Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

Geometrical and Topological Linking of Railway Systems

Ya-Chun Bollig

Masterthesis
für den Master of Science Studiengang Bauingenieurwesen

Autor:	Ya-Chun Bollig
Matrikelnummer:	
Betreuer:	Prof. Dr.-Ing. André Borrmann
	Sebastian Esser, M.Sc.
Ausgabedatum:	01. Dezember 2019
Abgabedatum:	31. März 2020

Abstract

Building Information Modelling (BIM) has already established itself as a best-practice approach for efficient productivity in the field of building design and operation, yet is still in its early days in the field of infrastructure. Recent efforts to promote the integration of BIM into large-scale infrastructure projects also target the construction and operation of railway networks. The existing methodologies traditionally used for railways are however neither horizontally, between sub-contractors, nor vertically, along the life-cycle, fully integrated, leading to possible loss of information.

There exist data exchange schemata to reduce the problem of storage and information flow, but they are not yet optimized towards railways, nor is there a common standard that construction industries have agreed upon. Industry Foundation Classes (IFC) recently added the concept of a linear reference system for infrastructure assets, yet is still only focused on the geometry needed during the construction phase. Railway Markup Language (railML) is an accepted data exchange schema tailored towards providing topology for railway operations, yet lacks full support of geometry data. A prototype tool linking these two schemata is developed in this thesis to ensure consistent data in both geometrical and topological representations.

The geometrical entity and element sets of IFC and railML show sufficient overlap to support a consistent data exchange with the exception of transition curves. However, the exchange of topological data is, for example, limited regarding the network stored in railML, as IFC does not yet have entities that describe the relationships among alignments. The developed prototype allows to reconstruct a simple track geometry from scratch for a given railML file with minimal user input but loses the relationship information as well as object aggregation after the transfer to IFC. Conversely, the IFC geometry can be converted reasonably well to railML but the user would need to manually input navigability between alignments.

Overall, this thesis demonstrates that reasonable consistency, beyond minor differences in geometry, can be maintained between IFC and railML, if IFC adds an entity set that can store the relationships among alignments. IFC already offers an analogue to topological representations in the form of distribution flow networks such as pipe networks, which could alternatively be abstracted to fit railway networks. Although IFC is extending in scope, a tool like the one developed in this thesis might still be required.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Goal . . . 3
1.3 Thesis outline . . . 4

2 Data Exchange Schemata in BIM **5**
2.1 Background . . . 5
2.2 Collaboration and Interoperability . . . 6
2.2.1 Common Data Environment . . . 6
2.2.2 Data Exchange and OpenBIM . . . 7
2.3 Data Exchange Schemata . . . 9
2.3.1 IFC and gbXML . . . 11
2.3.2 LandXML, InfraGML, and CityGML . . . 12
2.3.3 OKSTRA and RoadXML . . . 14
2.3.4 agcXML, BCF, and railML . . . 15
2.4 Limitations of Data Exchange . . . 16
2.4.1 Technical Issues . . . 16
2.4.2 Procedural Issues . . . 18

3 Industry Foundation Classes for Railways **20**
3.1 Overview . . . 20
3.1.1 Structure of IFC Schema . . . 21
3.1.2 STEP Format . . . 23
3.2 IfcAlignment . . . 25
3.2.1 Linear Reference System . . . 25
3.2.2 Structure of IfcAlignment . . . 25
3.2.3 Curve Representation in STEP . . . 26
3.3 Future Outlook . . . 28

4 RailTopoModel - Topological Model of railML **29**
4.1 Overview . . . 29

4.1.1 Structure of railML Schema . . . 30
4.1.2 XML Schema and XML Format . . . 30
4.2 RailTopoModel . . . 33
4.2.1 Existing Topological Models . . . 33
4.2.2 Classical Topological Representation . . . 34
4.2.3 Topological Representation in RailTopoModel . . . 36
4.3 Summary . . . 37

5 Solution Proposal and Implementation **39**
5.1 Problem Discussion . . . 39
5.1.1 Data Inconsistency Scenarios . . . 40
5.1.2 Comparisons of IfcAlignment and RailTopoModel . . . 42
5.1.3 Limitations of IfcAlignment and railML . . . 45
5.1.4 Solution Proposal . . . 49
5.2 Implementation . . . 50
5.2.1 Technical Framework . . . 50
5.2.2 Code Structure . . . 51
5.2.3 Geo2Topo . . . 53
5.2.4 Topo&Geo . . . 59
5.2.5 Topo2Geo . . . 60
5.2.6 Visualization . . . 63
5.2.7 GUI . . . 64
5.3 Tool Prototype . . . 64

6 Case Study **67**
6.1 Introduction . . . 67
6.1.1 Data Validation and Viewing . . . 67
6.1.2 Case Selection . . . 68
6.2 Tool Walkthrough . . . 70
6.2.1 IFC to railML . . . 70
6.2.2 railML to IFC . . . 73
6.3 Current Limitations . . . 78

7 Conclusion and Outlook **79**
7.1 Conclusion . . . 79
7.2 Future Outlook . . . 81

A Digital Appendix **82**

Nomenclature

AEC Architecture, Engineering, and Construction

AGC Associated General Contractors of America

AIA International Alliance for Interoperability

AIM Asset Information Model

BCF BIM Collaboration Format

BIM Building Information Modelling

bSDD buildingSMART Data Dictionary

BSI British Standards Institution

bSI buildingSMART International

CAD Computer-Aided Design

CDE Common Data Environment

CPC Construction Progress Coalition

DB Deutsche Bahn AG

DLL Dynamic Link Library

ETCS European Train Control System

gbXML Green Building XML

GIS Geographic Information System

GML Geography Markup Language

GUI Graphical User Interface

GUID Globally Unique Identifier

I Division Deutsche Bahn AG Infrastructure Division

I/O Input/Output

IDM Information Delivery Manual

IFC Industry Foundation Classes

IRS International Railway Standard

ISO International Organization for Standardization

LINQ Language-Integrated Query

LoD Level of Detail

LRS Linear Reference System

MVD Model View Definition

OGC Open Geospatial Consortium

OKSTRA Objektkatalog für das Straßen- und Verkehrswesen

PAS Publically Available Specification

PIM Project Information Model

railML Railway Markup Language

RINF Register of Infrastructure

RTM RailTopoModel

TIN Triangulated Irregular Network

UIC International Union of Railways

UML Unified Modeling Language

WPF Windows Presentation Foundation

XML Extensible Markup Language

XSD XML Schema Definition

XSD.exe XML Schema Definition Tool

Chapter 1

Introduction

1.1 Motivation

"Building Information Modelling (BIM) is the use of a shared digital representation of a built object (including buildings, bridges, roads, etc.) to facilitate design, construction and operation processes to form a reliable basis for decisions." in ISO 29481-1:2016 (ISO, 2016) is one of the BIM definitions. In practice, the BIM process uses a parametric, object-oriented, and dynamic 3D model with embedded geometrical and functional data to share and coordinate information between the modern fragmented industrial field from contractors to subcontractors. BIM strives to create an effective and dependable collaborative process among architecture, engineering, and construction (AEC) during the design, construction and maintenance phases. Therefore, BIM is playing a crucial role in the AEC industries, and thus many commercial BIM software packages, so-called BIM applications in general, are available. Ideally, considering the technical issues, collaboration within a vendor's software environment is easier than collaboration among software from different vendors; in practice, stakeholders in a project tend to adopt the latter type of collaboration. However, BIM applications from different vendors usually have their own internal data standards and formats, decreasing the collaboration efficiency and increasing the risk of information inconsistency. To avoid any misleading communication among different software packages, buildingSMART International (bSI) provides an OpenBIM approach Industry Foundation Classes (IFC) since 1996, certified as an international standard by International Organization for Standardization (ISO) in 2005 (buildingSMART International, 2020e).

The IFC schema defines a software-independent referencing data model that can store BIM models and minimize the loss of information during data exchange scenarios between different software standards. In its first version, IFC 2x3 schema was focused on 3D models, such as architectural buildings and interior components. Later, it extended its scope and included additions such as 3D models with time and cost aspects, often called 4D and 5D models,

and BIM to Geographic Information System (GIS) interoperability. The latest official schema IFC 4.1 introduced an alignment standard (buildingSMART International, 2020d). The extension by GIS and alignment sub-schemata enables IFC to consider the exterior environment and to provide the basis for large-scale infrastructures requiring a Linear Referencing System (LRS) during their design and construction phases, such as roads and railways.

Thanks to the BIM applications and methodologies, various fields, not just buildings but also civil infrastructures, are applying BIM to gain efficiency and reduce expenditures; nevertheless, the benefits of BIM as achieved for buildings are not yet fully realisable in civil infrastructures. Consequently, there are relatively fewer academic studies and industrial applications on infrastructures than those on buildings; BIM implementation is usually limited to the design stage of road, airport, and bridge industries (Shou et al., 2015). The current application on civil infrastructures, especially railways, is not as mature as on buildings due to their complexity. For example, it requires the data schema to not only support geometrical and topological information, but also the control system and functional components to model the complete system. Moreover, Deutsche Bahn AG (DB) started the project of implementing BIM in the Infrastructure Division (I Division) since 2015 and has been standardizing the BIM process, such as BIM applications and data management, in all business units, and yet does not expect an exchange format to meet the DB's complex requirements in the near future (Deutsche Bahn AG, 2019).

As seen from DB's experience, fulfilling the business use case, specifically a complete railway system, involves a high degree of collaboration, and thus requires a BIM application that can handle multiple Levels of Detail (LoD). On the one hand, commercial BIM software can provide sufficient detail on geometry and product information in the design and construction phases, but on the other hand, they can not manage the topological representation in the operation phase. As a result, railway operators tend to use their own internally developed programs and data formats that again increase the difficulty to maintain data consistency. For the same reasons that led to the development of IFC, railML.org introduced the open-source Railway Markup Language (railML) in 2005, enabling heterogeneous railway applications to communicate with each other; its new systemic model of topology-based railway network description RailTopoModel (RTM) was released as International Railway Standard (IRS) 30100 in 2016. The latest schema railML 3.1 provides the data structures, such as infrastructure, interlocking, rolling stock, and timetable, for different operational uses (railML.org, 2020; railML.org, 2016a).

As mentioned above, the purpose of the IFC schema is to ease the interoperability between BIM applications. Accordingly, it is being extended towards further alignment-based domains to keep up with the advancement of BIM applications (buildingSMART International, 2020d); the ongoing IFC Rail project proposed a description of a railway system, a software-independent

conceptual model under review (IFC Rail Project, 2019a,b). It is encouraging that IFC is trying to extend the standard to include the railway infrastructure domain, yet it remains a complex task that will take time to accomplish.

To summarize, bridging the gap between BIM applications for different target markets remains an important stepping stone before the introduction of an integrated BIM model. It is clear that it is impossible to manage railway systems in its entire life-cycle in a single BIM application. Thus, ensuring that there are multiple applications tailored to the specific needs of railways in all different phases, yet still provide compatible interfaces for data exchange, is essential. In particular, it is important to link construction-focused and operation-focused BIM applications through open data schemata such as IFC for construction and railML for operation. Typically the required geometrical data in the operation phase is less detailed than in the construction phase, yet sufficient information should remain available for future upgrades or maintenance. IFC and railML are open-source, vendor-neutral, certified as an international standard, and have great expandability; thus, we can expect that more BIM applications will apply these exchange data formats in the future, on the premise that communication between IFC and railML can be coordinated sufficiently to ensure that data is consistent and can form a complete BIM model.

1.2 Goal

As discussed in the introduction, it is known that the referencing data schemata IFC and railML, also used as exchange data specifications, provide geometrical and topological data models respectively and simplify the connection of various BIM software packages (see Figure 1.1). However, the transferred information from one data model to another can be inconsistent. The sources of possible inconsistency are for example: the topological representation lacking geometrical information to reconstruct the complete geometry; a topological data model consisting of multiple layers with vastly different amounts of information; data being assigned incorrectly due to different semantics between data schemata; the BIM software encoding data formats incorrectly or neglecting to update the topology after changes to the geometry.

Therefore, this master thesis aims to investigate the current exchange data schemata for both geometrical and topological representations for a railway system, discuss the possible reasons for data inconsistency between these two representations, propose and implement a prototype tool to consistently convert a geometrical representation into a topological representation and vice versa, verify the applicability on example cases, and give suggestions for further development. The discussion in this thesis focuses on the application of BIM on railway infrastructure and the relevant sub-schemata from IFC and railML.

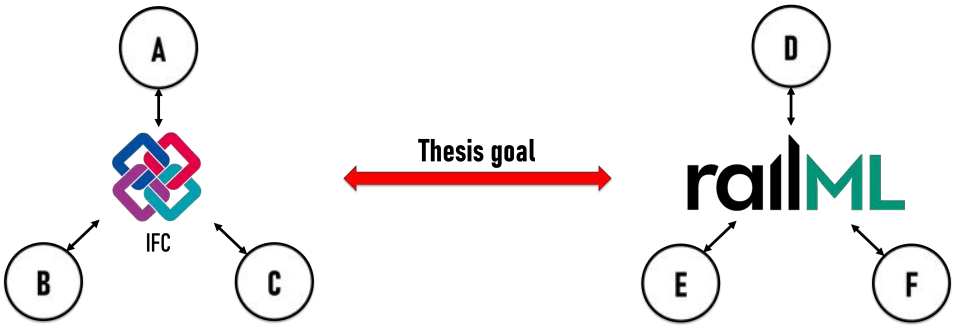


Figure 1.1: Thesis goal: to link the geometrical representation by IFC and the topological representation by RAILML (A to C denote the BIM software supporting IFC; D to F denote the BIM software supporting RAILML).

1.3 Thesis outline

In this thesis, Chapter 2 introduces the concepts of collaboration and interoperability in BIM, reviews and compares the existing open data standards and data formats, and discusses the limitations of data exchange schemata. Then, IFC and RAILML are introduced in detail in Chapter 3 and Chapter 4 respectively; Chapter 3 analyzes the IFC data structure, the format, the possible components to model railway infrastructure available in the current IFC 4.1 schema, and the description of alignments as an LRS. Subsequently, Chapter 4 introduces the RAILML schema, the topological model RTM, and the difference between RTM and a classical topological representation. After clarifying the issue, Chapter 5 discusses the data inconsistency scenarios and the data for reconstructing a geometrical representation from a topological representation and vice versa. I also propose a possible solution and implement the concept, followed by testing the prototype solution with cases in Chapter 6. Finally, Chapter 7 summarizes the results and gives an outlook and possible extensions.

Chapter 2

Data Exchange Schemata in BIM

Section 2.1 introduces why data exchange is important in BIM and how it influences collaboration and interoperability in BIM. Next, Section 2.2 explains the proposed solutions, i.e. Common Data Environment (CDE) as well as the idea of data exchange schemata. Section 2.3 simply introduces and compares the common data exchange schemata used in AEC industries. To conclude, Section 2.4 addresses the limitations of data exchange schemata.

2.1 Background

It is indispensable for AEC industries to implement a full BIM workflow, as it has the potential to improve both productivity and efficiency significantly; it is especially advantageous for railway infrastructure due to the complex business requirements. A complete model of a railway infrastructure encompasses a wide range of components in different LoDs from roughest to finest detail; for example, network topology for route design; track layout of the individual routes for a switching layout; description of individual objects and projects with all associated assets for detail design; and down to the smallest exchangeable units for manufacturing (Deutsche Bahn AG, 2019; Borrmann et al., 2012). It is difficult, however, for a single BIM application to manage vastly different scales in a single model or to provide multi-user capabilities, hence, a BIM project mostly involves multiple applications for different use cases and phases.

BIM applications can be grouped into three types based on their functions: BIM tools, BIM platforms, and BIM environments; a BIM tool can be any application used in the context of the BIM process; a BIM platform can create, modify, and maintain a BIM model, as well as host project information. A BIM environment is a set of BIM applications, supporting information for different businesses use cases (Sacks et al., 2018). The current design software packages available on the market belong mostly to the BIM platform group. These design software

packages provide an object-oriented parametric data model that supports the core concept of BIM. A parametric model aggregates multiple objects whose shape and other properties can be controlled parametrically by their relationships (Baldwin et al., 2018; Sacks et al., 2018); for example, a room has four walls, and each wall has windows; creating, modifying, or removing one object can affect the higher-level object. Therefore, a BIM model can maintain information consistency as changes automatically propagate up or down in level. Nevertheless, design software packages from different vendors commonly have diverging object definitions; that is, a BIM model created in a specific software can not be passed directly to another software. As the result of the independent models, it increases the difficulty to share common information and decreases the collaboration and interoperability between project members.

2.2 Collaboration and Interoperability

Collaboration and interoperability is the key to effective project management making full use of the advantages of BIM. In the modern AEC industries, a large-scale project is mostly divided into several parts and then assigned to multiple subcontractors; thus, a collaboration with partners from diverging industry sectors is unavoidable. In addition to the collaboration, the interoperability, which refers to the ability to exchange data among applications, is also a factor to save time and cost. To ensure optimal collaboration and interoperability, a well-structured information management system including a standardized data pool and loss-free data exchange is desired.

2.2.1 Common Data Environment

The scope of information management consists of collecting, updating, and synchronizing graphical or non-graphical documentation, models, and files. As the project progresses, more stakeholders will be involved, and thus the complexity of the information management will grow as well. To reduce communication conflicts, CDE provides a platform for joint, standardized, and systematic information management. British Standards Institution (BSI) published two Publicly Available Specifications (PAS) PAS 1192-2 and PAS 1192-3, introducing CDE formats in the data delivery and the operational phases respectively (British Standards Institution, 2020; Baldwin et al., 2018). PAS 1192-2 and PAS 1192-3 were later included into ISO 19650 in 2018. The standards define the CDE structure and give the data manager the criteria and guidelines to organize, update, verify, and transmit data resources.

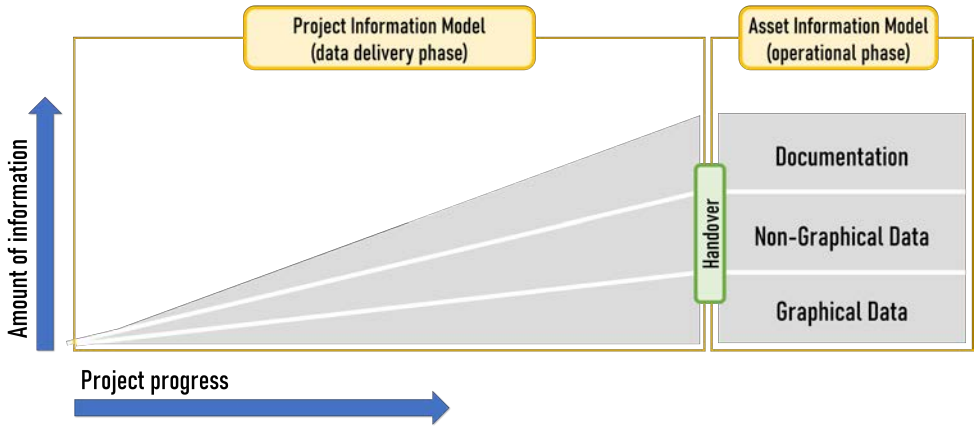


Figure 2.1: Project delivery in CDE (based on ISO 19650).

Figure 2.1 shows a project delivery using CDE. The project is progressing along the abscissa, and the amount of information is increasing along the ordinate accordingly. There are three types of data-subsets in CDE: graphical data, i.e. a 3D model; non-graphical data, i.e. object attributes; and documentation, i.e. analysis text files. The amount of all data increases as the project progresses during the data delivery phase, followed by the confirmed information being handed over to the owners in the operational phase. The information in the data delivery phase is called the Project Information Model (PIM), while the information in the operational phase is called the Asset Information Model (AIM). PIM represents the data which is created, confirmed, and utilized sequentially under design and construction, e.g. 3D models, construction workflows, and production plans. In contrast, the planning of activities, such as maintenance and extensions, is more flexible in the operational phase than in the data delivery phase. During design and construction, PIM data, such as geometry detail, is produced in abundance but not always necessary for the operational phase; hence, AIM is commonly just a subset of PIM (Baldwin et al., 2018). A good example is the timetable design for railways only requiring the topological model of tracks and stations rather than the geometrical model including curvature information.

Overall, it can be said that the collaboration between project participants during all phases is based upon the foundation laid by the CDE.

2.2.2 Data Exchange and OpenBIM

Having defined the shared information platform CDE, Section 2.2.2 is now moving towards how to coordinate information sharing between applications. As shown in Figure 2.2, a CDE enables project members to synchronize and share dynamic information regardless of their chosen applications. The CDE is however just a data pool collecting data from participants, which does not automatically mean that it is a single, centralized, and neutral model. As it is neither beneficial for companies to develop a single integrated product that can model

an object's full life-cycle, nor is it in the interest of end-users from various domains to be limited to a singular choice in software, how to obtain an integrated common model remains a significant issue.

An integrated model is not a single monolithic model but an aggregation of multiple data sets created by specialized applications best suited for each use-case scenario. To achieve an integrated model, frictionless data exchange between each authoring and simulation tool is needed. Possible approaches to data exchange are for example: (1) using BIM authoring software from the same vendor; (2) using BIM authoring software from different vendors that support shared proprietary file formats; (3) using BIM authoring software from different vendors that support common non-proprietary file formats, also known as open standards. The first and second are easier approaches than the third, as the applications can effortlessly import and export data through their internal common format schema such as *.DWG* for AutoCAD and *.RVT* for Revit of Autodesk, yet they give the participants less choice in applicable software packages. Moreover, it is the premise of data exchange that these software packages must cover all the required domains in the project. On the other hand, the third approach is more flexible than the first two and the optimal solution for interoperability (Sacks et al., 2018; Baldwin et al., 2018), but demands the extra implementation of open standards into the applications.

OpenBIM is a BIM process adopting open standards, while ClosedBIM is a BIM process adopting native data standards. An open standard, to be more specific, is a standard that is publicly available and not limited to specific software or technology. As all native models and data created in divergent software packages can be connected by using open standards (see Figure 2.2), architects, engineers, and manufacturers can fully concentrate on their tasks while the information transmission and workflow can proceed smoothly.

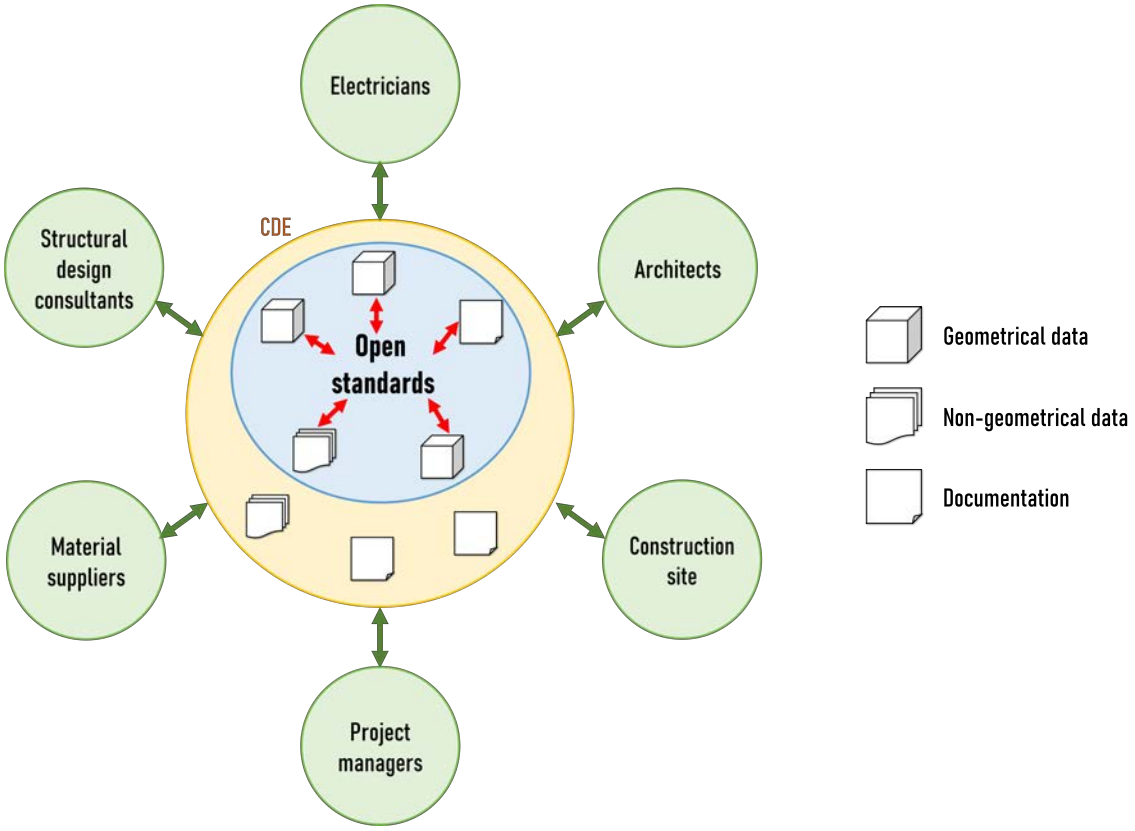


Figure 2.2: The concept of OpenBIM.

2.3 Data Exchange Schemata

With the development of Computer-Aided Design (CAD), digital documentation has replaced traditional paper-based documentation and became the primary documentation carrier in many industries, formalizing the need for standardized data exchange formats. The advantages being that project members can create a model in any given BIM application, export the model data into the shared exchange format, and then import the file into another BIM application more suited to the next use-case and continue working. That means, users do not need to first manually re-create the model or extract information from the model themselves but can rather start working immediately.

Open data exchange standards used for a BIM process specify the data schemata which are, for example, IFC, Green Building XML (gbXML), LandXML, CityGML, **O**bjekt**k**atalog für das **Stra**ssen- und Verkehrswesen (OKSTRA), railML, RoadXML, agcXML, BIM Collaboration Format (BCF), and InfraGML. Their initial release date, latest official version, supported formats, scopes, application and whether they support the description of railway geometry are listed in Table 2.1.

Standards	Initial release	Latest official version	Format	Scopes	Application	Supports LRS
IFC (bSI)	1996	4.1 (2018)	XML, STEP, (unofficial: JSON)	Buildings	Construction/facility management, ongoing extension towards further applications	allows LRS positioning via IfcAlignment
OKSTRA	1999	2.019 (2019)	XML, (deprecated: CTE)	Road transportation	Road transportation	Not complete; Trasse class
gbXML	2000	6.01 (2017)	XML	Buildings	Analyses of building energy, water use and cost, carbon emissions	No
LandXML	2002	1.2 (2008)	XML	Land development	Land development, transportation industries	Not complete; Alignment entity
CityGML (OGC)	2002	2.0 (2012)	XML	City, landscape	Urban planning, architectural design, tourist and leisure activities	Yes; Transportation module
railML	2005	3.1 (2019)	XML	Railway systems	Infrastructure, interlocking, rolling stock, timetable	Yes
RoadXML	2007	2.4.1 (2016)	XML	Logical description of road networks	Traffic simulation, scenario control car, truck and motorbike dynamics models	Not complete; XYCurve and SZCurve classes
agcXML	2008	-	XML	Construction business processes	Contractor record, application for payment, time sheets	No
BCF (bSI)	2009	-	XML	Construction workflow	BIM issue management separated from 3D model	No
LandInfra/ InfraGML (OGC)	2017	1.0 (2017)	XML	Infrastructure, land surveying	Land development, transportation industries	Yes; Part 3 Alignment Part 5 Railway

Table 2.1: Sample open data exchange schemata used in a BIM process (Data as of 27. February 2020) (buildingSMART International, 2020d; Open Green Building XML Schema, Inc., 2020; LandXML.org, 2017; Gröger et al., 2012; Bundesanstalt für Straßenwesen, 2020; railML.org, 2019b; Ducloux, 2016; AGC of America, 2017; Solibri, Inc. and Tekla Corp., 2019; Axelsson and Wikström, 2017).

As shown in Table 2.1, all listed data exchange schemata support the schema Extensible Markup Language (XML), which is a textual data format, both human- and machine-readable, and suitable for web services. The scopes of these schemata cover not just buildings, but also infrastructures, and land: IFC and gbXML are focused on building models and the relevant analyses from design, construction, and management; LandXML, InfraGML, and CityGML support the detailed description of surrounding land for the planning of large-scale civil infrastructures by adopting GIS; OKSTRA and RoadXML are targeted towards traffic and transportation engineering; in addition to the schemata focusing on design and construction, there are also schemata offering business-side, workflow management, and operation support, namely agcXML, BCF, and railML. The following sections introduce each data exchange schema respectively.

2.3.1 IFC and gbXML

IFC

IFC, initiated in 1994 by the International Alliance for Interoperability (AIA), now renamed to bSI, was the prototypical data schema in the construction sector and was inspiring AEC industries to implement the electronic exchange of technical data between native models. IFC is the most common data exchange schema in the AEC industries. Later, Chapter 3 introduces IFC in detail.

gbXML

Other than IFC, gbXML also aims to transfer BIM models, especially among the environmental analysis software packages. In 2000, the initial version of gbXML was published by Green Building Studio, Inc., which was later acquired by Autodesk in 2008. The core idea behind gbXML is to describe multiple related buildings located in the same climate region; for example, a campus and property set; each building in the set can be further represented in detail including the geometry, materials, air-side systems, water-side systems, as well as further properties that influence power consumption. Even though gbXML is weak on the building geometry such as only accepting rectangular shapes, it is still often used for the energy analysis due to its easy implementation (Adamus, 2013; Dong et al., 2007). According to the official website of gbXML, the team are still developing gbXML actively; 16 BIM authoring and CAD software packages as well as 42 building analysis tools integrated gbXML (Open Green Building XML Schema, Inc., 2020).

2.3.2 LandXML, InfraGML, and CityGML

The earliest developed data schemata mostly focused on building models with a maximum of detail in their geometry for the design and construction phases. Despite the AEC industry's acceptance of data exchange schemata, civil infrastructure projects did not yet enjoy the full benefit of the BIM process due to the lack of spatial or geographical analysis capability in BIM (Karan et al., 2016). The distinct feature of civil infrastructures, large land occupation, differs significantly from the land needs of buildings. Moreover, an infrastructure project, for example a railway system, includes not only the to be developed objects, such as stations and tracks, but also the existing environment, i.e. geography; hence, GIS and the accompanying GIS software packages were introduced to fulfil that need. Again, having multiple independent GIS applications hinders the data interoperability. Since 1994, Open Geospatial Consortium (OGC), an international voluntary consensus standards organization, addressed and has created non-proprietary formats for GIS as interfaces between GIS applications. Since then, many different data schemata have been developed, such as LandXML of LandXML.org, and InfraGML as well as CityGML of OGC.

LandXML

LandXML is an XML-based schema for civil engineering design and survey measurement data first published in 2002. The purposes of LandXML are transferring engineering design data between producers and consumers, providing a data format suitable for long-term data archival, and providing a standard format for electronic design submission (LandXML.org, 2017). LandXML used to be the most popular data schema in the land development and transportation industries, however, it is gradually being replaced by other GIS schemata due to several issues. Some examples are: the schema is too flexible as it contains a comprehensive set of properties with only optional implementation, which is opposed to the concept of interoperability; the schema is not organized in a structured manner, thus it is difficult to find entities without an electronic search; the name requirements and the naming logic are inconsistent between parent classes and their children classes (Scarponcini, 2013). Moreover, LandXML not being supported by a standards organization also confused the industrial uptake (Scarponcini, 2013). Following the latest schema LandXML 1.2, published in 2008, the in 2014 newly developed LandXML 2.0 is until today still only a working draft (LandXML.org, 2017).

InfraGML

Alternatively, OGC Land and Infrastructure Domain Working Group proposed a new solution in 2013 to replace LandXML, called InfraGML. The data model of InfraGML

adopted the XML-based Geography Markup Language (GML) issued by OGC and the ISO TC211, the new schema, therefore, benefits from the functionality already supported by GML, including features, geometry, coordinate reference systems, linear referencing, and surface modelling using a Triangulated Irregular Network (TIN) (Scarponcini, 2013). Compared to LandXML, InfraGML has a more comprehensive definition for infrastructure with multi-part standards for the description of alignment, facilities and projects, roads, railways, land surveying, and land division (Axelsson and Wikström, 2017). Due to the overlap in alignment functionality with IfcAlignment, it has been decided to harmonize the two schemata to guarantee future interoperability (Scarponcini, 2013). Even though the railway infrastructure geometry in InfraGML is not as detailed in IFC, it can be a good interface to connect IFC with other GIS data schema (Axelsson and Wikström, 2017; Kumar et al., 2019).

CityGML

In addition to the schema targeting civil infrastructures, OGC offers another GML-based schema CityGML for the representation, storage, and exchange of virtual 3D landscape models. The application scope includes, for example, urban and landscape planning, architectural design, tourist and leisure activities, environmental simulation, and mobile telecommunications (Gröger et al., 2012). The CityGML schema is thematically decomposed into a core module and thematic extension modules; the mandatory core module provides the basic functionality of the CityGML data model; the extension module builds upon the core module to provide the thematic extensions. The thematic modules encompass thirteen fields: digital terrain, buildings, tunnels, bridges, water bodies, transportation objects, vegetation objects, city furniture, land use, city object groups, generic city objects and attributes, and application domain extensions (Gröger et al., 2012). One other characteristic of CityGML is the multi-scale data schema supporting multiple representations in five LoDs from the coarsest level LoD 0 to the finest level LoD 4. To take railways as an example: LoD 0 shows the linear network of railways; starting from LoD 1 shows 3D geometrical features of railways. The information in different detail can, for example, be employed for route planning algorithms for transportation or pedestrians (Gröger et al., 2012). CityGML 3.0 has been recently presented in (Kutzner et al., 2020), however due to the as yet unfinished nature of the standard is not included in the thesis discussion.

2.3.3 OKSTRA and RoadXML

OKSTRA

The German schema OKSTRA is targeted at road and traffic engineering. OKSTRA was initiated in 1993, introduced by the Federal Ministry of Transport and Digital Infrastructure in Germany in 1999 and made mandatory for all road infrastructure projects in Germany the following year (Rüffer et al., 2001). It was introduced to harmonize the road network planning of the German federal states, while also providing interfaces for state-specific extensions. OKSTRA is one of the most extensive road data schemata in the world (Beetz and Borrmann, 2018); however, the schema and the documentation being solely available in German has limited the adoption to the German market only (Amann et al., 2014). The development goals were to provide data storage and facilitate the exchange of all information that might be relevant towards the design, construction, operation, and maintenance of a transportation network. The foundation of OKSTRA is made up of three basic schemata: a geometry schema containing all geometrical and topological information relevant to the object; a history schema serving as documentation storage for all versioned historical data of an object; and a general object schema providing the basis for the further 31 application-specific schemata. The application-specific schemata include for example schemata for accident statistics, traffic statistics, signalling, road condition, land-use, and administrative data (Hettwer, 2008).

RoadXML

In 2006, the french car manufacturer PSA Peugeot Citroen released the first version of RoadXML under its original name of RND. Later, the schema was integrated with other data schemata, e.g. RNS/RS, for traffic software by several vehicle related-companies (Ducloux, 2016). All the companies, including INRETS, OKTAL, PSA, Renault, Thales, and TRL, are currently responsible to review and maintain the schema (INRETS et al., 2016). Since the developers are mostly car manufacturers, the first concept of RoadXML was to have one format for all driving simulator modules and to be a single point of access to the network description (Chaplier et al., 2010). Today, RoadXML is also used for traffic simulation, sound control, and vehicle dynamic analysis (INRETS et al., 2016). RoadXML offers a four-layer description of the environment with different LoDs: topological, logical, physical, and visual layer. The topological layer represents the position of the road axis and the connections with the network; the logical layer contains the lanes with width and direction; the physical layer describes road attributes in more detail e.g. surface properties and the material; the visual layer has the most detailed information, e.g. road marking width (Ducloux, 2016).

2.3.4 agcXML, BCF, and railML

agcXML

agcXML aims to support construction business processes. It was developed by Associated General Contractors of America (AGC) in 2008 and then taken over as an open standard by Construction Progress Coalition (CPC) in 2018 (AGC of America, 2017). agcXML provides multiple schemata for various cases, for example, in the schema *Submit and Distribute Application for Payment*, it defines activities including *submitting the payment to the owners*, *receiving the payment by the owners*, and *distributing the payment to consultants* (AGC of America, 2017).

BCF

Other than most data exchange schemata that focus on describing models, BCF, originating from Solibri and Tekla in 2009 and owned by bSI today, primarily focuses on identifying and sharing model-based issues between BIM applications (buildingSMART International, 2020a). It is known from Section 2.2.2 that data exchange schemata allow team members to communicate among different BIM software packages. Nevertheless, the communication does not happen at the same time but is instead sequential. A typical workflow when BCF is considered in the design is shown in Figure 2.3; first, a model is created; then the model will be export into a data format such as IFC and be passed to the next stakeholders; BIM managers or coordinators verify and analyze the model using for example clash detection, and then report the issues with screenshots and comments using BCF; finally, the modellers modify the model according to the feedback. BCF optimizes the workflow and can be utilized not just for the model review in the design phase, but also for information coordination among suppliers in the procurement phase, for the quality control in construction, and upgrades in the operation phase (buildingSMART International, 2020a).

railML

railML is developed to bridge the railway operation applications. Compared to infraGML and CityGML, railML can describe the railway geometry and also have a comprehensive data set for the facilities as well as properties needed in the operation phase, i.e. the types of switches, platforms, signals, and the train speeds (railML.org, 2019b). As railML is focused on operational purposes, the data extent is relatively light-weight compared to IFC but complements the insufficient data content available in the current IFC version. Chapter 4 introduces railML in more detail.

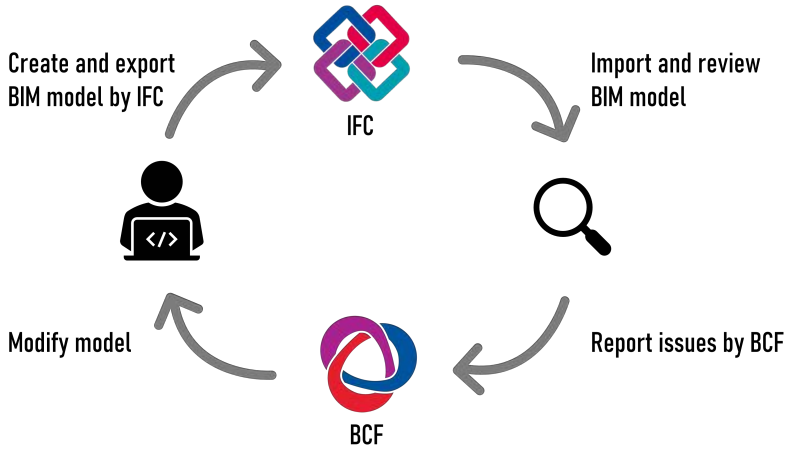


Figure 2.3: Collaboration workflow with BCF (Baldwin et al., 2018).

2.4 Limitations of Data Exchange

More and more data schemata being developed and extended aim to improve the interoperability in a BIM process; however, they can not guarantee the interoperability, for example, due to data inconsistency. The sources of the interoperability problems are mainly attributed to technical and procedural issues. The technical issues are: (1) incomplete coverage of a data schema; (2) translator problems; (3) system bugs; (4) software domain problems. On the other hand, the procedural problems are: (1) version control and concurrent engineering issues; (2) LoD issues. In addition to the reasons above, it can also be the fault of team members not willing to share data due to intellectual property (Lee, 2011; Sacks et al., 2018).

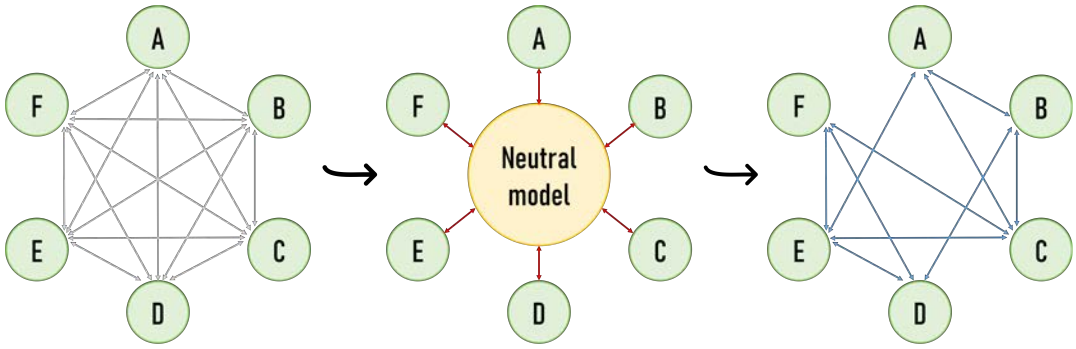
2.4.1 Technical Issues

Incomplete Coverage of a Data Schema

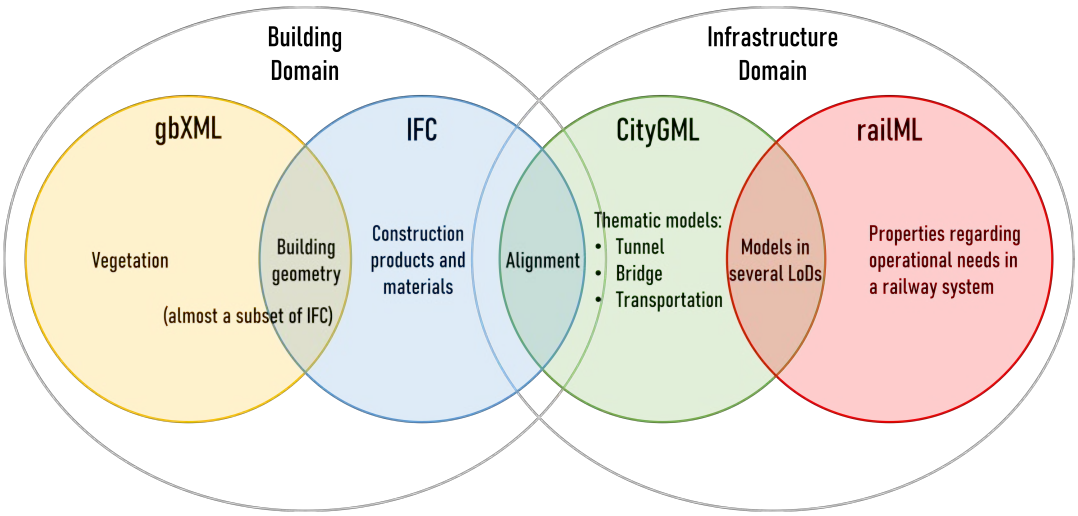
As introduced in Section 2.3, developers have been trying to have data exchange schemata cover the entire life-cycle of facilities. The motivation of a data exchange schema was to have a neutral model for multiple business processes, yet a neutral model does not exist. That is, each software requires one data schema to be translated to another one.

Assume there are n independent software packages used in a project, on the left in Figure 2.4a, then $n \cdot (n-1)$ direct translators without data exchange schemata are required to ensure the interoperability among n applications. The ideal solution is to have a neutral model connecting all applications as shown in the middle, reducing the direct translators to n; however, in practice, the problem just reduces from $n \cdot (n-1)$ to $n \cdot m$ direct translators where m refers to the number of data exchange schemata on the right (Gielingh, 2008).

Given that there is no neutral model, linking exchange schemata to maintain the information flow is unavoidable; nevertheless, only the common information existing in both schemata can be exchanged (Gielingh, 2008). For example, the current IFC contains comprehensive types, entities, and property sets for buildings, yet its scope is not covering the complete description of infrastructure; only the information of ALIGNMENT of CITYGML can be transmitted into the entity IFCALIGNMENT of IFC; unfortunately, the other infrastructure information, e.g. the properties relating to tunnels, bridges, and transportation systems, will be abandoned during translation, as there are no suitable entity types to store the information (see Figure 2.4b). Moreover, the natural information flow is usually from the detailed to the simple; it requires users to define extra information manually in the conversion from the opposite direction. For example, the building definition in GBXML is almost a subset of IFC, thus data transaction from GBXML to IFC is easy; however, the data schema of GBXML does not contain all information that IFC requires, such as construction products as well as materials (see Figure 2.4b) (Adamus, 2013; Dong et al., 2007; Daniotti et al., 2020).



(a) A neutral model does not exist (Gielingh, 2008).



(b) Only intersected information between schemata can be exchanged (the entries are examples).

Figure 2.4: Data exchange issues.

Figure 2.4b shows that data inconsistency occurs when there is no explicitly defined corresponding entity type. Rather than the naive assumption that the intersection of two schemata can be exchangeable (Gielingh, 2008), one must introduce the concept of *semantic intersection* (Lee, 2011) to determine the true extent of exchangeable information. To take an example, CityGML is typically used for the GIS approach in the city or neighbourhood-scale scenario, whereas IFC is used for the BIM approach in the building-scale scenario. Despite the common information, it is still necessary to link the information of both CityGML and IFC through an explicit conversion, because the unified model under development for the integration of both schemata loses the original semantics (Jusuf et al., 2017). For another example, HEIGHT, WIDTH, and DEPTH are three attributes defining a box volume in schema A; on the other hand, schema B uses VOLUME to represent the same thing; even though the volume can be determined by multiplying HEIGHT, WIDTH, and DEPTH, the attributes HEIGHT, WIDTH, and DEPTH can not be parsed from VOLUME.

The semantics of the property names are sensitive during encoding. If the names are ambiguous then they can not be interpreted correctly. These errors can, of course, be picked up by a human, but the merits of automatic data exchange would be lost. The semantics confusion is primarily caused by synonyms and homonyms (Lee, 2011). An example for a homonym, set A contains PROJECT NAME, LOAD, and MANAGER, while set B has STRUCTURE NAME, LOAD, and FRAME; only the information intersection LOAD can be exchanged, however, LOAD in A means structural load and in B means truckload. In the synonym case, the different attribute names PROJECT NAME and STRUCTURE NAME refer to the same thing (Lee, 2011). These semantic problems are considered to defined the data inconsistency scenarios in Chapter 5.

Translator Problems, System Bugs, and Software Domain Problems

Translator problems can occur, as a translator does not have a guideline to follow. Further problems can be caused just due to simple bugs; a bug in the visualization module of an application can lead to the wrong model appearance even though the data is read correctly. Besides, similar to problems caused by the incomplete coverage of a data schema, a BIM platform only reads the relevant entities in the corresponding domain and omits other information.

2.4.2 Procedural Issues

As multiple participants are in a project, unsynchronized information is a common cause for reduced interoperability as well. For example, an inconsistency can occur when engineers have modified a model and pushed the update, yet architects continue to work on the older version. The other reason is the different LoDs used in a BIM model. In most cases, a single BIM model can not include all the details required for different use cases, therefore the model is

split into several models with a different LoD; consequently, data might be lost among the models or not propagated correctly (Sacks et al., 2018).

Chapter 3

Industry Foundation Classes for Railways

First, Section 3.1 gives an overview on the background of IFC and a short introduction regarding the standards also established by bSI. Following this, the basic properties and architecture of the IFC schema is explained; an example for building elements is made and a sample STEP file is given. Section 3.2 discusses the motivation for the development of IfcAlignment and introduces the concept of the LRS as well as the structure of the subschema IfcAlignment. For a clear explanation, an example curve implemented in STEP is given. Finally, past and present research about the extension of infrastructures in IFC and the collaboration between data schemata is introduced.

3.1 Overview

The IFC schema of bSI, an open international standard ISO 16739-1:2018 for BIM, is the most commonly used data schema in AEC industries. The up to date schema IFC 4.1 was released in 2018. In addition to the schema entities aimed at storing the full life-cycle of buildings, the new updated IFC 4.1 includes alignment entities, providing the basis for many infrastructure domains using an LRS such as railways, roads, tunnels, ports, and waterways (buildingSMART International, 2018). Today, IFC is implemented and available in at least 88 BIM applications from well-known software enterprises, such as ACCA Software S.p.A, Autodesk, Bentley, NEMETSCHEK Allplan GmbH, and Tekla (Data as of 27. February 2020).

In order to facilitate implementation of the IFC schema, bSI also published several standards: Model View Definition (MVD), Information Delivery Manual (IDM), buildingSMART Data Dictionary (bSDD), and BCF (see Section 2.3.4). An MVD is a subset of the

IFC schema, allowing users to extract only the data information relevant to specific scenarios. The concept of MVDs is to reduce the model size and to simplify the implementation for various data exchange scenarios based on sub-domains. Moreover, the data exchange interface of sub-domain specific BIM software is validated against a selected MVD (Mahdavi et al., 2014). The MVDs' format MVDXML defines the data schema including all IFC expressions such as entities, relations, and attributes for particular views (Tim Chipman, 2016; Sacks et al., 2018). For example, structural engineers can export structure-related information for analysis; a wall-fabricator can export wall geometries and the material properties for production. To create an MVD, the relevant data exchange process needs to be analyzed; the analysis consists of what information is needed for the selected scenario and what information needs to be transmitted (Baldwin et al., 2018). The specification of the information requirements is called an Information Delivery Manual (IDM). The creation of an IDM is a collaborative process whereby a group of industry domain specialists collate the minimum exchange requirements for each domain into an exchange requirements model that forms the basis of the MVD (Richard See, 2012). bSDD is a dictionary to overcome the language barrier in an international collaboration. For example, the natural language names *window* in English and *Fenster* in German both refer to the same IFC entity IfcBuilding (buildingSMART International, 2018).

The IFC schema relies on the EXPRESS data specification language, defined in ISO 10303-11, and the XML Schema Definition (XSD) language, defined in XML Schema W3C Recommendation (buildingSMART International, 2018); the data model based on EXPRESS is stored in a STEP physical file with the file extension *.ifc*, whereas the one based on XSD is stored in an XML file with the file extension *.ifcXML*; also, the data model can be encoded in other formats such as JavaScript Object Notation with the file extension *.json*. Currently, the STEP physical file is the preferred choice (Liebich, 2009); it is introduced in Section 3.1.2.

3.1.1 Structure of IFC Schema

The data schema architecture of IFC can be decomposed into four parts; each part is assigned to one conceptual layer. The four conceptual layers from the lowest to the highest are the resource layer, core layer, interoperability layer, and domain layer (buildingSMART International, 2018). There are either rooted or non-rooted entities in each conceptual level; the resource layer includes only non-rooted entities, while the core layer, interoperability layer, and domain layer include only rooted entities. According to (buildingSMART International, 2018), the definition of each layer and an example of an IFC schema are given below.

Resource layer

The lowest layer includes all the individual base schemata for resource entities. These non-rooted entities do not have an identity, i.e. a Globally Unique Identifier (GUID), and can not exist independently. The resource entities need to be assigned to or referenced by an entity from a higher level directly or indirectly, and thus are reusable. Due to the reusable characteristics, the file size can be minimized by sharing instances of identical resources. The entities can be classified into 21 groups according to their use purposes; for example, geometry, topology, material, quantity, and cost.

Core layer, interoperability layer, and domain layer

All entities in this group are rooted; they are derived from the most abstract root class IfcRoot, which have their own identity, name, description, and owner history, hence, they can be used independently. The core layer defines both abstract objects, e.g. relationships among entities, and physical objects, e.g. building elements; these fundamental objects can be used for further specialization in aspect specific models. The next layer, interoperability layer, contains the entities that are specific to a general object, such as the derived entities of building elements, beams, columns, doors, and piles; these entities are called Shared Elements and are used for inter-domain exchange and sharing of construction information. In the top layer, the entities are domain-specific extensions and used for intra-domain exchange and sharing of information.

Example

IFC provides an object-oriented data model. All object information is organized following the inheritance hierarchy. Figure 3.1 shows a subset of IFC entities related to building objects. There are three abstract types of entities inheriting from IfcRoot; IfcObjectDefinition, defining a type or an occurrence as an object; IfcPropertyDefinition, defining an object's characteristics; and IfcRelationship, defining the relationship among objects. Starting with the children nodes of IfcProduct, IfcElement and IfcSpatialElement are derived from IfcProduct; IfcElement is further inherited by various elements that can be referred to by a spatial element through a relationship; IfcSpatialElement decomposes a building into several elements, namely buildings, stories, sites, and space, which can be aggregated through the entity IfcRelAggregates, inherited from IfcRelationship. To take a four-story building as an example, first, beams and columns are associated with each story; then four stories are aggregated and assigned to a building, followed by putting the building in a site; finally, the site is assigned to a project.

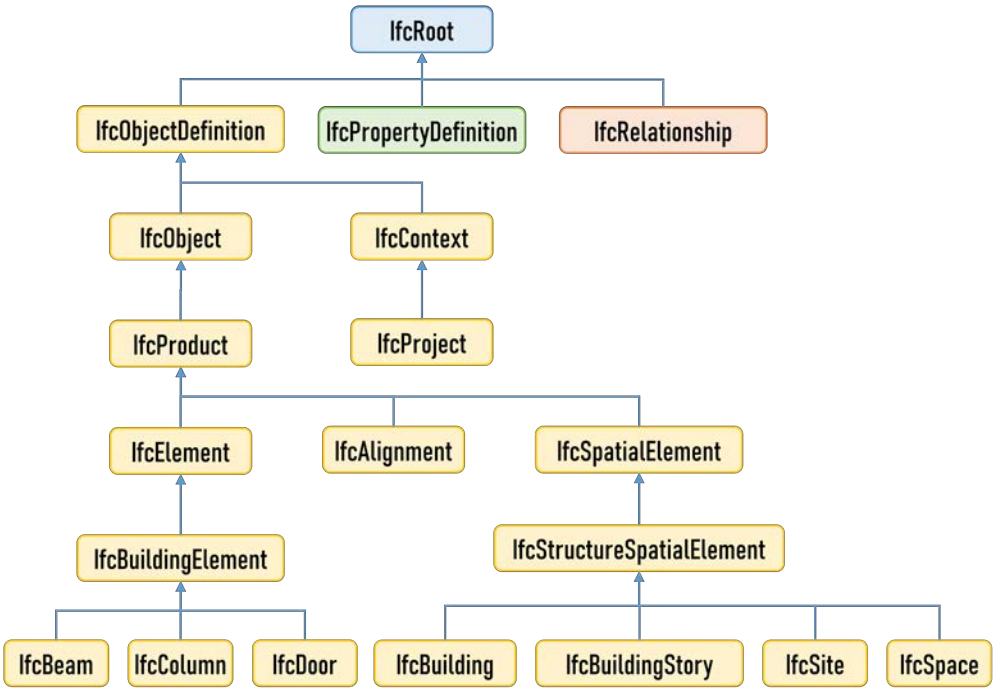


Figure 3.1: A subset of IFC entities related to building objects (the lines represent the inheritance hierarchy).

3.1.2 STEP Format

A simple example for the STEP format is shown in Code 3.1. The file is split into two sections, **HEADER** and **DATA**. The **HEADER** section has information about the IFC version, the owner, the application that created the file, and the date of exportation, whereas the **DATA** section contains instances of IFC entities used to construct a BIM model (Liebich, 2009). Each statement in the **DATA** section is composed of a STEP Id, the entity name, and its attributes. A STEP file is a human-readable text-based file, yet not possible to read sequentially, as the entities are nested in the attributes of other entities. For example, the entity IfcProject as seen in the line with the STEP Id #1 has the first three attributes GlobalId="1 Ed8FFl9LDCgUmtp0YgCdb", OwnerHistory= "#2", Name="IfcAlignmentProject"; the second attribute OwnerHistory refers to IfcOwnerHistory with the STEP Id #2 in Line 10, of which the first two attributes further refer to the entities with STEP Id #5 and Id #6. Additionally, as the STEP Id is only valid for a single exchange, the Id will change if the same file is exported again (Liebich, 2009). Thus, an IFC file should be utilized as a reference model, which needs to be loaded into a BIM platform and then exported again from the updated BIM model rather than trying to modify the file directly (buildingSMART International, 2020e).

```
ISO−10303−21;
HEADER;
FILE_DESCRIPTION ((''), '2;1');
FILE_NAME ('', '2020−03−14T22:41:34', (''), (''), 'Processor version  5.1.0.0',  'Xbim.IO.MemoryModel', '');
FILE_SCHEMA (('IFC4X1'));
ENDSEC;

DATA;
#1=IFCPROJECT('1Ed8FFl9LDCgUmtp0YgCdb',#2,'IfcAlignmentProject',$,$,$,$,$,#7);
#2=IFCOWNERHISTORY(#5,#6,$,.ADDED.,1584225694,$,$,0);
#3=IFCPERSON($,'last name','first name',$,$,$,$,$);
#4=IFCORGANIZATION($,'Technical University of Munich',$,$,$);
#5=IFCPERSONANDORGANIZATION(#3,#4,$);
#6=IFCAPPLICATION(#4,'1.0','TUM_CMS_SE','notDefined');
#7=IFCUNITASSIGNMENT((#8,#9,#10,#11));
#8=IFCSIUNIT(∗,.LENGTHUNIT.,$,.METRE.);
#9=IFCSIUNIT(∗,.PLANEANGLEUNIT.,$,.RADIAN.);
#10=IFCSIUNIT(∗,.AREAUNIT.,$,.SQUARE_METRE.);
#11=IFCSIUNIT(∗,.VOLUMEUNIT.,$,.CUBIC_METRE.);
#12=IFCLOCALPLACEMENT($,#13);
#13=IFCAXIS2PLACEMENT3D(#14,$,$);
#14=IFCCARTESIANPOINT((0.,0.,0.));
#15=IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,$,#13,$);
#16=IFCSITE('2LwWAD9XP9ZufFEuMlzxvE',#2,'SiteName','SomeTestSite',$,#12,$,$,$,$,$,0.,$,$);
#17=IFCRELAGGREGATES('0jaemTaLb3wO_sdAoezuA5',#2,$,$,#1,(#16));
#18=IFCPROJECTEDCRS('EPSG:31467','EPSG:31467 − DHDN / 3−Degree Gauss−Kruger Zone 3','EPSG:3146',$
      ,'Gauss−Kruger','3',#8);
#20=IFCMAPCONVERSION(#15,#18,1.,2.,0.,$,$,$);
#21=IFCALIGNMENT('0qpyZi7wPD58Gn0AbXm1IN',#2,'myCurve','myDescription',$,#12,$,#22,$);
#22=IFCALIGNMENTCURVE(#23,#24,$);
#23=IFCALIGNMENT2DHORIZONTAL($,(#27,#30));
#24=IFCALIGNMENT2DVERTICAL(());
#25=IFCLINESEGMENT2D(#26,1.5707963267948966,1000.);
#26=IFCCARTESIANPOINT((0.,0.));
#27=IFCALIGNMENT2DHORIZONTALSEGMENT($,$,$,#25);
#28=IFCCIRCULARARCSEGMENT2D(#29,1.5707963267948966,785.39816339744823,500.,.F.);
#29=IFCCARTESIANPOINT((0.,1000.));
#30=IFCALIGNMENT2DHORIZONTALSEGMENT($,$,$,#28);
#31=IFCRELCONTAINEDINSPATIALSTRUCTURE('11kFQ9Hu55kOD2ERHEAM$m',#2,$,$,(#21),#16);
ENDSEC;
END−ISO−10303−21;
```

Code 3.1: Simple example for a STEP physical file.

3.2 IfcAlignment

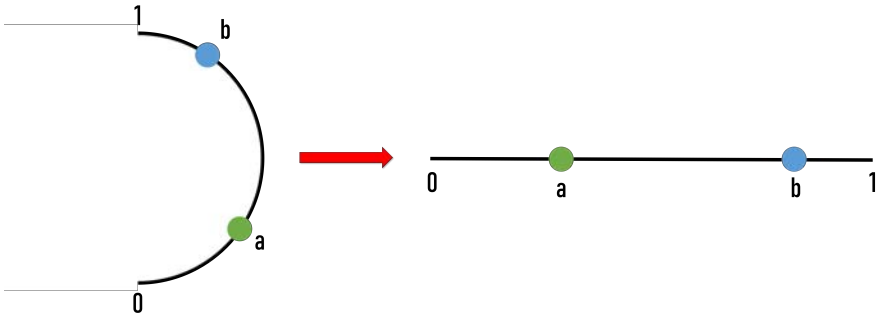
As introduced in Section 1.1 and Section 2.1, information from the large-scale to the small-scale is required to model a complete railway system, for example, the information about the surrounding environment, as well as topological network down to the tracks, signals, and switch systems. Therefore, several data exchange schemata, such as PlanPro and railML were developed to handle these specific sub-domains (Esser and Borrmann, 2019); furthermore, the most common data exchange format IFC has been extended to encompass the essential functions for railway systems. Since IFC 4.1, the entity IfcAlignment is available to define a reference system for linear construction work.

3.2.1 Linear Reference System

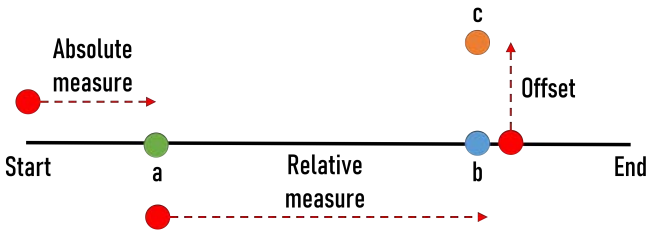
An LRS is a method to identify a specific location along a route using only a single coordinate, the distance measurement (Scarponcini, 2005). ISO 19148 Geographic information - Linear referencing and ISO 19133 Tracking and Navigation Standard explain how the LRS is implemented. The possibly complicated curve describing the route is unfolded into a single linear element, that is defined using the start point of the curve and the endpoint, given by the linear distance along the curve (see Figure 3.2a). For example in the case of a circle, the linear element is given by its circumference and every point on the circle can be referenced by a single distance measurement along the circumference. Any point on the curve is therefore converted from the complicated global 3D geometrical coordinates into a simple 1D local coordinate along the curve. Points along the linear element can be referenced by multiple methods; for example (see Figure 3.2b), an event can be located using the absolute measurement from the linear element start point or using a relative measurement from a referent, for example, a landmark, and a distance measure from this point. Furthermore, points not lying on the curve axis can be described using a referent and an offset distance away from the linear element. The advantage of an LRS is the simple expression for the location along extended linear elements, it is therefore suitable for a railway system to manage asset positions such as tracks, stations, and control systems.

3.2.2 Structure of IfcAlignment

IfcAlignment, a linear positioning element, belongs to the core layer and is inherited from IfcProduct (see Figure 3.1); the entity is the composition of other entities following the schema shown in Figure 3.3. The information of an alignment might be defined (1) in the X/Y plane of a Cartesian coordinate system accompanied with or without the elevation in



(a) Unfolding a circle to a linear element.



(b) Examples for an absolute reference, a relative reference, and an offset reference.

Figure 3.2: Sketches of an LRS.

the Z direction; (2) as a relative alignment based on another alignment; (3) and as a 3D alignment, either determined by combining a horizontal and a vertical alignment or extracted from geospatial data (buildingSMART International, 2018). To describe an LRS for railways, a set of horizontal alignments are considered in the thesis.

A basic horizontal alignment, IfcAlignment2DHorizontal, consists of one or more alignment segments connecting together by their start and endpoints. Each IfcAlignment2DHorizontalSegment segment can be defined by: a straight line, IfcLineSegment2D; an arc, IfcCircularArcSegment2D; or a transition curve, IfcTransitionCurveSegment2D. This horizontal alignment is assigned to the attribute of IfcAlignmentCurve, which becomes the axis of IfcAlignment. Finally, multiple IfcAlignments are grouped together to specify an LRS. An example of a geometrical model for IfcAlignment is made in Section 3.2.3.

3.2.3 Curve Representation in STEP

The **DATA** section in Code 3.1 represents a curve by one alignment composed of two separate alignment segments. The geographical reference system is not given in the example, thus the alignment is visualized in an arbitrary orientation as shown in Figure 3.4. Looking

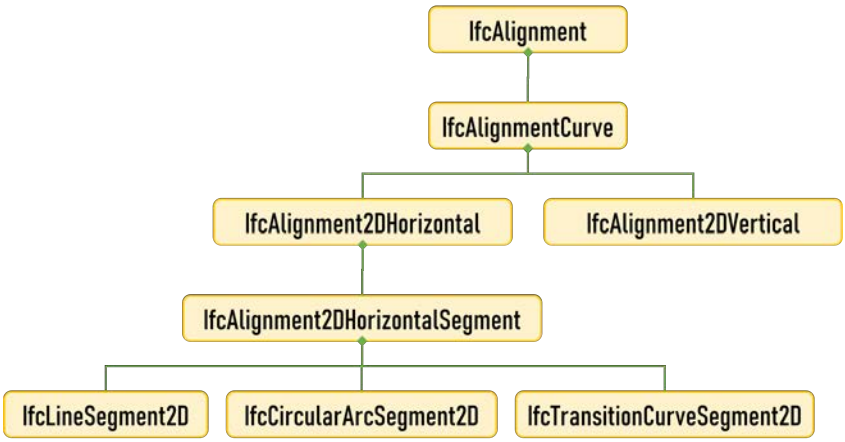


Figure 3.3: Alignment attributes in 2D (the lines represent the composition).

at IfcAlignment with STEP Id #21 in Line 28 in Code 3.1, it can be seen that the attribute Axis refers to the entity IfcAlignmentCurve with STEP Id #22 in Line 29, of which the attribute Horizontal refers to the entity IfcAlignment2DHorizontal with STEP Id #23 in Line 30. From the attribute Segments of IfcAlignment2DHorizontal, two IfcAlignment2DHorizontalSegment with STEP Id #27 in Line 34 and #30 in Line 37 are tracked. Then the basic segments, IfcLineSegment2D with STEP Id #25 in Line 32 and IfcCircularArcSegment2D with STEP Id #28 in Line 35, are found. All the geometry information, such as the Cartesian coordinates of start points, the segment start direction, and segment length, can be found in this entity level. The geometry definition for different segment types are introduced in detailed in Chapter 5.

```
#21=IFCALIGNMENT('0qpyZi7wPD58Gn0AbXm1IN',#2,'myCurve','myDescription',$,#12,$,#22,$);
#22=IFCALIGNMENTCURVE(#23,#24,$);
#23=IFCALIGNMENT2DHORIZONTAL($,(#27,#30));
#25=IFCLINESEGMENT2D(#26,1.5707963267948966,1000.);
#27=IFCALIGNMENT2DHORIZONTALSEGMENT($,$,$,#25);
#28=IFCCIRCULARARCSEGMENT2D(#29,1.5707963267948966,785.39816339744823,500.,.F.);
#30=IFCALIGNMENT2DHORIZONTALSEGMENT($,$,$,#28);
```

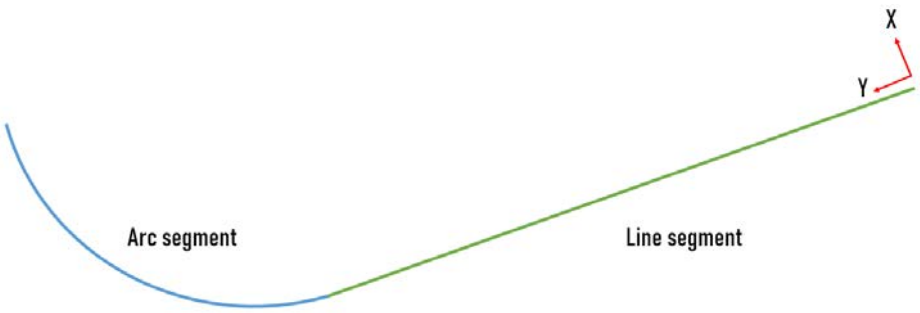


Figure 3.4: An example alignment in an arbitrary orientation.

3.3 Future Outlook

Currently, the research groups of bSI, Infrastructure Room and Railway Room, are working on several projects (buildingSMART International, 2020b). In addition to the published IfcAlignment, Infrastructure Room is continuously developing the common schema project to ensure the collaboration and interoperability among the ongoing projects related to civil infrastructures such as IFC Bridge, IFC Tunnel, and IFC Rail. In the meanwhile, Railway Room is focusing on the IFC Rail project and progressing on the development of sub-domains including track, energy, signaling, telecommunication, and technical services (buildingSMART International, 2020c).

Although BIM applications keep updating the certification with the latest IFC standard, it still takes time to completely support IfcAlignment. Besides, it is not possible for all applications to implement all domains, as software enterprises prefer to develop domain-specific applications. Therefore, enabling collaboration across standards through their overlap in scope is necessary for now and for the future. Some related research is, for example, the linking of InfraGML, CityGML, and IFC (Kumar et al., 2019); integrating the data model PlanPro for representing railway equipment components into IFC (Esser and Borrmann, 2019); as well as the linking of CityGML and IFC infrastructure (Vilgertshofer et al., 2017). This thesis also proposes a linking method between IFC and the operation-focused schema railML.

Chapter 4

RailTopoModel - Topological Model of railML

Chapter 4 introduces the operation-focused data schema railML and the novel topological model RTM. Section 4.1 introduces the background of railML and explains the structure of the data schema; then an example for railML and an example XML code are given. Section 4.2 introduces the topological model used in the latest railML and the motivation for its development; a simple comparison among existing topological data models; the classical representation of a railway network, as well as the representation using RTM.

4.1 Overview

RailML, an XML-based and open data exchange schema, aims to become an interface handling the data transfer from one application to another in the railway sector. RailML.org, originally founded by researchers from the German Fraunhofer Institute for Transportation Systems and Infrastructure in Dresden and the Institute of Transport Planning and Systems (IVT) of ETH Zürich, is developing railML for railway applications since 2001 (Nash et al., 2004). Today, the railML consortium includes certified software developers from Germany, Switzerland, and France, railway companies from all over Europe such as Deutsche Bahn, Österreichische Bundesbahn, and Trenitalia, as well as other research institutes from around the world.

The initial version of railML was released to exchange timetable data in 2005; the other sub-schemata such as rolling stock, and macroscopic infrastructure were introduced in the later versions of railML. Because the early railML schemata were built on a rather casually defined and incomplete topological basis that was not able to satisfy all railway data exchange requirements, many railways and European Train Control System (ETCS) suppliers

needed to extend the internal railML specifications themselves (Hlubuček, 2017; Seybold and Franke, 2013). To provide a well-defined basis for diverging use cases, a core topological model that can describe generic railway elements independent of any end purpose and process was demanded, thus RTM was introduced in 2013 and certified as an international standard IRS 30100 in 2016. RTM, as a component of railML, is available in the latest schema railML 3.1, published in 2019. Currently, some certified software packages are, for example, ControlGuide OCS for traffic management by SIEMENS; OpenTrack for capacity planning and timetabling by OpenTrack Railway Technology; and GPSinfradat for infrastructure survey by Bahnkonzept (railML.org, 2020).

4.1.1 Structure of railML Schema

In addition to the schema for general information, the current railML 3.1 schema can be divided into four sub-schemata for particular types of railway data: infrastructure, interlocking, rolling stock, and timetable (see Figure 4.1) (railML.org, 2019b). The infrastructure schema includes all fundamental elements that describe a railway network, namely elements for topological representations; geometrical representations; functional infrastructure such as platforms, signals, tracks, electrification, and switches; physical facilities; infrastructure visualizations; and infrastructure status. The interlocking schema provides elements for assets for interlocking, controllers, signal boxes, and specific infrastructure managers. The rolling stock schema contains the elements storing information about engines and wagon material. The final schema timetable permits the exchange of timetable data. Each schema is independent and can be combined by referring to the element identity. Furthermore, users only need to input their desired subset of attributes as the philosophy of railML is to keep as few required elements as possible and thus remain flexible.

RailML provides two approaches for data exchange; users can export data into railML and then import it into another application also supporting railML, or transfer data directly between applications through the means of interprocess communication (Nash et al., 2004; Ciszewski et al., 2017). The first approach requires a railML file and the format is introduced in the next section.

4.1.2 XML Schema and XML Format

As mentioned in Section 2.3, most data exchange schemata including railML are developed in the meta language XML as XML can describe both data and the data structure itself (Nash et al., 2004). The XML schema is defined by XSD, describing the structure and elements of XML as shown in Code 4.1. Similar to all XML-based data schema, the data structure is a hierarchical tree, and the sub-schemata are nested inside. The element <schema> in Line 2 is the root, with attributes describing the source of elements and data types used in the schema,

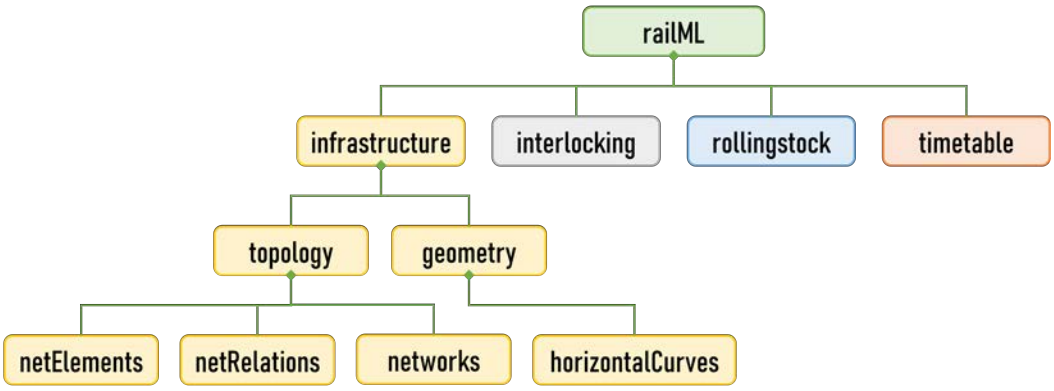


Figure 4.1: A subset of the RAILML 3.1 schema.

as well as the version of the schema. <schema> together with the element </schema> in Line 50 defines a body, and other elements located inside the body of another element are the children. From Line 13 to Line 40, it can be seen that the child elements, METADATA, COMMON, INFRASTRUCTURE, INTERLOCKING, ROLLINGSTOCK, and TIMETABLE are defined; the corresponding XSD files are included in this schema as shown from Line 3 to Line 7. The attributes' names, types, and other properties such as restrictions are defined within each element. For example, the element INFRASTRUCTURE in Line 20 has four attributes: the attribute NAME indicates that the element's name is INFRASTRUCTURE; the following attribute TYPE refers to the element type RAIL3:INFRASTRUCTURE; the last two attributes MINOCCURS and MAXOCCURS convey that this element is not necessary for creating a RAILML file and can only be defined once.

```
1  <?xml version="1.0" encoding="utf−8"?>
2  <xs:schema xmlns:rail3="https://www.railml.org/schemas/3.1" xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="https://www.railml.org/schemas/3.1" elementFormDefault="qualified" version="3.1">
3      <xs:include schemaLocation="timetable3.xsd"/>
4      <xs:include schemaLocation="rollingstock3.xsd"/>
5      <xs:include schemaLocation="interlocking3.xsd"/>
6      <xs:include schemaLocation="infrastructure3.xsd"/>
7      <xs:include schemaLocation="common3.xsd"/>
8      <xs:element name="railML" type="rail3:railML"/>
9      <xs:complexType name="railML">
10         <xs:annotation>
11             <xs:documentation>This is the root element of any railML file.</xs:documentation>
12         </xs:annotation>
13         <xs:all>
14             <xs:element name="metadata" type="rail3:Metadata" minOccurs="0" maxOccurs="1"/>
15             <xs:element name="common" type="rail3:Common" minOccurs="0" maxOccurs="1">
16                 <xs:annotation>
17                     <xs:documentation>root element for railML3 common model</xs:documentation>
18                 </xs:annotation>
19             </xs:element>
20             <xs:element name="infrastructure" type="rail3:Infrastructure" minOccurs="0" maxOccurs="1">
21                 <xs:annotation>
22                     <xs:documentation>root element for railML3 infrastructure model</xs:documentation>
23                 </xs:annotation>
24             </xs:element>
25             <xs:element name="interlocking" type="rail3:Interlocking" minOccurs="0" maxOccurs="1">
26                 <xs:annotation>
27                     <xs:documentation>root element for railML3 interlocking model</xs:documentation>
28                 </xs:annotation>
29             </xs:element>
30             <xs:element name="rollingstock" type="rail3:Rollingstock" minOccurs="0" maxOccurs="1">
31                 <xs:annotation>
32                     <xs:documentation>root element for railML3 rollingstock model</xs:documentation>
33                 </xs:annotation>
34             </xs:element>
35             <xs:element name="timetable" type="rail3:Timetable" minOccurs="0" maxOccurs="1">
36                 <xs:annotation>
37                     <xs:documentation>root element for railML3 timetable model</xs:documentation>
38                 </xs:annotation>
39             </xs:element>
40         </xs:all>
41         <xs:attributeGroup ref="rail3:aRailML"/>
42     </xs:complexType>
43     <xs:attributeGroup name="aRailML">
44         <xs:attribute name="version" type="xs:string" use="required">
45             <xs:annotation>
46                 <xs:documentation>the supported railML version should be declared for software compatibility
                       reasons, valid for all subschemas, don't mix railML versions between subschemas in one XML
                       file</xs:documentation>
47             </xs:annotation>
48         </xs:attribute>
49     </xs:attributeGroup>
50 </xs:schema>
```

Code 4.1: The root railML schema in XSD (railML.org, 2019b).

An example XML code is shown in Code 4.2. The element TOPOLOGY contains one child element NETELEMENTS. NETELEMENTS is a container of a set of elements NETELEMENT. Each NETELEMENT has two attributes, ID and LENGTH; the ID and LENGTH of the first NETELEMENT are *ne_a01* and 500.0 respectively.

```
1 <topology>
2     <netElements>
3         <netElement id="ne_a01" length="500.0"/>
4         <netElement id="ne_a02" length="500.0"/>
5         <netElement id="ne_a03" length="200.0"/>
6         <netElement id="ne_b01" length="500.0"/>
7     </netElements>
8 </topology>
```

Code 4.2: A subset of a RAILML file about NETELEMENTS in the sub-schema TOPOLOGY of INFRASTRUCTURE.

4.2 RailTopoModel

Train operation is a complex task, considering the intricate timetabling, dense traffic, and the need for continuous maintenance. Many data models were developed to handle the diverging topology needs respectively, however, there is no monolithic model that can manage topological descriptions on different levels. Each model is aimed towards different sub-domains of train operation and thus is suited only towards a specific purpose. Moreover, an exhaustive description of a railway network is counter-productive to scheduling optimization problems (Gély et al., 2010). Studies to build a generic data model for multi-scale descriptions started in 2010, eventually culminating in RTM (Gély et al., 2010). In addition, the INTERNATIONAL UNION OF RAILWAYS (UIC) launched a feasibility study to evaluate the existing topological models and proposed RTM for inclusion in RAILML in 2013.

4.2.1 Existing Topological Models

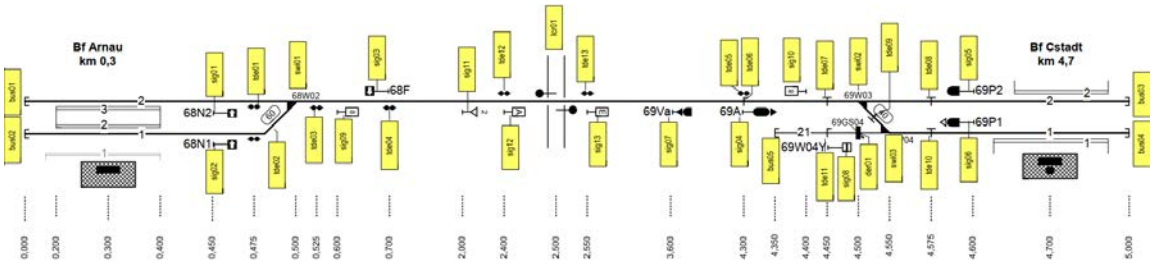
Several models were created to describe a fundamental topological model for railway business use cases and adopted by national railways and EU directives, for example, REGISTER OF INFRASTRUCTURE (RINF), INSPIRE, ARIANE, INFRANET, BANEDATA, RINM, and PLANPRO (see Table 4.1) (Seybold and Franke, 2013; Wunsch and Jaekel, 2017). These topological models are tailored towards national requirements, resulting in increased communication difficulty for railway networks crossing country borders. Furthermore, 95% of each model's features are compatible, as the basic concept of a rail network is similar in every country (Seybold and Franke, 2013). Therefore, a core model with an extension mechanism is the optimal solution.

Topological model	Organization	Country	Format	Purpose	Supports multi-scale
RINF	EU directive	-	XML	Description of network	Yes
INSPIRE	EU Joint Research Center	-	GIS based Geoportal	Description of environmental related themes	Yes; interpreted by nodes and links
ARIANE	Réseau Ferré de France	France	TEXT, JSON, XML	Description of network	Yes
InfraNet	Infrabel	Belgium	XML	Description of network	Yes; detailed grape for each node
Banedata	Jernbaneverket	Norway	railML, CSV, XLS	Description of network, maintenance of infrastructure objects	No; micro only
RINM	Network Rail	United Kingdom	XML	Description of network	Yes
PlanPro	Deutsche Bahn AG	Germany	XML	Description of network for signaling and safety equipment	Yes; Mostly micro with some nano

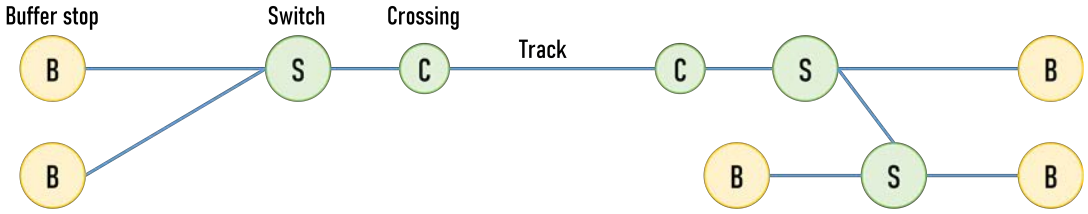
Table 4.1: Sample topological models for railways (Seybold and Franke, 2013; Wunsch and Jaekel, 2017).

4.2.2 Classical Topological Representation

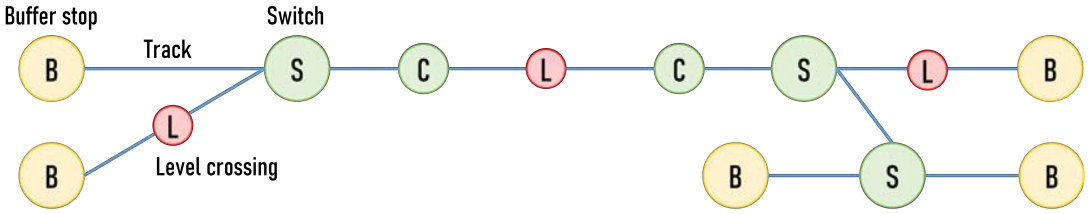
The classical representation of a railway is an undirected graph composed of nodes and edges, where nodes refer to special points, e.g. switches, buffer stops, and operational points, and edges refer to connecting track routes. An example track network with functional infrastructure elements including switches, signals, train detection elements, and operational points for a 5 km single-track bordered by two operational points is shown in Figure 4.2a; its classical topological representations in two levels are shown in Figure 4.2b and in Figure 4.2c respectively. Figure 4.2b shows that the track layout is divided into several blocks, and the tracks are connected by switches and buffer stops in the rough detail level, while Figure 4.2c shows that the track fragments are connected not only by switches and buffer stops but also level crossing systems in the more detailed level. It is known that the topological representation is dynamic for different sizes and precision requirements.



(a) A track network with functional infrastructure elements (railML.org, 2019a).



(b) The classical topological representation.



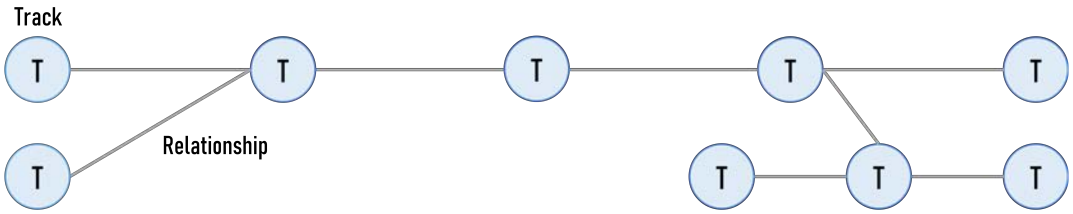
(c) The classical topological representation in another detail level (The level crossing nodes are added as an example).

Figure 4.2: A simple example in railML 3.1 regarding a 5 km single-track bordered by two operational points.

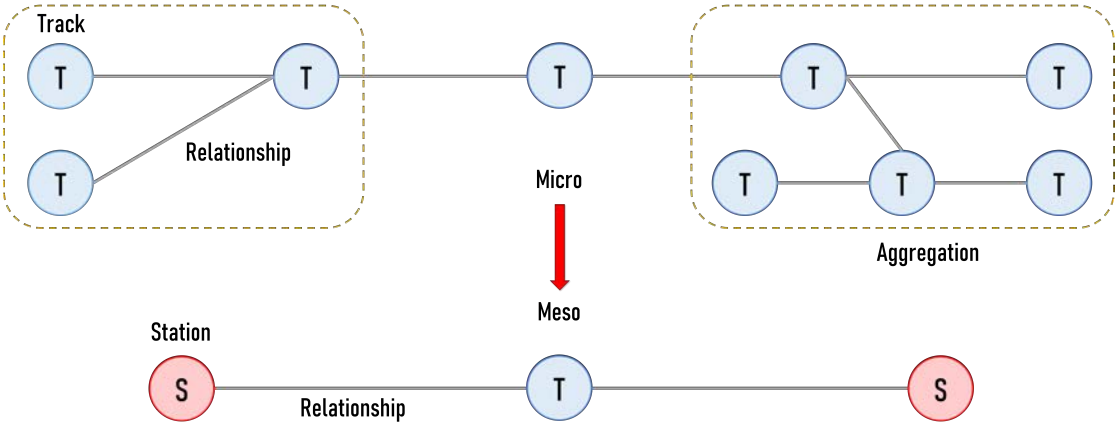
Most of the topological models listed in Table 4.1 adopt the classical topological model and support multi-scale representation (Seybold and Franke, 2013; Wunsch and Jaekel, 2017). However, the classical topological model does not follow the coherence rule, which means elements of the same nature are represented differently regarding the description level and thus does not allow an easy implementation of an object-oriented model (Gély et al., 2010). For example, the nodes of the level crossings in the classical topological model in Figure 4.2c are not shown in the rougher representation in Figure 4.2b, besides, the tracks connecting the level crossings became a single track; nevertheless, both Figure 4.2b and Figure 4.2c represent the same railway network. In the graph theory, nodes represent only the resources and edges represent only the relationships, which is not the case for a classical topological model for railways (railML.org, 2016a). For this reason, RTM was introduced to provide a core model that can manage data in any LoD.

4.2.3 Topological Representation in RailTopoModel

RTM is a topological model based on a connexity graph. Different to the classical topological model, all resources including linear elements such as tracks and non-linear elements such as functional infrastructure components are represented as nodes and the relationships between resources are represented as edges. The topology of Figure 4.2b can then be redrawn as shown in Figure 4.3a. This concept of representation allows all elements to be aggregated or disaggregated according to their specific purpose (see Figure 4.3b). Moreover, the relationships still retain the information about connectivity and navigability between elements.



(a) At *Micro* level.



(b) At *Meso* level.

Figure 4.3: The topological model of Figure 4.2a using RTM.

On a railway network, three types of objects can exist: point objects, e.g. signals, boundaries, and balises; linear objects, e.g. track and speed profile; and areal objects, e.g. stations, bridges, and catenary zones (Seybold and Franke, 2013). According to the needs, they can be further used in different levels. In the standard IRS 30100, three levels of detail are defined (railML.org, 2016a): (1) the level *Micro* contains detailed information at track level including tracks, switches, and buffer stops; (2) the level *Meso* focuses on functional information at track level including tracks and operational points; (3) the final level *Macro* describes a network at a regional level including major operational points and the track collections.

The RTM representation in Unified Modeling Language (UML) consists of four packages including Base, Topology, Positioning systems, and Net Entities; the simplified UML

diagram is shown in Figure 4.4. The class NETWORK is composed of the classes NETWORKRESOURCE and LEVELNETWORK. LEVELNETWORK describes a certain view of a network, namely the LoDs; The basic units to construct a topological model are NETELEMENT and NETRELATION, which are derived from the class NETWORKRESOURCE. NETELEMENT defines the basic elements for nodes in the topological representation and contains the aggregation of element parts defined by the class ELEMENTCOLLECTION. When a NETELEMENT belongs to the *Meso* level, then the NETELEMENT sets of ELEMENTCOLLECTION belong to the *Micro* level. Each pair of NETELEMENT is associated with a NETRELATION, which embodies edges defining the connexity relation between NETELEMENTS. Furthermore, each NETELEMENT is located by ASSOCIATEDPOSITIONINGSYSTEM. The last class NET ENTITIES defines physical or immaterial objects, e.g. switches, buffer stops, and speed limits; the instances can be located by using NETELEMENT as an LRS or by a geometrical point. At least one NETWORK at one level needs to be defined for a project, furthermore, NETELEMENT and NETRELATION from different levels need to be aggregated following a vertical relationship.

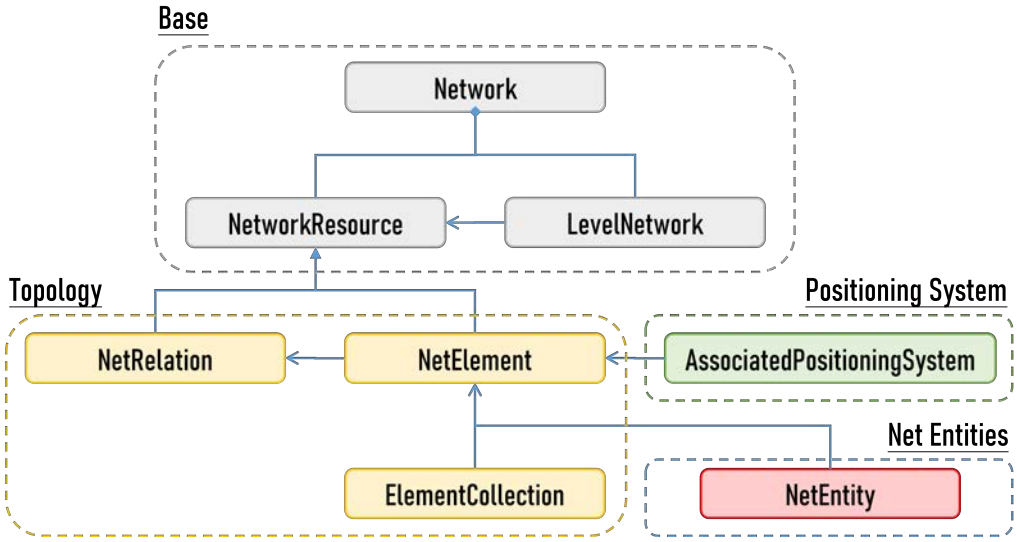


Figure 4.4: The simplified UML diagram with respect to the topology package in RTM (railML.org, 2016a).

4.3 Summary

RAILML 3.1, which adopted RTM, provides an integrated solution for railways. The topological model based on RTM offers a generic fundamental topology allowing users to extend it for specific purposes. As the RAILML schema focuses on railway operational purposes, detailed geometrical information is not always necessary; however, geometrical information is required for maintenance. Besides, although one NETWORK can be derived from another NETWORK, extra transition steps might be needed to fill the gaps caused by different LoDs. To ensure

data consistency, not just among data schemata for the operation phase but also among data schemata for design, construction, and maintenance phases, a solution for connecting both types of data schemata, i.e. IFC and RAILML, is proposed and prototyped in Chapter 5 and tested in case studies in Chapter 6.

Chapter 5

Solution Proposal and Implementation

After researching the concepts of geometrical and topological representations in both IFC and RAILML schemata, it can be seen that a data mapping between these two schemata without data-loss is not easy as both are made for diverging scopes. To propose an optimal data exchange approach, Section 5.1 discusses possible sources for data inconsistencies, identifies entities and elements from the schemata that might lead to such inconsistencies, and introduces ways to mitigate them. Then, Section 5.2 explains the implementation concerning the bi-directional mapping between geometrical representations and topological representations based on the results of the problem discussion. The functions of the developed tool are then introduced in Section 5.3.

5.1 Problem Discussion

Data exchange schemata are used to enhance the interoperability among applications, yet they do not guarantee data conversion without data loss as discussed in Section 2.4. In particular, incomplete coverage of a data schema is the largest source of data inconsistencies. Incomplete coverage is not just a result of a limited or non-existing intersection of two entity sets of two schemata, but also a result of ambiguous semantic definitions of entities despite them having the same or functionally dependent information; another reason might be biased due to different input precision of an entity value. These problems can decrease the efficiency of automatic data exchange. In the next section, several data inconsistency scenarios are discussed.

5.1.1 Data Inconsistency Scenarios

To give examples for the problem sources mentioned above, two sets are defined; A and B represent two different data schemata, and each set contains a set of elements representing the names of entities.

Same Elements

Case 1 Set A and set B include five element each, and four elements of set B, $\{x_1, x_2, x_3, x_4\}$, are the same as those of set A. As the four elements are identically named, the elements' values can be exchanged.

$$\begin{aligned} A &\equiv \{x_1, x_2, x_3, x_4, x_5\} \\ B &\equiv \{x_1, x_2, x_3, x_4, x_6\} \\ A \cap B &\equiv \{x_1, x_2, x_3, x_4\} \end{aligned}$$

Case 2 Set A keeps the same elements as in Case 1, and set B now has only common element x_1 shared with set A. The intersection range of the two sets is now smaller than in Case 1.

$$\begin{aligned} A &\equiv \{x_1, x_2, x_3, x_4, x_5\} \\ B &\equiv \{x_1, x_6, x_7, x_8\} \\ A \cap B &\equiv \{x_1\} \end{aligned}$$

Compared to Case 1, which loses only one element x_5, Case 2 loses four elements $\{x_2, x_3, x_4, x_5\}$ when set A is converted into set B. In contrast, when set B is converted into set A, Case 1 also loses one element x_6 while Case 2 loses elements $\{x_6, x_7, x_8\}$. It can be observed that two sets with a smaller intersection can be converted worse than those with a larger intersection regardless of the conversion's direction.

Disjoint Element Sets

Assume, the elements of set A are distinguished from the elements of set B; these two sets are called disjoint. In this case, no elements are exchangeable.

$$\begin{aligned} A &\equiv \{x_1, x_2, x_3\} \\ B &\equiv \{y_1, y_2, y_3\} \qquad y_i \neq x_i \qquad i = 1, 2, 3 \\ A \cap B &\equiv \varnothing \end{aligned}$$

Synonymous Elements

The element sets from set A and set B are completely different, so the intersection of the two sets contain zero elements; however, this assumption is false. Although the element x_2 of the set A is different from the element y_2 of the set B, they are semantically the same. Hence, they should be exchangeable.

$$\begin{aligned} A &\equiv \{x_1, x_2, x_3\} \\ B &\equiv \{y_1, y_2, y_3\} \qquad y_2 = x_2 \\ A \cap B &\equiv \varnothing \end{aligned}$$

Homonymous Elements

The element sets from set A and set B have one element sharing the same name, so it is assumed that x_2 is exchangeable; this is however false. The element x_2 of set A refers to a different property than the element x_2 of set B, thus it is not exchangeable.

$$\begin{aligned} A &\equiv \{x_1, x_2, x_3\} \\ B &\equiv \{y_1, x_2, y_3\} \qquad x_2 \in B \neq x_2 \in A \\ A \cap B &\equiv \{x_2\} \end{aligned}$$

Functionally Dependent Elements

Again, there can be a false exchange of data if the exchange is purely based on the comparison of elements in two sets. The intersection of set A and set B is an empty collection but their elements can actually be exchanged. This situation can happen when two sets have a different definition for the same object. Nevertheless, the conversion is not necessarily bi-directional. As shown below, the element y_1 of set B can be determined by the product of elements x_1 and x_2 of set A; this however also means that a conversion without loss of data is only possible from set A to set B but not vice versa.

$$\begin{aligned} A &\equiv \{x_1, x_2\} \\ B &\equiv \{y_1\} \qquad y_1 = x_1 \cdot x_2 \\ A \cap B &\equiv \varnothing \end{aligned}$$

Different Accuracy of Number Types

The precision of a number not only depends on the actual input given but also on the type of numerical representation used to store the number. Assume a variable is defined in either the FLOAT, DOUBLE, or DECIMAL types, which can store a different amount of digits from the least to the most, respectively. When a length defined in DOUBLE is converted into a FLOAT, the converted variable holds less digits and thus loses precision; correspondingly, a variable defined in DECIMAL loses even more digits. While the numerical precision is typically not an issue in small-scale structures, it can become significant in railway tracks that may span hundreds of kilometres. Small rounding errors in the dimensioning of each segment might compound into measurable differences along the length of the system. Especially if the location of segments is functionally dependent on the preceding chain of segments.

5.1.2 Comparisons of IfcAlignment and RailTopoModel

Having introducing IFCALIGNMENT in Section 3.2 and RTM in Section 4.2, it can be observed that their intersection so far only contains the entities or elements necessary to describe a railway track layout using an LRS. A railway track layout can be built by three basic geometrical elements, namely lines, arcs, and transition curves; transition curves are used to smoothly connect curves of different radius (Jochim and Lademann, 2017). To investigate how well a railway track layout can be created and how much information can be exchanged, a comparison between geometrical and topological representations is made according to the data inconsistency scenarios introduced in Section 5.1.1. IFCALIGNMENT is used to build the geometrical representation, while the latest RAILML provides not just the element sets for a topological representation, namely RTM, but also those for a geometrical representation. In the following sections, the sub-schema for the topological representation of RAILML is called RTM and the sub-schema for the geometrical representation is called GEOMETRY.

IfcAlignment

IFCALIGNMENT provides three basic units, IFCLINESEGMENT2D, IFCCIRCULARARCSEGMENT2D, and IFCTRANSITIONCURVESEGMENT2D, to compose a railway track layout. IFCLINESEGMENT2D and IFCCIRCULARARCSEGMENT2D support a line segment and an arc segment respectively (see Figure 5.1), whereas IFCTRANSITIONCURVESEGMENT2D supports transition curves that are parametrically defined, such as *Biquadratic parabola*, *Bloss curve*, *Clothoid curve*, *Cosine curve*, *Cubic parabola*, and *Sine curve.* A comparison of the attributes of IFCLINESEGMENT2D, IFCCIRCULARARCSEGMENT2D, and IFCTRANSITIONCURVESEGMENT2D is shown in Table 5.1.

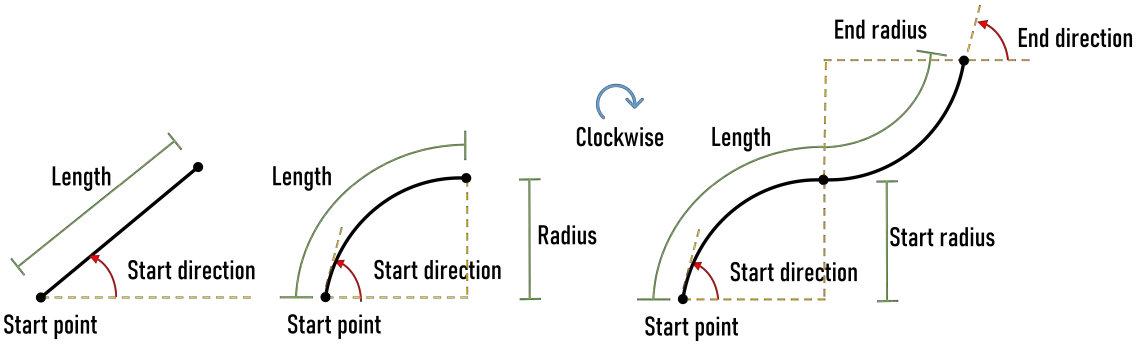


Figure 5.1: A sketch of a straight line, an arc, and a transition curve as defined in IFC.

	Attributes (Name)	IfcLine-Segment2D	IfcCircularArc-Segment2D	IfcTransitionCurve-Segment2D
Start	Point (StartPoint)	✓	✓	✓
	Direction (StartDirection)	✓	✓	✓
	Radius (Radius/ StartRadius)		✓	✓
	Counter-clockwise? (IsCCW/ IsStartRadiusCCW)		✓	✓
End	Point			
	Direction			
	Radius (EndRadius)			✓
	Counter-clockwise? (IsEndRadiusCCW)			✓
Length (SegmentLength)		✓	✓	✓
Supports transition curves (TransitionCurveType)				✓

Table 5.1: A comparison of the attributes of the IFC entities for lines, arcs, and transition curves.

RailTopoModel and Geometry

RTM and GEOMETRY contain the element sets, NETELEMENT, NETRELATION, and HORIZONTALCURVE; however, the definitions available in GEOMETRY are rather rudimentary as RAILML is geared towards the topological representation preferred for operational purposes. Table 5.2 indicates how well the RTM and GEOMETRY element sets can define a curve in RAILML. It can be seen that for a curve definition, NETELEMENT has three corresponding parameters that can only define a straight line without directional information; on the other hand, HORIZONTALCURVE has seven corresponding parameters that can describe lines, arcs, and transition curves such as *Sinusoids*, *Doucines*, *Wiener curves*, *Bloss curves*, *Cubic parabolas*, *Cosinusoids* and *Clothoids*. Nevertheless, HORIZONTALCURVE only explicitly requires the curve type and does not necessarily require related parameters such as coordinates, segment length, and directions.

RTM is used to build a topological railway network and describes the geometrical elements as nodes and the relationships among nodes as edges. To create a network, at least one RTM in any level is required. IRS 30100 (railML.org, 2016a) recommends three LoDs, *Micro*, *Meso*, and *Macro*, but is flexible and allows users to build a network in any LoD. Typically, it is preferred to start with a network in the *Micro* level, providing more detailed geometrical information compared to the other LoDs (see Figure 4.3b). Hence, if a network is initially built on a rough level and needs to be transferred to a geometrical representation, then complementary data input is required.

Comparisons

IFCALIGNMENT aims to construct a railway track layout for an LRS, while RTM aims to represent a railway network. Since the targets are biased, data loss after any transfer is to be expected. Comparing IFCALIGNMENT and the element sets for GEOMETRY of RAILML, it can be seen that IFCALIGNMENT defines every curve type explicitly whereas GEOMETRY of RAILML only defines them implicitly. That means, for each curve type, essential parameters are required in IFCALIGNMENT yet not mandatory in GEOMETRY of RAILML. Additionally the element set of GEOMETRY does not contain all necessary parameters needed to define a transition curve in IFCALIGNMENT; for example, ENDRADIUS and ISENDRADIUSCCW are not available in GEOMETRY and thus can not connect different curve radii. This complicates the conversion from RAILML to IFCALIGNMENT and so the conversion of transition curves is done only in the direction of IFCALIGNMENT to RAILML.

Additionally, there is even more missing information when comparing IFCALIGNMENT and the element sets for RTM. The topological information of RTM can not reflect a geometrical

Purposes	Attributes (Name)	RTM (NetElement, NetRelation)	Geometry (HorizontalCurve)
Railway track layout	Start Point (geometryCoordinateBegin)	✓	✓
	Endpoint (geometryCoordinateEnd)	✓	✓
	Type (curveType)		✓
	Direction (azimuth)		✓
	Direction Change (deltaAzimuth)		✓
	Radius (radius)		✓
	Length (length)	✓	✓
Network	Relationship among elements (relation)	✓	

Table 5.2: A comparison between the element sets of RTM and Geometry of railML (railML.org, 2019c).

model due to the lack of curve information. Similarly, a rough RTM is not enough to build a detailed RTM. On the other hand, the RTM stores the relationship data, which is missing in IfcAlignment.

5.1.3 Limitations of IfcAlignment and railML

Table 5.3 and Table 5.4 summarize the comparison results between IfcAlignment and Geometry as well as the RTM of railML in both directions regarding the different data inconsistency scenarios. Additionally, both IfcAlignment and RTM are also compared regarding the support of varying LoDs.

IfcAlignment → railML

Table 5.3 shows which information of railML might not be well interpreted by IfcAlignment. It can be seen that IfcAlignment lacks entities to describe the endpoint coordinates of a curve,

geometryCoordinateEnd, and the relationships between curves, relation. Although the endpoint coordinates can be easily determined by the parameters of the start point, the information relation is not stored at all in IfcAlignment and thus needs to be created. Other elements, i.e. geometryCoordinateBegin and length, are synonymous with the entities, StartPoint and SegmentLength, provided by IfcAlignment; nevertheless, some numerical length precision might be lost due to the different types of length and SegmentLength, *decimal* and *double.*

Alternatively, the geometrical information from IfcAlignment can be stored in Geometry of railML. Compared to RTM, Geometry contains relatively complete elements; all elements can be determined by the entities of IfcAlignment. There are only two elements that need special consideration, azimuth and deltaAzimuth, describing the angle of a segment; they are defined to see the north direction as 0°, the east as 90°, the south as 180°, and the west as 270°. Additionally, more elements exchanged from IfcAlignment to Geometry have precision differences, i.e. azimuth, deltaAzimuth, radius, and length.

	IFC	
Elements	**IfcAlignment vs RTM of railML**	**IfcAlignment vs Geometry of railML**
Same elements		radius
Disjoint element sets	geometryCoordinateEnd, relation	geometryCoordinateEnd, deltaAzimuth
Synonymous elements	geometryCoordinateBegin, length	geometryCoordinateBegin, Type, azimuth, length
Homonymous elements		
Functionally dependent elements	geometryCoordinateEnd	geometryCoordinateEnd, azimuth, deltaAzimuth
Different accuracy of number types	length	azimuth, deltaAzimuth, radius, length

Table 5.3: A comparison of IfcAlignment → railML regarding different data inconsistency scenarios.

railML → IfcAlignment

Compared to the scenario *Disjoint element sets* in IfcAlignment → railML, there are obviously more entities that need to be completed during the conversion from RTM of railML to IfcAlignment as shown in the scenario *Disjoint element sets* in Table 5.4. Because none of them can be determined from the existing elements of RTM, it is necessary to let users enter them explicitly. On the other hand, there are fewer entities in the scenario *Disjoint element set* and more entities in the scenario *Synonymous elements* from Geometry of railML to IfcAlignment. As discussed in Section 5.1.2, the sub-schema of Geometry is looser than IfcAlignment because Geometry does not define a curve explicitly. The curve data, especially for transition curves, consequently requires extra calculation and user-input values, such as radius and directional information. In translating railML to IfcAlignment it can also be expected to lose accuracy due to the different number types defined in the varying tools or user input.

	railML	
Entities	**RTM vs IfcAlignment**	**Geometry vs IfcAlignment**
Same elements		Radius
Disjoint element sets	StartDirection, Radius/ StartRadius, IsCCW/ IsStartRadiusCCW, EndRadius, IsEndRadiusCCW, TransitionCurveType	IsCCW/ IsStartRadiusCCW, EndRadius, IsEndRadiusCCW
Synonymous elements	StartPoint, SegmentLength	StartPoint, StartDirection, StartRadius, SegmentLength, TransitionCurveType,
Homonymous elements		
Functionally dependent elements		StartDirection
Different accuracy of number types	Radius/ StartRadius, EndRadius, SegmentLength	StartDirection, Radius, EndRadius, SegmentLength

Table 5.4: A comparison of railML → IfcAlignment regarding different data inconsistency scenarios.

Support of LoDs

RTM supports multiple LoDs as all objects of the same nature, including routes and operational points, are defined as a NetElement; on the other hand, the relationships are stored in a separate element NetRelation. Since all objects have the same nature, they can be aggregated and disaggregated without losing information about their relationship. In contrast, IfcAlignment is used to describe a detailed geometrical curve that can be composed of possibly many curve segments IfcAlignment2DHorizontalSegment. In general, IfcAlignment can not represent multiple LoDs. The curve segments can, of course, be seen as a set of curves in a more detailed level; nevertheless, the segments can not exist independently as the entity is in the resource layer. Furthermore, IfcAlignment can only represent one single continuous curve. Table 5.5 shows a possible LoD linking between RTM and IfcAlignment; the feasibility is investigated later in Section 5.2.5. In the prototype implementation, IfcAlignment is implemented in the *Micro* level and IfcAlignment2DHorizontalSegment in the *Nano* level.

RTM	**IfcAlignment**	
Nano	-	IfcAlignment2D-HorizontalSegment
Micro	IfcAlignment2D-HorizontalSegment	IfcAlignment
Meso	IfcAlignment	-
Macro	-	-

Table 5.5: Possible LoD linkings between RTM and IfcAlignment.

Summary

A summary of the comparison between the sub-schemata IfcAlignment, RTM, and Geometry is given below:

- There are almost no identical elements because they are guided by diverging standards and therefore define the same objects differently.

- The number of disjoint element sets depends on whether a conversion is between the same type of representations; for example, the conversion between IfcAlignment and Geometry has less disjoint element sets than between IfcAlignment and RTM.
- In most cases, the elements and entities are synonymous and dependent. Thus, the elements mostly just need to be recalculated. For example, both startDirection and azimuth describe the direction of a curve, yet the former is defined in the mathematical polar coordinate system with $\alpha = 0°$ in the east direction, while the latter is defined with $\alpha = 0°$ in the true north direction. The general unit of startDirection is *radians*, counting angles starting from a horizontal line in the counterclockwise direction; on the other hand, the unit of azimuth is degrees, counting angles from the north in the clockwise direction.
- From both directional comparisons, it can be observed that there are no homonyms.
- Elements with different number types such as SegmentLength and length can influence precision.
- The entity IfcAlignment is not designed to support multiple LoDs.

5.1.4 Solution Proposal

Having compared the sub-schemata of IFC and railML as well as having discussed the sources of data inconsistency, it can be understood that data loss can happen in a naive conversion between IfcAlignment and RTM and Geometry of railML. Therefore, this thesis proposes a prototype tool to link the topological representation, RTM of railML, to the geometrical representation, IfcAlignment of IFC; and to link geometrical representations, IfcAlignment of IFC and Geometry of railML. The proposed solution is to interpret the data between both schemata in a way that minimizes data inconsistency between geometrical and topological representations.

An intermediate interface is introduced to pre-process the semantic problems for bi-directional conversions, given that most of the possible data inconsistencies originate from synonymous and functionally dependent elements, especially in the direction of topological to geometrical representation. Furthermore, missing data caused by disjoint element sets are indicated in the tool, allowing users to supplement the necessary information themselves. Additionally, the tool can visualize the topological representation, allowing users to confirm a correct network.

5.2 Implementation

In this section, the proposed solution is implemented in a prototype tool that offers the full conversion of two types of curves: lines and arcs, and a partial conversion of transition curves between IFC and railML. The implementation is introduced separately for each important function in the tool.

5.2.1 Technical Framework

Before starting the discussion of the implementation, the technical framework of the solution including the tools and schema libraries is introduced in this section.

Programming Language

The .NET language C# is chosen for its object-oriented principle, interfaces to XML libraries, easy querying of data sets through Language-Integrated Query (LINQ), and ease of application creation for a prototype using Windows Presentation Foundation (WPF).

xBIM Toolkit

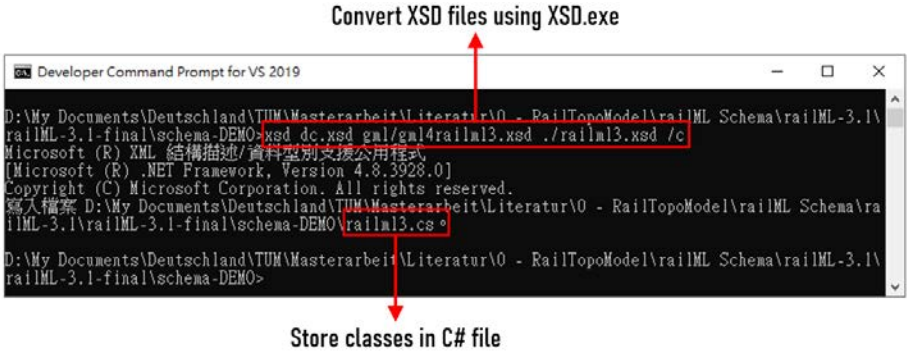
The xBIM toolkit is a .NET open-source software development toolkit that supports IFC, allowing developers to read, create, query, and modify an IFC file (Lockley, Benghi, and Černý, Lockley et al.).

Neo4j

Neo4j is a native graph database, which stores data as nodes and directly connects them by their relationships rather than storing data in a table and finding the relationships a posteriori using the Id as in a traditional database (Neo4j, Inc., 2020). The Neo4j platform is used as a visualization aid for the representation of topological information. A community driver Neo4jClient for C# is installed to interface with the language Cypher used in Neo4j.

Translation of XSD Schema to Object-Oriented Classes

As shown in Figure 5.2, XML Schema Definition Tool (XSD.exe) provided by Visual Studio is used to generate C# classes from the XSD files of railML (see Figure 4.1). The C# file is later converted into a Dynamic Link Library (DLL) used as a referencing library for railML.



(a) XSD.exe command prompt.

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("xsd", "4.8.3928.0")]
[System.SerializableAttribute()]
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Xml.Serialization.XmlTypeAttribute(Namespace="https://www.railml.org/schemas/3.1")]
2 references
public partial class Topology {

    private NetElement[] netElementsField;

    private NetRelation[] netRelationsField;

    private Network[] networksField;
```

(b) A sample class generated by XSD.exe.

Figure 5.2: Generating C# class files using XSD.exe.

5.2.2 Code Structure

Figure 5.3 gives a schematic overview of the code structure. The code structure can be divided into five parts: Graphical User Interface (GUI), Geo2Topo, Topo2Geo, Topo&Geo, and Visualization; inside each part, only the most important functionalities for processing and preparing data are shown. At the front of the tool, a simple GUI is available to visualize geometrical and topological data in tables, to input missing information needed for curve reconstruction, and to convert data files bi-directionally. Depending on the conversion type, the data either goes to Geo2Topo or Topo2Geo. Only a short overview of the code structure is given in this section; the detailed introductions of each part are given from Section 5.2.3 to Section 5.2.7.

Geo2Topo, one of the conversion types, transfers IFC to railML files. Once an IFC file is selected, the general project and alignment-related entities are retrieved. Then, a pre-processor uses the back-end Input/Output (I/O) to store data in a temporary network and to determine the relationships among alignments. After this stage, users can visualize the network in the external platform Neo4j, which is connected via the developed interface. Finally, an IFC file is created. This part still assigns preliminary random values for the navigability

between alignments, as the GUI does not yet allow full interaction with Geo2Topo. Therefore, the arrows only point in one direction.

Another conversion direction is from railML files to IFC files provided by Topo2Geo. Similar to the first stage in Geo2Topo, general and topological information is retrieved from the data file. The network can already be visualized in this stage as the topological representation a priori contains sufficient data. It is expected that there will be missing information after the conversion from a topological representation to a geometrical representation; therefore, the pre-processor uses front-end I/O to complete the geometrical data. If a railML file includes the element sets Geometry, then a geometrical representation can be generated directly without data supplementation. At the end, an IFC file is output that can be used further in a BIM application.

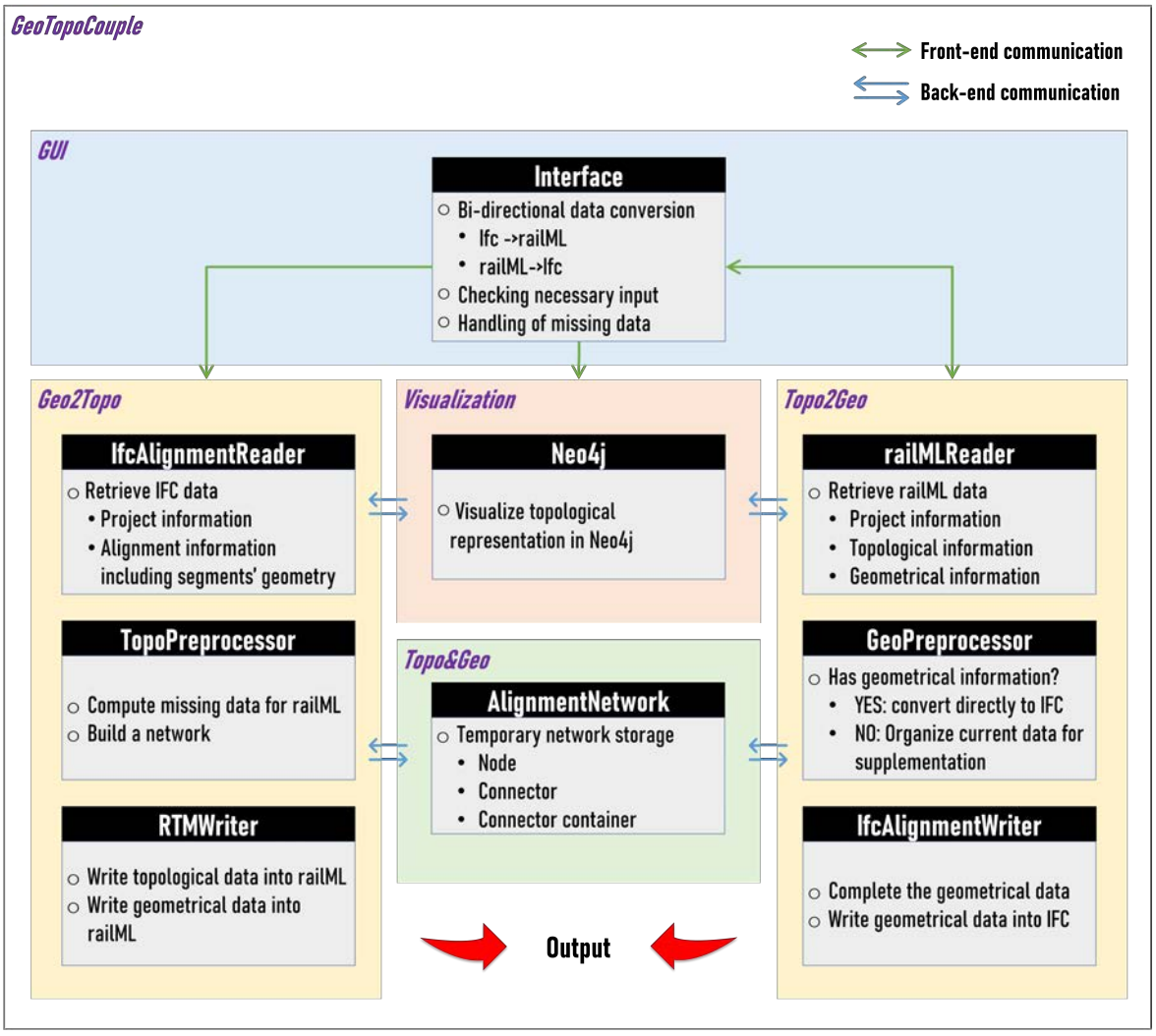


Figure 5.3: Schematic overview of the code structure.

5.2.3 Geo2Topo

Geo2Topo is used to convert an IFC file to a railML file, specifically, to allow consistent data transfer from IfcAlignment to RTM. Table 5.3 indicates that IfcAlignment lacks information on two elements, geometryCoordinateEnd and relation, and has indirect information related to geometryCoordinateBegin and length. geometryCoordinateEnd can be determined from the existing entities of IfcAlignment, while relation can only be found by comparing the geometrical information of existing IfcAlignment. On the other hand, every missing information for the elements of Geometry can be determined easily.

Geometrical Information at the Endpoint

The basic units of IfcAlignment are IfcLineSegment2D, IfcCircularArcSegment2D, and IfcTransitionCurveSegment2D. These three units store the non-redundant geometrical information, such as the coordinate of the start point, direction from the start point, segment length, and curve parameters for arcs and transition curves; the other geometrical information required for Geometry of railML, such as the coordinate of the endpoint and direction at the endpoint, can be calculated from the geometrical information at the start point. Geometry of railML does not define the curve types explicitly, thus the geometrical information is calculated according to the curve type defined in IFC and then assigned to each parameter in Geometry of railML. Length for an alignment is the summation of the individual segment lengths. The other parameters for lines, arcs and transition curves are calculated according to the formulas shown below respectively; the corresponding sketch for a line is shown in Figure 5.4, the sketch for an arc is shown in Figure 5.5, and the sketch for a clothoid transition curve is shown in Figure 5.6.

$$
\begin{aligned}
\text{Line}: \quad & \alpha_{\text{end}} = \alpha_{\text{start}} \\
& x_{\text{end}} = x_{\text{start}} + l_{\text{segment}} \cdot \cos\left(\alpha_{\text{start}}\right) \\
& y_{\text{end}} = y_{\text{start}} + l_{\text{segment}} \cdot \sin\left(\alpha_{\text{start}}\right)
\end{aligned}
$$

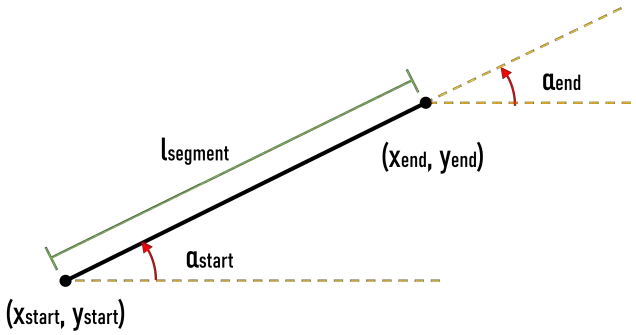


Figure 5.4: Parameters for a line segment.

Please note that the upper sign is for an arc defined in the counterclockwise direction, while the lower sign is for an arc defined in the clockwise direction.

$$
\begin{aligned}
\text{Arc}: \quad \beta_{\text{central angle}} &= l_{\text{segment}} / r_{\text{arc}} \\
x_{\text{centre}} &= x_{\text{start}} \mp r_{\text{arc}} \cdot \sin(\alpha_{\text{start}}) \\
y_{\text{centre}} &= y_{\text{start}} \pm r_{\text{arc}} \cdot \cos(\alpha_{\text{start}}) \\
\alpha_{\text{end}} &= \alpha_{\text{start}} \pm \beta_{\text{central angle}} \\
x_{\text{end}} &= x_{\text{centre}} \pm r_{\text{arc}} \cdot \sin(\alpha_{\text{end}}) \\
y_{\text{end}} &= y_{\text{centre}} \mp r_{\text{arc}} \cdot \cos(\alpha_{\text{end}})
\end{aligned}
$$

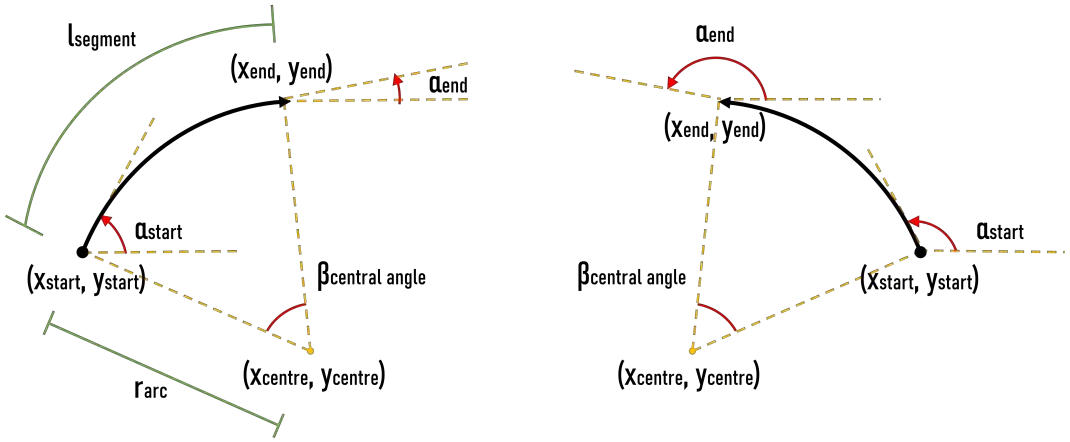


Figure 5.5: Parameters for an arc segment in clockwise (left) and counterclockwise (right) directions.

Transition curves For simplicity only the clothoid, more formally known as an Euler spiral, transition curve is implemented.

The clothoid does not have a closed-form analytic solution and must either be numerically integrated or approximated using series expansions of the FRESNEL integrals shown in Eqs. 5.1 and 5.2 for the x- and y-coordinates respectively. For numerical convenience, the FRESNEL integrals are integrated using a fixed 8-point GAUSS-LEGENDRE integration that provides sufficient accuracy for the prototype and forgoes the complicated series expansion necessary to calculate a transition curve from arc to arc. The sign of the curvature κ_i determines the handedness of the curve at start and endpoint; clockwise is positive, counterclockwise is negative (R.E.Deakin, 2005).

$$
\begin{aligned}
\text{Clothoid}: \quad \kappa_{\text{start}} &= \begin{cases} \pm\frac{1}{r_{\text{start}}} & r_{\text{start}} \neq 0, \text{tangent to arc at start} \\ 0 & r_{\text{start}} = 0, \text{tangent to line at start} \end{cases} \\
\kappa_{\text{end}} &= \begin{cases} \pm\frac{1}{r_{\text{end}}} & r_{\text{end}} \neq 0, \text{tangent to arc at end} \\ 0 & r_{\text{end}} = 0, \text{tangent to line at end} \end{cases} \\
\Phi &= \kappa_1 s + \frac{\kappa_2 - \kappa_1}{2 l_{\text{segment}}} s^2 \\
x_{\text{end}} &= \int_0^{l_{\text{segment}}} \cos\left(\Phi\left(s\right) + \alpha_{\text{start}}\right) \mathrm{d}s + x_{\text{start}} \qquad (5.1) \\
y_{\text{end}} &= \int_0^{l_{\text{segment}}} \sin\left(\Phi\left(s\right) + \alpha_{\text{start}}\right) \mathrm{d}s + y_{\text{start}} \qquad (5.2) \\
\alpha_{\text{end}} &= \alpha_{\text{start}} + l_{\text{segment}} \cdot \left(\kappa_1 + \frac{\left(\kappa_2 - \kappa_1\right)}{2}\right) = \alpha_{\text{start}} + \Phi\left(l_{\text{segment}}\right)
\end{aligned}
$$

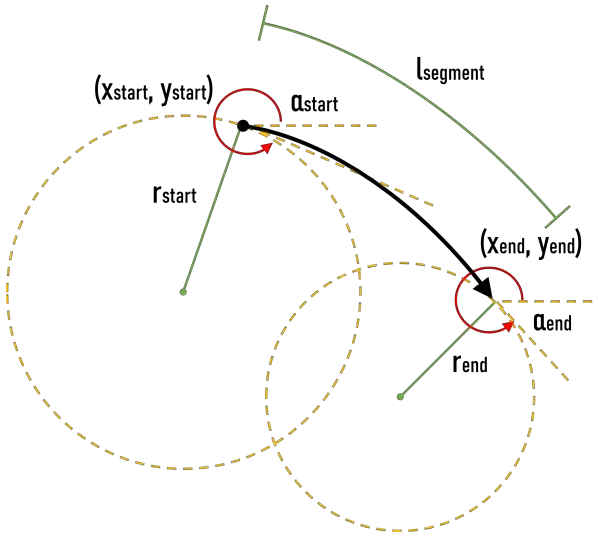


Figure 5.6: Parameters for a clothoid curve.

azimuth and deltaAzimuth

StartDirection of IFC adopts a plane angle usually with the unit *rad*, measuring the direction of the tangent at the start point from the positive x-axis; on the other hand, azimuth measures the direction angle of a horizontal curve clockwise from the north in degrees. In the implementation, the positive x-axis and the positive y-axis are assumed to indicate the east and the north respectively, the conversion of these two direction measure systems can therefore be simplified. deltaAzimuth is required when the direction of a curve has changed; it is equal to the difference between azimuth at the endpoint and azimuth at the start point.

$$\text{angle}_{\text{azimuth}} = \begin{cases} (\pi/2 - \text{angle}_{\text{plane}}) \cdot 180/\pi & \text{angle}_{\text{azimuth}} > 0 \\ (5 \cdot \pi/2 - \text{angle}_{\text{plane}}) \cdot 180/\pi & \text{angle}_{\text{azimuth}} < 0 \end{cases}$$

Relation

From Section 3.1.1, it is known that IFC objects are associated together by the entities inherited from IfcRelationship. Conversely, multiple IfcAlignments are only grouped and assigned to a project by the entity IfcRelContainedInSpatialStructure without real connections (see Code 3.1), as each IfcAlignment used as an LRS is defined independently. From visualizing alignments in a BIM platform such as Autodesk Civil 3D, it can be seen that the two alignments seem to be connected but can actually be split by moving the control points on both ends of each alignment. For this reason, finding relationships and connecting alignments is an essential step to build a topological network. Since IfcAlignment stores purely geometrical information, relationships can only be found by an exhaustive comparison of all alignment endpoint locations. To be specific, an alignment has two ends; each end needs to be compared with the ends of other alignments. If there are n alignments, the needed number of comparisons are $2 \cdot n \cdot (2 \cdot (n-1)) = 4 \cdot n^2 - 4$ in the worst case, with a complexity of $O\left(n^2\right)$. To increase efficiency, a quadtree algorithm with $O\left(n \cdot \log n\right)$ is adopted.

Using the quadtree algorithm can significantly ease the work required to locate alignments, besides, it also provides a simple approach to connect a newly inserted alignment into the network. There are several steps needed to find the relationships among alignments using a quadtree (please also see Figure 5.7, Code 5.1, and Code 5.2):

1. Determine the maximum and minimum extent of alignments and create a bounding box to define the root of a quadtree.

2. Separately feed both ends of an alignment as nodes into the quadtree. If the current quad is empty or has less than a preset number of unique nodes in a quad, then add the node into this quad; otherwise, divide the quad into four children quads.

3. Determine the destination of the node by checking which child quad contains the node coordinates, then insert the node into the corresponding quad.

4. Repeat from the second step if the node does not fit into the children quads.

```
1  public void InsertNode(Node node) {
2      if (!quadRect.Contains(node)) {
3          //alignment point is not inside the current rectangle
4          return;
5      }
6      if (nodes == null || (quadChildNE == null &&
7          UniqueCount(node) <= maxUniqueNodesPerQuad)) {
8          //Current rectangle is not filled yet
9          AddNode(node);
10     } else {
11         if (quadChildNE == null) {
12             //Subdivide the current rectangle and insert alignment
13             //into correct child rectangle
14             Subdivide();
15         }
16
17         //Child has been created or exists already
18         //Find out which child the alignment belongs to
19         QuadTree<T> destination = GetDestination(node);
20
21         if (destination == this) {
22             //We are in or have reached the correct rectangle
23             AddNode(node);
24         } else {
25             //Traverse down through the tree until
26             //we have found the correct rectangle
27             destination.InsertNode(node);
28         }
29     }
30 }
```

Listing 5.1: Insert an alignment node into a quad.

After a few recursive loops, all alignments' nodes are placed into the quadtree. As shown in Figure 5.7, a quad is not divided if it is empty or has only a single node, while a quad is further divided until one of the children quads can contain the node. The green dashed lines represent alignments that are unrelated, yet have nodes close to the black connected

alignments. Each quad can only contain a preset number of unique nodes, which means a quad only allows a limited number of unique node coordinates inside it; the fewer the numbers of unique nodes allowed, the deeper the quadtree is going to be. A limit of two unique nodes in a quad is chosen to balance the time needed to recursively reach the target quad and the iterative time needed to find the node inside the quad that is inside the search box.

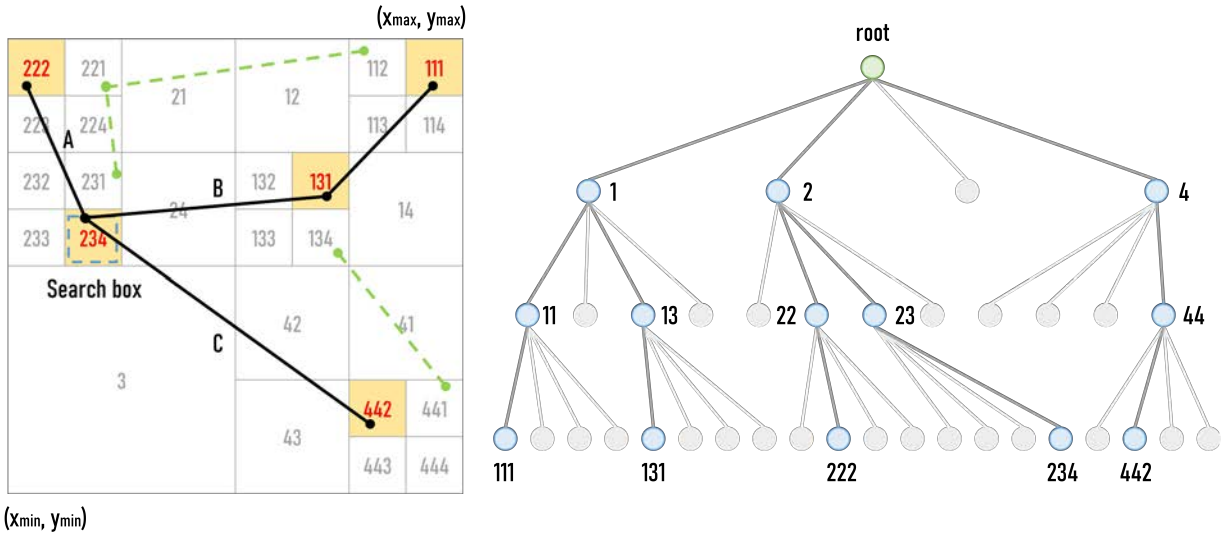


Figure 5.7: Finding the relationships among alignments by a quadtree.

After the nodes have been placed in a spatial 2D space via the quadtree, the relationship of each alignment can be determined. To achieve this, the nodes and their host alignments need to be retrieved from the quadtree. A node is retrieved by a recursive collision detection. First of all, the coordinates of a node of an alignment are surrounded by a box with a small tolerance in width and height to make a search box, then any node inside that search box is retrieved from the quadtree (see Code 5.2). As shown in Figure 5.7, assume a search box is made using the coordinates of the node of alignment A contained in the quad with Id 234, then the nodes of the alignments A, B and C are retrieved. According to this information, the alignment A, the alignment B, and the alignment C are known to be connected.

```
1  public void GetNodes(Rectangle rect, ref List<Node> results) {
2     if (results != null) {
3        if (rect.Contains(quadRect)) {
4           //Desired area contains the full quad, so just return it and all its child
                members
5           GetAllNodes(ref results);
6        } else if (rect.Intersects(quadRect)) {
7           // Check in current quad
8           if (nodes != null) {
9              for (int i = 0; i < nodes.Count; i++) {
10                if (rect.Contains(nodes[i])) {
11                   results.Add(nodes[i]);
12                }
13             }
14          }
15          // Check in all the child quads
16          if (quadChildNE != null) {
17             quadChildNE.GetNodes(rect, ref results);
18             quadChildNW.GetNodes(rect, ref results);
19             quadChildSW.GetNodes(rect, ref results);
20             quadChildSE.GetNodes(rect, ref results);
21          }
22       }
23    }
24 }
```

Listing 5.2: Get alignment nodes at the same quad.

5.2.4 Topo&Geo

In order to store the relationship among alignments, a temporary network storage for bi-directional conversions, Topo&Geo, is proposed. Topo&Geo provides an interface to fit the different object definitions of IfcAlignment and RTM. Topo&Geo defines three classes, Node, Connector, and ConnectorContainer, for a temporary network based on a classical topological representation. Continuing the example shown on the left side of Figure 5.7, three joined alignments are found in quad 234; nevertheless, the alignments are not sequentially connected as IfcAlignment does not contain any information on the orientation. To take an example as shown in Figure 5.8, the alignment A and the alignment B represent the LRS for a continuous track, but the endpoint of alignment A does not connect to the start point of alignment B; the mileage from the LRS increases from the start points of both alignments towards their endpoints and so switch direction when two endpoints meet. For this reason, it is important to not only find out the neighbour alignments but also their orientation. Because the relationship is found using the quadtree algorithm, with each node being identified as either a start or endpoint of an alignment, it is already known which end of the alignments are connected; thus, the next step is to store this information and complete the other properties properly.

The temporary network storage is composed of three units, from the smallest to the largest they are NODE, CONNECTOR, and CONNECTORCONTAINER (see Figure 5.8). NODE can be either the start point or endpoint, two NODEs are aggregated and assigned to a CONNECTOR. Because there are multiple NODEs existing at the same location, a container, CONNECTORCONTAINER, is created to store CONNECTORS. The type of the container depends on the number of CONNECTORS: (1) one CONNECTOR represents the end of a track; (2) two CONNECTORS represent a continuous track based on two alignments; (3) three CONNECTORS represent a switch. Additionally, the temporary network storage also offers a class to store the navigability, which is an essential information for a topological representation, e.g. RTM, but is not considered in a geometrical representation, e.g. IFCALIGNMENT. The navigability in the implementation uses two Boolean variables defining four scenarios, i.e., (1) bidirectional navigability A $\leftrightarrow$ B, (2) unidirectional navigability from A $\rightarrow$ B, (3) unidirectional navigability from B $\rightarrow$ A, and (4) no navigability; the navigability is assigned randomly in the prototype tool for IFCALIGNMENT.

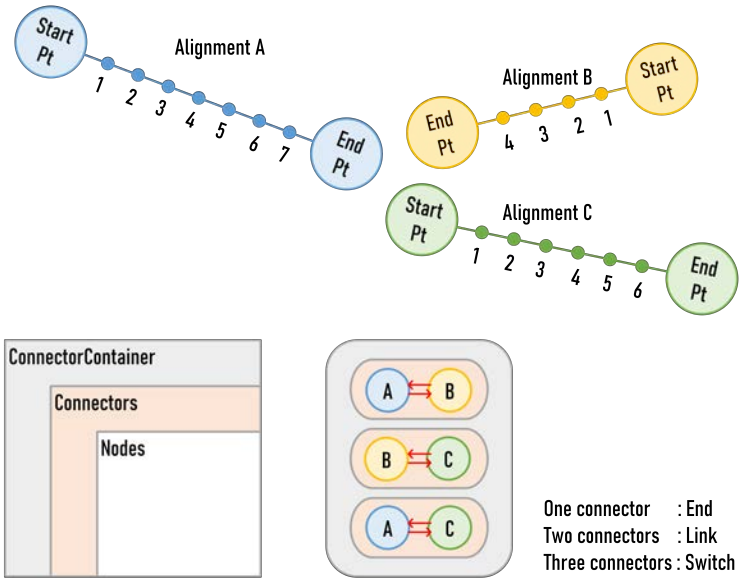


Figure 5.8: Temporary network storage.

5.2.5 Topo2Geo

TOPO2GEO allows users to transfer a RAILML file to an IFC file. The RAILML schema provides element sets for both topological and geometrical representations, RTM and GEOMETRY. According to the comparison result shown in Table 5.4, all entities for IFCALIGNMENT can be determined from the elements of GEOMETRY, whereas six entities describing curve shapes require external supplementation when reconstructing an LRS geometry from RTM. Here, the

discussion is focused on the conversion from RTM to IfcAlignment as it causes more data inconsistency than Geometry to IfcAlignment.

RTM offers three positioning methods, namely intrinsic coordinates, linear coordinates, and geometric coordinates, allowing users to locate railway components and networks (railML.org, 2016a). The intrinsic coordinates are used to specify a location inside a NetElement; the coordinate can be between 0 to 1, where 0 and 1 represent each end of a NetElement. The linear coordinates allow an LRS to locate NetElements; the location can be defined absolutely, relatively or laterally as shown in Figure 3.2b. The geometric coordinates are used to place NetElements in a Cartesian or spherical coordinate system. The intrinsic coordinates are mandatory to locate a NetElement in a network, but it is optional to associate with linear coordinates or geometric coordinates. However, no matter whether using linear coordinates or geometric coordinates, the shape of a NetElement still needs to be defined manually.

Depending on the referencing system, different extra information is required for building a geometrical representation. When linear coordinates are used, the length, measure, and relation are known; the geometric coordinates and curve type as well as curve parameters for each NetElement remain unknown. On the other hand, when geometric coordinates are used, the coordinates at both ends of NetElement and relation are known; the length and curve type as well as curve parameters for each NetElement remain unknown. In general, linear coordinates are used more often than geometric coordinates to describe a railway network, thus how to convert a topological representation using the linear coordinates defined in RTM into a geometrical representation in IfcAlignment is considered in the tool.

Developing a Complete Geometrical Representation Using Limited Parameters

To create consistent geometry using minimal manual input, an automatic geometry generation algorithm is developed in the tool. In a track layout, alignments must be connected tangentially; therefore, the generation algorithm only requires an initial node coordinate of a NetElement, the curve types, and curve parameters of each NetElement in the network. Figure 5.9 shows the flowchart of how to complete a geometrical representation. One NetElement with an initial coordinate and direction is chosen as a root element, and then the endpoint for the root element is determined according to its curve type. If the root NetElement has adjacent NetElements, then the tool will get these NetElements and classify them into two groups by checking whether the start point or the endpoint connects to the endpoint of the root element and assign the appropriate coordinates. Finally, each adjacent NetElement is recursively assigned as the root element, and the loop continues until every NetElement in the network has been processed.

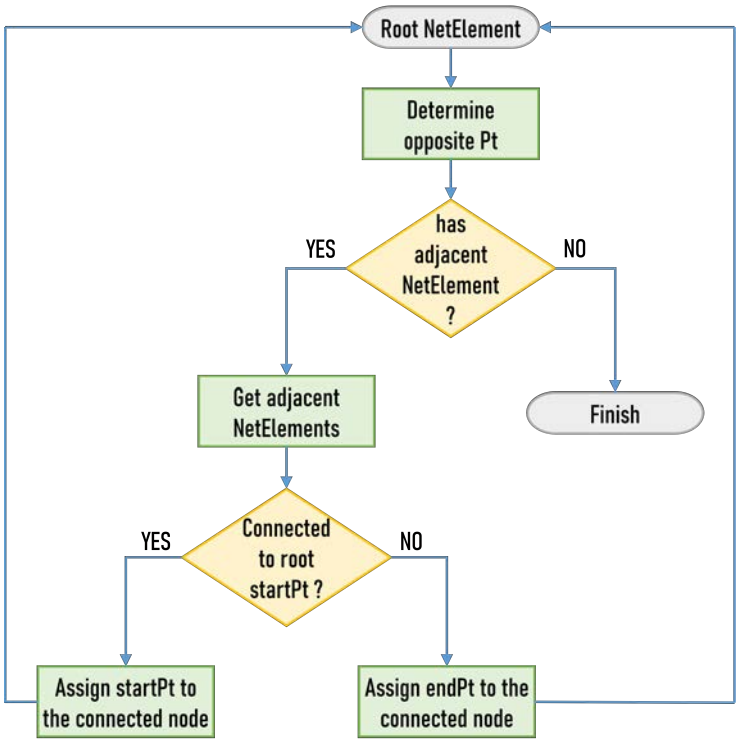


Figure 5.9: Flowchart of geometrical completion.

The LoD Issue

An RTM stores multiple LoDs, while an IfcAlignment stores only one level of geometrical information. Because the axis of an IfcAlignment is composed of segments, it can be considered as two levels; IfcAlignment is the top level representing rough detail, and IfcAlignment2DHorizontalSegment is the lower level representing precise detail. To investigate the feasibility of storing LoDs in IfcAlignment, the two suitable combinations of LoD linking between RTM and IfcAlignments, listed in Table 5.5, are created in IFC files and compared.

The first combination assumes an alignment is in the *Meso* level and segments are in the *Micro* level; the second combination assumes an alignment is in the *Micro* level and segments are in the *Nano* level. The created IFC files are visualized in the IFC viewing and validation application FZKViewer, developed by the Institut für Automation und angewandte Informatik of the Karlsruher Institut für Technologie; the results of both combinations are shown in Figure 5.10. The curve geometry on the left-hand side encompasses three alignments; each alignment has one or two segments that define a single continuous axis for the alignment. In contrast, the curve geometry on the right-hand side contains only one alignment including four segments; however, a false representation resulted from two diverging arc segments from a continuous curve.

According to this experiment, IfcAlignment is not suitable to describe geometry converted from the *Meso* level, in particular, IfcAlignment cannot represent joint segments that violate tangential continuity correctly. For this reason, this thesis considers IfcAlignment in the *Micro* level and IfcAlignment2DHorizontalSegment in the *Nano* level. Of course, this assumption is not perfect as the information from the *Meso* level, such as the aggregation of elements from the *Micro* level, is abandoned during the conversion from RTM to IfcAlignment. Besides, an alignment is created based on a single NetElement, which might not describe the detailed geometry. Nevertheless, this assumption represents a valid geometry in IFC; furthermore, the detailed geometry can be recreated by adding more NetElements in RTM. In addition, once the created IFC is transferred back to railML, the geometry can be referred to by the Id in the element sets of Geometry, which are associated with the elements in RTM.

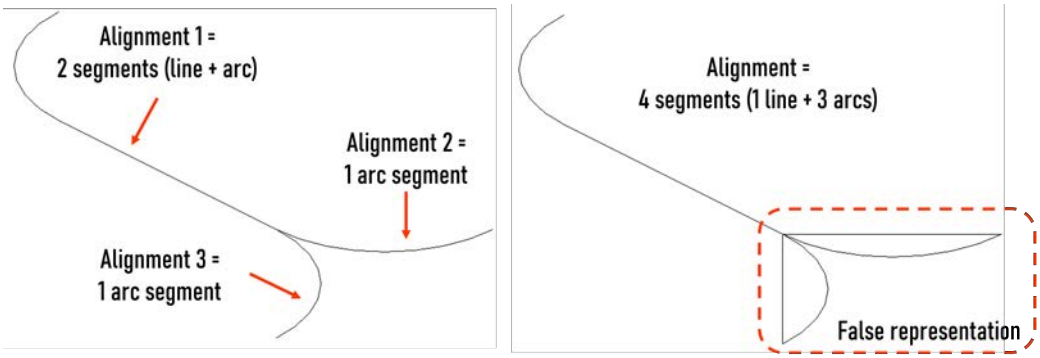


Figure 5.10: An experiment on the feasibility of IfcAlignment storing LoDs.

5.2.6 Visualization

The Neo4j graph platform is utilized to visualize the topological representation. The developed tool for this thesis parses the source data from the IFC and railML files and then creates and modifies a graph database in a local platform. Figure 5.11 shows the topological representation for the curve geometry created by three alignments as seen on the left-hand side in Figure 5.10. In the *Micro* level, one end of the alignment *1* connects to the diverging alignment *2* and the alignment *3*; in the *Nano* level, it can be seen that the alignment *1* is composed of one line segment and one arc segment. The tool first pre-processes the entities IfcAlignment, including finding the relationships and extracting the geometry of segments, and then implements the network in Neo4j. From RTM to IfcAlignment, the network can be created directly from NetElements and NetRelations. The visualization in Neo4j gives users a quick overview to debug the geometry or the network they are converting.

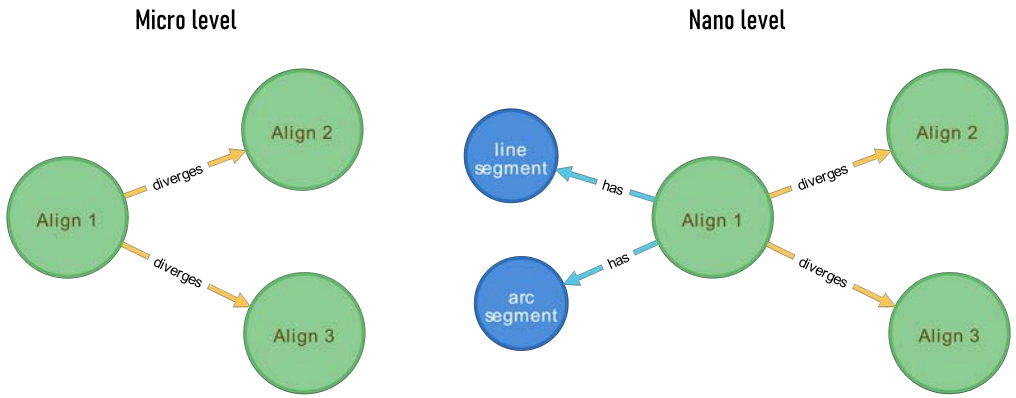


Figure 5.11: Topological representation in Neo4j.

5.2.7 GUI

Having implemented all the data exchange concepts introduced above, a GUI framework is developed using WPF, which provides a user-friendly interface to inspect and manipulate data. The GUI framework is made following the design pattern Model-View-ViewModel (MVVM). There are four parts of the implementation: DataAccess, Models, ViewModels, and Views. DataAccess is the interface between GUI and Geo2Topo as well as between GUI and Topo2Geo. After both Geo2Topo and Topo2Geo read data files, DataAccess accesses the data and organizes it based on the data model created in Models, which controls how the data is represented in the GUI. Although both STEP and XML languages are human-readable, it is still exhausting work when it comes to large data sets, so the data is organized in a list to facilitate data inspection. If data is missing and needs to be added manually, then ViewModel is responsible for these kinds of interactive events. The last part Views purpose is to design the graphical objects. This design pattern ensures every part is independent and has clear data interfaces.

A complete introduction regarding the developed tool is given in the next section.

5.3 Tool Prototype

A prototype tool is developed, supporting the data exchange from IFC 4.1 to railML 3.1 and vice versa. The tool offers five main functions. It reads both IFC and railML files and performs their conversion; connects to Neo4j and draws the relationships for checking; and displays the important parameters for geometry in different scales. Besides, users can enter any necessary information for creating the IFC geometry from railML. Furthermore, the tool can complete a geometrical representation from a topological representation using only minimal input and from any start point.

The main user interfaces are shown in Figure 5.12a and in Figure 5.12b. The explanations regarding the functional areas marked in both figures are listed below:

1. Two tabs that can be switched between the conversions IFC to railML and railML to IFC.

2. Four buttons that allow users to select a data file, convert data, visualize data in Neo4j, and open the converted file in a text editor. The tool is foolproof in the conversion, checking if data has been completed before converting; if a railML file contains Geometry, then the converted file is directly output, otherwise, the tool only converts data once sufficient information has been input.

3. This area shows diagnostic output and action events, giving users information on background processes.

4. For both conversion directions, data is grouped and listed by scales. Users can manually select the scale they are interested in. *Nano* and *Micro* levels are available for IFC; *Micro* and *Meso* levels are available for railML.

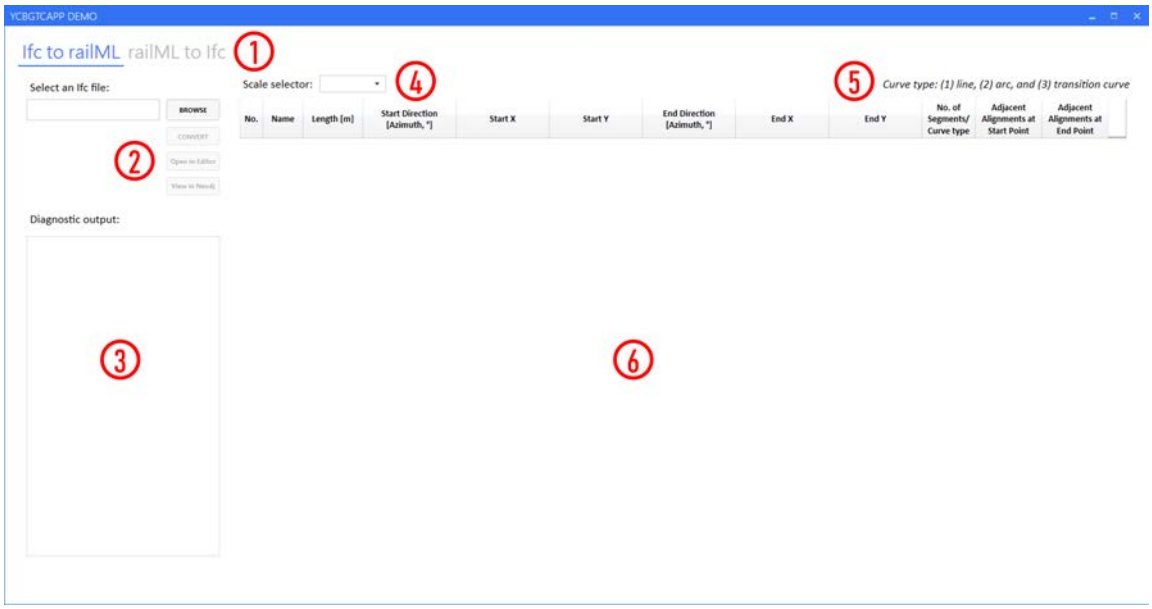
5. Hints for the curve types in the column *No. of Segments/Curve type* for an IFC file; *1* is a line, *2* means an arc, and *3* is a transition curve.

6. This area shows important curve information including name, length, start direction, coordinates of the start point, end direction, coordinates of the endpoint, numbers of segments in an alignment, and the adjacent alignments at both ends. The angle unit for IFC to railML is Azimuth in degrees, while the angle unit for railML to IFC is the plane angle in degrees.

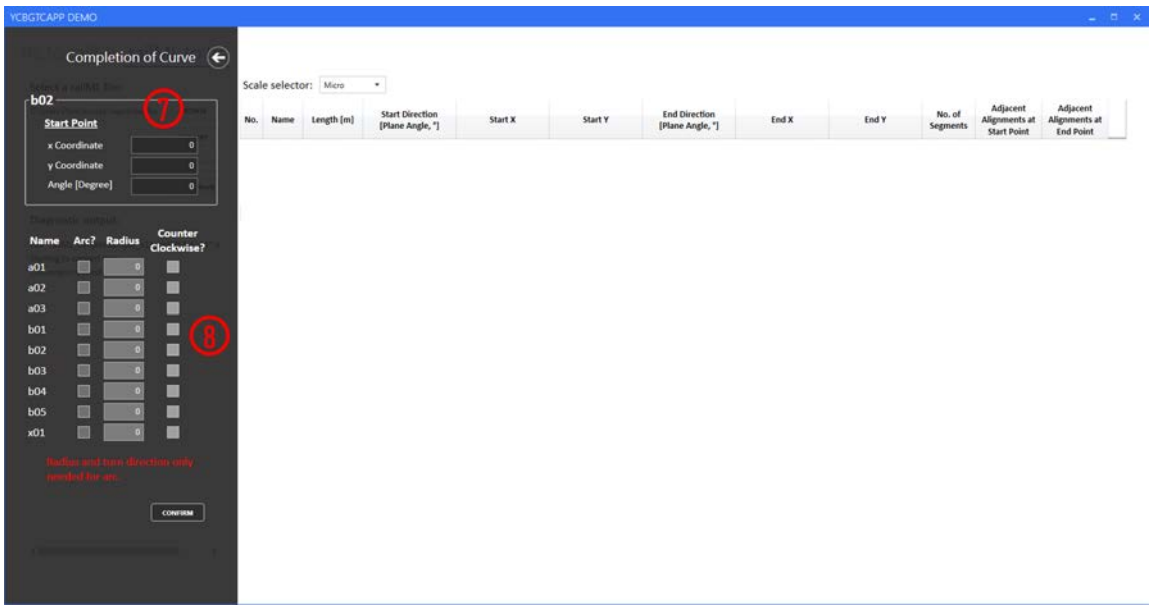
7. If a railML topology data set has no Geometry data sets, then users can choose any start point to complete the data; once an element is selected, the flyout pops out. Here, the start point coordinate and the start direction of the selected element are required.

8. From railML to IFC, the tool offers two curve types. Users need to select one curve type and give a radius with direction. Only if an arc has been selected are the boxes for radius and direction enabled.

In Chapter 6, the developed tool is evaluated regarding the data consistency after conversions by testing with some selected example cases.



(a) The interface of IFC to railML.



(b) The interface of railML to IFC.

Figure 5.12: The developed prototype tool.

Chapter 6

Case Study

This chapter uses the developed tool to discuss how much data loss can be avoided during the conversions between railML and IFC. A short introduction of the validation tools and cases is given in Section 6.1. Then a detailed tool walkthrough of the cases is shown in Section 6.2. Furthermore, Section 6.3 discusses the current limitations and areas that might need improvement in the tool.

6.1 Introduction

The ability of the developed tool to handle typical use case scenarios is evaluated using two different IFC files and one railML file. The IFC files are handcrafted using xBIM as only few BIM platforms correctly support the latest IFC schema and railway systems. The railML file is an example file provided by railML.org (railML.org, 2019a). The validation tools and cases are explained in the following sections.

6.1.1 Data Validation and Viewing

Validation of the produced output against the data schema standards is carried out using both publicly available and proprietary programs. Autodesk Civil 3D is used for effective visualization of IFC data, while FZKViewer is adopted for schema validation and quick visual confirmation of the created IFC files. Regarding railML 3.1 there is so far no proper publicly available tool. The validation tool railVIVID provided by UIC only supports versions 2.0, 2.1, 2.2, 2.3, and 3.0.0.8 of railML (railML.org, 2016b). Instead, the visualization of railML is done using Neo4j, and the data content is inspected before and after conversion.

6.1.2 Case Selection

In this section, two geometry cases and one topology case are chosen. Each case is converted by the developed tool and analyzed whether the output data remains consistent.

Geometrical Cases

As discussed in Section 5.2.3, IfcAlignments are grouped rather than connected, so the relationship and navigability among alignments are yet to be determined. IfcAlignment stores only the geometrical information of curves; after conversion using the tool, the geometrical information is properly stored in the corresponding Geometry element sets in railML. Moreover, topological information representing the track network is expected to be correctly determined.

Case 1 The first case is a track layout with a 34 km long main line, a rail yard and multiple branch lines, consisting in total of 49 IfcAlignments, each made up of multiple segments. The main line contains several parallel IfcAlignments; along the main line, some branch lines are splitting from the main line or merging into the main line. Shown in the zoomed-in part of Figure 6.1 are the top two alignments of the main line and a branch line that connects to it via a switch. Due to the lack of relationship and navigability information, whether the branch line merges or splits from the main line is not clear; this information needs to be determined and added manually. The navigability is however currently only assigned randomly in the tool. This case shows the alignments in different LoDs, how it is converted and what relationships are found.

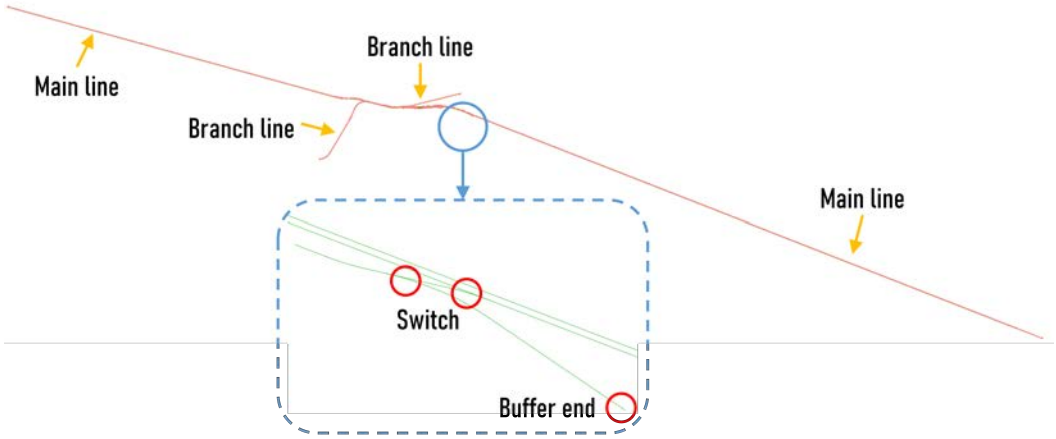


Figure 6.1: The track layout of Case 1.

Case 2 Case 2 is a step-by-step example. As shown in Figure 6.2, the first track layout is two sole IfcAlignments *A01* and *A02*, next, an unmodified layout with an additional diagonal *A03* is added; it is checked whether the diagonal changes the topological representation if it meets the two alignments without a joint. Later, the two original alignments are split into *A01*, *A011* as well as *A02*, *A022* and made to join the diagonal; the result is compared to the second example. Case 2 evaluates the criteria put upon the alignments to successfully construct a topological representation that mirrors the intent of the geometrical object.

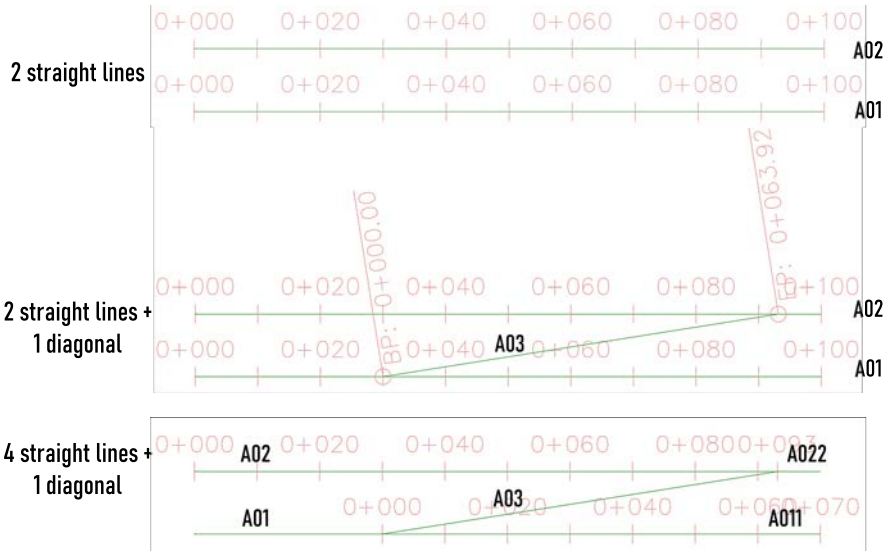


Figure 6.2: The track layout of Case 2.

Topological Case

Case 3 Case 3 is a railML 3.1 file of the topology of a track network in *Micro* and *Meso* levels as shown in Figure 6.3; the network has already been shown with detailed components in Figure 4.2a. The relationships and navigability of each element is visible in the *Micro* level, as well as their aggregation in the *Meso* level. The elements *a01*, *a02*, and *a03* are aggregated into the element *a11*; the elements *b01*, *b02*, *b03*, *b04*, and *b05* are aggregated into the element *b11*; both aggregations are connected by the element *x11*. Each element is implemented as a NetElement; the relationship and the navigability are implemented as NetRelations; there are two Networks separately storing the list of NetElements and NetRelations. Both sketches are drawn in a classical graph representation, the RTM is drawn in Neo4j later in Section 6.2.2. It can be expected that the tool asks users to input extra information to complete the geometrical representation from the given topological information.

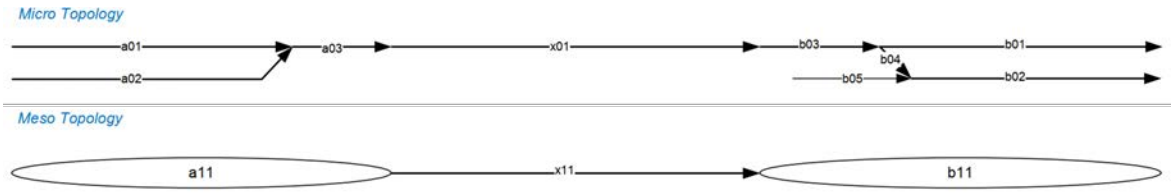


Figure 6.3: The topology of Case 3.

6.2 Tool Walkthrough

In this section, two types of conversions IFC to railML and railML to IFC provided by the tool are tested. A detailed walkthrough of both functions to introduce how to use the tool is made and possible data inconsistencies are discussed as well.

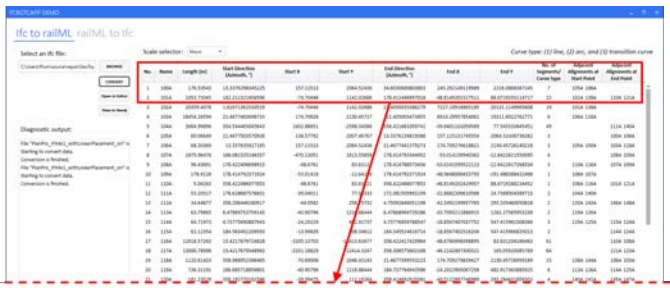
6.2.1 IFC to railML

Case 1

The IFC file for Case 1 is loaded into the tool. The project and geometrical data is parsed and listed in a table as shown in Figure 6.4. There are in total 49 IfcAlignments in the *Micro* level and 415 IfcAlignment2DHorizontalSegments in the *Nano* level. In the zoomed-in box, the parameters of the first row are stored in IFC, while the parameters of the second row, namely the coordinates and direction of the endpoints, numbers of segments in the *Micro* level and curve types in the *Nano* level as well as adjacent alignments at the start points and endpoints, are determined by the tool.

It can be seen that the first alignment *100A* is composed of seven line or arc segments shown in Figure 6.4b. The segments are ordered by the measure along the LRS of the alignment; for example, the coordinates of the start point of segment *100A_02* are $(165.17186, 2090.22919)$, which is very close to the coordinates of the endpoint of segment *100A_01* $(165.17185661882, 2090.22918140218)$; the slightly different values are due to the different precision of the stored start coordinates in IFC and the calculated end coordinates. On the other hand, alignments are neither ordered according to topological nor geometrical relationships to other alignments; for example, there are two alignments *105A* and *106A* at the start point of alignment *100A*, but no alignments at the other side.

The relationships among alignments can, of course, be checked from the table or in Neo4j. Figure 6.5a shows the data in an RTM; the large points represent alignments for IfcAlignment or NetElement for RTM; the small points represent the connexity relationship, Connector-



Original data

No.	Name	Length [m]	Start Direction [Azimuth, °]	Start X	Start Y
1	100A	176.53543	13.3376298345225	157.11513	2064.52436
2	101A	1053.73045	182.211321956596	-74.70446	1142.02686

Determined data

End Direction [Azimuth, °]	End X	End Y	No. of Segments/ Curve type	Adjacent Alignments at Start Point	Adjacent Alignments at End Point
34.6030600803803	245.292149119989	2216.0868387245	7	105A 106A	
178.412446997018	-48.8149201527511	88.6719255114717	22	102A 138A	110A 121A

(a) *Micro* level.

Original data

No.	Name	Length [m]	Start Direction [Azimuth, °]	Start X	Start Y
1	100A_01	26.96048	13.3376298345225	157.11513	2064.52436
2	100A_02	7.61141	21.4677144088425	165.17186	2090.22919
3	100A_03	28.14724	21.4677410181629	167.95746	2097.31255
4	100A_04	4.71166	32.9872135181472	180.81382	2122.29887
5	100A_05	3.94841	32.987201650495	183.37909	2126.25097
6	100A_06	78.94487	34.6030590865788	185.57522	2129.53211
7	100A_07	26.21136	34.6030600803803	230.40704	2194.51211

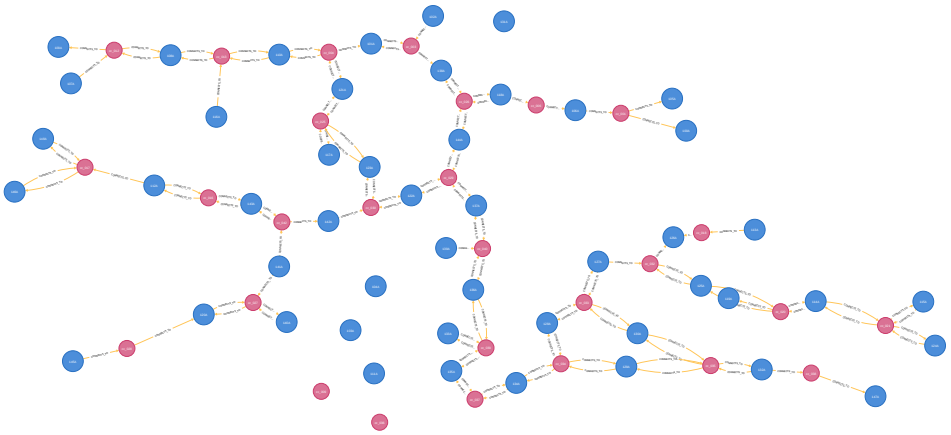
Determined data

End Direction [Azimuth, °]	End X	End Y	No. of Segments/ Curve type
21.467744137927	165.17185661882	2090.22918140218	2
21.4677144088425	167.957460142769	2097.31255036304	1
32.9871557106044	180.813818006094	2122.29886692815	2
32.9872135181472	183.379092043732	2126.25097314861	1
34.6031104275467	185.575222986175	2129.5321143282	2
34.6030590865788	230.407040053248	2194.51211007718	1
34.6030600803803	245.292149119989	2216.0868387245	1

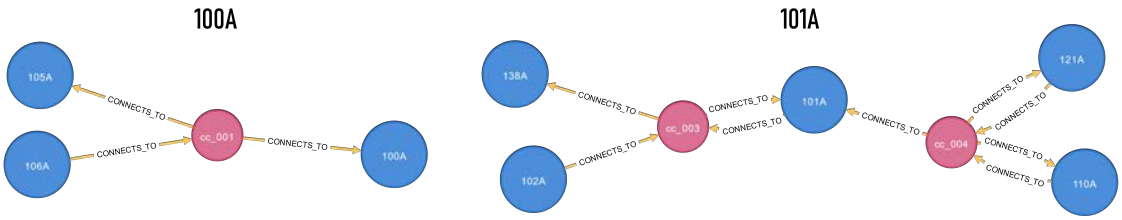
(b) *Nano* level.

Figure 6.4: The geometrical information loaded from the IFC file for Case 1.

ContainerRs for IfcAlignment or NetRelation for RTM; the arrows between alignments and relationships represent the navigability. As the navigability is still randomly created, some points are not connected. Figure 6.5b is focused on the node relations of *100A* and *101A*. It can be seen that *100A*, *105A*, and *106A* join together; the ConnectorContainerRs has three ConnectorRs storing the navigability between *100A* and *105A,* between *100A* and *106A,* and between *105A* and *106A*; *cc_001* represents a switch as there are three connected alignments, while one side of *100A* is a buffer stop. Similarly, *cc_003* and *cc_004* are switches containing the relationship information for the neighbour alignments (cf. Figure 6.4a).



(a) An overview of the topology.



(b) The nodes and the relationships for *100A* (left) and *101A* (right).

Figure 6.5: The topological representation of Case 1 in Neo4j.

To investigate whether the data exchange performed by the tool maintains data consistency, the converted railML file is reloaded by the other function railML to IFC and converted back to IFC. Because the geometrical information of IfcAlignment is stored in Geometry of railML, the geometry information remains the same without loss of data. However, one exception is the LoD. Although the relationships are found after conversion, there is no proper entities to store them, thus, the relationships need to be determined again in the next conversion.

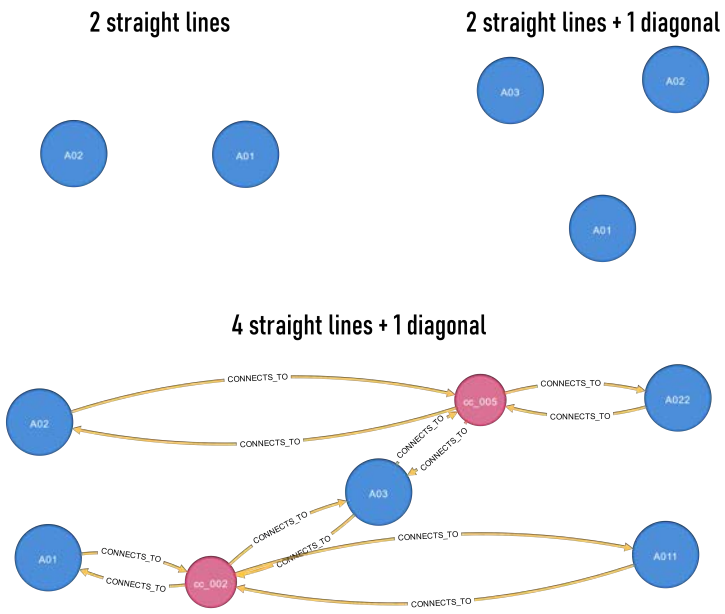


Figure 6.6: The topological representation of Case 2 in Neo4j.

Case 2

Three IFC files are converted into railML files for Case 2 and visualized in Neo4j as shown in Figure 6.6. In the first example, since the two lines *A01* and *A02* are parallel, the two nodes are independent. In the second example, even though the diagonal *A03* seems to join *A01* and *A02*, there is no real connection. For this reason, these three alignments are represented as independent in the topological representation. The diagonal is only related to the parallel lines, if there are real joints, i.e. *cc_002* and *cc_005*. According to the results of Case 2, it can be seen that relationships among alignments in a topological model only consider end-to-end joints. If a geometrical representation is converted to a topological representation in order to locate railway components, for example tracks and switches, as a reference system, it needs to be ensured that alignments join by start points or endpoints.

6.2.2 railML to IFC

Case 3

The example for Case 3 contains only topological information; the classical topology is shown in Figure 6.3 and the topological representation in RTM is shown in Figure 6.7. In Figure 6.3, it can be seen that the alignments *a01* and *a02* meet at a joint connecting *a03*, *x01*, and *b03*; the track line then splits into two directions starting from *b03*; one goes to *b01* and another

goes towards *b02* through *b04*; *b05* can be reached only from *b02*. The connexity is unclear in the traditional topology as the arrows only show the orientation of an alignment and does not give any information about the navigability. On the other hand, the connexity can easily be seen using NetRelation in RTM. As shown in Figure 6.7, it is clear that *a01* and *a02* have no connexity, nor between *b04* and *b05*; the same as described above, *b05* is reachable only through *b02* even though *b04* and *b05* are visually connected. Using NetRelation, the topological structures can also denote the correct relationships.

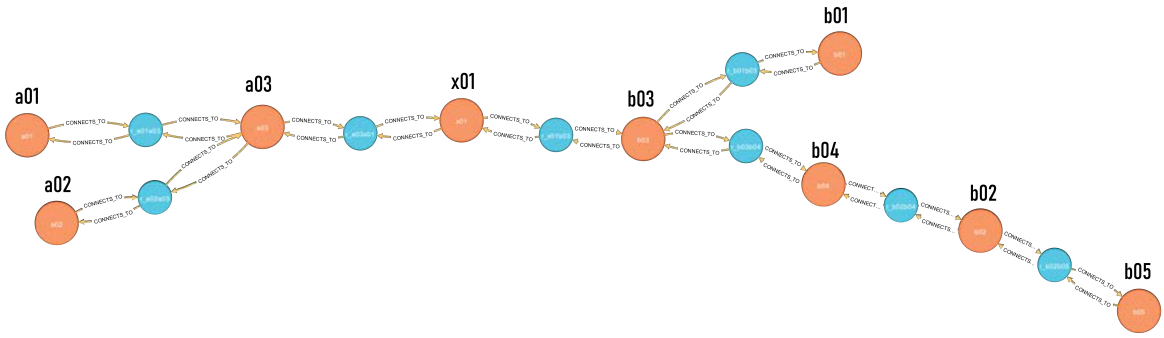


Figure 6.7: The topological representation of Case 3 in the *Micro* level in Neo4j.

Now, the tool transfers the RTM to IfcAlignment. If the corresponding elements of Geometry exist, the tool converts the railML directly to an IFC file; otherwise, the missing values for curve parameters are shown in red and need to be input (see Figure 6.8). Since an RTM only contains essential data for a pure topology, an assumed geometry is shown in Figure 6.9a. The alignments *a02*, *a03*, *b03*, *b04*, and *b05* are assumed to be arcs, and the remaining alignments are assumed to be lines. To complete the geometry, two parameters, radius and IsCCW, are required. The rule for IsCCW is based on the beginning of the selected start point, in the direction of the endpoint of each alignment; for example, if the start point is the *b05* start point, where the buffer stop is located, then IsCCW towards the joint is false; conversely, if the start point is the joint among *b02*, *b04*, and *b05*, the parameter IsCCW from the joint back to the buffer stop at *b05* is true. If IsCCW is chosen differently, the shape will change; for example, if the buffer stop at *b05* is the start point and IsCCW for *b03* is assigned false, then the whole geometry is horizontally flipped.

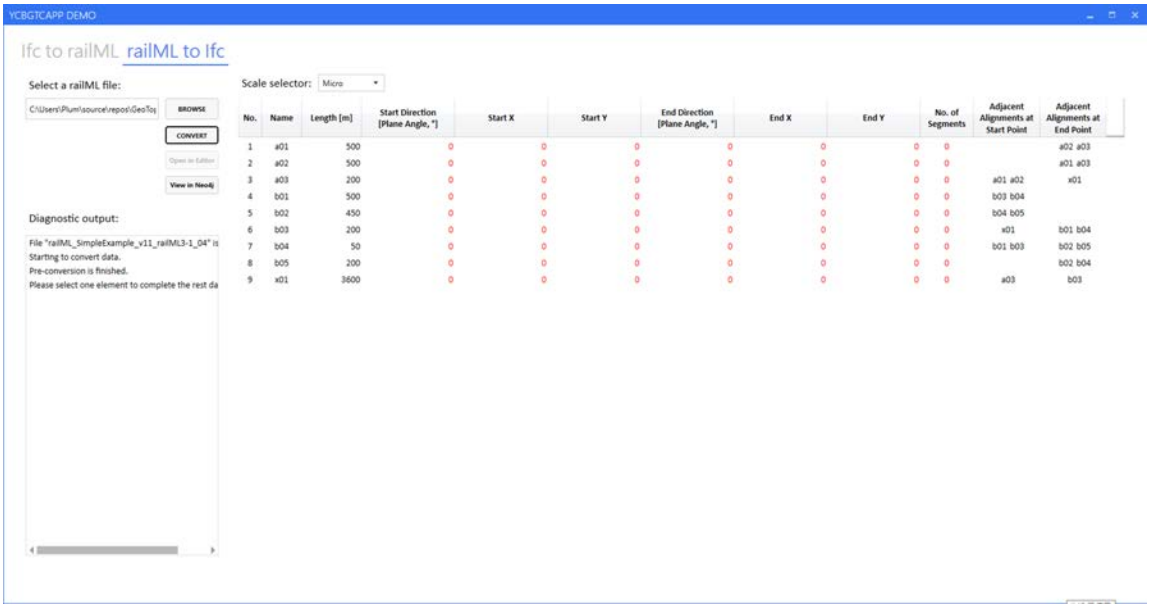
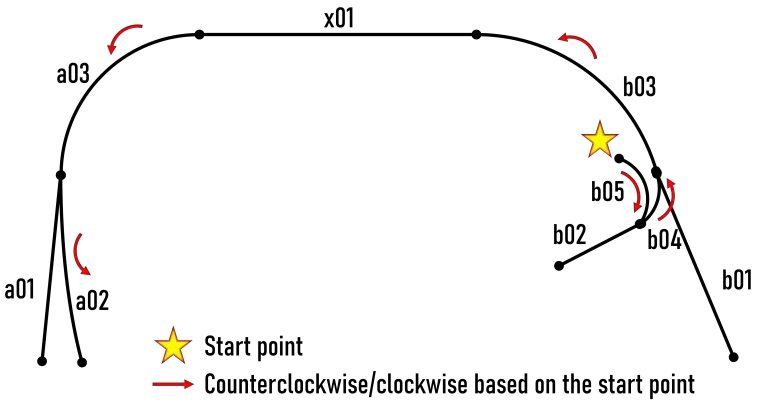
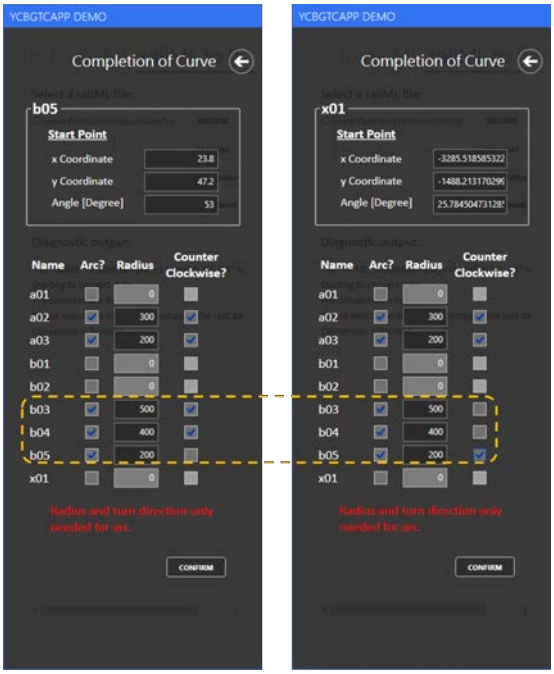


Figure 6.8: A railML file after pre-converting.

The tool offers users a simple way to give the curve parameters without knowing the detailed configuration of each alignment; for example, users do not need to know how the alignments connect together, such as whether the start point meets the endpoint of neighbour alignments or the other way around. The tool detects the start and end directions and modifies them if necessary. The logic behind the calculation is again that the relationships and measures along an LRS provided by RTM are used to assign the start points and endpoints of an alignment. Two start points meeting means the two tracks join in a switch; for example, *b04* and *b05*, while two endpoints meeting indicates two tracks splitting. Based on this information it can be easily decided if the IsCCW is reasonable for the geometry and complete the whole geometry by defining a single point of an alignment. This rule makes the geometry supplementation very flexible; the same shape can be arrived at by using a different start point and curve parameters. An example is shown in Figure 6.9b, two parameter sets that can create the same shape are shown in Figure 6.9a.



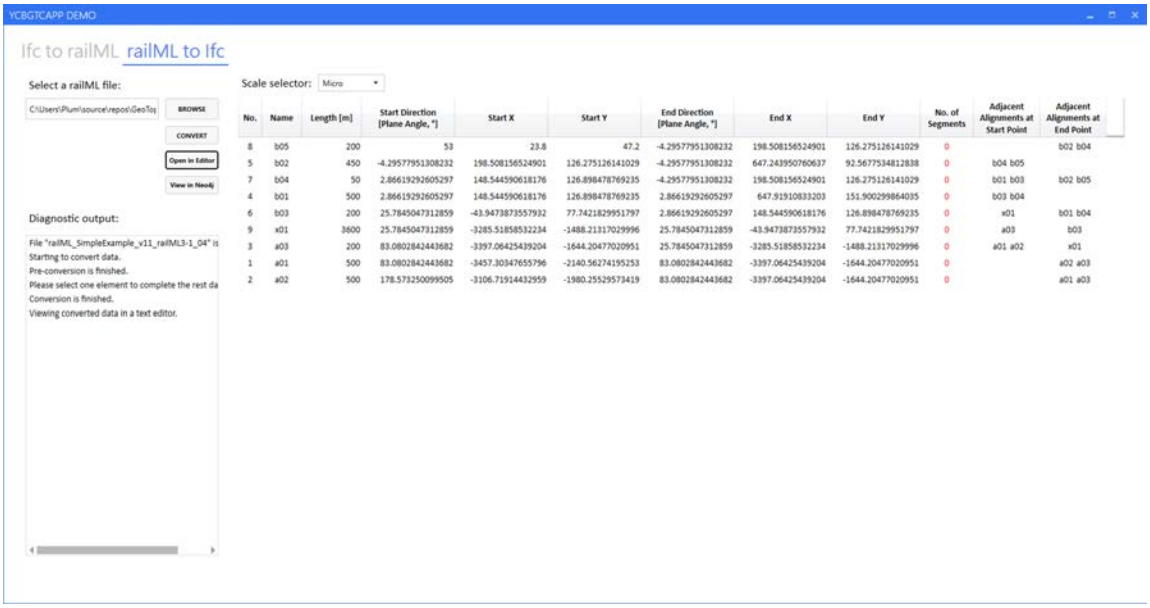
(a) A sketch of the desired geometry.



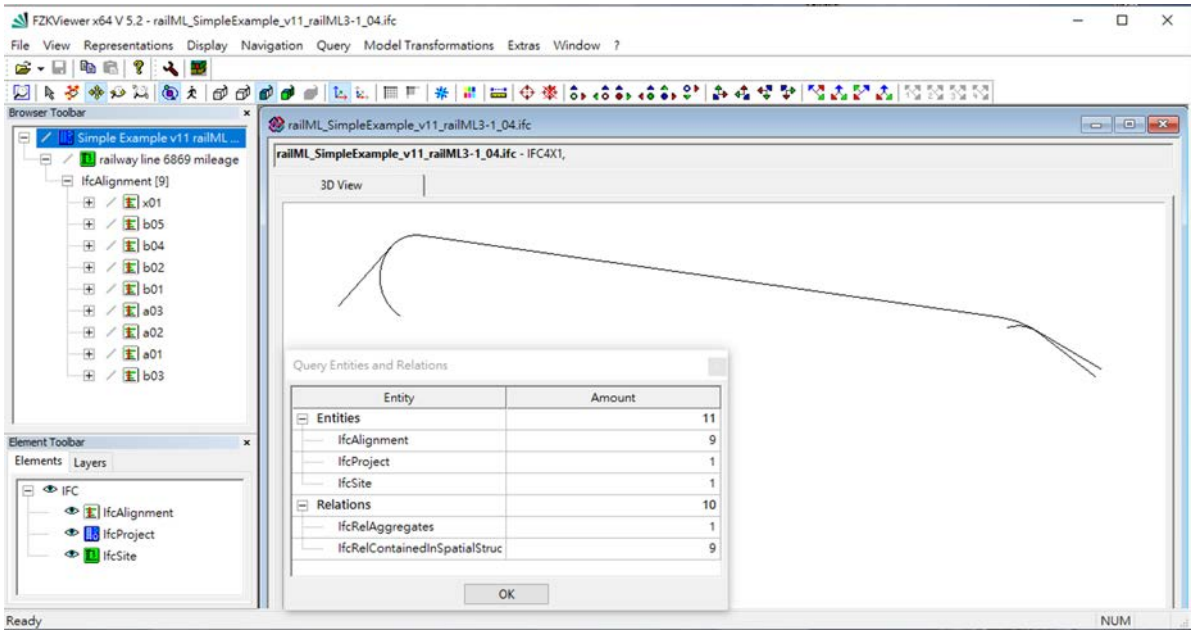
(b) Two parameter sets recreating the same geometry.

Figure 6.9: The desired geometry and the corresponding parameters.

After setting up the parameters, the tool calculates the curve geometries and converts the RAILML file to an IFC file (see Figure 6.10a). The created geometry is validated by FZKVIEWER without errors and shown in Figure 6.10b. Each NETELEMENT in the *Micro* level creates an IFCALIGNMENT, which means each IFCALIGNMENT contains only a single curve segment. The NETELEMENTS *a11*, *x11* and *b11*, giving the aggregation of the *Micro* elements into the *Meso* level, are lost in the conversion to IFC due to the LoD limitations mentioned in Section 5.2.5. Possible methods to mitigate that loss are described in Chapter 7.



(a) The missing parameters after reconstruction.



(b) The converted IFC file shown in FZKViewer.

Figure 6.10: The railML file converted to IFC.

6.3 Current Limitations

This thesis analyses possible data inconsistency sources and an tool has been developed that provides a possibility to link a geometrical and a topological representation while minimizing data loss. Nevertheless, there are still some remaining issues in the prototype that could be improved in the future. Some of these are for example:

- There are currently no proper entities for storing railway networks in IFC 4.1. Information regarding the connexity, the relationships, the navigability, LoDs and aggregation are lost during the conversion from railML to IFC.
- In both directional conversions, IfcAlignment refers to the NetElement in the *Micro* level, whereas IfcAlignment2DHorizontalSegment refers to the *Nano* level. The tool has no problem transferring from IFC to railML as the geometrical information in the *Nano* level is stored in Geometry, it can however only reconstruct the geometry from a topological representation in the *Micro* level.
- Similar to the previous issue, as the axis of IfcAlignment only supports a single continuous curve, it is not possible to store multiple curves merging or splitting from a curve that might be stored in the *Meso* level in an RTM.
- The current tool only converts the geometrical coordinates of an IFC clothoid curve to a railML file but can not recreate that transition curve from a railML file, as the railML schema does not contain sufficient elements to describe a transition curve to the same detail as IFC.
- The current tool only supports the conversions for topological and geometrical information but does not consider how to use the information stored in the LRS, such as signals locations or other trackside components.
- No automatic consistency check whether a given topology file is consistent with a given geometric file due to the previously mentioned shortcomings.
- The current tool does not support adding or removing nodes or alignments inside it, which might be very helpful in practice.

Chapter 7

Conclusion and Outlook

This chapter recapitulates the thesis, in particular regarding the BIM process, data exchange schemata, the analysis of IFC and railML, and the proposed solution. The thesis concludes by giving suggestions for possible future improvements and research directions.

7.1 Conclusion

With the development of CAD, the AEC industries have moved on from traditional paper documentation to digital documentation, and various BIM applications have been introduced to facilitate that transition. Nevertheless, project participants collaborate on diverse BIM applications specially focused on each sub-domain, which decreases interoperability. Railway infrastructure is especially affected by this problem due to the breadth and complexity of collaboration required. Multiple data exchange schemata are being continuously developed to achieve optimal collaboration efficiency, of which the most widespread one is IFC. The scope of the current IFC schema covers the full building domain but only the basic component of linear infrastructures, the LRS definition. Even though the IFC schema continually expands to better include infrastructures such as railways, it remains focused on the description of the geometry for the design and construction phases compared to the railML schema, which is focused on the operational aspects of a railway network (Augele, 2017). For this reason, linking data exchange schemata of different sub-domains, IFC and railML, is a worthwhile task to improve the interoperability of BIM applications for railways.

In practice, the complete IFC file describing the whole life-cycle of a facility is large, and therefore MVDs are used to extract only sub-domain relevant information rather than passing all extraneous information to another application. Following the spirit of an MVD, the pure information related to track layout is an essential basis for various use cases and needs to be exchangeable between IFC and railML. IFC 4.1 defines the geometry of a track layout

using the entity IfcAlignment, while railML uses RTM, namely NetElement and NetRelation in a Network, to define the network based on a track layout. Due to the diverging use cases, these two data exchange schemata are not fully compatible; the IFC schema defines the detailed geometrical representation for design and construction phases, whereas the railML schema defines the topological representation for the operational phase. This thesis analyzed the architecture of both schemata to evaluate a possible linking, investigated the sources of data inconsistency during data exchanges, proposed a solution, and developed a prototype tool.

Data consistency is core to maintaining good interoperability. However, it is naive to expect all data exchange to proceed without any data loss. A set of IfcAlignments itself is an LRS to specify the location of railway components, in contrast, a NetElement, related with other NetElements in a Network, is an instance located on an LRS or in a geometrical reference system. Moreover, an IfcAlignment supports multiple kinds of curves that can not be defined properly in the corresponding element set Geometry of railML. In addition, RTM supports the description of multiple LoDs yet IfcAlignment does not. Consequently, whether transferring from IFC to railML or the other way around, data inconsistency can occur. The proposed tool was implemented to minimize the loss of essential information and to reconstruct geometrical and topological representations.

The developed tool can read and write IFC and railML formats. It extracts and displays the geometrical and topological information and then exchanges that information. From IFC to railML, the tool determines the implicit geometrical information and from that the relationships among IfcAlignments. From railML to IFC, the tool determines the curve geometry from minimal input based on the topological information. Nevertheless, information regarding LoDs higher than the *Micro* level, the connexity, and the relationship is lost after a conversion from railML to IFC due to incomplete coverage of the data schema. Although the current tool does not guarantee perfect data consistency after a conversion, the implementation keeps the data in temporary storage and provides for future additional capabilities. As mentioned, it is not practicable to transfer all data from one to another data file as some of the data might not be necessary for the sub-domain one is interested in. For example, an engineer might use an IFC file to design the track layout according to the curve geometry but does not need the navigability; on the other hand, a railway system manager uses the network to control train scheduling but does not require detailed geometrical information. If, however, the geometry of the track layout needs to be changed or a part of a track is to be dismantled due to an operational reason, then the corresponding alignments can be found easily using the linked geometrical and topological representations, and any modifications kept synchronised. Therefore, the developed tool is very useful in such cases.

To conclude, compared to the data exchange inside a single domain, i.e. only inside geometrical applications and inside topology focused applications, this thesis developed a concept and

a tool to bridge the gap between geometrical and topological representations, enhancing productive collaboration and interoperability for railway infrastructures using the BIM process. The prototype tool linking these two representations simplifies data transfer, completes missing information with minimal data loss, and provides for future expandability.

7.2 Future Outlook

Due to the lack of proper entities and element sets in both IFC and railML schemata, some data inconsistency remains. Some issues could be solved temporarily using stopgap measures, such as storing topological information into the entities for pipe networks in an IFC file; some issues do not have a solution yet, such as a sub-schema that can store LoDs. Although future IFC extensions might include these related entities, the concept of the track topology introduced by the on-going project IFC Rail, (IFC Rail Project, 2019a) differs to RTM. Compared to the connexity graph theory in RTM where all nodes represent all kinds of resources, the IFC track topological model is based on a classical graph representation; a Track Node can be either a junction or a track end; a Track Edge is a simple railway track that refers to an alignment. Due to the different definitions of the topology, it is still not possible to have a straight forward data exchange by just transferring property values between IFC and railML, but extra validation in a tool such as the one developed for this thesis is needed. So, further improvements of the current tool, such as adding functions to modify nodes and alignments, to organize nodes into different LoDs, and to solve the other limitations introduced in Section 6.3, remain worthwile.

Data inconsistency caused by technical issues aside, data inconsistency due to procedural issues also remains to be solved; for example, how to detect modified alignments between two IFC files as the express files are not deterministically organized, meaning changing a single alignment modifies the entire file. Communication issues and versioning issues, where different sub-contractors work with different versions of files, are not easily detectable as identical geometry can be represented by many different permutations of express files. An automatic tool that can verify the consistency of geometrical and topological representations that should describe an identical state of the network could mitigate such issues.

Appendix A

Digital Appendix

The following items are included in the digital appendix of this thesis:

- One original PDF and one redacted PDF
- The English and German titles in a text file
- Figures with high resolution in *.png* format and code examples used in this thesis
- Example IFC and RAILML files used in Chapter 6
- Abstract in *.txt* format
- The Visual Studio project of the implementation
- The developed tool

Bibliography

Adamus, L. W. (2013). BIM: Interoperability for sustainability analysis in construction. *Central Europe Towards Sustainable Building: Integrated building design BIM*, 1–4.

AGC of America (2017). agcXML. https://www.constructionprogress.org/agc-xml-standards.html/. Accessed: 2020-02-28.

Amann, J., J. Jubierre, A. Borrmann, and M. Flurl (2014). An alignment meta-model for the comparison of alignment product models. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*, 351.

Augele, V. (2017). Comparative Analysis of Building Information Modelling (BIM) and RailTopoModel/railML in View of their Application to Operationally Relevant Railway Infrastructure. Technical report, Technical University of Dresden. Project Paper in the Field of Intelligent Transportation Systems.

Axelsson, P. and L. Wikström (2017, August). OGC City Geography Markup Language (InfraGML) Encoding Standard.

Baldwin, M., D. I. N. e.V., and M. und Maschine Schweiz AG (2018). *Der BIM-Manager : Praktische Anleitung für das BIM-Projektmanagement*. Berlin, GERMANY: Beuth Verlag.

Beetz, J. and A. Borrmann (2018). Benefits and limitations of linked data approaches for road modeling and data exchange. In *Workshop of the European Group for Intelligent Computing in Engineering*, pp. 245–261. Springer.

Borrmann, A., Y. Ji, J. Ramos-Jubierre, and M. Flurl (2012). Procedural Modeling: A new approach to multi-scale design in infrastructure projects.

British Standards Institution (2020). PAS 1192-2:2013 and PAS 1192-3:2014. https://www.bsigroup.com/en-IE/search-results/?q=pas1192/. Accessed: 2020-02-21.

buildingSMART International (2018, June). Industry Foundation Classes (IFC) Verson 4.1.0.0 Schema Documentation.

buildingSMART International (2020a). BIM Collaboration Format (BCF). https://technical.buildingsmart.org/standards/bcf/. Accessed: 2020-03-01.

buildingSMART International (2020b). Current Project and Activities in Infrastructure Room. https://www.buildingsmart.org/standards/rooms/infrastructure/. Accessed: 2020-03-11.

buildingSMART International (2020c). Current Project and Activities in Railway Room. https://www.buildingsmart.org/standards/rooms/railway/. Accessed: 2020-03-11.

buildingSMART International (2020d). IFC Specifications Database. https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/. Accessed: 2020-02-14.

buildingSMART International (2020e). Industry Foundation Classes (IFC). https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/. Accessed: 2020-02-14.

Bundesanstalt für Straßenwesen (2020). Objektkatalog für das Straßen-und Verkehrswesen (OKSTRA). http://www.okstra.de/. Accessed: 2020-02-28.

Chaplier, J., T. N. That, M. Hewatt, and G. Gallée (2010). Toward a standard: RoadXML, the road network database format.

Ciszewski, T., W. Nowakowski, and M. Chrzan (2017). Railtopomodel and railml - data exchange standards in railway sector. *Archives of Transport System Telematics 10*.

Daniotti, B., A. Pavan, S. Lupica Spagnolo, V. Caffi, D. Pasini, and C. Mirarchi (2020). *BIM-Based Collaborative Building Process Management*. Cham: Springer International Publishing.

Deutsche Bahn AG (2019, February). BIM Strategy - Implementation of Building Information Modeling (BIM) in the Infrastructure Division of Deutsche Bahn AG. Technical report, Infrastructure Division of Deutsche Bahn AG.

Dong, B., K. Lam, Y. Huang, and G. Dobbs (2007, 01). A comparative study of the IFC and gbXML informational infrastructures for data exchange in computational design support environments. *IBPSA 2007 - International Building Performance Simulation Association 2007 3*, 1530–1537.

Ducloux, P. (2016). RoadXML 2.4.1 - Road Network Description, XML Format Specification .

Esser, S. and A. Borrmann (2019). Integrating Railway Subdomain-Specific Data Standards into a common IFC-based Data Model. In *Proc. of the 26th International Workshop on Intelligent Computing in Engineering*.

Gély, L., G. Dessagne, P. Pesneau, and F. Vanderbeck (2010). A multi scalable model based on a connexity graph representation. *Computers in Railways XII, Beijing, China 1*, 193–204.

Gielingh, W. (2008). An assessment of the current state of product data technologies. *Computer-Aided Design 40*(7), 750 – 759. Current State and Future of Product Data Technologies (PDT).

Gröger, G., T. H. Kolbe, C. Nagel, and K.-H. Häfele (2012, April). OGC City Geography Markup Language (CityGML) Encoding Standard.

Hettwer, J. (2008). Objektkatalog für das Straßen-und Verkehrswesen (okstra). *Buhmann/Pietsch/Heins (Eds.): Digital Design in Landscape Architecture*, 234–239.

Hlubuček, A. (2017). RailTopoModel and RailML 3 in overall context. *Acta Polytechnica CTU Proceedings 11*, 16–21.

IFC Rail Project (2019a). WP2 – Requirement Analysis Report. Technical report, buildingSMART International.

IFC Rail Project (2019b). WP3 – Conceptual Model Report. Technical report, buildingSMART International.

INRETS, Oktal, PSA, Renault, Thales, and TRL (2016). RoadXML. https://www.road-xml.org/index.php/. Accessed: 2020-02-28.

ISO (2016). ISO 29481-1:2016 Building information models - Information delivery manual - Part 1: Methodology and format.

Jochim, H. and F. Lademann (2017). *Planung von Bahnanlagen*. Carl Hanser Verlag GmbH & Co. KG.

Jusuf, S. K., B. Mousseau, G. Godfroid, and V. S. J. Hui (2017). Integrated modeling of CityGML and IFC for city/neighborhood development for urban microclimates analysis. *Energy Procedia 122*, 145 – 150. CISBAT 2017 International ConferenceFuture Buildings & Districts – Energy Efficiency from Nano to Urban Scale.

Karan, E. P., J. Irizarry, and J. Haymaker (2016). BIM and GIS Integration and Interoperability Based on Semantic Web Technology. *Journal of Computing in Civil Engineering 30*(3), 04015043.

Kumar, K., A. Labetski, K. A. Ohori, H. Ledoux, and J. Stoter (2019, July). The LandInfra standard and its role in solving the BIM-GIS quagmire. *Open Geospatial Data, Software and Standards 4*(1), 5.

Kutzner, T., K. Chaturvedi, and T. H. Kolbe (2020). Citygml 3.0: New functions open up new applications. *PFG - Journal of Photogrammetry, Remote Sensing and Geoinformation Science*.

LandXML.org (2017). LandXML. http://www.landxml.org/. Accessed: 2020-02-25.

Lee, G. (2011, January). What Information Can or Cannot Be Exchanged? *Journal of Computing in Civil Engineering 25*.

Liebich, T. (2009, May). IFC 2x Edition 3 Model Implementation Guide. Technical report, buildingSMART International.

Lockley, S., C. Benghi, and M. Černý. Xbim.Essentials: A library for interoperable building information applications. *2*(20), 473.

Mahdavi, A., B. Martens, and R. Scherer (2014). *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*. CRC Press.

Nash, A., D. Huerlimann, J. Schütte, and V. P. Krauss (2004). railML - a standard data interface for railroad applications. *WIT Transactions on The Built Environment 74*.

Neo4j, Inc. (2020). Neo4j Graph Database. https://neo4j.com/. Accessed: 2020-03-19.

Open Green Building XML Schema, Inc. (2020). gbXML. https://www.gbxml.org/index.html/. Accessed: 2020-02-22.

railML.org (2016a, September). RailTopoModel - Railway infrastructure topological model IRS30100.

railML.org (2016b). railVIVID: The railML Viewer and Validator – powered by UIC. https://en.wiki.railvivid.railml.org/index.php?title=Main_Page#About_railVIVID. Accessed: 2020-03-27.

railML.org (2019a). railML example data. https://www.railml.org/en/user/exampledata.html. Accessed: 2020-03-13.

railML.org (2019b). railML Schema Version 3.1. https://www.railml.org/en/download.html. Accessed: 2020-02-08.

railML.org (2019c). railML® Use Case Definition Network Statement .

railML.org (2020). Introduction. https://www.railml.org/en/introduction/background.html/. Accessed: 2020-02-1.

R.E.Deakin (2005). Engineering Surveying - Horizontal Curves. http://www.mygeodesy.id.au/geodesy/. Accessed: 2020-03-26.

Richard See, Jan Karlshoej, D. D. (2012). An Integrated Process for Delivering IFC Based Data Exchange.

Rüffer, W., B. Feser, and F.-J. Knelangen (2001). Objektkatalog für das Straßen- und Verkehrswesen (OKSTRA) - Der Schlüssel zu Straßen- und Verkehrdaten. Technical report, Forschungsgesellschaft für Straßen- und Verkehrswesen (FGSV).

Sacks, R., C. Eastman, G. Lee, and P. Teicholz (2018). *BIM Handbook* (Third ed.). John Wiley & Sons, Ltd.

Scarponcini, P. (2005). ISO 19133 Tracking and Navigation Standard: 6.6 Linear Reference System Standard. *Transportation Research Record 1935*(1), 77–84.

Scarponcini, P. (2013). InfraGML Proposal (13-121). Technical report, OGC Land and Infrastructure Domain Working Group.

Seybold, B. and B. Franke (2013). Feasibility Study - UIC RailTopoModel and data exchange format. Technical report, trafIT solutions gmbh.

Shou, W., J. Wang, X. Wang, and H.-Y. Chong (2015). A Comparative Review of Building Information Modelling Implementation in Building and Infrastructure Industries. *Archives of Computational Methods in Engineering 22*, 291–308.

Solibri, Inc. and Tekla Corp. (2019). BIM Collaboration Format (BCF). https://www.bimcollab.com/en/Resources/OpenBIM/BCF/. Accessed: 2020-02-28.

Tim Chipman, Thomas Liebich, M. W. (2016). mcdXML - Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules.

Vilgertshofer, S., J. Amann, B. Willenborg, A. Borrmann, and T. H. Kolbe (2017). Linking bim and gis models in infrastructure by example of ifc and citygml. In *Computing in Civil Engineering 2017*, pp. 133–140.

Wunsch, S. and B. Jaekel (2017). Modellprinzipien des RailTopoModel. Technical report, Technologie Gesellschaft für Bauingenieurleistungen und Arbeitsvorbereitung mbH.