Employing existing standard BIM formats as the source of building information in Indoor Positioning

Dissertation

For acquiring the title

Doctor technicae (Dr. techn.)

Submitted by

Sergej Muhič

Graz University of Technology

Faculty of Civil Engineering

Institute of Structural Analysis

Supervisors

Univ.-Prof. Dr.techn. Dipl.-Bauing. Ulrich Walder

Prof. Dr. Dipl. -Ing. Danijel Rebolj

Assessor

prof. dr. Žiga Turk

Graz, 2015

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, …… ……..……

(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz …… ……..

(signature)

Kurzfassung

Die Urbanisierung der Welt hält unvermindert an, weshalb wir unseren Alltag immer mehr in Gebäuden verbringen. Daraus hat sich der Bedarf nach einem robusten Indoor Positionierungssystem entwickelt. Insbesondere für First Responder ist das Wissen, wo man sich in einem Einsatz genau befindet zentral; aber auch für Sehbehinderte und Blinde ist die Orientierung in einer immer komplexeren Umwelt zunehmend schwierig.

Weltweit sind eine Vielzahl von Forschungseinrichtungen daran, mit ganz unterschiedlichen Technologien Lösungen zu entwickeln. An der TU Graz wurden unter der Leitung von Prof. Walder am Institut für Bauinformatik mit den Projekten CADMS und AIONAV erste Schritte in Richtung einer praktischen Implementation gemacht. Ein autonomes System auf Basis der Inertialtechnologie wurde entwickelt und im praktischen Einsatz getestet. Es gilt als eines der weltweit genausten „foot-mounted inertial systems“. Die systembedingte Drift kann mit einem erweiterten Kalman Filter nur bedingt stabilisiert werden, weshalb als weitere Korrekturmöglichkeit die Interaktion mit einem Gebäudemodell entwickelt wurde. Die Entwicklung der Gebäudeinteraktion im Projekt CADMS hat den Hauptimpuls für die Forschung über die Gewinnung von den Gebäudedaten aus vorhandenen Quellen geliefert. Das Gebäudemodellformat IPS XML (Indoor Positioning XML), das im Rahmen von CADMS entwickelt wurde, wird dabei als Referenz und Zielformat verwendet und definiert die Anforderungen, für die Gebäudedatenextraktion aus anderen Gebäudeinformationssystemen (CAD, BIM, CAFM).

Auch in anderen wissenschaftlichen Projekten hat sich der Bedarf nach einer allgemeinen, automatisierten Schnittstelle für Gebäudedaten gezeigt, da die Vorgehen für das Erzeugen von Gebäudedaten bislang stets noch mit einem enormen, manuellen Aufwand verbunden sind. Aus diesem Grund wurden zuerst einmal existierende aktuelle Dateiformatstandards für Gebäudedaten evaluiert und verglichen. Als Beispiele verschiedener Vorgehensweisen wurden zwei Formate näher untersucht, die in der Praxis schon eine weitere Verbreitung in der Gebäudeplanung gefunden haben: das an die Autodesk BIM Software Revit gebundene Format (RVT) und der internationale Standard IFC. RVT ist ein

geschlossenes, über eine API zugängliches Format, während IFC als offenes ASCII Format definiert ist.

Beide obengenannten Formate haben sich als geeignet erwiesen. Ein neu entwickelter Algorithmus kann aus beiden Formaten die benötigten geometrischen und semantischen Daten entnehmen und topologische und geometrische Informationen erzeugen, die gebraucht werden, um eine IPS XML Datei zu erzeugen. Die Tatsachen, dass IFC ein offenes Format ist, sowie seine hohe Verfügbarkeit und Flexibilität, dürften dazu führen, dass es sich als Datenquelle für IPS in der Praxis durchsetzt. Die weitere Untersuchung von IFC hat die Möglichkeit einer endgültigen Lösung für ein allgemeines IPS Format mit der Definition einer MVD (Model View Definition) für die Indoor Positionierung aufgezeigt.

Schlüsselwörter: BIM, building information modelling, semantisches Gebäudemodell, IPS, indoor positioning, IFC, MVD, Topologie

Abstract

The rapid urbanization of the world's population is the main reason that everyday life is increasingly taking place in buildings. Therefore, the need for a robust indoor positioning system is becoming apparent. Notably, for first responders the current position in field operations is of critical importance; as well as for the blind and visually impaired the orientation in a complex environment.

Worldwide the issue has become the priority of many research projects, and the solution is being addressed using various technologies. In the scope of the research projects CADMS and the subsequent AIONAV from the Graz University of Technology Institute of Building Informatics under the leadership of Prof. Ulrich Walder, the first steps towards real life implementation were taken. An autonomous system based on inertial technology was developed and tested. However, the system-induced drift can be addressed only to a certain extent with an extended Kalman filter. Therefore, additional methods of position correction are required and a building model interaction was developed. This method from the CADMS project has provided the main incentive for our research into exploiting existing sources of building information for this purpose. The underlying indoor positioning building model format (IPS XML) developed for CADMS also serves as the reference for basic requirements and as the destination format for building data extraction from building information systems like CAD, BIM or CAFM.

Other research projects that take advantage of Indoor Positioning have been analysed, and the demand for a common source for building data would be beneficial, since even now they rely predominantly on the manual generation of building models. Therefore, several modern file format standards that store building information for different purposes were evaluated and compared. As an example of two different approaches, the already established file formats of a native application specific file format from the Autodesk building modelling application Revit (RVT format) and the international standard IFC, were chosen as the most suitable. These formats represent different types of data storage and availability. Whereas the RVT format is a closed format accessible through an API, the IFC format is an open ASCII format file.

Both of the aforementioned formats have proven suitable. We managed to extract building data and create the building information required for IPS XML from each of them with a novel algorithm that creates topological information from the geometric and semantic data provided by the building models. The fact the IFC is an open format as well as its availability and flexibility may lead to the general acceptance of the format as the main source of building data for IPS. The further study of IFC has also offered a possible final solution for a common IPS format through the exploitation of model view definitions.

Keywords: BIM, building information modelling, semantic building model, IPS, indoor positioning, IFC, MVD, topology

Acknowledgments

After accomplishing the undertaking of writing my Dissertation and reflecting upon the process, it is clear that it could not have come into existence if I had had to face the task alone. Now the pleasant time has come when I can appreciate the most important contributors. Working at the Institute of Building Informatics of the Graz University of Technology was both a pleasure and a privilege that secured the conditions in which I was even able to think about academic endeavours.

The institution alone is, of course, nothing without people, and the first person I have to mention is my main supervisor and employer who saw the potential in me even as we first met, the head of the Institute of Building Informatics Prof. Dr. techn. Ulrich Walder. With his specific style of leadership and view of academic institutions, the basis for critical observation is formed. My second supervisor, prof. dr. Danijel Rebolj at the time Rector of the University of Maribor, deserves the same level of attention. Despite his function as rector taking much of his time, his interest in the successful and meritable completion of the dissertation enabled me to overcome one of the most difficult obstacles I had to face after I began writing. I would also like to acknowledge my third assessor prof. dr. Žiga Turk, for his responsiveness and willingness to deal with yet another of my theses.

The last year I have spent at the Graz University of Technology was signified by my switchover to the Institute of Structural Analysis. For such a big change being hardly stressful I have to thank Prof. Dr Thomas-Peter Fries. The fact that I was accepted into the team and got the chance to contribute as well as his genuine interest in my work and wellbeing at work were instrumental in what I consider a very if not the most effective year at the TUG.

My colleagues at the Institute of Building Informatics were naturally second to none. I would especially like to thank Martin for all the constructive discussions and criticism, and Thomas for his expert help and support when I needed it the most. However, I will mostly remember the times away from work, the breaks and merriment.

The people who had to suffer the most during my endeavours are the ones closest to me. This is my family, my bright beacon that constantly reminded me of what is important in life throughout the process. My wife Veronika had to put up with all the

rage outbursts and always seemed to find the right words so that I kept pushing on. I will not even start on her valued opinion I got to rely on. My three children, who were all born during the height of my efforts, will finally get their father back. I have to admit it was not easy, but despite draining my energy more often than not, though unexplainable by physical laws, they always gave me more in return that made me continue with even more enthusiasm and vigour.

Last but certainly not least, I would like to thank my parents for the obvious reasons of supporting me (most of the time) and giving me the gift of life. I was never an easy and soothing individual; puberty comes to mind. My brother deserves special mention for keeping my feet on the ground throughout these years.

Table of Contents

1 Introduction

The importance of buildings in our everyday life is growing as the time we spend inside them keeps increasing and is predicted to increase even more in the future (Pérez-Lombard, et al., 2008). Despite that, the productivity of the AEC industry has been stagnant or even declining slightly over the years due to several reasons; most notably the slow introduction of new business practices and the fragmented introduction of new technologies (Eastman, et al., 2008). Therefore, to improve the efficiency of building construction and management information technologies, these are being adopted in the processes of the architecture, engineering and construction (AEC) Industries at a growing rate, thus leading to the development of Building Information Modelling (BIM). Standing at the core is a digitally constructed virtual model of a building that contains precise geometry and relevant data for the support of construction, fabrication and procurement activities during building realization (Eastman, et al., 2008). Additionally, BIM incorporates many of the functions needed to model the lifecycle of the building. Recently, after the development of functional BIM related tools and their implementation in real life construction projects, a number of return on investment (ROI) studies have proven a rise in efficiency and quality of the construction process and a high return of investment (Kelly, 2015; Giel & Issa, 2011). Following good results from early adopters, a wider acceptance of these tools and related processes was established in the AEC industries resulting in the quality and quantity of semantic building models' availability to increase substantially. Challenges of implementing an integrated workflow are becoming apparent now more than ever. One of them is the need for regulations that define the workflow and responsibilities each related discipline such as architects, contractors or the MEP (mechanical, electrical, plumbing) sector should have. The general lack of such regulations; with exceptions in countries like the USA, Nordic countries, Singapore etc. (Khemlani, 2012) or the UK (Waterhouse, et al., 2015); makes it difficult to implement these technologies to their fullest. Compared to e. g. the automotive industry, where the entire production process of a product is concentrated in one company standardization, the AEC industry is more complicated since it requires the coordination of several different agents who more often than not use different data formats. Additionally, data ownership needs to be determined. A solution lies in an integrated approach where in general architects, engineers, construction managers and contractors work together in

either fully integrated firms or multifirm partnerships (Elvin, 2007). As suggested by Matthews and Howell (2005) with Integrated Project Delivery (IPD) risks and profits are shared between primary team members (PTM). Primary team members consist of the architect, key technical consultants, a general contractor and key subcontractors. Elvin (2007) goes a step further and suggests that the IPD team offer expanded building related services to their clients through the entire life cycle of the building relying on a central data storage location enabled by building information modelling.

As we have previously established (Muhič & Krammer, 2014), drawing from this central data storage the generated and stored building data can benefit other areas of expertise. The first responders in emergency situations (Rueppel & Stuebbe, 2008), (Li, et al., 2014) or (Walder, 2006), location based services in public buildings (Krammer, et al., 2012), escape plans and simulations of crowd dynamics (pedestrian crowd movement, evacuation dynamics) (Kneidl, et al., 2012), blind and visually impaired navigation aid (Hub, et al., 2004) etc. can and should make use of a centralized building model. After all, this type of model provides the best quality and up-to-date information of the state of the building at any given point in its life cycle. A common denominator for all these potential uses is that only a small portion of building data from a complete building information model is actually required. Another similarity is that building data in all of these cases serves at least partly as a positioning, localization and navigation aid. Therefore, the first obstacle to overcome is identifying and specifying which data is required from the central data model. This is mostly information about geometry and topology since building data is supposed to provide a basis for determining the location of users. The building model acts as a map that enables indoor positioning. Once the required data has been specified the next task is to extract the specified data from the Building Information Model and convert it into useful information. This can be done in various ways and also depends on the data sources available. The data can be extracted directly from the data model or already stored beforehand in a format that allows direct access. The various data formats that can serve as a basis range from complete building model representations like IFC to specialized formats like gbXML (Green Building XML). Most, if not all, current approaches to modelling indoor environments, like the Geometric Topology Network (Taneja, et al., 2011) or the Multi-layered Space-Event model (Becker, et al., 2009), lean on IFC as a source of building data. Current indoor positioning methods, espe-

cially indoor navigation, have inherent sensor shortcomings such as drift in the case of inertial sensors or interference in the case of stationary sensors based on radio waves (Reynolds, 2003) inside buildings, which is why we require these models to put raw positioning data into context and thus create location information that can further serve indoor navigation purposes.

1.1 Problem formulation and analysis

Looking at the present circumstances, indoor positioning remains a relatively new and dynamic field with a range of existing approaches already available. As of yet there is no one perfect approach that would solve all issues related to the field. However, we can break the subject down roughly into these individual problem areas that need to be dealt with:

1. Sensory equipment for position estimation,
2. Position verification and correction,
3. Communication between all devices (network),
4. User interface.

The main differences between contemporary methods of Indoor Positioning can be determined by the initial choice of sensory equipment. A general differentiation between sensors in this respect is that the equipment is either fixed or mobile. Both have their strengths and weaknesses. Fixed installations have the advantage of permanent sensor positions i.e. the location of the sensor is definite and final. Thus, positioning can be accomplished absolutely with reference to these fixed sensors by measuring the distance and the angle from them. With access to more than two sensors only distance measurements are required and the position is determined by trilateration. In the case of required repositioning of the subject due to possible interference, there remains the possibility of falling back on sensor location. The method in use for determining the position of the user is absolute. However, fixed installations are extremely unreliable especially in the case of the use of Indoor Positioning in emergency situations like first responders in the case of disasters. The sensors can either change their positions or even be destroyed due to the external influences of such situations. Additionally, disturbances like smoke, fire and destroyed building elements can hamper position determination from external sources.

This is where autonomous sensors gain their value. Extreme situations can benefit greatly from sensors that are not installed in the building. The lack of fixed installations, however, comes with a downside since no absolute position can be deter-

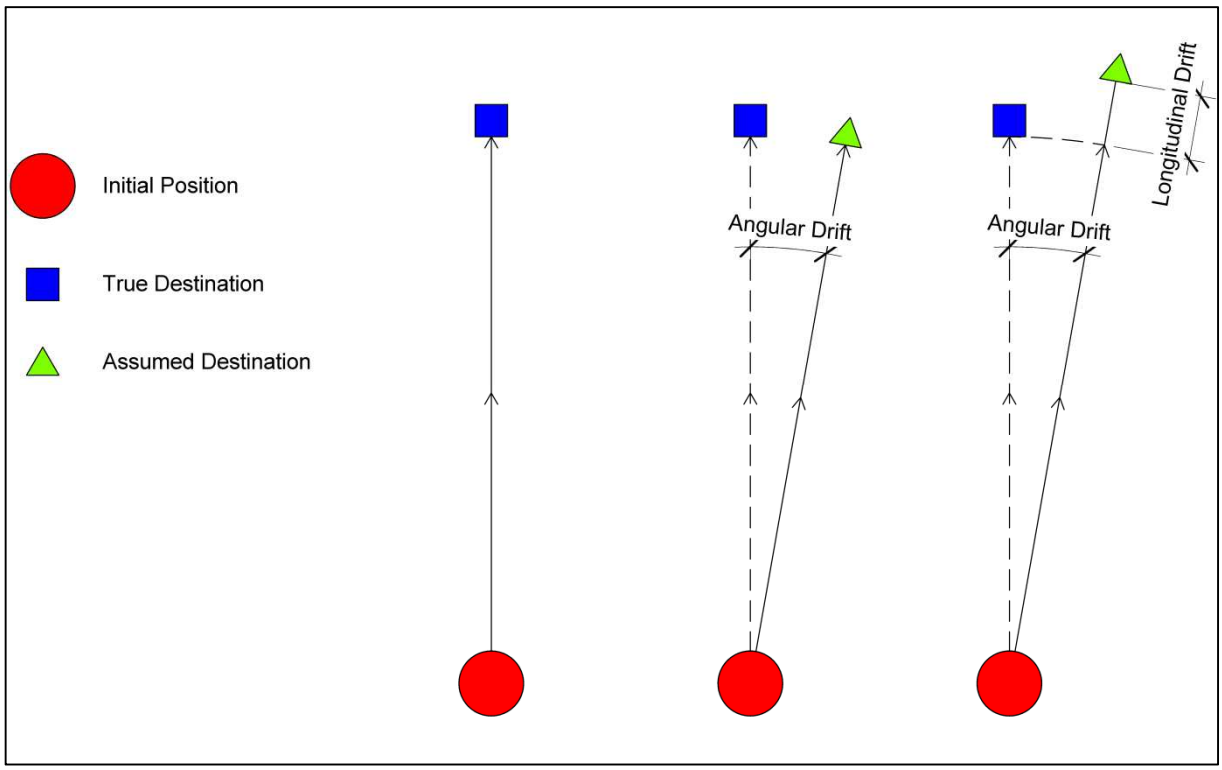


Figure 1: Dead reckoning measurement error

mined. Therefore, a relative positioning method is needed and comes in the form of dead-reckoning (Figure 1) in which the current position is calculated from a known fixed initial position and alignment, the distance travelled, and the bearing calculated from sensor measurements in given time intervals. As is the case with every measurement these measurements produce errors, and because the method is relative to an initial absolute position, these errors always stack (Figure 2) as they are added to the previous ones in every additional step. Therefore, regarding relative methods with autonomous sensors, some kind of repositioning is desired and in most cases even required to attain quality results. Repositioning can be achieved through various means. Most commonly a reference point can be taken from an absolute measurement; another possibility is to take a fixed reference point.

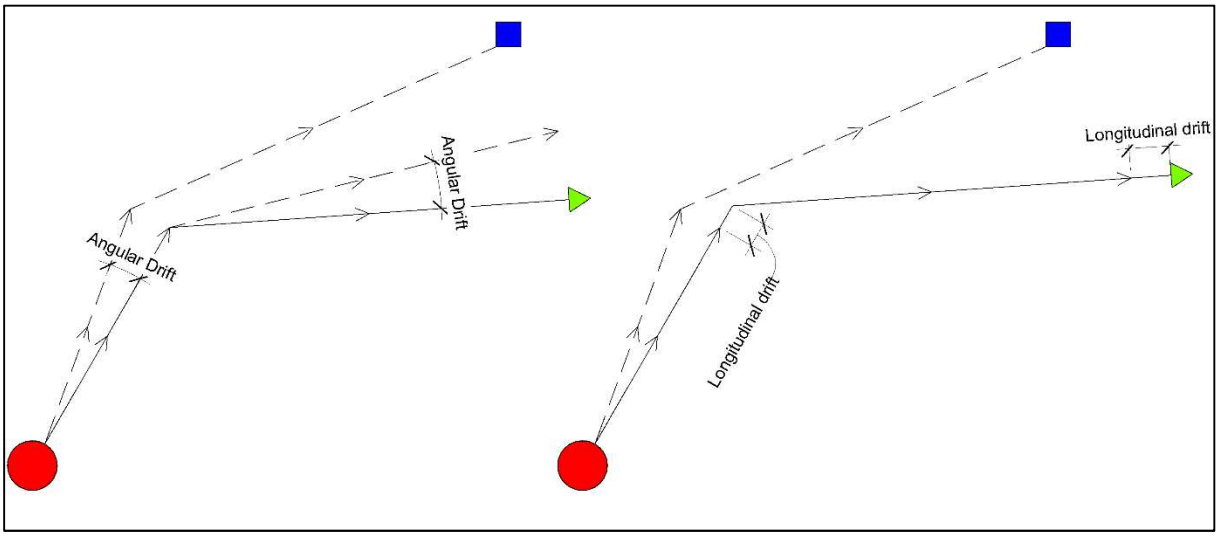


Figure 2: Dead reckoning error propagation over multiple measurements

Lately however, the aid of building information models is becoming essential in the various state of the art methods of indoor positioning, repositioning and navigation. Ranging from emergency response situations (Li, et al., 2014; Walder, 2006), for the aid of the blind and visually impaired (Hub, et al., 2004; Hub, et al., 2006) or pedestrian simulations (Hartmann, 2010; Kneidl, et al., 2012), all of these diverse application fields rely on a building model as a source of fixed positioning references. Moreover, building information models can be further exploited for improving the accuracy of positioning algorithms and the graphical representation and interaction (the user interface) of indoor positioning.

1.2 Hypothesis

With building information models getting established as one of the fundamentals of indoor positioning and thus playing a greater role, the question arises of how to get quality building information models and further of how to utilize or prepare the data from building information models so that it can provide useful information in the process of indoor positioning. There are two fundamentally different ways of how to generate a building information model for indoor positioning (IPS BIM) that can be laid out as follows:

1. Creating an IPS BIM manually from scratch with a modelling tool or,

2. Exploiting existing standard BIM formats and generating an IPS BIM from the available data.

The first option requires a dedicated computer application and can create redundant information when a building information model is already available. This approach can generate certain issues like information not being up to date or making it necessary to record the current state from the building.

The second option makes use of building information models in standard formats that were generated in the process of designing and constructing the building, thus providing the most recent state of the building. This holds especially true in the case of the building model also being used in facility management and, thus being constantly updated with changes during the building's life cycle. In this case building data is not created redundantly making it the preferred option. However, building information models are not structured with Indoor Positioning in mind, hence the data is not yet in the right form to be used directly for Indoor Positioning. This is the issue that will be addressed. First, the state of both types of model has to be established, and through an analysis parallels have to be drawn between the existing building element entities from the building information model and the Indoor Positioning building model. The question that needs to be answered is whether a typical building information model holds the right data in the right form to be utilized directly or if some kind of transformation is necessary. Another issue in this relation is whether building information models in general provide the right data to be transformed and, in the case that they do, whether a standard building information model is suitable or if a specialized Indoor Positioning building information model format should be standardized for common practice. If only some enhancements are needed a standard building information model format might also be a solution.

With this dissertation the effort will be made to prove that building data available from existing building model formats is sufficient to generate a complex indoor positioning building model. Furthermore, I will approach the issue by examining whether the aforementioned questions can be addresses with contemporary BIM technologies. Application programming interfaces (API) of modern BIM modellers offer direct access to the data stored in a building model along with all semantic information. Moreover, the international standard ISO 16739:2013 (IFC – Industry Foundation Classes)

provides an independent open platform source of building data that is readily accessible. By analysing and testing accessible building data formats under strict conditions and the requirements of the Indoor Positioning building model format IPS XML (Muhic, et al., 2012) I will prove that existing building data can and should be used as a source for Indoor Positioning building models.

1.3 Dissertation structure

This dissertation is structured in four chapters corresponding to thematic sections with various subchapters. The first (introductory) chapter offers a general overview of BIM, the availability of building models, and the motivation behind the work, and is followed by three other chapters.

The second chapter deals more closely with the newest findings about the various parts necessary to this research. It is structured into three areas in which each successive area is to some extent dependent on the previous one. First, the various building model formats available as potential data sources are discussed; next, the current methods of indoor positioning are investigated; and the final discussion is about the most important approaches of merging the previous areas, resulting in finding out how building data is used for indoor positioning.

The third chapter provides a detailed overview of the core research accomplished as well as a comparison of the two chosen most relevant data sources; the commercial closed Autodesk Revit file format and the open IFC file format. Initially, the options of how to create an Indoor Positioning Building Model are given (either manually or automatically from different sources) with data extraction from existing formats presenting the main research interest. The chapter concludes with a comparison of the two chosen formats, analysing their strengths and weaknesses with a discussion of the preferred option.

The fourth and last chapter serves as both a conclusion and discussion of the implementation possibilities of the newly acquired data. The preferred open file format IFC offers a variety of choices for implementing this data and the formats currently available are introduced. This chapter also offers an insight into possible future research directions and provides an incentive to the community to start a discussion about the

standardization of the required data for Indoor Positioning in building information model file formats, particularly the IFC file format.

2 Present state and related work

Before focusing on the main thesis, some fundamentals have to be established. Among the technologies used in Indoor Positioning and the diverse sensory equipment available, which define the method of determining a position indoors, the building information model and the IPS BIM are essential and have to be examined first. The obvious reasons for this are that the BIM and IPS BIM serve as the core of the research, and that establishing the method of Indoor Positioning will define all the later requirements for every component involved. Thus, an analysis and introduction of possible BIM formats is required.

With a range of existing approaches to Indoor Positioning already available, a solid basis to start the research already exists. Initially, an overview of various methods to determine a position indoors will be created. The selection of available methods is diverse, ranging from fixed equipment installed in buildings to sensors worn by individuals. Special emphasis will be put on the CADMS project started by Prof. Ulrich Walder (2006) at the Technical University Graz. The Indoor Positioning sub-module of this project is based on an Inertial Measurement Unit (IMU) as the main sensor. The requirements of this project sparked the idea and enabled the research of BIM for use in Indoor Positioning.

Another highlight will be the current definition of BIM, and on that basis the discussion of the various BIM formats available today that could prove suitable candidates to form the data source for the generation of an IPS BIM.

2.1 Semantic building information model

"Building Information Modeling (BIM) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition." (National BIM Standard - United States™, 2014)

Commonly referred to as a Building Information Model or BIM (also stands for Building Information Modelling) this is usually a three dimensional model of a building as-

sembled from building blocks (objects) equipped with semantic information that gives them a certain context in respect to what they represent. Moreover, an individual building block's relationship to other objects is also defined in the building information model. Generally, BIM software can interpret this information and create understanding e. g. a Wall object "knows" how to connect to another Wall object, a Door object "knows" it can exist only inside a Wall object.

"A basic premise of BIM is collaboration by different stakeholders at different phases of the life cycle of a facility to insert, extract, update or modify information in the BIM to support and reflect the roles of that stakeholder." (National BIM Standard - United States™, 2014)

This intelligence and modularity of the building model and the objects that it consists of makes it possible to store the entire lifecycle of the building. Different stakeholders have access to information meaning they can use it and modify it according to their own needs. On the down side, this fact raises the question of the ownership of the building model and data. According to Simonian (2010), several other legal issues must also be addressed:

- Ownership of the BIM and data – the owner of the building does not own the exclusive rights to the building model. The architect, engineers and equipment suppliers have to be considered as well.
- Allocation of risks – responsibility for up-to-date and accurate building data involves a certain degree of risk.
- Privity of contract and third party reliance – the nature of collaborative work on the building model lessens the risk of privity of contract as a legal defence.
- Professional design responsibility – the risk of non-licensed participants' data input due to their access to the building model.
- Standard of care – despite the virtual ownership of the designed building components other parties have access to the data and therefore the responsibility is distributed and the standard of care unravelled.
- Spearin doctrine – collaborative work might deprive contractors of the protection from design errors provided by this doctrine.
- Economic loss rule – a collaboratively designed building model undermines a contractor's claim for recuperation of economic losses due to malpractice.

As we can see, the availability and easy accessibility of data that is stored in the building model has both advantages and drawbacks. Additionally, it should be noted that BIM is not merely software it is a process. Therefore, these advantages and drawbacks should be considered and addressed before implementing BIM so as to take advantage of the process in its fullest and minimize the risks that come along. Accordingly, the requirements for information in the BIM are that it must be interoperable and therefore based on open standards.

"The US National BIM Standard will promote the business requirements that BIM and BIM interchanges are based on:

- *a shared digital representation,*
- *that the information contained in the model be interoperable (i.e.: allow computer to computer exchanges), and*
- *the exchange be based on open standards,*
- *the requirements for exchange must be capable of defining in contract language."* (National BIM Standard - United States™, 2014)

Despite a clear definition, the nature of BIM means that different stakeholders have their own view on what BIM actually is. Depending on the perspective of the beholder BIM can fulfil several different roles and be several different things.

"As a practical matter, BIM represents many things depending on one's perspective:

- *Applied to a project, BIM represents Information management—data contributed to and shared by all project participants. The right information to the right person at the right time.*
- *To project participants, BIM represents an interoperable process for project delivery—defining how individual teams work and how many teams work together to conceive, design, build & operate a facility.*
- *To the design team, BIM represents integrated design—leveraging technology solutions, encouraging creativity, providing more feedback, empowering a team."* (National BIM Standard - United States™, 2014)

Consequently, a number of formats for BIM have been developed depending on particular stakeholder demands, with each fulfilling the demands it was designed for.

Eventually, most of these formats (e.g. CIS/2 for steel) converged into IFC as the current international standard. For the purposes of Indoor Positioning the following formats were analysed in more detail:

- IFC – Industry Foundation Classes is a data format that can store the entire lifecycle of the building.
- CityGML – City Geographic Markup Language is similar to IFC but for 3D landscape and city models.
- gbXML – Green Building Extensible Markup Language is a specialized format for the needs of sustainable design.
- BIM Application specific formats – data can be extracted and modified directly from the various software vendors BIM applications specific data formats.

These formats were chosen because they should contain the necessary data a specialized BIM or view of a BIM for Indoor Positioning will require. As IFC and data formats from various software vendors are designed to contain more or less all data through the entire lifecycle of the building, geometry and topology comprises only a small portion of that. CityGML also features individual buildings, despite being designed for 3D city and landscape models. Lastly, gbXML does not feature general information about the building. It does, however, feature the geometry of spaces and openings, which fits the requirement for geometry and topology.

2.1.1 Industry foundation classes (IFC)

IFC is an international open standard for BIM developed by buildingSMART. It is registered at the International Standard Organisation (ISO) as ISO 16739:2013, a standard for data sharing in the construction and facility management industries (ISO, 2013). It specifies a conceptual data schema represented as an EXPRESS schema specification and an exchange format for Building Information Model (BIM) data.

The common data schema that was developed should make it possible to exchange data between different proprietary software applications, eliminating the need for the software to support numerous native formats. However, as proven in practice this does have its limitations. Software applications developed by different vendors have

an independent manner of storing building data, which in turn translates into the IFC files generated by these applications. The data schema consists of information about the building covering its entire lifecycle: *"from conception, through design, construction and operation to refurbishment or demolition."* (buildingSMART, 2008)

Several versions of IFC exist, since it is constantly being developed further. The newest version is IFC4, which was released in March 2013. Each new version is accompanied by new certification for software applications, which must be fulfilled to be able to support the new version and must display a buildingSMART logo, which determines the ability of the application to work with the specific IFC version. (buildingSMART, 2014)

The IFC data schema comes in two representations, the previously mentioned EXPRESS schema and an ifcXML XSD schema. The ifcXML file size is roughly 300-400% times larger than the EXPRESS schema IFC file size. There is one additional format, the ifcZIP, which is merely a compressed version of the two formerly mentioned schemas. Because both EXPRESS and ifcXML are ASCII formats ifcZIP is very efficient at compressing them; .ifc by 60-80% and .ifcXML even by 90-95%. (buildingSMART)

2.1.1.1 IFC version history

Before explaining the official start of IFC development, I should mention STEP (Standard for the Exchange of Product Model Data) because it laid the basis for the development of IFC. The purpose of STEP was to create an open computer modelling standard for multiple industrial and manufacturing industries; one of them being AEC/FM. Along with STEP for data models, an information modelling language called EXPRESS was developed, which can define a data model in two ways, textually and graphically. The graphical representation is called EXPRESS-G (ISO, 2004). The initial development resulted in STEP becoming an international standard. Further development in the field of AEC/FM defined the Building Construction Core Model (BCCM) as the central data model for AEC/FM. The five proposed pillars for an open building product model, as proposed by (Björk & Penttilä, 1989) or similarly defined by (Eastman, et al., 1991), are as follows:

1. Encompass all building information,

2. Meet the information needs of all stakeholders,
3. Be non-redundant,
4. Be software independent,
5. Be format independent.

Despite being abstract, these principals were later integrated in the IFC standard as was the EXPRESS modelling language.

Having a universal product model standard across all industries is a tempting concept enabling cross industry collaboration and eliminating redundant standardization work (Laakso & Kiviniemi, 2013). Nevertheless, the AEC/FM industry considered the process too slow and unresponsive and thus began the initiative of 12 US companies to develop an open standard that would enable interoperability of building information modelling software in development. The companies founded IAI (Industry Alliance for Interoperability, which in 1995 changed to International Alliance for Interoperability), which was responsible for the development of IFC.

The first IFC version, IFC1.0, was released back in 1997 (see Figure 3) by the IAI with formal development starting as early as 1995. This first release covered mainly the architectural part of the BIM evident from the structure, and featured five processes for architecture, two for HVAC design, two for construction management and one for facility management. It was only used as a prototype and was followed by the version IFC1.5, also released in the year 1997. The next version, IFC1.5.1, fixed some problems with the model implementation of the previous version in 1998 and became the first version that was used in commercial BIM software.

Along with the first international release, IFC2.0 in 1999, these versions are outdated and no longer maintained. IFC2.0's primary focus was to incorporate schemas for building services, cost estimation and construction planning.

These first releases did not find widespread acceptance in the industry and were not often used in real life projects. The initial enthusiasm dwindled and there were no long-term plans. This was mainly due to the focus of IAI on general implementation and certification with insufficient implementation of support processes, resulting in the absence of a unified robust certification process. In this time the IFC standard was limited to members of IAI, which changed in 1999, when it was made openly availa-

ble for free as the IFC2x release. Soon after, in 2001, ifcXML was developed and released. However, ifcXML files are much less efficient in terms of their size compared to the traditional EXPRESS format. Following IFC2x was submitted to the ISO/PAS process and reached ISO/PAS 16739 status in the year 2005. Unusually, ISO was now involved in two similar or even competing standards for AEC/FM; STEP and IFC. Because of the multitude of new information supported by IFC2.0, the IFC2x release was more or less a stability release. Essentially, IFC2x was the stable core of IFC2.0 submitted to ISO for the process of standardization.

In the meantime, in 2003, IFC2x2 was released as the next major update delivering new features such as: 2D model space geometry, presentation, structural analysis and detailing, extension of the building service component breakdown, support for building code verification, and facility management. (Liebich, 2010)

In order to bring IFC closer to business IAI implemented certain changes in 2006. Their organization did not change, but their methods and approach did, wherein they shifted from a completely technical standpoint to a more business and user oriented one. Data exchange was narrowed down to manageable, predictable and implementable specifications. Consequently, this aspect of minimum data exchange made IFC a better exchange format while respecting the wishes of the industry. Furthermore, with the increase of standard users exchanges would be easier and clearer.

Two concepts were developed for this purpose (Laakso & Kiviniemi, 2013). The first of which are Information Delivery Manuals (IDM). The specification was introduced in 2007 as part of the official IFC standardization. Their main goal is supporting the integrated construction process by aiding software developers with implementation and by providing role-based process workflows for end users.

"The Information Delivery Manual (IDM) aims to provide the integrated reference for process and data required by BIM by identifying the discrete processes undertaken within building construction, the information required for their execution and the results of that activity. It will specify:

- *where a process fits and why it is relevant*
- *who are the actors creating, consuming and benefitting from the information*

• *what is the information created and consumed*

• *how the information should be supported by software solutions."* (Wix, 2010)

The second concept developed is the Model View Definition (MVD). It was adopted by buildingSMART in 2005 and became a part of the IFC standardization in 2006. An MVD defines a legal subset of the IFC schema that satisfies the exchange requirements of the AEC industry defined in an IDM. An exchange requirement is the information that must be exchanged to support a particular building requirement at a certain stage of the project. It also provides implementation guidance for all IFC concepts from this subset, thereby defining the software requirement specification for the implementation of an IFC interface. MVDs are defined either externally or by buildingSMART. The externally defined MVDs must be submitted, reviewed and accepted by buildingSMART before they are considered buildingSMART MVDs. (buildingSMART)

The third and last pillar of IFC data exchange (along with IDM and MVD) is the International Framework for Dictionaries (IFD). It provides a mechanism for the creation of dictionaries and ontologies for the purpose of connecting existing databases (e.g. materials or other descriptions) to the IFC data model. The content of existing libraries is completely open and free. In comparison to IFC, IFD information is not stored in a text format. In IFC it is tagged with a Global unique ID (GUID) and referenced to a local or remote library. Ideally all IFD libraries should be unified in one global library to which all information would be referenced.

After IFC2x2 and all its new additions, as stated by Liebich (2010), the release of IFC2x3 provided a quality assured and improved makeover of the previous release and was conservative on introducing new features. The current and newest release, from March 12th 2013, is IFC4, which, in respect to the previous release, includes new features and improvements as well as corrections for technical problems. Soon after, on March 21st 2013, IFC4 was also released as a full international standard ISO 16739. Some of the new features include: support for new BIM workflows – 4D and

5D[1] model exchanges, manufacturer, product libraries, BIM to GIS interoperability, thermal simulations and sustainability assessments, full integration with mvdXML, and the extension of IFC to infrastructure and other parts of the built environment. In addition, this new release contains the ifcXML4 schema, which is integrated into the IFC specification along with the EXPRESS schema. (Liebich, 2013)

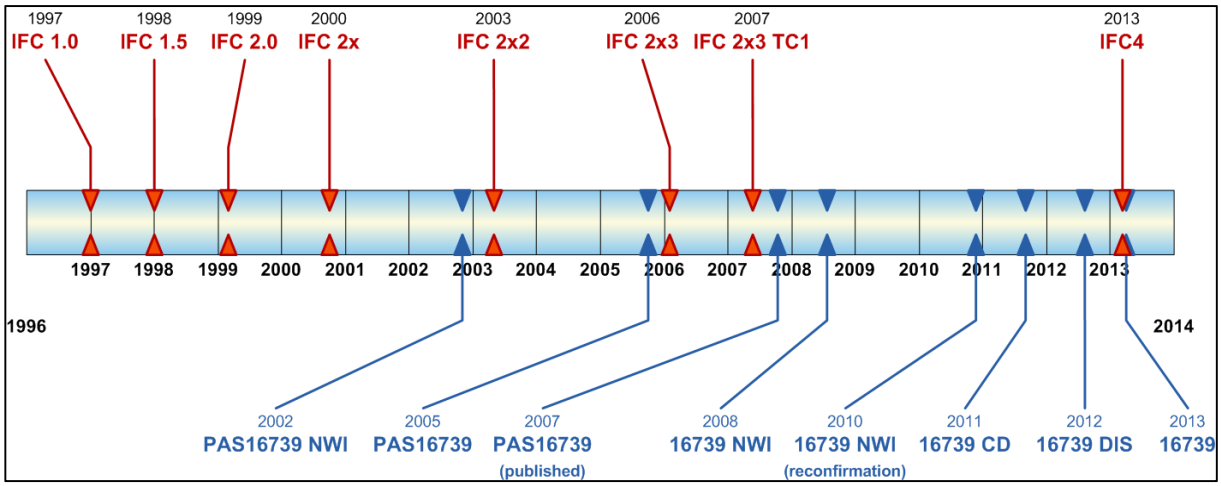


Figure 3: History of IFC releases (Liebich, 2013)

During this time a lot of emphasis was put on a new IFC certification process that was developed in 2007. The former process of certifying the quality of implementation had issues such as being inconsistent in its methods, covering only IFC data export and excessive simplicity, which did not ensure the possibility for real-life project adoption. The new and also current certification process was labelled "IFC Certification 2.0". Special attention was placed on MVDs along with a detailed quality control of the IFC interfaces in addition to the self-check of the software developer in comparison to the older "IFC2x Certification" that has been an ability check of the software developer to implement the IFC standard by following the outlined procedures (Groome, 2010). Therefore, the most important principles of the new certification process are the following:

- Certification of software is bound to a specific IFC MVD,

[1] - general consensus is that the fourth dimension is time and the fifth is cost

- Software certification is provided through a conformance test suite (model creation guideline sets for export test files with test instructions, automated export file testing based on rules imposed by model view definition, calibration test files for import checking),

- Software certification is managed by a web-based database (test guidelines, files and instructions, capturing and reporting of test results, running automatic export testing along with workflow support for manual evaluation). (buildingSMART, 2010)

(Laakso & Kiviniemi, 2013)

2.1.1.2 Technical specifications and system architecture of the IFC format

As mentioned in 2.1.1.1, IFC is based on the ISO STEP standard, most notably on the concepts of BCCM, the EXPRESS modelling language, and definitions for geometric representation (Laakso & Kiviniemi, 2013). Although the objectives of the AEC part of ISO STEP and IFC were the same, it has been noticed that ISO STEP could lead to incompatible standards in AEC. Therefore, the AEC software industry decided to develop a standard based on ISO STEP in an international consortium.

The IFC data model incorporates building components, e. g., doors, windows, walls, furniture and so on, as well as intangible components such as schedules, spaces, cost etc. These objects are IFC entities. These entities can have various properties such as a name, material and geometric properties, relationships to other entities and so on, rounding up the IFC model as a complete building information model.

In analysing the system architecture of IFC, the primary division is found to be in four conceptual layers representing four different levels. Each of these layers is comprised of several different categories or schemas that in turn hold the definitions of all the aforementioned entities. An important note on this layered structure is that an entity at any given level can only reference or be related to an entity at the same or lower level, but not entities at higher levels (Khemlani, 2004). The four layers are as follows (Figure 4):

1. Resource Layer – lowest layer containing all individual schemas with resource definitions,
2. Core Layer – next layer including the kernel schema and the core extension schemas,
3. Interoperability Layer – contains entity definitions that are specific to a general product,
4. Domain Layer – highest layer that contains schemas with entity definitions that are specializations of products, processes or resources specific to a certain discipline.

At the most basic level the division in the IFC schema is between rooted and non-rooted entities i.e. derived from *IfcRoot* or not. As seen in Figure 4 (buildingSMART), the bottom part where the schemas are in octagons is the resource layer with the non-rooted part of the schema. These schemas consist of supporting data structures and cannot exist independently unless they are referenced. Resource definitions do not have a concept of identity (no globally unique identifier – GUID) meaning that multiple objects can (should) but do not have to share an identical resource definition.

The core layer defines the most basic structure in the IFC schema, establishing the fundamental relationships and common concepts for all further specializations in aspect specific models. Entities from this layer derive from *IfcRoot*, and therefore already have unique identification (GUID), optional name and description and optional ownership and change information. The core layer is further broken down into four schemas:

1. *IfcKernel* – general constructs with different semantic meaning like object, property and relationship.
2. *IfcControlExtension* – basic classes for control objects.
3. *IfcProcessExtension* – mapping of processes in a logical sequence of planning and scheduling of work and the tasks required for its completion expressed in classes in exactly the same way as product information.
4. *IfcProductExtension* – specializes the concept of a (physical) product. Information for individual products is provided as subtypes of *IfcProduct* and for specific product types as subtypes of *IfcTypeProduct*.

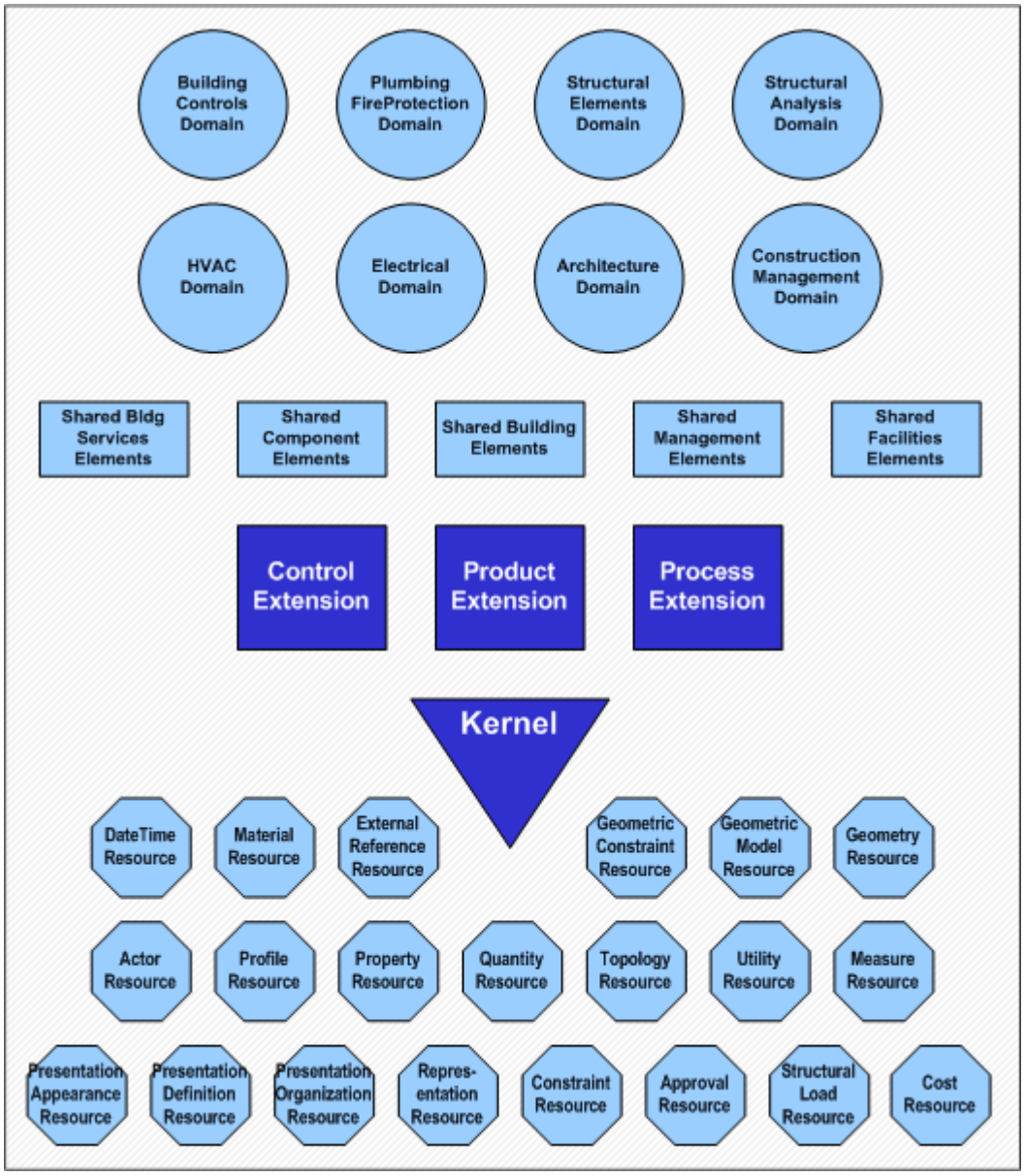


Figure 4: Core Data Schemas (buildingSMART)

IfcKernel should be looked at in more detail as it provides basic definitions for the Core layer and all layers above. The general constructs with various semantic meanings are specialized into non-AEC/FM specific constructs (product, process, control and resource). Basic attributes and relationships, such as relative location of products in space, and sequence of processes in time, are also specified in *IfcKernel*. According to the common understanding of an object model (object, relationship, property), the three fundamental entity types that are derived from *IfcRoot* are object definitions (abstract supertype *IfcObjectDefinition*), relationships between objects (*IfcRelationship*) and object characteristics (property definitions – *IfcPropertyDefinition*),

which together construct the first level of specialization. All three fundamental entity types are further specialized. Object definitions are specialized into the following:

- *IfcObject* – individual object in space,
- *IfcTypeObject* – common definitions identical for all objects,
- *IfcContext* – basic project or library context,

where IfcObject forms the basis for the following fundamental entity types:

- Products – physical objects, physically existing objects and conceptual objects with a shape representation and location in the coordinate space,
- Processes – actions in a project with the intention of acquiring, constructing or maintaining objects placed in time,
- Controls – concepts that control or constrain objects, requirements the object has to fulfil,
- Resources – describe the usage of an object mainly in a process,
- Actors – human agents involved in a project during its life cycle,
- Groups – arbitrary collections of objects.

Likewise, *IfcRelationship* forms the basis for these fundamental relationship types that are derived from it:

- Assignment – one object (the client) requests the services of other objects (the suppliers),
- Association – associates external sources of information to objects or property definitions,
- Decomposition – defines the relationship of the whole and the part where navigation between them is possible,
- Definition – uses a type definition or property set definition to define the properties of the object,
- Connectivity – connectivity of objects,
- Declaration – declaring context between linked object definitions and property definitions.

The same is true for *IfcPropertyDefinition*, which acts as a basis for these two fundamental property definition types:

- Property and property set template – syntax and data types for individual properties and sets of properties,
- Property set occurrence – subsets of shared property information attachable to objects in question.

At first glance the geometry information could be extracted directly from the Product entity types that are derived from *IfcObject*, while the relationships between spaces, especially the ability to transition from one space to another through any kind of transitional object, could be extracted from the fundamental relationship derived from *IfcRelationship* type Connectivity. As we can see from the description, the entity types from the fundamental group Products contain the geometrical representation and location of physical (walls, columns, beams) and physically existing objects, e.g., spaces establishing the entities from these groups as the default entities carrying geometry data. If we extend our view to fundamental relationship entity types, the Connectivity group shows promise regarding the solution to the space connectivity issue. However, in comparison to the Products group, where it was immediately clear that entities would provide geometry data, it is not as clearly stated and therefore required a closer analysis of individual entities if the required type of connections would be possible to create with the available entities, specifically in the case of the *IfcSpace* to *IfcSpace* connection. Thus, the entity types from the Products and Connectivity groups will be examined and analysed first. Most, if not all required data is expected to come from these two groups.

2.1.2 CityGML (City Geography Markup Language)

CityGML is an information model based on GML with XML type encoding and its purpose is the representation, storage and exchange of (as its name says) primarily virtual 3D city models but also of landscape models. Incorporating a unique five detail level system (LOD – Level of Detail), 3D objects are described regarding their geometry, topology, semantics and appearance. The core information for objects, therefore, is very similar to 3D objects in an IFC building information model, making it possible to bridge the gap between Urban Information Models and Building Information Models in respect to interoperability.

The data stored in CityGML is applicable in several different areas. Most notably in sophisticated analysis and display tasks in different applications, e. g., environmental simulations, energy demand simulations, urban facility management, disaster management, pedestrian navigation, location based services, and urban data mining. Additionally, the 3D visualization purposes for cities are also worth mentioning.

CityGML, like IFC, is an open standard that can be used free of charge. It is implemented as an application schema for the Geographic Mark-up Language 3.1.1 (GML3), which is the extendible international standard for spatial data exchange issued by the Open Geospatial Consortium (OGC) and the ISO TC211. As stated in the CityGML Encoding standard (Gröger, et al., 2012), the main goal of the development of CityGML is to reach a common definition of the basic entities, attributes and relations of a 3D city model, thereby preventing data redundancy between different application fields.

At its core the concept of the CityGML data model is a modular composition. Because of the possibility that different implementations do not have to support the whole CityGML data model, this core concept is essential for the efficiency of the data model and its ability to be utilized in this way. The environment in the 3D model is described by class definitions for the most important types of objects that are manifested inside a virtual 3D city model. The available classes were identified to be either important or required for many distinct application areas. Fundamentally, it is distinguished into a core module and thematic extension modules, where the core module is required in any special implementation because it features the basic concepts and components of the CityGML data model. Based on the core module, thirteen individual thematic extension modules can be distinguished: *Appearance*, *Bridge*, *Building*, *CityFurniture*, *CityObjectGroup*, *Generics*, *LandUse*, *Relief*, *Transportation*, *Tunnel*, *Vegetation*, *WaterBody* and *TexturedSurface* (deprecated). A detailed overview is in Table 1. Other than the core module, which is imported by every thematic extension module, individual thematic extension modules are independent from each other and are specified by their own XML schema.

Table 1: Thematic extension modules for CityGML

Module Name	**Module Description**
CityGML Core	Basic components of the CityGML data model. Abstract base classes from which all thematic classes are derived.
Appearance	Enables modelling of the appearance of CityGML features.
Bridge	Allows for the representation of thematic and spatial aspects of bridges, bridge parts installations and interior bridge structures in 4 LOD (1-4)
Building	Allows for the representation of thematic and spatial aspects of buildings, building parts, installations and interior building structures in 5 LOD (0-4)
CityFurniture	City furniture comprises fixed objects like lanterns, traffic signs, advertising columns, benches etc.
CityObjectGroup	Grouping concept for CityGML. Arbitrary city objects can be grouped with respect to user-defined criteria.
Generics	The transfer of additional attributes and features that are not covered by other CityGML modules.
LandUse	The division of the surface of the earth according to specific land use.
Relief	The representation of terrain with various levels of detail.
Transportation	Transportation features within a city, such as roads, railway tracks, and squares.
Tunnel	Thematic and spatial aspects of tunnels, tunnel parts and installations, and interior tunnel structures in 4 LOD (1-4).
Vegetation	Thematic classes for the representation of vegetation objects. Distinguishes between solitary objects and vegetation areas.
WaterBody	Thematic aspects of the 3D geometry of rivers, canals, lakes and basins.
TexturedSurface	Adds visual appearance properties and textures to 3D surfaces. Expected to be deprecated due to inherent limitations. Information transferred to the appearance module.

Any specific implementation of CityGML incorporates the core module and any combination of extension modules. These individual implementations are called CityGML profiles. The combination of the core module and all of the thematic extension modules is the CityGML base profile. Any CityGML profile is therefore a valid subset of the base profile.

2.1.2.1 History and background of CityGML

In recent years, various municipalities and companies started creating 3D city models for the purpose of tourism, disaster management, telecommunication, vehicle and pedestrian navigation etc. Most of these initial models were based solely on geometry, lacking any kind of semantic or topological data. This meant that the models could be used for visualization purposes but not much more. However, the requirements for these models are much higher, demanding a much broader field of use including analysis tasks and data mining.

CityGML wants to fulfil these needs by incorporating semantic and topological data. Members of the Special Interest Group 3D (SIG 3D), which has been part of the initiative of Spatial Data Infrastructure Germany (GDI-DE) since 2010, have been developing CityGML since 2002. SIG 3D is an international open group of more than 70 members from Germany, Great Britain, Switzerland and Austria including companies, municipalities and research institutions.

Ever since the first successful implementation and evaluation by a subset of CityGML on a project called "Pilot 3D", the standard has found its place worldwide in the following cities, which provide their 3D models with various levels of detail:

- Berlin, Cologne, Dresden and Munich,
- Paris, Lille, Nantes, Marseilles – LOD 2
- Monaco, Geneva, Zurich, Leewarden – exchange data LOD 2 or 3
- Cities in Denmark – LOD 2, 3 or 4
- Asia: Istanbul (LOD 1 and 2), Doha and Katar (LOD 3), Yokohama (LOD 2).

Along with a growing base of users many tools have been developed that support CityGML. The developers come from either a commercial (companies) or academic background (academic institutions).

The following developments are among the most notable:

- 3D City Database (free and open 3D geo database) based on Oracle 10g R2 and 11g R1/R2,
- FME (Feature Manipulation Engine) from Safe Software Inc.,
- BentleyMap from Bentley Systems,
- Support GIS from CPA Geo-Information,
- Aristoteles Viewer from the University of Bonn,
- LandXplorer CityGML Viewer from Autodesk Inc.,
- FZKViewer for IFC and CityGML from KIT Karlsruhe,
- BS Contact from Bitmanagement Software GmbH – CityGML plugin for the geospatial extension BS Contact GEO.

Additional information about software tools can be found on the official website of CityGML (Open Geospatial Consortium, 2012) or the CityGML Wiki.

These tools provide mostly Import and export interfaces, 3D views and representation, but also the ability to manage and store city information. In addition, an open source Java class library API for the processing of CityGML models has been made available by the Technische Universität Berlin. (Gröger, et al., 2012)

2.1.2.2 Building Information in CityGML

By general definition a city is comprised of various objects such as buildings, vegetation, water bodies, "city furniture" etc. Buildings are one of the most important parts of any city. Therefore, special attention is devoted to storing building information. A model with five levels of detail is used specifically for buildings, representing the buildings according to a specific level of information. The levels of detail are referenced as LOD plus a natural number from 0 to 4 to determine the level as follows:

- First level of detail – LOD 0 – coarsest level, which is essentially a two-and-a-half dimensional Digital Terrain Model. Buildings are either represented by their footprint or roof edge polygons. (regional landscape)
- Second level of detail – LOD 1 – is a block model where the buildings are represented by rectangular solids with flat roofs. (city, region)

- Third level of detail – LOD 2 – unlike LOD 1 the roofs and side surfaces are defined. (city districts, projects)
- Fourth level of detail – LOD 3 – walls and roofs are detailed and feature doors and windows. (architectural models – exterior, landmarks)
- Fifth level of detail – LOD 4 – adds interior objects and structures to LOD 3, comprising of rooms, furniture, stairs etc. (architectural models – interior)

Each object in CityGML can have a different representation in every individual LOD. Different objects from the same LOD can also be represented by an aggregate object in a lower LOD.

In addition to the differentiation above, which is only descriptive, there is a differentiation based on differing accuracies and the minimal dimensions of objects. Accuracy is described as the standard deviation of absolute 3D point coordinates. The accuracy requirements, however, are presently only a proposal and subject to debate. Future versions of the standard will introduce accuracy based on relative 3D point coordinates, which is generally much higher than accuracy based on absolute 3D point coordinates.

The obvious level of detail that is of interest for a building model for Indoor Positioning is LOD 4 since it covers the interior of the building. The question that arises (CityGML being first and foremost a city and landscape standard) is about the source of this data. As discussed by Isikdag & Zlatanova (2009) one of the potential and best sources for automatically generating building data for cityGML should be the IFC standard. As an international standard it offers all the required standardized data sets for this. Similarly to the IFC schema, CityGML also supports a model with semantics and geometrical properties, incorporating attributes, relations and aggregation hierarchies (part-whole relations). Basically, the model can be described as consisting of two hierarchies, the semantic and the geometrical objects of which are linked by relationships. This enables model navigation, which allows us to perform geometrical and thematic queries and analyses.

A dilemma arises here, since IFC as a data source can be accessed directly without an intermediate step through CityGML. Hence, acquiring data from CityGML would actually introduce two steps, because data has to be transferred from IFC to

CityGML and again to IPS XML, which can cause certain errors and the occurrence of redundancy. Additionally, despite the similarity of the CityGML and IFC model in terms of semantics and geometrical relationships, CityGML might not import the entire IFC schema due to certain information not being needed for city management, leaving out crucial information that is required for the IPS schema.

Having said that, cityGML does not offer any added value in comparison to IFC. Moreover, it is most likely that building information will come directly from IFC, which could add certain complications. Therefore, cityGML will not be considered further and IFC will take precedence.

2.1.3 Green Building XML (gbXML)

Green Building XML is an open schema with the main purpose of transferring building information from CAD applications, specifically building properties from building information models, to various engineering analysis tools. By doing this, the time consuming and error prone process of acquiring information from plans is made obsolete, thereby providing a means of streamlining the design process of cost and resource efficient buildings. Due to a high level of acceptance among leading vendors (such as Bentley, Autodesk, Graphisoft) and the development of import and export tools, it became a de facto industry standard schema.

Through the use of the computer language XML (extensible markup language) data transfer between different software tools is made possible with minimal or no human interaction. Consequently, the building design process is freed from this tedious task and can focus on the main mission of designing efficient and aesthetically pleasing buildings.

Consequently, gbXML is a schema that can potentially provide the source of building data required for an IPS building model in a streamlined manner. The nature of the format ensures that the geometry of spaces will be defined with no additional data that could make the files unnecessarily big. According to Rueppel & Stuebbe (2008), the relations of doors belonging to walls it is possible to create routing networks. The downside is that the data provided by gbXML might not be enough to create a fully functional IPS building model.

2.1.3.1 gbXML history

The Green Building XML schema, also referred to as gbXML, started development in 1999. The first version of the schema was published in the next year in June, and the official website supporting and promoting the gbXML schema was launched in 2002. By 2009 the new gbXML advisory board was formed with members of software development firms. The official name of Open Green Building XML Schema, Inc. was adopted. Several new versions of the schema have been released to date. The present version, gbXML 5.12, was released in 2014 along with a new schema validator (Version 2).

2.1.3.2 gbXML specification

The Green Building XML schema incorporates two different types of objects used to describe a building. The first type is called Elements; these are containers of building data. The elements are predominantly built to store the physical characteristics and geometry of indoor spaces. This mirrors the main purpose of the gbXML schema as a specialized building model for the transfer of building data for various energy analyses. The other type of objects is called Simple types; these are enumerations with predefined types of certain attributes used in Elements.

Since Elements are the objects that contain building data, this is the type that needs further attention. This type of data storage closely resembles the formats described in the previous chapters on IFC and CityGML: namely, all data in a specific Element is stored in attributes. These Elements can be grouped into three distinct subsets; the administrative (person data, identification, location, dates etc.), geometric (spaces, surfaces, boundaries, areas, volumes) and physical characteristics (heat, lighting, airflow, temperature etc.). Elements that define physical characteristics are of no particular interest for Indoor Positioning and can be omitted from further research. However, both other subsets can provide valuable data, especially the geometrical Elements, which should be designated as essential. The ones that have been identified as contributing most are listed alphabetically: *AdjacentSpaceId*, *Space* and *SpaceBoundary* (gbXML, 2014). Their attributes and descriptions are introduced in the following Tables 2 through 5. Additionally, we have to list the hierarchy of children, starting with *SpaceBoundary*, which finally lead us to the geometric definition of the

boundaries of a particular Space. These elements are the following: *PlanarGeometry*, *PolyLoop*, *CartesianPoint* and *Coordinate*.

Elements that are referenced by another Element are called children. A child element can in turn have its own children (gbXML, 2014). In this way, relationships between different Elements are established. Among the Elements described above only *AdjacentSpaceId* does not have any children. *SpaceBoundary* only references one other *Element*: *PlanarGeometry*. The remaining *Space Element* is more complex. Its children can be found in Table 6 respectively.

Table 2: AdjacentSpaceId

Attribute	**Description**
spaceIdRef	ID of a space that is bounded by this surface
surfaceType (optional use)	Distinguishes between different Surface types from the surfaceTypeEnum

The *Space Element* (see Table 3), which represents the volume enclosed by surfaces, is essential for the definition of topological space and features boundary geometry through the next *Element* listed, *SpaceBoundary*.

Table 3: Space – A volume enclosed by surfaces.

Attribute	**Description**
Id	Identification number
spaceType	Determines space usage from the spaceTypeEnum
zoneIdRef	
scheduleIdRef	ID of the schedule of transmittance of a shading surface
lightScheduleIdRef	ID of the lights schedule
equipmentScheduleIdRef	ID of equipment use
peopleScheduleIdRef	ID of schedule of people
conditionType	
buildingStoreyIdRef	ID of building storey the space is on
ifcGUID	GUID from IFC

The *SpaceBoundary Element* in Table 4 represents the boundary of spaces. The *PlanarGeometry* of the *SpaceBoundary Element* is part of the given space *ShellGeometry*, defining the interior surface bounding the space.

Table 4 : SpaceBoundary Element

Attribute	**Description**
ifcGUID	GUID from IFC
isSecondLevelBoundary	Boolean if true the boundary is important in heat flow calculation
surfaceIdRef	Connects the space boundary to a surface representing a building element (or open air)
oppositeIdRef	References the space boundary of the opposing space

The last *Element* is the *PlanarGeometry Element* (see Table 5), which provides a list of points defining a loop.

Table 5: PlanarGeometry Element

Attribute	**Description**
Id	Identification number
unit	Type of surface from surfaceTypeEnum

After *PlanarGeometry*, a series of *Elements* that are referenced as children establish the geometrical representation of any given boundary. Apart from the data these *Elements* store, they lack their own ID and do not feature any attributes. Starting with *PolyLoop*, which is the child of *PlanarGeometry*, this *Element* features a list of coordinates that make up a polygon in 3D space, applying the right-hand rule for the definition of the outward normal. The child of *PolyLoop* is the *CartesianPoint*, defining the x, y and z distances from the coordinate system origin. To conclude the list, the *Coordinate Element* being the child of *CartesianPoint* defines a single coordinate as the length measurement of x, y or z from the coordinate system origin.

The most complex *Element* discussed here is the *Space Element*, featuring several children that are listed in Table 6.

Table 6: Children of the Space Element

Child Element	**Description**
Name	Name of type String
Description	Description of type String
Lighting	Lighting in space
LightingControl	Lighting control
InfiltrationFlow	Flow of air through building envelope
PeopleNumber	Space occupancy
PeopleHeatGain	Amount of heat added to a space by people
LightPowerPerArea	Amount of power used by lighting in an area
EquipPowerPerArea	Amount of power used by equipment in an area
AirChangesPerHour	Amount of air changes per every hour
Area	Area enclosed by the physical boundaries of a space
Temperature	Temperature
Volume	Volume of enclosed physical boundaries of a space
PlanarGeometry	Planar polygon of a space perimeter (floor area)
ShellGeometry	Planar polygon of interior bounding surfaces
AirLoopId	ID of air loop
HydronicLoopId	ID and type of hydronic loop
MeterId	ID of a resource meter
IntEquipId	ID of an interior equipment object
AirLoopEquipmentId	ID of an air loop equipment object
HydronicLoopEquipmentId	ID of a hydronic loop equipment object
CADObjectId	Maps unique CAD object ids to gbXML elements
TypeCode	User or project defined code for Space
SpaceBoundary	Establishes a logical relation of Shell geometry so that PlanarGeometry is part of an interior surface bounding the space.

We can immediately notice that gbXML deals mostly with spaces and the relationships between them. On the one hand, this fact enables the format to be sleek and efficient, which is one of the requirements for the IPS building models. On the other hand, this same fact acts as a limiting factor. Moving around in spaces requires addi-

tional information about obstacles such as furniture, vertical movement installations such as elevators, escalators or stairs, and other obstructions. For this reason, gbXML is not the best choice from the perspective of creating a complete IPS building model, which would include all of the factors mentioned above. Eventually, it would be desirable for the IPS building model to incorporate semantics to work along with geometry data spaces. Therefore, despite being a promising alternative to IFC, gbXML was characterized as not suitable, mainly because IFC offers more data with which to work.

2.2 Indoor Positioning

The tendency of cities to become even more densely populated is increasing, leading to ever more human activities taking place indoors. (Walder, 2006) Indoor Positioning supports a number of these activities. As stated in 1.1, this is the reason why there are several different methods of Indoor Positioning available. The approaches and consequently the solutions are tailored to the specific needs of the projects that the Indoor Positioning method was developed for. Predominantly, Indoor Positioning solutions are developed in the scope of projects that can benefit greatly from a well determined position inside a building, such as developing aids for the blind or visually impaired, command and control of first responder operations, applications for shopping malls, simulations of crowd dynamics (pedestrian movement, evacuation dynamics) etc. Therefore, a common approach has yet to be determined. Each approach works differently in addressing the specific needs of the main research topic. Looking at the basic requirements, we have categorized Indoor Positioning approaches according to the installation of sensory equipment, either fixed installations or portable sensors worn autonomously by each individual user of the positioning equipment.

GPS is a widely adopted and used technology for outdoor positioning, but has certain limitations if used indoors. Newer buildings with steel frames or reinforced concrete constructions and energy efficient coated glass windows weaken and absorb the microwave signals from GPS satellites. In fact the signals become so weak that they cannot be received indoors. Solutions have been suggested in the form of augmented GPS systems (Reynolds, 2003), but even these methods do not solve all of the

related issues and only provide a partial solution. Augmented GPS systems also require additional infrastructure that might not provide coverage in all buildings. As Reynolds (2003) also emphasizes, GPS might be available globally, but it is controlled by the US government, which may deny the right to GPS as it pleases.

Fixed sensor installations inside the building usually consist of a transmitter and a receiver (Reynolds, 2003). Which one is fixed varies, depending on the technology used. Low frequency (LF) indoor positioning methods require fixed transmitting stations, whereas High Frequency (HF) RFID (Radio Frequency Identification) systems require fixed receivers and portable readers. Other possible technologies are Ultra-Wide Band (UWB) and Wireless LAN (WiFi). With fixed positions the overall accuracy of the system does not suffer because the method of determining positions is absolute, therefore no drift occurs. Accuracy is only dependant on the quality of the sensors and the infrastructure around them. This gives the whole system that uses fixed sensor installations a certain consistency and reliability in typical, every-day situations. There is a considerable downside, however, as fixed installations increase the cost of buildings. Depending on the technology used, the installation and maintenance can even exceed the cost of the installed equipment. Another downside is that in case of emergencies, when infrastructure can be destroyed, this kind of fixed sensor installation can be a liability as, along with infrastructure, sensors can get destroyed as well. Even in the case of the sensors surviving, the whole environment can change and the absolute position of sensors might not correspond with their initial position.

However, we also have Indoor Positioning systems with portable autonomous sensors. Since the whole principle of these sensors is that they are portable and autonomous, they do not have absolute references and, therefore, determine the position relatively to the previously measured position. For that reason, an initial position and alignment of the sensor that puts the particular session of positioning in perspective is required, thereby turning the raw measurement data of bearing and distance into information about the location. Typically, the sensors being used for this are inertial measurement units (IMU), incorporating the following four types of sensors:

- Accelerometer – measures the current rate of sensor acceleration in three directions,

- Gyroscope – measures changes in the rotation of the sensor,
- Magnetometer – measures the magnetic field,
- Barometer – measures atmospheric pressure.

Over time, this approach suffers from progressively larger errors in position determination. Relatively small errors in the measurement of acceleration and angular velocity are integrated into larger errors in position, and because every new position is calculated from the previous one these errors are added together and prove to be significant as time passes. Certain methods have been developed to cope with drift. However, none of them can eliminate it completely and drift still remains the greatest enemy of inertial positioning and navigation, which is why it is mostly used as a supplement to other methods of positioning. The biggest advantage of inertial positioning, in comparison to fixed sensors, is that it is completely independent from infrastructure, and that the relative positions measured (the changes) are very accurate since the error becomes significant only when it has accumulated over time. A comparison of the accuracy of different systems in respect to their coverage can be found in Figure 5.

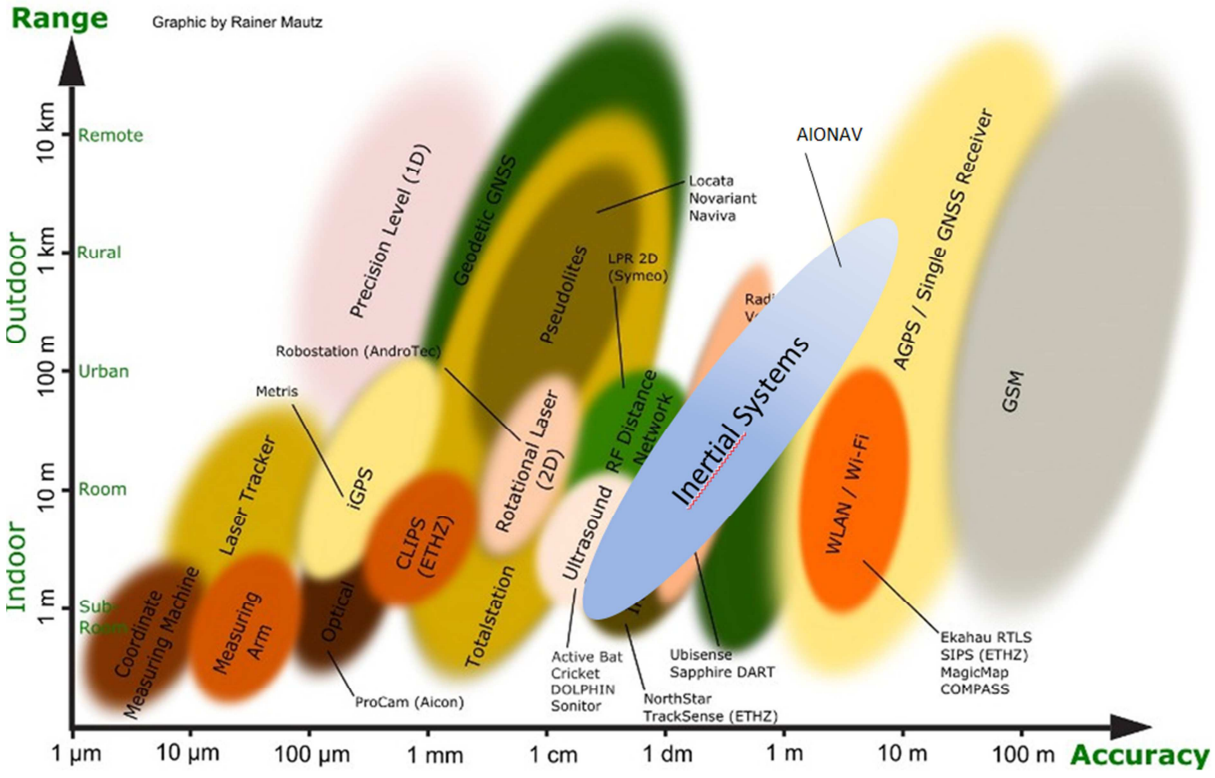


Figure 5: Accuracy comparison of IPS systems with AIONAV taking the middle ground (in blue) (Walder, 2012)

The following chapters will deal with the description of some of these approaches.

2.2.1 BIM-Based Indoor-Emergency-Navigation-System for Complex Buildings

This approach was developed in the scope of researching an emergency system for the Frankfurt Airport (Rueppel & Stuebbe, 2008). It is based on building models and uses several methods of determining indoor positions. Satellite based navigation, such as GPS and Galileo, has been dubbed unsuitable, and technologies like WLAN, RFID and UWB are used instead. All three methods are combined and integrated into the navigation integration platform. Building models, specifically gbXML, are used for the visual representation of the current user position as well as for generating routing networks. The aim of the system is to improve disaster responsiveness by providing real time positioning information (RTLS – Real-Time Location System).

The three technologies used for Indoor Positioning each have their own purpose to take advantage of their strengths. The RFID component uses active RFID tags that are predominantly located in sections of the building where there is little or no technical infrastructure that could be used to enhance positioning. These are, for example, underground car parks or cellars. Active tags have been chosen because of their smaller antennas, which are more suitable for portable devices and allow for reception in multiple directions. However, it has been found that the signal strength of active RFID tags does not suffice for an effective and precise calculation of distances. Therefore, measures have been taken to increase accuracy.

Ultra-wide band is particularly effective for use in halls, because it has a lower sensitivity to metal and high humidity in comparison to other radio communication technologies. Passenger and baggage halls in the Frankfurt Airport are therefore suitable candidates. The concept of positioning is based on the calculation of distance between sender and receiver through timed short pulses on a wide frequency range. Short pulses make it possible to distinguish between direct and reflected signals, because there is no overlapping of signals.

The last RTLS component is WiFi, which makes use of existing wireless LAN routers (especially in office areas) and does not require the installation of any additional

equipment. Positioning can be achieved with two different methods. Triangulation requires at least three WiFi points from which the distance is calculated and position determined. The other method is comparing pre-measured data to current measurements, i.e., the fingerprint method. However, the signal is so sensitive can be disturbed by individuals walking by.

The data from all three RTLS methods is then combined and displayed on an underlay extracted from gbXML that is accessible online on the navigation integration platform. Mobile devices are in direct communication with WiFi and UWB, and each individual device has its own RFID reader. The data is transferred to the navigation integration platform through web services, and the position of users can then be put into context with building data. Internet access is essential for mobile devices. (Rueppel & Stuebbe, 2008)

2.2.2 Environment-Aware Sequence Based Localization (EASBL)

The EASBL algorithm (Li, et al., 2014) strongly relies on the building information model to improve accuracy at room level. The fundamental aspect is to utilize Sequence Based Localization, which has proven successful for Indoor Localization. SBL revolves around the division of an area into special sub-regions. The area consists of n reference nodes, and for any two of the reference nodes a perpendicular line is created to the line segment joining the two reference nodes. Consequently, each of these perpendicular lines divides the space into three parts that are distinguished by their respective proximity to either reference node. These regions then acquire unique location sequences of nodes' ranks that are based on the distance of nodes to the region assigned and are inserted into a location sequence table. The localization of a target node is achieved by creating a location sequence of the target node that is based on received radio signal strengths from all reference nodes. This sequence is then compared to the sequences from the location sequence table and the centroid of the location sequence that is nearest to the target node location sequence, and is used as an region for estimating the target node location.

This method of localization's accuracy is heavily dependent on the number and distribution of reference nodes. Generally, the more reference nodes and the more even the distribution of them, the higher the accuracy. In emergency situations, when time

is of the essence and some areas might be difficult to access, a compromise has to be reached for the placement of radio frequency beacons that represent the reference nodes. Some spaces might not be covered by the ad-hoc sensor network.

The EASBL algorithm addresses the distribution of sensors by using the building model. It strives for a balance between localization accuracy and on scene deployment effort. The building model provides room-level accuracy, which helps the coordinate-level accuracy of SBL. The regions of SBL might not be restricted to rooms, and room-level accuracy may be low even though the coordinates can be determined accurately. Additionally, the building model provides the locations of building elements, such as walls and doors, which helps immensely with the deployment of the sensor network. In general, the building model in the EASBL algorithm provides unique environmental awareness to SBL, which is used for the aforementioned increase in room-level accuracy and the deployment of the ad-hoc sensor network. Two other benefits of building models are addressed, namely the labelling of rooms and the graphical user interface that the model provides.

A critical part in the calculation of sensor distribution in the EASBL algorithm is also the employment of metaheuristics. Soft computing techniques that make use of readily accessible and loosely applicable information to control problem solving in human beings and machines reduce the computation time significantly. The generic algorithm (GA), simulated annealing (SA) and tabu search (TS) are the most widely adopted metaheuristics dealing with optimization problems. Though not guaranteed to provide optimal solutions the results are deemed satisfactory.

The performance of the EASBL algorithm regarding accuracy was assessed as room-level accuracy (ratio of correct room-level estimations to the total number of targets) and coordinate-level accuracy (average error distance of meter-level estimation of all targets); repeated 100 times, the confidence intervals were calculated at a 95% confidence level. Testing was undertaken in two simulated fire emergency scenarios that were designed with the suggestions of a number of first responders and were reviewed and verified by two incident commanders from the LAFD. The resulting confidence intervals of the mean room-level accuracy and mean error distance were 87,0 ± 3,6% and 1,78 ± 0,24m in the first scenario, and 87,2 ± 3,8% and 1,57 ± 0,22m in the second scenario. Consequently, the overall deployment effort for sensor

placement in establishing ad-hoc networks was reduced by 32.1% on average compared to random placement of RF sensors. It was established that one of the crucial drawbacks to the implementation of the algorithm is the first responders' limited access to BIM. (Li, et al., 2014)

2.2.3 CADMS

The project started in 2007 under the leadership and management of Professor Ulrich Walder. After the past increase in terrorist attacks, certain deficiencies in the process of first responders and rescuers became apparent. The goal of the project was to use modern technology and especially information technology to improve the effectiveness of Disaster Management, and to help the rescue teams and consequently save more lives by making their work safer and more coordinated. In general, the system can be divided into the following five components (Schütz, et al., 2008):

1. Building model – the IPS BIM, specially designed for use in Indoor Positioning and Navigation.
2. Position estimation – real time raw data from sensors.
3. Position verification and correction – based on the building model, positions are checked and verified.
4. User interface – one for mobile devices (touch interface) and the other for the stationary command centre (conventional keyboard and mouse).
5. Communication – all devices are connected into a wireless network for coordination between users.

The position estimation component has required the most attention barring the building model. The condition of an autonomous system has eliminated most methods and technologies that could otherwise be used. WLAN, RFID and even GSM/UMTS are not suitable because of the high chance of fixed installations being destroyed in disaster scenarios. GNSS or GPS is not suitable either. The environment where the CADMS system must work in is mostly indoors or underground. Because of signal interference the results are rather poor and do not provide the required accuracy or consistency.

The sensor for determining the position had to be on the user, narrowing down the available possibilities. A portable Inertial Measurement Unit (IMU) was chosen (Figure 6). The robust casing holds these four separate components:

1. An accelerometer – measures acceleration of the sensor in space (in 3 directions).
2. Gyroscopes – measurements of the sensor orientation (roll, pitch and yaw).
3. Magnetometer – for measuring the magnetic field at the sensor location.
4. Barometer – changes in vertical height.

Figure 6: portable IMU MTw development kit (xsens)

In Figure 7 a recent development of the Xsense IMU can be observed. A particularly small unit that features most required sensors on-board, lacking only the barometer.

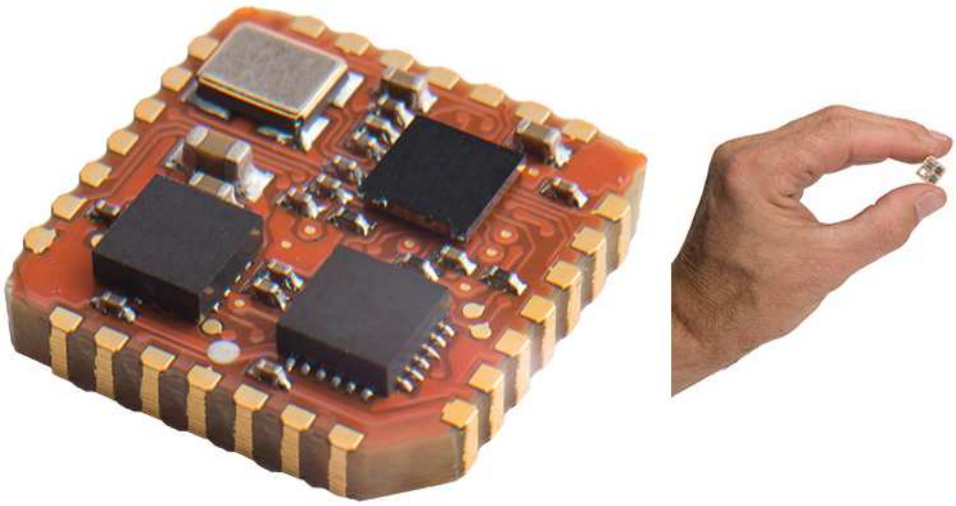


Figure 7: portable IMU MTi 1-series and size comparison to a human hand (xsens)

What makes this unit especially potent is the low price compared to other available alternatives.

An IMU cannot determine an absolute location. The principle it works on is "dead reckoning" which means that the position is measured relatively to a fixed starting position that has to be determined with other methods. The fixed starting point can be determined either manually by clicking on a map or automatically over any fixed installation (e.g. RFID) or GPS. With double integration of the raw measurements from the IMU the angle (direction) and distance of movement can be calculated and the current position determined. In Figure 8 we can observe two real-life track recordings. The track from the Graz University of Technology has a part of the plan modelled as the IPS XML. The Sydney opera house example on the other hand is only on an aerial image.

Figure 8: Left track from the TUG with the current room highlited in blue; right track Sydney opera house

The "dead reckoning" method, however, is not without flaws. Its greatest asset of being autonomous also has consequences. Since the measured positions are relative, not absolute, any error that happens is dragged along through all of the following measurements and determined positions and is added up with all of the following measurement errors (Chapter 1.1). Another downside of the system is that errors will

happen because no measurement can be perfectly accurate. The nature of determining a position with an IMU is such that accelerometer and gyroscope measurements will always produce "drift", along with any disturbances in the magnetic field picked up by the magnetometer. By drift I am referring to the accumulation of small errors over time leading to an increasing deviation of the actual position in comparison to the measured position. The reason this happens is that measurements are not continuous, they happen in predefined intervals. Modern equipment can take several measurements per second. Despite that, we always have to work with averages, and the acceleration could have varied greatly in the given time interval. These inaccuracies are addressed by recognizing motion patterns like walking or running etc., and applying the appropriate zero velocity updates (ZUPT) developed for a foot mounted sensor as well as the interaction of sensors with the building model (Walder & Bernoulli, 2010). Another possible alternative or addition to repositioning was developed and tested with the HSLU (Hochschule Luzern) by analysing video content (Bernoulli, et al., 2011).

2.3 Building information for Indoor Positioning

The introduced approaches to Indoor Positioning differ in the choice of sensors. The goal is to achieve an acceptable balance between the following demands: accuracy, autonomy, ease-of use and cost effectiveness. Because of this compromise, there is always some innate discrepancy of at least one of the aforementioned parameters. By incorporating building information, and specifically a building information model, all of these methods aim to even out the initial balancing, thereby making the system more effective in all crucial demands. However, the way building information is incorporated and utilized is tailored to each individual approach of Indoor Positioning and no standardized way of extracting and taking advantage of the information is presented.

Essentially, we can differentiate between two possible options of how to incorporate building information in Indoor Positioning: either by defining a completely new format for every individual method, or defining a standardized format that can meet the needs of each individual case. We can acquire data for these formats either by relying on existing building models or generating them manually.

In the following chapters, the various formats that were developed to satisfy the specific needs of Indoor Positioning will be introduced. Apart from IPS XML the other formats were chosen because all of them are in some manner based on extracting building data from existing formats (specifically IFC). The introduced models can be differentiated according to the general purpose they are supposed to fill, and on that basis they can be categorized into two groups. The first two models (GTN and Sparse Navigation Graph) are more or less graphs that serve the Dijkstra algorithm for finding the shortest path, whereas the second two models advocate a more general approach, where the model has several layers that can serve purposes other than finding the shortest path. Locating the user and navigating in an indoor environment as well as displaying that information is possible through incorporating the geometry of building objects like walls, doors, furniture etc. into the model. Regarding the nature of these formats it has to be emphasized that the IPS XML has been developed as a standalone format with all of the associated requirements such as that of an editor (for manual creation of the model) and an XML schema. The emphasis will be on IPS XML since it was chosen as the basis for the development of an algorithm to extract data from IFC.

2.3.1 Geometric Topology Network (GTN)

GTN is a spatial model that can be used in navigation guidance and positioning data correction in indoor environments. Looking at the basic concept, data from the IFC file, specifically building spaces and spatial connections, is transformed into a graph network. These are the three requirements that GTN must meet for efficient correction of erroneous indoor positioning data:

1. Dimensionally weighted topology network – accurately represents indoor route lengths,
2. Network based representation of indoor navigation routes – from planar polygons (hallways),
3. Medial axis based – furnishings elements on the periphery of spaces.

The third requirement was selected because of limited computational resources available on mobile devices. Other possibilities include a visibility-based navigation network, as suggested by Lee et al. (2010). The visibility network, however, is not

unique as it is in the case of medial axis based networks, because different routes produce different networks. Therefore, it is not suitable for mobile devices as a result of the difficulties associated with creating such a network. Suitable medial axis algorithms include the following:

1. Medial Axis Transform (MAT) – due to curvature and complexity involved, used for robot navigation.
2. Generalized Voronoi graphs
3. Straight Medial Axis Transform (S-MAT) – simplification of MAT with no curved segments.

MAT defines the medial axis as the locus of corners (see Figure 9. points C1 through C4) that appear in the propagating waves during the process of generating these parallel to the original form (Blum, 1967). The resulting locus is the set of points internal to the polygon that are equidistant from at least two points closest on the boundary. In Figure 9 the original form is a bold line and the medial axis formed by the process of propagating waves is a dashed line. The first corner does not appear immediately in the wave front because the original shape is smooth.

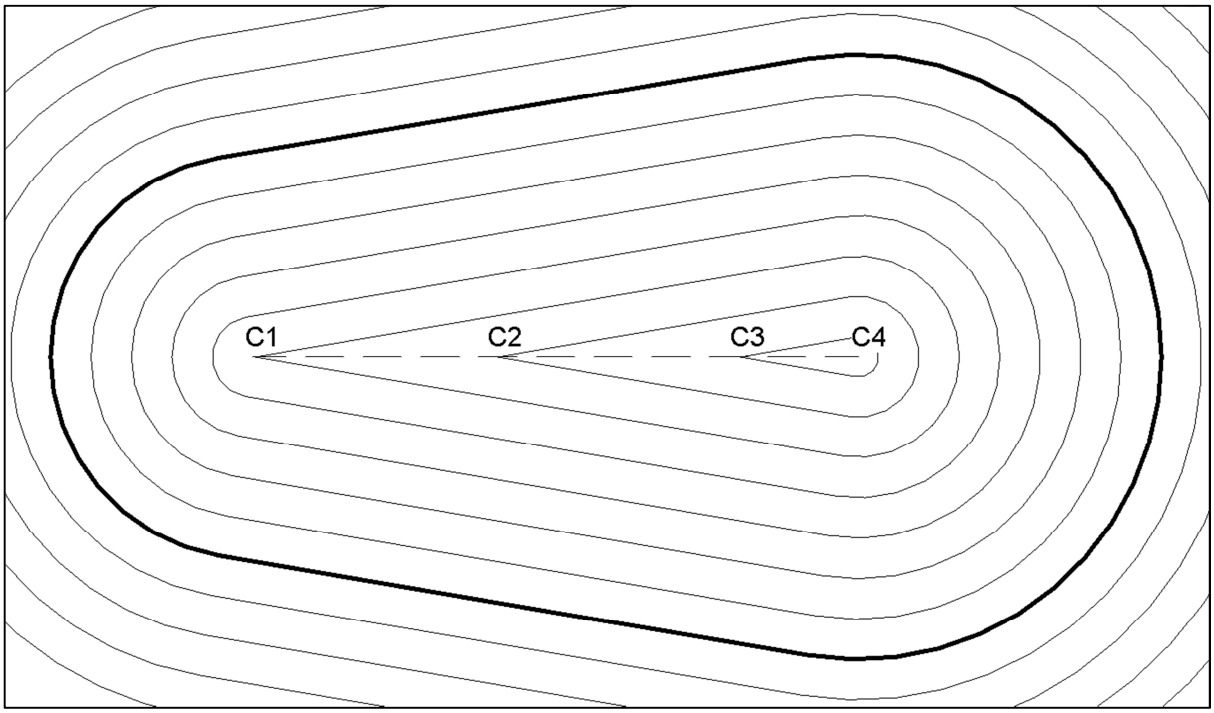


Figure 9: Propagation of waves for the formation of a medial axis

However, Generalized Voronoi Graphs (GVG) are based on Generalized Voronoi Diagrams (GVD), in which a plane is divided into cells and corresponding points (pi), and the Euclidian distance of this point pi to q is shorter that the Euclidian distance of any other point to q. Every point lies in one cell. A Voronoi diagram is constructed by drawing perpendicular bisectors between two points (called Voronoi sites). (Souvaine, et al., 2004) A corresponding GVG is constructed from a GVD with the addition of vertices in every intersection and at the end of bisectors (Wallgrün, 2005). We can observe the S-MAT and Voronoi diagrams (without edges from concave vertices) in Figure 10. As stated by Lee (1982), the medial axis transform is the same as the set of Voronoi edges without the edges from concave vertices.

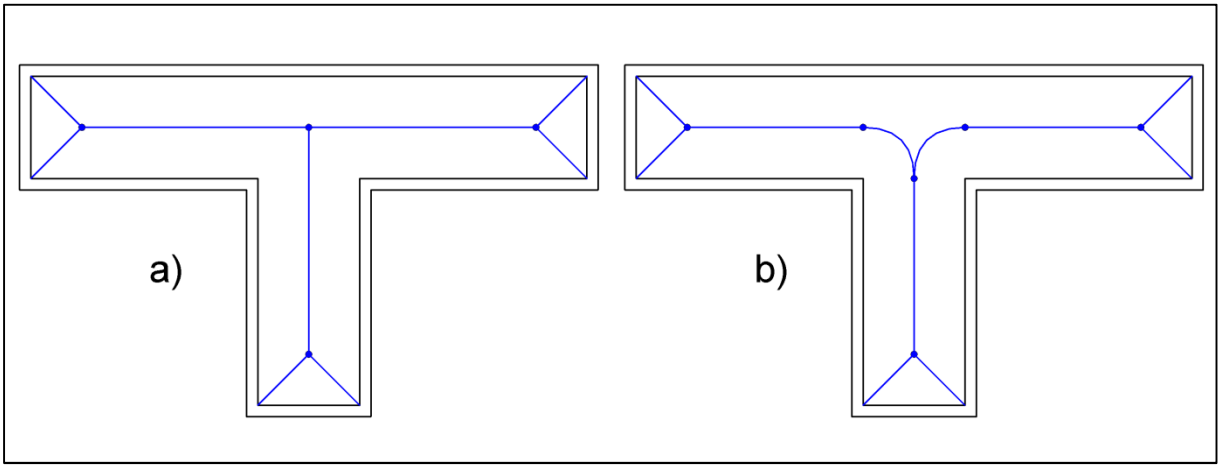


Figure 10: Diagrams a) S-MAT and b) schema of Voronoi/MAT

The first two methods contain points, lines and parabolic curves, and require significant memory and computational power to be effective. Therefore, a simplification with only linear elements in a graph network was needed. Under the last point the S-MAT algorithm was developed by (Lee, 2004), who modified another algorithm developed by (Yao & Rokne, 1991). This algorithm creates a 3D topology network with straight line elements, no curved elements are used. Another algorithm specifically for the GTN that excludes curved elements and is based on MAT was also developed. This is the Modified Medial Axis Transform (modified-MAT), which was created to address the shortcomings that have been identified when using the S-MAT though generating some shortcomings of its own. The S-MAT algorithm creates bisectors of all convex vertices forming regions with the edges of the polygon. In the next step, bisectors are created from the intersection points of the first level bisectors, forming

second level bisectors. The drawback of this approach is that because bisectors are created only from convex vertices if there is a connection at a concave vertex that algorithm gets stuck. Modified-MAT addresses this issue by constructing bisectors from all the concave and convex vertices. However, the parabolic bisector of a concave vertex is replaced by two perpendicular bisectors. These two perpendicular bisectors are not a part of the medial axis of the polygon and are merely used for determining the nodes of the medial axis where these perpendicular bisectors intersect with other bisectors. In Figure 11 the perpendicular bisectors are depicted in red and marked from b1 to b4. The segments added to the medial axis are blue and marked as s1 and s2.

Building data is needed to create a GTN with either of the above mentioned algorithms. The IFC format was used with the OpenIFCTools Java parser that creates

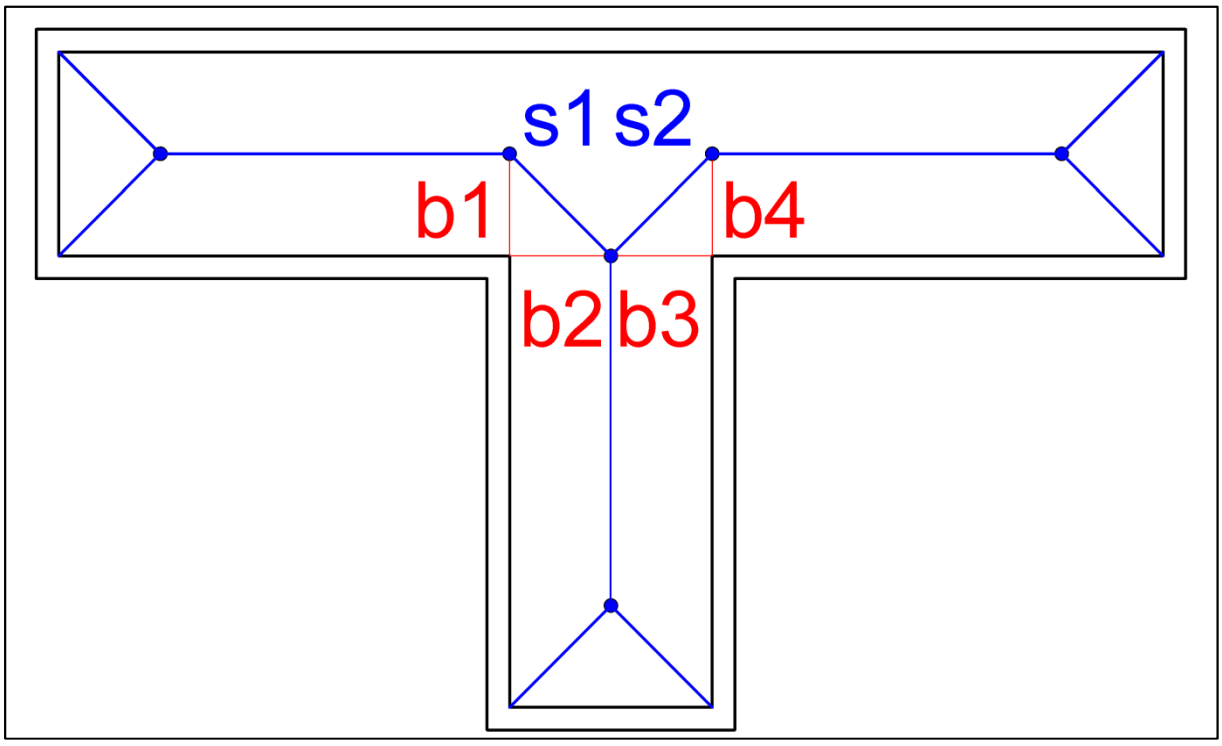


Figure 11: Modified S-MAT medial axis of the polygon

Java objects from IFC elements in the EXPRESS schema. The elements of IFC are used to determine the vertical structure of the building, in particular, which spaces belong to a given floor. This way an individual GTN is created for every floor. The topology of floors is extracted from *IfcRelSpaceBoundary* and dimensionally

weighted to accurately represent the distances of route segments. The geometry for dimensional weighting comes from *IfcSpace* objects.

2.3.2 Sparse Navigation Graph

This graph was specifically designed for pedestrian simulations (Kneidl, et al., 2012) and its unique purpose is to allow pedestrian behaviour simulations to be based on navigation algorithms that run on it. Therefore, in comparison to GTN, which is based on Voronoi type graphs, the Sparse Navigation Graph is based on visibility graphs to mimic human behaviour more closely. Working with Voronoi graphs as a base produces graphs with vertices equidistant from the obstacles, while visibility graphs' vertices take a more natural position closer to obstacle corners and edges. The basic principle behind a visibility graph is that it consists of vertices defined by sources, destinations and obstacles. Two vertices count as being connected if there is an unobscured line of sight between them. This creates a very dense and, consequently, a computationally intensive graph. The Sparse Navigation Graph tackles the issue by placing the nodes at a certain distance from each other and relinquishing the excessive nodes, thereby creating a sparser and more manageable graph.

The generation of Sparse Navigation Graphs is based on given geometry and is achieved in multiple steps, as follows:

1. Derivation of navigation points (vertices) for two types of obstacles: one-dimensional (e.g. wall) and two-dimensional (e.g. house), see Figure 12– navigation points are placed around convex corners of obstacles under the following criteria:
    a. the line of sight between the vertex and the corresponding corner must not be obstructed (e.g. obstacle is next to a wall) – points are relocated or not placed,
    b. redundant points are merged – points belonging to the same corner that are too close to each other are merged.

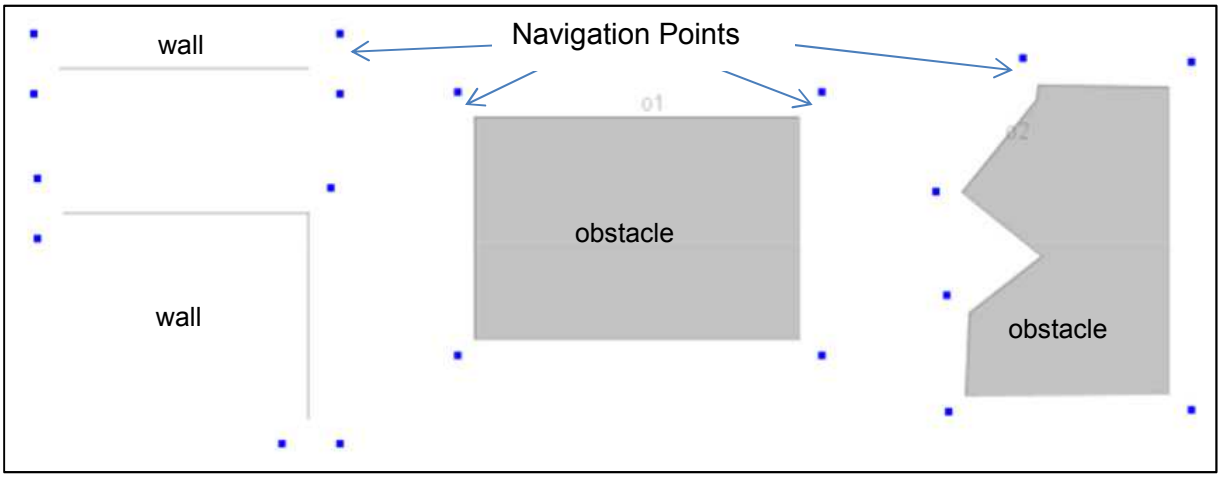


Figure 12: Navigation Point placement for walls and obstacles (Kneidl, et al., 2012)

2. Connecting vertices (cone based search method) according to the following criteria: line of sight, and avoiding redundant edges (edges that form loops which do not lead to the desired destination). The algorithm starts with an arbitrary vertex vi for which the following steps are taken:
    a. define a rectangular search area containing any obstacles that obstruct line of sight between vi and reachable destinations, Figure 13 (a),
    b. conduct search for all vertices within the sight of vi, sorted by their distance to vi, and create an edge for the closest vertex, Figure 13 (b),
    c. define a cone-shaped area around the edge with an angle α_{cone} and subtract from the original search area, Figure 13 (c),
    d. choose the next closest vertex to vi and repeat the procedure, Figure 13 (d).
3. Connectivity check – discards vertices along with the corresponding edges that are not connected to any of the sources or destinations. These vertexes are identified by a breadth-first iterator, which starts at each source vertex and establishes whether at least one destination can be reached. In Figure 14 a connectivity check on the left image discards the outer graph because it does not contain the source and destination.

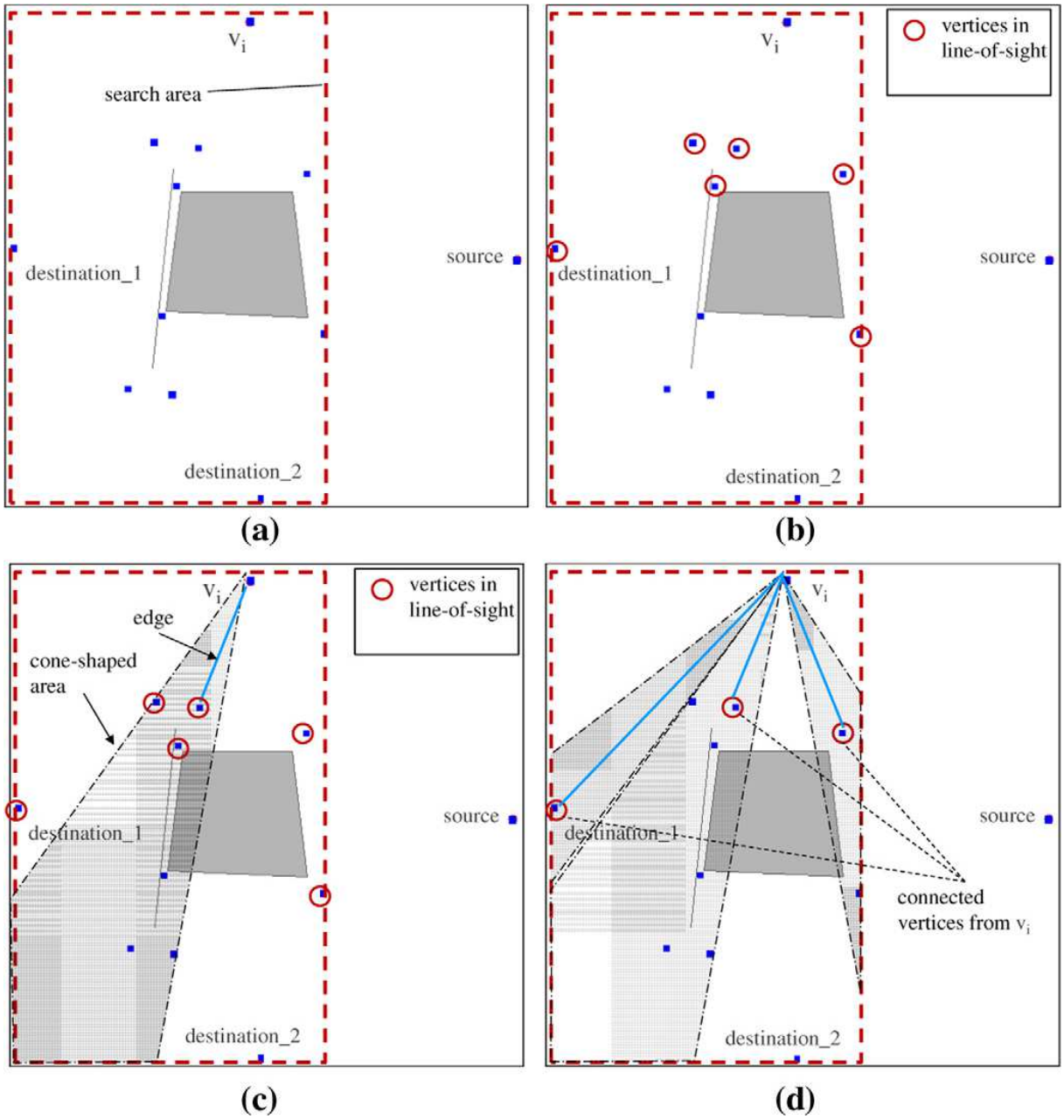


Figure 13: Cone based search algorithm (Kneidl, et al., 2012)

The cone based search method connects vertices to their most relevant neighbours. The angle between the outgoing edges has to be greater than a certain threshold. If the angle is smaller the longer edge can be discarded.

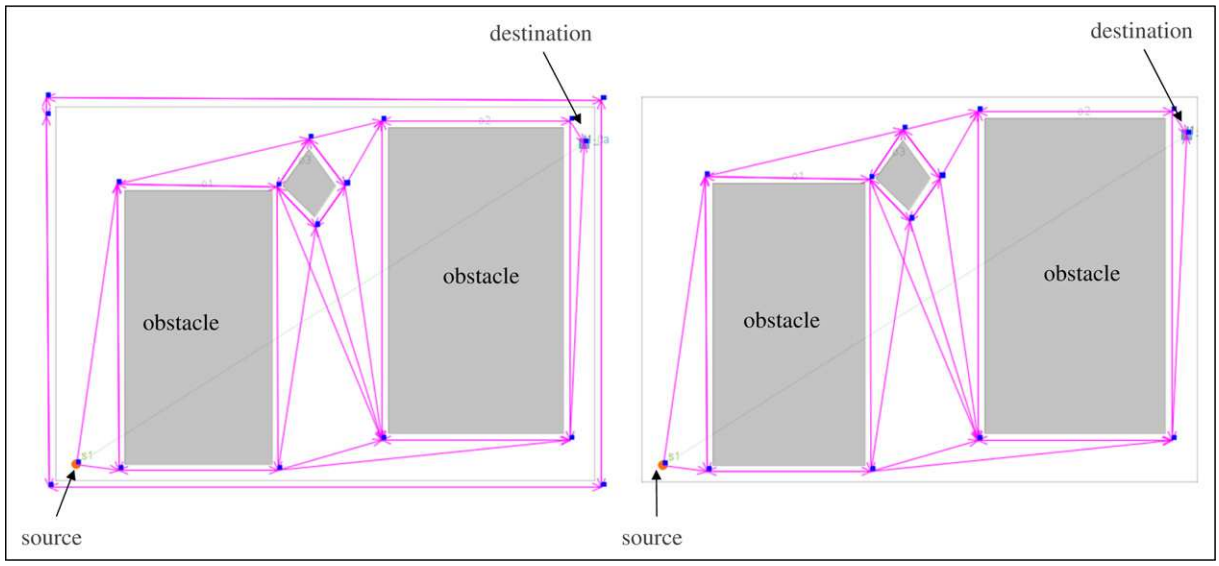


Figure 14: A Connectivity check on the left image results in the right image (Kneidl, et al., 2012)

By utilizing this graph in pedestrian simulations the individual pedestrian gains large scale orientation and thus can react to occurrences farther away. When the pedestrian has knowledge of other possible routes the, e. g., congestion can be avoided by choosing another route. (Kneidl, et al., 2012)

2.3.3 Multilayered Space-Event Model for Navigation in Indoor Spaces

As the name suggests, this approach focuses on a model with a crucial differentiation between different space models, e. g., topographic and sensor space (see Figure 15). Individual spaces are decomposed into smaller units according to respective semantics with no influence on other space representations (Becker, et al., 2009). The layers themselves are connected, which enables them to form joint states that enable, e. g., localization and route planning.

The differentiation into layers is based on various notions of space (movement space, activity space, visual space etc.) and is therefore unbounded. The two layers implemented for indoor positioning are the following:

1. Topographic space/layer – represents the interior of buildings and their decomposition into building elements (rooms, doors); facilitates route planning,

2. Sensor space/layer – represents sensor coverage areas (transmission ranges of sensors) with voids for areas where there is no sensor coverage; facilitates localization.

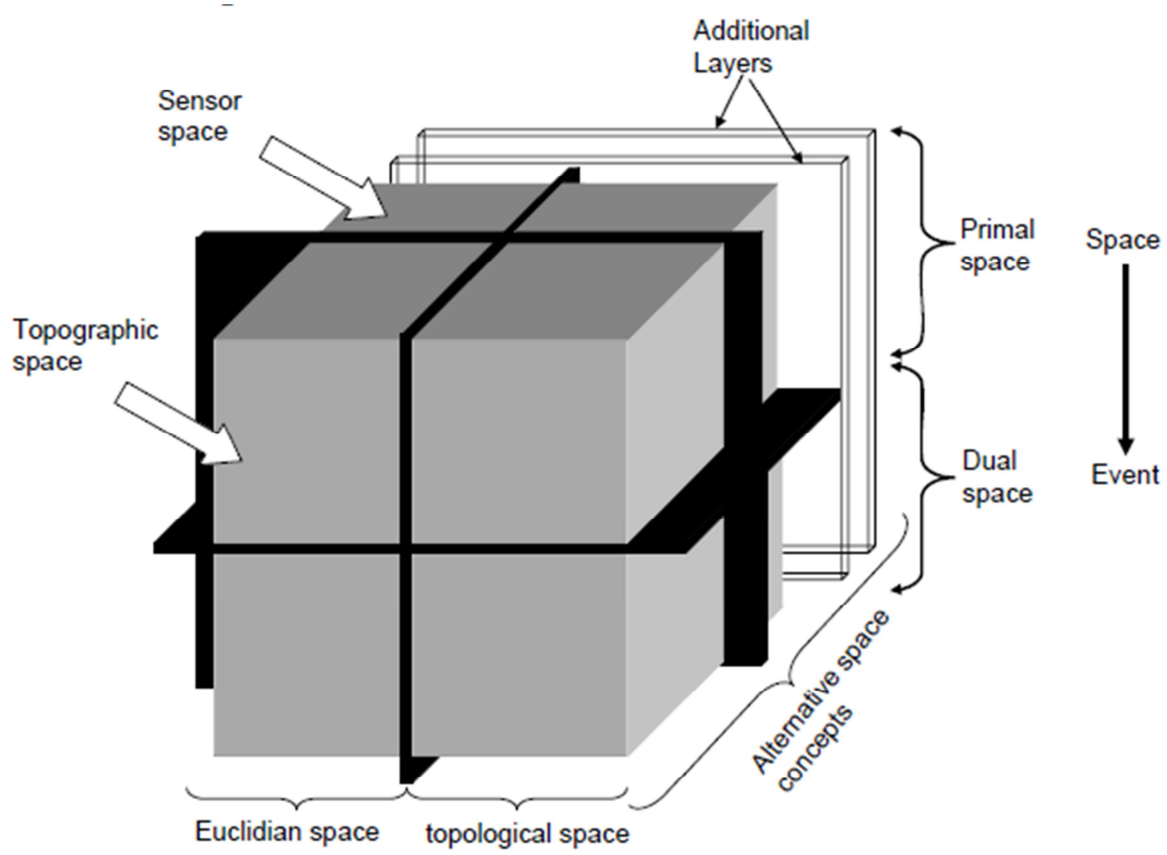


Figure 15: Representation of the model with alternative space concepts (Becker, et al., 2009)

The notion of sensor space and topographic space is completely different. Where sensor space focuses on sensor signal strengths and propagation, topographic space focuses on the representation of space geo-objects such as buildings, rooms, walls etc., using semantic 3D building objects (Becker, et al., 2009). For scenarios such as planning a fire escape the room can be fragmented into cells resulting in a cell to cell connectivity graph. Together, the two layers facilitate navigation.

Poincaré duality is utilized to describe the spatial relationships between 3D objects inside layers/spaces. As seen in Figure 16, objects in primal space are mapped to their counterparts defined in dual space, i.e., 3D cells in primal space are mapped to vertices (0D) in dual space, and the adjacency relationships between 3D cells are transformed into edges connecting the vertices.

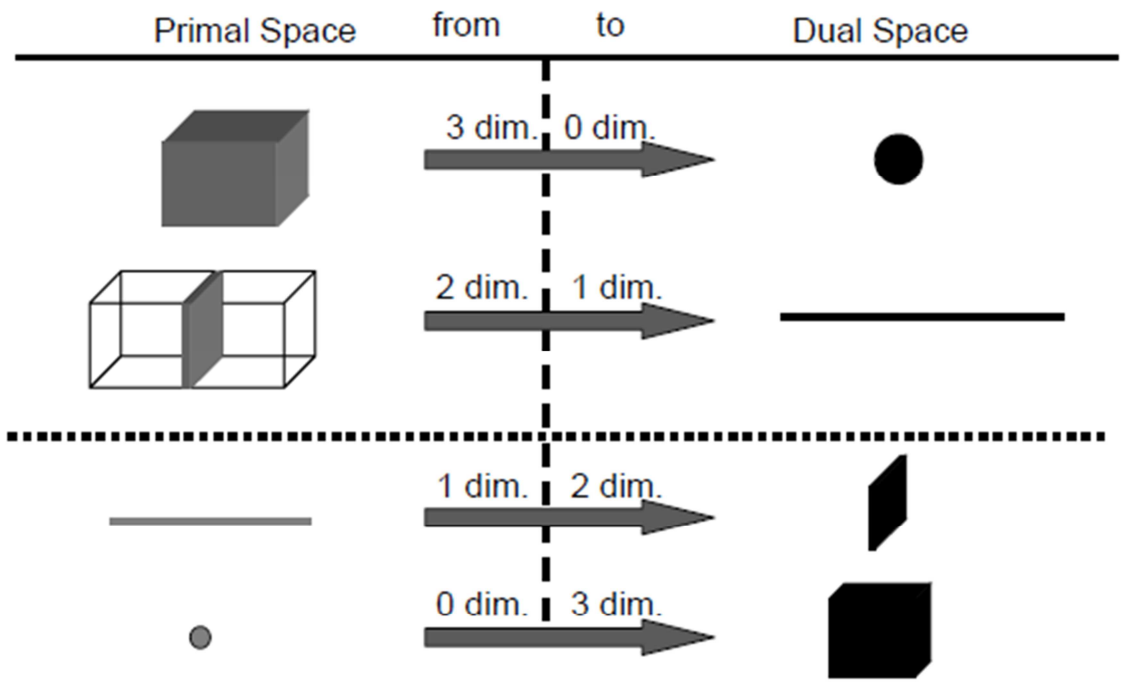


Figure 16: Principles of Poincaré duality for primal and dual space (Becker, et al., 2009)

This node edge relation structure constitutes a graph where nodes represent possible states and edges represent transitions between these states. For a navigating subject only one state can be active at a certain point in time in every layer. Only certain combinations of states are possible between layers, and these are represented by additional edges (joint-state edges), which are derived from the pair-wise intersection of respective geometries between different layers.

2.3.4 Occupancy octree of building interiors

For the purpose of global path planning in 4D environments, as a means of determining efficient routes for equipment movements in dynamic environments (Figure 17), specifically at building construction sites, an octree structure was utilized by Semenov, et al. (2012). The octree is not a building model by itself; it is merely a different representation of an IFC building model. All building elements provided by an IFC file are converted into a three-coloured octree, consisting of black, grey and white octants. The colours determine the occupancy status of individual octants where black means entirely full, grey partially full and white entirely empty. Single octants are additionally equipped with distance field values.

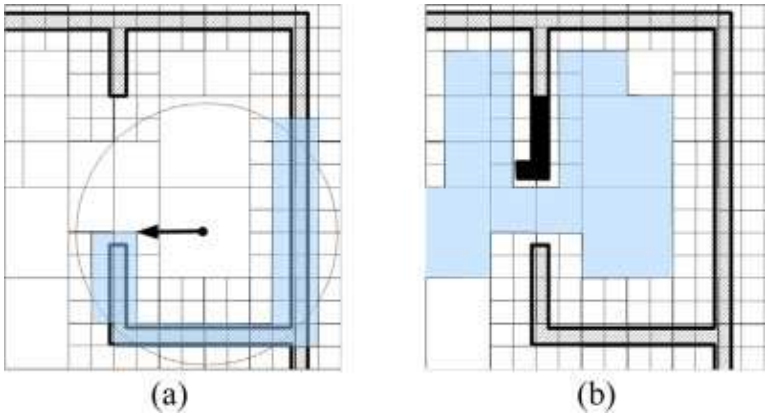


Figure 17: Change in pseudo-dynamic environment reflected in the octree (Semenov, et al., 2012)

After determining the regions of octants that are empty, these regions are classified as either spaces or gates. The differentiation between them is that spaces are large wide free areas and gates are narrow small free areas. Assigning octants to spaces

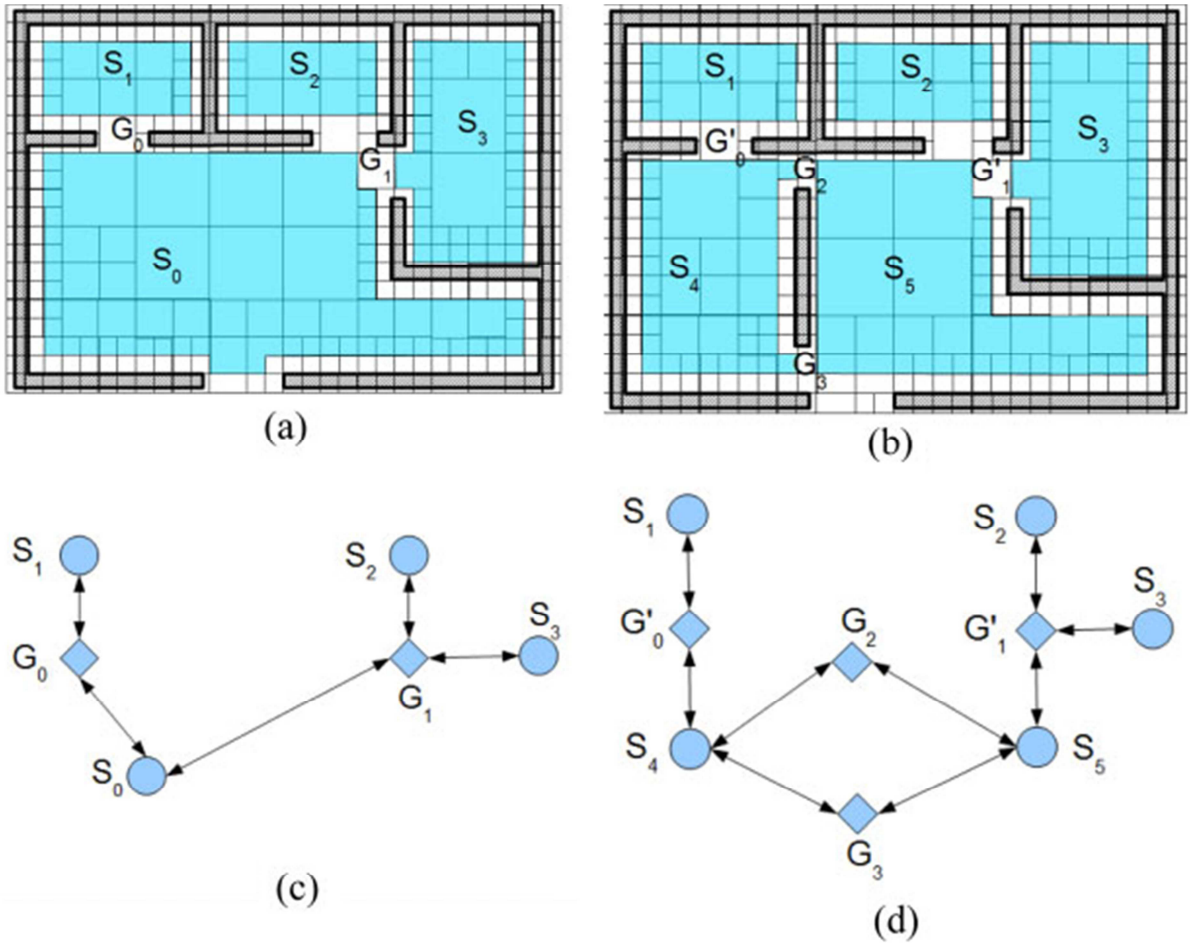


Figure 18: Original metric representations (a),(b) and their topological counterparts (c),(d) (Semenov, et al., 2012)

follows the octants' distance field values that are under a local maximum. If an octant can be assigned to two different spaces it is classified as a gate.

A bipartite topological graph is generated from gates and spaces, with one type of vertices representing spaces and the other type representing gates (see Figure 18). Space vertices are connected to gate vertices along their edges, but are not connected to space vertices. Similarly, gate vertices are connected only to space vertices, resulting in alternating space and gate vertices when travelling the graph.

2.3.5 IPS XML schema Building Information Model

In the frame of the main research project described in 2.2.3, from the Institute of Building Informatics at the Graz University of Technology, the Computer Aided Disaster Management (CADMS) and the follow up Autonomous Indoor Outdoor Navigation (AIONAV) project, the need for a building model that would help with navigation and positioning became apparent relatively early. The way CADMS was envisioned as a central and basic part of the system was as Indoor Positioning and Navigation for the better coordination of rescue forces. One important condition that needs to be met is that the system is required to be autonomous in all aspects. This means that it should not require any fixed installations and should be mobile with minimal requirements for the system set up. Ideally, the entire system would be wearable by the end user and a command centre set up in the rescue vehicle, which would mean complete autonomy from the existing infrastructure.

Professional use of the system is by no means the only possible option of using it. Among the options for personal use, the concept of a positioning and navigation application for the blind and visually impaired was developed and tested in collaboration with FH Joanneum (Kiers, et al., 2011). Other options that were developed and tested include a shopping application and a botanical garden guide application from the Karl Franzens University Graz. The positioning and navigation experience gained from research done on CADMS was used in other projects such as NAVCOM, ways4all and ways4all complete (ways4all, 2014), as well as OnPoi (ONPOI, 2014).

The purpose of a Building Information Model in Indoor Positioning and Navigation is specific. It is comprised of the four following sub models, each responsible for its own domain (Bernoulli, et al., 2008):

1. Graphical Representation Model – models' available data in displayable formats,
2. Accessibility Model – models' physical transitions between rooms, floors and surroundings,
3. Structural Model – simplified model (not fully integrated),
4. Building-Linked Data Model – manages data linked to the building model, not attached to previous sub models.

The BIM acts as a base for displaying and putting data from positioning and navigation algorithms (Bernoulli, et al., 2010) in context, thus creating information that can be read and interpreted by a certain user for either personal or professional use. The BIM also defines topology, which is essential in navigating through the interior of buildings. However, topology cannot be directly referenced from most building models. Among other reasons, the fact mentioned has led to the development of a specialized and purposely built data format that is supposed to be used with indoor positioning and navigation algorithms developed at the Institute of Building Informatics at the TUG. The format is a subset of a complete BIM, featuring only the required building component and relationship classes. The model focuses on defining areas (rooms) for valid locations and connections between these areas. Currently, no obstacles are yet defined, but are certainly needed to complete the model. Focusing on the geometrical definition of areas and the connections to provide topology information enables the model to be very efficient in terms of required space. Additionally, this enables the size of the model to be relatively small compared to traditional complete building models and, hence, enables its use in mobile devices since system resource requirements are kept low.

The model (an example IPS XML schema code cut-out is Code 1) itself consists of individual objects (Bernoulli, et al., 2010; Muhic, et al., 2012), which can be divided into two groups. The objects from the first group store geometrical information (coordinates, in particular) and define boundaries. This group contains two classes called the Point and the Line class:

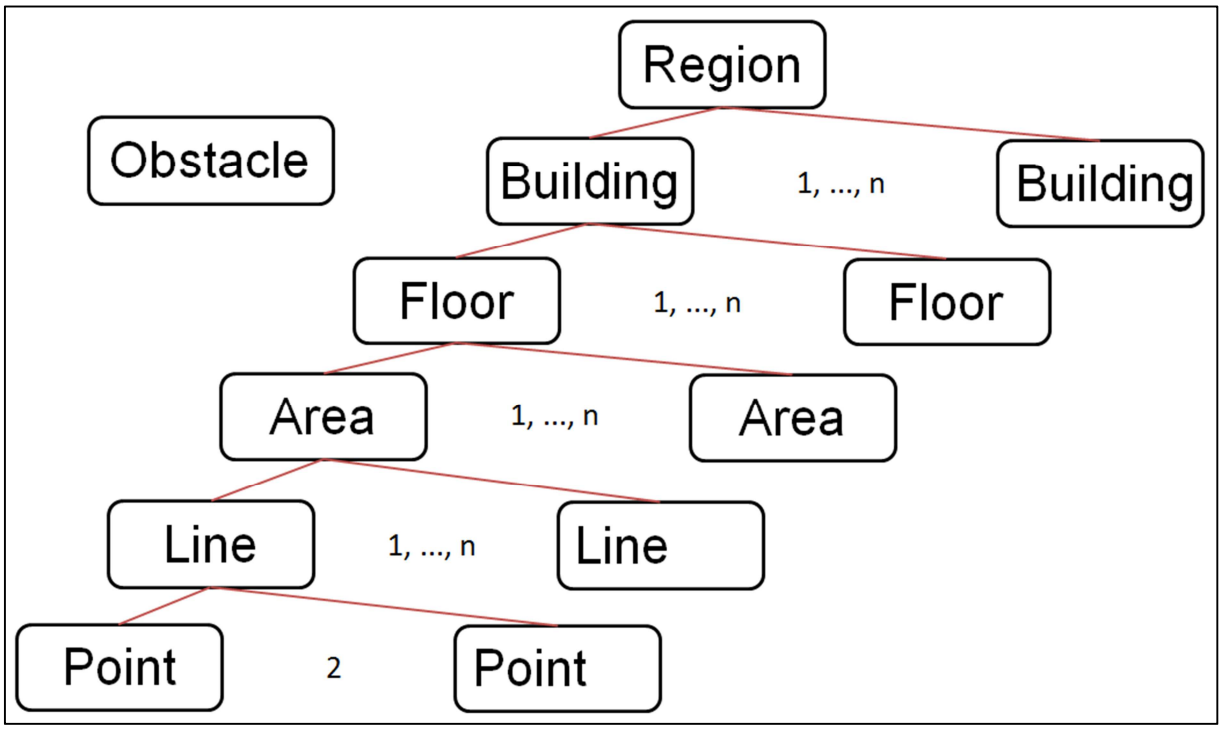


Figure 19: Simplified diagram of the IPS XML

1. Point – stores the X and Y coordinates that are geo-referenced and form the baseline geometrical shape of the building. All other objects directly or indirectly reference Point objects for coordinates.
2. Line – consists of two Point objects. It forms the edges of other higher ranked objects. Several connected Line objects form polygons, which in turn form object boundaries.

These two classes (Point and Line) form the basis on which the model is built, in terms of geometrical data.

The second group is comprised of complex classes that are created from lower ranked objects, forming a hierarchy of classes (in the IPS BIM), which define the framework of the building model:

1. Area – the Area class uses Line objects to define its boundary. A series of Line objects form a polygon that describes the boundary of an Area.
2. Connection – another class formed from Line objects. Two Line objects form a LinePair class that serves as the basis for the Connection class, which defines transitional segments between Area objects.

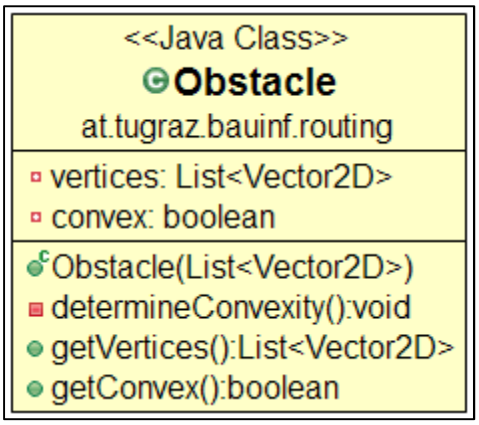


Figure 20: Obstacle class proposal

3. Floor - Floors are comprised of a number of Areas and completely define the level of a building. Additionally, the Floor class usually contains the information about height.
4. Building – the Building class is comprised of one or more Floor objects and wraps up the building model. Looking strictly from the point of view of indoor positioning, the Building class contains all of the required information.
5. Region – the Region class describes a Building object's surroundings and can contain more than one Building object. The Region class can also be described as the emergency site.

The overall framework of the building model is very similar to other types of BIM formats that also use related hierarchical structures. A simplified diagram is presented in Figure 19, showing the mentioned classes along with the proposed Obstacle class (see Figure 20) that should be able to cover the requirements for any objects inside Areas. For testing purposes, the Obstacle class incorporates geometry in the form of a list of vertices (IPS XML class Vector2D) and a test of Convexity that is determined the instant an Obstacle object is created. The consequence of framework similarity is that geometrical data can be extracted; transferred and reused in a different format in a fairly straightforward manner, but only if we start from the ground-up.

The most crucial information about topology is stored in two higher ranked classes; the Area class and the Connection class. Both of these are based on the two basic classes, the Point and the Line. An Area defines a topological space, while a Connection defines the crossing between two Areas. A Connection benefits from each Area having a boundary of Lines as boundary segments. Where two different Areas have overlapping Lines as boundary segments, a Connection is established and a Connection object defined.

Every single object in the model has a unique ID that differentiates it from the rest. This way objects can be easily differentiated between and identified in the case of a query. Essentially, the unique ID can be a random integer number, but there are certain rules as to how these integers are kept unique globally. For example, two separate Area objects can have boundary segments (Lines) with the same integers as IDs. These Lines, however, still have different global IDs. The way the IDs work is as follows; every object that is lower ranked in the hierarchy (e.g. Point is bellow Line as Line is bellow Area) has a global ID that consists of the ID of higher ranked objects that it belongs to followed by the ID of the object in question. In other words, two Lines from different Areas with the same line ID (e.g. line ID is 001) will have a different global ID, because they belong to two separate Areas. The individual IDs are separated by a point-".". Examples of this can be found in the following chapters describing the individual classes of the IPS XML schema and in (Code 1). A general overview of the model will be provided, however, the descriptions will not deal with the specifics of the program code of individual classes or the IPS XML schema. A more thorough description of the IPS XML building model and schema and the source of information for this overview will be available by its author Thomas Ber-

```
<ips:region xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ips="http://bauinformatik.tugraz.at/ips-bim-editable" id="0">
  <ips:building id="0">
    <ips:floor id="60">
      <ips:area type="flat" height="0" id="80">
        <ips:line id="0">
           <ips:point xs:type="ips:EditablePoint2D" x="-5938.14776379652"
            y="1169.00378005655" id="0"/>
           <ips:point xs:type="ips:EditablePoint2D" x="-10938.1477637965"
             y="1169.00378005655" id="1"/>
        </ips:line>
        <ips:line id="1">
           <ips:point xs:type="ips:EditablePoint2D" x="-10938.1477637965"
            y="1169.00378005655" id="0"/>
          <ips:point xs:type="ips:EditablePoint2D" x="-10938.1477637965"
            y="-3330.99621994345" id="1"/>
        </ips:line>
        ...
      </ips:area>
      <ips:connection>
        <ips:pair first="0.0.60.182.3" second="0.0.60.182.5" type="door"/>
      </ips:connection>
    </ips:floor>
  </ips:building>
</ips:region>
```

Code 1: IPS XML schema file cutout with a Region, Building, Floor, Area, two Line and four point objects

noulli (2015) whose dissertation is in preparation.

2.3.5.1 Point

The most basic class in the IPS BIM is the *Point*. It holds most of the geometric data. The coordinates x and y can be found only in *Point* objects, however, *Points* do not hold the z coordinate. From a top-down view standpoint practically all geometrical information comes from Point objects.

As the most basic object, it is in all hierarchically higher ranked objects. It can, however, be directly referenced only from *Line* objects. Every unique *Point* also has an ID parameter that differentiates it from other *Points*.

Table 7: IPS XML schema Point class ID

Region	**Building_ID**	**Floor_ID**	**Area_ID**	**Line_ID**	**Point_ID**
001	001	001	001	001	001
Global Point_ID: 001.001.001.001.001.001					

2.3.5.2 Line

Two *Points* define a *Line* and since no other forms are currently possible an approximation has to be used for curved geometry. One of the *Points* must be the starting point and the other the ending point, creating a directed line. Other *Lines* can only be connected at either of the *Points*.

Generally, at the level of *Lines* there is no information yet about height, though in special circumstances a Line must store its own height. This happens when the *Line* defines the entrance or the exit boundary segment to a sloped *Area* object e.g. stairs, ramps, etc. In those cases the height of the lines is stored in the *Line* object.

Table 8: IPS XML schema Line class ID

Region ID	**Building_ID**	**Floor_ID**	**Area_ID**	**Line_ID**
001	001	001	001	001
Global Line_ID: 001.001.001.001.001				

2.3.5.3 Area

An *Area* class defines the type of an area in a building. It has a boundary that consists of at least three *Line* objects, which define the limits of certain properties an area in a building has. Additionally, no object inside one *Area* can cross into another *Area* or into empty space. This makes the *Area* class a topological space by definition. One of the reasons why the boundary also stores *Lines* and not only *Points* (which would be enough for the purpose of providing geometrical information) is that a particular boundary segment, besides defining the boundary of an *Area* (topological space), can have other properties as well, e.g., a segment can define a window or a door, meaning that this segment should be traversable and hence not define a boundary. However, traversing is not completely random.

The IPS BIM allows for several different types of areas that are directly connected to the needs of defining indoor movement. Essentially, we can make the following division (from the IPS BIM xml schema):

- Flat area ("flat") – a basic type suitable for most cases and most similar to a room. This type of area is defined as a constant height area and is static in the model (does not change).

- Sloped area ("ramp") – this type offers the feature of not having constant height. Two of the boundary segments can have a height assigned to them, which in turn defines the angle of the slope. This type can be used for stairs or ramps.

- Varying height area ("varying-height") – this type's height can vary, meaning that the area can "move" vertically by changing the height. The purpose is to define an elevator.

- Dynamic ramp ("ramp-moving") – a sloped area with the additional property of a movable surface. Specifically designed to define escalators.

Two more types exist ("flat-surrounding-area" and "ramp-surrounding-area"); however, they have not been implemented to date. These two types would enable an *Area* to include openings. The "flat-surrounding-area" type is for "flat" type *Areas* with

openings, and the "ramp-surrounding-area" type is for "ramp" *Areas* with openings. Currently, an *Area* with openings is modelled as two (or more) separate *Areas* (see Figure 21 and Figure 22). This is because a method of navigation (the graph model) has not yet been specified and the current method of dealing with areas allows for a wider array of options, while making the modelling more complicated to a certain degree.

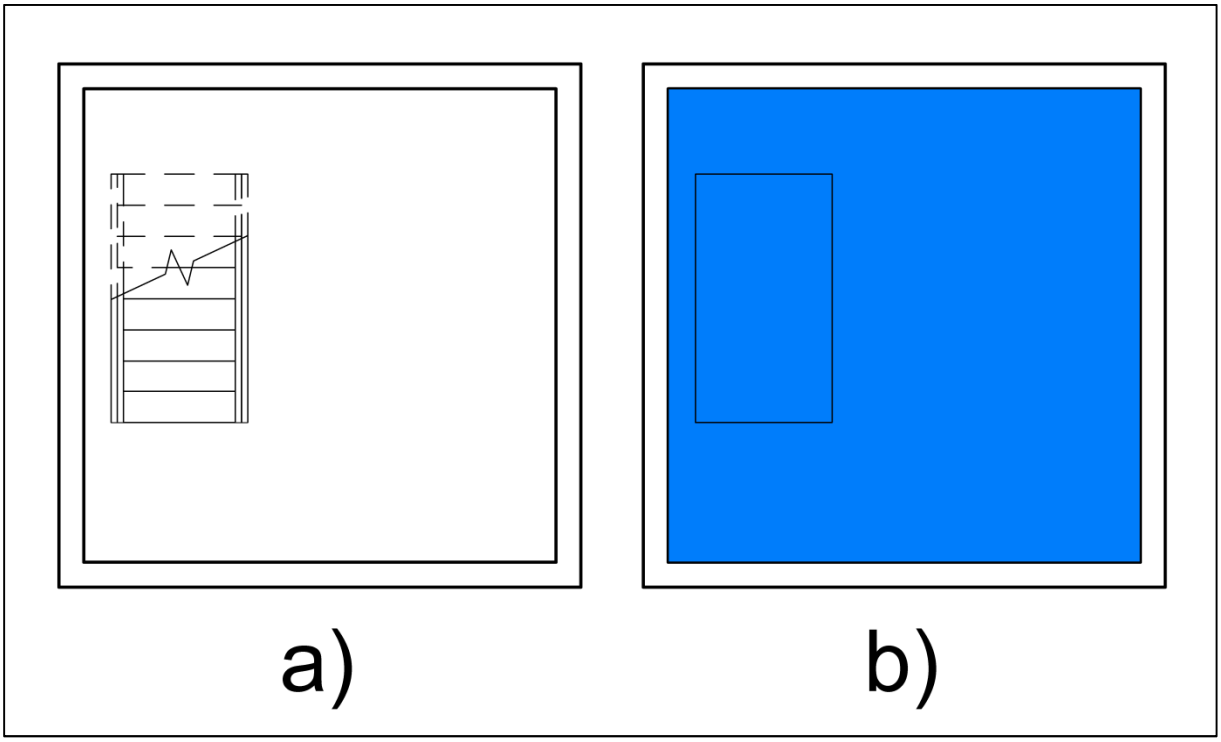


Figure 21: Space from building model and incorrect modelling technique

From a general point of view, areas can be compared to rooms or spaces (although they are not a 3D object), but cannot be identified as one of them. A particular room can consist of several different *Areas*. As an example we can take a staircase. A staircase is either enclosed in an area of a building or it is in a room. In both cases the ramp with stairs should be modelled as a separate "ramp" *Area* for the purpose of defining a slope, while the rest of the room should be a "flat" *Area*. One additional issue to be considered here is that in case the staircase is in the centre of the room the "flat" type *Area* will have to be modelled as at least two separate *Areas* to avoid the need to model *Area* openings, because of the contemporary lack of the option to model an *Area* with openings (see Figure 22). Additionally, we must not overlook the fact that a room can be of varying height. By definition, an *Area* does not vary in

height (apart from the "ramp" type that defines a slope). Consequently, each region in the room with constant height has to be modelled as a separate *Area* object.

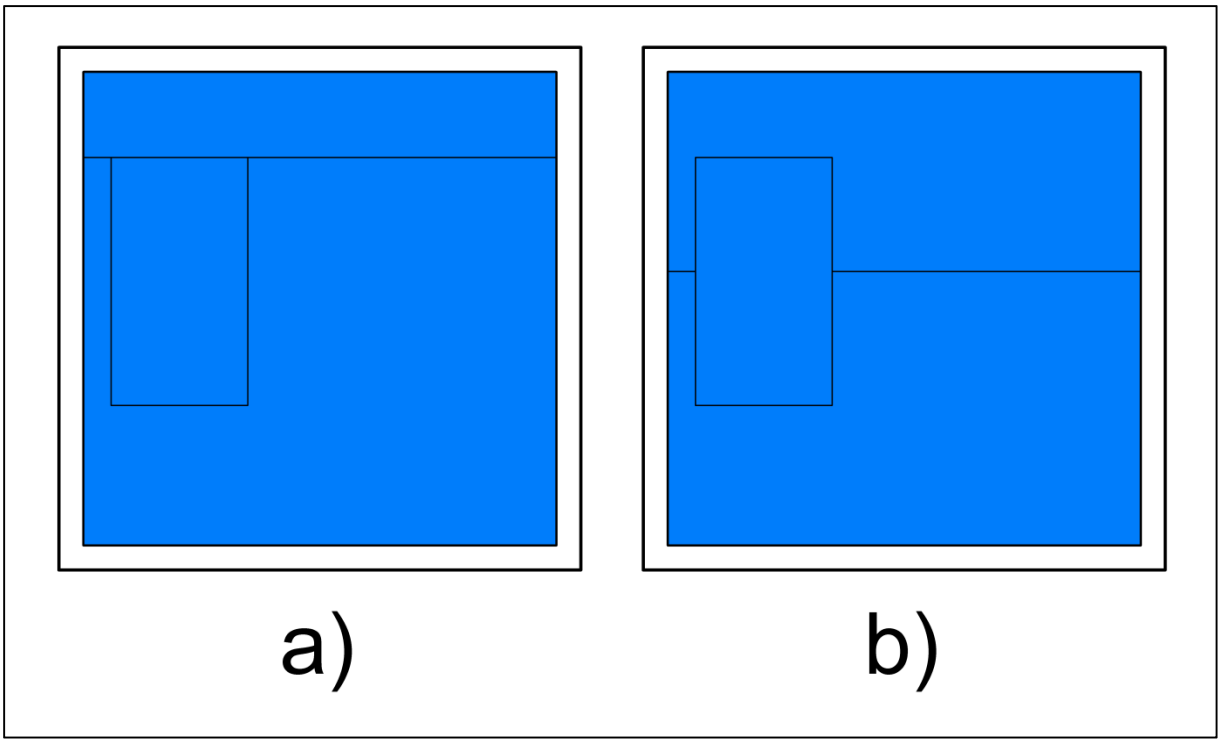


Figure 22: Two examples of correctly modelling an Area containing other elements

With the described boundary conditions more or less any building floor plan can be modelled with several floor plans composing the entire building. Because *Areas* are used as basic building blocks, the end result is a 2.5D building model, in which we have access to all three dimensions (x, y and z), while movement is restricted to 2D plains (the floor plans) and *Areas*.

Table 9: IPS XML schema Area class ID

Region ID	**Building_ID**	**Floor_ID**	**Area_ID**
001	001	001	001
Global Area_ID: 001.001.001.001			

2.3.5.4 Connection and LinePair

A *Connection* is defined by two overlapping boundary segments of two separate *Areas*. These two overlapping Lines form a *LinePair* object, which constitutes the ba-

sis of a *Connection*. The *Lines* from *LinePair* hold the information concerning which *Areas* are connected.

To cross over from one *Area* object to another or to leave an *Area* object entirely there has to be a boundary segment (a *Line* object) that is defined as a *Connection*. The following two types are possible:

1. A *Connection* with a *LinePair* of two *Lines* – for crossing over from one *Area* (topological space) to another
2. A *Connection* with only one Line – defines an exit from a *Building*, a *Connection* to the outdoors

The sole purpose of the *Connection* is to define *Area* segments that can be traversed. Therefore, it is always built on the basis of one of the two available segment types, i.e., either the *Line* or *LinePair*. Because a *Connection* is always defined on the basis of a single *Line* or a *LinePair*, it does not require an ID of its own and the ID of the underlying line segment type is used in its place. Thus, a *LinePair* object can be identified by the two IDs of the Lines that it consists of. Consequently, *Connection* objects can be grouped directly under the *Floor* object and are not assigned to individual *Area* objects, enabling the *Connection* to be accessed independently of the *Area*. After all, no single *Area* can claim ownership of a Connection since more than one *Area* is needed to form a *Connection*.

Table 10: IPS XML schema LinePair class ID

LinePair	
First Line	Second Line
Global Line _ID: 001.001.001.001.001	Global Line_ID: 001.001.001.001.001

A room comprised of more than one *Area* also requires *Connections* between these *Areas*. By default all *Areas* are closed, therefore, connections have to be created. For example, a staircase in the middle of a room, as stated in the previous chapter, would require the room to be modelled with at least two *Areas* and these *Areas* require *Connection* objects to be defined on the boundary segments where it is possible to cross over to the other *Area*, for the room to function as a single undivided zone.

2.3.5.5 Floor

Floor objects represent building storeys. All objects that can be found on a certain level of the building are organized under the common *Floor*. This means *Areas* and *Connections*.

Floor objects are also the main carriers of height information, which coincides with the 2.5D nature of the building model where movement is allowed in 3D only on certain levels (heights) where the *Floor* is defined. *Areas* and *Connections* that belong to a particular *Floor* share the same height. This can be overridden by assigning a height to *Areas* or *Lines*.

Table 11: IPS XML schema Floor class ID

Region ID	**Building_ID**	**Floor_ID**
001	001	001
Global Floor_ID: 001.001.001		

2.3.5.6 Building

The entire building model of a single building can be tracked back to the *Building* object. By having access to all *Floor* and *Connection* objects (if they cannot be assigned to a *Floor*), all objects from this building are in the *Building* object.

Accessing any information about a specific building goes through this object. As with all previous objects, a *Building* object also has a unique ID.

Table 12: IPS XML schema Building class ID

Region ID	**Building_ID**
001	001
Global Building_ID: 001.001	

2.3.5.7 Region

Because of the nature of operations carried out by first responders, which is also the main motivation behind the development of CADMS, the operation site is not limited to buildings. Therefore, higher up the hierarchy the *Region* class takes over from

Building. This class enables several separate buildings to be in the model as well as the geo referencing of the entire site. Therefore, absolute position measuring is possible (e.g. GNSS/GPS). With the relative method of measuring positions through an IMU, this kind of addition is not only welcome, but also a requirement for the system to function seamlessly.

The *Region* is the highest class in the hierarchy and consists of one or more buildings with their accompanying environment. Consequently, all other objects can be accessed directly through the *Region*. Additionally the *Region* also maintains aerial images of the site, providing a map of the environment.

Table 13: IPS XML schema Region class ID

Region ID
001
Global Region_ID: 001

3 Acquiring building data for Indoor Positioning from building models

The methods of generating building data for Indoor Positioning described in the previous chapter focus more or less on generating geometrical and topological information about spaces resulting in a graph network of the building. The graphs created are tailored to the individual approach and needs of projects that motivated the research. Due to this fact, the resulting graphs mirror the different requirements and hence can be very dissimilar despite following the same goal. Additionally, depicting a building with a graph is in the typical use case not enough as additional semantic information is required e.g. LBS – Location Based Services, interior objects like furniture etc. The IPS XML format already tries to address this issue somewhat by providing a complete building model in a scaled down format to fit into mobile devices.

Therefore, a more general approach was desired that would satisfy all the individual needs of any method. A standardized Indoor Positioning building data format would unify the data source for creating graph networks as well as enable the usage of the model for other purposes like a graphical user interface or the carrier of semantic information. Several requirements have to be met however, for the format to be effective and data generally accessible:

1. Building data has to come from a standardized building information model format to eliminate the need for translator applications.
2. Topology and geometrical information should be readily available and easily accessible.
3. The format should be compact to run on mobile devices.
4. A modular structure would enable to tailor the format to individual needs of different users.

Taking IPS XML as a reference a method was developed to feed data from IFC to a converter and create an IPS XML model from the IFC format. With additional research basically all relevant scenarios are covered and will be looked into and described in the following chapters. But first it has to be established what kind of methods can be used to even create the IPS building model according to the IPS XML

schema. These methods range all the way from manual creation to the automatic data extraction from IFC that will be discussed later.

3.1 Creating the IPS BIM

All IPS BIM classes described in chapter 2.3.5 are coded in Java and compose the IPS BIM. The format in which data is stored and can be transferred is XML. An XML schema defines the structure of the XML file.

With the information above, the foundation is set to create usable information for the CADMS, especially the indoor positioning component of CADMS. However, the question remains of how to create real building data and information. Two different options can be identified: either existing building information is used, or building information is created from scratch or from certain sources such as fire escape plans, non-digitalized drawings of floor plans or even on-site laser measurements of the interior of buildings. Site data is easier to come by with a ready supply of aerial or satellite images from various commercial or open code providers. Additionally, it has to be mentioned that this data can be used directly; no special conversion is required as in the case of building data from building models.

For the purpose of creating an IPS BIM from scratch, an Editor application has been developed that also enables the coupling of building data with site data in a sense geo referencing the IPS BIM. As a source of existing building data it can be differentiated from CAD and CAFM (Wießflecker, 2009), on the one hand, or BIM on the other hand. This differentiation has been made because of the nature of the input data. While CAD and CAFM data is generally a two dimensional geometry of individual building floor plans, a three dimensional building model (referring to a BIM) provides not only 3D geometry but also semantic information for all building blocks and objects being used. In general, this means that no manual input is required to create an IPS BIM in the case of an available BIM, whereas in the case of CAD or CAFM floor plans the process cannot be automatized and requires manual creation of the IPS BIM. Naturally, it would be more convenient to use a BIM every time, but there are issues that have to be overcome for this to become a viable option. First off all, not only do most current buildings lack a building information model, they also do not have available digitized and vectorised floor plans. Secondly, even new buildings

being constructed suffer from the same handicap, because despite the availability of the required technology the majority of countries lack a common act or document to regulate this field. Therefore, the "classical" data sources must also be considered. Shifts in the recent years, however (adoption of BIM acts in the USA, several European countries, Singapore etc.), promise that more and more buildings will be built with BIM in mind, and as a consequence, quality building information will become more widely available.

The following chapters will look into the methods of creating an IPS BIM with special focus on data extraction from IFC.

3.1.1 The Editor and the manual method of creating an IPS BIM

A new approach to data management for indoor positioning brought along the need for new software that would help in creating the necessary data. Therefore, the IPS Editor application was developed. It is a relatively simple tool that enables the import of data from other sources, the manual creation of an IPS BIM, and the possibility of editing existing data. The Editor deals with creating geometry by drawing Area and Connection objects on levels (Floors) and assigning each object a particular type. The Editor also enables the coupling of aerial or satellite images that are geo referenced with the building model. The format in which the data is saved is an XML file that uses the IPS XML schema.

With manual creation of building models being the primary function of the Editor, a comparison of the advantages and disadvantages of this method can be made. While offering independence from all other sources, manually creating an IPS BIM is typically redundant because the data is either stored digitally or on paper in the form of floor plan drawings or in the worst case only a constructed building exists. Sadly, not all forms of data can be automatically extracted and used to create the IPS BIM. Therefore, in certain cases, such as when there are no available floor plans or they are available but not in digital form, at least some level of manual input is required. For example, in the ideal case of having digitized and vectorised floor plans, manual input is still required to define the type of Area (e.g. RAMP, FLAT etc.) and to define the Connections between Areas. Hence, in the current state, where building data can be available in a number of different forms, an Editor is required in every case.

The developed Editor is flexible and allows for the import of existing geometry data in the form of DXF, which can then be used as an underlay to define the geometry of Areas. The drawing of objects, such as Area and Connection, however, still has to be performed manually. The user interface will eventually serve the purpose of IFC import as well.

3.1.2 Creating the IPS BIM from CAFM data

Because of the relative complexity and the lack of broader acceptance of BIM formats, Wießflecker (2009) has been looking for other ways of utilizing existing building data for the purpose of creating building information viable for indoor positioning. His suggestion involved databases developed for Computer Aided Facility Management (short CAFM) as the obvious choice. These databases already made use of the geometry and types of Areas that could be taken over to the IPS BIM. In the case of Wießflecker, the editor for creating building models and the building model of the commercial application Speedikon were used. Despite the completeness of the Speedikon model for facility management uses, certain adjustments had to be made to fashion a model that could be used for testing and for use in the indoor positioning field. For example, the Speedikon model does incorporate doors, but these objects are very complex and carry a lot of information that is of no particular use for indoor positioning, therefore, a new connection (named Übergang – transition) object was added by the company Speedikon with the specific purpose of connecting two Areas or an Area to the outdoors, thus providing information on which Areas are accessible through a particular connection.

With these required objects, an environment inside Speedikon has been generated that can be used to create the IPS BIM from an existing facility management database. The graphic data usually comes from CAD floor plans, while information about rooms and connections is provided by the facility management data. This combined model with certain additions, which can be added inside the Speedikon database and are particularly required for indoor positioning, can then be exported, according to the IPS XML schema, and applied to cases of indoor positioning use.

Several cases have been tested with this method. Existing floor plans from different sources, either vector formats like DWG, or DXF file formats, or even pixel formats that were vectorised, have been used for various buildings and building sites in Austria, Switzerland and Germany. The model presented below is of a military complex in Walenstadt in Switzerland. The entire complex was modelled in Speedikon (see Figure 23) using a DWG template. The complex features several buildings and even an underground passage.

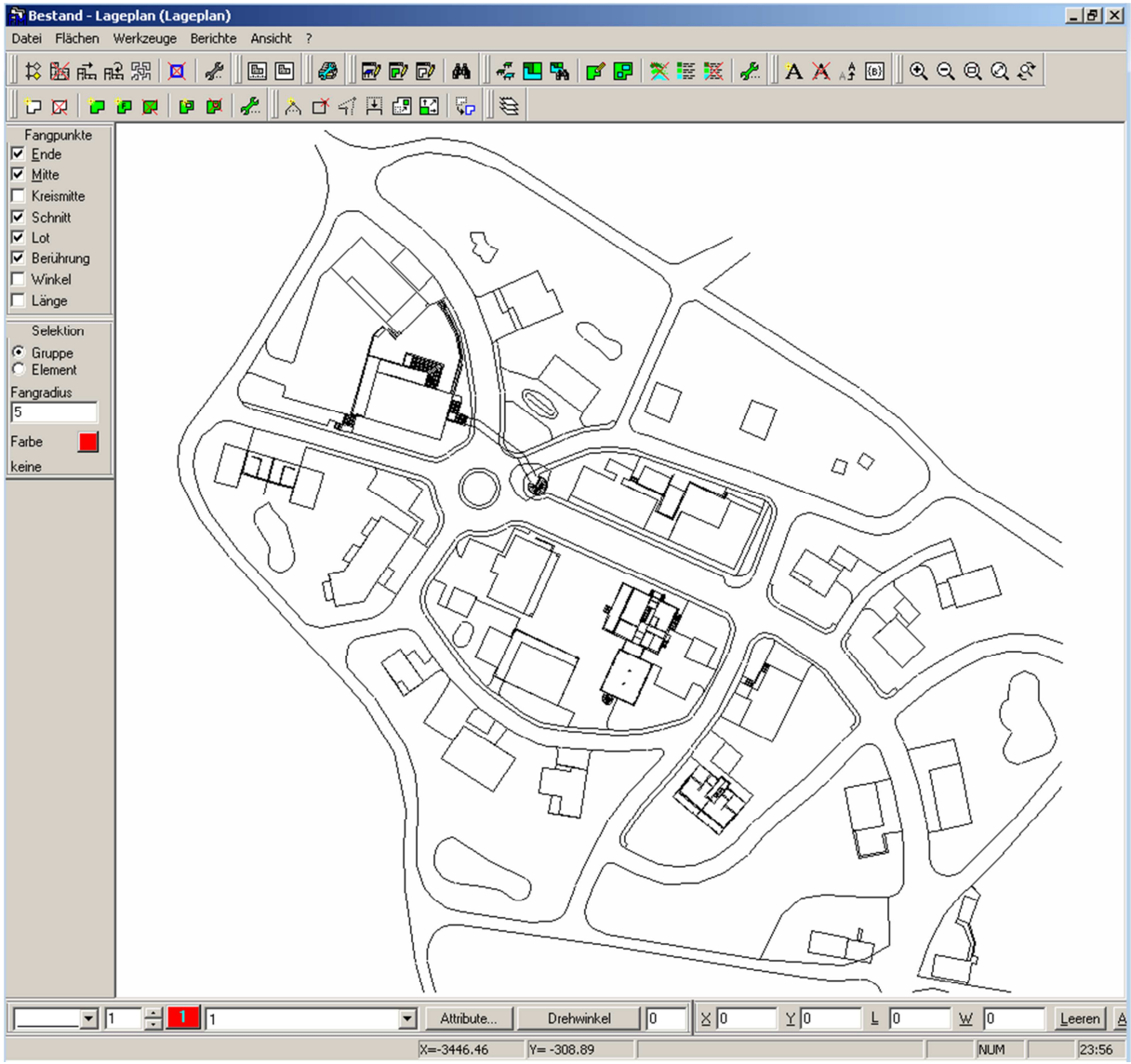


Figure 23: Model of the Walenstadt military complex in Speedikon

The final version of the model that was used in AIONAV can be observed in Figure 24. This model was acquired directly from Speedikon through an XML file export, re-

sulting in an IPS XML file. The tedious process of several intermediate steps can be considered a big part of the motivation for the research in this thesis.

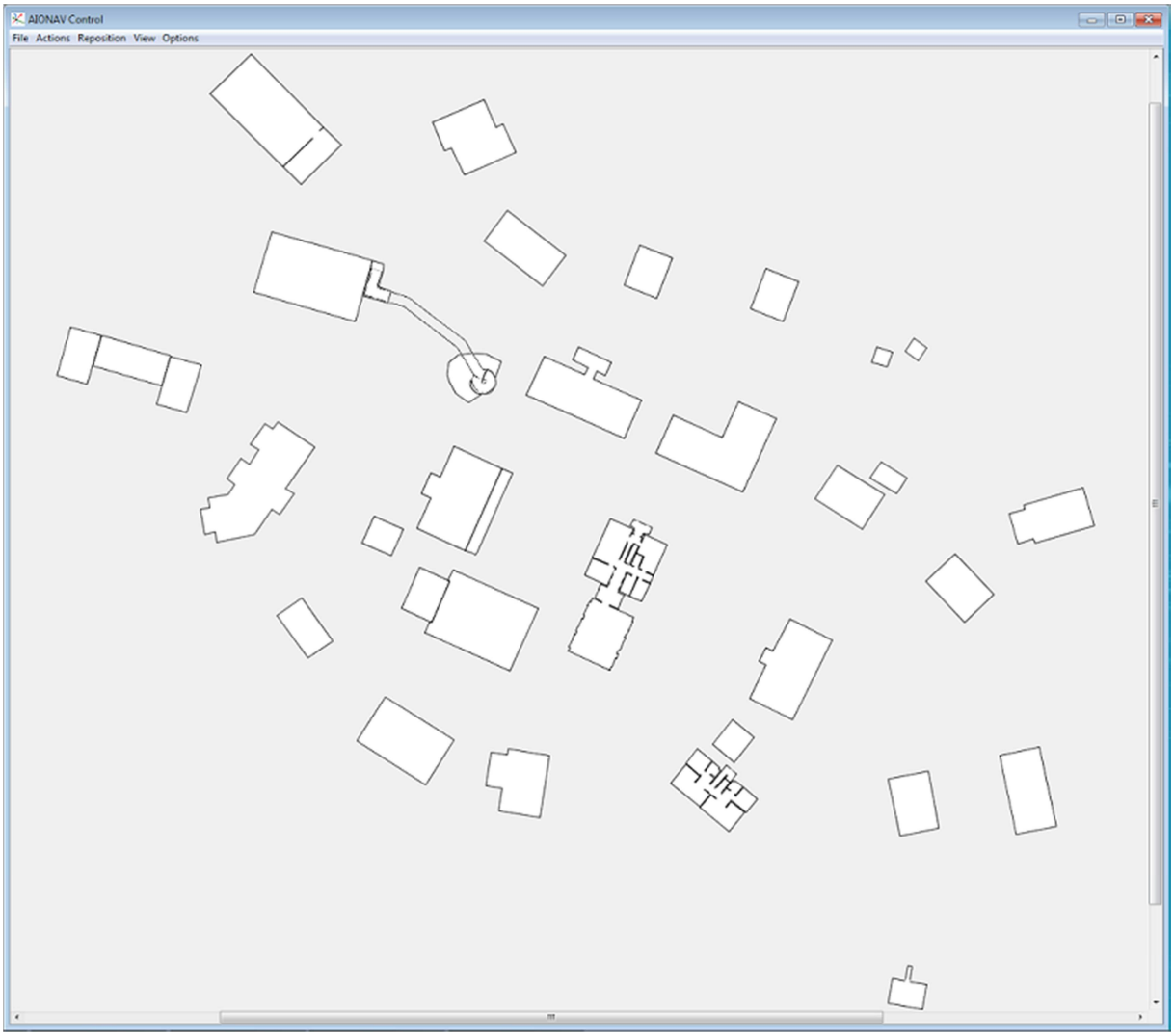


Figure 24: Walenstadt military complex in the final AIONAV version

3.1.3 Extracting building data from existing building models

Two approaches of how to utilize existing building models can be suggested. Since a BIM is not viable to be used in its original form, for the reasons previously discussed, the data can either be converted into the more efficient form first, or the building model itself is altered and modified into a more efficient form suitable for mobile devices. While the first method with the IPS XML as the more efficient form will be discussed further in the current chapter, the second method, which is made possible

with the introduction of IFC model view definitions, will be discussed further in the conclusion.

3.1.3.1 Converting the data from existing building models

Using an existing BIM for the creation of an indoor positioning BIM seems to be straightforward and to be the obvious and best route for the goal of using existing building data for indoor positioning. However, it does have its drawbacks and limitations. First of all it has to be noted that quality building information models are in short supply. Not only do building information models not exist for most older buildings, it is even hard to find vectorised or digitalized pixel floor plans. There is also a lack of building models in newer buildings. Currently, building information models are not yet the standard for building data storage. However, the trend is developing in the direction of standardizing BIM formats (especially IFC) as an international standard for building documentation.

Whenever a BIM is available the data can be utilized in the process of creating an indoor positioning model. However, a typical BIM includes information that is not needed for indoor positioning and, most of all, lacks topological information from which a graph model could be generated. Hence, the information that is stored in the BIM has to go through a process of extraction, conversion and adaptation into the right form, which in this case is the IPS XML schema. The main challenges are the following:

1. Filtering the right data from the BIM
2. Analysing the data and identifying potential flaws and inconsistencies
3. Correcting potential flaws and inconsistencies and building up the IPS BIM
4. Exporting an XML file according to the IPS XML schema

In these four steps data is made available in the form that can be further utilized by indoor positioning software.

The different BIM formats that are currently available provide a wide variety of potential sources. However, because of the differences in the nature of these formats, each requires an individual approach and treatment to effectively complete the objective of extracting and converting the data into a suitable form.

3.1.3.2 Modifying the BIM format

IFC, the international standard BIM format, offers a way to split information in the form of model view definitions, which make only the required data from a BIM available. Thus, different professional fields can suppress certain information from the BIM that is not in their line of interest and only use the information relevant to them. The approach of modifying the BIM format requires the implementation of the buildingSMART tool ifcDoc. This tool is specifically designed to allow users to design model view definitions according to their needs.

Compared to the data conversion method workflow, the workflow for acquiring the required data by modifying the IFC building model format is much simpler. First a properly defined MVD is needed that serves as the basis for IFC file generation according to that particular MVD. Preferably, this MVD should also be internationally certified by buildingSMART in order to make it official. The next step is to develop and then certify building information modelling software (or develop a general application) for the export of the aforementioned MVD. This way the process of generating quality information for Indoor Positioning would be streamlined.

3.2 Data extraction and conversion

Among all of the possible analysed and described choices of building model formats, two have been looked at in more detail. The selection was not random and followed the research path of findings, while also considering other analysed possibilities. The Autodesk Revit application and its API were chosen for the introduction to and the first glimpse of the data structure in building models. The API offers a relatively straightforward and simple way of working with the complex data structure of the Revit building model. Working with the Revit API, however, does not offer a general solution, and only Revit models could be utilized for creating models according to the IPS XML schema. With this approach every single building model data format would need an application for data conversion. This is definitely not acceptable in the long run, and following this intermediate step a more general approach was pursued. The most important requirement for the new approach was that it had to offer a general solution and, hence, be platform independent. In other words, most if not all modern BIM platforms should be covered. Revit building models as well as Archicad, Allplan

etc. building models should be able to provide the necessary information. Therefore, a common denominator had to be found and was identified in the IFC international standard format for building information models. Practically all modern platforms for building information modelling offer at least some kind of support for IFC, making it the go-to choice for any kind of general coverage. A list of platforms particularly for the IFC2x3 Coordination View MVD, according to the V2.0 certification process, can be found on the internet (buildingSMART International Ltd.).

This approach was chosen as a step by step method, as it involves first looking at the Autodesk Revit model through the provided API by the Autodesk Developer Network (ADN) and following up with IFC, with the intention of a future possibility of making an official Indoor Positioning MVD. The initial objective was to analyse and get acquainted with the special data structures of building information models. The extension created with the API for Autodesk Revit was never intended to provide a complete solution. Knowledge acquired during the process enabled the thorough research of relevant IFC classes, and their comparison to their counterparts from the IPS XML format was made possible, thus opening a way of enabling IFC as the main format for building data mining with the purpose of creating viable information.

In the following chapters both processes will be introduced, similarities will be pointed out and differences made apparent. Starting with the Revit API and following up with IFC, an insight into the thought process behind the work will be presented and the workflow itself revealed. Both applications have been coded in C# with the future possibility of the code being ported to other programming languages. The user interface for the Revit extension is, needless to say, Autodesk Revit itself, while for the IFC converter a simple interface for Windows was designed by a student of civil engineering of the Technical University Graz (Haas, 2013) in the scope of his bachelor project. Additionally, a short overview into the general approach will be offered and some greater challenges presented. With code snippets and comments, an insight into the data gathering and conversion processes is supposed to enlighten and make the method easily approachable. Furthermore, certain test scenarios with various building models are going to present a possible application of this type of data conversion in practical situations.

3.2.1 RVT format and the Revit API approach

The Autodesk Revit API was used as an introductory step into building information model data structures. Autodesk provides the members of their ADN website access to various tools and support for programming extensions on their Revit application in the form of an ever evolving API. The API offers more or less unhindered access to the Revit model in the form of C# or Visual Basic class libraries that mirror the structure of the Revit model and allow access to that same model through methods and variables. With the API, add-ins for Revit can be programmed that extend the capabilities of the basic application. These add-ins can be accessed and used inside Revit in the form of additions to the user interface as new buttons in the *Extensions* tab in the ribbon.

The basic principle of creating Revit extension applications can be broken down into the four steps depicted in Figure 25 bellow.

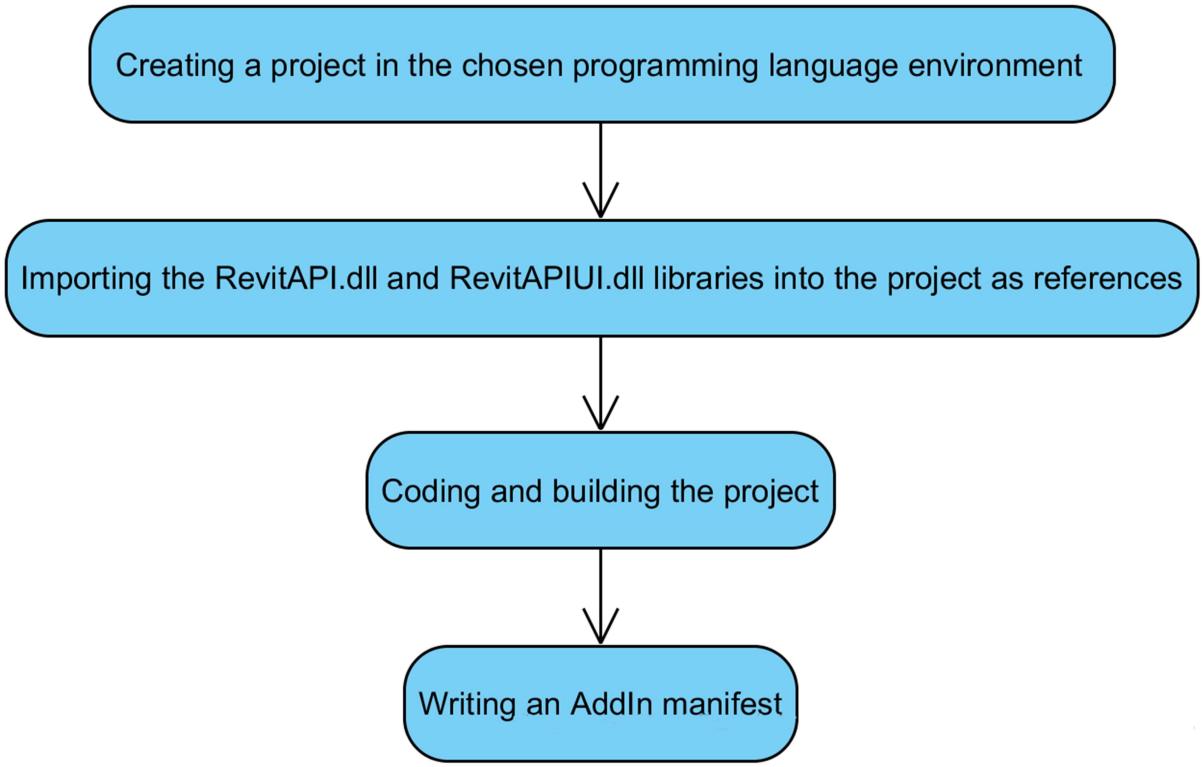


Figure 25:. The four steps for creating a Revit extension application (Autodesk, Inc, 2014)

The first two steps are relatively straightforward. Creating a project and adding references are typical procedures when starting any new project in a programming environment. Steps three and four, however, are specific to the project in question with step four being unique when creating Revit extension applications.

Creating a new project in the chosen programming language and naming the project more or less only has an impact on the path to the library files. Nevertheless, in our particular scenario it has consequences further down the path, because the main Revit application requires access to these files for the add-in to run properly. With the import of the two Revit libraries, access to the API classes is granted in the created add-in project. These classes enable work on and with the Revit file (the building model).

Undoubtedly, writing code is individual for the task ahead and the problem that is to be solved by the Revit extension. It has to be mentioned that unlike a typical application where the start is from a random class that holds the *main* method, the Revit add-ins require a specific class that extends the *IExternalCommand* class and starts in the *Execute* method with a return type *Result*.

The purpose of the fourth step is to prepare all the necessary information for loading and running the extension with the next launch of Revit. The add-in manifest is a text file with the extension ".addin" at a specific location, which Revit always checks before launch. If any add-in manifests are found during the start-up procedure and the form is correct the add-in is loaded. The correct form of the manifest holds some specific information crucial for a correct add-in launch. The information about every add-in that can be found in the add-in manifest is a unique ID, a path to the location of the library (the dll file), the name of the extension, and the name of the command that is used to start it up. This information is required for Revit to find the right library, launch the add-in and add a button to the Revit user interface.

The integrated development environment (IDE) that was used is Microsoft Visual Studio and the programming language C# was chosen predominantly because of its similarity to Java, in which the IPS building model is coded. Continued use of Java for extracting data from Revit was not possible, because at the time of the project there was no Revit API library available for Java. To challenge this difficulty a C# li-

brary was created, according to the IPS XML schema, and was then used as the basis for creating IPS XML files, according to the IPS XML schema from Revit building models. The Liquid XML Studio software was used so that the Java code of the IPS building model would not have to be ported to C# manually. As well as an XML Editor, this software also features automatic code generation directly from XML schemas. Additionally, all classes created by the software feature special methods that enable the direct export of the code to XML.

With the primary goals of the Revit add-in being the filtering of the relevant Revit families and the extraction of the required data for constructing the IPS XML file, the process was broken down into the four parts presented on the diagram in Figure 26.

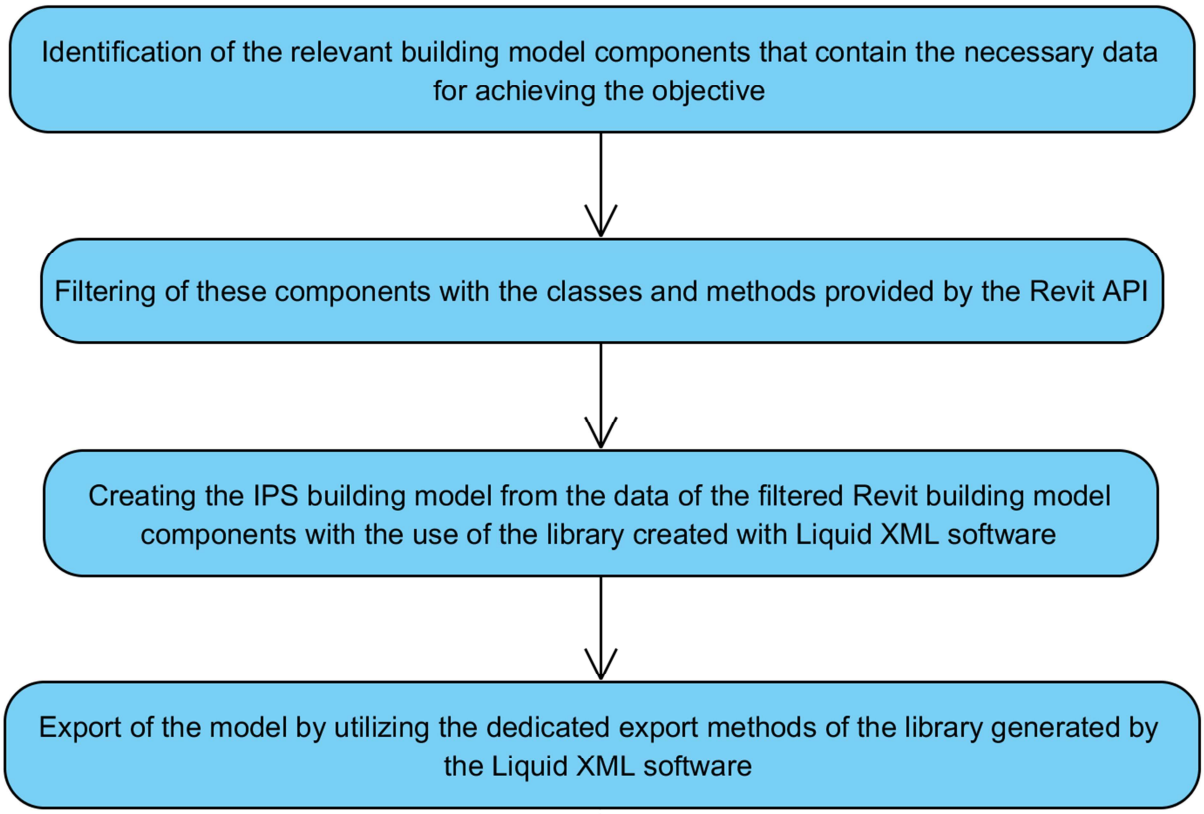


Figure 26: The process of generating the IPS XML from a Revit model

For better comprehension, the following chapters will provide a more detailed description of the four listed steps. The first step was completed manually by consulting the help documentation of the Revit API and using general knowledge about Revit. As a result, a list of plausible Revit families was created, which was then analysed

and tested with the help of short practical examples in the Revit add-in by debugging and analysing variable values. After the most suitable Revit families were chosen, steps two through four had to be implemented. The algorithm is straightforward and simple. It was coded in C#, and a functional Revit add-in was created and tested in three Revit versions.

3.2.1.1 Identification of relevant building model components (Revit families)

Initially, the various components that construct the Revit building model had to be examined to identify the ones with valid content for Indoor Positioning. In general, the data in a Revit document consists of a collection of elements. *Element* (see *Table 14*) corresponds to a single component of a building, e.g., wall, door, window etc. Albeit, various elements can be more abstract, such as views, or spaces.

Our method consisted of looking at Revit families first, identifying potential candidates among them, finding the corresponding classes in the Revit API and using those classes in the construction of the IPS model. Objects from the Room family promised to provide the needed geometry, while *Door* and *Window* family objects provide *Room* connectivity (where individual rooms can be crossed over to other rooms). Additionally, the *Room* family objects also enabled the extraction of the corresponding level (floor), which was necessary for creating the vertical structure of the building. With the information about *Floors* already available in the *Rooms*, no additional query regarding *Floors* would have to be executed. For these reasons the *Room* family was chosen as the basis from which the IPS building model would be built. The *Room* family almost completely mirrors the *Area* class from the IPS XML format, although it comes with certain significant differences.

By comparing the Revit and IPS classes we have established the following; the *Area* class is more general and in that respect somewhat more complex. Therefore, some specific families had to be accounted for to fulfil the gap between the Area class of IPS XML and the Revit *Room* family. One such Revit family of building objects that had to be considered is *Stairs*. Whereas the Revit model defines stairs as a separate entity (Revit family), the stairs in the IPS building model are a part of the *Area* class: i.e., stairs in IPS XML are merely a special type of *Area* (defined as an *AreaType* enumeration) and not an individual class as in Revit. Therefore, when building the

structure of *Area* objects in IPS XML from a Revit model, both *Room* families and *Stair* families have to be considered to capture all necessary data and construct the complete geometry of the building.

Table 14: Element class

Name	**Description**
AssemblyInstanceId	Id of the elements assembly instance.
BoundingBox	Returns a box that circumscribes the element's geometry.
Category	Category of the object the element resides in.
DesignOption	Design option of the element.
Document	The document in which the element resides.
Geometry	The geometric representation of the element.
Group	Group to which the element belongs.
Id	Unique identification of the element within a Revit project.
Level	Level of the element.
Location	Location of the element.
Materials	All materials present in the element.
Name	Human readable name of the element.
Number	The number.
ObjectType	Obsolete. Instance type of this object.
OwnerViewId	Id of the view that the element belongs to.
Parameter	Retrieves a parameter of the element given an identifier.
Parameters	A set of all parameters contained within the element.
ParametersMap	A map of all parameters contained within the element.
Perimeter	The perimeter.
PhaseCreated	The phase in which the element was created.
PhaseDemolished	The phase in which the element was demolished.
Pinned	Identifier of whether the element is pinned.
SimilarObjectValues	Obsolete. Types similar to the current instance.
UniqueId	A stable unique identifier for an element in the document.
ViewSpecific	Identifies whether the element is owned by a view.
WorksetId	Id of the Workset that owns the element.

As well as a class from IPS XML having to extract data from more than one Revit familiy, another issue has to be taken into consideration. Revit families are only a baseline, which does not necessarily directly coincide with Revit families. For example, in the case of the Architectural *Room* family; all *Area* defining families can be tracked back to the super class *SpatialElement*. This class also features other families such as *Area* (a Revit family with no volume compared to a *Room*) and *Spaces* for mechanical analyses. This means that classed from IPS XML do not mirror families from Revit, i.e., data has to be gathered from different Revit families to form a particular class in the IPS XML (Table 15). The Revit API classes are typical representations of object oriented programming classes, where we have to distinguish between two types of members. The first type is methods, which literally translate into actions we can perform with an object of a particular class. The second type is properties, which are values directly accessible from an object of the class in question or significant objects of other dependant sub classes.

Table 15: Employed Revit classes with their IPS XML counterparts

Revit classes	IPS XML classes
SpatialElement	Area
Stairs	Area
Floor	Floor
Element (Enumeration OST_Door)	LinePair/Connection
BoundarySegment	Line

The Revit API classes that we have used are listed in the following Tables with their properties and descriptions of those properties: Starting with the *Element* class (*Table 14*), which is the base class for most data in Revit. Additionally, the differentiation between families through the system of a filter class *ElementCategoryFilter*, which relies on the enumerations of families (*BuiltInCategory* enumeration), e.g., *OST_Doors* to filter the model it, also forms the backbone to the more abstract elements as their parent class, e.g., the *SpatialElement* inherits from the *Element* class. Particularly the *SpatialElement* class differs from the parent *Element* class in respect to properties only in one additional property, which is the Area of the *SpatialElement*. Certain other properties from the parent *Element* class (such as *Level*) are overridden in the *SpatialElement* class.

Furthermore, if we wanted to work with the *Room* class instead of the more general *SpatialElement,* some additional properties of the *Element* could be accessed, e.g., *Volume*, the *Offset* of the *Room* (*BaseOffset* and *LimitOffset*). These properties were not needed and would restrict the flexibility of the *SpatialElement* class that features any kind of bounded areas in the model.

Similarly to the *SpatialElement* class, the *Floor* class is derived from the base *Element* class and inherits most if its methods and properties from its parent class. It does, however, receive six additional properties as listed in Table 16.

Table 16: Additional properties of the Floor class compared to the Element class

Name	**Description**
FloorType	Retrieve or set the type of the Floor.
Parameter	Overloaded. Retrieves a parameter based on an identifier.
SlabShapeEditor	For Slab shape editing.
SpanDirectionAngle	Span direction angle for the floor.
SpanDirectionSymbols	Array of span direction symbols of this floor.
StructuralUsage	Either bearing or non-bearing.

The last class that requires attention is the *BoundarySegment*. This is a different type of class, as it does not represent any building components. It serves to define the exterior boundary of an *Area*. Therefore, it was used to acquire geometry data from the Revit model. The class with its properties can be observed in the following table (Table 17).

Table 17: BoundarySegment

Name	**Description**
Curve	The curve formed by the element of the edge of the area.
Document	The document of the element that created this segment.
Element	The element responsible for producing this segment.
IsReadOnly	Identifies if the object is read-only or modifiable.

3.2.1.2 Filtering the model with the Revit API

The Revit API allows us to select and work with the entire model of an active file with the use of the code fragment displayed in Code 2 a). Once this selection has been completed the created *Document* object, and along with it the building parts that the object contains, can be accessed with the use of the *FilteredElementCollector* class. The objects of this collector class enable the filtering of the initial building model selection, according to different types of other Revit classes. An example code fragment for the *SpatialElement* object is displayed and can be examined in Code 2 b). The active document (doc) is used as a parameter in the creation of a *FilteredElementCollector* of the *SpatialElement* type. The result of the filtering process is a list of the desired type of object stored in a collector object, and can be accessed correspondingly by either iterating through the list with a loop or by referencing individual elements from the list.

```
a) Document doc = commandData.Application.ActiveUIDocument.Document;

b) FilteredElementCollector collector = new
   FilteredElementCollector(doc).OfClass(typeof (SpatialElement));
```

Code 2: a) active document call; b) creating a filter of the type SpatialElement

To summarize, the whole process of filtering individual types of Revit families from the Revit building model is simplified by the use of element collectors. In a relatively simple way the entire model can be scanned and lists of the desired family types created. Needless to say, this is perfect for our cause and confirmed that using the Revit API as a means of gathering information about the BIM was a good approach. As we will see further on when introducing the approach with an IFC file, the filtering of specific building blocks from building models is not a trivial task and requires some effort to be done efficiently.

3.2.1.3 IPS XML schema library in C#

Since the IPS XML building model was coded in Java and the Revit API in C#, one of the libraries had to be ported to the other programming language. Translating the Revit API was out of the question because the goal was to test the code in Revit as a Revit extension and these cannot be written in Java since Revit is based on .NET

and, therefore, is required to make an extension work properly in Revit. This fact narrowed down the available alternatives, and only these three options remained:

1. Creating an IPS XML library in C# from scratch,
2. Porting the existing IPS XML library from Java, while also creating an exporter for XML,
3. Making use of the IPS XML Schema for an automatic generation of a C# library.

The first option was never seriously considered. It would require too much time and effort for it to work, and in the end nothing new would have been created other than some redundant code in a different programming language. Therefore, this option will not be discussed further.

The second option provided the initial path. The existing code from Java was supposed to be used to generate a C# library. In a process called "porting", code from a certain programming language (in our case Java) is taken and written in a different programming language (in our case C#). However, the syntax and the framework from different programming languages differ, which means that the code has to be rearranged in addition to being translated. After some research it was established that no effective and robust enough automatic solution for porting code from Java to

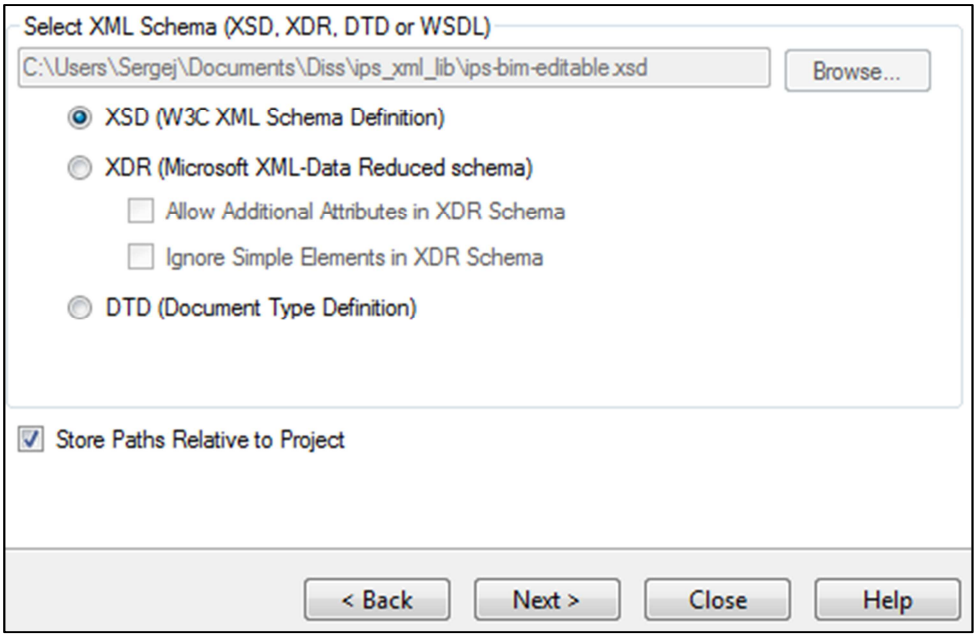


Figure 27: Creating a library from an XML schema in Liquid Studio

C# currently exists. Thus, the whole process would have to be done manually, as in the first option, which was to create a library in C# from scratch, and would require too much time and effort. Consequently, this alternative will also not be discussed further.

Finally, the remaining possibility, which has also proven to be the most time efficient compared to the other two, was to utilize the existing IPS XML Schema from previous work on the CADMS project and, on that basis, to create a library in C#. For this purpose one software package was analysed and inspected that offered automatic code generation (the needed C# library of the IPS building model) directly from an XML schema. The chosen software package, Liquid XML Studio 2012 (Developer Edition version 10.1.2.4113), comes from Liquid Technologies Limited and provides the necessary capabilities required for the task. Additionally, the package contains other useful applications, which proved effective while working with XML files and schemas. These applications include an XML Editor, the ability to generate XML documentation from an XML schema and vice versa, etc. The process itself is relatively simple and straightforward. From the main user interface window the option "Generate Source Code from an XML Schema" is selected and a Wizard that guides us through the process is started. In the first stage we can select from four different schema formats (XSD, XDR, DTD, WSDL; Figure 27), and since the IPS schema is in XSD format, this particular option was chosen. In the next stage various output options are available for selection. These include the most widely used and popular programming languages such as C++, C# .NET, Java, Visual Basic .NET and Visual Basic 6.0, from which C# was selected. After offering further processing options, the application will not only generate the code but also the documentation for the library. The generated code is a complete C# project with all the classes extracted from the XML schema and can then be built inside of Visual Studio and a .dll library generated. The library can also be used as a reference in any other C# project.

An important note about the generated classes and the code in them is that whatever information can be extracted from the schema finds its place in the code. In our case the parameters and enumerations, e.g., the types of *Areas* (Flat, Ramp etc. already discussed in chapter 3.2.3) are particularly important. Additionally, every generated class receives some methods that are not provided by the XML schema. All these methods have the purpose of exporting the model from code to the XML format in

common. Essentially, what this means is that every object created with these classes can be individually exported to an XML format file by using these methods. As a consequence, exporting the object highest in the hierarchy (the *Region* in the IPS BIM) indicates that the complete model is exported to an XML file.

3.2.1.4 Generating the IPS building model

In the previous points we described the necessary preparation work for building an IPS model and exporting it to an IPS XML file, according to the IPS XML schema. Now we can focus on the actual process of extracting the right required data and using it to build the model from the bottom up. This is literally the case, since the model will be built starting with *Points*, followed by *Lines*, and continued through *Areas*, *Floors* and *Building*, all the way up to the *Region*.

The purpose of a typical building information model format and the IPS XML format is fundamentally different; therefore, the data and the manner in which it is stored in either of the mentioned formats must be appropriate for that purpose. Even though they represent the same real life object, we have to be aware of the differences in the formats enabling us to determine and use only the viable part of a building information model in the generation of the IPS XML. A safe way to begin is with geometry, since both formats have that in common.

```
foreach (SpatialElement spatialElement in collector)
{
       IList<IList < Autodesk.Revit.DB.BoundarySegment >> boundariesOfArea =
       spatialElement.GetBoundarySegments(new SpatialElementBoundaryOptions());

       foreach (IList<Autodesk.Revit.DB.BoundarySegment> boundarySegments in
       boundariesOfArea)
       {
    com.aionav.bim.ips.Area area = new com.aionav.bim.ips.Area();

              foreach (Autodesk.Revit.DB.BoundarySegment boundarySegment in
              boundarySegments)
    {
      line.Point.X = boundarySegment.Curve.GetEndPoint(0).X;
      ...
          }
       }
}
```

Code 3: Accessing the geometry of a Revit SpatialElement and using it for the creation of an IPS XML Line object

From the preliminary analysis (chapter 3.2.1.1), the Revit family that suits the needs of the IPS BIM geometry and definition of space the most is the Architectural *Room* for the Architecture or *Space* for the MEP (Mechanical, Engineering, Plumbing) version of Revit. The Revit API uses the common class *SpatialElement* for these two families, thus making our algorithm easier and usable for either type of Revit model. When one particular type of *SpatialElement* is needed a cast to that class is possible. As described in chapter 3.2.1.2**Error! Reference source not found.**, by filtering with the *FilteredElementCollector* a list of *SpatialElements* is generated. By iterating in a loop through all *SpatialElement* objects from the collector, every individual *SpatialElement's* geometry can be accessed (Code 3) through boundary segments (*BoundarySegment*) of each individual object from the collector. A second loop can then check all *BoundarySegment* objects in a *SpatialElement* and create the points in the IPS XML from the start and end point of the *BoundarySegment Curve*.

By using the IPS XML library generated with Liquid XML Studio 2012, as described in chapter 3.2.1.3, all IPS XML classes can be individually exported to XML, as a method (*toXML()*) is generated for them. This way, if the *toXML()* method is applied to the class highest in the hierarchy, the entire building model is exported to XML. The strict hierarchy of the IPS XML is kept throughout the defining schema. In this specific case, the class highest in the hierarchy was not *Region* or *Site*, as the IPS XML allows: it was *Building*, since we were not working with the site tools available in Revit. For testing purposes, *Building* (as the highest class in the hierarchy) was considered valid, since the main objective was to model the interior and analyse the geometric data structure in a building model.

3.2.2 Data extraction from IFC

Because IFC is the international standard for building models, attaining models of newer buildings is relatively easy in the case of either the planning, construction or both phases involving building information modelling principles. With more and more countries working or implicating new building information modelling standards, models of recently designed and constructed buildings are abundant, and most of the modern building modelling software features certified export for the IFC file formats

(either EXPRESS or ifcXML). The IFC file schema is, therefore, the perfect candidate to be taken into consideration.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]'),'2;1');
FILE_NAME('Project1.ifc','2013-06-21T15:26:09',(''),(''),'Autodesk
Revit Architecture 2013','20120221_2030(x64)','');
FILE_SCHEMA(('IFC2X3'));
ENDSEC;
```

Code 4: IFC file Header

Our focus was mainly on the EXPRESS file encoding because it is currently the preferred file structure (Liebich, 2009). A typical STEP physical file is a structured ASCII text file, consisting of a header (Code 4) and a data section (Code 5). The header is a general section located at the beginning of the text file and typically includes the following information:

- The IFC version,
- The application used for exporting the file,
- The date and time of the export,
- Optionally, the name, the company and the authorizing person.

```
DATA;
#1= IFCORGANIZATION($,'Autodesk Revit Architecture 2013',$,$,$);
#2= IFCAPPLICATION(#1,'2013','Autodesk Revit Architecture
2013','Revit');
#3= IFCCARTESIANPOINT((0.,0.,0.));
#5= IFCCARTESIANPOINT((0.,0.));
#7= IFCDIRECTION((1.,0.,0.));
#9= IFCDIRECTION((-1.,0.,0.));
#11= IFCDIRECTION((0.,1.,0.));
```

Code 5: IFC file Data section

The basic principle of how building information is stored in an IFC file is similar to the way in which it is accessible through the Revit API. For this reason, the previous research about which Revit families are suitable and provide the required data is also valid for the IFC file. However, it has to be taken into account that objects in IFC will not have the same annotation as families in Revit. The task at hand is to identify the relevant objects first. Again, we have to look for space bounding objects, which in

IFC were first expected to be a part of the *IfcSpace* objects. Indirectly, this was correct. The boundary of spaces is not part of *IfcSpace,* but can be found in an object that references *IfcSpace.* Any *IfcSpace* is referenced by as many objects of the *IfcRelSpaceBoundary* entity type as there are segments in the boundary of the aforementioned *IfcSpace*. The purpose of these segments is to define the boundary of a particular *IfcSpace*; i.e. where the open space of a room (open building space) stops and the bounding object (e.g. wall or connection object like a door or window) starts. The connection surface itself is defined by an object of the *IfcConnectionSurfaceGeometry* entity type that references a profile definition (*IfcArbitraryOpenProfileDef*) for the boundary segment. The geometric data for the segment is then referenced further, according to the type of segment, and can be defined by various geometric shapes available in the IFC schema. These range from polylines (*IfcPolyline*), arcs (*IfcCircle*), splines (*IfcBSplineCurve*) etc., and provide most geometric data through vertices (*IfcCartesianPoint*) and parameters specific to the shape of the boundary.

As the IFC standard provides abstract and concrete real life building objects for constructing a building model, this leaves us with the possibility for imperfections. This means that different users could generate different building models as well as different approaches to modelling that would result in the possibility of particular IFC objects not being used. In regard to this, we can differentiate between complete and incomplete IFC building models. With respect to the main subject discussed, the lack of *IfcSpace* objects is the most plausible and relevant case that can complicate the process. *IfcSpace* is often not needed, because the information it contains could be redundant with respect to other building objects, on the other hand, *IfcSpace* is critical for our method. Therefore, in addition to the complete IFC model with the desired *IfcSpace* objects, these special cases, where *IfcSpace* objects are not available in the provided models, will be dealt with in the following chapters. Additionally, an IFC file can exist without specialized space boundary segment objects (*IfcRelSpaceBoundary*) that define the boundary of rooms, and can be used to great effect when extracting and creating topological spaces representing rooms. These cases of special use require individual and specialized approaches to deliver the required data and satisfactory results. The different representation of geometry through *IfcSpace* and *IfcRelSpaceBoundary* entities is depicted in Figure 28. Since *IfcSpace* geometry

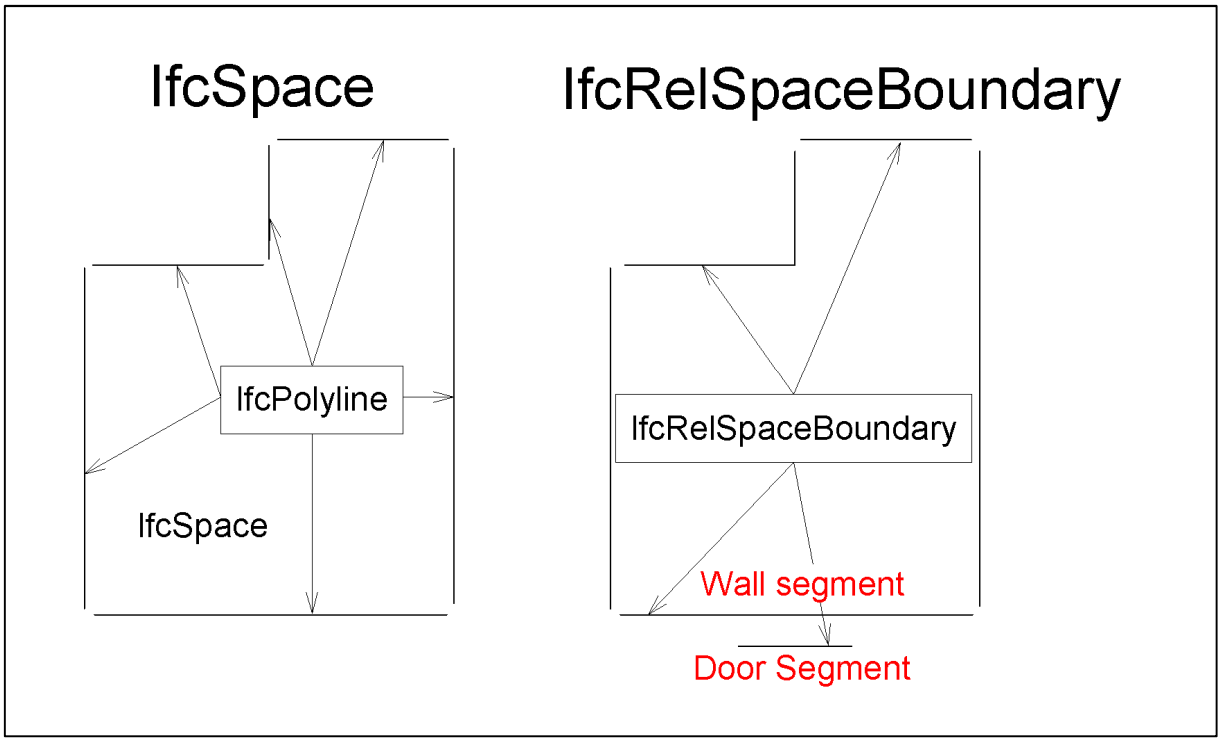


Figure 28: Geometry of an IfcSpace entity with geometry entities (IfcSpace geometry representation) on the left, and IfcRelSpaceBoundary entities on the right

representation is achieved through geometrical entities (like *IfcPolyline*), the *IfcRelSpaceBoundary* also represents door segments as well as differentiating between wall and door segments.

The next chapters will look into the required critical IFC objects in detail. The approach from the Revit API method proved to be successful; therefore it was adapted and applied in the same manner to the IFC format. However, not all of the geometric and shape IFC objects will be addressed individually. The shape objects are important for creating the right geometry and can be addressed according to the shape of the building. These objects can be considered trivial: for example, a circle is defined by a centre point and a radius and does not offer any added value in the understanding of the model. Additionally, the most important scenarios of available IFC files will be analysed and possible solutions suggested. In particular, we will introduce the algorithm for extracting the needed data from an IFC file with and without *IfcRelSpaceBoundary* to create valuable information.

3.2.2.1 Required IFC entities

In chapter 2.1.1.2 we identified the *Products* object entity types derived from *IfcObject* as the potential source of geometric data (location and shape of objects), and the *Connectivity* relationship entity types derived from *IfcRelationship* as the potential source of connectivity data. From these two fundamental entity types we further analysed the best candidates of entities for our objective. The resulting selection of entities with the relevant attributes will be introduced in this chapter, along with the insights gained from the initial study. Not all of the listed entities proved to be beneficial and, therefore, were not used further.

As defined in the previous chapter, two basic scenarios that split into a number of additional more specific options should be taken into consideration. The key entity that differentiates these two scenarios is *IfcSpace*. An IFC model can either include it or not.

"A space represents an area or volume bounded actually or theoretically. Spaces are areas or volumes that provide for certain functions within a building.

A space is associated to a building storey (or in case of exterior spaces to a site). A space may span over several connected spaces. Therefore a space group provides for a collection of spaces included in a storey. A space can also be decomposed in parts, where each part defines a partial space. This is defined by the Composition-Type attribute of the supertype IfcSpatialStructureElement which is interpreted as follow:

COMPLEX = space group

ELEMENT = space

PARTIAL = partial space" (BuildingSMART, 2013)

The *IfcSpace* entity is used to define this spatial structure of a building, and individual *IfcSpace* entities residing on the same storey are linked together and to the storey with the relationship entity *IfcRelAggregates* (see Code 6), linking the *IfcSpace* entities and the *IfcBuildingStorey* entity on which they are located. The *IfcRelAggregates* forms a composition/decomposition (whole/part) relationship, where the whole

(*IfcBuildingStorey*) depends on the composition of the parts (*IfcSpace*) and the parts depend on the existence of the whole.

```
#113=IFCBUILDINGSTOREY('1obQpJ1wf95we7UfPVdZhL',#41,'Level 1'
     ,$,$,#111,$,'Level 1',.ELEMENT.,0.);
#139=IFCSPACE('2rvVtukHjE7v3QNTb$i7vv',#41,'2',$,$,#122,#135,
     'Room',.ELEMENT.,.INTERNAL.,$);
#277=IFCSPACE('2rvVtukHjE7v3QNTb$i7vw',#41,'3',$,$,#264,#275,
     'Room',.ELEMENT.,.INTERNAL.,$);
#396=IFCSPACE('2rvVtukHjE7v3QNTb$i7vd',#41,'4',$,$,#383,#394,
     'Room',.ELEMENT.,.INTERNAL.,$);
#1055=IFCRELAGGREGATES('3Zu5Bv0LOHrPC100A6FoQQ',#41,$,$,#113,
     (#139,#277,#396));
```

Code 6: IfcSpace entities on one IfcBuildingStorey linked by IfcRelAggregates (references are bold)

IfcSpace consists of the explicit attributes that can be observed in Table 18. For the purpose of Indoor Positioning, two attributes are especially important: namely, *ObjectPlacement* and *Representation*. The first one defines the position of *IfcSpace* and the second one the geometry.

Table 18: IfcSpace

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
LongName	IfcLabel (STRING)
CompositionType	IfcElementCompositionEnum (ENUM)
PredefinedType	IfcSpaceTypeEnum (ENUM)
ElevationWithFlooring	IfcLengthMeasure (REAL)

Another object that should be considered, especially if an IFC model lacks *IfcSpace,* is the *IfcWall* or *IfcWallStandardCase*. These are the three separate objects for common cases of walls:

1. *IfcWallStandardCase* – used for the most basic walls with a constant thickness along the wall path and where the thickness parameter can be fully described by a material layer set. This type of wall is geometrically represented by an axis and a swept solid,
2. *IfcWallElementedCase* – walls that are aggregated from subordinate elements, following specific decomposition rules,
3. *IfcWall* – used for all other wall types not covered by the previous two cases, especially for walls with changing thickness along the wall path, non-rectangular cross sections and non-vertical walls. Geometric representation is Brep or SurfaceModel.

(buildingSmart, 2013)

For the sake of simplicity I will focus on *IfcWallStandardCase,* as these kinds of walls cover the majority of cases in buildings and provide the necessary geometric information. *IfcWallElementedCase* focuses on walls that are decomposed into parts, which are of no particular interest for the purpose of defining the borders of a building space, and *IfcWall* handles special geometric cases that can be dealt with individually. Additionally, the attribute layout of *IfcWall* is identical to *IfcWallStandardCase,* giving no added value if both were analysed individually, whereas only the reference to the attribute Representation differs in the fact that more complicated shape representation objects can be referenced by the *IfcWall* object.

The *IfcWallStandardCase* and *IfcWall* explicit attributes are listed and can be observed in Table 19.

The information that is required from a wall is its location and geometry. Hence, the *ObjectPlacement* and *Representation* attributes are the two most important to consider.

Table 19: IfcWall and IfcWallStandardCase

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
TAG	IfcIdentifier (STRING)
PredefinedType	IfcWallTypeEnum (ENUM)

Another IFC object that has been mentioned, and provides not only some additional geometry information but should also deliver far more important connectivity (topology) information between spaces, is the *IfcDoor*. A list of explicit attributes for an *IfcDoor* can be found in Table 20.

Table 20: IfcDoor

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
TAG	IfcIdentifier (STRING)
OverallHeight	IfcPositiveLengthMeasure (REAL)
OverallWidth	IfcPositiveLengthMeasure(REAL)
PredefinedType	IfcDoorTypeEnum (ENUM)
OperationType	IfcDoorTypeOperationEnum (ENUM)
UserDefinedOperationType	IfcLabel (STRING)

Additional geometry data can again be found in the *ObjectPlacement* and *Representation* attributes, however, it is supplemented by the *OverallHeight* and *OverallWidth* attributes, which give a quick insight of the wall opening geometry of a particular door.

Since stairs are not a part of any of the aforementioned objects but are essential for defining the area of possible movement, due to being accessible by building occupants, the dedicated entity *IfcStair* (see Table 21) and particularly *IfcStairFlight* (see Table 22) have to be taken into consideration as well. *IfcStairFlight*, especially, provides specific information about the boundaries of the treads (in the *Representation* attribute) and is required for extracting the geometric representation of the stairs footprint.

Table 21: IfcStair

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
TAG	IfcIdentifier (STRING)
PredefinedType	IfcStairTypeEnum (ENUM)

The *IfcStair* entity either provides data about the whole stair assembly (stair flight, landing, railings), where all the related stair entities are associated by the *IfcRelAggregates* entity, or a single stair entity without decomposition including the whole stair representation. The *IfcStairFlight* focuses on the stair flight exclusively. As an assembly of building components, such as stringers and steps, it represents a single run of steps without any landings in between. The *Representation* types currently supported are *Axis*, *FootPrint*, *Body* and *Box* (*BoundingBox*).

Table 22: IfcStairFlight

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
TAG	IfcIdentifier (STRING)
NumberOfRiser	INTEGER
NumberOfTreads	INTEGER
RiserHeight	IfcPositiveLengthMeasure (REAL)
TredLength	IfcPositiveLengthMeasure (REAL)
PredefinedType	IfcStairFlightTypeEnum (ENUM)

When *IfcSpace* objects are used in the model definition and, therefore, can be found in an IFC building model we can further differentiate between two possible scenarios regarding the *IfcRelSpaceBoundary* entity: firstly, when the *IfcRelSpaceBoundary* entity (Table 23) is included in the model, and secondly, when left out of the model. In fact, the *IfcRelSpaceBoundary* entity is crucial in our developed algorithm to establish connectivity between individual rooms that are connected by doors.

In addition to providing the geometric representation data of the boundary segment through the *ConnectionGeometry* attribute, the *IfcRelSpaceBoundary* provides information about the element type (the attribute *RelatedBuildingElements*) that defines the boundary segment, and the *IfcSpace* element (attribute *RelatingSpace*) that is bounded by it. The *PhysicalOrVirtualBoundary* and *InternalOrExternalBoundary* attributes describe the nature of this particular boundary segment by determining whether the boundary is physical or virtual (*PhysicalOrVirtualBoundary*) and whether the boundary segment faces the interior or exterior (*InternalOrExternalBoundary*). Both have a big impact when generating information for indoor positioning. Determining whether the boundary is physical or virtual will define the accessibility of the space on the other side of the boundary. In the case of a virtual boundary, there is

the need for a connection object on its location, as the boundary should be possible to cross when there is no physical obstruction preventing movement between the spaces. The information about a boundary being external or internal helps with ascertaining whether the boundary connects to another space or to the exterior, thereby avoiding additional searching for a connected space in the case of an exterior boundary segment.

Table 23: IfcRelSpaceBoundary

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
RelatingSpace	IfcSpaceBoundarySelect (SELECT)
RelatedBuildingElement	IfcElement (ENTITY)
ConnectionGeometry	IfcConnectionGeometry (ENTITY)
PhysicalOrVirtualBoundary	IfcPhysicalOrVirtualEnum (ENUM)
InternalOrExternalBoundary	IfcInternalOrExternalEnum (ENUM)

These are the fundamental entities that directly address the subject of our research, namely, extracting data from an IFC file and creating geometric and topological information for the creation of an Indoor Positioning building model. However, we do require some additional data to complete the model. The following objects generally do not provide any additional geometric data. The purpose of these objects is to define the hierarchical structure of our building model and place the model into georeferenced maps.

IfcBuildingStorey defines the vertical arrangement of the building by storing individual storeys. Each storey has an assigned elevation in the *Elevation* attribute that is a real number. This attribute proves useful, especially when all *IfcSpace* entities on a particular storey are on the same height, thereby eliminating the need to deal with the Z coordinate when creating IPS *Area* geometry. The attributes and their types can be observed in Table 24.

Table 24: IfcBuildingStorey

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
LongName	IfcLabel (STRING)
CompositionType	IfcElementCompositionEnum (ENUM)
Eleveation	IfcLengthMeasure (REAL)

The *IfcBuilding* entity defines individual buildings and since the IPS XML can include several buildings, this entity has to be considered in the data extraction process. Individual buildings can be placed into a global coordinate system and an address assigned to each of them with the *BuildingAddress* attribute.

```
#104=IFCBUILDING('1obQpJ1wf95we7UfQWOSGJ',#41,'',$,$,#32,$,'',
     .ELEMENT.,$,$,#100);
#113=IFCBUILDINGSTOREY('1obQpJ1wf95we7UfPVdZhL',#41,'Level 1',
     $,$,#111,$,'Level 1',.ELEMENT.,0.);
#119=IFCBUILDINGSTOREY('1obQpJ1wf95we7UfPVdZbK',#41,'Level 2',
     $,$,#118,$,'Level 2',.ELEMENT.,4000.);
#1492=IFCRELAGGREGATES('27PCKGLxT4mxtV9cw6mgBW',#41,$,$,#104,
     (#113,#119));
```

Code 7: The relationship between an IfcBuilding element and IfcBuildingStorey elements (references are bold)

Moreover, the elevation of the surrounding terrain at the building perimeter can be found in the *ElevationOfTerrain* attribute, and the reference height of the building in the *ElevationOfRefHeight* attribute. *IfcBuildingStorey* elements are related to the *IfcBuilding* that they are members of with the *IfcRelAggregates* relationship entity (see Code 7). These and other attributes of the *IfcBuilding* are listed in Table 25.

Table 25: IfcBuilding

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
LongName	IfcLabel (STRING)
CompositionType	IfcElementCompositionEnum (ENUM)
ElevationOfRefHeight	IfcLengthMeasure (REAL)
ElevationOfTerrain	IfcLengthMeasure (REAL)
BuildingAddress	IfcPostalAdress (ENTITY)

IfcSite contains information about the surroundings of buildings, particularly the real estate belonging to the building or a group of buildings. It provides the latitude and longitude coordinates, which place the site in the geographic coordinate system with the *RefLatitude* and *RefLongitude* attributes. Furthermore, the height of the site is determined separately by *RefElevation*. Similarly to the *BuildingAddress* attribute of the *IfcBuilding* entity, the address of the site can be obtained from the *SiteAddress* attribute. The *IfcSite* entity attributes are listed in Table 26.

The *IfcRelAggregates* entity again serves as a means to link the *IfcBuilding* objects to the *IfcSite* object they reside on.

```
#104=IFCBUILDING('1obQpJ1wf95we7UfQWOSGJ',#41,'',$,$,#32,$,'',
     .ELEMENT.,$,$,#100);
#1352=IFCSITE('1obQpJ1wf95we7UfQWOSGG',#41,'Default',$,'',#1351,
     $,$,.ELEMENT.,(42,21,30,344238),(-71,-3,-35,-194702),0.,
     $,$);
#1478=IFCRELAGGREGATES('0FcrEmaGbDl8E_BqaIGcbN',#41,$,$,#1352,
     (#104));
```

Code 8: Relationship between an IfcBuilding and the IfcSite the building is located on

Table 26: IfcSite

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ObjectType	IfcLabel (STRING)
ObjectPlacement	IfcObjectPlacement (ENTITY)
Representation	IfcProductRepresentation (ENTITY)
LongName	IfcLabel (STRING)
CompositionType	IfcElementCompositionEnum (ENUM)
RefLatitude	IfcCompoundPlaneAngleMeasure (AGGREGATE)
RefLongitude	IfcCompoundPlaneAngleMeasure (AGGREGATE)
RefElevation	IfcLengthMeasure (REAL)
LandTitleNumber	IfcLabel (STRING)
SiteAddress	IfcPostalAddress (ENTITY)

With the *IfcSite* entity*,* the geometry and structure of the site and building model are complete. We still have to address the connectivity data. As was laid out in 2.1.1.2, the connectivity data can come from the Connectivity entity types. The entity *IfcRelConnects* (Table 27), which is the abstract supertype of several connection type entities, was singled out as the most suitable candidate. The subtypes of *IfcRelConnects* define the applicable object types for the particular connection and the semantics of that connection. The relationship we are looking for is the connection between two *IfcSpace* entities. Considering this requirement, we have focused on the *IfcRelConnectsElements* entity (Table 28) under the assumption that the supertype of the *IfcSpace* entity is the *IfcElement* entity. In addition to the Attributes of *IfcRelConnects,* the *IfcRelConnectsElements* entity includes the connecting and connected element of the *IfcElement* type and the geometry of the connection in the *ConnectionGeometry* attribute, which references geometry entities of the IFC schema, thereby determining the geometry of the connection.

Table 27: IfcRelConnects

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)

After some discussion about the subject with Chi Zhang of the Eindhoven University of Technology, it was quickly established that *IfcSpace* is not a subtype of *IfcElements* and is in fact based on the *IfcSpatialStructureElement* leading us to the base type of IfcSpatialElement. The fact that the supertype of *IfcSpace* is not *IfcElement* consequently excluded the possibility of using *IfcRelConnects* as a source of connectivity data. No other entity could be identified that could fulfil this role. Thus, alternatives had to be considered and will be discussed in more detail in the following chapters.

Table 28: IfcRelConnectsElements

Attribute	**Type**
GlobalId	IfcGloballyUniqueId (STRING)
OwnerHistory	IfcOwnerHistory (ENTITY)
Name	IfcLabel (STRING)
Description	IfcText (STRING)
ConnectionGeometry	IfcConnectionGeometry
RelatingElement	IfcElement
RelatedElement	IfcElement

The attributes that are specific to each class were explained in detail in the previous paragraphs for individual IFC entities. There are, however, other attributes on the top of each entity attribute table that were overlooked and are common to most if not all IFC entities. These attributes are derived from the general parent entities of core IFC schemas, for example, *IfcRoot*, *IfcElement*; that individualize every object in a building model through a unique ID, name, object placement and representation. The following attributes can be identified:

- *GlobalId* – unique ID for every object
- *OwnerHistory* – ownership history of the object
- *Name*
- *Description* – optional textual description
- *ObjectType*
- *ObjectPlacement* – references the placement of the object in the global IFC coordinate system of the model
- *Representation* – references the representation of the object and further references to geometrical data
- *LongName* – optional long name
- *CompositionType* – in case of composed objects this attribute will reference the composition

Through object individualization it is possible to have several elements of the same entity (element type) in a building model.

3.2.2.2 IFC model lacking *IfcSpace* entitites

Despite the fact that the *IfcSpace* provides very specific information such as area, volume, occupancy, etc. about the closed spaces, located in the building interior, it has been recognized by Daum, et al. (2015) that it is by no means a rule that every building model also contains these *IfcSpace* objects. As a consequence, building models without *IfcSpace* objects make up a significant part of available building models and, therefore, have to be taken into consideration.

These are the two possible options of approaching the generation of information for an indoor positioning model:

1. Generate the information from the existing incomplete model with the help of other available objects,
2. Generate *IfcSpace* objects first and make use of the "corrected" complete building model for data extraction.

Looking at the first option, the geometry would have to be generated from *IfcWall* objects and the topological information would come from *IfcDoor* objects. The first part, creating the geometry from *IfcWall* objects, is straightforward. First of all, we have to

look at an *IfcWall* and how this entity is defined in the IFC Schema, we have to find out what kind of information can be extracted from the attributes and which other entities an *IfcWall* has to reference to round up a complete wall representation without generating redundant information. The challenge when using the *IfcWall* entity for geometry is that a wall does not necessarily end at the edge of a space (see Figure 29 below), thereby requiring additional operations for determining intersections. It also has to be noted that the shape definition for *IfcWall* entities is a combination of a simple 2D curve and an extrusion for the 3D representation, as can be observed in the code snippet in Code 9.

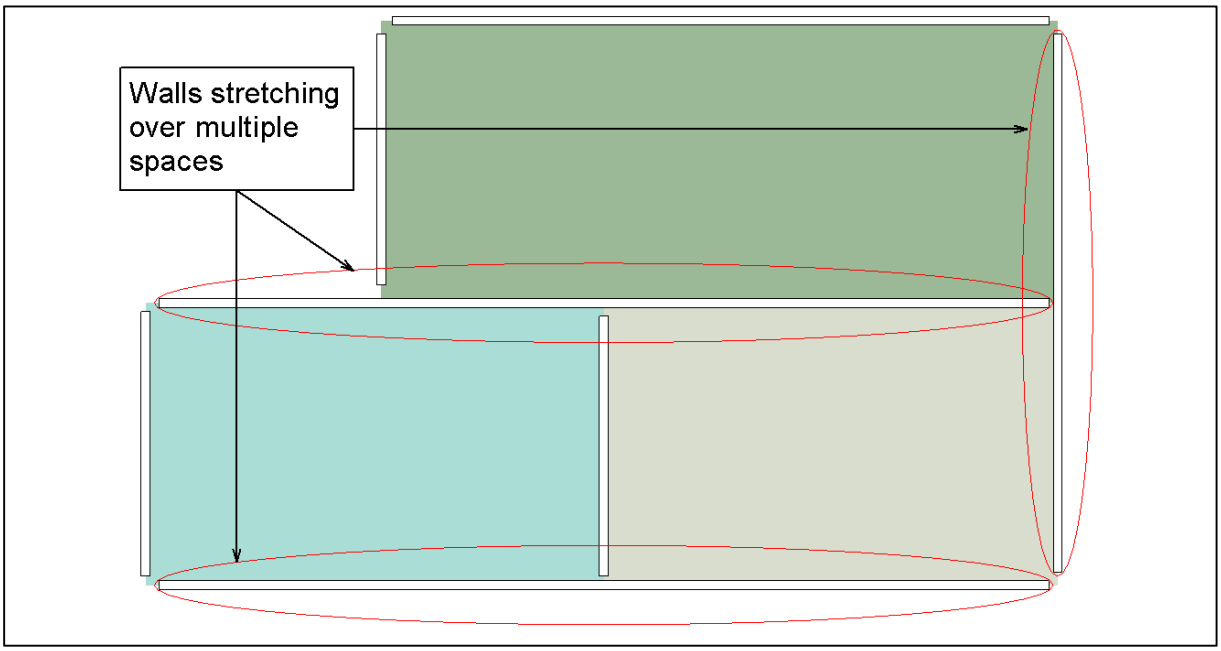


Figure 29: Walls that stretch over, and thereby define multiple spaces

Unfortunately, the explicit attributes of *IfcDoor* do not provide any direct means of identifying the connectivity between spaces. Walls that contain a door and are shared by separate rooms have to be identified. In IFC terms this means that an *IfcWall* object that has an opening filled with an *IfcDoor* object, and is the boundary of two separate *IfcSpace* objects, would be a potential candidate for these two *IfcSpace* objects connecting. However, this is not necessarily the case. An *IfcWall* object can be the boundary of more than two *IfcSpace* objects (see Figure 29), in which case the *IfcDoor* object in this *IfcWall* object can theoretically connect any pair of the *IfcSpace* objects that are bounded by the *IfcWall*. This slight issue can be solved with the geometrical data from *IfcSpace*. By locating the *IfcDoor* and *IfcSpace* boundary seg-

ments in relation to each other we can establish whether the door lies within any of the boundary segments, and consequently, whether it connects the related spaces or not.

Another possibility that we looked into was the use of the inverse attributes of *IfcDoor*; the *IfcRelConnects* and its subtype entity *IfcRelConnectsElements* showed some promise of potentially providing the missing connectivity information between spaces. But as we established in the previous chapter the *IfcRelConnectsElements* entity connecting element attributes do not correspond to the entity type of *IfcSpace*. In that respect, we could not reproduce the utilization of the *IfcRelConnects* element in conjunction with *IfcDoor* elements in the exported IFC files while working with the current export tools of commercial BIM applications. The usage of *IfcRelConnects* relationships was limited to the connections of individual walls to each other through the *IfcRelConnectsPathElements* subentity. Unfortunately, other *IfcRelConnects* subtypes were even farther from providing a potential match. Truthfully, even if *IfcRelConnects* could provide the right type of derived entity for the purpose of connecting spaces in the near future, it cannot be expected that the AEC industry would be willing to create this type of additional information, since the main functionality of applications that can exploit space connectivity information are beneficial for the industry only in niche cases. It would also create additional work for the creator of the model under the assumption that no algorithm in the background would be taking care of this task automatically, and manual input of the model creator would be required.

```
#271= IFCSHAPEREPRESENTATION(#43,'Axis','Curve2D',(#269));
#287= IFCSHAPEREPRESENTATION(#44,'Body','SweptSolid',(#278));
#289= IFCPRODUCTDEFINITIONSHAPE($,$,(#271,#287));
#291= IFCWALLSTANDARDCASE('2Uz$rHhK96aA7mOWJEWz1P',#52,'Basic Wall
     :Generic - 200mm:193825',$,'Basic Wall:Generic - 200mm:249',
     #266,#289,'193825');
```

Code 9: Entities defining the geometry of an IfcWallStandardCase

In the second approach it was suggested to first create *IfcSpace* objects along with connectivity information; in a way enhancing the model with the objects and information we need, i.e., perfecting it. The enhanced IFC model can then be used as it was originally intended. As developed by (Daum, et al., 2015) a method of using octrees and flooding algorithms can be used to create *IfcSpace* objects and connectivity information on the basis of an existing imperfect IFC building model. The method itself generates adjacency/accessibility graphs in the process, but in an intermediate step before creating the graphs *IfcSpace* objects are created from existing building elements for every storey. With a directional-based QL4BIM query or by using the *IfcRelContainedInSpatialStructure* relationship building elements from one storey can be isolated and extracted from the model. The results of this query are *IfcProducts* and their BRep geometry structures that are to be used further. The first step is checking the existing geometry for unwanted holes (everything not a transition), such as breakthroughs in slabs. These objects have to be sealed, meaning temporarily removed, for the flooding algorithm to work.

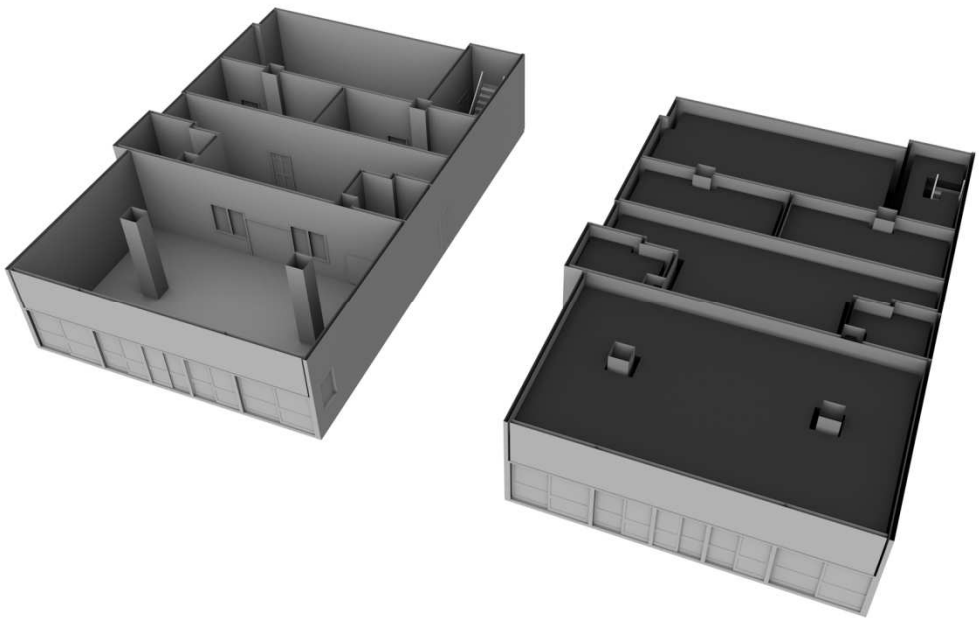


Figure 30: The representation of the storey on the left and the created IfcSpece objects right (Daum, et al., 2015)

From the BRep of the storey that the data was extracted from, a three-coloured octree data structure of that storey is created. These octants are coloured white, grey and black, with each colour representing a special attribute of a particular octant;

white being an exterior octant, black an interior octant, and grey an octant representing the boundary of the geometry. Only the black octants are needed for the generation of *IfcSpace* objects, therefore the white and grey octants can be deleted. Connected clusters are created from the black octants, where the bigger clusters represent actual *IfcSpace* objects, and the smaller clusters, which cannot represent *IfcSpace* objects, are deleted. An example of these smaller clusters appearing is when clusters have to be created inside of column geometry. The geometry of *IfcSpace* objects is represented by a BRep structure that is created on the outer hulls of octree clusters (see Figure 30).

Summing up, we can see that dealing with IFC files lacking *IfcSpace* elements already presents a challenge in the department of defining the geometry of spaces. Consequently, creating connectivity information is also hindered. In general, more effort is required to create the topology of a building, but it is still possible.

3.2.2.3 IFC model containing the *IfcSpace* entity

This is the desired and expected scenario for the IFC building models currently available. Creating space objects in most contemporary applications for building modelling is straight-forward and offers a wide variety of options for the storage of specific information in a building model such as energy analysis. Once these Space objects have been created in a building modelling application they can then be exported to the IFC format in the form of an *IfcSpace* entity.

The *IfcSpace* entity is crucial for extracting data from the IFC format in our chosen approach. It defines the space volume inside a building where movement is possible, thereby providing the basic environment for indoor positioning. The geometry is available either through the *ShapeRepresentation* attribute of *IfcSoace* or the *IfcRelSpaceBoundary* entity. The second option with the *IfcRelSpaceBoundary* entity is better and more convenient as a result of several aspects. A single *IfcRelSpaceBoundary* defines a polyline that represents the boundary segment of a referenced *IfcSpace* object. If a space is accessible, at least one of these segments represents a door. As a rule, the *ShapeRepresentation* attribute of *IfcSpace* only offers a shape's type and dimensions in the local coordinate system with the position of the element inside the global IFC coordinate system through the *ObjectPlacement* attribute. The

coordinates of *IfcRelSpaceBoundary* are already in the global IFC coordinate system thereby avoiding the additional step of allocating the *IfcSpace* object in the global coordinate system, and with the segments that reference *IfcDoor* objects data, which helps with determining connectivity, the information is also readily available.

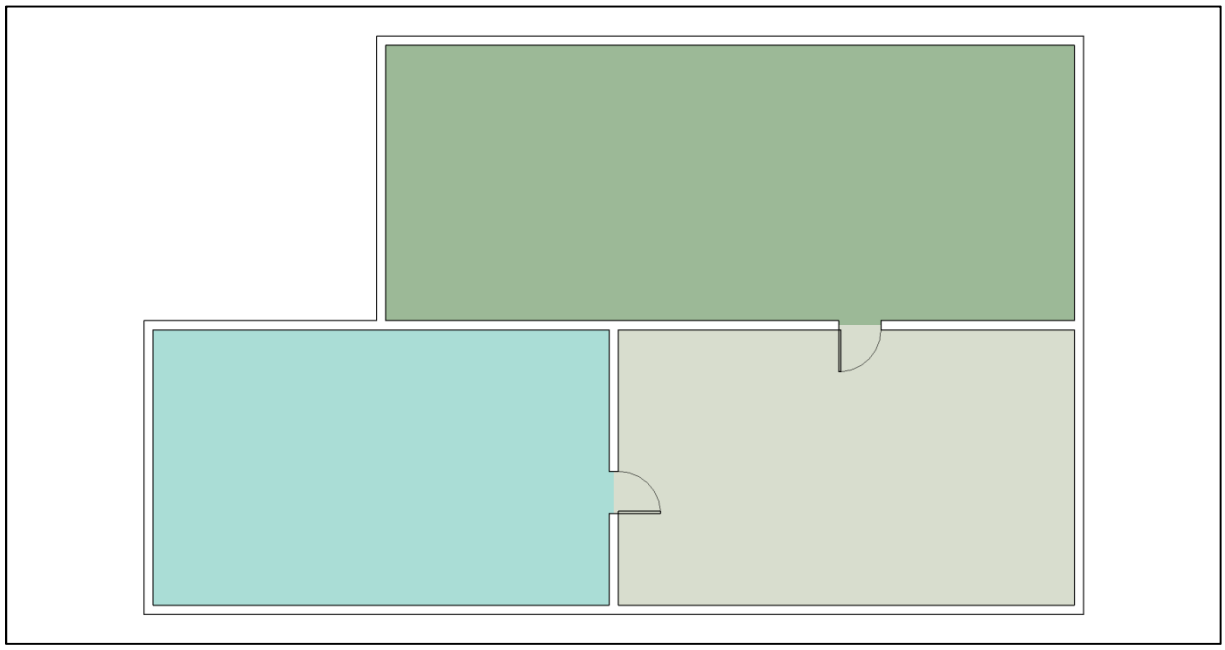


Figure 31: Multiple spaces sharing one wall containing a door

In chapter 3.2.2.2 we have established that when only *IfcSpace* objects are present in a building model, generating connectivity information requires some additional effort because it can only be generated through *IfcWall* objects that are referenced by two *IfcSpace* objects and contain an *IfcDoor*. Even then there are cases in which this information does not suffice, e.g., when more than two *IfcSpace* objects share the same *IfcWall* that contains an *IfcDoor* (see Figure 31) we cannot say for certain which spaces are connected or if two spaces share the same wall but not the door located in the wall (Figure 32); and an additional step has to be taken to determine the position of the *IfcDoor* in relation to all of the *IfcSpace* objects. If the *IfcDoor* is geometrically located in a segment of a particular *IfcSpace* object, then the *IfcSpace* is accessible through the door. The two *IfcSpace* objects that fulfil this condition are connected by the *IfcDoor*.

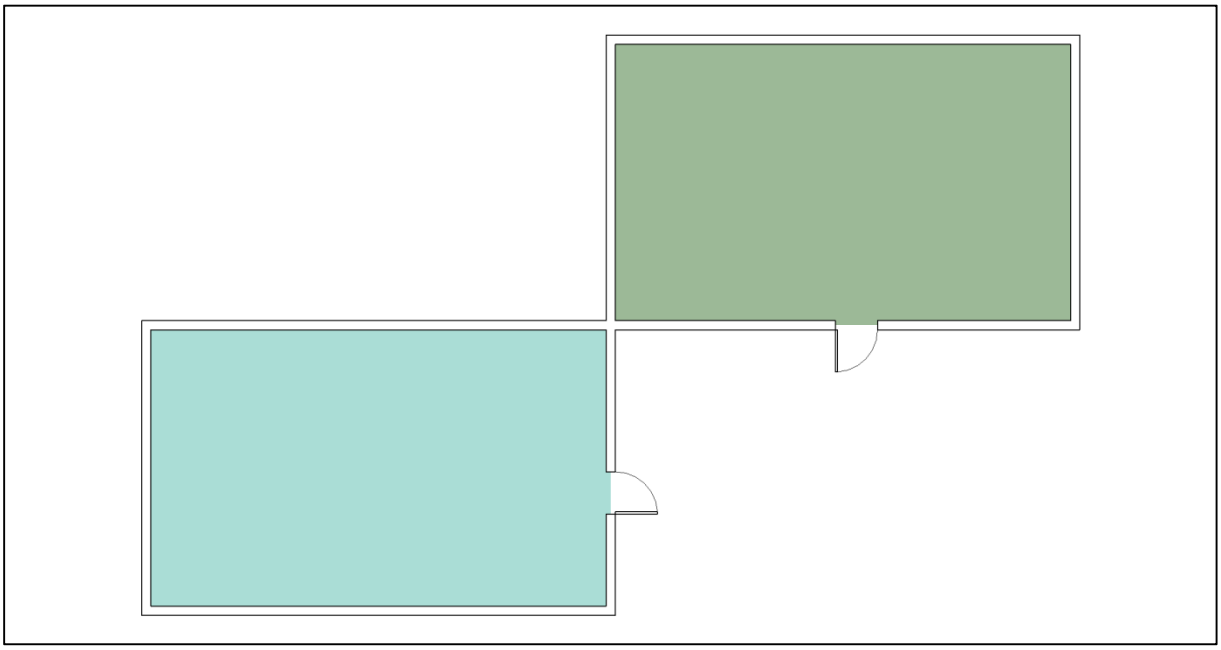


Figure 32: Rooms referencing the same wall and not sharing a door

Because of this fundamental difference in acquiring data about the spaces and the connectivity between them, two separate cases will be analysed in the following chapters. In one case the *IfcRelspaceBoundary* in the building model is at our disposal, in the other it is not. We have used and tested these cases by generating IFC files from two different building modelling applications. The Graphisoft Archicad application was used when the *IfcRelSpaceBoundary* entities were missing, whereas when the *IfcRelSpaceBoundary* entities were available, Autodesk Revit was employed. The reasoning behind using two different applications for generating IFC files was two-fold. On the one hand, we had many more building models created and exported from Revit available for testing. Mainly because the testing model subjects came from models created by students for the BIM lecture offered by the Institute of Building Informatics at the TU Graz, and most students decided to create their projects in Revit.

On the other hand, the Revit IFC export module did not offer as many options for exporting building models to the IFC format as was expected. The Graphisoft Archicad module provided a much wider range of options for exporting into the IFC format in comparison to Autodesk Revit. The IFC Manager provides a hierarchical overview of the current model in the IFC format, providing several functions that allow for the editing of the entities in the IFC model (Graphisoft, 2013). Another advantage is that the

Archicad IFC export module provides more MVDs as export options as well as the possibility of working with custom model view definitions and making use of more than just the provided MVDs. If this export option eventually allows for the import of MVD specifications, it will be crucial for the new approach, which will be discussed later, that uses existing building models for Indoor Positioning without the intermediate step of data extraction and conversion by making use of a well-defined and officially certified MVD for Indoor Positioning.

The downside of Archicad and also the main reason it was not used as extensively for testing purposes (apart from specific purpose models) was that it has proven more difficult to acquire IFC files exported from Archicad or Archicad models to export from sources other than the students of the BIM course. However, one complete model could be acquired in the scope of the (Haas, 2013) bachelor thesis, and some special smaller models were created for controlled environment scenarios (e.g. model contained walls only) that had to be analysed individually to create an independent and robust algorithm.

The following two chapters will deal with these two special cases, where the IFC building model does contain *IfcSpace* entities and either does or does not contain the *IfcRelSpaceBoundary* entity. The second case in which the *IfcRelSpaceBoundary* is available in an IFC building model is of special interest, since our algorithm was developed on that basis.

3.2.2.4 IFC without IfcRelSpaceBoundary

In the case of an IFC file with no *IfcRelSpaceBoundary* entities we mostly worked with purposely created building models when developing and implementing the code. For final code testing the one complete building model was utilized. The platform that was used to generate the building models was exclusively Graphisoft Archicad, for reasons discussed in the previous chapter (3.2.2.3). An IFC file generated without the *IfcRelSpaceBoundary* entity results in a file size somewhat smaller, compared to when the option is enabled.

The fact that there is no *IfcRelSpaceBoundary* entity in the IFC building model means that geometry and topology information have to be generated somewhat differently and in a more complicated manner, since we cannot rely exclusively on

IfcRelSpaceBoundary to provide the link from the element defining the boundary segment (*IfcWall* or *IfcDoor*) to the *IfcSpace* element the boundary belongs to. The entity *IfcSpace* still forms the basis for the framework because it defines closed spaces compared to data coming from *IfcWall* elements, where closed spaces have to be defined first through individual wall intersections. As a result, instead of relying on only one additional entity (*IfcRelSpaceBoundary*), we must consider the *IfcWall* and the *IfcDoor* location in particular, as well as geometrical representation in relation to the *IfcSpace* boundary segments, to determine the adjacency and, more importantly, the connectivity of *IfcSpace* objects; therefore, some additional steps need to be taken to acquire the same quality of data.

First we have to determine the relationship between the *IfcDoor* and *IfcWall* objects. The *IfcDoor* entity represents the transitional object between spaces (*IfcSpace*) where movement is allowed, thus connecting them and enabling crossing over from one space to another. *IfcDoor* by itself does not hold any information about connectivity. It does, however, determine the location and the shape of the door. This information in conjunction with other geometrical and semantic data can be used further to create connectivity information.

The other entity is the *IfcWall*. It forms the boundary to the space and the carrier of the transition object, the *IfcDoor*. One condition must be satisfied: the *IfcWall* can only be transitioned at the location of an *IfcDoor*. An *IfcWall* object can be associated with a particular *IfcDoor* object through the opening that the *IfcDoor* cuts into the *IfcWall* (seeCode 10). The following entities define an opening:

1. *IfcRelVoidsElement* – the relationship between an element (creating a void in it) and an opening element.
2. *IfcOpeningElement* – represents a void in any element that has a physical manifestation.
3. *IfcRelFillsElement* – the relationship between an opening element and an element that fills.

In general, geometry is cut from an object by establishing a relationship (*IfcRelVoidsElement*) between the object that is being cut (*IfcWall*) and an opening object (*IfcOpeningElement*). The *IfcRelFillsElement* entity defines the relationship of

an object filling an opening object, in our case connecting the *IfcDoor* and the *IfcOpeningElement*. Through these relationships and geometry, cutting the *IfcDoor* can be indirectly related to the *IfcWall*.

```
#478=IFCWALLSTANDARDCASE('2Uz$rHhK96aA7mOWJEWz1S',#52,
     'Basic Wall:Generic - 200mm:193828',$,
     'Basic Wall:Generic - 200mm:249',#459,#476,'193828');
#566=IFCOPENINGELEMENT('2EiZacDmT5R90RDMfpSESz',#52,
     'M_Single-Flush:0915 x 2134mm:0915 x 2134mm:193959:1',
     $,'Opening',#565,#560,$);
#567=IFCRELVOIDSELEMENT('3T7EQYu2n64uILIoXyOxJ7',#52,$,$,#478,
     #566);
#703=IFCDOOR('2Uz$rHhK96aA7mOWJEWz3V',#52,'M_Single-Flush:
     0915 x 2134mm:0915 x 2134mm:193959',$,'0915 x2134mm',
     #702,#695,'193959', 2134.,915.);
#759=IFCRELFILLSELEMENT('3gargcnzX1qvJklsHdp7gB',#52,$,$,#566,
     #703);
```

Code 10: Indirect relationship of IfcWallStandardCase and IfcDoor in the IFC EXPRESS schema file

When establishing connections between *IfcSpace* objects, only *IfcWall* objects containing *IfcDoor* objects are relevant, since the rule is that transitions between spaces are only possible through doors. With the connection between *IfcDoor* and *IfcWall* established, what remains to be determined is which two spaces are linked and made mutually accessible by that particular wall containing a door. There are various ways of accomplishing this task, one of which has been described by Daum et al. (2015). It is based on the method for automatic generation of *IfcSpace* objects from existing data in the IFC file by flooding the considered space with octants, which has been described in the previous chapter. Similarly, existing building blocks, such as walls and doors, can be converted into octants as well. By scaling the octants generated by converting existing *IfcWall* and *IfcDoor* geometry, and thereafter determining the amount of intersecting octants, when the amount reaches a certain threshold, one can establish adjacency, in the case of walls, and connectivity, in the case of doors (see Figure 33).

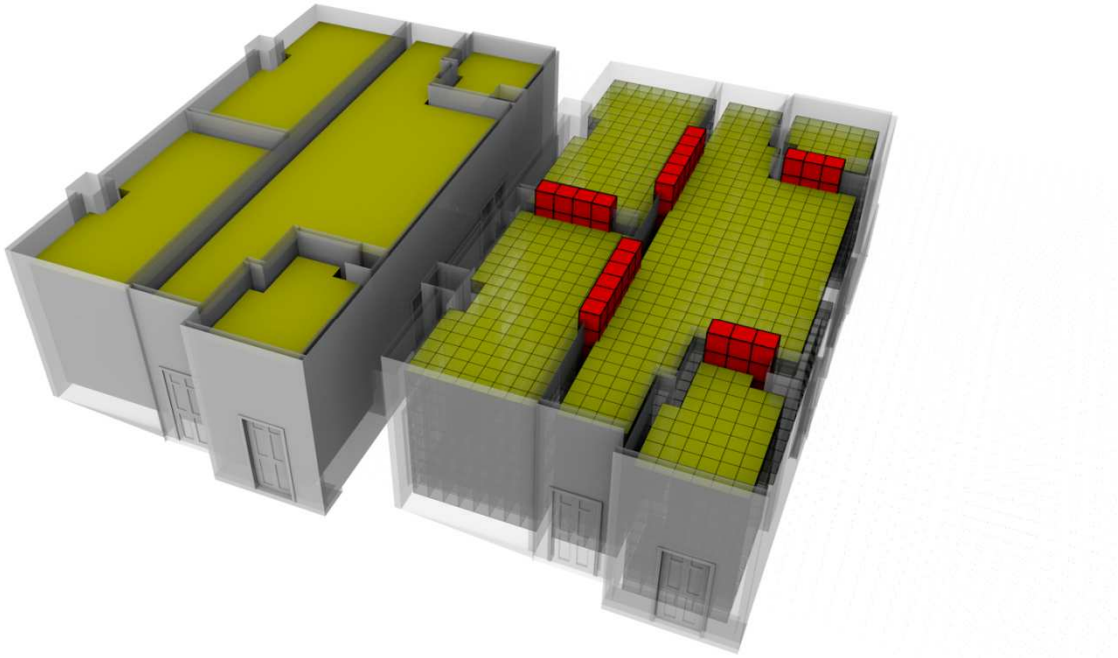


Figure 33: Left storey with IfcSpace entities, right scaled octants of opening elements lifted in red (Daum, et al., 2015)

The main issue with this kind of IFC file is that there is no direct access to information regarding which walls define (are the boundaries of) which space. Therefore, the only means of generating adjacency and connectivity information is by leaning on the geometry data of spaces and walls, and determining the overlapping geometry of a bounding wall and two adjacent spaces. Since the only issue in our case is connectivity, the geometry overlap needs to be established only for walls that were proven to contain a door. Determining the adjacency of spaces is, therefore, only important for the spaces that are bound by a wall containing a door and is, not required to be established for all spaces and walls, since it is not crucial for creating the IPS BIM. This is why no particular emphasis has to be put on establishing the adjacency for all spaces, thereby narrowing down the necessary operations. This approach, in which we have to compare coordinate values, is not completely reliable for many reasons, e.g., tolerances, limited memory space for computer variables etc., and can lead to unexpected errors. A workaround to the direct comparison of coordinate values would be to determine only the general position of elements (orthogonal projections) in relation to each other. As a result, the described method of determining connectivity was not implemented fully, partly because the octree method suggested by Daum et al. (2015) does the same, and should generally produce more reliable results than

comparing coordinate values. Consequently, the author's suggestion for advancing in this area would be to adjust Daum's algorithm and implement it or continue developing code on the suggested workaround.

3.2.2.5 IFC with IfcRelSpaceBoundary

Most, if not all recent IFC files take advantage of the *IfcRelSpaceBoundary* entity to describe interior spaces. The narrow specialization of this particular entity makes it the ideal candidate for the extraction of building floor plan geometry in open building spaces. Additionally, as the entity is not independent (i.e. it is not directly created by the user) it has to reference its parent entity, and thus can be traced back to the element (see chapter 3.2.2.1 Table 23 the attribute *RelatedBuildingElement* of the type *IfcElement*) that is responsible for the creation of any particular space boundary segment. We have been able to identify the following two separate entities that can define *IfcRelSaceBoundary* objects, and consequently form the boundary of spaces (*IfcSpace*):

- *IfcWall* and its subentities *IfcWallStandardCase* and *IfcWallElementedCase* – for boundary segments formed by walls,
- *IfcDoor* – for boundary segments formed by doors.

This fact has been exploited to create the connectivity information crucial for any usable IPS XML model. Acquiring multiple models of this type was relatively simple, since we were able to use the Revit files created by students of the Building Information Modelling course.

Essentially, the *IfcRelSpaceBoundary* entity is used as the fundamental source of data. It supplies all geometry data for individual floor plans that are created by interconnecting IPS XML schema *Area* class objects, as well as the location of *LinePair* class objects that define Connection class objects of the IPS XML schema, which in turn define the connectivity of Area objects. Needless to say, entities that relate to the *IfcRelSpaceBoundary,* such as *IfcDoor, IfcWallStandardCase,* as well as space defining entities, such as *IfcSpace*, *IfcFloor* etc. and their semantics, still form the backbone when defining the spatial structure and topological information of the building. Through these entities the relationships between elements, and consequently the

framework for the topological map of the IPS XML building model, can be established.

The principle thought is to first filter the IFC building model file storey by storey for all available *IfcRelSpaceBoundary* objects. However, this is not the only required way of filtering and differentiation when dealing with boundary segments, since we also have to establish the assignment of *IfcRelSpaceBoundary* objects according to which type of boundary they represent (either formed from the *IfcDoor* or the *IfcWall*). This step is crucial since the boundaries from *IfcWall* and the boundaries from *IfcDoor* have a fundamental difference in the function they must fulfill. The *IfcWall* is supposed to represent a non traversable boundary that keeps a space closed, whereas the *IfcDoor* is supposed to form a connection between two distinct spaces where it is possible to cross over from one space to the other, i.e., leave one space and enter the other. No additional effort is required to assign the boundary segments to spaces since that information is in the parameters of the *IfcRelSpaceBoundary* entity under the *RelatingSpace* attribute (Table 23), thereby specifying and assigning boundaries as a series of boundary segments to individual spaces.

The next chapter will focus on the case of relying on *IfcRelSpaceBoundary* elements in the IFC file. The developed algorithm will be analysed step by step and certain test cases will be discussed.

3.2.2.6 Generating the IPS XML from data extracted from IFC

For the obvious reasons already discussed in chapter 3.2, IFC was chosen as the standard building model format that IPS XML will rely on as the basis for building data extraction. The idea behind this approach is relatively simple: select a standard that provides most, if not all, required data for the generation of information for IPS XML, and then create an algorithm that will produce a valid and complete IPS XML file. The important steps the algorithm for the IFC schema has to cover are as follows:

1. Filter the IFC entities that were identified as essential from the IFC file,
2. Break down every object from these entities into an ID, the name of the entity and the parameters,

3. From the parameters, save the valid data and follow the references to other required objects,
4. Besides extracting data from IFC elements, create the appropriate IPS XML objects,
5. Whichever IPS XML objects and information cannot be created directly from building data require further processing:
    a. *Connectivity* information,
    b. *LinePair* objects,
6. Export the assembled IPS XML model to an XML file.

The following chapters will introduce the steps listed above separately. All steps were analysed individually and algorithms for their solutions were created. Afterwards, each individual step was coded in C# as a separate class, resulting in a simple application for Windows.

The class framework (Appendix 4) of the application, as well as activity diagrams representing parts of the algorithm, can be found in the Appendix. The activity diagram has been dissected into three parts with the main thread depicted in Appendix 1, the connection definition and list in Appendix 2, and the adjusting of connection geometry in Appendix 3.

3.2.2.7 The algorithm for filtering the IFC EXPRESS file

An algorithm has been developed in collaboration with Maximilian Haas (2013) in the scope of his bachelor project at the Institute of Building Informatics at the TU Graz. As it was ascertained, the *IfcSpace* forms the basic framework from which data can be extracted to generate the required information. The *IfcSpace* holds data about its location and geometrical shape in two attributes. The location can be determined by following the reference under the *ObjectPlacement* attribute, and the shape can be acquired from the Representation attribute (chapter 3.2.2.1 Table 18), leading to shape representation entities.

The algorithm starts with the content of an IFC EXPRESS file being read and passed on to the selected reader class, depending on whether *IfcRelSpaceBoundary* entities are present in the file or not. Reading is commenced at the body of the IFC file, which begins after the keyword DATA (see Code 5 chapter 3.2.2) of the IFC file, while the

header (see Code 4 chapter 3.2.2) is left out. After the content of the file is read it gets passed over to the appropriate converter class according to the choice about *IfcRelSpaceBoundary* entities made by the user in the form of the *IFCFile* class. For working with IFC entities the IFC TOOLS Project (2013) was considered but decided against. The reason for this was the state of the license when the algorithm was developed. Key requirements also had to be fulfilled, which meant that it should be possible to use the algorithm commercially as well as for it to remain independent from exterior sources.

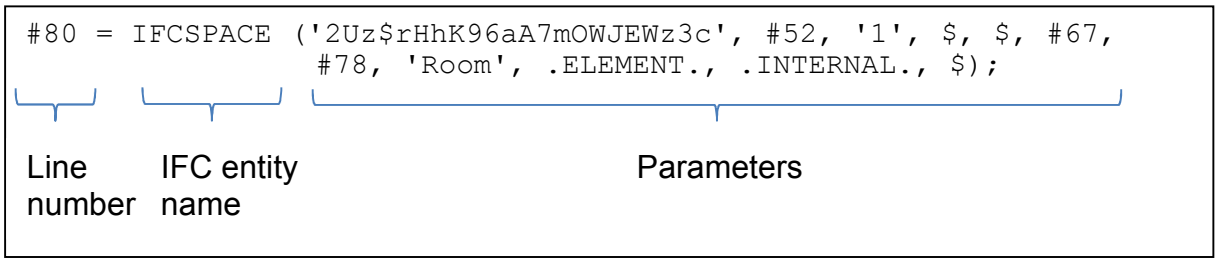


Figure 34: Dissection of a typical line defining an IFC element from an IFC file according to IFCTextLine definition

The first step in the filtering algorithm is to collect all desired entities from the IFC building model. For this purpose, a filter or query was developed as a method of the *IFCFile* class that, with the use of certain keywords for IFC entities e.g. "*IfcWallStandardCase*", can create a list of all of those particular objects from the IFC building model in question for further use. Line by line the entire IFC file is read, and for every line an IFCTextLine object is created and added to a list. The *IFCFile* and the *IFCTextLine* classes are completely independent and can be utilized for cases other than the scope of our project. Wherever specific entities are needed and have to be filtered from the IFC file the query can provide a solution. The way in which *IFCTextLine* objects are created is that individual lines from an IFC file are broken down into the following three parts (as depicted in Figure 34):

1. Line number – stored as the integer ID in *IFCTextLine*,
2. IFC entity name – stored as a *String* keyword in *IFCTextLine*,
3. The parameters – following the entity label, the list of parameters in brackets is stored as a list of *String* objects (*List*<*String*>).

These three parts of an IFC line are essential when creating *IFCTextLine* objects, which cannot be created without them. Because every line in an IFC file represents one element of a specific IFC entity, every *IFCTexLine* object added to the list also represents an individual object of the same entity that the keyword variable defines. Since no special operations between objects are required and all required data is stored in the parameters, the class *IFCTextLine* offers a high enough level of detail and can thus be utilized without further diversification into individual IFC entity classes as a basis for the generation of IPS XML objects.

3.2.2.8 Generating IPS XML Area objects and connectivity between them

When analysing the IFC file, we can establish that despite the use of *IfcRelSpace-Boundary* connectivity between *IfcSpace* objects, which has been deemed a crucial aspect of a building model, it is still not directly available for Indoor Positioning. Therefore, a solution is needed to complete the conversion of data into the new format. Several different approaches already exist, some of which have been introduced in chapter 2.3. These can already be distinguished at their base, regarding how connectivity graphs are implemented. So far no common method has been established.

For the sake of a clearer description of the process we will focus on the relationship between the IPS XML Area and the *IfcSpace* entity. In our mission to create the IPS XML we have focused on IFC models with *IfcRelSpaceBoundary* entities to determine the connectivity. The IPS XML building model is special in that it is supposed to help with determining the current position; accordingly, it requires exact room geometry with transitions between rooms providing support for Indoor Positioning. The model is not a mere graph for navigation; although the navigation graph can easily be extracted from the building model because the outline of the model focuses on the topography of the building and has in fact already been implemented. In other words, the IPS XML is tailored to the need of adjusting and helping to determine the current position of the user by logical corrections. Therefore, the basic framework or layout is already determined; by this I am referencing the *Area* – connections between *Areas* relationship. Since we have already established how to differentiate between and group up individual *IfcRelSpaceBoundary* entity objects (chapter 3.2.2.5) the task ahead requires us to use these preliminary preparations and generate the missing information to complete the IPS building model.

Therefore, when building the IPS XML file we can distinguish between three basic separate steps in the listed order (see Figure 35):

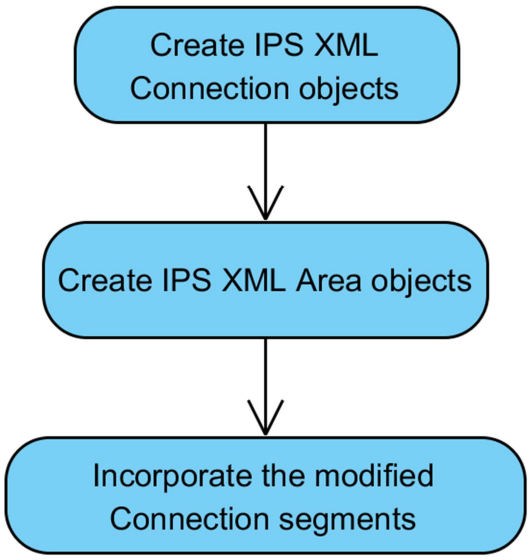


Figure 35: Building the IPS XML

1. Create IPS XML *Connection* objects by making use of the *IfcRelSpaceBoundary* objects that represent a door. Additionally, this phase also creates the connectivity information between *Area* objects.
2. Create IPS XML *Area* objects – different types of *Area* objects require different IFC entities as a basis, e.g., stairs, rooms, ramps and elevators are all represented as an *Area* object in IPS XML.
3. Incorporating the modified *Connection* segments into the corresponding *Area* object to create the correct geometry.

The *IfcRelSpaceBoundary* entity represents an individual boundary segment of an *IfcSpace* entity. Stemming from from this fact, generating the geometry of an IPS XML Area from *IfcSpace* is relatively straightforward. An individual *IfcRelSpace-Boundary* can be directly associated with an IPS XML *Line* object. Likewise, all *IfcRelSpaceBoundary* objects that reference the same *IfcSpace* object (forming the boundary of this space) can therefore be associated with the IPS XML *Line* objects that make up an IPS XML *Area*. Consequently, the *IfcRelSpaceBoundary* objects representing an *IfcDoor* are merely one type of boundary segment (see Figure 36 left-hand side). However, the IPS XML requires exact geometry even for these segments, and therefore some adjusting is required to achieve the desired end state presented on the right-hand side of Figure 36. In Figure 36, notice on the left-hand side where the IFC graphical representation resides that the wall segments of the walls containing doors remain uncut as one uniform segment as though no other element could interact with them. The door segments have been purposely translated from the wall plane for a better representation. On the left-hand side the geometry of the door segment has been corrected and represents the state in the intermediate step with a *DoorBuffer*, which is used for the IPS XML.

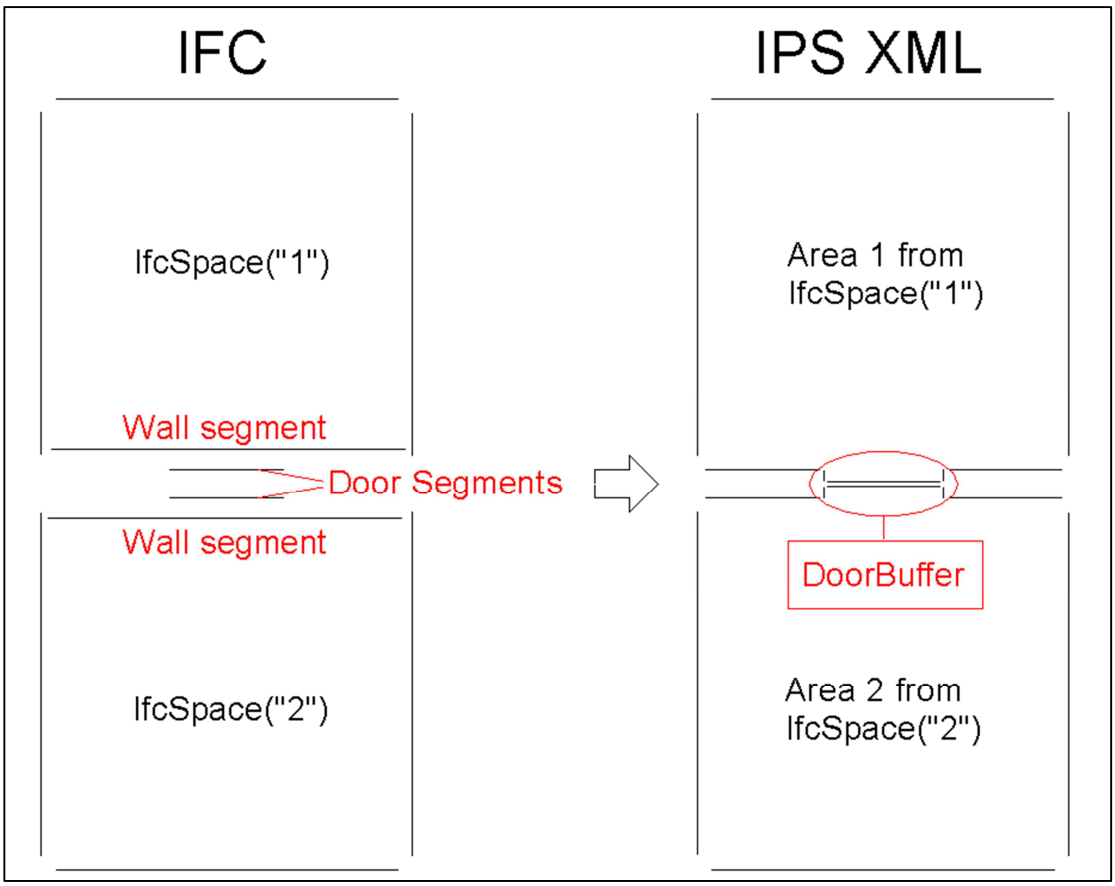


Figure 36: Boundary segment representation of IfcSpace with IfcRelSpaceBoundary on the left, and IPS XML Area with IPS XML Line on the right

From this initial analysis we can continue with the first phase of the algorithm, from which a code snippet can be observed in Code 11. The list of *IfcRelSpaceBoundary* objects that were created by the filtering algorithm (described in 3.2.2.7) has to be scanned for boundary segments representing *IfcDoor* objects, which is completed for every storey of the building. The *ReferencingElement* attribute can be queried for this information. Every interior *IfcDoor* has two *IfcRelSpaceBoundary* segments that reference it, which comes from the fact that a door connects two rooms (*IfcSpace*), and each one of these has one boundary segment dedicated to the *IfcDoor* to completely close the boundary polygon. Consequently, we can easily identify doors facing the exterior because these types of doors do not have a second *IfcRelSpaceBoundary* referencing them, since the exterior is not defined as an *IfcSpace*.

```
//Filter the IFC file for storeys and boundaries
IFCTextLine[] levels = ifcFile.filterIFCTextLines("IFCBUILDINGSTOREY");
IFCTextLine[] boundaries = ifcFile.filterIFCTextLines("IFCRELSPACEBOUNDARY");

foreach (IFCTextLine level in levels)
  { //for each level create a new floor and assign the ID
  floor = new Floor();
  floor.Id = level.ID;

  //DoorBuffer list
  List<DoorBuffer> doorBuffers = new List<DoorBuffer>();

  //Query for doors
  foreach (IFCTextLine boundary in boundaries)
  {
    //Determining the segment type (wall or door)
    IFCTextLine door = ifcFile.getIFCTextLine(boundary.getParameterAsInt(5));
    try
    {
      if (door.Keyword.Equals("ifcdoor"))
      {...
```

Code 11: Query the list of boundaries from an IFC file for IfcDoor segments

The boundary segments belonging to the same *IfcDoor* are paired up in a *DoorBuffer* (see Figure 37 and Figure 38) object. As well as storing the two boundary segments, the *DoorBuffer* class also incorporates a method (*createNewDoorGeometry()* see Figure 38) that creates the adjusted exact geometry by displacing the initial *IfcRel-SpaceBoundary* segment provided by the IFC building model and filling the gaps with two additional boundary segments for the IPS XML, thus closing the boundary polygon (see Figure 36 and Figure 37). The method is called as soon as a *DoorBuffer* object is completely populated, i.e., it contains the two relevant boundary segments from the IFC building model's *IfcRelSpaceBoundary*. Therefore, no additional action is required once the door segments have been paired.

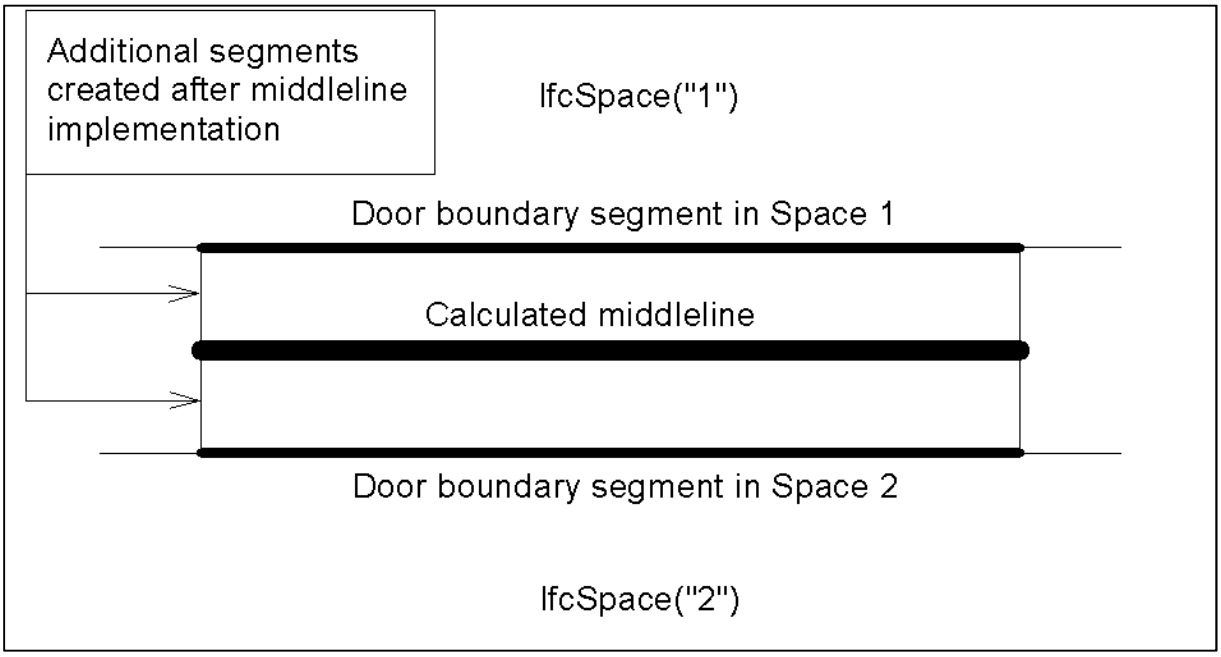


Figure 37: Graphical representation of DoorBuffer geometric attributes

By completing the *DoorBuffer* list for one storey we can move on to the rest of the *IfcRelSpaceBoundary* objects in our initial list (from this same storey). These generally reference one of the IFC wall entity types described in 3.2.2.1. As walls are predominantly of the vertically extruded type these building elements are represented by the *IfcWallStandardCase* entity in the IFC schema. Similarly to the boundary segment referencing the same *IfcDoor* object, the wall boundary segments defining the boundary of one *IfcSpace* can also be identified by looking up their references to the same *IfcSpace*. Unlike querying the *RelatedElement* attribute for the type of element defining the boundary segment, we have to query the *RelatedSpace* attribute to extract the information about which *IfcSpace* these *IfcRelSpaceBoundary* segments belong to.

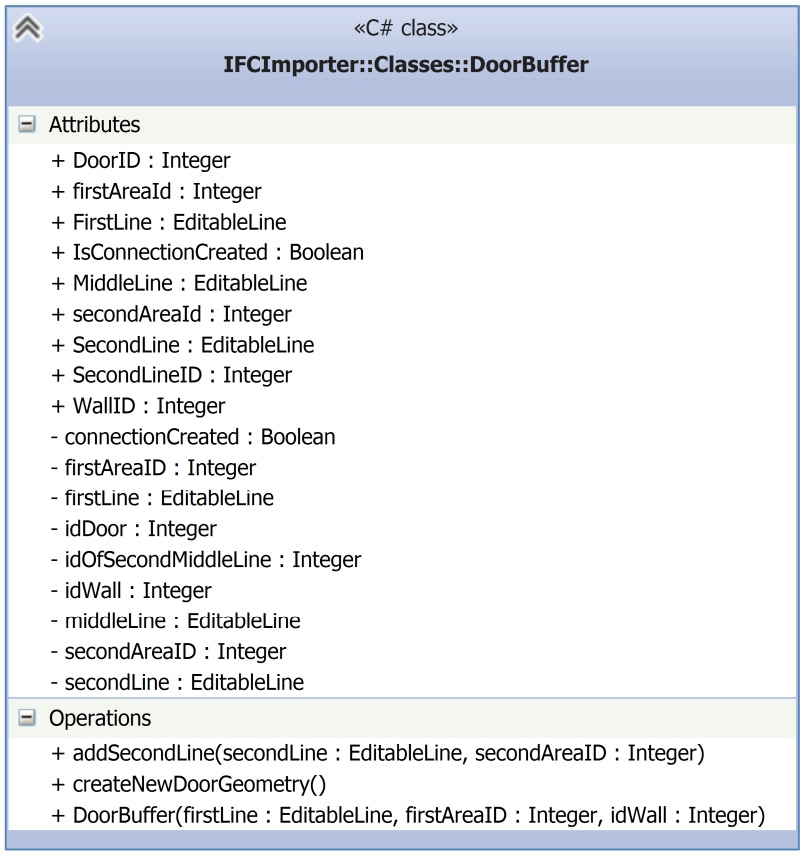


Figure 38: DoorBuffer class UML diagram

In the next phase, the *IfcRelSpaceBoundary* list is scanned for segments representing walls, i.e., the keywords we are looking for are "*IfcWallStandardCase*" and "*IfcWall*" (see Code 12). The segments are filtered and arranged according to their *RelatedSpace* attribute so that at any given moment we are only working with segments that form the boundary of the same *IfcSpace*. Consequently, two processes run in parallel at this stage. The first one deals with geometry extraction from *IfcRelSpaceBoundary* objects that reference the current *IfcSpace*. The second one relies on the acquired data to form an IPS XML *Area* object from the current *IfcSpace* object and boundary segments from the corresponding *IfcRelSpaceBoundary* objects, which are added to the current IPS XML *Area*.

```
foreach (IFCTextLine space in spaces)
{//Create a new Area
  Area area = new Area();
  area.Height = level.getParameterAsDouble(9);
  area.Id = space.ID;
  int lineId = 0;

  //query for boundaries with the same IfcSpace ID
  foreach (IFCTextLine boundary in boundaries)
  {
    if (boundary.getParameterAsInt(4).Equals(area.Id))
    {//filtering only boundaries of walls
    IFCTextLine wall = ifcFile.getIFCTextLine(boundary.getParameterAsInt(5));
    try
    {
      if (wall.Keyword.Equals("ifcwallstandardcase") ||
      wall.Keyword.Equals("ifcwall"))
      {...
```

Code 12: Query for boundary segments formed by walls

EditableLine (corresponds to the IPS XML Line from 2.3.5.2) objects are created from these segments, and each segment is also checked for the presence of doors (see Code 14). If doors are present, the geometry of the wall segment has to be adjusted with the geometry from the related *DoorBuffer* object. This is achieved by breaking up the wall segment at the location of the door and deleting the part of the wall segment between door edges, then inserting and incorporating the corresponding half of a *DoorBuffer* object geometry. The old boundary segment is discarded in favour of the newly created line segments, and these are then added as the correct boundary segments into their parent *Area*. In this way, the modified wall segments with doors, along with the unmodified wall segments that did not contain doors, form closed boundary polygons and are assigned to the right *Area* object, completing the third and final phase of forming boundaries for *Areas*.

```
LinePair lp = new LinePair();
lp.Type = ipsbimeditableLib.Enumerations.LinePair_Type.Door;
lp.First = region.Id.ToString() + "." + building.Id.ToString() + "."
  + floor.Id.ToString() + "."
  + listOfDoorBuffersAtThisWall[i].firstAreaId + "."
  + lineId;
```

Code 13: Formation of the first part of the LinePair ID

Before adding the newly constructed *Area* to the *Floor*, we need to generate the *LinePair* objects and from them the *Connection* objects. While iterating over the *DoorBuffer* list, each object is queried if it has already created a *Connection*. The in-

formation about this is stored in the *IsConnectionCreated* attribute (see Figure 38). There are two possible scenarios for a *Connection*. Either the *DoorBuffer* has two door lines or only one representing an interior or exterior door respectively. Generating the ID for a *LinePair* is required, as it is the only class of ID that consists of two IDs of two other objects, i.e., two *EditableLine* IDs. And since the middle line (see Figure 37) of the *DoorBuffer* was formed, the ID is generated from the current upper hierarchical objects (see Code 13).

```
//Iterating over all IFCRELFILLSELEMENT elements
IFCTextLine[] fillsElements = ifcFile.filterIFCTextLines("IFCRELFILLSELEMENT");
IFCTextLine[] voidElements = ifcFile.filterIFCTextLines("IFCRELVOIDSELEMENT");
foreach (IFCTextLine fillElement in fillsElements)
{//Does the IFCRELFILLSELEMENT fill the current door
  if (fillElement.getParameterAsInt(5).Equals(door.ID))
  {//From the IFCRELFILLSELEMENT Get the corresponding IFCOPENINGELEMENT
    IFCTextLine opening = ifcFile.getIFCTextLine(fillElement.getParameterAsInt(4));
    foreach (IFCTextLine voidElement in voidElements)
    {//From all IFCRELVOIDSELEMENT find the one that fits IFCOPENINGELEMENT
      if (voidElement.getParameterAsInt(5).Equals(opening.ID))
      {//Acquire the ID of the wall containing the IFCRELVOIDSELEMENT
        wallIdOfDoorSegment = voidElement.getParameterAsInt(4);
      }
    }
  }
}
```

Code 14: Determining whether the wall of the current boundary segment contains a door

As a result, the IPS model is built from *Points* up to *Areas*. What still remains, is closing the loop from the floor query and assigning the generated *Areas* to the corresponding *Floor*. Complete *Floor* objects can be added to the corresponding *Building* objects (completing the IPS building model) as well as complete *Building* objects added to the corresponding *Region* object, thereby completing the construction of the IPS model.

3.2.2.9 Testing

IFC files generated with Autodesk Revit and Graphisoft Archicad were used to test the algorithm. The reasons for choosing the two applications were already discussed in chapter 3.2.2.3 and will not be dealt with here. No other BIM applications were used because our requirements were satisfied with the two applications mentioned.

Also, the main goal of our research was not to test different IFC exporters, and using more than the two mentioned would cause distraction from the main research topic.

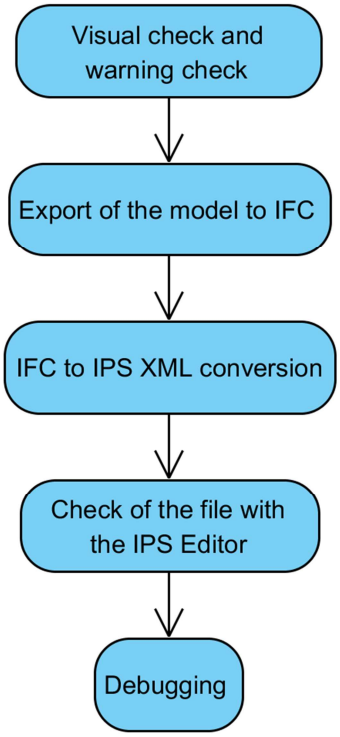


Figure 39: Testing procedure

Our testing was straightforward and can be described in the following steps (see Figure 39):

1. Visual and warning check of either the Revit or Archicad model for general mistakes,
2. Exporting the model to the IFC format with the desired options,
3. Converting the IFC file with the developed software,
4. Checking the converted file with the IPS Editor,
5. Debugging.

In the first step a visual inspection of the model from the chosen modeller was carried out. Errors and warnings were expected since the models were created by students that generally had no prior experience with the program. Continuing with the second step, the options from the export dialogues of either Revit or Archicad were designated to produce an IFC file with the right properties. For the third step a basic graphical user interface was developed by a bachelor student (Haas, 2013) to simplify the file conversion process. The generated IPS XML file was checked with the dedicated IPS Editor (Muhic, et al., 2012) for consistency. The errors were analysed and debugged in the final step.

Several of our test cases feature a controlled environment, and therefore focus on individual features of the building model, e.g., stairs, spaces that needed to be developed. This sterile environment, however, does not represent the real life issues that can arise from different approaches to developing building models. This area was covered by our students' models, because the students were not experienced users and, more often than not, relied on using tools that were not developed for the specific purpose that they were implemented for. I will focus predominantly on a test case with an Autodesk Revit generated IFC file as it has revealed the largest weaknesses that need to be addressed before a fully functional process can be established.

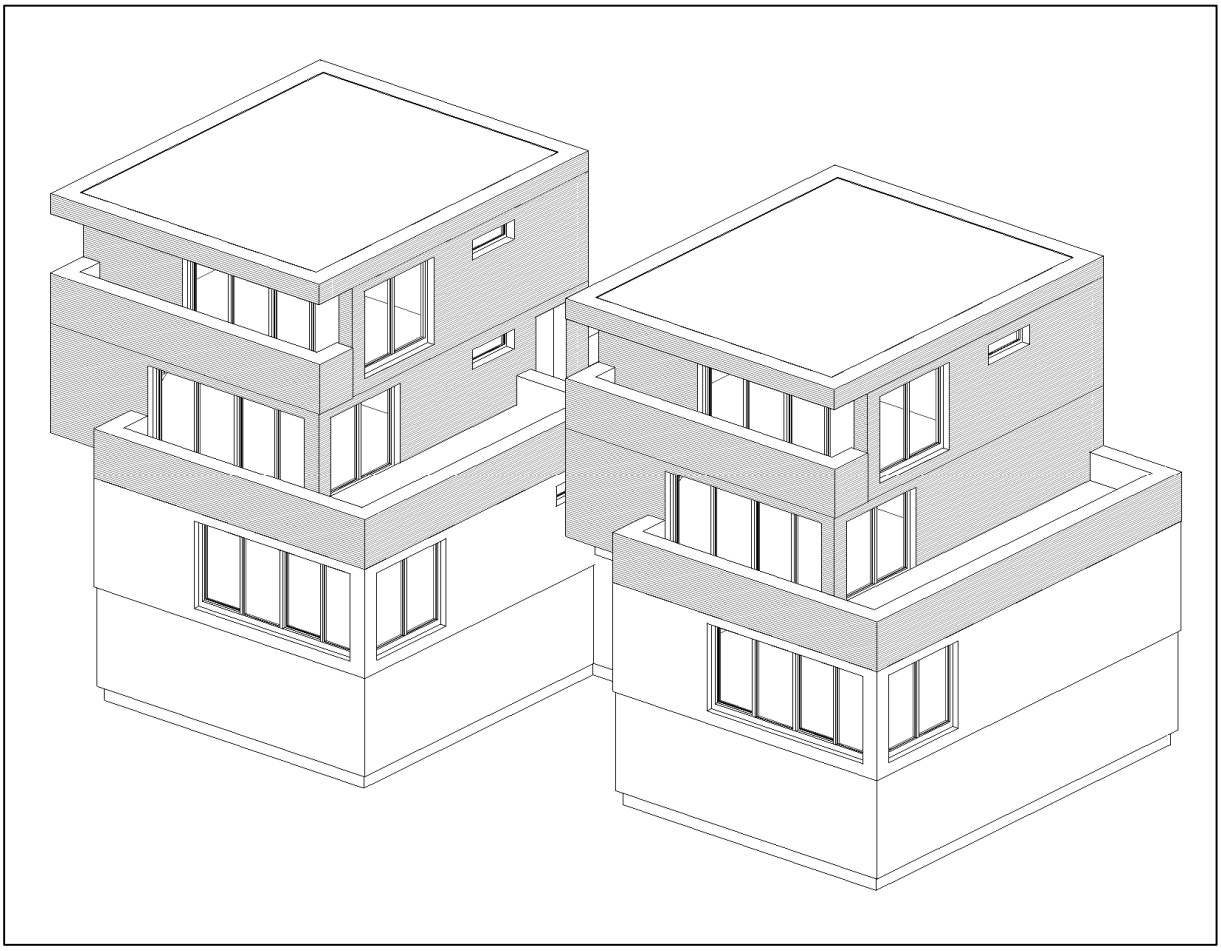


Figure 40: Revit building model from Anja Bläsche and Angelika Strmschek

The model was supplied by PhD students Angelika Strmschek and Anja Bläsche. The building model from Revit can be observed on Figure 40. Initially, exporting the IFC file and creating the IPS XML went smoothly without any issues. However, everything changed once the last check was attempted with the IPS Editor. We quickly realized that the data coming from the IFC file was flawed. One of the *IfcSpace* objects did not have a closed boundary, leading to an error when attempting to open the IPS XML file in the IPS Editor that requires closed boundaries for Area objects to fulfil the conditions of topological spaces.

This error was completely unexpected, which prolonged the search for its background. To begin with, we had to exclude the possibility that our algorithm code was responsible. A step-by-step debugging confirmed our previous tests and proved that there was no flaw and that the data was being extracted and used correctly as provided by the IFC file. Despite offering slight relief, the fact that we now had to widen the search for errors to other areas that lay outside our responsibility meant that de-

bugging and fixing the problem would take significantly more time with the possibility of no end solution being achieved. That said, two potential sources of data corruption still had to be looked at:

1. The export from Revit to IFC – desirable with a potential short term solution,
2. The Revit building model itself – only long term solutions possible with the company Autodesk showing interest in a solution.

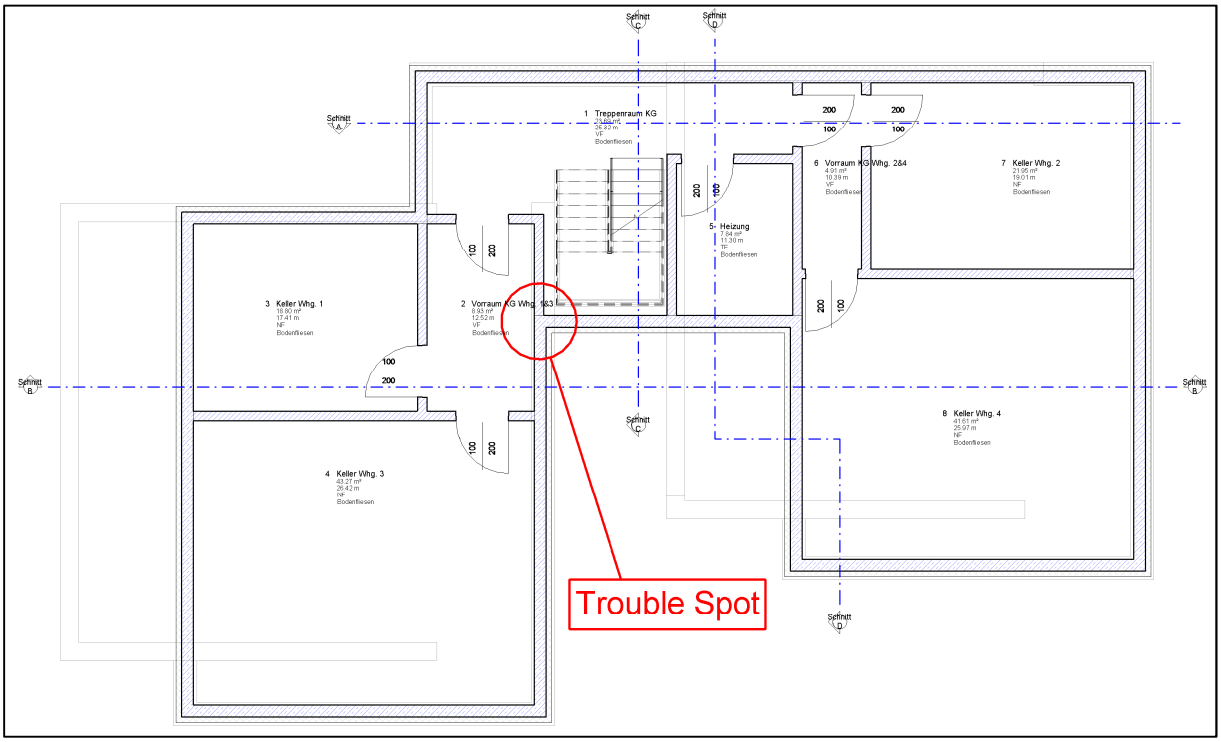


Figure 41: Floor plan with the problem wall connection marked

The second option was considered a very remote possibility and was never seriously taken into consideration with the knowledge base at the time. Regarding the export from Revit to IFC Angel Velez, a Senior Principal Engineer at Autodesk responsible for IFC export was contacted to offer some insight. With some effort we established that the IFC export tool in Autodesk Revit operates only with data from the Revit model, and therefore mirrors what the Revit building model stores. In the case of errors in the IFC file, these errors also have to be present in the Revit building model. On that basis we came to the conclusion that the Revit model itself has to be looked at. However, as Revit is a commercial application with a closed file format we did not have direct access to the building model. Our previous experience with the Revit API offered a solution without the need to involve higher-ranking Autodesk involvement.

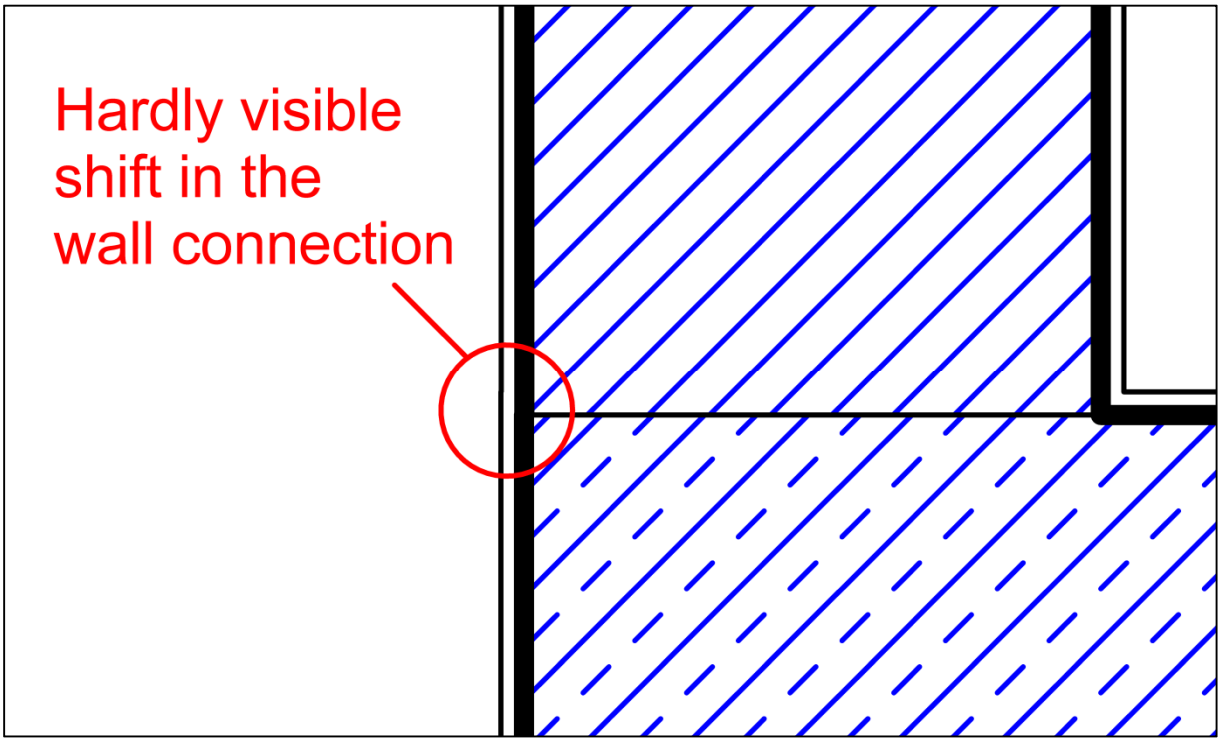


Figure 42: Detail of the wall shift problem

The approach was straightforward. The Revit API extension, which was previously developed and described in chapter 3.2.1, was used on the same building model from Figure 40. In this way, it was ensured that the data coming into the algorithm was from the Revit building model. With Autodesk Developers Network (ADN) support and Consulting Analyst Jeremy Tammik, we managed to narrow down the issue and identify its source. Apparently, the fault for the error lies both with the users and the Revit application.

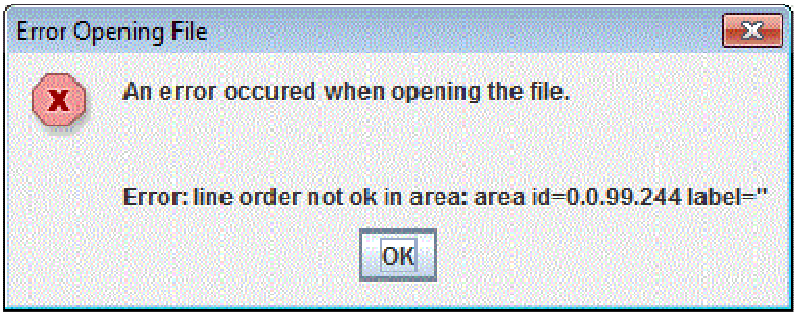


Figure 43: Error from the attempt to open the file in the IPS Editor with the Area id

As we described in earlier work (Muhič & Krammer, 2014), the program users made the mistake of not aligning the walls properly. This slight shift of walls (Figure 41 and Figure 42) was small enough for the algorithm, which creates the boundary of a Room object in Revit, to ignore, and for the visual inspection of the model from the first step in the testing phase to fail in find-

ing any issues. In Code 15 the resulting part of the IPS XML file with the problematic wall connection can be observed. The x coordinate of two supposedly continuous lines does not match, thus creating an undesired shift of the boundary segments and forming a gap. By not addressing this issue the boundary polygon of the space is not closed. Consequently, while the Revit export tool uses the exact data provided by the Revit building model, the shift of the segments and the gap in the boundary polygon extend into the exported IFC file. Through the IFC file, our algorithm copied the problem into the IPS XML file from where the inspection by the IPS Editor finally intercepted the issue and produced an error. Notice the error message from Figure 43, where the area id matches the area id from the IPS XML file in Code 15, thus making our error search easier.

```
<ips:region id="0">
  <ips:building id="0">
    <ips:floor id="99">
      <ips:area type="flat" height="-2.75" id="244">
        ...
        <ips:line id="5">
          <ips:point xs:type="ips:EditablePoint2D" x="-1.53719553629864"
          y="-8.7750465994859" id="0"/>
          <ips:point xs:type="ips:EditablePoint2D" x="-1.53719553629863"
          y="-6.92786560532477" id="1"/>
        </ips:line>
        <ips:line id="6">
          <ips:point xs:type="ips:EditablePoint2D" x="-1.53708974602005"
          y="-6.92786560532477" id="0"/>
          <ips:point xs:type="ips:EditablePoint2D" x="-1.53708974602004"
          y="-4.81086560532477" id="1"/>
          ...
        </ips:line>
      </ips:area>
    </ips:floor>
  </ips:building>
</ips:region>
```

Code 15: Part of the IPS XML file, where the wall shift can be identified by the underlined coordinates

As was stated in the beginning of the test case description, this one slight flaw of inexperienced Revit users in modelling with the application (by not using the Align command) revealed that building models, despite being advertised as perfect, do not stand up to those claims and have to be taken with reservation. As long as issues are detected, however, we have the means to correct them. The perfect solution would be for the companies themselves to fix these kinds of issues, but a more realistic and

preferable approach would be to correct the IFC file or the IPS XML file. Undoubtedly, correcting the IFC file is the better choice and it could be accomplished. Checking the IFC file on a BIM server, for example, and storing only the correct version of the building model is one option. This correct version of the building model could then be used by other parties, who would benefit from the data it provides, and thus allow access to the corrected file version.

3.3 Revit API and IFC comparison

The main differences and similarities between using the Revit API and the IFC standard have already been addressed. The main difference we observed was that taking advantage of the fact that Revit API data is drawn from a closed format offers good accessibility, although it remains only one format in a world where interoperability is a standard. Therefore, the international IFC standard offers far greater flexibility and should be the preferred option when creating a general solution to the problem of a common data source for generating Indoor Positioning building models. Another difference that needs to be mentioned is that the Revit API does not own an equivalent to the IFC *IfcRelSpaceboundary* entity. Despite its similarity to the *BonundarySegment* class, an important difference is that *BoundarySegment* objects are not produced by *Door* elements. As a consequence, no direct relationship between neighbouring *SpatialElements* can be established, as in the case of *IfcRelSpaceBoundary* objects, where two of these objects can reference the same *IfcDoor* and offer indirect information about the connectivity of two *IfcSpace* objects. Other than that, smaller differences were noticed that were not overly important for testing purposes.

On the other hand, the similarities in the data structure of building models allowed for a seamless transition between different standards. The manner in which building elements are stored and structured especially the geometrical data of building elements is almost identical. Building elements were expected to be structured at least very similar if not identical, after all building elements are predefined e.g. walls, columns, slabs etc. However, the geometrical data allows for more freedom. In the case of IFC the *IfcRelSpaceBoudnary* can be addressed very similar and identified with the *BoundarySegment* of the Revit API. The similarities of the data structure also had

the consequence that the approach to filtering that was used for the IFC format was modelled in a simpler manner after the Revit API collector/filter approach. A few of the in our case more important differences and similarities are gathered and can be observed in the following Table 29. The class from Revit API to entity from IFC comparison is merely informative as the formats do not offer equivalent basic building blocks.

Table 29: Comparison of the IFC and Revit API approach

Revit API	**IFC**
Models are available in the Revit certified file format	Models are available in most if not all currently available building model formats
Binary format	Text format
Building comprised of families	Building comprised of entities
Basic geometry curves and points	Basic geometry curves and points
No explicit connectivity data for rooms	No explicit connectivity data for rooms
SpatialElement	*IfcSpace*
BoundarySegment	*IfcRelSpaceBoundary*
Wall	*IfcWall*
Stairs	*IfcStair*/*IfcStairFlight*
Coolection of doors - OST_Door	*IfcDoor*

4 Discussion and outlook

Throughout our research, we strived to extract the required data for Indoor Positioning from existing building model formats. In the process several distinct standard building formats, as well as formats containing building data, were analysed and tested. Other approaches to solve this issue were looked into as well. From all of these sources our initial inquiry as to whether it is possible to create data required for Indoor Positioning from existing building data came out as a surprisingly convincing alternative to creating building data for Indoor Positioning manually. As expected, the data is not directly applicable, since the purpose of a complete building information model is the exact opposite of a building model for Indoor Positioning regarding its use, i.e., a complete BIM is generally compared to a specialized building model for Indoor Positioning. Therefore, specific algorithms have to be developed regarding individual use. All analysed approaches use at least some existing building data as a basis. The relation strength is relative, depending on what is expected from the Indoor Positioning building model. Mostly, the existing building data is used for the generation of geometry, and the geometry is used for graph generation; however, acquiring connectivity and adjacency data is more complex, and not all approaches choose to use the building information model as a basis for this part. Our method specifically, falls back to the semantics of a building model to create connectivity data. It was tried and tested on two building model formats, and in the end we decided that sticking to the international IFC standard is the preferable option on account of some obvious reasons, one of which is its independence from commercial applications along with the fact that it is a free and open standard.

The downside of using existing building data as a basis for Indoor Positioning building model creation is that well-structured building data in digital form is scarcely available, especially for older buildings. However, the premise for creating a method of utilizing existing building data was to avoid redundancy, which means that we never intended to completely replace the manual methods, but rather to simplify and streamline the building data gathering process. However, even in cases when building data is not available in digital form, it is still possible to acquire and record it from whatever kind of documentation exists or even directly from the standing building. A positive side effect is that with quality building information models the data can be

kept up to date with any potential changes on the building. With recent development, such scenarios are not too farfetched and are in fact possible. Modern construction processes, along with regulations regarding BIM, take full advantage of a building information model throughout its entire lifecycle, which in turn means continuous updates of the building model. Accordingly, contemporary data about the building is readily available at any given time. Looking not too far into the future we can expect building information models to be available for most, if not all, constructions.

The work described in this dissertation proves that a typical building information model can serve as a data source for an indoor positioning building model. Furthermore, it describes, in two different formats, exactly how data from existing building models can be utilized as a basis for Indoor Positioning. Although several different methods have been analysed, in addition to the solution provided here, no general solution has been discussed yet. With the data that was gathered in the processes and some additional research we can, however, list and consider the various possibilities. Focusing on IFC, being the international standard for the building product and process model, as the source of building data is obvious. The possibility of creating a Model View Definition inside IFC offers another stimulus in favour of IFC. Using a well-designed MVD for Indoor Positioning, no other conversions are required, as the format would offer a solution to our initial requirements for an Indoor Positioning format for building data. For a complete, standardized, long term and accepted solution to storing and providing building data for Indoor Positioning, a broader consensus would have to be reached in the international community. A basic outline for the required IFC entities can be obtained from chapter 3.2.2.1. The following chapters describe some possibilities and suggestions for how an IFC file format and an MVD could be fashioned to contain and provide the required data efficiently. These options can serve as a basis for future discussions and should serve as a solid foundation in this respect. Finally, the author's insight and an outline for possible future discussions on this topic are given.

4.1 IFC

Taking the approach of utilizing a bare IFC file is essentially what was described here. However, the option of creating an MVD has not yet been discussed. By creat-

ing an MVD, the file size of the building model would be drastically reduced and the process of filtering the required IFC entities shortened, because all the entities included in this MVD file would already correspond to the required IFC entities analysed and introduced in chapter 3.2.2.1. The list of entities mentioned here is by no means final, as it represents the particular needs of the IPS XML format. For an internationally accepted MVD, wider acceptance is strived for and encouraged by the author.

A first attempt at creating an MVD with the ifcDoc tool was initiated with the aforementioned IFC entities. The ifcDoc tool is the official application for developing new MVD content for IFC, and is provided by buildingSMART and downloadable at their official site. It is constantly being updated, but we used version 8.6. The most recent version of IFC, the IFC 4 Repository with MVD definitions from the buildingSMART tech website, was used as a baseline. Creating MVDs requires substantial knowledge of the IFC schema, because, in addition to defining which entities are supposed to make up the MVD, the relationships between them also have to be defined.

Currently, this MVD holds little value for the CADMS project because, as of yet, there are no automatic ways to create it. However, the algorithm for translating IFC into IPS XML could be adjusted. On the other hand, this MVD holds greater value for being an attempt at creating an Indoor Positioning MVD and opening a new research field, which should eventually result in a standardized Indoor Positioning Building Information Model format. We have established that every approach to creating building models for Indoor Positioning can, and does in fact, lean on existing building data, at least to a certain degree. Therefore, each of these methods would benefit greatly from a standardized data structure of existing building data.

One issue that is still present despite using a standardized IFC format and an MVD dedicated to Indoor Positioning is that IFC does not provide all required information. We have proven that it is possible to generate it with smart algorithms that depend on certain types of IFC files, but the information itself is not present in the IFC file directly. Therefore, the next two chapters will deal with providing alternative solutions to the need to implement algorithms every time Indoor Positioning information is required from a building model. We can approach this in two ways; either the basic IFC

schema is enhanced by the required information, or the schema is enhanced by external libraries (a case with Resource Description Framework will be presented). Essentially, the question is whether the user side (likely to be a mobile device) requires more processor power in the case of having to generate this information in real time, or more storage space if we decide to attach the information to the IFC file, thereby enhancing it.

4.2 Extending the IFC

Meddling with an existing standardized data schema is not the most desirable option. However, it would be possible. Essentially, we would have to decide on how to implement the added information. If it were possible to avoid the need for creating new IFC entities that provide the necessary environment for data storage, that would be the desired scenario. For this reason we embarked on finding suitable IFC entities by analysing the available existing IFC schema and isolating promising candidates that we later tested. One entity especially, the *IfcRelConnects* (and its sub-entities) stood out as a potential workaround to securing the connectivity information between *IfcSpace* entities. However, the results were somewhat disappointing, albeit expected and even *IfcRelConnects* proved unsuitable.

Along with the attributes listed in Table 27 from chapter 3.2.2.1, sub-entities of *IfcRelConnects* feature the references to the entity types the *IfcRelConnects* entity is connecting. We can already identify this from the name, since the second part of the sub-entity name is also the name of the entities being connected. The following sub-entities are available in the schema:

- *IfcRelConnectsElements*
- *IfcRelConnectsPathElements*
- *IfcRelConnectsPorts*
- *IfcRelConnectsPortToElement*
- *IfcRelConnectsStructuralActivity*
- *IfcRelConnectsStructuralElement*
- *IfcRelConnectsStructuralMember*
- *IfcRelConnectsWithEccentricity*

- *IfcRelConnectsWithRealizingElements*

Having listed the available *IfcRelConnects* sub-entities, the entities that we need to connect are *SpatialElements*. The list proves that *IfcRelConnects* can be used on various different IFC entities, which allows us to consider the alternative of enhancing the IFC schema. All that would be needed is for us to take *IfcRelConnects* as the basis for a potential new *IfcRelConnectsSpatialElements* entity. We can approach the issue either by sticking to the original principle of changing as little of the IFC file as possible, or allowing ourselves more freedom. The resulting proposals are in Table 30 and Table 31.

Both proposals are similar in form and practically identical to the existing *IfcRelConnectsElements* entity, because the purpose is essentially the same, with the distinction of the type of elements being connected and, in the first proposal, two *ConnectionGeometry* attributes. The reasoning behind having two *ConnectionGeometry* attributes is the fact that both the relating and the related *SpatialElement* already have their own connection geometry defined, and since we would like to use as many of the existing elements as possible we will reference the existing ones.

Table 30: IfcRelConnectsSpatialElements proposal (the least changes possible to the IFC file)

Attribute	Type
GlobalId	IfcGloballyUniqueId
OwnerHistory	IfcOwnerHistory
Name	IfcLabel
Description	IfcText
ConnectionGeometry	IfcConnectionGeometry
ConnectionGeometry	IfcConnectionGeometry
RelatingSpatialElement	IfcSpatialElement
RelatedSpatialElement	IfcSpatialElement

Table 31: IfcRelConnectsSpatialElements proposal (with corrected connection geometry)

Attribute	Type
GlobalId	IfcGloballyUniqueId
OwnerHistory	IfcOwnerHistory
Name	IfcLabel
Description	IfcText
ConnectionGeometry	IfcConnectionGeometry
RelatingSpatialElement	IfcSpatialElement
RelatedSpatialElement	IfcSpatialElement

The first four attributes in both cases are naturally the same as when they are laid out by the parent entity *IfcRelConnects*. The remaining four in the first case are used to describe the nature of the connection: it consists of two spatial elements that are connected and the connection geometry between them. A parallel to the *DoorBuffer* class, described in 3.2.2.8, is clearly identifiable, and thus can be considered the inspiration behind it. The *ConnectionGeometry* attribute could contain the same information about geometry as the *DoorBuffer*. However, since the geometry of the connection is already defined by other elements, we could reference those existing geometry elements. In terms of an IFC file, that would mean that the new entity and its *IfcRelConnectsSpatialElements* could be referencing the same *IfcConnectionSurfaceGeometry* as the *IfcRelSpaceBoundary* elements that define the two door boundary segments of the two spaces that are accessible through a particular *IfcDoor*. Thus, the requirements for change are kept to a minimum, since all entities are already present. The slight drawback is that the specific corrected geometry that a *DoorBuffer* class object provides is not available and, consequently, the two topological spaces do not connect geometrically.

In the second proposal we are dealing with only one single *ConnectionGeometry* attribute. Here the *IfcRelConnectsSpatialElements* element would be defined by a new polyline in the middle of the two existing *IfcRelSpaceBoundary* elements, exactly the way the *DoorBuffer* is defined.

A third option is possible, where the new *IfcRelConnectsSpatialElements* would look the way they are depicted in Table 31, and only one *IfcConnectionSurfaceGeometry* would be defined in the middle of the spaces. However, this alternative seems more distant, since the *IfcRelSpaceBoundary* geometry definition of *IfcSpace* elements would have to be redesigned so that it would match the description from chapter 3.2.2.8.

Creating this additional information for the IFC file still requires some input in the form of an analysis of the IFC file with the developed algorithm for creating the connectivity information. Just enhancing the schema does not avoid the need for actually creating this information from geometric data and the general knowledge of the accessibility of building interior space.

4.3 RDF as an extension to IFC

While relying on the basic IFC schema and supplementing additional information with external algorithms or proposing an extension to the schema with the desired entities present the obvious possibilities, alternatives do exist. External stashes of data can provide additional flexibility as well as additional functionality to the IFC schema. As was proven by Beetz, et al. (2015), for the purpose of quay walls in harbours external libraries in the form of Resource Description Framework (RDF) vocabularies can enhance the functionality of the IFC schema.

The approach is fairly simple. As an alternative to extending the IFC schema directly, which is a tedious process due to the effort needed for implementing such changes and because serving niche markets does not offer a good enough return of investment to software vendors (Beetz, et al., 2015), the RDF vocabulary approach builds upon extending the buildingSMART data dictionary (bSDD). RDF enables the creation, reference and extension of data sets. Compared to extending the bSDD directly without an external vocabulary, the RDF approach requires no additional implementation effort on the client software side by providing interfaces to engineering end-users (Beetz, et al., 2015). In the quay wall project mentioned above, the bSDD vocabulary has been transformed into a configurable RDF dataset.

Looking more closely at RDF vocabularies we can establish that it is a framework for presenting information on the web. It is a graph based data model (RDF graph) in which triples (RDF triple) represent the core structure. A triple consists of a subject, a predicate and an object (Cyganiak, et al., 2014). The graph structure of RDF vocabularies enables the application of the SPARQL query language to manipulate and query the RDF graph content (W3C SPARQL Working Group, 2013).

While reflecting upon this approach it is clear that a similar RDF vocabulary could be established for Indoor Positioning. By offering greater flexibility, compared to the other alternatives mentioned, it represents a keystone for further research into the implementation of indoor positioning building model technology.

4.4 Summary

Looking at the three options of creating only an MVD, extending the IFC file with additional entities or creating an additional external library, we are faced with a dilemma. As it currently stands, the bare minimum of added information is only one additional entity to the IFC schema, which seems twofold. On the one hand, adding only one entity is not a big interference, on the other we have already proven that the information can be extracted from the existing schema; therefore an additional entity is not required. However, the future is not yet certain and other requirements might lead to the need for more entities. We also have to consider the following: given the fact that modern BIM tools do not focus on the specific needs of indoor positioning and navigation, these tools do not feature functions that could add this kind of information, and therefore after the model is created the required additional indoor positioning information would have to be attached either manually or with some additional software application.

After careful consideration, by assessing all of the facts, the course of action including starting to expand the IFC schema by adding entities, albeit appearing quite elegant, seems to be the worst choice. Without further research there is no assurance that one entity providing the connectivity information of spaces might be sufficient for a final solution. In our current state, therefore, the other two alternatives are in precedence. The first one, while relying only on the existing IFC schema, still requires some additional input to provide a possible solution. This can be achieved either by

building the functionality into the Indoor Positioning software or by using the IFC IPS MVD as a data source for a new file format or database entry.

The second option can be considered an upgrade to the first one. Taking the IFC Indoor Positioning MVD as a basis and running our proposed algorithm on it could generate an entry for the RDF vocabulary. By accessing both the IFC IPS MVD and the RDF vocabulary, no additional computation would be required.

There is no clear winner between these two alternatives. While the IFC file already holds all of the data needed, and it simply has to be extracted and processed, it, in fact, does what a file format should do, which is provide data. However, by providing additional functionality and modularity for potential additions required in the future, the external library approach offers added value in a relatively simple manner. In any case though, the initial step of creating and certifying an Indoor Positioning MVD is required, and should be dubbed a priority before developing alternative model formats, which present mostly redundant information to IFC.

4.5 Outlook and future research

Utilizing existing building data, and especially existing semantic building models for Indoor Positioning, opens a range of possible improvements to the construction process and, not to mention, the use phase of the building. A modern approach to the construction process requires a constantly updated building model and, consequently, an updated Indoor Positioning model. With these additions, documenting the construction phase can be revolutionized. Real time positioning of workers, machines and material will make every-day planning a lot more efficient and exact, and as a result, it will enable a more efficient realization of general construction planning. Later on in the use phase, the maintenance, management and every day usage of buildings can be improved in a similar way. Buildings can be monitored using updated building models and real time indoor positioning processes, and decisions can be made accordingly. Moreover, through the synergy that is provided by indoor positioning, the building model can be kept updated and flexible to the challenges that come with everyday building operation. The goal that should be strived for here is implementing indoor positioning as an integral part of building information modelling as the

modern approach to the planning and construction phase of buildings on the basis of a standardized format for indoor positioning as a subset of the building model.

Constructing buildings on the premise that building information modelling is the standardized approach at its core opens up new possibilities. With such efficient and up-to-date data structures available, we can start to think about the automation of individual processes in a much broader sense. The classic wet processes in the construction phase, performed mainly by workers on site, are facing evolution. As in the car industry, where 3D assembly models of car parts enabled the production of them without technical drawings, BIM can provide similar benefits for the construction industry. Technologies based on the building information model are completing the puzzle until a complete and working process is finally developed. One such puzzle piece is the integration of Indoor Positioning.

Another aspect that has not been considered is that with the increasing quality of building information, the quality of routing algorithms can also increase. Much the same as creating a building model according to reality can benefit from good positioning and vice versa, with ever better building information models algorithms for finding the shortest path can become increasingly complex. Therefore, relying on the best possible building data enables better and more complex research regarding shortest path algorithms.

References

Autodesk, Inc, 2014. *My First Plug-in Training.* [Online]
Available at:
http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=16849339
[Accessed 21 8 2014].

Becker, T., Nagel, C. & Kolbe, T. H., 2009. A multilayered space-event model for navigation in indoor spaces. In: *3D Geo-Information Sciences.* Berlin: Springer, pp. 61-77.

Beetz, J. et al., 2015. *Interoperable data models for infrastructural artefacts - a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors.* Vienna, CRC Press/Balkema.

Bernoulli, T., 2015. *Flexibilität durch Modularität: Ein Framework für sensorbasierte Positionierungssysteme.* Graz: Graz University of Technology.

Bernoulli, T., Glanzer, G., Walder, U. & Wießflecker, T., n.d. *Infrastructureless Indoor Positioning System for First Responders.* Seattle, s.n.

Bernoulli, T. et al., 2008. *A Building Information Model for a Context-Adaptive Disaster Management System.* Plymouth, s.n.

Bernoulli, T. et al., 2011. *Improvement of Inertial Sensor Based Indoor Navigation by Video Content Analysis.* Guimaraes, IEEE Xplore.

Bernoulli, T., Wießflecker, T., Glanzer, G. & Walder, U., 2010. *Infrastructureless Indoor Positioning System For First Responders.* Seattle, s.n.

Björk, B.-C. & Penttilä, H., 1989. A scenario for the development and implementation of a building product model standard.. *Advances in Engineering Software,* 11(4), pp. 176-187.

Blum, H., 1967. A Transformation for Extracting New Descriptors of Shape. *Models for the perception of speech and visual form,* pp. 362-380.

buildingSMART International Ltd., n.d. *Participants of the official buildingSMART IFC2x3 Coordination View V2.0 certification process.* [Online]
Available at: http://www.buildingsmart-tech.org/certification/ifc-certification-2.0/ifc2x3-cv-v2.0-certification/participants
[Accessed 20 8 2014].

buildingSMART, 2008. *Industry Foundation Classes (IFC) data model.* [Online]
Available at: http://www.buildingsmart.org/standards/ifc/model-industry-foundation-classes-ifc
[Accessed 13 2 2014].

buildingSMART, 2010. *certification 2.0 summary.* [Online]
Available at: http://www.buildingsmart-tech.org/certification/ifc-certification-2.0/certification-2.0-summary
[Accessed 3 March 2014].

BuildingSMART, 2013. *IfcSpace.* [Online]
Available at: http://www.bhttp://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcproductextension/lexical/ifcspace.htmuildingsmart-tech.org/ifc/IFC2x3/TC1/html/ifcproductextension/lexical/ifcspace.htm
[Accessed 15 October 2014].

buildingSmart, 2013. *IfcWall.* [Online]
Available at: http://www.buildingsmart-tech.org/ifc/IFC4/final/html/schema/ifcsharedbldgelements/lexical/ifcwall.htm
[Accessed 16 October 2014].

buildingSMART, 2014. *Currently certified software products.* [Online]
Available at: http://www.buildingsmart.org/certification/currently-certified-software-products
[Accessed 2014].

buildingSMART, 2015. [Online]
Available at: http://www.buildingsmart-tech.org/specifications/specification-tools/ifcdoc-tool/ifcdoc-baselines
[Accessed 2 February 2015].

buildingSMART, n.d. *buildingSMART a council of the Institute for BIM in Canada.* [Online]
Available at: http://www.buildingsmartcanada.ca/resources/resourcesterms-and-definitions/
[Accessed 24 February 2014].

buildingSMART, n.d. *buildingSMART-tech.* [Online]
Available at: http://www.buildingsmart-tech.org/specifications/ifc-overview
[Accessed 13 February 2014].

buildingSMART, n.d. *Industry Foundation Classes Release 4 (IFC4) Release Candidate 4 (RC4).* [Online]
Available at: http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/
[Accessed 2 May 2015].

Butz, A., Baus, J., Krüger, A. & Lohse, M., 2001. *A hybrid indoor navigation system.* Santa Fe, ACM, pp. 25-32.

Cyganiak, R., Wood, D. & Lanthaler, M., 2014. *RDF 1.1 Concepts and Abstract Syntax.* [Online]
Available at: http://www.w3.org/TR/rdf11-concepts/
[Accessed 8 May 2015].

Daum, S., Borrman, A., Langenhan, C. & Petzold, F., 2015. *Automated generation of building fingerprints using a spatio-semantic query language for building information models.* s.l., CRC Press/Balkema.

Di Giampaolo, E., 2010. *A passive-RFID based indoor navigation system for visually impaired people.* ISABEL, s.n.

Eastman, C. M., Bond, A. H. & Chase, S. C., 1991. A formal approach for product model information. *Research in Engineering Design,* 2(2), pp. 65-80.

Eastman, C., Teicholz, P., Sacks, R. & Liston, K., 2008. *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors.* 1 ed. Hoboken: John Wiley & Sons.

Elvin, G., 2007. *Integrated Practice in Architecture.* Hoboken: John Wiley & Sons.

gbXML, 2014. *Current Schema.* [Online]
Available at: http://www.gbxml.org/currentschema.php
[Accessed April 2015].

gbXML, n.d. *About gbXML.* [Online]
Available at: http://www.gbxml.org/aboutgbxml.php
[Accessed 30 May 2014].

Giel, B. & Issa, R. R., 2011. BIM return on rnvestment: A case study. *Journal of Building Information Modeling,* pp. 25-27.

Graphisoft, 2013. *IFC Reference Guide for ArchiCAD 17.* [Online]
Available at:
http://www.graphisoft.com/ftp/techsupport/documentation/IFC/IFC%20Reference%20Guide%20for%20ArchiCAD%2017.pdf
[Accessed 8 June 2015].

Graphisoft, 2013. *IFC Translators.* [Online]
Available at: http://helpcenter.graphisoft.com/guides/archicad-17-int-reference-guide/interoperability/file-handling-and-exchange/working-with-ifc/ifc-translators/
[Accessed 27 March 2014].

Gröger, G., Kolbe, T. H. & Nagel, K., 2012. *CityGML.* [Online]
Available at: http://www.opengeospatial.org/standards/citygml
[Accessed 11 March 2014].

Groome, C., 2010. *certification 2.0 summary.* [Online]
Available at: http://www.buildingsmart-tech.org/certification/documents/bSI_IFC_Certification_2%200_Specification_Final.pdf
[Accessed 3 March 2014].

Haas, M., 2013. *Entwicklung einer Softwareanwendung zur Konvertierung von IFC Dateien in das CADMS-XML Format.* Graz: s.n.

Hartmann, D., 2010. Adaptive pedestrian dynamics based on geodesics. *New Journal of Physics,* 12(4), p. 043032.

Hub, A., Diepstraten, J. & Ertl, T., 2004. *Design and Development of an Indoor Navigation and Object Identification System for the Blind.* New York, ACM, pp. 147-152.

Hub, A., Hartler, T. & Ertl, T., 2006. *Interactive Tracking of Movable Objects for the Blind on the Basis of Environment Models and Perception-Oriented Object Recognition Methods.* New York, ACM, pp. 111-118.

IFC TOOLS PROJECT, 2013. *IFC TOOLS PROJECT Information.* [Online]
Available at: http://www.ifctoolsproject.com/info.html
[Accessed 29 3 2015].

Isikdag, U. & Zlatanova, S., 2009. Towards Defining a Framework for Automatic Generation of Buildings in CityGML Using Building Information Models. In: J. Lee & S. Zlatanova, eds. *3D Geo-Information Sciences.* Berlin: Springer Berlin Heidelberg, pp. 79-97.

ISO, 2004. *ISO 10303-11:2004(en).* [Online]
Available at: https://www.iso.org/obp/ui/#iso:std:iso:10303:-11:ed-2:v1:en
[Accessed 26 3 2015].

ISO, 2013. *ISO 16739:2013.* [Online]
Available at: http://www.iso.org/iso/catalogue_detail.htm?csnumber=51622
[Accessed 23 3 2015].

Kelly, D., 2015. *Investigating the Relationships of Project Performance Measures with the Use of Building Information Modeling (BIM) and Integrated Project Delivery (IPD).* Ypsilanti: Master's Theses and Doctoral Dissertations.

Khemlani, L., 2004. *AECbytes.* [Online]
Available at: http://www.aecbytes.com/feature/2004/IFCmodel.html
[Accessed 10 March 2014].

Khemlani, L., 2012. *Around the World with BIM.* [Online]
Available at: http://www.aecbytes.com/feature/2012/Global-BIM.html
[Accessed July 2015].

Kiers, M., Bischof, W., Krajnc, E. & Dornhofer, M., 2011. *Evaluation and improvements of an rfid based indoor navigation system for visually impaired and blind people.* Guimarães, s.n.

Kneidl, A., Borrman, A. & Hartmann, D., 2012. Generation and use of sparse navigation graphs for microscopic pedestrian simulation models. *Advanced Engineering Informatics,* 26(4), pp. 669-680.

Kolbe, T. H., 2012. *Exchange and Storage of Virtual 3D City Models.* [Online]
Available at: http://www.citygml.org/index.php?id=1523
[Accessed 11 March 2014].

Krammer, M., Bernoulli, T., Muhič, S. & Walder, U., 2012. *A Smartphone Application for an Innovative User Supporting Location Based Shopping Experience.* Sydney, s.n.

Laakso, M. & Kiviniemi, A., 2013. The IFC standard: A review of History, development, and standardization, Information Technology. *ITcon,* pp. 134-161.

Lee, D.-T., 1982. Medial axis transformation of a planar shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* Issue 4, pp. 363-369.

Lee, J., 2004. A spatial access-oriented implementation of a 3-D GIS topological data model for urban entities. *GeoInformatica,* pp. 237-264.

Lee, J.-K.et al., 2010. Computing walking distances within buildings using the universal circulation network. *Environment and planning: Planning and design,* pp. 628-645.

Lee, J. & Zlatanova, S., 2009. *3D Geo-Information Sciences.* Berlin: Springer.

Liebich, T., 2009. *IFC Implementation Guide v2.0.* [Online]
Available at: http://www.buildingsmart-tech.org/downloads/accompanying-documents/guidelines/IFC2x%20Model%20Implementation%20Guide%20V2-

0b.pdf/view
[Accessed 8 June 2015].

Liebich, T., 2010. *Unveiling IFC2x4 - The next generation of openBIM.* Cairo, s.n.

Liebich, T., 2013. *IFC4 – the new buildingSMART Standard.* [Online]
Available at: http://www.buildingsmart-tech.org/specifications/ifc-releases/ifc4-release/buildingSMART_IFC4_Whatisnew.pdf
[Accessed 26 February 2014].

Liebich, T., 2014. *IFC Overview Presentation.* [Online]
Available at:
http://www.bre.co.uk/filelibrary/events/BRE%20Events/BIM%20Conference%20Season/Delivery%20of%20IFC/2-Thomas-Liebich.pdf
[Accessed 22 August 2015].

Li, K.-J., 2008. *Indoor Space: A New Notion of Space.* Shanghai, Springer Berlin Heidelberg, pp. 1-3.

Li, N., Becerik-Gerber, B., Krishnamachari, B. & Soibelman, L., 2014. A BIM centered indoor localization algorithm to support building fire emergency response operations. *Automation in Construction,* pp. 78-89.

Mäs, Stephan; Reinhardt, Wolfgang; Wang, Fei;, 2006. Conception of a 3D Geodata Web Service for the Support of Indoor Navigation with GNSS. In: *Innovations in 3D Geo Information Systems.* Berlin: Springer Berlin Heidelberg, pp. 307-316.

Matthews, O. & Howell, G. A., 2005. Integrated project delivery an example of relational contracting. *Lean Construction Journal,* 2(1), pp. 46-61.

Miesenberger, Klaus; Klaus, Joachim; Zagler, Wolfgang; Karshmer, Arthur;, 2008. *Computers Helping People with Special Needs.* Berlin: Springer Berlin Heidelberg.

Muhic, S. & Krammer, M., 2013. *Verwendung von bestehenden Gebäudemodellstandards für die Anwendung in Indoor Navigation.* Munich, Shaker Verlag.

Muhic, S., Krammer, M., Bernoulli, T. & Walder, U., 2012. *Defining a Building Information Model for Emergency Management.* Reykjavik, CRC Press/Balkema.

Muhič, S. & Krammer, M., 2014. *Utilizing IFC for Indoor Positioning.* Vienna, CRC Press, p. 681–686.

National BIM Standard - United States™, 2014. *National BIM Standard – United States™ Version 2.* [Online]
Available at: http://www.nationalbimstandard.org/faq.php

ONPOI, 2014. *ONPOI.* [Online]
Available at: http://www.onpoi.at/
[Accessed 16 12 2014].

Open Geospatial Consortium, 2012. *Online Resources.* [Online]
Available at: http://www.citygml.org/index.php?id=1522
[Accessed 3 August 2015].

Pérez-Lombard, L., Ortiz, J. & Pout, C., 2008. A review on buildings energy consumption information. *Energy and buildings,* 40(3), pp. 394-398.

Reynolds, M. S., 2003. *Low Frequency Indoor Radiolocation,* s.l.: s.n.

Rueppel, U. & Stuebbe, K. M., 2008. BIM-based indoor-emergency-navigation-system for complex buildings. *Tsinghua Science and Technology,* Volume 13, pp. 362-367.

Schütz, R., Bernoulli, T., Wießflecker, T. & Walder, U., 2008. *A CONTEXT-ADAPTIVE BUILDING INFORMATION MODEL FOR REAL-TIME STRUCTURAL ANALYSIS IN A DISASTER MANAGEMENT SYSTEM.* [Online]
Available at: http://itc.scix.net/cgi-bin/works/Show?w78-2008-4-01
[Accessed 17 June 2014].

Semenov, V. A., Kazakov, K. A. & Zolotov, V. A., 2012. *Global path planning in 4D environments using topological mapping.* London, CRC Press Taylor & Francis Group.

Shayeganfar, Ferial; Anjomshoaa, Amin; Tjoa, A Min;, 2008. *A Smart Indoor Navigation Solution Based on Building Information Model and Google Android.* Linz, Springer Heidelberg, pp. 1050-1056.

Simonian, L. & Korman, T., 2010. *Legal Considerations Associated with Building Information Modeling.* Paris, s.n.

Souvaine, D., Horn, M. & Weber, J., 2004. *Voronoi Diagrams.* [Online]
Available at: http://www.cs.tufts.edu/comp/163/notes05/voronoi_handout.pdf
[Accessed 9 12 2014].

Taneja, S. et al., 2011. *Transforming an IFC-based Building Layout Information into a Geometric Topology Network for Indoor Navigation Assistance.* Miami, American Society of Civil Engineers, pp. 315-322.

W3C SPARQL Working Group, 2013. *SPARQL 1.1 Overview.* [Online]
Available at: http://www.w3.org/TR/sparql11-overview/
[Accessed 8 May 2015].

Walder, U., 2006. *Integration of Computer Aided Facility Management Data and Real-time Information in Disaster Management.* Valencia, Taylor & Francis/Balkema.

Walder, U., 2012. *State-of the-art of pedestrian navigation with foot-mounted IMU,* Sydney: s.n.

Walder, U. & Bernoulli, T., 2010. *Context-Adaptive Algorithms to Improve Indoor Positioning with Inertial Sensors.* Zurich, IEEE.

Wallgrün, J. O., 2005. Autonomous construction of hierarchical voronoi-based route graph representations. In: *Spatial Cognition IV. Reasoning, Action, Interaction.* s.l.:Springer, pp. 413-433.

Waterhouse, R. et al., 2015. *NBS National BIM Report 2015,* s.l.: s.n.

ways4all, 2014. *ways4all.* [Online]
Available at: http://www.ways4all.at/index.php?lang=en
[Accessed 16 12 2014].

Wießflecker, T., 2009. *Zur Integration von Sensor- und Gebäudedaten für das Katastrophenmanagement Ein neuer Ansatz zur Innenraumpositionierung.* Graz: s.n.

Wix, J., 2010. *buildingSMART.* [Online]
Available at: http://iug.buildingsmart.org/idms/development/IDMC_004_1_2.pdf
[Accessed 21 February 2014].

xsens, n.d. *MTw Development Kit.* [Online]
Available at: https://www.xsens.com/products/mtw-development-kit/
[Accessed 18 May 2015].

Yao, C. & Rokne, J., 1991. A straightforward algorithm for computing the medial axis of a simple polygon. *International Journal of Computer Mathematics,* pp. 51-60.

Table of figures

Table of tables

Table of code

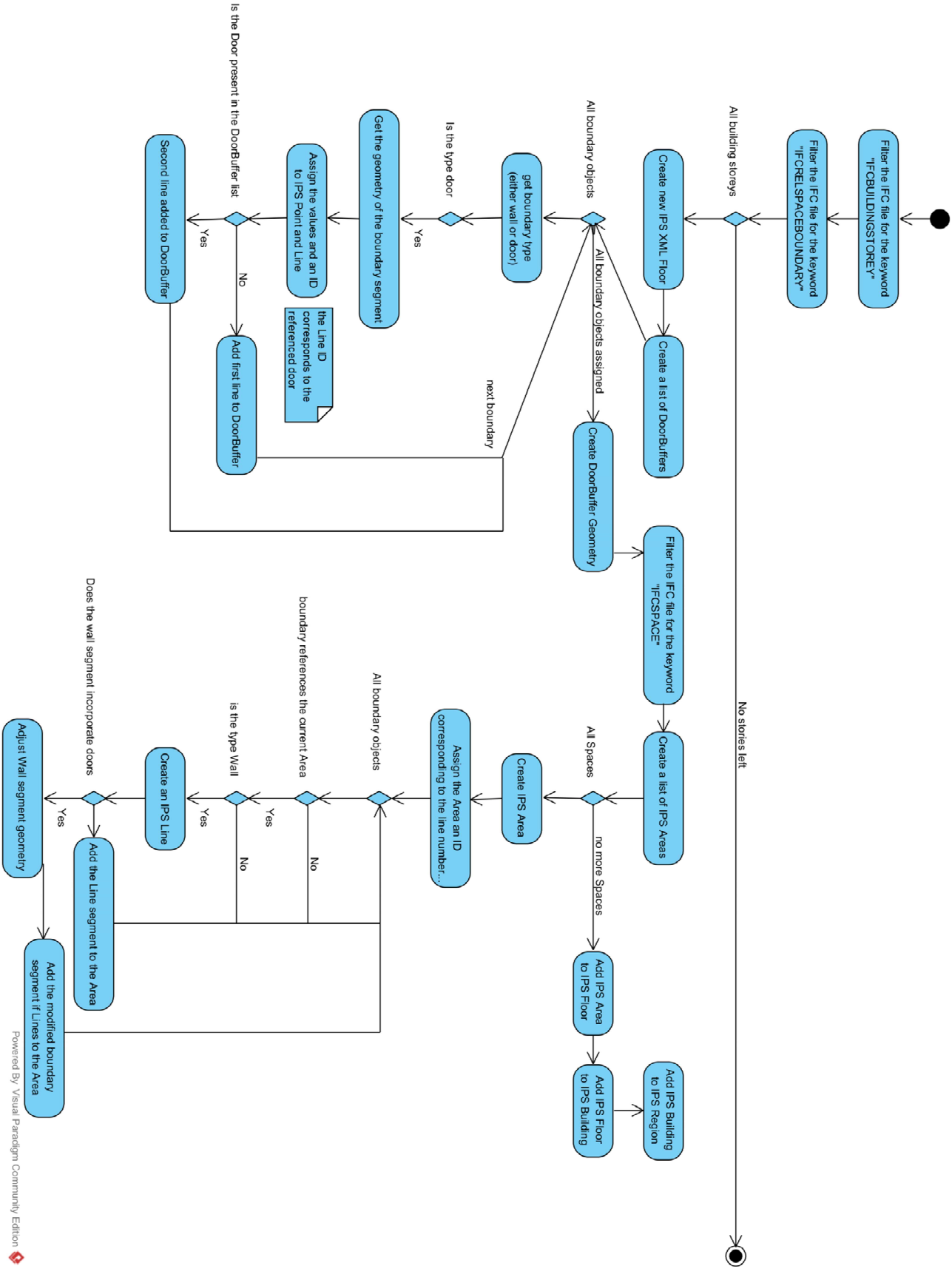
List of abbreviations

AEC	Architecture Engineering Construction
AIONAV	Autonomous Indoor and Outdoor Navigation
API	Application programming interface
ASCII	American Standard Code for Information Interchange
BIM	Building Information Modelling/Model
BRep	Boundary representation
bSDD	buildingSMART data dictionary
CADMS	Computer Aided Disaster Management System
CAFM	Computer Aided Facility Management
CityGML	City Geographic Markup Language
gbXML	Green Building Extensible Markup Language
IAI	International Alliance for Interoperability
IDE	Integrated Development Environment
IFC	Industry Foundation Classes
IMU	Inertial Measurement Unit
IPD	Integrated Project Delivery
IPS	Indoor Positioning System
MEP	Mechanical, Electrical and Plumbing
MVD	Model View Definition

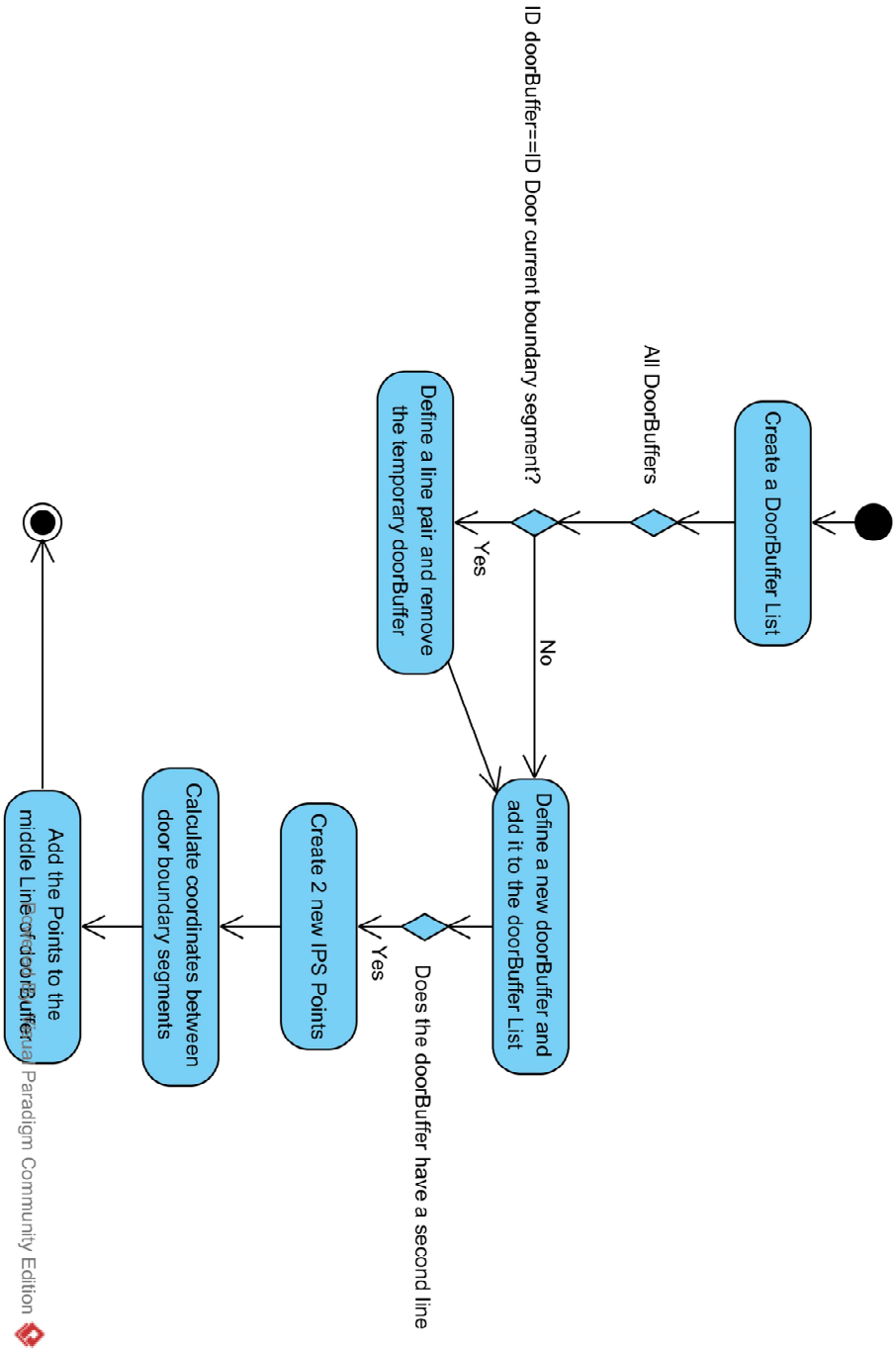
RDF	Resource Description Framework
SPARQL	Protocol and RDF Query Language
STEP	Standard for the Exchange of Product Model Data
UML	Unified Modelling Language
XML	Extensible Markup Language

Appendix

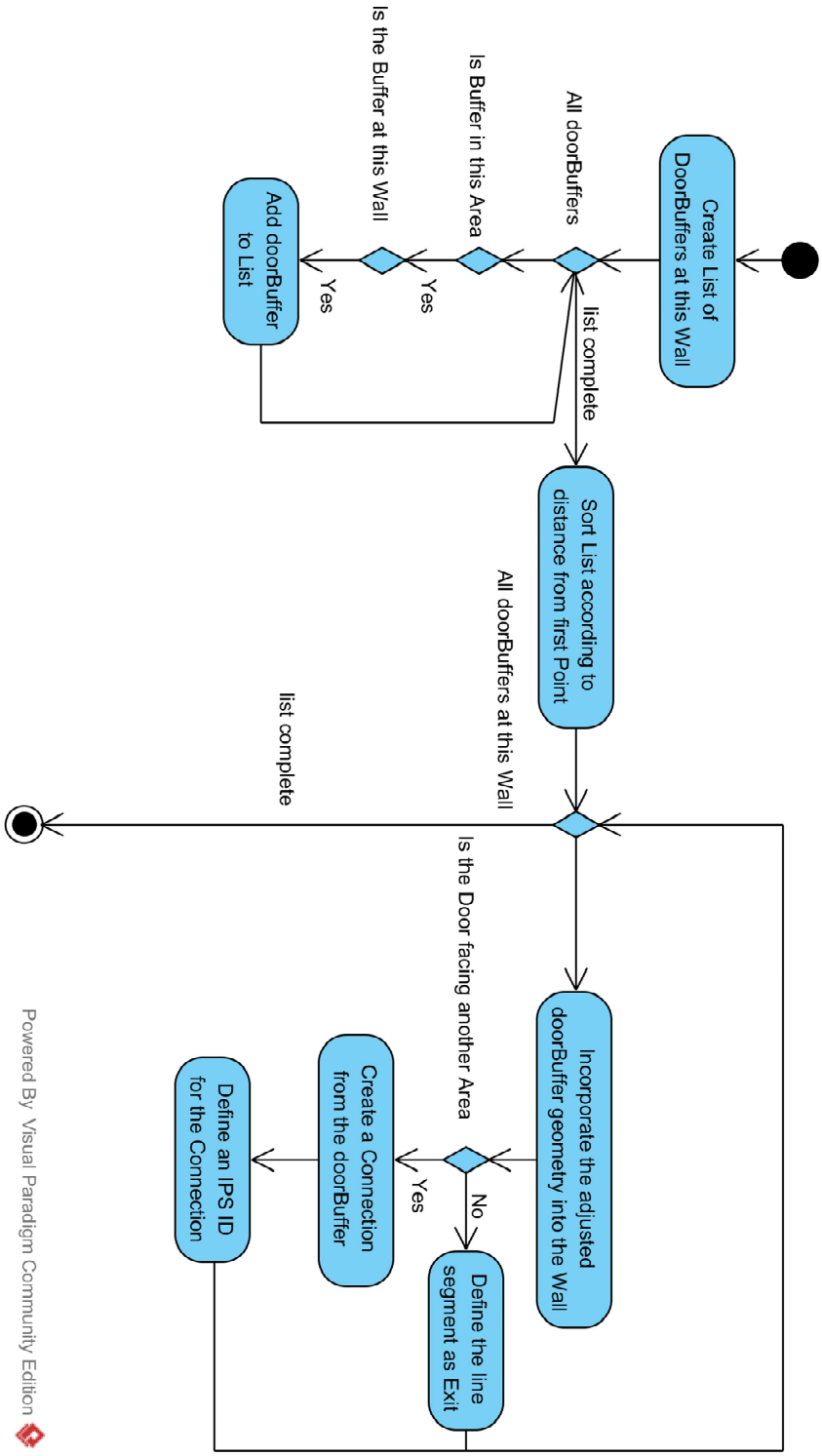
Appendix 1: Activity diagram of the IFC to IPS XML conversion



Appendix 2: Activity diagram for creating a DoorBuffer list



Appendix 3: Activity diagram for adjusting the geometry of wall segments when inserting DoorBuffer objects



Appendix 4: UML class diagram of the IFC exporter application