerk
- Standort
- Adresse
- Fläche
- DIN277 Untergruppe
- TUM-Nutzung
- Belegungsplan
- Bodenbelag

This information was added to the IndoorGML dataset by merging the relevant features and copying the attributes from the CityGML dataset to the matched IndoorGML features. The exact process is described in the FME workflow, in section 6.7. It should be noted that only the GenericCityObejct features (POI_Room and POI_Door) were utilized. In addition, the geometry of the features was ignored, as the geometry of the IndoorGML features (states) was used instead.

6.6 GTFS Transit Data

OTP supports Transit trips as part of its multi-modal routing. In order to enable Transit, a valid GTFS (General Transit Feed Specification, as defined by Google Developers https://developers.google.com/transit/gtfs/) file needs to be provided alongside the OSM data used for OTP Graph building.

Such files are typically available from transit agencies, such as Munich's MVV. However, data for the Munich transit network is unfortunately not released publicly from MVV. As a workaround, data from the Verkehrsverbund Berlin-Brandenburg (VBB) agency was downloaded (VBB Data retrieved from Berlin Open Data website: http://daten.berlin.de/kategorie/verkehr) and examined alongside the specification reference available via the Google Developers website.

Based on the above, a mock GTFS dataset was created with the two stations nearest to the TUM building of the OSM dataset (Theresienstrasse and Koenigsplatz). A fake agency was created, along with 1 minute routes between the two stations at 5 minute intervals and a daily trip schedule for the second half of 2015. The dataset was correctly parsed by OTP, allowing for successful testing of the multimodal transit capabilities for the outdoor route segment. The mock GTFS dataset is partially available under Appendix I.

6.7 FME Workflow

The majority of the data migration tasks were completed mainly through the use of Safe Software's FME Workbench tool. The precise workflow will be examined in depth in the current section. The relevant Workbench for the main conversion workflow is displayed in Figure IX of Appendix I, however, a graphical representation of the overall workflow can be seen below:

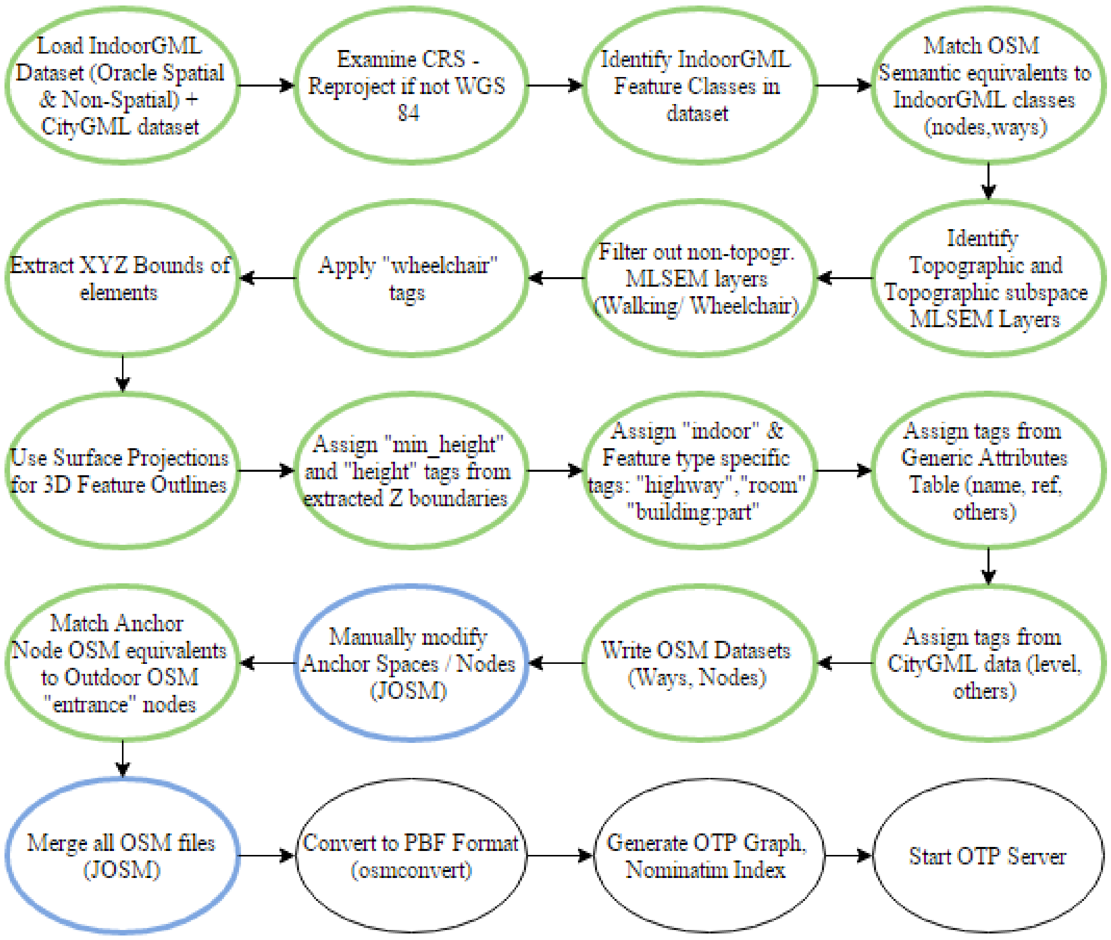

Load IndoorGML Dataset (Oracle Spatial & Non-Spatial) + CityGML dataset
Examine CRS - Reproject if not WGS 84
Identify IndoorGML Feature Classes in dataset
Match OSM Semantic equivalents to IndoorGML classes (nodes,ways)
Identify Topographic and Topographic subspace MLSEM Layers
Filter out non-topogr. MLSEM layers (Walking/ Wheelchair)
Apply "wheelchair" tags
Extract XYZ Bounds of elements
Use Surface Projections for 3D Feature Outlines
Assign "min_height" and "height" tags from extracted Z boundaries
Assign "indoor" & Feature type specific tags: "highway","room" "building:part"
Assign tags from Generic Attributes Table (name, ref, others)
Assign tags from CityGML data (level, others)
Write OSM Datasets (Ways, Nodes)
Manually modify Anchor Spaces / Nodes (JOSM)
Match Anchor Node OSM equivalents to Outdoor OSM "entrance" nodes
Merge all OSM files (JOSM)
Convert to PBF Format (osmconvert)
Generate OTP Graph, Nominatim Index
Start OTP Server

Figure 42 - Proof of Concept Implementation Workflow (Green = FME / Blue = JOSM / Black = Linux OSM Tools)

As we have mentioned, the source data was retrieved from an Oracle database, hosted by the TUM chair of Geoinformatics. One point of note is that by default, such a mixed dataset (Spatial & non-spatial information) cannot be fully read by a single FME reader (the data parsing engines developed by Safe Software as part of the FME series of tools). In order to mitigate that, both an Oracle Spatial and Non-spatial reader were added, in order to process the complete dataset.

As an overview, the below reader setup was used:

FME Oracle non-spatial Reader

IDML_EXTERNAL_REFERENCE	(Disabled, no data)
IDML_INDOOR_OBJECT	(Disabled, only ID data)
IDML_JOINT_STATE_RELATIONS	(Disabled, no data)
IDML_SPACE_BOUNDARY_TRANSITION	(Disabled, no data)
IDML_SPACE_LAYER	(Disabled, only Layer ID and Name data)
IDML_SPACE_STATE	(Disabled, no data)
IDML_SPACE_STATE_GENERICATTRB	

FME Oracle Spatial Reader

IDML_SPACE_BOUNDARY_TRANSITION	(Transition geometry / DS edges + attributes)
IDML_SPACE_STATE	(State geometry / DS nodes + attributes)

FME Oracle Spatial Reader #2

IDML_SPACE_STATE	(State geometry / PS spaces + attributes)

As we can see above, FME Oracle spatial readers were used to retrieve the dual space node (state) and edge (transition) elements from the IDML_SPACE_STATE and IDML_SPACE_BOUNDARY_TRANSITION tables respectively. A second Oracle Spatial reader retrieved the primal space state geometries, again from the IDML_SPACE_STATE table.

The non-spatial Oracle reader was used to retrieve the generic attributes table. Duplicate features from the non-spatial reader were disabled (IDML_SPACE_STATE and IDML_SPACE_BOUNDARY_TRANSITION). Additionally, any tables that did not contain data were also disabled in the reader. The IDML_INDOOR_OBJECT table was also not used, as it only contained the indoor object IDs, which were not necessary, considering the lack of data on the other non-spatial tables. Finally, the IDML_SPACE_LAYER table was not directly utilized in the FME workflow, due to awareness of the layers used.
An additional FME reader was used for the GenericCityObject features (POI_Room & POI_Door) of the CityGML dataset.

As a first step, all geometry containing features (state point and space features, transition edges) are first passed through a Reprojector transformer, in order to be reprojected from the EPSG:31468 CRS to LL-WGS84 which is typically used in OSM data.

The spatial features are then put through an AttributeFilter transformer, where the Layer ID values are used to separate the topography (used for walking) and wheelchair (used for wheelchair access) layers. As OTP considers features wheelchair accessible by default (with the exception of staircases and features tagged with “wheelchair=no”), an attribute creator is used to tag all layer 1 (topography) features with “wheelchair=no”, and all layer 2 (wheelchair subspace) features with “wheelchair=yes”. This allows for the creation of a single graph containing both possible modes of transport and enables wheelchair accessible route selection via the client. While this method creates some overhead by defining two networks of nodes and edges with possible overlaps, it is also the most straightforward way to deal with the use of multiple IndoorGML/ MLSEM layers to store different modes of transport.

In order to mitigate the lack of support for Z values in OSM features, we make use of the “level” tag. Ideally, the semantic information could be used to derive the appropriate floor/level of each element. The room number was initially considered, as the room naming scheme of the TUM includes the floor the room belongs to. However, since features like hallways have no room number assigned, this was deemed insufficient. As a workaround, a BoundsExtractor transformer is used in the features, in order to identify the higher and lower boundary Z values of each object from the geometry enabled features (states, spaces, transitions). State derived elements only receive a “min_height” tag, considering that they can only have a single Z value. Way elements, including both solid derived surface projections/outlines and transitions, have both a “min_height” and “height” tag, in order to represent the solid feature height and any non-horizontal transitions (e.g. steps/ramps). The values are then separated into different streams via an AttributeRangeFilter transformer and their “_zmin”/“min_height” (original transformer and final OSM attribute name) attribute values. The applicable ranges for each floor where initially defined via the use of features with other available floor indicators (e.g. rooms) and then refined by hand in order to more correctly cover the Z Ranges for each floor. The various ranges output by the filters are then fed into another AttributeCreator transformer which assigns the features with their respective level tags.

As OSM does not support 3D solids, the space state features featuring solids geometry are additionally processed with the SurfaceFootprintReplacer transformer which allows for the generation of 2D footprints of the various spaces, which can be saved and viewed correctly in OSM. In regards to conveying the complete dimensions of the solid, the “min_height” and “height” values are stored as tags/attributes in the features from the previous process step.

Finally, the geometry enabled features are fed through a series of FeatureMerger Transformers, in order to match and copy the semantic information available via the non-spatial generic attributes table, as well as the CityGML dataset. An AttributeRenamer transformer is used to rename attributes to known OSM keys. Namely, the “name” and “ref” keys are populated with the original room name and current room name values retrieved from the generic attributes table. The reason for creating these specific tags is that they are used for the Nominatim geocoder index as searchable text strings where addresses are not available (which is naturally the case in the indoor dataset). The AttributeRenamer can also be used at this point to change other attributes to their closest OSM key equivalents if needed. However, as OSM has a free-tagging system, this step is not necessary unless a specific key is required for targeted usage.

The features are then processed by one more AttributeCreator transformer, which assigns an indoor=yes tag to all features, along with feature specific tags (room=yes for the state node features, building:part=yes for the space state footprints and highway=corridor/step/elevator for the edge features). This is done to add the features to the list of indexable OSM features in the Nominatim geocoder. It should be noted that by default, building:part and room features are not indexed.

The differentiation between the possible highway tag values is done via the use of an AttributeFilter Transformer on the Space State elements to separate them based on their STATE_NAME attribute values. The elements are separated between typical transitions ("highway=corridor" tagging) and steps ("highway=steps" tagging) depending on the STATE_NAME values, which is then used alongside two FeatureMerger transformers, merging the transitions (requesters) to their point of origin states (suppliers). The matched features are then tagged separately with the appropriate value for the "highway" tag, before being directed to the OSM writer. It should be noted that as no elevator elements were clearly defined in the IndoorGML dataset, the step for creating "highway=elevator" tags was omitted from the implementation workbench.

As a final step, the features are directed to FME's OSM Writers. It should be noted that when nodes are used as parts of way features (e.g. edges or footprints), the individual nodes retain no information and are only selectable as part of the way feature they belong to. As a workaround, two separate OSM Writers are used to store a ways OSM file containing the linear/way features (dual space transition edges and space state footprints) and a nodes OSM file containing the point/node features (dual space state nodes).

One important task for the correct implementation of indoor/outdoor routing solution is ensuring that the outdoor OSM dataset is correctly connected to the generated indoor OSM dataset. In order to achieve this, a separate FME Workspace was defined, which parses data from the OSM Ways file generated in the previous workflow, as well as the surrounding dataset. The relevant Workbench is displayed in Figure VIII of Appendix I.

Currently, this is achieved via a manual download of the surrounding area data via the OSM webpage or the JOSM client. The data could also possibly be retrieved automatically via a call to the OSM API via FME. However, as the API often rejects queries with a large bounding box or complex data, the manual process was deemed as adequate for the purposes of this implementation.

One additional step that is completed in JOSM is the definition of anchor nodes/ spaces based on the sample IndoorGML dataset. This manual editing is necessary as the sample IndoorGML data does not contain any Anchor Space / Node features. Loading the IndoorGML-derived OSM dataset in JOSM, we can filter for elements with the attribute/value pair "STATE_NAME=Door".

We can then proceed to manually identify door features that can be changed to Anchor Space / Nodes. Once these are located, the first step is the addition of the “IndoorGML:Navi=anchor” key/value pair, which we can then use for filtering. It is then recommended to change the value of the “STATE_NAME” attribute to “Anchor”, for all the features where the “IndoorGML:Navi=anchor” tag was added. While redundant, this tagging allows us to maintain a data structure similar to that of the original dataset.

Once the two datasets are read, they are passed through AttributeFilter transformers. These are used to isolate features from the external dataset that have a value for the “entrance” tag. The reason for not only selecting “entrance=yes” tagged features is that values can be entered to denote main and alternative entrances in a building. The indoor dataset is also filtered similarly, for features where the STATE_NAME attribute has the “Anchor” value (stored in the feature as part of the attributes for space states from the IDML_SPACE_STATE table).
The FME GeographicNeighborFinder transformer is then used to locate indoor candidate Anchor features that are within a pre-defined distance (selected a 5m distance, after trial and error to provide reasonable results) from the outdoor entrance base features. An additional check that could be potentially implemented, is the automated review of the base and candidate feature angles (as provided by the GeographicNeighborFinder transformer), in order to confirm that matched anchor and entrance features are facing in the same direction. As the Anchor features were manually derived for the implementation, this was not a necessary check, however it could be a potential improvement for dealing with fully automated parsing of datasets with Anchor Spaces / Nodes.

Three FME VertexCreator transformers are then used to generate the points and edge connecting them based on the coordinates of the matched anchor and entrance features. An AttributeCreator transformer is used to tag the generated edges with “highway=corridor”, “IndoorGML:Navi=anchor”, and “indoor=yes” tags so that they can be parsed from OTP for any routing request.

The line features are then saved in a separate file via the FME OSM Writer, containing the OSM connecting edges. This is the final step in generating OSM data from the IndoorGML dataset via FME. It should be noted that the merging of the derived OSM datasets into a single final set was completed via use of the JOSM OSM editor. The reason for this was that merging the files directly via FME resulted in issues regarding overlapping features, which are obviously present in the datasets, considering the above workflow.

All the OSM data, including partial and combined datasets is then synchronized over to the VM via use of the Dropbox cloud storage platform.

6.8 Data Processing – Nominatim Index

The OSM XML format our OSM data is available in is not the preferred format for use with OTP and Nominatim. For that reason, the open source osmconvert command line tool is used, in order to convert the files to compressed PBF format. Osmconvert is part of the osmctools Debian package which can be installed directly in Ubuntu Server, so as to avoid building the necessary osmconvert binaries from source. Once the files have been converted to the PBF format, they are ready for processing by OTP and Nominatim. We have previously mentioned that some modifications to the two tools were needed, in order to enable functionality that would help us meet the thesis goals. These modifications will be examined further in this subsection.

Nominatim, the OSM geocoder, works by parsing the OSM data files and creating an index of all searchable features. Out of the features that mostly interest us, Nominatim has coverage of building and highway tags enabled by default, along with most typical outdoor features like POIs, stores, etc. However, features (tags combined with nodes and ways) typically used for indoor mapping, are not included in the standard configuration.

As the Nominatim OSM parser is not configurable via the use of external configuration files or command line to the needed extent, some modifications need to be made to the source code of the project, in order to enable support for such tags. After some failed attempts at modification, the question on how to better implement this support was raised in the Nominatim Github page, where some feedback was provided by a member of the community.

As mentioned, Nominatim is based around the osm2pgsql tool which handles the parsing of OSM data and storage of relevant information for the geocoder's index in a pgSQL database. In order to expand the tag indexing capabilities, the source code of the ***osm2pgsql/output-gazetteer.cpp*** (CPP file source code available in Appendix II, including modifications) file, responsible for the indexing of data, had to be modified. More specifically, the ***place_tag_processor::process_tags()*** function was modified to include "room", "building:part" and "buildingpart" tags in the index.

After the above modifications, both osm2pgsql and Nominatim were rebuilt. Testing confirmed that features tagged as "room", "building:part" or "buildingpart" were now searchable, based on the values of the "name" or "ref" tags that were also defined where available (e.g. rooms).

It is worth noting at this point that the Nominatim geocoder works by creating a "nominatim" table in the pgsql database running at the host machine. In order to build a new index (e.g. when completely changing source data), the existing "nominatim" database needs to be manually removed or renamed. The PGAdmin 3 graphical dB management tool was installed for this purpose on the VM, and used in order to keep backups of old index databases or clean up before creation of a new one. However, the index can be simply updated via use of the appropriate commands, in case of small updates to a large dataset. In our implementation, the dataset was small enough to allow for rebuilding the index for every change.

6.9 Data Processing – OTP Graph

One of the features that we wanted to enable in OTP, was the option for setting a preference towards indoor or outdoor route segments, in order to increase context awareness. In order to implement this, we need to first explain a bit more in depth how OTP actually works.

As with Nominatim, the OSM data is initially parsed by OTP's graph builder. The OSM reader parses the way network of "highway" tagged elements (including streets, steps, corridors, elevators, etc.) and stores them as graph edges. Various types of edges are stored, with specific functions. The various tags assigned to the OSM way network are also parsed, and if applicable for routing purposes, stored as flags. An example of such a flag is wheelchair accessibility, defined by the "wheelchair" tag. In order to also take the "indoor" tag into consideration, some modifications to the source code were necessary, in order to both parse the tag and store it as "flag" for the appropriate edge types.

The source code was examined for every location where wheelchair accessibility was examined, and a similar modification was made, in order to also include the indoor tag and flag. The exact files and functions that were modified are available in Appendix II, with the changes to the code highlighted.

One important difference in the handling of indoor preference and wheelchair accessibility, is that wheelchair accessibility is a binary option, completely allowing or not allowing passage through the appropriate segments depending on the route settings. Naturally, this is not the same for indoor/outdoor preference, as completely avoiding indoor or outdoor segments would not allow for a complete route in any combined indoor/outdoor scenario. In order to implement this, we used OTPs capability to dynamically modify the weights (preference) of routes, depending on input from the user. This is a feature already implemented and used for OTP bicycle trip planner, allowing for modification of the route preference depending on three factors (safety, flatness, speed).

In our implementation, a dropdown box was added to the standard client, allowing for selection of a neutral option (default OTP weighting of route), "prefer indoor" (reduced weight/higher preference for "indoor" tagged segments and increased weight for rest) and "prefer outdoor" (increased weight/lower preference for "indoor" tagged segments and reduced weight for rest). The necessary code modifications to the client are available in Appendix II, with the changes and additions highlighted.

6.10 Other changes – OTP Client

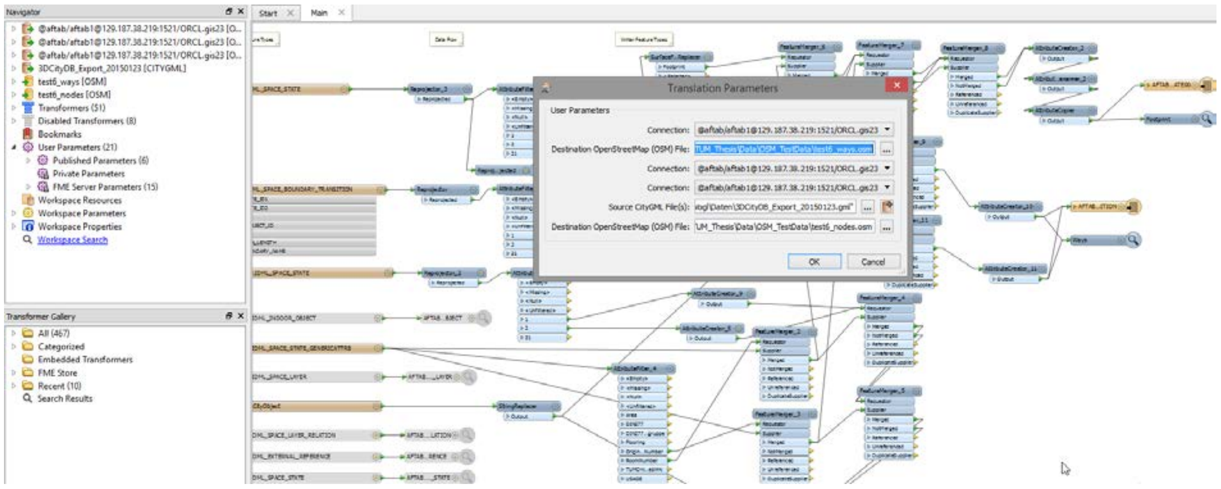
In addition to the previously mentioned modifications, some further changes to the OTP source files was needed, in order to configure the client closest to the needs of the implementation. Enabling the Nominatim geocoder for our local instance was such a change, managed through the modification of OTP’s client configuration files so that it can correctly locate and query the local Nominatim instance.

Additional changes to the client required for the implementation of the indoor preference dropdown were also made, with the respective files and highlighted changes available in Appendix II.

6.11 Proof of concept process – Overview, Testing and Results

Below is an overview of the proof of concept workflow and testing process, including screenshots:

As a first step, we use FME and the main “oracle_all2osm” FME Workbench, in order to generate our OSM dataset from the IndoorGML data.



Translation Parameters

User Parameters

Connection: @aftab/aftab1@129.187.38.219:1521/ORCL.gis23

Destination OpenStreetMap (OSM) File: TUM_Thesis\Data\OSM_TestData\test6_ways.osm

Connection: @aftab/aftab1@129.187.38.219:1521/ORCL.gis23

Connection: @aftab/aftab1@129.187.38.219:1521/ORCL.gis23

Source CityGML File(s): eogl\Daten\3DCityDB_Export_20150123.gml"

Destination OpenStreetMap (OSM) File: UM_Thesis\Data\OSM_TestData\test6_nodes.osm

Figure 43 - Use of FME Workbench to generate OSM way and node files from IndoorGML Oracle Spatial dataset

Following that, we retrieve the “outdoor” OSM data via JOSM, and load the two FME generated OSM XML files in JOSM. When selecting an outdoor dataset, the JOSM tool provides feedback on the likelihood of the data being retrieved without issues from the OSM API. It is important to check this information, as direct retrieval from OSM is limited, being able to cover simple larger datasets, but allowing much smaller selections within densely mapped urban areas. In the case that a larger dataset is needed, country or city level extracts are available from alternative sources, as mentioned in section 4.3.2.

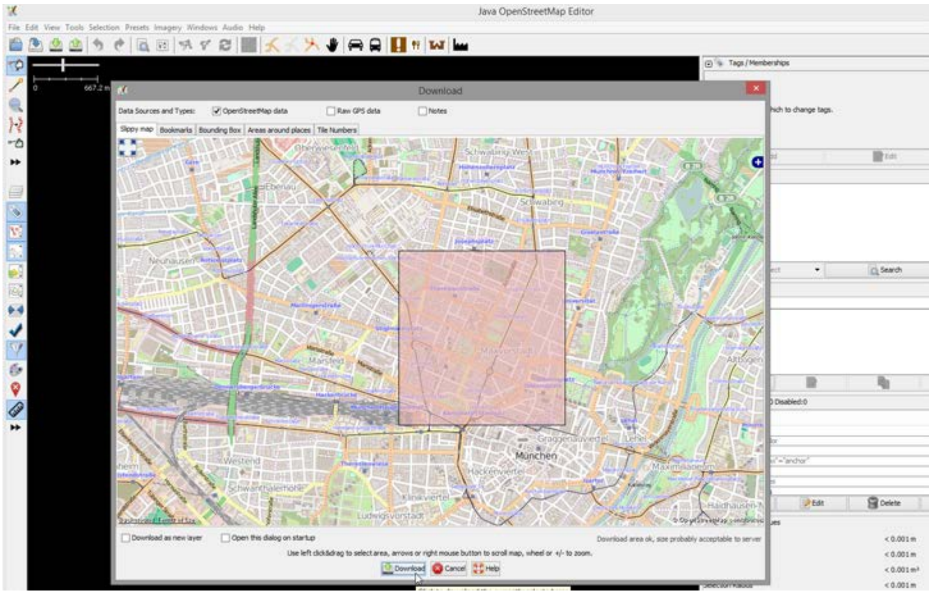


Figure 44 - OSM API "outdoor" OSM data download via JOSM

Once the “outdoor” OSM data is downloaded and saved as an OSM XML file, we can proceed with loading the two OSM files generated via FME from the IndoorGML dataset.

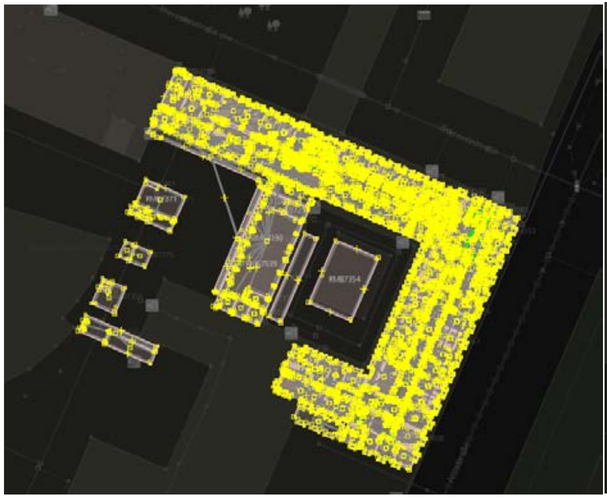


Figure 45 - FME generated Way OSM File

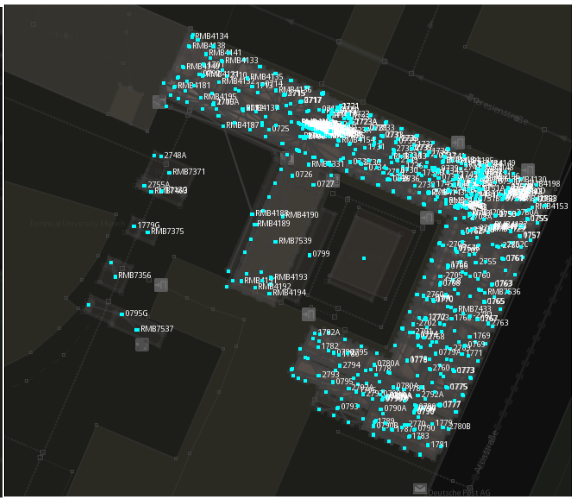


Figure 46 - FME generated Node OSM File

As discussed in sections 5.6 we need to have Anchor Space / Node features from our IndoorGML dataset, in order to be able to match the “outdoor” OSM data to that derived from IndoorGML. However, as mentioned in section 6.7, Anchor Spaces need to be defined manually via editing of feature tags in JOSM. Once the editing is completed and the Ways OSM XML file is saved with the updates as per the instructions on section 6.7, we return to FME Workbench, in order to proceed with the matching of the building outdoor “entrance=*” tags and the Anchor Spaces / Nodes that we defined.

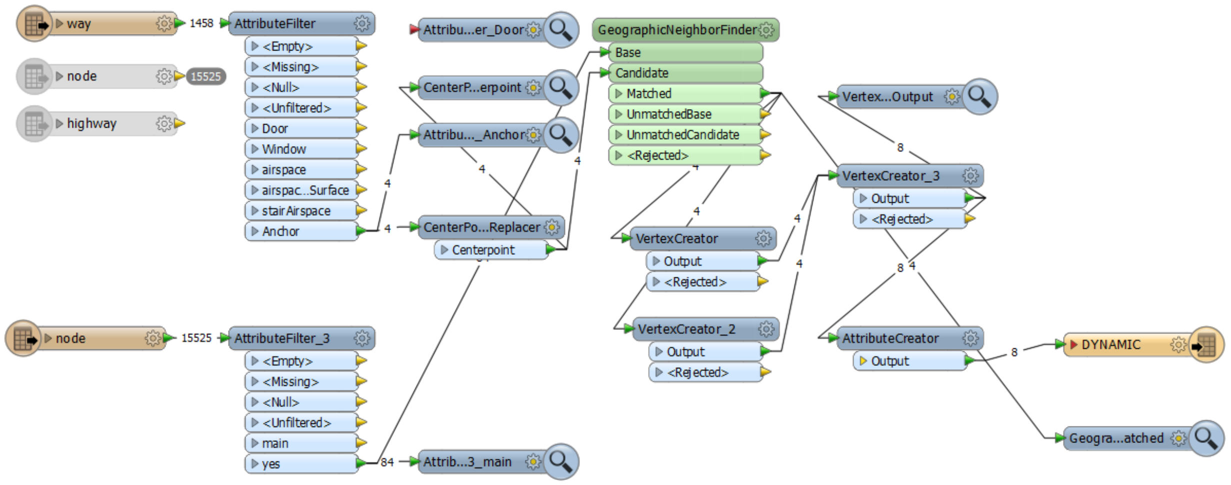


Figure 47 - "Outdoor" OSM "entrance=*" feature to Anchor Space / Node feature matching via FME

In the FME Workbench, we load the updated Way and “Outdoor” OSM files that we have via FME and JOSM respectively, and run the workbench. A new OSM file is generated, which contains the “highway” tagged elements connecting the matched Anchor and “entrance” features.

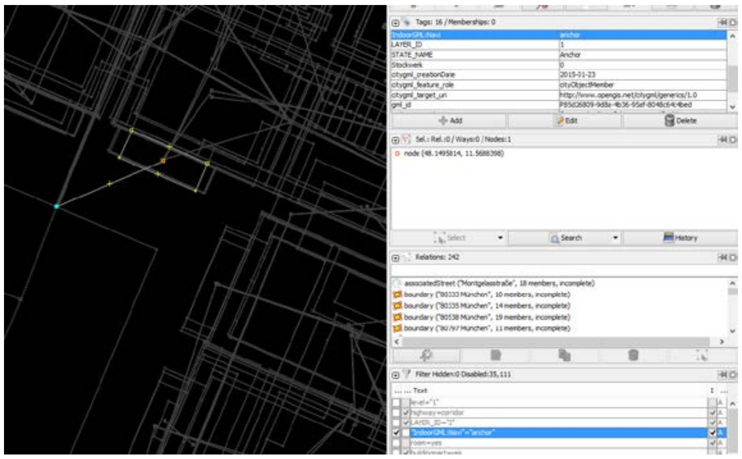


Figure 48 - Anchor Node / Space equivalent OSM feature, Nearest Neighbor “entrance” Outdoor feature and generated Way / Transition

After loading the newly created connections file in JOSM, we proceed with merging the layers in JOSM, first merging the nodes layer to the ways, the combined layer to the “outdoor” dataset and finally merging the anchor to “entrance” connections to the rest of the data.

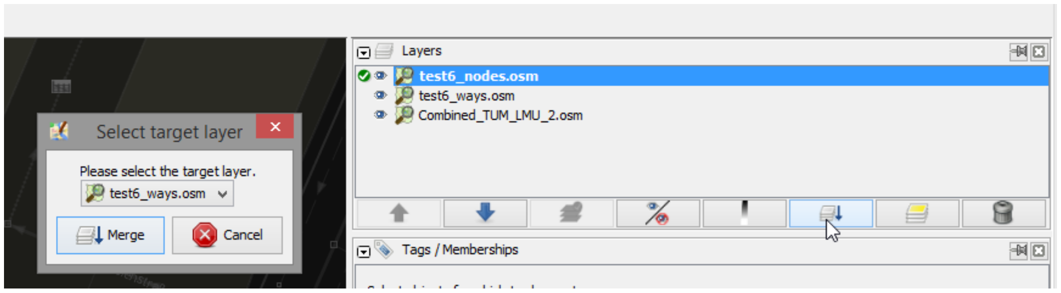
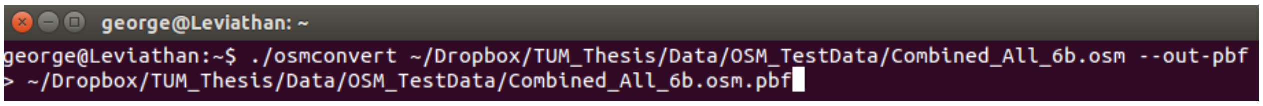


Figure 49 - Merging OSM data layers in JOSM

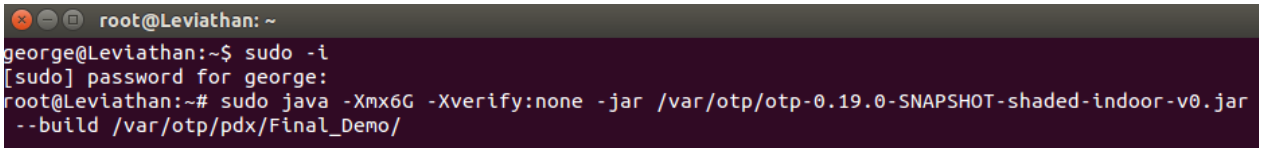
After merging the data to a single file, we need to use the osmconvert command line tool in order to change the data to the OSM PBF file format.



```
george@Leviathan: ~
george@Leviathan:~$ ./osmconvert ~/Dropbox/TUM_Thesis/Data/OSM_TestData/Combined_All_6b.osm --out-pbf
> ~/Dropbox/TUM_Thesis/Data/OSM_TestData/Combined_All_6b.osm.pbf
```

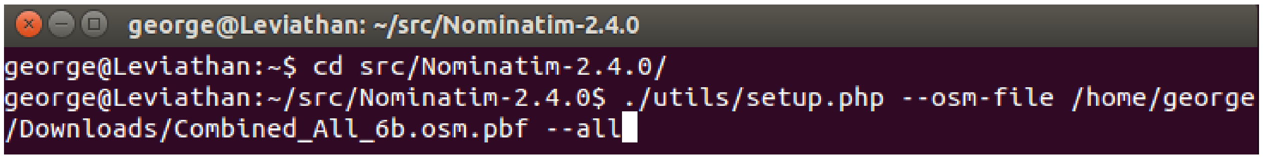
Figure 50 - osmconvert tool usage for OSM XML to OSM PBF file conversion

The generated OSM PBF file can then be used to build the OTP Graph and Nominatim Index, via use of the applications' command line tools.



```
root@Leviathan: ~
george@Leviathan:~$ sudo -i
[sudo] password for george:
root@Leviathan:~# sudo java -Xmx6G -Xverify:none -jar /var/otp/otp-0.19.0-SNAPSHOT-shaded-indoor-v0.jar
 --build /var/otp/pdx/Final_Demo/
```

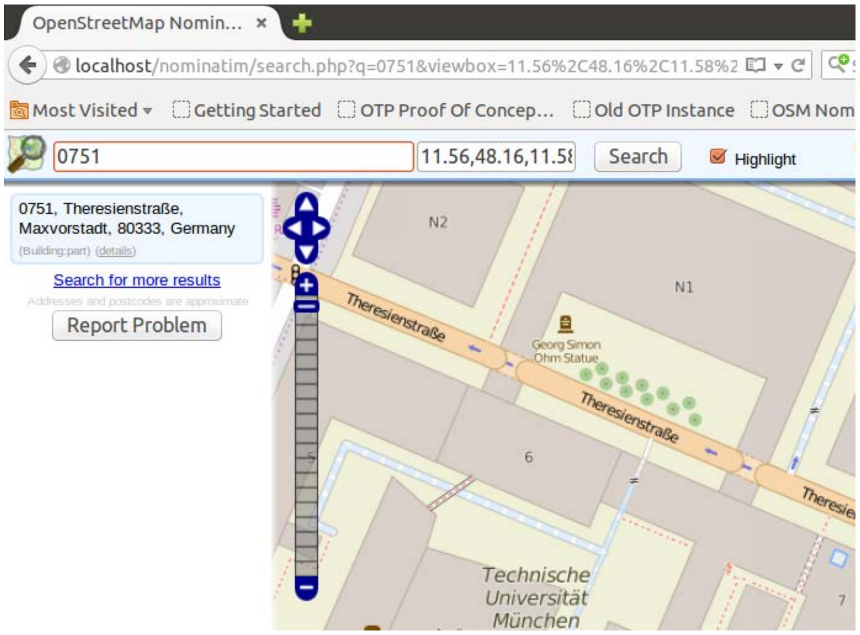
Figure 51 - OTP Graph build command syntax. All necessary files (OSM PBF, GTFS ZIP) are included within the target folder



```
george@Leviathan: ~/src/Nominatim-2.4.0
george@Leviathan:~$ cd src/Nominatim-2.4.0/
george@Leviathan:~/src/Nominatim-2.4.0$ ./utils/setup.php --osm-file /home/george
/Downloads/Combined_All_6b.osm.pbf --all
```

Figure 52 - OSM Nominatim Index setup command syntax, incl. OSM PBF source file definition

After the OTP Graph and Nominatim Index are created, we can begin testing their functionality. For the Nominatim Geocoder, we can simply navigate to http://localhost/nominatim via a browser and do a test search to confirm the geocoder works as intended.

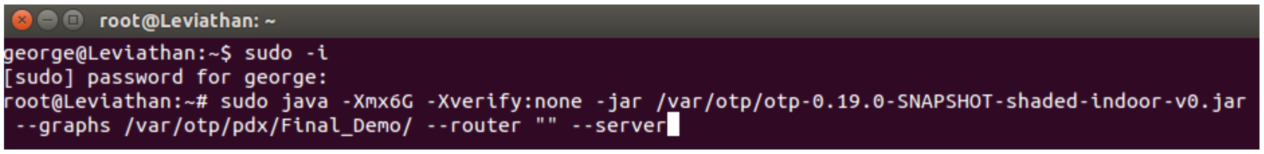


localhost/nominatim/search.php?q=0751&viewbox=11.56%2C48.16%2C11.58%2

0751, Theresienstraße, Maxvorstadt, 80333, Germany
(Building:part) (details)

Figure 53 - Nominatim geocoder instance test

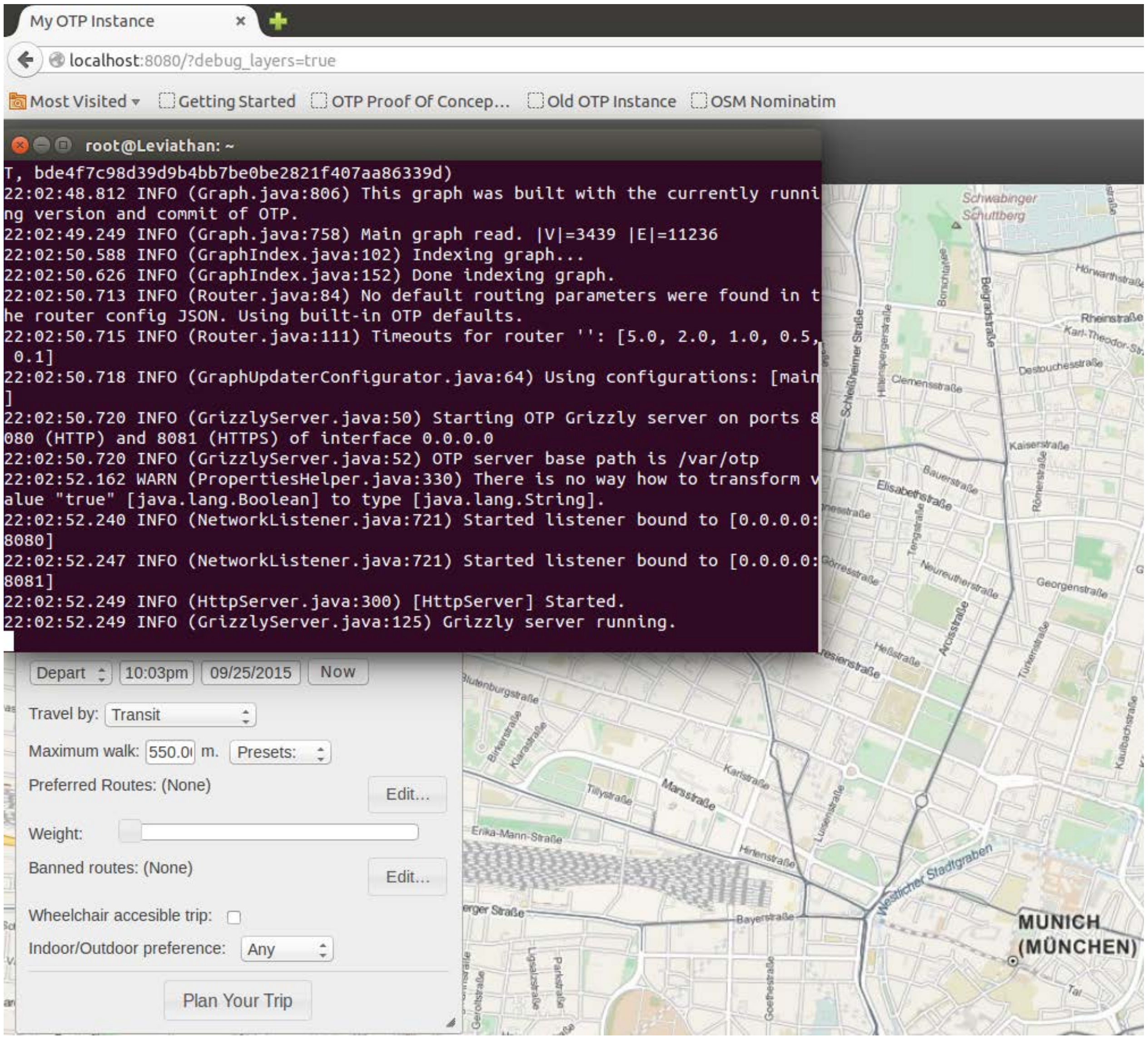
For OTP, we need to manually start the server via command line, as below:



```
george@Leviathan:~$ sudo -i
[sudo] password for george:
root@Leviathan:~# sudo java -Xmx6G -Xverify:none -jar /var/otp/otp-0.19.0-SNAPSHOT-shaded-indoor-v0.jar
 --graphs /var/otp/pdx/Final_Demo/ --router "" --server
```

Figure 54 - OTP Server start command syntax. OTP Graph file is included within the target folder

Once the server has initialized, we can load a tab in our browser with the location of our OTP server (http://localhost:8080/?debug_layers=true). Please note that we are enabling the built-in OTP debug layers via the "?debug_layers=true" parameter for our testing.



```
T, bde4f7c98d39d9b4bb7be0be2821f407aa86339d)
22:02:48.812 INFO (Graph.java:806) This graph was built with the currently runni
ng version and commit of OTP.
22:02:49.249 INFO (Graph.java:758) Main graph read. |V|=3439 |E|=11236
22:02:50.588 INFO (GraphIndex.java:102) Indexing graph...
22:02:50.626 INFO (GraphIndex.java:152) Done indexing graph.
22:02:50.713 INFO (Router.java:84) No default routing parameters were found in t
he router config JSON. Using built-in OTP defaults.
22:02:50.715 INFO (Router.java:111) Timeouts for router '': [5.0, 2.0, 1.0, 0.5,
 0.1]
22:02:50.718 INFO (GraphUpdaterConfigurator.java:64) Using configurations: [main
]
22:02:50.720 INFO (GrizzlyServer.java:50) Starting OTP Grizzly server on ports 8
080 (HTTP) and 8081 (HTTPS) of interface 0.0.0.0
22:02:50.720 INFO (GrizzlyServer.java:52) OTP server base path is /var/otp
22:02:52.162 WARN (PropertiesHelper.java:330) There is no way how to transform v
alue "true" [java.lang.Boolean] to type [java.lang.String].
22:02:52.240 INFO (NetworkListener.java:721) Started listener bound to [0.0.0.0:
8080]
22:02:52.247 INFO (NetworkListener.java:721) Started listener bound to [0.0.0.0:
8081]
22:02:52.249 INFO (HttpServer.java:300) [HttpServer] Started.
22:02:52.249 INFO (GrizzlyServer.java:125) Grizzly server running.
```

Figure 55 - OTP Server initialized and instance loaded in browser

We can use the debug layers functionality to display the available/loaded OTP graph and confirm that both our building/"indoor" dataset and "outdoor" OSM data is included in the graph.

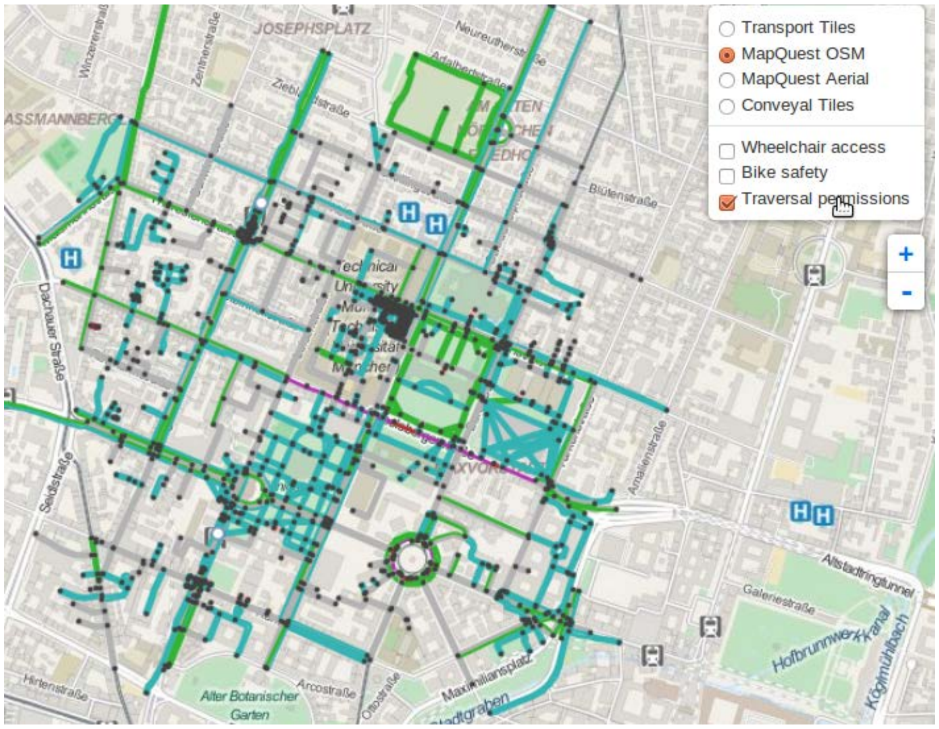


Figure 56 - Display of "Traversal permissions" debug layer to review complete OTP Graph

The difference in graph complexity between typical “outdoor” OSM data and an “indoor” IndoorGML derived dataset becomes rather pronounced when zooming to the relevant section of the graph. For a quick comparison, the entire “outdoor” dataset, spanning 88.729 hectares, consists of 2001 ways and 9619 nodes, while the “indoor” dataset contains 4252 ways and 9503 nodes at 1.572 hectares. These values result in close to 120 times higher way and 55 times higher node density in an indoor dataset, compared to a well mapped area such as the center of Munich.

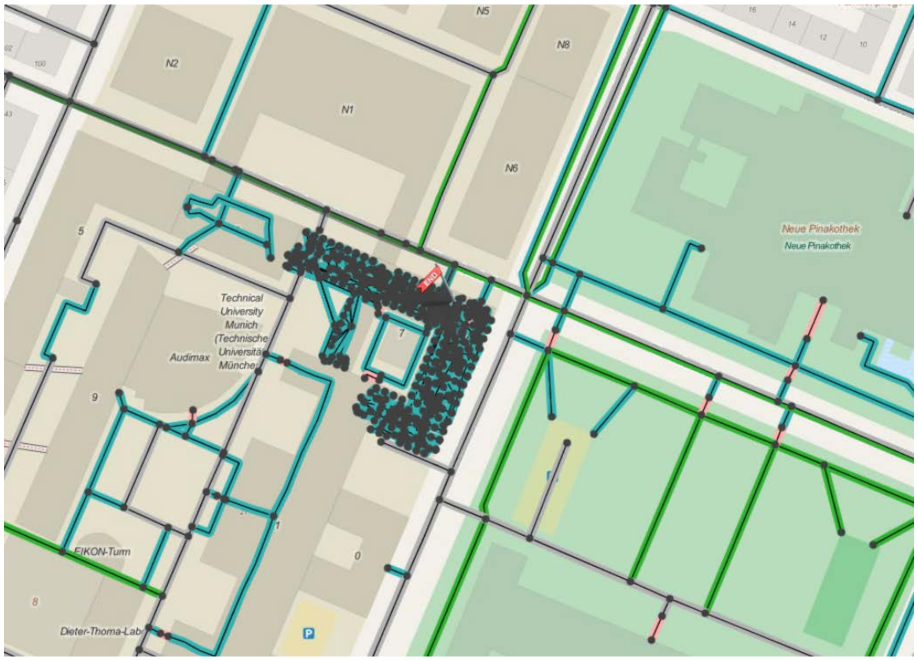
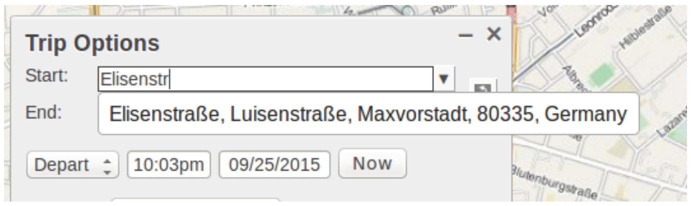


Figure 57 - Review of indoor segment of OTP Graph via display of the "Traversal permissions" debug layer

We can now start by entering the address of a start point in our “outdoor” OSM data, simultaneously testing the Nominatim Geocoder for the “outdoor” data segment. We can see that the value we are entering is autocompleted from the geocoder if a near match is found.



Trip Options

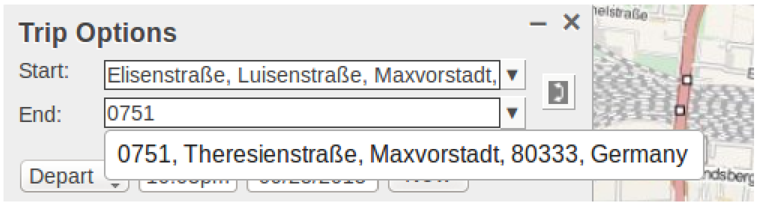
Start: Elisenstr

End: Elisenstraße, Luisenstraße, Maxvorstadt, 80335, Germany

Depart 10:03pm 09/25/2015 Now

Figure 58 - Start point from “outdoor” OSM data

We can then use a room with a known name in the building as the trip end point to confirm that the Nominatim Geocoder is also handling the “indoor” data segment correctly.



Trip Options

Start: Elisenstraße, Luisenstraße, Maxvorstadt,

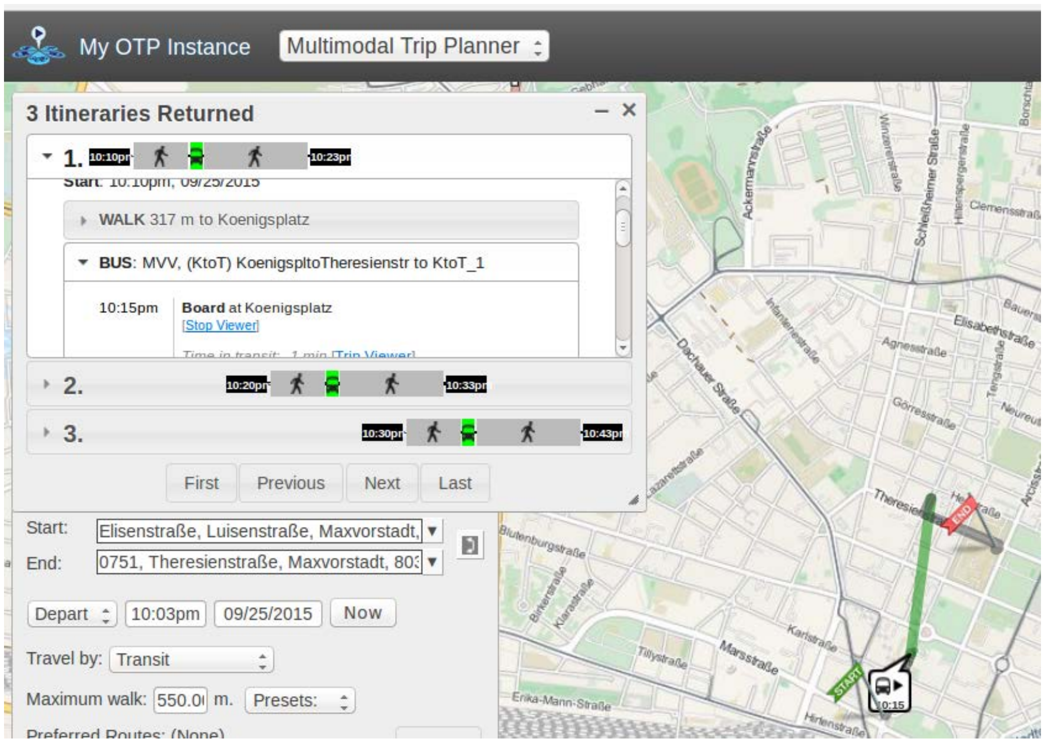
End: 0751

0751, Theresienstraße, Maxvorstadt, 80333, Germany

Depart

Figure 59 - End point from “indoor” OSM data

Once the two points have been defined, we can modify any other parameters such as the maximum walking distance and click the Plan Your Trip button. The OTP routing engine will return results based on the specified input.



My OTP Instance Multimodal Trip Planner

3 Itineraries Returned

1. 10:10pm 10:23pm

Start: 10:10pm, 09/25/2015

WALK 317 m to Koenigsplatz

BUS: MVV, (KtoT) KoenigspltoTheresienstr to KtoT_1

10:15pm Board at Koenigsplatz [Stop Viewer]

Time in transit: 1 min [Trip Viewer]

2. 10:20pm 10:33pm

3. 10:30pm 10:43pm

First Previous Next Last

Start: Elisenstraße, Luisenstraße, Maxvorstadt,

End: 0751, Theresienstraße, Maxvorstadt, 80:

Depart 10:03pm 09/25/2015 Now

Travel by: Transit

Maximum walk: 550.0 m. Presets:

Preferred Routes: (None)

Figure 60 - OTP Route result for transit with outdoor/indoor start and end points

From the above routing result, we can confirm both the general routing functionality of the engine, as well as OTP's multimodal support for use of Transit, based on our mock dataset.

Additionally, if we zoom in to the point of entry to the building dataset and enable the debugging layer, we can see the improvement discussed in section 6.7 regarding the use of Anchor Space / Node features instead of matching "entrance" elements to doors, as OTP routes through one of the defined anchor connections.

However, one other point that becomes apparent is that review and debugging of indoor route network graphs with the default 2D representation of 3D data in OTP is at best cumbersome, if not almost unfeasible.

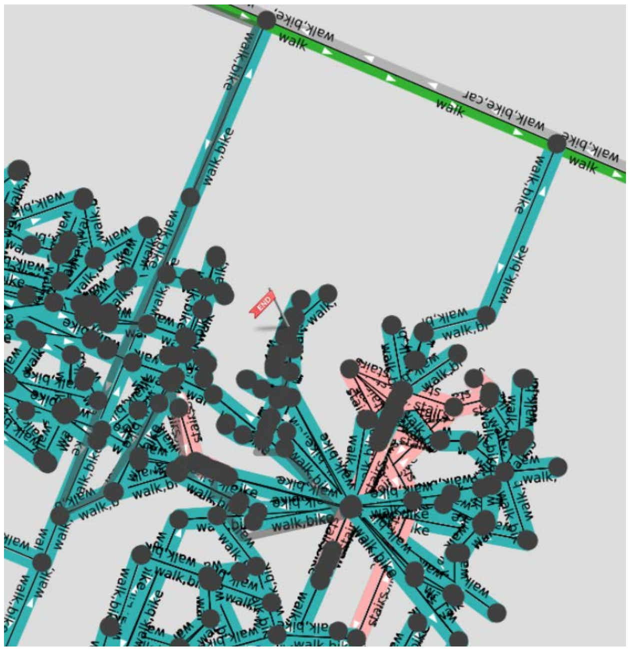
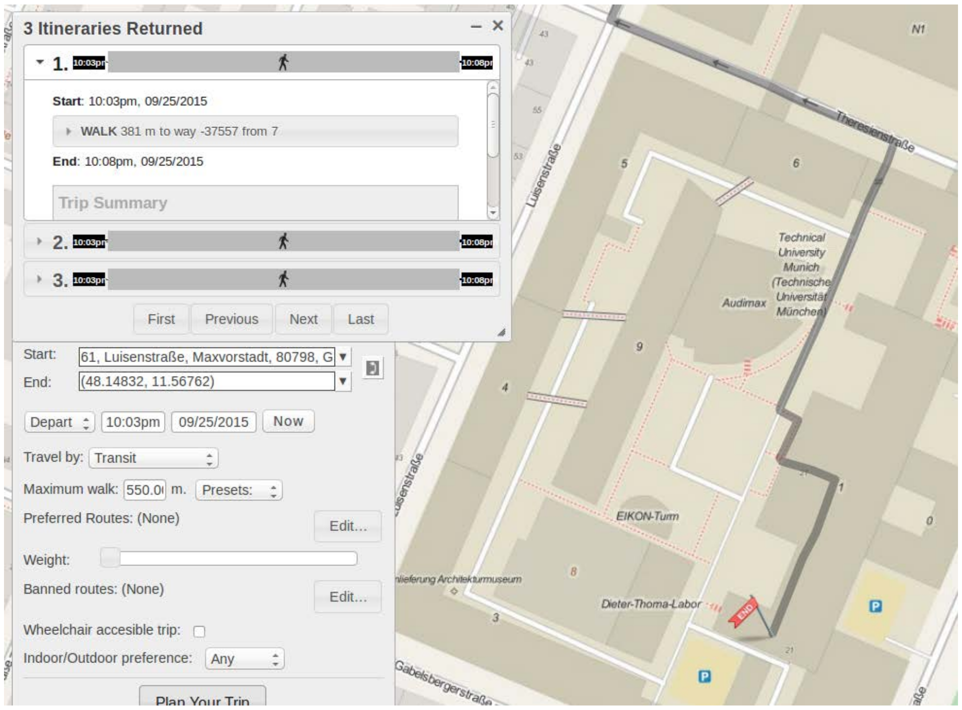


Figure 61 - Route through anchor point connection - Complexity of 2D representation of indoor network graph

For the next step in our testing, we confirm the functionality of simple walking routes. If "Transit" is selected as the mode of transport, we can still receive simple walking-only results, if they are preferred to using a transit route. As we are using nearby points in a small dataset, this happens to be the case. Again, we define an outdoor and an indoor point for our routing.

It should be noted that the indoor point used for the below testing segment was created manually in order to better showcase the added functionality in regards to context awareness. The same functionality could be replicated in an IndoorGML derived OSM dataset created from our workflow, however the current building does not readily provide a good network segment for testing of the added functionality.


3 Itineraries Returned

1. 10:03p — 10:08p

Start: 10:03pm, 09/25/2015

WALK 381 m to way -37557 from 7

End: 10:08pm, 09/25/2015

Trip Summary

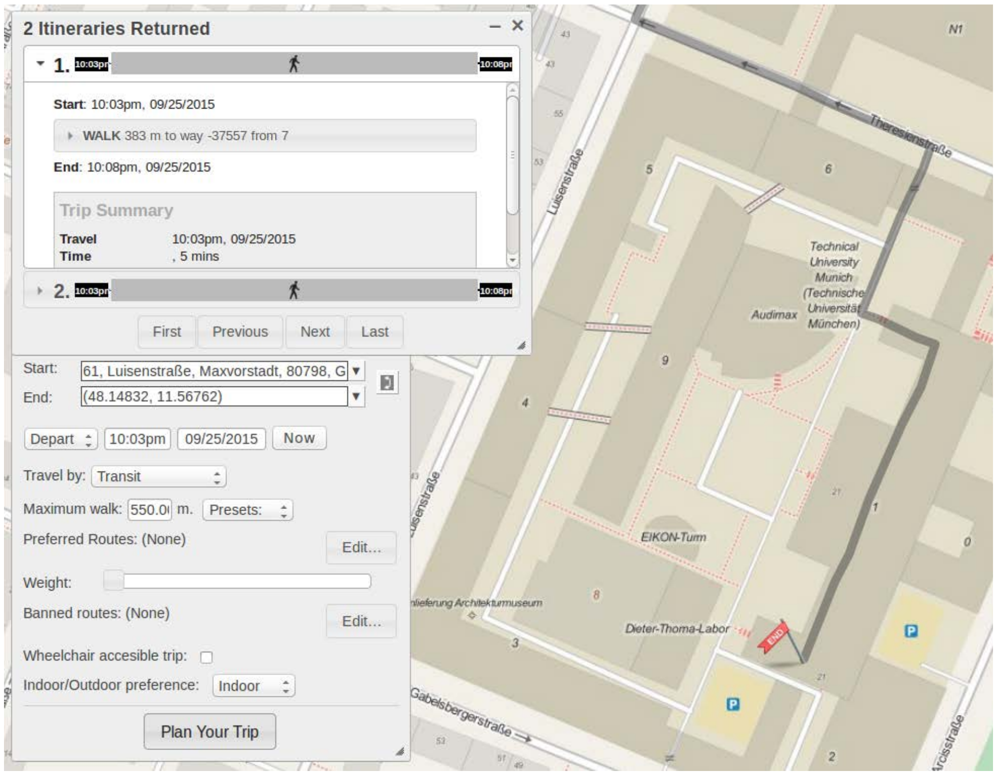
2. 10:03p — 10:08p

3. 10:03p — 10:08p

Start:	61, Luisenstraße, Maxvorstadt, 80798, G
End:	(48.14832, 11.56762)
Depart	10:03pm 09/25/2015 Now
Travel by:	Transit
Maximum walk:	550.0 m. Presets:
Preferred Routes:	(None)
Weight:	
Banned routes:	(None)
Wheelchair accesible trip:	☐
Indoor/Outdoor preference:	Any

Figure 62 - Standard routing result with no Indoor/Outdoor or wheelchair preference

Once we have our standard route, we can test the indoor/outdoor preference functionality, by selecting the “Indoor” option from the trip planner dropdown. After a short calculation, we can see that the route is updated, containing a larger indoor segment.


2 Itineraries Returned

1. 10:03p — 10:08p

Start: 10:03pm, 09/25/2015

WALK 383 m to way -37557 from 7

End: 10:08pm, 09/25/2015

Trip Summary

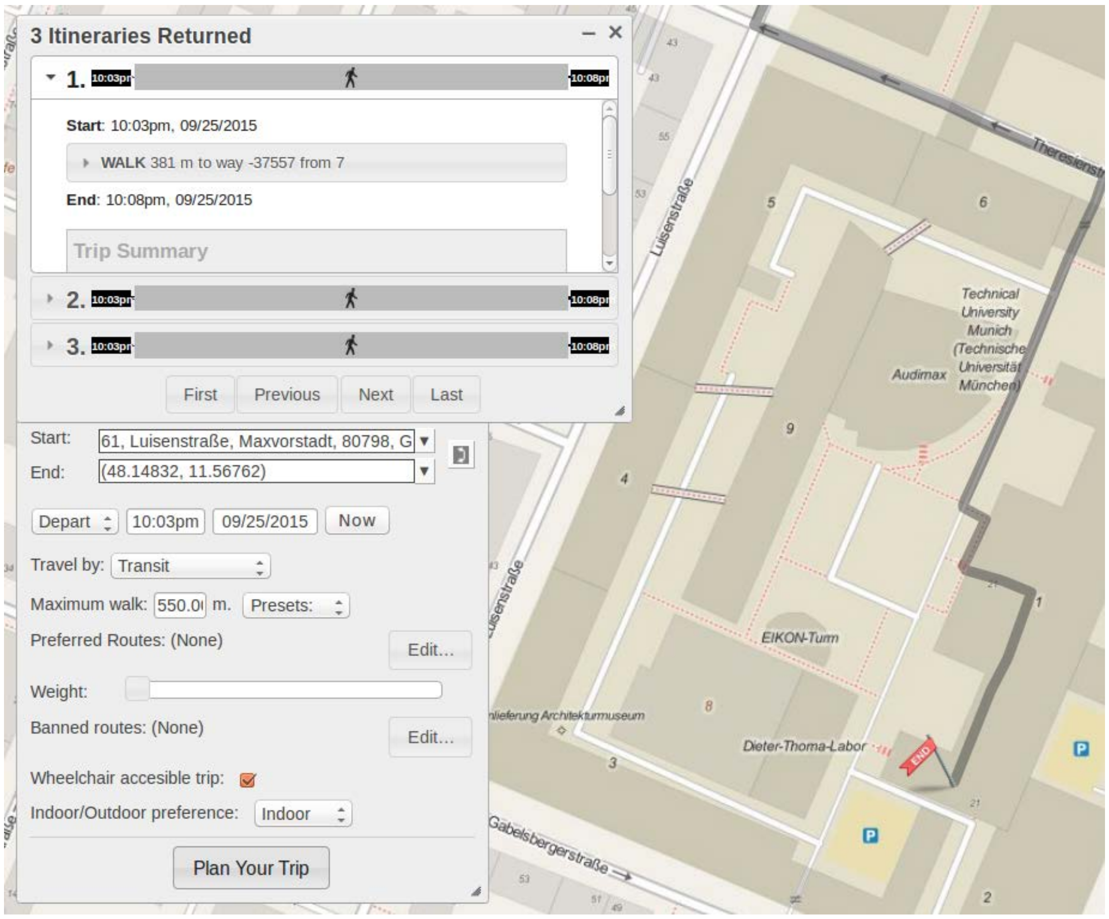
Travel Time	10:03pm, 09/25/2015 , 5 mins

2. 10:03p — 10:08p

Start:	61, Luisenstraße, Maxvorstadt, 80798, G
End:	(48.14832, 11.56762)
Depart	10:03pm 09/25/2015 Now
Travel by:	Transit
Maximum walk:	550.0 m. Presets:
Preferred Routes:	(None)
Weight:	
Banned routes:	(None)
Wheelchair accesible trip:	☐
Indoor/Outdoor preference:	Indoor

Figure 63 - Updated routing result with Indoor preference and no wheelchair preference

In order to test wheelchair accessibility, we check the “Wheelchair accessible trip” option. We can see that the route is updated again, directing us through a path and entrance that are wheelchair accessible. It’s also important to note that wheelchair accessibility takes precedence over indoor/outdoor preference, as intended.


3 Itineraries Returned

1. 10:03pm 10:08pm

Start: 10:03pm, 09/25/2015

WALK 381 m to way -37557 from 7

End: 10:08pm, 09/25/2015

Trip Summary

2. 10:03pm 10:08pm

3. 10:03pm 10:08pm

First Previous Next Last

Start: 61, Luisenstraße, Maxvorstadt, 80798, G

End: (48.14832, 11.56762)

Depart 10:03pm 09/25/2015 Now

Travel by: Transit

Maximum walk: 550.0 m. Presets:

Preferred Routes: (None) Edit...

Weight:

Banned routes: (None) Edit...

Wheelchair accesible trip: ✓

Indoor/Outdoor preference: Indoor

Plan Your Trip

Figure 64 - Updated routing result with Indoor preference and wheelchair preference, wheelchair constraint precedence

As we have seen from the above overview, the proof of concept implementation was tested with indoor only, outdoor only and combined indoor outdoor requests. The standard routing functionality was confirmed, including transit + walking and walking only requests, meeting our criteria for multi-modal routing.

In regards to context awareness, time constraints were only usable in regards to OTP’s built in time dependent routing for Transit trips. Time-based constraints regarding the indoor dataset were examined, however, there were significant difficulties in implementing them, a fact confirmed with feedback from the OTP development team. The same limitations applied for the considered option of clearance-based access constraints. The main reason is the simultaneous definition of multiple constraints in OSM via use of the “access” tag, which created problems with the correct parsing of the various constraint possibilities.

However, both wheelchair accessibility and indoor/outdoor preference were successfully implemented as constraints, providing alternative routes depending on the user input during the routing request.

Some issues that were initially identified regarding the connectivity of the datasets due to the lack of Anchor states in the source dataset, were overcome via the manual definition of Anchor Space / Node features to be used for matching instead of the previous Door states. This addition, along with slight modifications to the relevant workbench resolved previous issues. One such example was the outdoor dataset being connected to a door belonging to a room near the actual building entrance. Such issues can be easily overcome via the accurate definition of Anchor spaces and states, and a, so that Anchor spaces & states are used instead of doors.

During testing, we had a chance to review the performance of a 2D representation while dealing with a 3D dataset such as a building. While 2D is generally adequate for outdoor routing/navigation, its use for indoor navigation proves to be much more problematic. The limitations of a 2D representation can be overcome to a certain extent through the use of multiple levels/layers, however, this is still not optimal for more complex or extensive structures which are better visualized through the use of a 3D model.

One of the platforms increasingly utilized for 3D representation of data is Cesium, an open-source Javascript library utilizing WebGL for the creation and display of 3D globes as well as 2D maps in mobile and desktop browsers without the need for additional plugins.

With built-in support for OpenStreetMap base imagery layers, JSON & GPX vector data (e.g., a route produced in JSON from the OTP routing engine) and COLLADA 3D models, Cesium (http://cesiumjs.org/) offers an interesting potential option for the 3D visualization of a navigation implementation.

As some research has been performed from the chair regarding the combination of semantic spatial databases (3dCityDB) and a Cesium based viewer (**Chaturvedi, K., Yao, Z., & Kolbe, T. H., 2015)**, Cesium could be considered as an option for 3D representation of a route. An existing prototype for 3D visualization of IndoorGML based indoor routing has already been developed from the TUM chair of Geoinformatics **(Khan, A. A., Yao, Z., & Kolbe, T. H., 2015)** and could potentially offer a starting point for combined indoor/ outdoor routing implementations.

Other alternatives naturally exist, with a significant amount of research into the 3D representation of data coming directly from the OSM community, as depicted in section 4.3.7. In regards to Indoor 3D representation of OSM data, a prototype application using XML3D and building data defined as per the IndoorOSM proposal was developed from the University of Heidelberg and is available online (http://indoorosm.uni-hd.de/3d/Indoor_examine.xhtml#). Another prototype implementation, based on the OSMBuildings project is available online, however again with quite limited functionality (http://osmbuildings.org/examples/indoor/).

Challenges to be considered with the 3D representation are the performance impact & requirements for the representation of complex building indoor models, in combination with the vector representation of a route and the 3D “outdoor” dataset. Naturally, new development of a 3D capable and indoor optimized routing engine (or extensive modification of an existing solution) would be necessary in order provide more than simple visualization of the data.

Another issue that was encountered during testing, which may be related to the small extent of the outdoor dataset and the use of a mock GTFS dataset with only two stops in close proximity to the target “indoor” area is that the preferred route output often provides walking only instructions despite selection of the “Transit” mode of transport. This may be due to the fact that OTP adds an additional weight cost to the route for boarding and leaving transit, which, in shorter routes such as those possible within our combined outdoor/indoor dataset, leads to a preference towards walking instead of utilizing available transport.

One final and rather important challenge that was faced during testing of the implementation was the inability to use the Nominatim geocoder to provide OSM IDs of indoor elements to the routing engine as trip start/end points. In the current implementation, the X,Y coordinates of the element or its center point in the case of areas is provided, and the OTP routing engine matches it with vertices sharing the same X,Y coordinates. This does not appear to take into account the level tagging, only navigating to a different level if the matched vertex has no other vertices in different levels that overlap its X,Y coordinates. As a result, requests towards rooms from the geocoder that are in different levels essentially provide the same input to the routing engine.

While the geocoder request and response can be reconfigured to provide only the OSM ID as its response, this cannot be utilized by OTP, as it appears to use an internal ID within its Graph which is different to that available from the OSM dataset and Nominatim. As the geocoder has no way of parsing the internal Graph IDs of elements based on their OSM ID, it is obvious that either extensive reworking of OTP and Nominatim is required, or an alternative solution in regards to routing or trip start/end point selection. One potential solution that could at least offer a possible improvement in indoor trip star/end point selection would be to implement a selection method using a layer/level based representation of building indoors, as per the OSMTools or OpenLevelUp projects, referenced in section 4.3.7.

6.12 Proof of Concept Implementation - Conclusion

Overall, the performance of the proof of concept solution exceeded our initial expectations in some aspects, considering the lack of built-in support for indoor datasets, while leaving room for improvement in other aspects.

Regarding the migration of data from IndoorGML to OSM, it should be noted that, at its current state, the generated OSM data is still simplified compared to the initial IndoorGML data, mostly due to the limitations of OSM's simple data model compared to IndoorGML. However, the conflation process provides a navigable combined dataset, including the identification of outdoor entrance & indoor anchor matches and generation of connecting pathways, when both the OSM and IndoorGML datasets are well defined. This provides datasets that can be readily utilized for indoor/outdoor routing requests, with confirmed functionality in single level indoor datasets.

It should be noted that the OSM data generated for the proof of concept implementation was mostly geared towards having a dataset working with the limitations that OTP and Nominatim introduce, rather than being as close to the original data as possible. With that in mind, there is still room for improvement in this regard, which should also be easier to implement with the complete conceptual conflation model defined and a dataset based on the IndoorGML OGC standard and fully utilizing all available IndoorGML feature classes.

Regarding the performance of the routing solution used for the proof of concept, we were able to obtain working routing results for transit and walking. Naturally, a more extensive mock GTFS dataset or an actual GTFS dataset from a transit agency would provide much better results. The current proximity of the mock GTFS stops and the tested indoor area often result in walking routes being preferred, due to the routing engine adding a boarding and disembarking weight cost to transit usage for small distances. Generally, improvements in this regard depend greatly on the accuracy of the external OSM data, which, if supplemented with parking spots for bikes and cars, could also be used to power multimodal planning for other options like car and bike, with a few further modifications to the OTP client.

Context awareness and application of constraints was achieved in regards to wheelchair accessibility and indoor/outdoor preference, with slight modifications to the OTP routing engine source code and appropriate tagging of the generated OSM features. Again, especially in regards to wheelchair accessibility, the accuracy of the routing results depends greatly on the accuracy and completeness of both the "outdoor" OSM data and the IndoorGML dataset from which we derive our "indoor" OSM.

One significant problem that remains is the correct provision of trip start/end points to the routing engine. As OSM and its tools, and more specifically the OTP routing engine, are aimed towards outdoor navigation, accurate selection of trip points in a dataset of features with overlapping X,Y coordinates but different Z values is problematic. An attempt to resolve this issue was made through the use of the geocoder, which can identify the target points correctly, but cannot communicate the information to the routing engine in a manner that would force routing through the appropriate path to the level the element belongs to. This causes confirmed issues when attempting to route to overlapping features (X,Y overlapping rooms in different floors), where the route generated simply directs to the feature that is at the same level as the outdoor dataset and building entrance. When directing to features with non-overlapping X,Y network elements (nodes or ways), the implementation acknowledged the need to route through an elevator feature and change levels in order to reach the target room (this functionality was tested with manual additions to the combined indoor/outdoor OSM dataset).

This could be potentially resolved via the use of a different routing solution, one with better management of overlapping features, or one not relying on a graph prebuilt from the OSM data, but real time graph building and direct use of the OSM data. This would allow the use of a geocoder as initially planned, due to the geocoder also utilizing the same OSM data and being able to offer node IDs that would refer to the same features in both the routing network graph and the geocoder OSM feature index.

One final issue is the representation of 3D based data in a 2D oriented environment. The problem becomes increasingly apparent when trying to review the network graph of the “indoor” OSM data in the OTP client, but is already obvious when loading the generated data in an OSM viewer/editor such as JOSM, where efficient navigation and work within the dataset is only achievable through extensive use of filters. In that regard, there is work within the OSM community for development of tools and a tagging schema that could be more oriented towards work with 3D building datasets, however, there is little in the way of a production-ready solution or implementation at the moment.

7 Conclusion

As a conclusion to this thesis, we will provide an overview of the goals achieved through the related research and work of this thesis, while indicating areas of potential for further research and providing some overall final feedback.

As part of the goals, after extensive review of both the IndoorGML and OSM data models, their structure, content and capabilities, a proposal for a Conceptual Conflation Model for migrating data from IndoorGML to OSM has been created. This provided a basis for some aspects of the proof of concept implementation, while also giving a good basis for the general transformation of IndoorGML data, including similar future implementations and some perspective on how to best deal with indoor data within OSM.

Additionally, a proof of concept implementation was developed, covering the thesis requirements for creation of a navigable OSM dataset out of IndoorGML source data, via a conversion process based to a certain extent to our conceptual conflation model, but modified to better work with both the specific structure of our sample IndoorGML dataset, and the base requirements of our OSM tools and thesis. An automated process was also defined via FME for the connection of available "outdoor" OSM datasets and the "indoor" OSM data that was generated from IndoorGML.

The tools selected for the proof of concept implementation provided a good starting point for meeting the thesis requirements for a working combined routing solution, but were modified to meet further goals, in respect to Context Awareness, by the addition of an Indoor / Outdoor routing preference option and validation of support for wheelchair accessibility. Additional modifications were made to provide a geocoder based option for entering trip start/end points, including indoor POIs such as rooms, via simple name entry.

Additionally, support of multimodal trip planning was confirmed in testing, via the use of a mock GTFS Transit feed that was created for the implementation. The transit data, combined with the OTP routing engine's built in support for multimodal trip planning via GTFS, allowed us to meet the goal of multimodal trip planning, despite occasional mode of transport preference issues due to the small extent of the GTFS dataset and its proximity to the singular indoor target area.

While the work done in this thesis offers an initial approach towards developing a full-fledged combined indoor/outdoor routing solution, there is still significant potential for further research and development.

One of the areas of future research would be the refinement of the semantics migration process, both in regards to further definition of the IndoorGML Constraints concept, as well as the identification of matching OSM tags and best tagging practices for the accurate and effective conveyance of these attributes to OSM.

This would in turn allow for much more fine-tuned context awareness, provided that the OSM tools (routing engines), include and extend their support for more complex constraint awareness and definition for route planning, both in regards to indoor only and especially towards combined indoor/outdoor data.

However, what is possibly the most important point for better indoor and combined indoor/outdoor navigation is the need for significant improvements in the available options for representing and working with complex, 3D based datasets such as buildings and their indoor areas, or at the very least, complex network graphs which also include vertical transitions. This would also allow for better routing clients, with options for trip start/end point selection in 3D space that will greatly improve both the user experience, as well as the quality of results, by enabling drastically better testing and validation of the routing engines' behavior and results.

While on the point of validation, it should be noted that there already exists a plethora of proposals from the OSM community regarding indoor tagging and representation of buildings within OSM. While the freedom of the OSM platform allows for direct import of the data derived via our implementation and/or based on the Conceptual conflation OSM model, it is still advised to allow for review of both the created data as well as the model from the OSM community, before proceeding with large scale building indoor uploads or en masse modifications to already tagged buildings.

This has been a proven method to identify any potential issues with a proposed mapping and tagging schema, particularly in regards of technical or interoperability issues that may arise, which are spotted much more effectively when reviewed by the active OSM community, compared to what has been already identified during the concept development.

In regards to review of existing data models, some points also became apparent during work with v1 of the IndoorGML OGC standard. One initial point is the need for definition of generic attributes for the IndoorGML elements, either via inclusion of support for generic attributes in the Core Module (as per the MLSEM approach from **(Nagel, C. 2014)**), or via a new thematic extension.

This appears to be the intent with the Constraints Concept, however, with the Constraints Module being more tied to the Navigation rather than the Core module, it is important to avoid further nesting / dependence of thematic modules on each other, as this approach seems to add unnecessary complexity to the model structure. In any case, the Constraints concept needs to be fully implemented as a module for effective context aware navigation.

Additionally, best practices have to be established regarding the representation of specific cases within the model. As an example, wheelchair constraints can either be implemented via use of the Navigation & Constraints Modules, or via definition of a separate Space Layer in the core module. A best practice for such cases needs to be defined, in order to have a consistent modeling approach that will help with the development of tools that can successfully parse an IndoorGML dataset

In conclusion, based on the current state of tools, it is clear that we are still in the early development stages of indoor and combined indoor/outdoor navigation solutions, particularly when comparing to the maturity of outdoor LBS and navigation. However, judging from the increasing amount of interest shown from companies, academia and even user communities, and considering the many applications for such solutions (both commercial and noncommercial), one could expect a growth in the speed of new solution development, similar to that of outdoor LBS in recent years.

8 References

West, M. (2011). Developing high quality data models. Elsevier.

Steve Hoberman, "Data Modeling for MongoDB", Technics Publications, LLC 2014

Khan, A. A., Donaubauer, A., & Kolbe, T. H. (2014). “A multi-step transformation process for automatically generating indoor routing graphs from existing semantic 3D building models.” In Proceedings of the 9th 3D GeoInfo Conference.

Open Geospatial Consortium. (2008). OGC IndoorGML. OGC standard available from: Open Geospatial Consortium – IndoorGML Standard, last retrieved May 2015 http://www.opengeospatial.org/standards/indoorgml

Afyouni, I., et al. (2014). "Spatial models for context-aware indoor navigation systems: A survey." Journal of Spatial Information Science(4): 85-123.

Nagel, C. (2014), “Spatio-Semantic Modelling of Indoor Environments for Indoor Navigation.” Dissertation an der Technischen Universität Berlin, Fakultät VI – Planen Bauen Umwelt

Ramm, F. ; Topf, J.: "OpenStreetMap: Die freie Weltkarte nutzen und mitgestalten", Lehmanns Media, 2010

Goetz, Marcus, and Alexander Zipf. "Extending OpenStreetMap to indoor environments: bringing volunteered geographic information to the next level.” *Urban and Regional Data Management: Udms Annual 2011* (2011).

Hubel A (2011) “Webbrowser based indoor navigation for mobile devices based on OpenStreet-Map.” http://andreas-hubel.de/ba/ba_V2.0.pdf (Accessed June 2015) http://andreas-hubel.de/ba/demo/

Hashemi, P. and R. Ali Abbaspour (2015). Assessment of Logical Consistency in OpenStreetMap Based on the Spatial Similarity Concept. OpenStreetMap in GIScience. J. Jokar Arsanjani, A. Zipf, P. Mooney and M. Helbich, Springer International Publishing: 19-36.

Khan, A. A., & Kolbe, T. H. (2012, November). “Constraints and their role in subspacing for the locomotion types in indoor navigation.” In Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on (pp. 1-12). IEEE.

Donaubauer, A., Straub, F., Panchaud, N., & Vessaz, C. (2013), “A 3D indoor routing service with 2d visualization based on the multi-layered space-event model.” In Progress in Location-Based Services (pp. 453-469). Springer Berlin Heidelberg.

S. Bauer, T. Vogl, (2014) “Automatische Ableitung von 3D Points of Interest aus CityGML LOD4”, Abschlussbericht Angewandte Geoinformatik 2

Chaturvedi, K., Yao, Z., & Kolbe, T. H. (2015). “Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL.”

Khan, A. A., Yao, Z., & Kolbe, T. H. (2015). “Context Aware Indoor Route Planning Using Semantic 3D Building Models with Cloud Computing.” In 3D Geoinformation Science (pp. 175-192). Springer International Publishing.

OpenStreetMap Wiki (Accessed May 2015)
http://wiki.openstreetmap.org/wiki/Main_Page

OSM Taginfo project homepage
https://taginfo.openstreetmap.org/

W3C XML reference site
http://www.w3.org/XML/

Geofabrik OSM data extract for Bavaria (Retrieved June 2015)
http://download.geofabrik.de/europe/germany.html

Mapzen OSM data extract for Munich (Retrieved June 2015)
https://mapzen.com/metro-extracts/

OSMBuildings project homepage
http://osmbuildings.org/

OpenTripPlanner (Accessed May 2015)
http://www.opentripplanner.org/

OpenTripPlanner Deployments Worldwide
http://docs.opentripplanner.org/en/latest/Deployments/

TRIMET Trip Planner (Accessed May 2015)
http://trimet.org/

OpenLevelUp! project homepage
http://github.pavie.info/openlevelup/

OSMTools - Indoor building browser example (Accessed May 2015)
http://clement-lagrange.github.io/osmtools-indoor/home.html

Keep Right OSM Quality Assurance tool (Homepage and OSM Wiki Page)
http://keepright.at/ & http://wiki.openstreetmap.org/wiki/Keep_Right

OmniClass Classification Codelists homepage
http://www.omniclass.org/

OpenTripPlanner Github page
https://github.com/opentripplanner/OpenTripPlanner

OSM Nominatim Github page
https://github.com/twain47/Nominatim

Oracle Spatial and Graph product site
http://www.oracle.com/us/products/database/options/spatial/overview/index.html

Oracle Instant Client downloads page
http://www.oracle.com/technetwork/database/features/instant-client/index-097480.html

Oracle SQL Developer and SQL Developer Data Modeler downloads page
http://www.oracle.com/technetwork/developer-tools/index.html

Safe Software - Feature Manipulation Engine (FME) Homepage
http://www.safe.com/fme/

Google Developers GTFS Reference page
https://developers.google.com/transit/gtfs/

Open Data Berlin – VBB GTFS Dataset source
http://daten.berlin.de/kategorie/verkehr

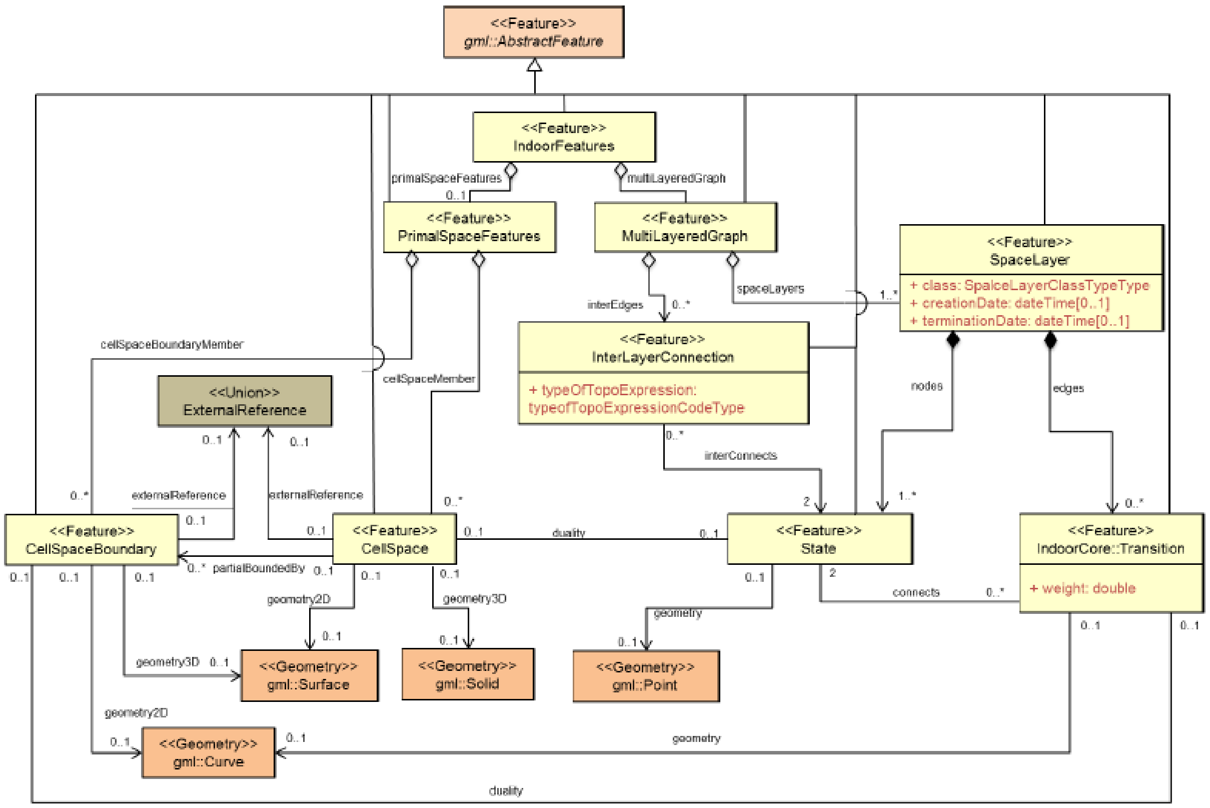
Cesiumjs – Cesium Javascript library website (Accessed May 2015)
http://cesiumjs.org/

University of Heidelberg – IndoorOSM 3D building indoor representation and routing
http://indoorosm.uni-hd.de/3d/Indoor_examine.xhtml#

OSMSharp Github Wiki – OSM Data Model (Accessed May 2015)
https://github.com/OsmSharp/OsmSharp/wiki/OpenStreetMap-data-model

Appendix I – UML & Structure Diagrams, Sample Data Examples

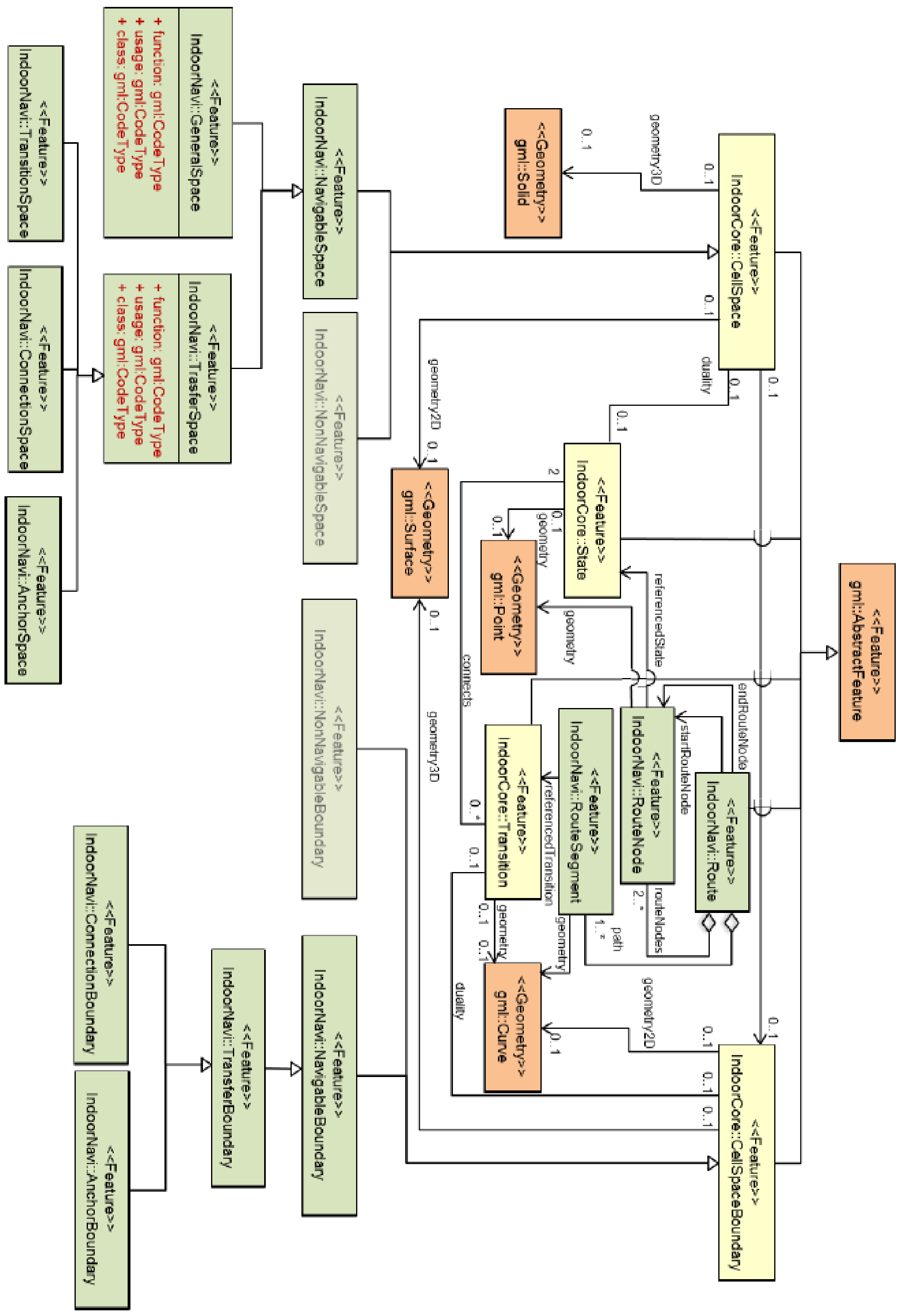
IndoorGML Core Module



<<Feature>> gml::AbstractFeature
<<Feature>> IndoorFeatures
primalSpaceFeatures
0..1
multiLayeredGraph
<<Feature>> PrimalSpaceFeatures
<<Feature>> MultiLayeredGraph
<<Feature>> SpaceLayer
+ class: SpalceLayerClassTypeType
+ creationDate: dateTime[0..1]
+ terminationDate: dateTime[0..1]
interEdges
0..*
spaceLayers
1..*
<<Feature>> InterLayerConnection
+ typeOfTopoExpression: typeofTopoExpressionCodeType
cellSpaceBoundaryMember
cellSpaceMember
<<Union>> ExternalReference
nodes
edges
interConnects
externalReference
<<Feature>> CellSpaceBoundary
<<Feature>> CellSpace
duality
<<Feature>> State
<<Feature>> IndoorCore::Transition
+ weight: double
partialBoundedBy
geometry2D
geometry3D
geometry
connects
<<Geometry>> gml::Surface
<<Geometry>> gml::Solid
<<Geometry>> gml::Point
<<Geometry>> gml::Curve
duality

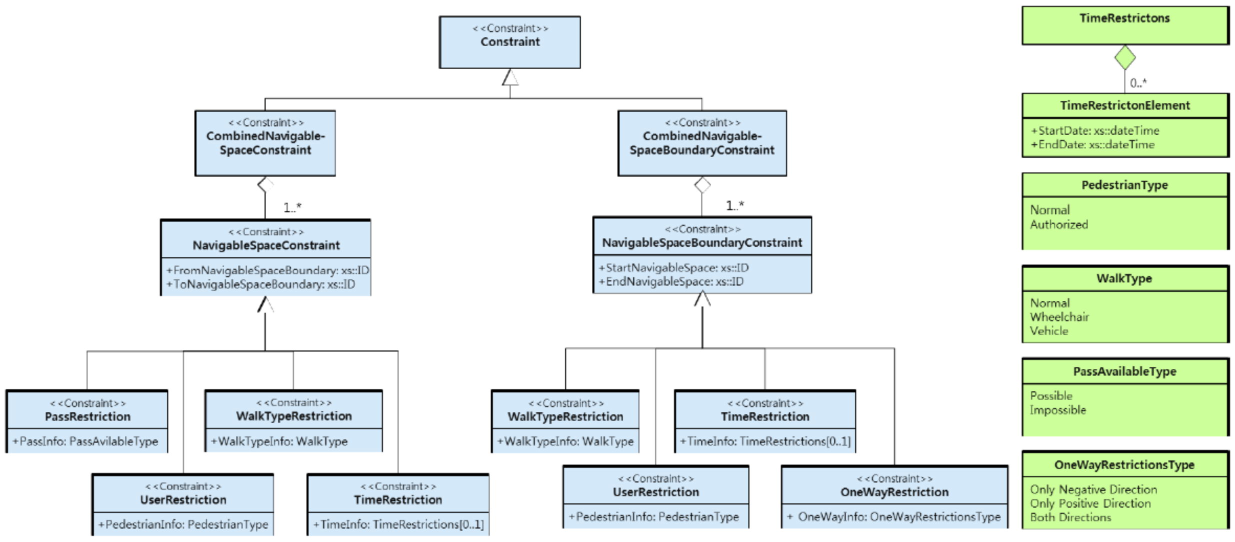
Appendix Figure I - UML diagram of IndoorGML Core Module (OGC, 2014)

IndoorGML Navigation Module



Appendix Figure II - UML diagram of IndoorGML Navigation module (OGC, 2014)

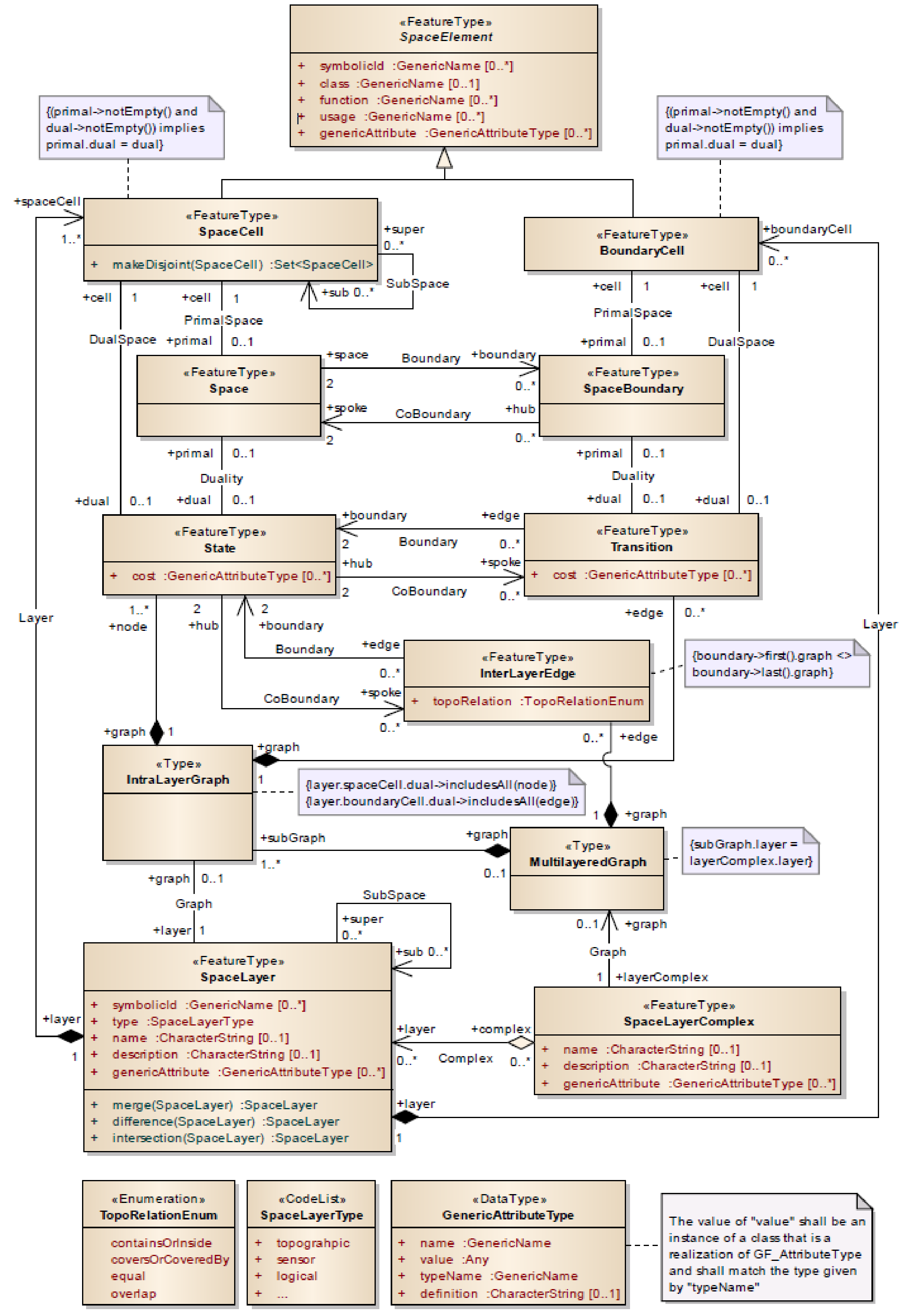
IndoorGML Constraints Concept



<<Constraint>> Constraint
<<Constraint>> CombinedNavigable-SpaceConstraint
<<Constraint>> CombinedNavigable-SpaceBoundaryConstraint
1..*
<<Constraint>> NavigableSpaceConstraint
+FromNavigableSpaceBoundary: xs::ID
+ToNavigableSpaceBoundary: xs::ID
1..*
<<Constraint>> NavigableSpaceBoundaryConstraint
+StartNavigableSpace: xs::ID
+EndNavigableSpace: xs::ID
<<Constraint>> PassRestriction
+PassInfo: PassAvilableType
<<Constraint>> WalkTypeRestriction
+WalkTypeInfo: WalkType
<<Constraint>> UserRestriction
+PedestrianInfo: PedestrianType
<<Constraint>> TimeRestriction
+TimeInfo: TimeRestrictions[0..1]
<<Constraint>> WalkTypeRestriction
+WalkTypeInfo: WalkType
<<Constraint>> TimeRestriction
+TimeInfo: TimeRestrictions[0..1]
<<Constraint>> UserRestriction
+PedestrianInfo: PedestrianType
<<Constraint>> OneWayRestriction
+ OneWayInfo: OneWayRestrictionsType
TimeRestrictons
0..*
TimeRestrictonElement
+StartDate: xs::dateTime
+EndDate: xs::dateTime
PedestrianType
Normal
Authorized
WalkType
Normal
Wheelchair
Vehicle
PassAvailableType
Possible
Impossible
OneWayRestrictionsType
Only Negative Direction
Only Positive Direction
Both Directions

Appendix Figure III - Concept diagram of IndoorGML Constraints module concept (OGC, 2014)

MLSEM Constraints Concept


«FeatureType» SpaceElement
+ symbolicId :GenericName [0..*]
+ class :GenericName [0..1]
+ function :GenericName [0..*]
+ usage :GenericName [0..*]
+ genericAttribute :GenericAttributeType [0..*]
{(primal->notEmpty() and dual->notEmpty()) implies primal.dual = dual}
«FeatureType» SpaceCell
+ makeDisjoint(SpaceCell) :Set<SpaceCell>
«FeatureType» BoundaryCell
+spaceCell
+super 0..*
+sub 0..*
SubSpace
+boundaryCell
PrimalSpace
DualSpace
+cell 1
+primal 0..1
«FeatureType» Space
«FeatureType» SpaceBoundary
+space
Boundary
+boundary
+spoke
CoBoundary
+hub
Duality
+dual 0..1
«FeatureType» State
+ cost :GenericAttributeType [0..*]
«FeatureType» Transition
+ cost :GenericAttributeType [0..*]
+edge
Layer
+node
«FeatureType» InterLayerEdge
+ topoRelation :TopoRelationEnum
{boundary->first().graph <> boundary->last().graph}
+graph
«Type» IntraLayerGraph
{layer.spaceCell.dual->includesAll(node)} {layer.boundaryCell.dual->includesAll(edge)}
+subGraph
«Type» MultilayeredGraph
{subGraph.layer = layerComplex.layer}
Graph
+layer
+layerComplex
«FeatureType» SpaceLayer
+ symbolicId :GenericName [0..*]
+ type :SpaceLayerType
+ name :CharacterString [0..1]
+ description :CharacterString [0..1]
+ genericAttribute :GenericAttributeType [0..*]
+ merge(SpaceLayer) :SpaceLayer
+ difference(SpaceLayer) :SpaceLayer
+ intersection(SpaceLayer) :SpaceLayer
+complex
Complex
«FeatureType» SpaceLayerComplex
+ name :CharacterString [0..1]
+ description :CharacterString [0..1]
+ genericAttribute :GenericAttributeType [0..*]
«Enumeration» TopoRelationEnum
containsOrInside
coversOrCoveredBy
equal
overlap
«CodeList» SpaceLayerType
+ topograhpic
+ sensor
+ logical
+ ...
«DataType» GenericAttributeType
+ name :GenericName
+ value :Any
+ typeName :GenericName
+ definition :CharacterString [0..1]
The value of "value" shall be an instance of a class that is a realization of GF_AttributeType and shall match the type given by "typeName"

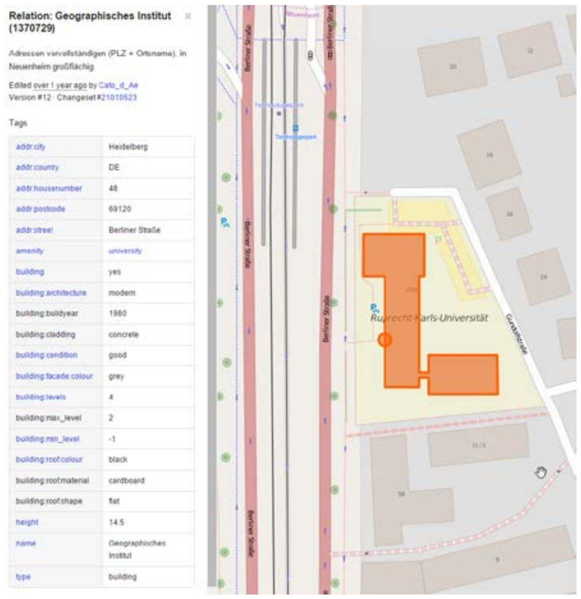
Appendix Figure IV - The semantic concepts of the Space Representation package (Nagel, C., 2014)

OSM XML Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900"
maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882"
visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744"
visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>
 <node id="1831881213" version="1" changeset="12370172" lat="54.0900666"
lon="12.2539381" user="lafkor" uid="75625" visible="true" timestamp="2012-07-
20T09:43:19Z">
  <tag k="name" v="Neu Broderstorf"/>
  <tag k="traffic_sign" v="city_limit"/>
 </node>
 ...
 <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882"
visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
 <way id="26659127" user="Masch" uid="55988" visible="true" version="5"
changeset="4142606" timestamp="2010-03-16T11:47:08Z">
  <nd ref="292403538"/>
  <nd ref="298884289"/>
  ...
  <nd ref="261728686"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Pastower Straße"/>
 </way>
 <relation id="56688" user="kmvar" uid="56190" visible="true" version="28"
changeset="6947637" timestamp="2011-01-12T14:23:49Z">
  <member type="node" ref="294942404" role=""/>
  ...
  <member type="node" ref="364933006" role=""/>
  <member type="way" ref="4579143" role=""/>
  ...
  <member type="node" ref="249673494" role=""/>
  <tag k="name" v="Küstenbus Linie 123"/>
  <tag k="network" v="VVW"/>
  <tag k="operator" v="Regionalverkehr Küste"/>
  <tag k="ref" v="123"/>
  <tag k="route" v="bus"/>
  <tag k="type" v="route"/>
 </relation>
 ...
</osm>
```

OSM XML Example, including Node, Way and Relation Elements **(OSM Wiki, 2015)**

OSM Relation Example



Relation: Geographisches Institut (1370729)

Adressen vervollständigen (PLZ + Ortsname), in Neuenheim großflächig

Edited over 1 year ago by Cato_d_Ae
Version #12 · Changeset #21010523

Tags

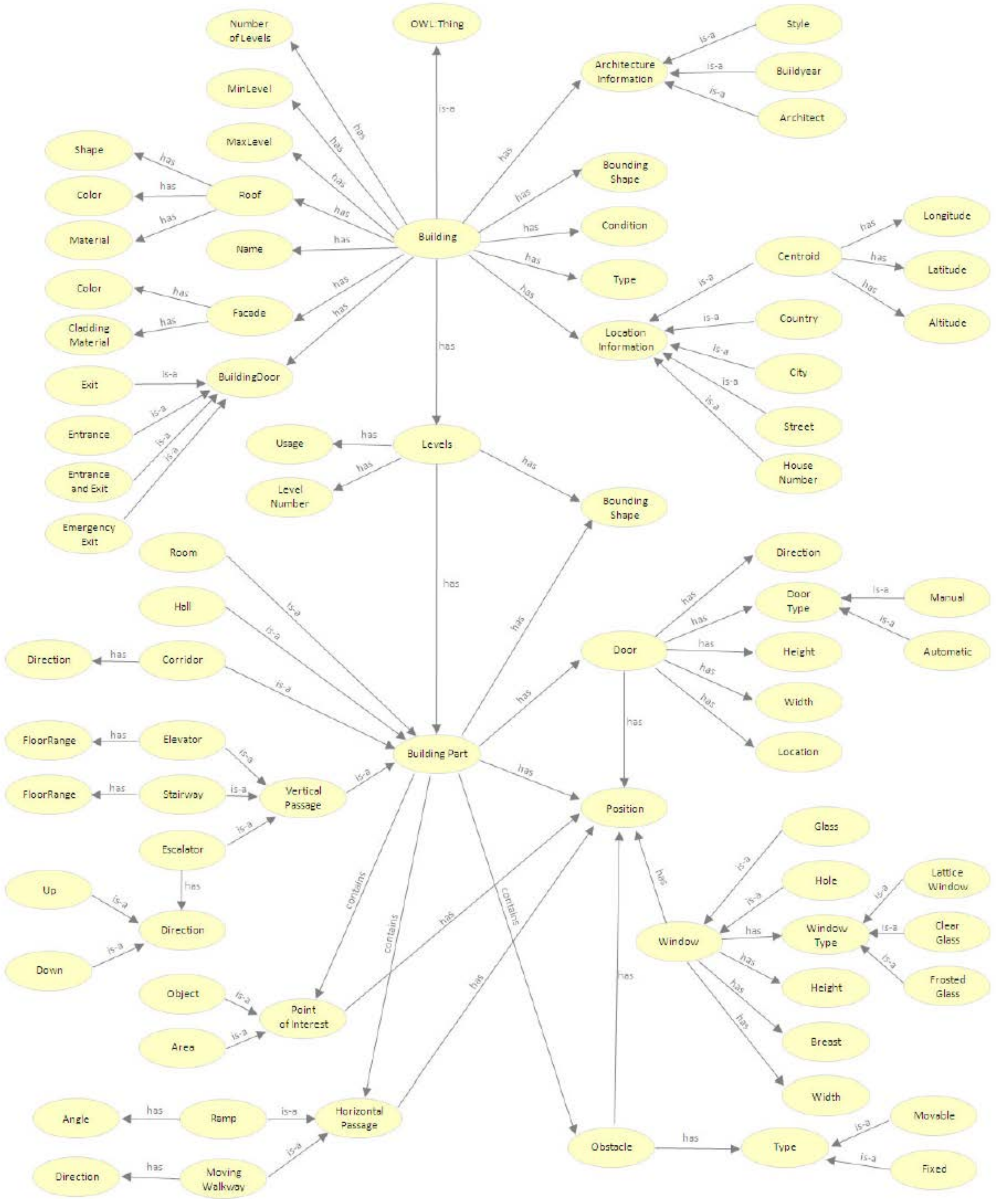
addr:city	Heidelberg
addr:country	DE
addr:housenumber	48
addr:postcode	69120
addr:street	Berliner Straße
amenity	university
building	yes
building:architecture	modern
building:buildyear	1980
building:cladding	concrete
building:condition	good
building:facade:colour	grey
building:levels	4
building:max_level	2
building:min_level	-1
building:roof:colour	black
building:roof:material	cardboard
building:roof:shape	flat
height	14.5
name	Geographisches Institut
type	building

Appendix Figure V - OSM Relation Example – OSM Website representation

```
<osm version="0.6" generator="CGImap 0.4.0 (23030 thorn-
02.openstreetmap.org)" copyright="OpenStreetMap and
contributors"attribution="http://www.openstreetmap.org/copyright" license="http://opend
atacommons.org/licenses/odbl/1-0/">
<relation id="1370729" visible="true" version="12" changeset="21010523" timestamp="2014
-03-09T18:00:32Z" user="Cato_d_Ae" uid="1679807">
<member type="relation" ref="1370727" role="level_-1"/>
<member type="relation" ref="1370728" role="level_0"/>
<member type="relation" ref="1370725" role="level_1"/>
<member type="relation" ref="1370726" role="level_2"/>
<member type="node" ref="1098227410" role="entrance"/>
<member type="way" ref="94551367" role="outer"/>
<tag k="addr:city" v="Heidelberg"/>
<tag k="addr:country" v="DE"/>
<tag k="addr:housenumber" v="48"/>
<tag k="addr:postcode" v="69120"/>
<tag k="addr:street" v="Berliner Straße"/>
<tag k="amenity" v="university"/>
<tag k="building" v="yes"/>
<tag k="building:architecture" v="modern"/>
<tag k="building:buildyear" v="1980"/>
<tag k="building:cladding" v="concrete"/>
<tag k="building:condition" v="good"/>
<tag k="building:facade:colour" v="grey"/>
<tag k="building:levels" v="4"/>
<tag k="building:max_level" v="2"/>
<tag k="building:min_level" v="-1"/>
<tag k="building:roof:colour" v="black"/>
<tag k="building:roof:material" v="cardboard"/>
<tag k="building:roof:shape" v="flat"/>
<tag k="height" v="14.5"/>
<tag k="name" v="Geographisches Institut"/>
<tag k="type" v="building"/>
</relation>
</osm>
```

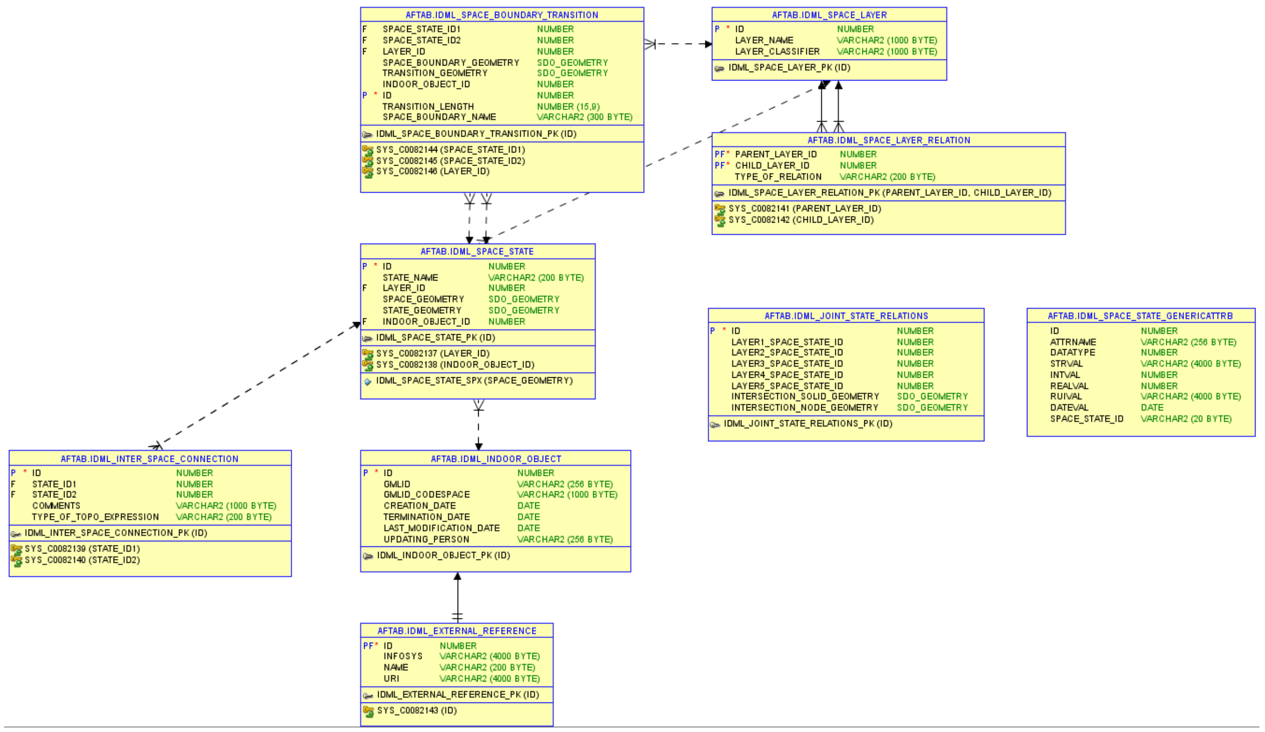
OSM Relation Example – OSM Website representation and XML Data **(OSM, 2015)**

IndoorOSM - 3D Building Ontology



Appendix Figure VI - IndoorOSM - 3D Building Ontology for describing the inside and outside of a building (Goetz & Zipf, 2011)

Sample IndoorGML data structure



```
AFTAB.IDML_SPACE_BOUNDARY_TRANSITION
F      SPACE_STATE_ID1            NUMBER
F      SPACE_STATE_ID2            NUMBER
F      LAYER_ID                   NUMBER
       SPACE_BOUNDARY_GEOMETRY    SDO_GEOMETRY
       TRANSITION_GEOMETRY        SDO_GEOMETRY
       INDOOR_OBJECT_ID           NUMBER
P  *   ID                         NUMBER
       TRANSITION_LENGTH          NUMBER (15,9)
       SPACE_BOUNDARY_NAME        VARCHAR2 (300 BYTE)
IDML_SPACE_BOUNDARY_TRANSITION_PK (ID)
SYS_C0082144 (SPACE_STATE_ID1)
SYS_C0082145 (SPACE_STATE_ID2)
SYS_C0082146 (LAYER_ID)
```

```
AFTAB.IDML_SPACE_LAYER
P  *   ID                   NUMBER
       LAYER_NAME           VARCHAR2 (1000 BYTE)
       LAYER_CLASSIFIER     VARCHAR2 (1000 BYTE)
IDML_SPACE_LAYER_PK (ID)
```

```
AFTAB.IDML_SPACE_LAYER_RELATION
PF*    PARENT_LAYER_ID      NUMBER
PF*    CHILD_LAYER_ID       NUMBER
       TYPE_OF_RELATION     VARCHAR2 (200 BYTE)
IDML_SPACE_LAYER_RELATION_PK (PARENT_LAYER_ID, CHILD_LAYER_ID)
SYS_C0082141 (PARENT_LAYER_ID)
SYS_C0082142 (CHILD_LAYER_ID)
```

```
AFTAB.IDML_SPACE_STATE
P  *   ID                   NUMBER
       STATE_NAME           VARCHAR2 (200 BYTE)
F      LAYER_ID             NUMBER
       SPACE_GEOMETRY       SDO_GEOMETRY
       STATE_GEOMETRY       SDO_GEOMETRY
F      INDOOR_OBJECT_ID     NUMBER
IDML_SPACE_STATE_PK (ID)
SYS_C0082137 (LAYER_ID)
SYS_C0082138 (INDOOR_OBJECT_ID)
IDML_SPACE_STATE_SPX (SPACE_GEOMETRY)
```

```
AFTAB.IDML_JOINT_STATE_RELATIONS
P  *   ID                              NUMBER
       LAYER1_SPACE_STATE_ID           NUMBER
       LAYER2_SPACE_STATE_ID           NUMBER
       LAYER3_SPACE_STATE_ID           NUMBER
       LAYER4_SPACE_STATE_ID           NUMBER
       LAYER5_SPACE_STATE_ID           NUMBER
       INTERSECTION_SOLID_GEOMETRY     SDO_GEOMETRY
       INTERSECTION_NODE_GEOMETRY      SDO_GEOMETRY
IDML_JOINT_STATE_RELATIONS_PK (ID)
```

```
AFTAB.IDML_SPACE_STATE_GENERICATTRB
       ID                   NUMBER
       ATTRNAME             VARCHAR2 (256 BYTE)
       DATATYPE             NUMBER
       STRVAL               VARCHAR2 (4000 BYTE)
       INTVAL               NUMBER
       REALVAL              NUMBER
       RUIVAL               VARCHAR2 (4000 BYTE)
       DATEVAL              DATE
       SPACE_STATE_ID       VARCHAR2 (20 BYTE)
```

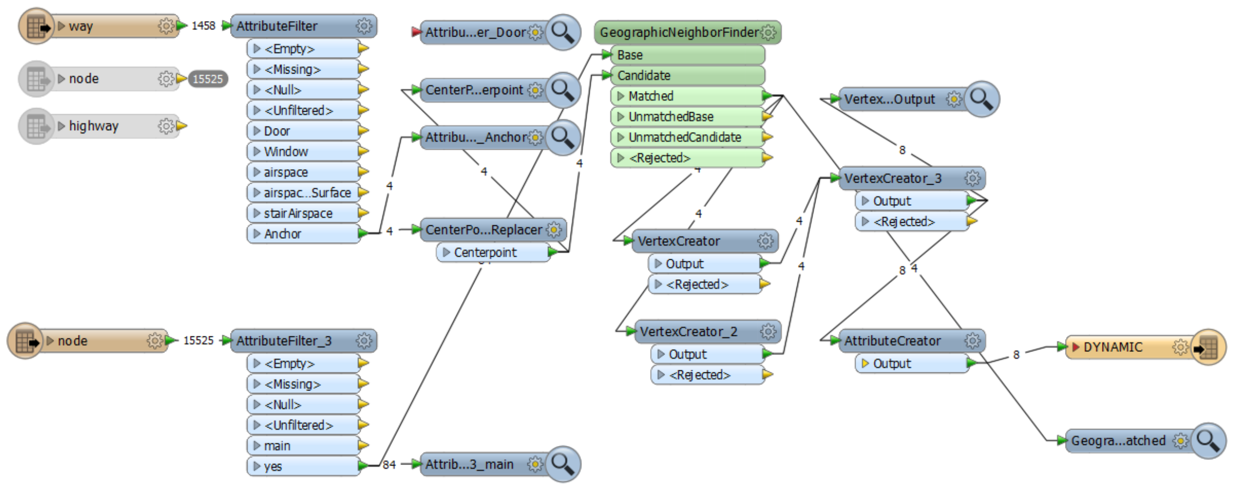
```
AFTAB.IDML_INTER_SPACE_CONNECTION
P  *   ID                          NUMBER
F      STATE_ID1                   NUMBER
F      STATE_ID2                   NUMBER
       COMMENTS                    VARCHAR2 (1000 BYTE)
       TYPE_OF_TOPO_EXPRESSION     VARCHAR2 (200 BYTE)
IDML_INTER_SPACE_CONNECTION_PK (ID)
SYS_C0082139 (STATE_ID1)
SYS_C0082140 (STATE_ID2)
```

```
AFTAB.IDML_INDOOR_OBJECT
P  *   ID                          NUMBER
       GMLID                       VARCHAR2 (256 BYTE)
       GMLID_CODESPACE             VARCHAR2 (1000 BYTE)
       CREATION_DATE               DATE
       TERMINATION_DATE            DATE
       LAST_MODIFICATION_DATE      DATE
       UPDATING_PERSON             VARCHAR2 (256 BYTE)
IDML_INDOOR_OBJECT_PK (ID)
```

```
AFTAB.IDML_EXTERNAL_REFERENCE
PF*    ID            NUMBER
       INFOSYS       VARCHAR2 (4000 BYTE)
       NAME          VARCHAR2 (200 BYTE)
       URI           VARCHAR2 (4000 BYTE)
IDML_EXTERNAL_REFERENCE_PK (ID)
SYS_C0082143 (ID)
```

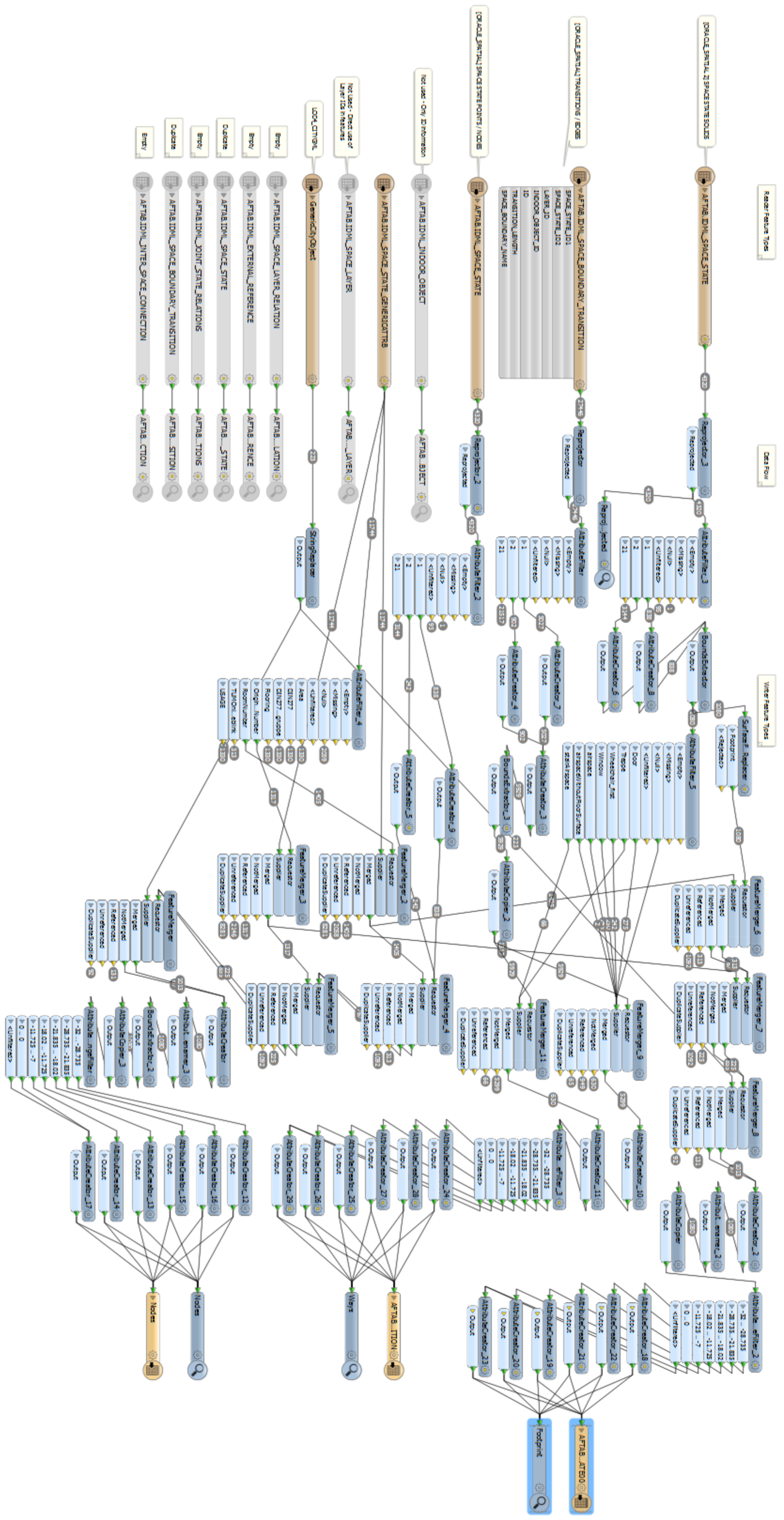
Appendix Figure VII - Sample IndoorGML data structure in Oracle Spatial DB – Oracle SQL Developer Data Modeller representation

"Anchor2Entrance" FME Workbench



Appendix Figure VIII – "Anchor2Entrance" FME Workbench for "outdoor" OSM "entrance" to "indoor" OSM "anchor" feature matching

“IndoorGML_Oracle2OSM” FME Workbench



Appendix Figure IX - “Anchor2Entrance” FME Workbench for "indoor" OSM data generation from IndoorGML data stored in Oracle Spatial / Non-Spatial DB

Mock GTFS Dataset

The mock GTFS dataset that was created for the implementation represents two stops for a bus type transit. Routes were created for both directions of travel at 5 minute intervals for the entire day, with a schedule covering every day in the second half of 2015. The dataset consists of the following files: agency.txt, calendar.txt, calendar_dates.txt, routes.txt, stop_times.txt, stops.txt and trips.txt. Samples and complete files will be presented below:

agency.txt – Complete file with mock Agency information: Agency ID, Name, URL, Timezone, Language.

```
agency_id,agency_name,agency_url,agency_timezone,agency_lang,agency_phone
MVV,MVV,http://www.mvv.de,Europe/Berlin,de,
```

calendar.txt – Complete file with mock calendar information: Service ID, days of service and start/end dates.

```
service_id,monday,tuesday,wednesday,thursday,friday,saturday,sunday,start_date,end_date
01,1,1,1,1,1,1,1,20150604,20151212
02,1,1,1,1,1,1,1,20150604,20151212
```

calendar_dates.txt – File sample with mock calendar dates information: Service ID, dates, exceptions in schedule.

```
service_id,date,exception_type
01,20150604,1
01,20150605,1
01,20150606,1
…
02,20151210,1
02,20151211,1
02,20151212,1
```

routes.txt – Complete file with mock route information: Route ID, Agency ID, route short and long names, route type (Bus), route URL.

```
route_id,agency_id,route_short_name,route_long_name,route_desc,route_type,route_url,rou
te_color,route_text_color
01,MVV,TtoK,TheresienstrtoKoenigspl,,3,http://www.mvv.de,,
02,MVV,KtoT,KoenigspltoTheresienstr,,3,http://www.mvv.de,,
```

stop_times.txt – File sample with mock stop times information: Trip ID, arrival & departure time, stop id, stop sequence, pickup & drop off type.

```
trip_id,arrival_time,departure_time,stop_id,stop_sequence,stop_headsign,pickup_type,dro
p_off_type,shape_dist_traveled
1,4:45:00,4:45:00,01,1,,0,0,
1,4:46:00,4:46:00,02,2,,0,0,
2,4:55:00,4:55:00,01,1,,0,0,
2,4:56:00,4:56:00,02,2,,0,0,
…
285,4:15:00,4:15:00,02,1,,0,0,
285,4:16:00,4:16:00,01,2,,0,0,
286,4:25:00,4:25:00,02,1,,0,0,
286,4:26:00,4:26:00,01,2,,0,0,
```

stops.txt – Complete file with mock stops information: Stop ID, Name, Stop latitude and longitude, location type.

```
stop_id,stop_code,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_ur
l,location_type,parent_station
01,,Theresienstrasse,,48.15176,11.56465,,,0,
02,,Koenigsplatz,,48.14519,11.56338,,,0,
```

trips.txt – File sample with mock trips information: Route ID, Service ID, Trip ID, trip short name.

```
route_id,service_id,trip_id,trip_headsign,trip_short_name,direction_id,block_id,shape_i
d
01,01,1,"TtoK_1",,,,
01,01,2,"TtoK_1",,,,
01,01,3,"TtoK_1",,,,
…
02,02,284,"KtoT_1",,,,
02,02,285,"KtoT_1",,,,
02,02,286,"KtoT_1",,,,
```

Appendix II – OTP & Nominatim Source Code Modifications

This Appendix includes changes made to the original OTP and Nominatim source code in order to enable functionality needed for their use in the thesis project proof of concept implementation. OTP Source Code changes were generated using the GIT diff functionality of Github's GIT Shell tool and follow the standard formatting settings. Removed lines are indicated by a leading minus sign (-) while added lines are indicated by a leading plus sign (+). The files modified are named at the beginning of each code snippet, with indicators for the source, a being the official OTP code repository and b being the local custom code repository. Nominatim required minimal alteration to the source code (one location) and is thus added manually, following the same formatting (leading plus sign for additions).

Inclusion of Nominatim Geocoder script in the built-in Leaflet OTP client index.html page

```
diff --git a/src/client/index.html b/src/client/index.html
index 572667f..3a83de8 100644
--- a/src/client/index.html
+++ b/src/client/index.html
@@ -85,6 +85,7 @@
 <script src="js/otp/core/Geocoder.js?v=1"></script>
 <script src="js/otp/core/GeocoderBuiltin.js?v=1"></script>
 <script src="js/otp/core/GeocoderBag.js?v=1"></script>
+<script src="js/otp/core/GeocoderNominatim.js?v=1"></script>
 <script src="js/otp/core/SOLRGeocoder.js?v=1"></script>
 <script src="js/otp/core/TransitIndex.js?v=1"></script>
 <script src="js/otp/core/IndexApi.js?v=1"></script>
```

Modifications in the built-in Leaflet OTP client config.js JavaScript file:
Change of web service location to http://localhost
Change of OTP units to metric
Replacement of built-in OTP Geocoder with Nominatim Geocoder

```
diff --git a/src/client/js/otp/config.js b/src/client/js/otp/config.js
index 5cae24e..b4ba652 100644
--- a/src/client/js/otp/config.js
+++ b/src/client/js/otp/config.js
@@ -44,7 +44,7 @@ otp.config = {
     /**
      * The OTP web service locations
      */
-    hostname : "",
+    hostname : window.location.protocol+"//"+window.location.host,
     //municoderHostname : "http://localhost:8080",
     //datastoreUrl : 'http://localhost:9000',
     // In the 0.10.x API the base path is "otp-rest-servlet/ws"
@@ -121,7 +121,7 @@ otp.config = {
     showLogo            : true,
     showTitle           : true,
     showModuleSelector  : true,
-    metric              : false,
+    metric              : true,


     /**
@@ -163,9 +163,10 @@ otp.config = {

     geocoders : [
         {
-            name: 'OTP built-in geocoder',
-            className: 'otp.core.GeocoderBuiltin'
-            // URL and query parameter do not need to be set for built-in geocoder.
+            name: 'Nominatim',
+            className: 'otp.core.GeocoderNominatim',
+            url: '/nominatim/search.php?format=json',
+            addressParam: 'q'
         }
     ],
```

Nominatim Geocoder JavaScript .js file, created via modification of Geocoder script samples included in OTP

```
diff --git a/src/client/js/otp/core/GeocoderNominatim.js
b/src/client/js/otp/core/GeocoderNominatim.js

new file mode 100644
index 0000000..d716f7c
--- /dev/null
+++ b/src/client/js/otp/core/GeocoderNominatim.js
@@ -0,0 +1,33 @@
+/* This program is free software: you can redistribute it and/or
+ modify it under the terms of the GNU Lesser General Public License
+ as published by the Free Software Foundation, either version 3 of
+ the License, or (at your option) any later version.
+
+ This program is distributed in the hope that it will be useful,
+ but WITHOUT ANY WARRANTY; without even the implied warranty of
+ MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ GNU General Public License for more details.
+
+ You should have received a copy of the GNU General Public License
+ along with this program.  If not, see <http://www.gnu.org/licenses/>.
+ */
+
+otp.namespace("otp.core");
+
+otp.core.GeocoderNominatim = otp.Class({
+
+     url:
window.location.protocol+'//'+window.location.hostname+'/nominatim/search.php?format=js
on',
+
+    initialize : function(url, addressParam) {
+        // Do nothing, the proper address and query param are already known.
+    },
+
+    geocode : function(address, callback) {
+        // The built in geocoder returns results in the form expected by the client:
+        // A JSON array of objects containing lat, lng, and description fields.
+        $.getJSON(this.url, {q: address}, function(response) {
+            callback.call(this, response);
+        });
+    }
+
+});
\ No newline at end of file
```

Addition of Indoor Preference selector in Multimodal Planner JavaScript Module of the built-in Leaflet OTP client. (Based on Wheelchair code)

```
diff --git a/src/client/js/otp/modules/multimodal/MultimodalPlannerModule.js
b/src/client/js/otp/modules/multimodal/MultimodalPlannerModule.js
index 1d8b9e6..99c5da7 100644
--- a/src/client/js/otp/modules/multimodal/MultimodalPlannerModule.js
+++ b/src/client/js/otp/modules/multimodal/MultimodalPlannerModule.js
@@ -72,6 +72,7 @@ otp.modules.multimodal.MultimodalPlannerModule =
         modeSelector.addModeControl(new
otp.widgets.tripoptions.PreferredRoutes(this.optionsWidget));
         modeSelector.addModeControl(new
otp.widgets.tripoptions.BannedRoutes(this.optionsWidget));
         modeSelector.addModeControl(new
otp.widgets.tripoptions.WheelChairSelector(this.optionsWidget));
+        modeSelector.addModeControl(new
otp.widgets.tripoptions.IndoorSelector(this.optionsWidget));

         modeSelector.refreshModeControls();
```

Addition of Indoor Preference selector default values in Planner JavaScript Module of the built-in Leaflet OTP client. (Based on Wheelchair code)

```
diff --git a/src/client/js/otp/modules/planner/PlannerModule.js
b/src/client/js/otp/modules/planner/PlannerModule.js
index 93e1a70..66e265d 100644
--- a/src/client/js/otp/modules/planner/PlannerModule.js
+++ b/src/client/js/otp/modules/planner/PlannerModule.js
@@ -21,6 +21,7 @@ otp.modules.planner.defaultQueryParams = {
     date                          :
moment().format(otp.config.locale.time.date_format),
     arriveBy                      : false,
     wheelchair                    : false,
+    indoor                        : 0,
     mode                          : "TRANSIT,WALK",
     maxWalkDistance               : 804.672, // 1/2 mi.
     metricDefaultMaxWalkDistance  : 750, // meters
@@ -339,6 +340,7 @@ otp.modules.planner.PlannerModule =
             };
             if(this.arriveBy !== null) _.extend(queryParams, { arriveBy :
this.arriveBy } );
```

```
             if(this.wheelchair !== null) _.extend(queryParams, { wheelchair :
this.wheelchair });
+            if(this.indoor !== null) _.extend(queryParams, { indoor : this.indoor });
             if(this.preferredRoutes !== null) {
                 queryParams.preferredRoutes = this.preferredRoutes;
                 if(this.otherThanPreferredRoutesPenalty !== null)
```

Addition of Indoor Preference selector functionality in Trip Options JavaScript Widget of the built-in Leaflet OTP client.

```
diff --git a/src/client/js/otp/widgets/tripoptions/TripOptionsWidget.js
b/src/client/js/otp/widgets/tripoptions/TripOptionsWidget.js
index df05261..254a2e3 100644
--- a/src/client/js/otp/widgets/tripoptions/TripOptionsWidget.js
+++ b/src/client/js/otp/widgets/tripoptions/TripOptionsWidget.js
@@ -451,6 +451,48 @@ otp.widgets.tripoptions.WheelChairSelector =
     }
 });

+//** IndoorSelector **//
+
+otp.widgets.tripoptions.IndoorSelector =
+    otp.Class(otp.widgets.tripoptions.TripOptionsWidgetControl, {
+
+    id           :  null,
+    //TRANSLATORS: label for checkbox
+    label        : _tr("Indoor/Outdoor preference:"),
+
+    initialize : function(tripWidget) {
+
+
otp.widgets.tripoptions.TripOptionsWidgetControl.prototype.initialize.apply(this,
arguments);
+
+        this.id = tripWidget.id;
+
+
+        ich['otp-tripOptions-indoor']({
+            widgetId : this.id,
+            label : this.label,
+        }).appendTo(this.$());
+
+    },
+
+    doAfterLayout : function() {
+        var this_ = this;
+
+        $("#"+this.id+"-indoor-input").change(function() {
+            this_.tripWidget.module.indoor = $("#"+this_.id+"-indoor-input").val();
+        });
+    },
+
+    restorePlan : function(data) {
+        if(data.queryParams.indoor) {
+            $("#"+this.id+"-indoor-input").val(data.queryParams.indoor);
+        }
+    },
+
+    isApplicableForMode : function(mode) {
+        return true;
+    }
+});
+

 //** ModeSelector **//
```

Addition of possible values for the Indoor Preference selector in trip options template HTML file of the built-in Leaflet OTP client.

```
diff --git a/src/client/js/otp/widgets/tripoptions/tripoptions-templates.html
b/src/client/js/otp/widgets/tripoptions/tripoptions-templates.html
index 81dec7b..abad9db 100644
--- a/src/client/js/otp/widgets/tripoptions/tripoptions-templates.html
+++ b/src/client/js/otp/widgets/tripoptions/tripoptions-templates.html
@@ -74,6 +74,21 @@

 </script>

+<!-- IndoorSelector -->
+
+<script id="otp-tripOptions-indoor" type="text/html">
+
+    <div class="notDraggable">
+        {{label}} &nbsp;
+                <select id="{{widgetId}}-indoor-input">
+                    <option value="0" selected="selected">Any</option>
+                    <option value="1">Indoor</option>
+                    <option value="2">Outdoor</option>
+                </select>
+    </div>
+
+</script>
+
 <!-- MaxDistanceSelector -->

 <script id="otp-tripOptions-maxDistance" type="text/html">
```

Addition of indoor trip type in the Trip Types java source code of the OTP API. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/api/adapters/TripType.java
b/src/main/java/org/opentripplanner/api/adapters/TripType.java
index 8421616..6a45108 100644
--- a/src/main/java/org/opentripplanner/api/adapters/TripType.java
+++ b/src/main/java/org/opentripplanner/api/adapters/TripType.java
@@ -101,6 +101,10 @@ public class TripType {

     @XmlAttribute
     @JsonSerialize
+    Integer indoor;
+
+    @XmlAttribute
+    @JsonSerialize
     Integer tripBikesAllowed;

     @XmlAttribute
```

Addition of indoor trip type in the Routing Resources java source code of the OTP API. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/api/common/RoutingResource.java
b/src/main/java/org/opentripplanner/api/common/RoutingResource.java
index 4215a15..dee9add 100644
--- a/src/main/java/org/opentripplanner/api/common/RoutingResource.java
+++ b/src/main/java/org/opentripplanner/api/common/RoutingResource.java
@@ -90,6 +90,10 @@ public abstract class RoutingResource {
     @QueryParam("wheelchair")
     protected Boolean wheelchair;

+    /** Whether the trip must prefer indoor streets. */
+    @QueryParam("indoor")
+    protected Integer indoor;
+
     /** The maximum distance (in meters) the user is willing to walk. Defaults to
unlimited. */
     @QueryParam("maxWalkDistance")
     protected Double maxWalkDistance;
@@ -402,6 +406,9 @@ public abstract class RoutingResource {
         if (wheelchair != null)
             request.setWheelchairAccessible(wheelchair);

+        if (indoor != null)
+            request.setIndoor(indoor);
+
         if (numItineraries != null)
             request.setNumItineraries(numItineraries);
```

Addition of support for indoor tags in the OSM Module java source code of the OTP Graphbuilder.
Default value is set to false. (Based on Wheelchair code)

```
diff --git
a/src/main/java/org/opentripplanner/graph_builder/module/osm/OpenStreetMapModule.java
b/src/main/java/org/opentripplanner/graph_builder/module/osm/OpenStreetMapModule.java
index 6752785..9ce675f 100644
---
a/src/main/java/org/opentripplanner/graph_builder/module/osm/OpenStreetMapModule.java
+++
b/src/main/java/org/opentripplanner/graph_builder/module/osm/OpenStreetMapModule.java
@@ -773,6 +773,7 @@ public class OpenStreetMapModule implements GraphBuilderModule {
```

```
                     // default permissions: pedestrian, wheelchair, and bicycle
                     boolean wheelchairAccessible = true;
+                    boolean indoor = false;
                     StreetTraversalPermission permission =
StreetTraversalPermission.PEDESTRIAN_AND_BICYCLE;
                     // check for bicycle=no, otherwise assume it's OK to take a bike
                     if (node.isTagFalse("bicycle")) {
@@ -782,6 +783,9 @@ public class OpenStreetMapModule implements GraphBuilderModule {
                     if (node.isTagFalse("wheelchair")) {
                         wheelchairAccessible = false;
                     }
+                    if (node.isTagTrue("indoor")) {
+                        indoor = true;
+                    }

                     // The narrative won't be strictly correct, as it will show the
elevator as part
                     // of the cycling leg, but I think most cyclists will figure out
that they
@@ -790,6 +794,8 @@ public class OpenStreetMapModule implements GraphBuilderModule {
                     ElevatorHopEdge backEdge = new ElevatorHopEdge(to, from,
permission);
                     foreEdge.wheelchairAccessible = wheelchairAccessible;
                     backEdge.wheelchairAccessible = wheelchairAccessible;
+                    foreEdge.indoor = indoor;
+                    backEdge.indoor = indoor;
                 }
             } // END elevator edge loop
         }
@@ -1079,6 +1085,10 @@ public class OpenStreetMapModule implements GraphBuilderModule {
                 street.setWheelchairAccessible(false);
             }

+            if (way.isTagTrue("indoor")) {
+                street.setIndoor(true);
+            }
+
             street.setSlopeOverride(wayPropertySet.getSlopeOverride(way));

             // < 0.04: account for
```

Addition of support for indoor tags in the OSM Walkable Area Builder java source code of the OTP Graphbuilder. (Based on Wheelchair code)

```
diff --git
a/src/main/java/org/opentripplanner/graph_builder/module/osm/WalkableAreaBuilder.java
b/src/main/java/org/opentripplanner/graph_builder/module/osm/WalkableAreaBuilder.java
index b77a20b..37fe41c 100644
---
a/src/main/java/org/opentripplanner/graph_builder/module/osm/WalkableAreaBuilder.java
+++
b/src/main/java/org/opentripplanner/graph_builder/module/osm/WalkableAreaBuilder.java
@@ -396,6 +396,10 @@ public class WalkableAreaBuilder {
                 street.setWheelchairAccessible(false);
```

```
             }

+            if (areaEntity.isTagTrue("indoor")) {
+                street.setIndoor(true);
+            }
+
             street.setStreetClass(cls);
             edges.add(street);

@@ -415,6 +419,10 @@ public class WalkableAreaBuilder {
                 street.setWheelchairAccessible(false);
             }

+            if (areaEntity.isTagTrue("indoor")) {
+                street.setIndoor(false);
+            }
+
             backStreet.setStreetClass(cls);
             edges.add(backStreet);
```

Addition of support for indoor routing requests in the Routing Request java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/core/RoutingRequest.java
b/src/main/java/org/opentripplanner/routing/core/RoutingRequest.java
index 0203332..50fcd18 100644
--- a/src/main/java/org/opentripplanner/routing/core/RoutingRequest.java
+++ b/src/main/java/org/opentripplanner/routing/core/RoutingRequest.java
@@ -112,6 +112,9 @@ public class RoutingRequest implements Cloneable, Serializable {
     /** Whether the trip must be wheelchair accessible. */
     public boolean wheelchairAccessible = false;

+    /** Whether the trip should prefer indoor streets. */
+    public int indoor = 0;
+
     /** The maximum number of itineraries to return. */
     public int numItineraries = 3;

@@ -515,6 +518,10 @@ public class RoutingRequest implements Cloneable, Serializable {
         this.wheelchairAccessible = wheelchairAccessible;
     }

+    public void setIndoor(int indoor) {
+        this.indoor = indoor;
+    }
+
     /**
      * only allow traversal by the specified mode; don't allow walking bikes. This is
used during contraction to reduce the number of possible paths.
      */
@@ -876,6 +883,7 @@ public class RoutingRequest implements Cloneable, Serializable {
                 && maxTransfers == other.maxTransfers
                 && modes.equals(other.modes)
                 && wheelchairAccessible == other.wheelchairAccessible
+                && indoor == other.indoor
```

```
                 && optimize.equals(other.optimize)
                 && maxWalkDistance == other.maxWalkDistance
                 && maxPreTransitTime == other.maxPreTransitTime
@@ -929,6 +937,7 @@ public class RoutingRequest implements Cloneable, Serializable {
                 + new Double(carSpeed).hashCode() + new Double(maxWeight).hashCode()
                 + (int) (worstTime & 0xffffffff) + modes.hashCode()
                 + (arriveBy ? 8966786 : 0) + (wheelchairAccessible ? 731980 : 0)
+                + (int) indoor + (int) indoor * 11027
                 + optimize.hashCode() + new Double(maxWalkDistance).hashCode()
                 + new Double(transferPenalty).hashCode() + new
Double(maxSlope).hashCode()
                 + new Double(walkReluctance).hashCode() + new
Double(waitReluctance).hashCode()
```

Addition of requirements for indoor routing in the Traversal Requirements java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/core/TraversalRequirements.java
b/src/main/java/org/opentripplanner/routing/core/TraversalRequirements.java
index 62029d2..4af001c 100644
--- a/src/main/java/org/opentripplanner/routing/core/TraversalRequirements.java
+++ b/src/main/java/org/opentripplanner/routing/core/TraversalRequirements.java
@@ -26,6 +26,11 @@ public class TraversalRequirements {
     private boolean wheelchairAccessible = false;

     /**
+     * If true, trip must prefer indoor streets.
+     */
+    private int indoor = 0;
+
+    /**
      * The maximum slope of streets for wheelchair trips.
      *
      * ADA max wheelchair ramp slope is a good default.
@@ -77,6 +82,7 @@ public class TraversalRequirements {
     private static void initFromRoutingRequest(TraversalRequirements req,
RoutingRequest options) {
         req.modes = options.modes.clone();
         req.wheelchairAccessible = options.wheelchairAccessible;
+        req.indoor = options.indoor;
         req.maxWheelchairSlope = options.maxSlope;
         req.maxWalkDistance = options.maxWalkDistance;
     }
```

Addition of indoor flags for edges in the Elevator Hop Edge java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/edgetype/ElevatorHopEdge.java
b/src/main/java/org/opentripplanner/routing/edgetype/ElevatorHopEdge.java
index 315b8bb..b2e6af2 100644
--- a/src/main/java/org/opentripplanner/routing/edgetype/ElevatorHopEdge.java
+++ b/src/main/java/org/opentripplanner/routing/edgetype/ElevatorHopEdge.java
@@ -36,6 +36,8 @@ public class ElevatorHopEdge extends Edge implements ElevatorEdge {
     public boolean wheelchairAccessible = true;

+    public boolean indoor = false;
+
     public ElevatorHopEdge(Vertex from, Vertex to, StreetTraversalPermission
permission) {
         super(from, to);
         this.permission = permission;
```

Addition of indoor flags for edges in the Street Edge java source code of the OTP Routing Engine. (Based on Wheelchair code)
With Street Edges comprising the bulk of edges in Graph Builder, an output command was included for logging/ debugging here.
Custom weighting for Indoor Preference defined here, with following preference options:
Neutral = Standard routing & weighting behavior
Indoor = Preferred indoor routing & custom weighting:
(2 * outdoor segment and 0.5 * indoor segment weight)
Outdoor = Preferred outdoor routing & custom weighting behavior:
(2 * indoor segment and 0.5 * outdoor segment weight)

```
diff --git a/src/main/java/org/opentripplanner/routing/edgetype/StreetEdge.java
b/src/main/java/org/opentripplanner/routing/edgetype/StreetEdge.java
index de92300..fdb3ce9 100644
--- a/src/main/java/org/opentripplanner/routing/edgetype/StreetEdge.java
+++ b/src/main/java/org/opentripplanner/routing/edgetype/StreetEdge.java
@@ -84,6 +84,7 @@ public class StreetEdge extends Edge implements Cloneable {
     private static final int STAIRS_FLAG_INDEX = 4;
     private static final int SLOPEOVERRIDE_FLAG_INDEX = 5;
     private static final int WHEELCHAIR_ACCESSIBLE_FLAG_INDEX = 6;
+    private static final int INDOOR = 7;

     /** back, roundabout, stairs, ... */
     private byte flags;
@@ -137,6 +138,7 @@ public class StreetEdge extends Edge implements Cloneable {
         this.setPermission(permission);
         this.setCarSpeed(DEFAULT_CAR_SPEED);
         this.setWheelchairAccessible(true); // accessible by default
+        this.setIndoor(false); // Outdoor by default
         if (geometry != null) {
             try {
                 for (Coordinate c : geometry.getCoordinates()) {
@@ -352,6 +354,24 @@ public class StreetEdge extends Edge implements Cloneable {
             }
         }

+        if (options.indoor == 1) {
+            if (isIndoor()) {
+                weight *= 0.5;
+                System.out.println("Wanted indoor, WAS indoor. new weight: "+weight);
+            } else {
```

```
+                weight *= 2;
+                System.out.println("Wanted indoor, wasn't indoor. new weight: 
"+weight);
+            }
+        } else if (options.indoor == 2) {
+            if (isIndoor()) {
+                weight *= 2;
+                System.out.println("Didn't want indoor, WAS indoor. new weight: 
"+weight);
+            } else {
+                weight *= 0.5;
+                System.out.println("Didn't want indoor, wasn't indoor. new weight: 
"+weight);
+            }
+        }
+
         if (isStairs()) {
             weight *= options.stairsReluctance;
         } else {
@@ -619,6 +639,14 @@ public class StreetEdge extends Edge implements Cloneable {
         flags = BitSetUtils.set(flags, WHEELCHAIR_ACCESSIBLE_FLAG_INDEX, 
wheelchairAccessible);
       }

+      public boolean isIndoor() {
+             return BitSetUtils.get(flags, INDOOR);
+      }
+
+      public void setIndoor(boolean indoor) {
+        flags = BitSetUtils.set(flags, INDOOR, indoor);
+      }
+
      public StreetTraversalPermission getPermission() {
             return permission;
      }
@@ -769,6 +797,7 @@ public class StreetEdge extends Edge implements Cloneable {
             e.setStairs(isStairs());
             e.setWheelchairAccessible(isWheelchairAccessible());
             e.setBack(isBack());
+            e.setIndoor(isIndoor());
         }

         return new P2<StreetEdge>(e1, e2);
```

Addition of indoor flags for edges in the Transfer Edge java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/edgetype/TransferEdge.java 
b/src/main/java/org/opentripplanner/routing/edgetype/TransferEdge.java
index e629824..287750c 100644
--- a/src/main/java/org/opentripplanner/routing/edgetype/TransferEdge.java
+++ b/src/main/java/org/opentripplanner/routing/edgetype/TransferEdge.java
@@ -37,6 +37,8 @@ public class TransferEdge extends Edge {

     private boolean wheelchairAccessible = true;
```

```
+    private boolean indoor = false;
+
     /**
      * @see Transfer(Vertex, Vertex, double, int)
      */
@@ -104,6 +106,10 @@ public class TransferEdge extends Edge {
         this.wheelchairAccessible = wheelchairAccessible;
     }

+    public void setIndoor(boolean indoor) {
+        this.indoor = indoor;
+    }
+
     public boolean isWheelchairAccessible() {
         return wheelchairAccessible;
     }
```

Addition of indoor support for edges in the edge loader Link Request java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/edgetype/loader/LinkRequest.java
b/src/main/java/org/opentripplanner/routing/edgetype/loader/LinkRequest.java
index 679dc12..3afeee6 100644
--- a/src/main/java/org/opentripplanner/routing/edgetype/loader/LinkRequest.java
+++ b/src/main/java/org/opentripplanner/routing/edgetype/loader/LinkRequest.java
@@ -294,6 +294,8 @@ public class LinkRequest {
             backward2.setStairs(e2.isStairs());
             backward1.setWheelchairAccessible(e2.isWheelchairAccessible());
             backward2.setWheelchairAccessible(e2.isWheelchairAccessible());
+            backward1.setIndoor(e2.isIndoor());
+            backward2.setIndoor(e2.isIndoor());
             addEdges(backward1, backward2);
         }

@@ -311,6 +313,8 @@ public class LinkRequest {
         forward2.setStairs(e1.isStairs());
         forward1.setWheelchairAccessible(e1.isWheelchairAccessible());
         forward2.setWheelchairAccessible(e1.isWheelchairAccessible());
+        forward1.setIndoor(e1.isIndoor());
+        forward2.setIndoor(e1.isIndoor());

         // swap the new split edge into the replacements list, and remove the old ones
         ListIterator<P2<StreetEdge>> it = replacement.listIterator();
```

Addition of indoor support for Street edge vertices in the Street Vertex Index Service Implementation java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git
a/src/main/java/org/opentripplanner/routing/impl/StreetVertexIndexServiceImpl.java
b/src/main/java/org/opentripplanner/routing/impl/StreetVertexIndexServiceImpl.java
index da0bb66..882a1cf 100644
--- a/src/main/java/org/opentripplanner/routing/impl/StreetVertexIndexServiceImpl.java
```

```
+++ b/src/main/java/org/opentripplanner/routing/impl/StreetVertexIndexServiceImpl.java
@@ -125,6 +125,7 @@ public class StreetVertexIndexServiceImpl implements
StreetVertexIndexService {
     public static TemporaryStreetLocation createTemporaryStreetLocation(Graph graph,
String label,
             I18NString name, Iterable<StreetEdge> edges, Coordinate nearestPoint,
boolean endVertex) {
         boolean wheelchairAccessible = false;
+        boolean indoor = false;

         TemporaryStreetLocation location = new TemporaryStreetLocation(label,
nearestPoint, name,
                 endVertex);
@@ -132,6 +133,7 @@ public class StreetVertexIndexServiceImpl implements
StreetVertexIndexService {
             Vertex fromv = street.getFromVertex();
             Vertex tov = street.getToVertex();
             wheelchairAccessible |= ((StreetEdge) street).isWheelchairAccessible();
+            indoor |= ((StreetEdge) street).isIndoor();
             /* forward edges and vertices */
             Vertex edgeLocation;
             if (SphericalDistanceLibrary.distance(nearestPoint, fromv.getCoordinate())
< 1) {
@@ -162,6 +164,7 @@ public class StreetVertexIndexServiceImpl implements
StreetVertexIndexService {
             }
         }
         location.setWheelchairAccessible(wheelchairAccessible);
+        location.setIndoor(indoor);
         return location;

     }
```

Addition of indoor support for Street edge vertices in the Street Location java source code of the OTP Routing Engine. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/routing/location/StreetLocation.java
b/src/main/java/org/opentripplanner/routing/location/StreetLocation.java
index dd82c6f..c29e6c8 100644
--- a/src/main/java/org/opentripplanner/routing/location/StreetLocation.java
+++ b/src/main/java/org/opentripplanner/routing/location/StreetLocation.java
@@ -24,6 +24,7 @@ import org.opentripplanner.util.NonLocalizedString;
  */
 public class StreetLocation extends StreetVertex {
     private boolean wheelchairAccessible;
+    private boolean indoor;

     // maybe name should just be pulled from street being split
     public StreetLocation(String id, Coordinate nearestPoint, I18NString name) {
@@ -47,6 +48,14 @@ public class StreetLocation extends StreetVertex {
         return wheelchairAccessible;
     }

+    public void setIndoor(boolean indoor) {
+        this.indoor = indoor;
+    }
+
+    public boolean isIndoor() {
+        return indoor;
+    }
+
     public boolean equals(Object o) {
         if (o instanceof StreetLocation) {
             StreetLocation other = (StreetLocation) o;
```

Addition of indoor support for OTP Routing Requests in the Otps Routing Request java source code of the OTP API. (Based on Wheelchair code)

```
diff --git a/src/main/java/org/opentripplanner/scripting/api/OtpsRoutingRequest.java
b/src/main/java/org/opentripplanner/scripting/api/OtpsRoutingRequest.java
index fb7561e..5d2763a 100644
--- a/src/main/java/org/opentripplanner/scripting/api/OtpsRoutingRequest.java
+++ b/src/main/java/org/opentripplanner/scripting/api/OtpsRoutingRequest.java
@@ -95,6 +95,10 @@ public class OtpsRoutingRequest {
         req.wheelchairAccessible = wheelchairAccessible;
     }

+    public void setIndoor(int indoor) {
+        req.indoor = indoor;
+    }
+
     public void setClampInitialWait(long clampInitialWait) {
         req.clampInitialWait = clampInitialWait;
     }
```

Addition of indoor tagged features to the list of indexable elements in Nominatim. Done via modification of OSM2PGSQL tool’s Output Gazeteer code (Based on POI code):

osm2pgsql/output-gazetteer.cpp – Lines 196-207

```
                   item->key == "leisure" ||
                   item->key == "office" ||
                   item->key == "shop" ||
                   item->key == "tunnel" ||
+                  item->key == "room" ||
+                  item->key == "building:part" ||
+                  item->key == "buildingpart" ||
                   item->key == "mountain_pass") {
            if (item->value != "no")
            {
                places.push_back(*item);
            }
```

Appendix III – IndoorGML v1 OGC reference Class, Function, Usage Codelists (Omniclass and CityGML)

The current Appendix contains all the Omniclass and CityGML based codelists that are used for the definition of Class, Function and Usage in the IndoorGML v1 OGC reference **(OGC, 2014)**. As our conflation model uses a single tag to represent the *Space types, the codelist values must be taken into consideration after reviewing the "IndoorGML:Navi" tag assigned to the feature.

GeneralSpace

GeneralSpaceClassType			
Code list derived from OmniClass Table 13 and CityGML			
1000	Administration	1060	Laboratory
1010	Business, trade	1070	Service
1020	Education, training	1080	Production
1030	Recreation	1090	Storage
1040	Art, performance	1100	Security
1050	Healthcare	1110	Accommodation, Waste management

GeneralSpaceFunctionType			
Code list derived from OmniClass Table 13			
1000	Elevator Machine Room	2550	Lecture Classroom
1010	Fire Command Center	2560	Lecture Hall
1020	Men's Restroom	2570	Seminar Room
1030	Unisex Restroom	2580	Astronomy Teaching Laboratory
1040	Refrigerant Machinery Room	2590	Research/non-class Class Laboratory
1050	Incinerator Room	2600	Training Space
1060	Gas Room	2610	Woodshop/Metalshop
1070	Liquid Use, Dispensing and Mixing Room	2620	Religious Education Space
1080	Electrical Room	2630	Study Service
1090	Telecommunications Room	2640	Basketball Courts
1100	Hazardous Waste Storage	2650	Team Athletic Recreation Spaces
1110	Building Manager Office	2660	Volleyball Court
1120	Guard Stations	2670	Boxing Ring
1130	Women's Restroom	2680	Circuit Training Course Area
1140	Furnace Room	2690	Aerobic Studio
1150	Fuel Room	2700	Swimming Pool
1160	Liquid Storage Room	2710	Firing Range
1170	Hydrogen Cutoff Room	2720	Hobby and Craft Center
1180	Switch Room	2730	Exercise Room
1190	Classrooms	2740	Skating Rink
1200	Assembly Hall	2750	Climbing Wall
1210	Physics Teaching Laboratory	2760	Diving Tank
1220	Open Class Laboratory	2770	Game Room

1230	Laboratory Service Space	2780	Fitness Center
1240	Computer Lab	2790	Weight Room
1250	Training Support Space	2800	Courtroom
1260	Study Room	2810	Jury Room
1270	Evidence Room	2820	Jury Assembly Space
1280	Witness Stand	2830	Judge’s Chambers
1290	Robing Area	2840	Hearing Room
1300	Council Chambers	2850	Legislative Hearing Room
1310	Armory	2860	Acting Stage
1320	General Performance Spaces	2870	Performance Rehearsal Space
1330	Orchestra Pit	2880	Banding Training Space
1340	Performance Hall	2890	Pre-Function Lobby
1350	Audience Space	2900	Supporting Performance Space
1360	Audience Seating Space	2910	Catwalk
1370	Projection Booth	2920	Motion Picture Screen Space
1380	Stage Wings	2930	Exhibit Gallery
1390	Art Gallery	2940	Display Space
1400	Sculpture Garden	2950	Artist’s Studio
1410	Recording Studio	2960	Media Production
1420	Photo Lab	2970	Museum Gallery
1430	Library	2980	Baptistery
1440	Mediation Chapel	2990	Cathedra
1450	Reflection Space	3000	Clean Room
1460	Chapel	3010	Data Center
1470	Shrine	3020	Computer Server Room
1480	Confessional Space	3030	Exam Room
1490	Tabernacle	3040	General Examination Space
1500	Choir Loft	3050	Labor, Delivery, Recovery, Postpartum Room
1510	Marriage Sanctuary	3060	Newborn Nursery
1520	Mental Health Quiet Room	3070	Patient Room
1530	Bone Densitometry Room	3080	Clean Supply Room
1540	CT Simulator Room	3090	Consultation Room
1550	Head Radiographic Room	3100	Equipment Storage Room
1560	Mobile Imaging System Alcove	3110	Nurse Workspace
1570	MRI System Component Room	3120	Nurse Triage Space
1580	PET/CT Simulator Room	3130	Mental Health Multipurpose room w/Control Room
1590	Radiographic Room	3140	Holding Room, Secured
1600	Stereotactic Mammography Room	3150	Anteroom
1610	Ultrasound/Optical Coherence Tomography Room	3160	Medical Information Computer System Room
1620	Angiographic Control Room	3170	Nursery Transport Unit Alcove
1630	Angiographic Procedure Control Area	3180	Clean Linen Storage Room
1640	Silver Collection Area	3190	Clean Utility Room
1650	Computer Image Processing Area	3200	Mental Health Interview/Counseling Room
1660	CT Control Area	3210	Medical Records Storage room
1670	Image Quality Control Room	3220	Nurse Station

1680	X-Ray, Plane Film Storage Space	3230	Soiled Utility Room
1690	MRI Control Room	3240	Resuscitation Cart Alcove
1700	MRI Viewing Room	3250	Angiographic Procedure Room
1710	Radiographic Control Room	3260	CT Scanning Room
1720	Tele-Radiology/Tele-Medicine Room	3270	Cystoscopic Radiology Room
1730	Radiation Diagnostic and Therapy Spaces	3280	Mammography Room
1740	Health Physics Laboratory	3290	MRI Scanning Room
1750	Linear Accelerator Entrance Maze, Healthcare	3300	PET/CT Scanning Room
1760	Radioactive Waste Storage Room, Healthcare	3310	Radiographic Chest Room
1770	Nuclear Medicine Scanning Room	3320	Radiology Computer Systems Room
1780	Patient Dose/Thyroid Uptake Room	3330	Ultrasound Room
1790	Radiopharmacy	3340	Whole Body Scanning Room
1800	Radiation Therapy, Mold Fabrication Shop	3350	Angiographic Instrument Room
1810	Hearth and Lung Diagnostic and Treatment Spaces	3360	Angiographic System Component Room
1820	Cardiac Catheter Instrument Room	3370	Computed Radiology Reader Area
1830	Cardiac Catheter Control Room	3380	X-Ray, Digital Image Storage Space
1840	Cardiac Electrophysiology Room	3390	CT power and Equipment Room
1850	Echocardiograph Room	3400	Image Reading Room
1860	Extended Pulmonary Function Testing Laboratory	3410	Mammography Processing Room
1870	Pacemaker ICD Interrogation Room	3420	MRI Equipment Storage Room
1880	Procedure Viewing Area	3430	PET/CT Control Room
1890	Pulmonary Function Treadmill Room	3440	Radiographic Darkroom
1900	Respiratory Therapy Clean-up Room	3450	Viewing/Consultation Room, Diagnostic Imaging
1910	General Diagnostic Procedure and Treatment Spaces	3460	Equipment Calibration Space, Radiation Diagnostic and Therapy
1920	Endoscopy/Gastroenterology Spaces	3470	Linear Accelerator Component Room, Healthcare
1930	Clinical Laboratory Spaces	3480	Linear Accelerator Room, Healthcare
1940	Pharmacy Spaces	3490	Nuclear Medicine Dose Calibration Space
1950	Rehabilitation Spaces	3500	Nuclear Medicine Patient “Hot” Waiting Room
1960	Medical Research and Development Spaces	3510	Radiation Dosimetry Planning Room
1970	Chemistry Laboratories	3520	Radium Cart Holding Space
1980	Physical Sciences Laboratories	3530	Sealed Source Room
1990	Earth and Environmental Sciences Laboratories	3540	Brachytherapy Room
2000	Psychology Laboratories	3550	Cardiac Catheter System Component Room
2010	Dry Laboratories	3560	Cardiac Catheter Laboratory

2020	Wet Laboratories	3570	Cardiac Testing Room
2030	Biosciences Laboratories	3580	EKG Testing Room
2040	Astronomy Laboratories	3590	Microvascular Laboratory
2050	Forensics Laboratories	3600	Pacemaker/Holter Monitor Room
2060	Bench Laboratories	3610	Pulmonary Function Testing Laboratory
2070	Integration Laboratories	3620	Pulmonary Screening Room
2080	Laboratory Storage Spaces	3630	Respiratory Inhalation Cubicle
2090	Office Spaces	3640	Eye and Ear Healthcare Spaces
2100	Dedicated Enclosed Workstation	3650	Surgical Spaces
2110	Open Team Setting	3660	Clinical Laboratory Support Spaces
2120	Shared Equipment Station	3670	Medical Services Logistic Spaces
2130	Banking Spaces	3680	Dental Spaces
2140	Automatic Teller Machine Space	3690	Press Conference Room
2150	Trading Spaces	3700	War Room
2160	Demonstration Spaces	3710	Waiting Space
2170	Checkout Space	3720	Waiting Room
2180	Fitting Space	3730	Office Service
2190	Auction Room	3740	Shared Open Workstation
2200	Commercial Service and Repair Spaces	3750	General File and Storage
2210	Hotel, Motel, Hostel, and Dormitory Service Spaces	3760	Lookout Gallery
2220	Hotel Residence Room	3770	Bank Teller Space
2230	Commercial Support Spaces	3780	Vault
2240	Dormitory	3790	Trading Floor
2250	Information Counter	3800	Sales Spaces
2260	Post Office Space	3810	Display Space
2270	Mail Room Space	3820	Vending Machine Area
2280	Conference Room	3830	Pet Shop Animal Space
2290	Grooming Activity Spaces	3840	Makeup Space
2300	Haircutting Space	3850	Food Service
2310	Cooking Spaces	3860	Kitchen Space
2320	Food Preparation Space	3870	Cooking Space
2330	Dishwashing Station	3880	Dining and Drinking Spaces
2340	Dining Room	3890	Banquet Hall
2350	Food Court	3900	Snack Bar
2360	Salad Bar	3910	Liquor Bar
2370	Beverage Station	3920	Table Bussing Station
2380	Serving Station	3930	Vending Perishable Product Space
2390	Cafeteria Vending Space	3940	Tray Return Space
2400	Food Discard Station	3950	Coffee stations
2410	Child Care Spaces	3960	Daycare sickroom
2420	Child Day Care Space	3970	Play Room
2430	CLD–Child Care	3980	Resting Spaces
2440	Rest Area	3990	Break Room
2450	Laundry/Dry Cleaning Space	4000	Smoking Space
2460	Locker Room	4010	Filing Space
2470	Supply Room	4020	Unit Storage

2480	On-call Room	4030	Bathroom
2490	Shower Space	4040	Toilet Space
2500	Ablution Room	4050	Combination Toilet and Bathing
2510	Mud Room	4060	Laundry Room
2520	Bedroom	4070	Mental Health Resident Bedroom
2530	Mental Health Resident Bedroom,	4080	Nursery
2540	Kitchen		

GeneralSpaceUsageType
Code list identically specified as GeneralSpaceFuntionType

TransitionSpace

TransitionlSpaceClassType			
Code list derived from OmniClass Table 13			
1000	Horizontal Transition	1010	Vertical Transition

TransitionlSpaceFuntionType			
Code list derived from OmniClass Table 13			
1000	Corridor	1070	Concourse
1010	Breezeway	1080	Moving walkway
1020	Box Lobby	1090	Entry Lobby
1030	Elevator Lobby	1100	Jet way
1040	Landing	1110	Elevator Shaft
1050	Aisle	1120	Stair
1060	Ramp	1130	Chute

TransitionSpaceUsageType
Code list identically specified as TransitionSpaceFuntionType

ConnectionSpace

ConnectionSpaceFunctionType			
Code list derived from OmniClass Table 13			
1000	Door	1060	Sally port
1010	Vestbule	1070	

ConnectionSpaceClassType			
Code list derived from OmniClass Table 13			
1000	Door	1020	Sally port
1010	Vestbule		

ConnectionSpaceUsageType
Code list identically specified as ConnectionSpaceFunctionType

AnchorSpace

AnchorSpaceClassType			
Code list derived from OmniClass Table 13			
1000	Vestibule	1020	Gate

AnchorSpaceFuntionType			
Code list derived from OmniClass Table 13			
1000	Entry Vestibule	1020	Gate
1010	Exterior door	1030	Emergency door

AnchorSpaceUsageType
Code list identically specified as AnchorSpaceUsageType