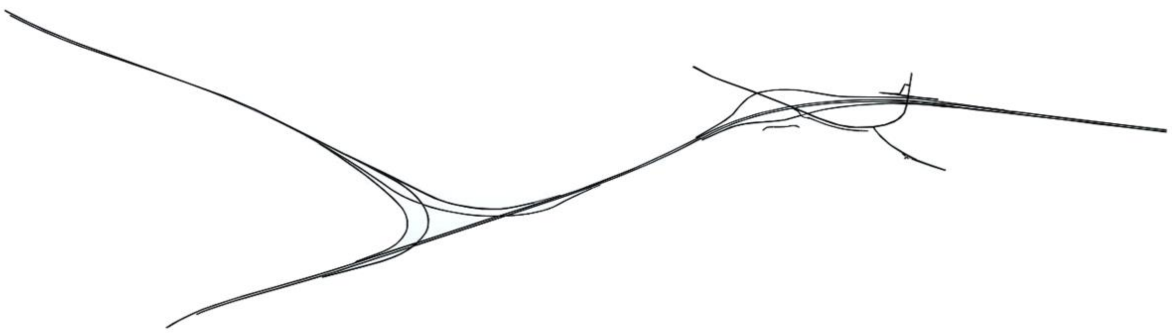
AUTOMATED GEOMETRY CHECKING FOR INFRASTRUCTURE PROJECTS

A research of implementing interoperable validation for IFC Alignment models in the Planning Phase of the System Engineering process based on open source



L.I.H. Wijnholts
Eindhoven University of Technology
Grontmij N.V., Part of Sweco

Final version:
January 27, 2016

COLOPHON

GENERAL	
Report	: Automated Geometry Checking for Infrastructure Projects
Status	: 1.0
Date	: January 27, 2016
Date final presentation	: February 3, 2016
Place	: Houten
STUDENT	
Author	: L.I.H. (Luuk) Wijnholts
Student number	: 0786166
E-mail	: l.i.h.wijnholts@student.tue.nl
University	: Eindhoven University of Technology
Master track	: Construction Management and Engineering
Chair	: Design Decision Support Systems
IN COLLABORATION WITH	
Company	: Grontmij N.V., Part of Sweco
Department	: Infrastructure and Mobility
GRADUATION COMMITTEE	
Chairman	: Prof. dr. ir. B. (Bauke) de Vries
University supervisor	: Dr. Dipl.-Ing. J. (Jakob) Beetz
University supervisor	: PhD Stud. T.F. Krijnen
Grontmij external supervisor	: N. (Niels) van Beurden
Grontmij external supervisor	: E. (Edwer) Veldhuijzen

PREFACE

This report is the result of my graduation research carried out in collaboration with the Eindhoven University of Technology and Grontmij N.V., Part of Sweco. This is my final piece for completing the master track Construction Management and Engineering. The master years and this final research were of great importance for my personal development and I am very proud to graduate from such an innovative and renowned University.

The formulation of my research was not easy. The lack of knowledge of data structures and especially the Ifc data model was a big problem for me. In the first three months I studied these structures and formulated an innovative research on the new IfcAlignment schema. After formulating, the research went smoothly and my knowledge grew. I am very grateful for the opportunity to write a scientific paper of my research to submit to the academic community. My passion for Building Information Modeling is more alive than ever and this research gives me good foundation for my next steps.

I never could produce this level of report without the help of several people. The feedback and support from the University and especially Jakob Beetz and Thomas Krijnen are greatly appreciated. Additionally I would like to thank the engineering company Grontmij for this opportunity and in particular Niels van Beurden and Robert van Eerden, for the feedback and support on process level. On technical level I would like to thank Edwer Veldhuijzen and Christiaan Post, for the support in the data structures and applications.

Last but not least, I would like to thank my family and friends for pushing me towards my goal.

I hope you will enjoy reading and learning from this research as I had.

[illegible]

Luuk Irenëus Hendrikus Wijnholts

CONTENT

COLOPHON 3
PREFACE 5
CONTENT 7
SUMMARY (ENGLISH) 9
SUMMARY (DUTCH) 13
1 INTRODUCTION 17
1.1 PROBLEM DEFINITION 17
1.2 RESEARCH QUESTIONS 17
1.3 RESEARCH DESIGN 18
1.4 EXPECTED RESULTS 19
2 GLOSSARY 21
2.1 ABBREVIATIONS 21
2.2 FIGURES 21
2.3 TABLES 23
3 DATA REQUIREMENT 25
3.1 INFORMATION MODELING OVERVIEW 25
3.1.1 INFORMATION MODELING 25
3.1.2 BIM IN INFRASTRUCTURE 25
3.1.3 POTENTIAL BENEFITS AND BARRIERS 26
3.2 INTERVIEW OUTCOMES 27
3.2.1 BIM IN INFRASTRUCTURE 28
3.2.2 GUIDELINES AND REGULATIONS 29
3.3 MODEL CHECKING 30
3.3.1 TYPES OF MODEL CHECKING & PLATFORMS 30
3.3.2 PROCESS 32
3.3.3 GEOMETRY 34
3.4 INTEROPERABILITY OF DATA 35
3.4.1 INFORMATION EXCHANGE 36
3.4.2 DATA MODELING 37
3.4.3 ROAD DATA MODELS 38
3.4.4 DISCUSSION ON ROAD DATA MODELS 42
4 SCIENTIFIC ARTICLE 47
4.1 INTRODUCTION 48
4.1.1 BIM IN INFRASTRUCTURE 48
4.1.2 VALIDATING COMPLIANCE CHECKING 49
4.2 RELATED WORK 50
4.2.1 TYPES OF MODEL CHECKING 50
4.2.2 RULE CHECKING PLATFORMS 50
4.3 ROAD DATA MODELS 51

4.3.1 QUALITY OF ALIGNMENT MODELS 51
4.3.2 LANDXML 51
4.3.3 ROADXML 52
4.3.4 OKSTRA MODEL 53
4.3.5 IFCALIGNMENT MODEL 54
4.3.6 REVIEW ALIGNMENT MODELS 55
4.4 IFCALIGNMENT CHECKING APPROACH 56
4.4.1 DEVELOPMENT OF RULE-SETS 57
4.4.2 PARSING IFCALIGNMENT FILE 57
4.4.3 CHECK EXECUTION 60
4.4.4 REPORT GENERATION 61
4.5 USE-CASES 61
4.5.1 CLASS 1 – RULES CHECKED WITH ONE VARIABLE OF EXPLICIT DATA 61
4.5.2 CLASS 2 – RULES CHECKED WITH MULTIPLE VARIABLE OF EXPLICIT DATA 62
4.5.3 CLASS 3 – RULES CHECKED WITH GENERATED DATA FROM GEOMETRY 64
4.5.4 CLASS 4 – RULES CHECKED WITH EXTERNAL DATA STRUCTURES 65
4.6 UNIT TESTS 66
4.6.1 LINE_SEGMENT 66
4.6.2 LINE_OF_SIGHT 67
4.7 CONCLUSSION AND DISCUSSION 69
4.7.1 CONCLUSION 69
4.7.2 DISCUSSION 70
4.8 LIMITATIONS AND FURTHER RESEARCH 70
4.8.1 MODEL CHECKING 70
4.8.2 IFCALIGNMENT SCHEMA 70
4.9 REFERENCES 71
5 CONCLUSION AND RECOMMENDATIONS 73
5.1 CONLUSION 73
5.2 RECOMMENDATIONS AND FUTURE RESEARCH 74
6 REFERENCES 77
7 APPENDICES 81
APPENDIX A – INTERVIEW QUESTIONNAIRE 81
APPENDIX B – UNIT TESTS 83
APPENDIX C – NEEDED SOFTWARE 93
APPENDIX B – OVERVIEW ALIGNMENT CHECKER 95
APPENDIX C – FLOWCHART PARSING IFCALIGNMENT 97
APPENDIX D – FLOWCHART CHECK EXECTUTION 99
APPENDIX E – SCRIPT ALIGNMENT CHECKER 101

SUMMARY (ENGLISH)

The growing interest and usage of Building Information Modeling (BIM) in the Architecture, Engineering and Construction (AEC) industry has a positive influence towards the Infrastructure industry, due to the benefits. Literature states many benefits of the usage of Building Information Modeling, such as visualization, interoperability, analysis, clash detection, etc. (Azhar et al., 2007b; Czmoch & Pękala, 2014; Strafaci, 2008; Volk et al., 2014). The forecasted growth of BIM use for Infrastructure is no surprise given the expertise available from the AEC industry, the high level of complexity involved in large Infrastructure projects, the increased use of prefabrication in Infrastructure, and the growing need for greater efficiency and effectiveness on all aspects of Infrastructure projects (Jones & Bernstein, 2012). The growing need for greater efficiency is due to the scarcer financing and the increasing demand for Infrastructure. Therefore the industry develops alternatives for financing and development methods, such as Public Private Partnerships (PPPs). In such alternatives, collaboration is of high importance and Building Information Modeling is recognized as a process that enables collaboration. The findings of (Jones & Bernstein, 2012) confirm the trend that BIM use in the Infrastructure industry, and the extent of that use, lags several years behind the AEC industry.

There are several barriers to overcome to adopt BIM in the Infrastructure. Through thorough literature review, these barriers are structured in three main categories, product, process and people. The conducted interviews in different disciplines of the Infrastructure industry conclude, the category people is the main barriers of adoption. Addressing these barriers, the subject of model checking is proposed for the Infrastructure industry. Model checking affects all three groups; interoperability of models (product), verification of model (process) and expertise / knowledge of disciplines (people).

Guidelines and contract requirements require the industry to check their designs for compliance. Manual construction compliance checking is time-consuming and error-prone, due to a lot of reasons. Reasons are unfamiliarity with or even lack of the guideline expertise knowledge, or being overwhelmed of the amount of guideline text, engineers own way of quality check based on experience or complexity of the regulations. (Nawari, 2012; Zhong et al., 2012). There are four types of model checking:

- *Validating model checking;* check if the model is according specific codes and regulations.
- *Model content checking or pre-checking;* analyze the professional content of a BIM model for a specific use.
- *Guiding model checking;* provide the designer a large set of solutions for a problem to consider.
- *Adaptive model checking;* an object itself, analysis and act on its environment based on with predefined rules.

This research is focused on validating model checking, due to the time-consuming and error-prone process of verifying guidelines and contract requirements. The geometry within the topology is the basis and the end result of an Infrastructural project, which is formed in the Planning phase of the System Engineering process (SE). This early design determines the eventual success and impact of a project in terms of planning, construction, costs and maintenance aspects. Within all the Dutch guidelines for roads, there are three guidelines which apply for the geometric infrastructure of roads, the Richtlijn Ontwerp Autosnelwegen (ROA) 2014, the Handboek Wegontwerp 2013 and the Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom (ASVV) 2012. These guidelines are committed in a contract and therefore have a legal status. Infrastructural projects have to be imbedded in the topological surroundings, wherefore these guidelines are not legally before including in a contract. If necessary, it is possible to adjust some rules of the guidelines in the contract for embedding the design in the environment.

Before developing the checker, a good understanding of the different road data models is obtained. Building Information Modeling software applications should allow for the import of relevant data (for creating and editing a design) and export of data in various standards (to support integration with other application and workflows) (Eastman et al., 2011). This can be done by staying within one software vendor's products or to use software products which can exchange data between different software applications, using open road data standards. These open standards provide a mechanism for interoperability among applications with different internal standards. Several standards developed over the years, the most used and maintained are LandXML, RoadXML and OKSTRA. LandXML is worldwide the most applied exchange standard. Next to these, the Industry Foundation Classes (IFC) (steered by the buildingSMART organization) provides a standardized product model for the AEC industry which is highly adopted and is updated with an alignment model. The alignment model is the highest level of abstraction of linear projects, which is the basis of the geometry of road design. It defining the course of

the road in horizontal and vertical plane and is specified by the superposition of two two-dimensional curves, the horizontal and the vertical alignment. Usually, the horizontal alignment consists of lines, arcs and transition curves which defines the course of an alignment in the XY plane. The vertical alignment consists of lines, parabola arcs and circular arcs which defines the corresponding Z-coordinates as a function of the length of the horizontal alignment curve up to a certain point (Amann et al., 2014).

An classification of the geometric guidelines is made to determine which rules applies for the alignment model and which have to have further development. The classification consist of horizontal alignment, vertical alignment, cross sections, discontinuity and line of sight. For the rules in the classes cross sections and discontinuity additional information is needed, so these are not within scope. The development of the Automated Geometry Checker contains four steps to compute the geometry validation: (1) RuleSet interpretation of rules from the general guidelines and contract requirements, (2) parsing the IfcAlignment file, based on IfcAlignment data schema (Technical Universität München, 2015), (3) the execution of the checks and (4) the reporting in the BIM Collaboration Format (BCF) (Stangeland, 2011). For this research the first three steps are implemented (Figure 1). Step 2, parsing the IfcAlignment file is done by the conversion of a LandXML or OKSTRA file in a IfcAlignment file, by using the Open Infra Platform of the Technical University of München.

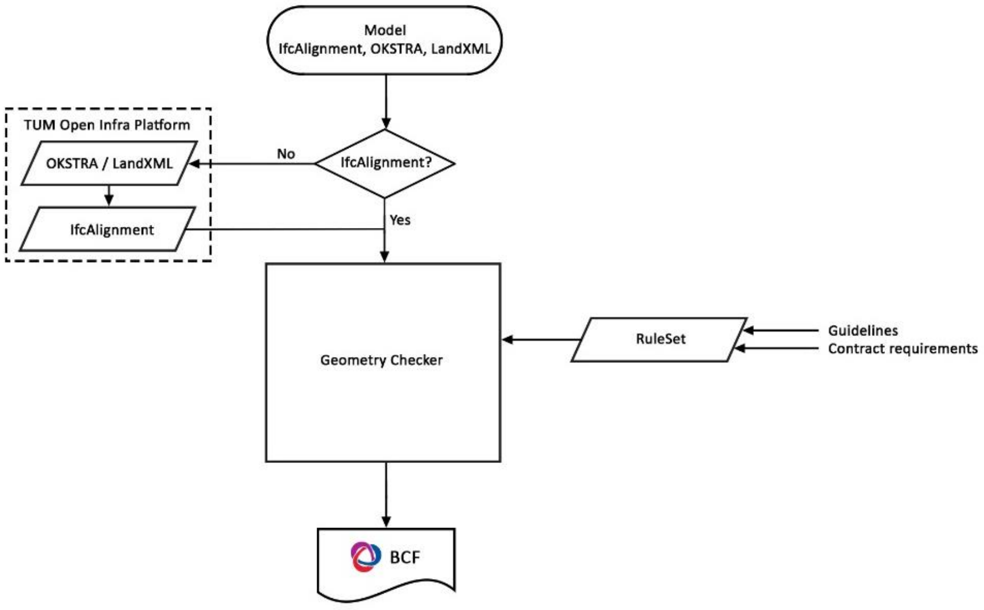


Figure 1: General architecture Automated Geometry Checker

The rules in the classification are written in human language and have to be formally interpreted and translated into computer processable code. Considering the computation into formal code, the fundamentals of data checking is reading the attribute values of this model. This explicit data checking is not enough to fulfill all rules, therefore implicit data checks are required. Implicit data is data which is generated from the explicit data. This data can be computed from the geometry of the alignment model and therefore can fulfill complex rules. This research classifies four types of rules, from the complexity of the rules processing. For each class a use-case is presented.

Class 1 – Rules checked with one explicit attribute;
Class 2 – Rules checked with multiple explicit attributes;
Class 3 – Rules checked with computed data from geometry;
Class 4 – Rules checked with external data structures.

One of the important advantages of Building Information Modeling is the visualization and in model checking the visual feedback. To create this visual feedback, the IfcAlignment model is displayed in a viewer containing the 3D view, the 2D horizontal view and the 2D vertical view. When the IfcAlignment file is parsed and displayed in the viewer, the checking of the ruleset can be processed. This is done by the Checker and the RuleSet as illustrated in Figure 1. The Checker defines what and how to display and the RuleSet checks the data for errors. After the checks are executed, every issue should be captured in a BIM Collaboration Format. This is not conducted in this research, due demarcation of the research.

The research's main objective is to explore how the validation of official regulations of roads can be automated in the Planning phase of the System Engineering process, based on open standards and software. A prototypical implementation of the Automated Geometry Checker is proposed. A big part of the requirements of the classes horizontal alignment, vertical alignment and line of sight can directly be checked. However, the IfcAlignment schema does not include design speed and superelavation. These properties are of high importance for each element in the geometrics of road design, due to several rules in the guidelines. These attributes are specifically for alignment models and including these in the data schema should be considered. Next to these technical issues, there are some legal issues to address, such as the interpretation of guidelines and contract requirements into formal code. The guidelines and requirements are written in human language and have a legal status and therefore the interpretation into formal code is vulnerable for legal issues. The responsibility of formalizing these documents relies on software developers, or directly into computer processable code or into formal code and then translated into computer processable code. Another legal issue is the derivation of data. When a model requires complex calculations or analyses on derivate data, or the application derives new data itself, this can lead to vulnerability and legal risks.

Overall, the IFC standard represents one of the largest scale and most mature efforts to standardize facilities design and construction data. The IFC model defines a multilayer, integrated schema that represents the structure and organization of data in the form of a class hierarchy. The hierarchy covers the core project information such as building and elements (infrastructure) geometry, materials, properties of products, project costs, schedules, and organizations (Halfawy et al., 2006). The IfcAlignment is the first step into the Infrastructure industry and there are more extensions in development, such as IfcRoad and IfcBridge. Many applications in the building industry have implemented this data schema and this is promising for the Infrastructure industry. This prototypical implementation could be a starting point in Infrastructure model checking.

SUMMARY (DUTCH)

De groeiend interesse en gebruik van Building Information Modeling (BIM) in de bouw heeft, door de voordelen een positieve invloed op de Infrastructurele industrie. In de literatuur zijn vele voordelen te vinden van het gebruik van Building information Modeling, zoals visualisatie, interoperabiliteit, analyses, clash detectie, etc. (Azhar et al., 2007b; Czmoch & Pękala, 2014; Strafaci, 2008; Volk et al., 2014). De voorspelde groei van BIM gebruik in de Infrastructuur is geen verassing door verschillende facetten: de expertise beschikbaar vanuit de bouw, de hoge complexiteit van grote Infrastructurele projecten, de toename van prefab elementen, en de groeiende noodzaak voor efficiëntie en effectiviteit binnen de Infrastructuur (Jones & Bernstein, 2012). De noodzaak van efficiëntie komt door de schaarser wordende financieringen en de groeiende projecten. De Infrastructurele industrie ontwikkeld hiervoor alternatieve methoden van financiering, zoals Publieke Private Samenwerkingen (PPS). In zulke methoden is samenwerking van groot belang en Building Information Modeling kan hierbij helpen. De bevindingen van (Jones & Bernstein, 2012) bevestigen de trend dat BIM binnen de Infrastructuur een aantal jaren achter de bouw aan loopt.

Er zijn nog een aantal barrières die overwonnen moeten worden voordat BIM geadopteerd kan worden in de Infrastructuur. Door grondig literatuur onderzoek kunnen deze barrières in drie categorieën gestructureerd worden, product, proces en mensen. Uit de afgenomen interviews binnen de verschillende disciplines van Infrastructuur kan worden geconcludeerd dat de categorie mensen de grootste barrière is. Om deze barrières te tackelen, stelt dit onderzoek model checking voor de Infrastructuur voor. Model checking pakt alle drie de categorieën aan; interoperabiliteit van modellen (product), verificatie van modellen (proces) en de expertise en kennis van de disciplines (mensen).

De Infrastructurele industrie moet het ontwerp van projecten checken aan richtlijnen en contract eisen. Handmatig checken van deze richtlijnen en eisen is erg tijdrovend en erg fout-gevoelig, door meerdere redenen. Reden hiervoor zijn onbekendheid of zelf gebrek aan kennis over de richtlijn, overspoeld worden door de hoeveelheid richtlijnen, ingenieurs eigen manier van checken gebaseerd op ervaring, complexiteit van de richtlijnen, etc. (Nawari, 2012; Zhong et al., 2012). Er zijn vier soorten model checking:

- *Validatie model checking;* checken van modellen volgens specifieke wetten en richtlijnen.
- *Model inhoud checking;* analyse van de inhoud van het model voor een specifieke functie.
- *Begeleidend model checking;* om de ontwerper te begeleiden in het overwegen van oplossingen.
- *Adaptieve model checking;* analyse van een object, welke zich aanpast aan de omgeving gebaseerd op voorgeselecteerde regels.

Dit onderzoek is gefocust op het validatie model checking, door het tijdrovende en foutgevoelige proces van verifiëren van richtlijnen en contract eisen. De geometrie in de topologie is de basis en het eind resultaat van een Infrastructureel project, welke is gevormd in de Plan fase van het System Engineering proces (SE). Het ontwerp in de Plan fase bepaald het succes en de impact van een project op planning, realisatie, kosten en onderhoud aspecten. Voor wegontwerp in Nederland zijn drie richtlijnen voor geometrie van belang, de Richtlijn Ontwerp Autosnelwegen (ROA) 2014, het Handboek Wegontwerp 2013 en de Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom (ASVV) 2012. De richtlijnen zijn opgenomen in een contract van een Infrastructureel project en verkrijgen daarmee juridische status. Infrastructuur projecten moeten geïmplementeerd worden in de omgeving en daarom hebben deze richtlijnen nog geen juridische status voordat deze in het contract zijn opgenomen. Wanneer het nodig is kan er een regel uit de richtlijnen veranderd worden en los opgenomen worden in het contract, om zo goed ingepast te kunnen worden in de omgeving.

Voordat de Automated Geometry Checker ontwikkeld kan worden moet er een goede kennis opgedaan worden van de verschillende weg data modellen. Building Information Modeling software moet relevante data kunnen importeren (voor het creëren en aanpassen van een ontwerp) en exporteren in verschillende data standaarden (om integratie tussen verschillende applicaties te ondersteunen) (Eastman et al., 2011). Dit kan door binnen één software ontwikkelaars producten te blijven of om gebruik te maken van data standaarden die tussen verschillende software ontwikkelaars gebruikt kunnen worden, open data standaarden voor wegontwerp. Er zijn meerdere standaarden ontwikkeld over de jaren. LandXML, RoadXML en OKSTRA zijn het meest gebruikt en het meest onderhouden. LandXML is wereldwijd de meest gebruikte open data standaard. Naast deze standaarden voor wegontwerp is de Industry Foundation Classes (IFC) (gestuurd door de buildingSMART organisatie) een veel gebruikte en gewaardeerde open data standaard voor de bouw. De IFC data standaard is geüpdatet met een

alignement model, IfcAlignment. Het alignement model is het hoogste level van abstractie voor ontwerp van lineaire projecten en is de basis van de geometrie voor wegontwerp. Het definieert het verloop van de weg in het horizontale en het verticale vlak en is gespecificeerd door de positionering van twee tweedimensionale bogen, het horizontale alignement en het verticale alignement. Normaal bestaat het horizontale alignement uit lijnen, cirkelbogen en overgangsbogen welke het verloop in het XY vlak bepalen. Het verticale alignement bestaat uit lijnen, parabolen en cirkelbogen welke de corresponderende coördinaat in de Z-richting geeft (Amann et al., 2014).

Er is een classificatie gemaakt om vast te stellen welke regels in de richtlijnen op het alignement model van toepassing zijn en voor welke regels een uitgebreider model nodig is. De classificatie bestaat uit regels voor het horizontale alignement, het verticale alignement, dwarsdoorsneden, discontinuïteit en zichtafstanden. Voor de regels in de classes dwarsdoorsneden en discontinuïteit is meer informatie nodig dan alleen het alignement model. Het ontwikkelen van de Automated Geometry Checker bevat vier stappen: (1) interpretatie van de regels van de richtlijnen en contract eisen, (2) parsing het IfcAlignment bestand, gebaseerd op het IfcAlignment schema (Technical Universität München, 2015), (3) het uitvoeren van de checks en (4) de verslaglegging van de problemen in het BIM Collaboration Format (BCF) (Stangeland, 2011). In dit onderzoek zijn de eerste drie stappen uitgevoerd (Figure 2). Stap 2 wordt gedaan met behulp van de mapping tussen LandXML of OKSTRA naar IfcAlignment door het Open Infra Platform van de Technische Universiteit van München.

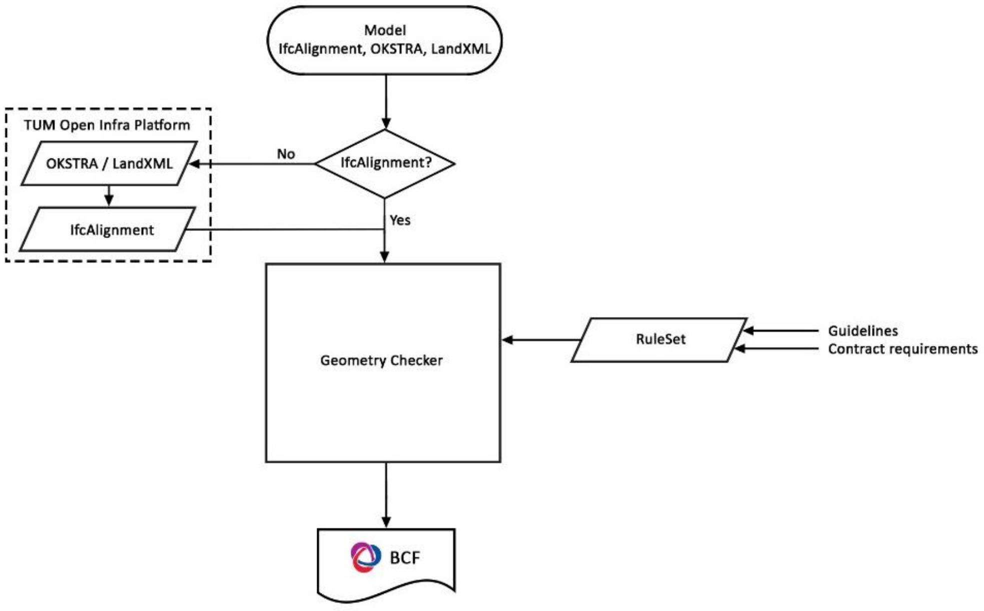


Figure 2: Algemeen architectuur Automated Geometry Checker

De regels, onderverdeeld in de classificatie, zijn geschreven in menselijke taal, zoals tekst, tabellen, etc. Dit moet formeel geïnterpreteerd en vertaald worden in computer leesbare taal. Het fundamentele van deze vertaalslag is het lezen van de attribuut waarden. Deze expliciete informatie in de attributen is niet voldoende om alle checks uit te voeren, hiervoor zijn impliciete data checks nodig. Impliciete data is data wat is gegenereerd aan expliciete informatie opgeslagen in de attributen. Deze data kan gegenereerd worden aan de hand van de geometrie van een alignement model en daarmee kunnen complexere regels worden gecheckt. Dit onderzoek classificeert vier typen regels, oplopend in complexiteit. Van iedere klasse is een use-case gepresenteerd.

Klasse 1 – Regels checkt door één expliciete attribuut;
Klasse 2 – Regels checkt door meerdere expliciete attributen;
Klasse 3 – Regels checkt door gegenereerde data van de geometrie;
Klasse 4 – Regels checkt door externe data structuren.

Eén van de belangrijkste voordelen van Building Information Modeling is de visualisatie en in model checking is dit ook het geval, de visuele feedback. Om deze visuele feedback te creëren wordt het alignement model geparsed in een viewer. Deze viewer bevat een 3D view, een 2D horizontale view en het 2D verticale view. Wanneer het alignement model is geparsed kunnen de checks uitgevoerd worden. Dit wordt gedaan door de Checker en de RuleSet zoals te zien is in Figure 2. De Checker bepaald wat en hoe het model laten zien wordt in de viewer en de RuleSet checkt het model of dit correct is. Wanneer de checks gedaan zijn, zouden de problemen vastgelegd moeten worden in het BIM Collaboration Format. Dit is niet opgenomen in dit onderzoek wegens afbakening.

Het belangrijkste onderdeel van dit onderzoek is het onderzoeken hoe het validatie proces van richtlijnen en contract eisen in de Plan fase geautomatiseerd kan worden. Dit gebaseerd op open standaarden en software. Een prototypische implementatie van een Automated Geometry Checker is gepresenteerd. Een groot deel van de classes horizontale alignement, verticale alignement en zichtafstanden kan hierdoor direct worden gecheckt. Het IfcAlignment schema mist wel de eigenschappen Ontwerpsnelheid en Verkanting. Deze eigenschappen zijn erg belangrijk voor ieder segment in het alignement model door meerdere regels in richtlijnen. Deze eigenschappen zijn specifiek voor het alignement model en het implementeren hiervan moet overwogen worden. Naast deze technische problemen zijn er ook een aantal juridische problemen, zoals het interpreteren van de richtlijnen en contract eisen in formele computer leesbare code. De richtlijnen zijn geschreven in menselijke taal en hebben een juridische status, waardoor de interpretatie gevoelig is voor juridische problemen. De verantwoordelijkheid hiervan ligt bij de software ontwikkelaars. Het genereren van data kan ook juridische problemen opleveren.

In het algemeen is de IFC standaard de meest geïmplementeerde en volwassen standaard voor data in de bouw. Het IFC model definieert meerdere lagen, geïntegreerde data schema dat de structuur en organisatie van dat hiërarchisch representeert. De hiërarchie bevat de kern van de project informatie, zoals de bouwelementen, (infrastructurele) geometrie, materialen en eigenschappen van producten, kosten, planningen, etc. (Halfawy et al., 2006). De IfcAlignement is de eerste stap in de Infrastructuur en er zijn meerdere innovaties in ontwikkeling, zoals IfcRoad en IfcBridge. Vele applicaties in de bouw hebben het IFC schema al geïmplementeerd en dit is een veelbelovend uitgangspunt voor de Infrastructuur. Deze prototypische implementatie kan een start zijn voor model checking in de Infrastructuur.

1 INTRODUCTION

As Building Information Modeling (BIM) has become standard practice in the Architecture, Engineering and Construction (AEC) industry, it is still in the early stages of adoption for the Infrastructure. The widely acknowledged economic an environmental benefits of BIM in the AEC industry (Eastman et al., 2011) has caught the attention of the Infrastructure industry.

As BIM tools become more familiar, models become more complex and detailed. It is no longer practical for users to rely on visual inspection to ensure the models are of good quality and adhere to requirements. Therefore automated rule checking has been identified as potentially providing significant value to the industry (Solihin & Eastman, 2015). This research will provide a methodology, and a prototypical implementation of an Automated Geometry Checker in the Planning phase of the System Engineering process, based on open source.

1.1 PROBLEM DEFINITION

Guidelines, together with the requirements of a contract, play a major role in assuring the quality within the Infrastructural engineering artefacts. It is of high importance to inspect the construction process according to these guidelines and requirements. Manual construction quality compliance checking is a time-consuming and error-prone, due to a lot of reasons. Examples are unfamiliarity with or even lack of the guideline expertise knowledge, or being overwhelmed of the amount of guideline text, engineers own way of quality check based on experience or complexity of the regulations. (Nawari, 2012; Zhong et al., 2012).

The need for computerizing the construction guidelines and automating the compliance checking is becoming more critical. The application of such automated rule checks would reduce quality inspections errors, consequently improve quality compliance and reduce violations to the guidelines and requirements. The Dutch Directorate General for Public Works and Water Management stated that from the moment the first line is drawn onto a map, there have to be compliance checking according the construction guidelines to avoid problems in a later stage. Next to these compliance checking of construction guidelines, there is also a set of requirements within the contract. These requirements have a legal status and also should be checked. Some of the compliance checks of construction guidelines and contract requirements should be checked in an earlier stage than other, due to the specifics of these guidelines and requirements.

Within the Infrastructure industry an alignment is the baseline for further development of the road design. The alignment provides the course of the road in the horizontal and vertical plane. There are several rules in the guidelines applicable on solely the alignment model. The Industry Foundation Classes (IFC) is a data model for the Architecture, Engineering and Construction (AEC) industry and has developed an alignment model, IfcAlignment. The manual compliance checks are time-consuming and error-prone, so the need for computerized compliance checks is high. Therefore the data has to be analyzed, if the information can be directly extracted from the model or if there are any calculations and rationalizations necessary. If there are compliance rules, which cannot be extracted from the IfcAlignment, it should be possible to appoint these to other IFC related extensions of the future. After all data is analyzed and the code is generated, this has to be formalized and transformed in computable code to other software.

1.2 RESEARCH QUESTIONS

Based on the problem area, this section outline the research questions of the proposed research. The main research question is:

> How can the validation of guidelines for roads be automated in the Planning phase of the System Engineering process, based on open standards and software?

The main research question can be divided into a number of sub-questions in two sections: Guidelines & contract requirements and Data & Coding.

Guidelines & contract requirements:

1. How are the guidelines and requirements tested at this moment?
2. What are the guidelines and requirements and how can these be classified?
3. Which requirements can be extracted from the Design and Build contracts, and how can these be classified?
4. In which phase should these requirements be tested?

Data & Coding

5. Which requirements can be directly checked based on IfcAlignment, and for which requirements are further processing steps such as calculations, reasoning or inferences necessary?
6. What requirement cannot be extracted from IfcAlignment, and where should these be included within future versions of the IFC model specification and its extensions such as IfcRoad, IfcBridge?
7. What calculations and inferences of the geometry are necessary to check these requirements?
8. How can these requirements be formalized and captured in an interoperable format?

1.3 RESEARCH DESIGN

This research is conducted in three phases, the theoretical research, the empirical research and the design & development phase. There are three methods combined to ensure a solid end result, a literature study, qualitative semi-structured interviews and the development of the Automated Geometry Checker. This research overview is explained below and shown in Figure 3.

Starting with the literature study for the Automated Geometry Checker, general information about the current BIM adoption within the Infrastructure, road construction and road guidelines is needed. After the general information the Design and Build contract is analyzed to check for specific requirement. When all guidelines and requirements are analyzed, a part of the theoretical model is been setup. This model consists of a classification of all guidelines. After this, the literature study is focused on rule checking and human and formal languages. Therefore the IfcAlignment model is analyzed. From this literature the second part of the theoretical model is generated, the setup of the automated rule checker. This answers research questions 1, 2 and 3.

After the literature study, an analysis is needed to make a diagnosis if the model is correctly setup. This is done by semi-structured interviews. Semi-structured is chosen above structured, because it will give the opportunity to gather systematic information about the guidelines and requirements, while also allowing some exploration when new issues emerge (Wilson, 2014). The interviews provide insight in the specifics of the requirements of the Design and Build contract and in what phase what requirements is checked. The data of the interviews is used to check the theoretical model and checks the answers of research questions 1, 2 and 3 and answers question 4.

The last phase of this research is the design & development phase of the stand-alone Automated Geometry Checker. Therefore an extension of IfcOpenShell is made to include the new schema. After this the classification of the guidelines and requirements and the core logic of the checker is developed. The checker will first focus on the pre-checking and preconditions of the data, this is the needed data in the specific models. After this, an algorithm is set up to develop the classified ruleset into a stand-alone checker and a small part of the geometry is parsed and displayed in a viewer. The result of the checker is validated by comparing the automated checking results with manual test. When there is enough development time, the formalization into an Application Programming Interface is done. This defines the functionalities that are independent of their respective implementation and provides the code in an interoperable form. This answers research questions 5, 6, 7 and 8.

1.4 EXPECTED RESULTS

The final results of this research is a literature review with a theoretical model and the stand alone Automated Geometry Checker of the IfcAlignment data, consisting of a classification of the regulations and requirements, a core logic of the checker and an research paper to present to the academic community.

The literature study includes the current knowledge to get a solid understanding of the topics of research. Scientific articles, book chapters, presentations, etc., are studied to conduct the literature, which will use the most important key words 'Building Information Modeling, BIM, IFC, IfcAlignment, System Engineering, Infrastructure, Automated Geometry Checking, Rule checking, Model Checking'.

This literature study is transformed into a theoretical model, consisting of theoretical constructs (latent variables), causal relationships and measures (observed variables). The theoretical model is generally developed based on analysis of the literature and may be modified and build on as a result of the research (Moody, 2002). This first part of the theoretical model will consists of a classification of all guidelines and the second part of the model will be the setup of the automated rule checker.

The stand-alone Automated Geometry Checker consist of three parts, namely the classification, the core logic and the display. Before these parts can be set up, IfcOpenShell has to be extend with the IfcAlignment schema. When this is done, the classification of the guidelines and the requirements of the Design and Build contract are generated in natural language and formal language and the core logic is coded. This is a stand-alone application.

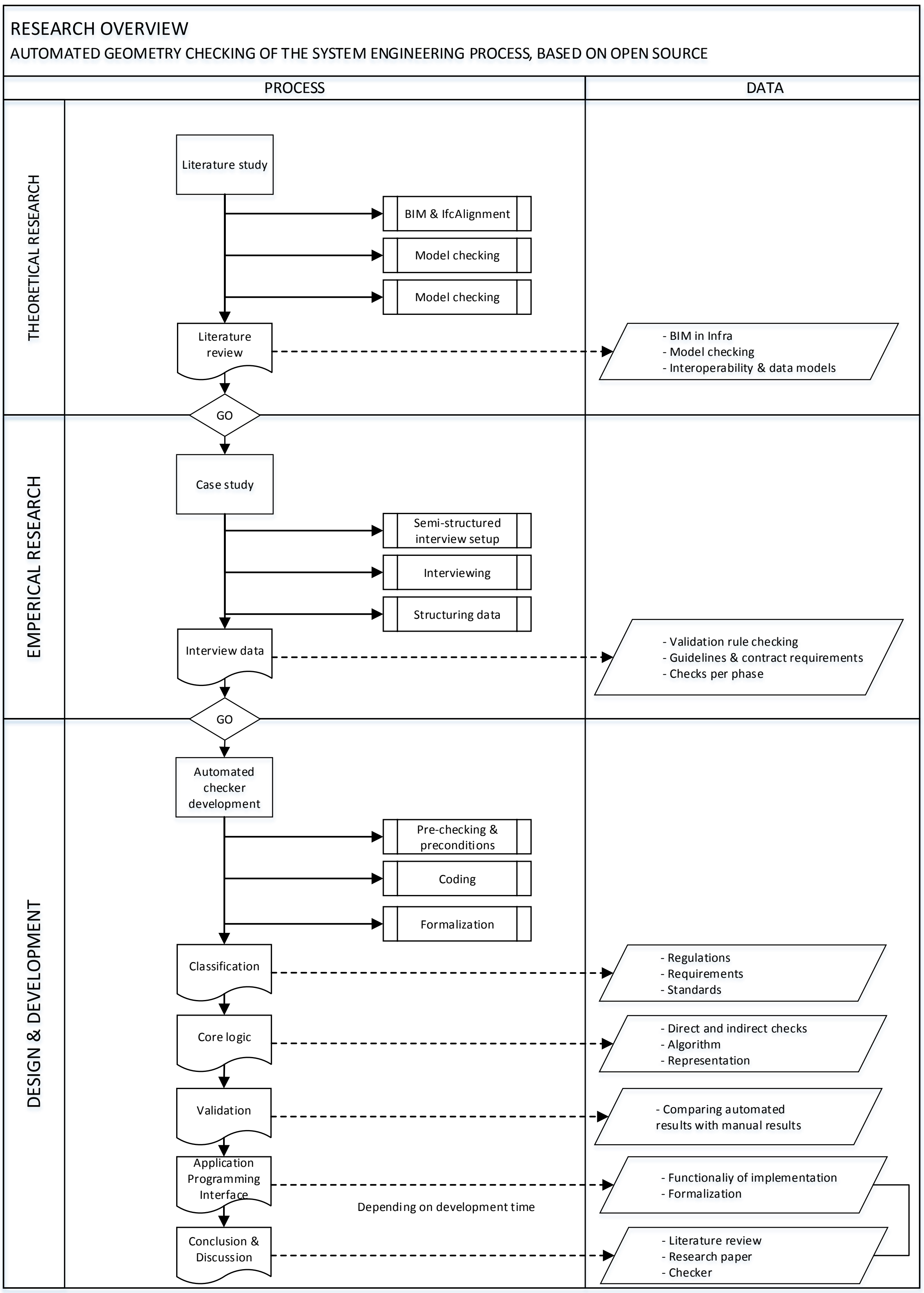


Figure 3: Research overview

2 GLOSSARY

2.1 ABBREVIATIONS

2D	: Two dimensional
3D	: Three dimensional
AEC	: Architecture, Engineering and Construction
ASVV	: Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom 2012
API	: Application Programming Interface
BCF	: BIM Collaboration Format
CAD	: Computer Aided Design
EDM	: Jotne EDModelChecker
GIS	: Geographic Information Systems
GUID	: Global Unique Identifier
HWO	: Handboek WegOntwerp 2013
IAI	: International Alliance for Interoperability
IFC	: Industry Foundation Class
IPD	: Integrated Project Delivery
ISO	: International Organization for Standardizations
MV	: Model View
MVD	: Model View Definition
NBIS	: National BIM Standard
OGC	: Open Geospatial Consortium
OIP	: Open Infra Platform
PPP	: Public Private Partnership
PVI	: Point of Vertical Intersections
RE	: Reverse Engineering
ROA	: Richtlijn Ontwerp Autosnelwegen 2014
SE	: System Engineering
SMC	: Solibri Model Checker
TUM	: Technical University of München
XML	: Extensible Markup Language

2.2 FIGURES

Figure 1: General architecture Automated Geometry Checker 10
Figure 2: Algemeen architectuur Automated Geometry Checker 14
Figure 3: Research overview 20
Figure 4: MacLeamy Curve, Effect / Cost / Effort (Integrated Project Delivery, 2004) 26
Figure 5: Different industries of the interviewees 28
Figure 6: Requirements checked in System Engineering phases 29
Figure 7: The four stages of a rule-based checking system with their aspects (Eastman et al., 2009) 34
Figure 8: Horizontal alignment segments 35
Figure 9: Vertical alignment segments 35
Figure 10: Graphical and Textual representation of entity Person (TU/e, 2014) 37
Figure 11: LandXML data model 39
Figure 12: RoadXML data model 40

Figure 13: OKSTRA data model 40
Figure 14: IfcAlignment data model 41
Figure 15: Entities and general attributes IfcAlignment data model 42
Figure 16: Smooth transitions between lines and arcs (Amann et al., 2014) 43
Figure 17: Attributes of transition curves 'Spiral' type (LandXML.org, 2015) 44
Figure 18: LandXML data model 52
Figure 19: RoadXML data model 53
Figure 20: OKSTRA data model 53
Figure 21: IfcAlignment data model 54
Figure 22: Entities and general attributes IfcAlignment data model 55
Figure 23: General architecture Automated Geometry Checker 56
Figure 24: Parsed IfcAlignment file in 3D view, 2D horizontal view and 2D vertical view 58
Figure 25: IfcAlignment model with surroundings 60
Figure 26: Checked curves (green = passed, red = failed) 62
Figure 27: Line segment with adjoined curve segments with same and opposite directions 63
Figure 28: Stepwise decreasing design speed with associated curve lengths 63
Figure 29: Checked HeadRoom with two intersections 65
Figure 30: Alignment with sound barriers in the line of sight 66
Figure 31: Checked line segments with adjoined arcs with the same or opposite directions 67
Figure 32: Calculation of surface of line of sight 68
Figure 33: Checked line of sight with one object clash 68
Figure 34: Ontology of types of model checking (Hjelseth & Nisbet, 2010) 69
Figure 35: Rounding error 1 : 0.00085 shown in the Checker and schematic 71
Figure 36: Guidelines checked in different phases of the SE process 74
Figure 37: Curve length of 100 meters at 120 km/h 83
Figure 38: Curve length of 99 meters at 120 km/h 83
Figure 39: Curve length of 60 meters at 90 km/h 84
Figure 40: Curve length of 60 meters at 70 km/h 84
Figure 41: Passed line segments at 120 km/h 85
Figure 42: Failed line segments at 120 km/h 85
Figure 43: Failed and passed line segments at 90 km/h 86
Figure 44: Passed and failed line segments at 70 km/h 86
Figure 45: Failed and passed line segments with opposite direction circular arcs at 90 km/h 86
Figure 46: Failed and passed line segments with opposite direction circular arcs at 70 km/h 87
Figure 47: Mixed passed and failed line segments with same and opposite direction circular arcs at 120 km/h 87
Figure 48: Top view unit test HeadRoom free space 88
Figure 49: Passed HeadRoom free space of 5 meters 88
Figure 50: Failed HeadRoom free space of 4.5 meters 88
Figure 51: Top view unit test HeadRoom free spaces 89
Figure 52: Passed HeadRoom free spaces of 5 meters 89
Figure 53: Failed HeadRoom free spaces of 4.5 meters 89
Figure 54: Failed and passed HeadRoom free spaces of 4.5 and 5 meters 90
Figure 55: Line of sight of 40 meters 90
Figure 56: Line of sight of 80 meters 91
Figure 57: Line of sight all passed 91
Figure 58: Line of sight with one shape failed 91
Figure 59: Line of sight with two shapes failed 92
Figure 60: Line of sight with all shapes failed 92

2.3 TABLES

Table 1: Overview of rule-checking platform (Eastman et al., 2009) 32
Table 2: Graphical versus Textual structures (Heinen, 2015) 37
Table 3: Lines of Code data models 43
Table 4: Support of different transition curves across different standards (Amann et al., 2014) 43
Table 5: Rule categories of government documents 49
Table 6: Lines of Code data models 55
Table 7: Support of different transition curves across different standards (Amann et al., 2014) 55
Table 8: Minimal curve length (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015) 61
Table 9: Segment length presented as the ROA (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015) 66
Table 10: Minimal curve length (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015) 83
Table 11: Segment length presented as the ROA (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015) 84

3 DATA REQUIREMENT

3.1 INFORMATION MODELING OVERVIEW

The level of information within projects has rapidly grown in the last decades, from 2D drawings on paper to 3D models. This section presents an overview how to handle this information exchange between parties and gives the potential benefits and barriers for adopting Building Information Modeling in the Infrastructure.

3.1.1 INFORMATION MODELING

Due to the widely acknowledged economic and environmental benefits of Building Information Modeling (BIM), it has become standard practice in the Architecture, Engineering and Construction (AEC) industry. The development of Building Information Modeling started in the 1970s, based on the first Computer Aided Design software which replaced the drawings on paper. This Computer Aided Design software generates digital files, consisting of vectors, associated line-types and layer identifications. With the introduction of 3D modeling, more information is added to these files and the CAD software became more intelligent. A building model can be described by its content (what objects it describes) or its capabilities (what kinds of information requirements it can support). The latter approach is preferable, because it defines what you can do with the model rather than how the database is constructed (which will vary with each implementation) (Eastman et al., 2011). The National Building Information Modeling Standard (NBIS) vision for BIM is 'an improved planning, design, construction, operation, and maintenance process using a standardized machine-readable information model for each facility, new or old, which contains all appropriate information created or gathered about that facility in a format useable by all throughout its lifecycle'.

The major advantage of the 3D Building Information Model is the visualization of these drawings. Where a 3D model only consist out line-types, a BIM consist of objects with their geometric dimension and these object can be enriched by 'Information'. The object consist Architectural, Structural, HVAC, Electrical, and all sorts of installation information. This enrichment of information makes the model intelligent which have major benefits. These benefits will be elaborated in chapter 3.1.3 Potential benefits and barriers.

3.1.2 BIM IN INFRASTRUCTURE

The growing interest and usage of using Building Information Modeling in the AEC industry has a positive influence towards the Infrastructure industry, due to the benefits. A survey executed by the Royal Institution of Chartered Surveyors (2013) stated that BIM is suitable for larger and more complex projects. The advanced features of Building Information Modeling software have contributed to a shift in the way IT can be used in the industries, going beyond simple visual representation of the building to an integrated semantic product and process model (Laakso & Kiviniemi, 2012). BIM for the Infrastructure industry is just beginning and a plethora of terms have been created for BIM for Infrastructure, such as Civil BIM or CIM, virtual design and construction (VDC) and Heavy BIM, but all refer to the same capability to create data-rich models in three or more dimensions that facilitate better design, enhance construction efficiency and enable collaboration.

The forecasted growth of BIM use for Infrastructure is no surprise given the expertise available from the AEC industry, the high level of complexity involved in large Infrastructure projects, the increased use of prefabrication in Infrastructure, and the growing need for greater efficiency and effectiveness on all aspects of Infrastructure projects (Jones & Bernstein, 2012). The growing need for greater efficiency is due to the scarcer financing and the increasing demand for Infrastructure. Therefore the industry develops alternatives for financing and development methods, such as Public Private Partnerships (PPPs). In such alternatives, collaboration is of high importance and Building Information Modeling is recognized as a process that enables collaboration. The findings of (Jones & Bernstein, 2012) confirm the trend that BIM use in the Infrastructure industry, and the extent of that use, lags several years behind the AEC industry.

3.1.3 POTENTIAL BENEFITS AND BARRIERS

The BIM Handbook (Eastman et al., 2011) states several benefits of implementing Building Information Modeling in the AEC industry. Most of these benefits can be extrapolated to the Infrastructure industry (Jones & Bernstein, 2012; Royal Institution of Chartered Surveyors, 2013). One of the main benefits of BIM for the Infrastructure industry is better designs and increased efficiency and productivity. The MacLeamy Curve of (Integrated Project Delivery, 2004), shown in Figure 4 states the effect, cost and effort of the BIM workflow in the Design phase in contrary to the drafting-centric workflow in the Construction Documentation phase. This BIM workflow essentially facilitates collaboration between the architect and the engineers. Development of the design is time consuming and expensive when using traditional design methods (Czmoch & Pękala, 2014).

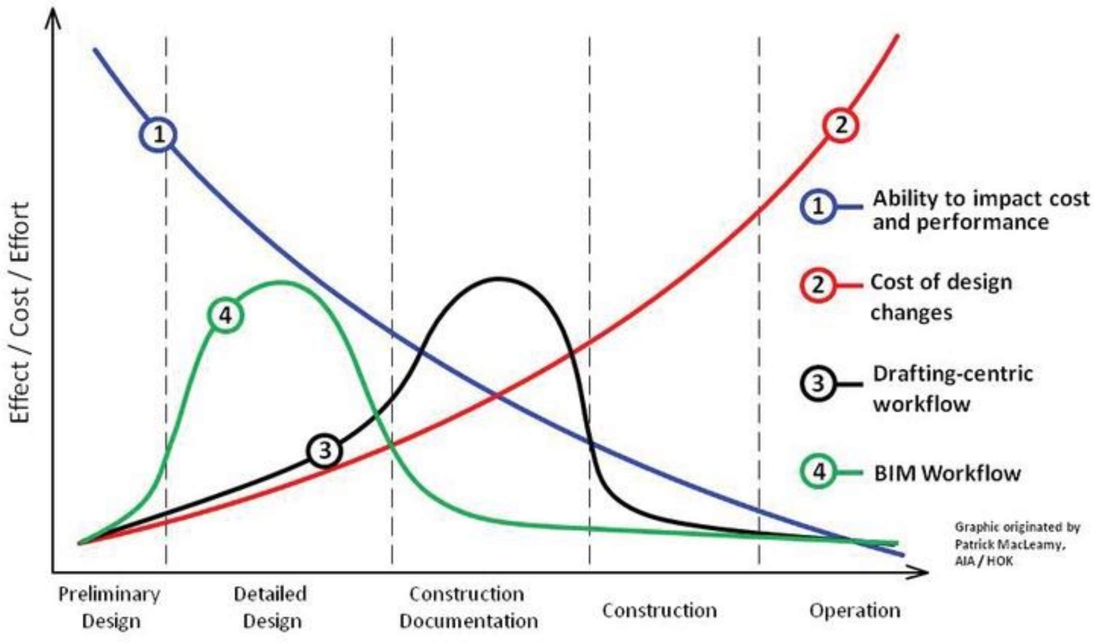


Figure 4: MacLeamy Curve, Effect / Cost / Effort (Integrated Project Delivery, 2004)

Literate contains many benefits of the usage of Building Information Modeling and in summary the following are found (Azhar et al., 2007b; Czmoch & Pękala, 2014; Strafaci, 2008; Volk et al., 2014):

- Visualization;
- Design consistency;
- Interoperability;
- Analysis;
- Clash detection;
- Simulation;
- Planning;
- Cost estimations;
- Monitoring;
- Life cycle data.

Countering the potential benefits of BIM to project is the challenges that need to be overcome if effective multi-disciplinary collaborative team working, supported by the optimal use of BIM, is to be achieved. Not least the changing roles of key parties, such as clients, architects, contractors, sub-contractors and suppliers, the new contractual relationships and the re-engineered collaborative processes (Bryde et al., 2013).

Given the slow adoption rate in the Infrastructure industry, the challenges must be analyzed. One of the biggest challenges is the lack of application between BIM systems and 3rd party application of choice (Royal Institution of Chartered Surveyors, 2013). This is one of the three major technical challenges to adopt BIM in the industry. There are three major technical issues which can be grouped into three categories (Azhar et al., 2007a):

1. The need for well-defined transactional construction process models to eliminate data interoperability issues,
2. The requirements that digital design data be computable, and
3. The need for well-developed practical strategies for the purposeful exchange and integration of meaningful information among the BIM model components.

Next to these technical challenges the barriers can be grouped into three main categories, product, process and people (Ning et al., 2008). The literature states a lot of challenges, which are added to these groups (Eadie et al., 2013; Heinen, 2015; Muz, 2014; Ning et al., 2008; Royal Institution of Chartered Surveyors, 2013; Volk et al., 2014).

Product:

- Poor data Interoperability (data import/ export issues);
- Lack of application interfaces between BIM systems and 3rd party applications of choice;
- Data safety;
- Liability in open and closed software platforms.

Process:

- Lack of investment cost in new software and education/training;
- Lack of project finance to support translation of 2D drawings into BIM models ;
- Lack of standards;
- BIM use-cases limited to design construction project phases;
- Lack of immediate benefits of projects.

People:

- Lack of expertise;
- Lack of demand;
- Cultural resistance;
- Lack of government lead/direction;
- Uncertainties over ownership of data and responsibilities.

3.2 INTERVIEW OUTCOMES

To validate this research, seven interviews have been conducted to determine the potentials and the barriers of BIM and the need of model checking in the Infrastructure industry. The interviews are from different industries within the Infrastructure, as shown in Figure 5.

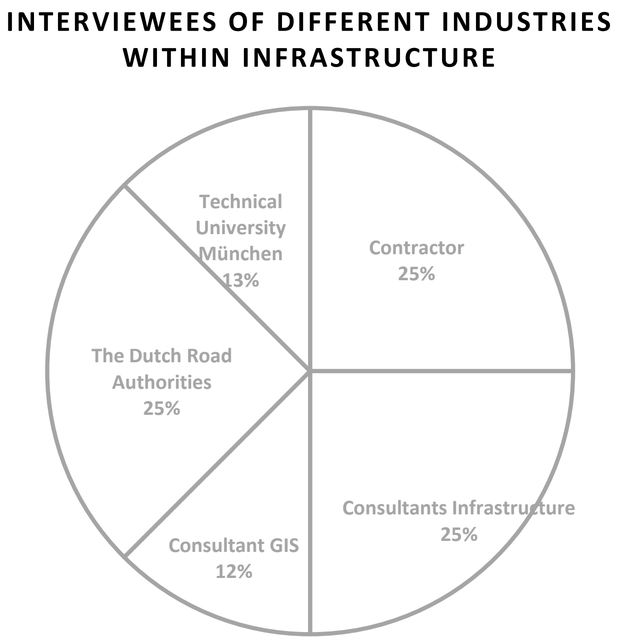


Figure 5: Different industries of the interviewees

3.2.1 BIM IN INFRASTRUCTURE

Interviews with industry experts provided a reliable view on the current BIM advantages, potentials and barriers. Therefore a good understanding of BIM is necessary. The experts see BIM as a digital representation of the real world, where information is added at object level. A BIM is a working process for decision making during its life-cycle. One of the most mentioned advantages is the 3D visualization of a project. The interviewees mentioned this is a great tool to communicate project designs. The following advantages are currently most beneficial:

- Visualization;
- Improved communication between disciplines;
- Interoperability;
- Clash detection.

Most of the interviewees also mentioned time and budget savings, but these are hard to measure. Therefore these are not committed in the current advantages. Before achieving these saving, some barriers have to be overcome. Despite continuously increasing adoption and the growing demand from clients' side, some people have hard time adjusting to new processes and hold on to old traditional project delivery processes. Several issues were mentioned, all related to people and process:

- Lack of experience and knowledge;
- Lack of immediate benefits;
- Fear of changing traditional working methods.

The attitude towards automated model checking was mostly positive. The time savings and therefore cost savings and the correctness of automated checking are the most mentioned potentials. There are some concerns about losing know how of the experts field, the confidence in the correctness of the application and when a check passes based on wrong information, this can lead to false passing of a design. Despite these concerns, the experts are of opinion that automated model checking is a great improvement.

3.2.2 GUIDELINES AND REGULATIONS

The main part of the interviews were projected at model checking, to determine the current method of model checking and the guidelines and contract requirements to be checked. The current way of model checking is based on expert judgement of the engineers. The engineers check each other's work, mostly by a checking list. This is an excel based file, where all contract requirements are included and an engineer simply has to check off a box "Passed". For complex projects an application is used to specify all contract requirement and within this application, document can be added to prove a requirement passes. The geometric requirements are briefly described, and further referred to the guidelines. All interviewees agreed this way of checking is not sufficient, because sometimes rules are missed during this process and errors occur in later phases. Additionally it is very time-consuming and costly process.

For complex projects, there are a lot of guidelines and requirements. The guidelines are included in the contract and therefore they are also of legal status. Road design has several field of guidelines, geometrics, geospatial, constructions, loads, signage, noise, etc. It can take up to 50 documents, which are all applicable for road design. The guidelines for geometrics of road design are:

- Richtlijn Ontwerp Autosnelwegen (ROA) 2014;
- Handboek Wegontwerp part 1 to 4;
- Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom (ASVV) 2012.

Sometimes there are some adjustments on these guidelines, the range of an error can be adjusted, or some elements following each other can be modified, etc. This is normally done to make the design fit into their topological surroundings, especially when engineering new road exits. For these exits, there is little space to work with and then the guidelines have to be adjusted for that specific case. The guidelines are included in the contract with an exclusion of the paragraphs regarded. This will be further specified in the contract. This is the reason why there are no legal codes for infrastructure geometrics.

Finally the interviews were asked what geometric requirement are checked in which phases of the System Engineering process. A system passes a life cycle and is based on 'system thinking'. According to System Engineering, system thinking offers a structure of systems wherein a project can be developed imitable and demonstrably, realized and maintained (Werkgroep Leidraad SE, 2013). Be aware, within SE a system is always a part of bigger picture, called 'system of systems'. Figure 6 presents what requirements are checked in which phases. The phases are:

1. Planning;
2. Concept and Technology Development;
3. Preliminary Design and Technology Completion;
4. Final Design and Fabrication;
5. System Assembly, Integration, Test and Launch;
6. Operation and Sustainment.

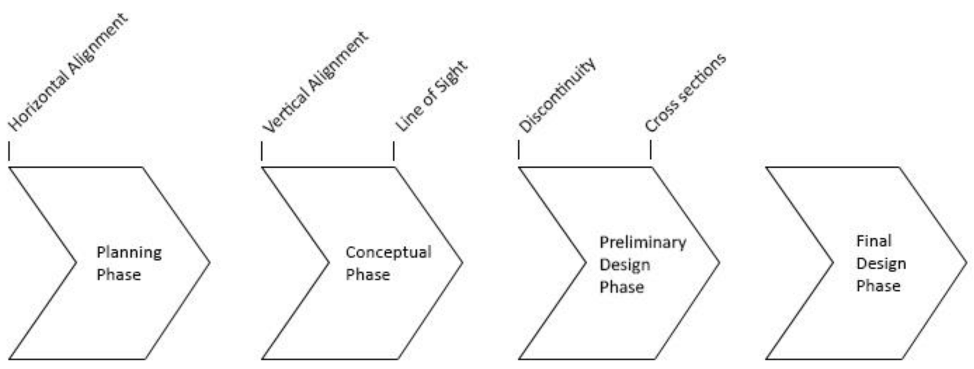


Figure 6: Requirements checked in System Engineering phases

3.3 MODEL CHECKING

The building models are become more complex and detailed, therefore evaluating and validating these designs is also becoming more complex. Conventionally, evaluating designs manually is a time-consuming, expensive and error-prone process. Hjelseth & Nisbet (2010) states as much as 40% of the defects can be related to blunders in the design process. Automated rule checking has been identified as potentially providing significant value to the industry (Solihin & Eastman, 2015). With these automated rule checking, design can be checked by automated interfaces witch are more quickly and reliable (Ding et al., 2006; Han et al., 1998).

A rule-based checking system is defined as a piece of software that does not modify a building design, but rather evaluates it on the basis of configured building objects. Rule-based systems assist users to define and apply rules that identify conditions of importance in the model by executing them on a given model, and return the reports, which basically consist of "pass" or "fail" (Eastman et al., 2009). Before rule checking can be applied, syntax checking is needed. This pre-checking is needed to determine if the needed information is within the data model, such as the properties, names, object, etc.

3.3.1 TYPES OF MODEL CHECKING & PLATFORMS

Research development of rule-based checking system for the building industry started two decades ago (Garrett & Fenves, 1988). The technology is still young and rapidly evolving. In general the rule-based systems are applications which require significant software utilities to provide following functionalities:

Validating model checking:
Within validation based checking there are two kind of checking, compliance checking and geometry based checking. Compliance checking is to check if the model is in accordance with building codes, regulations and so on. Geometry based checking is to check if there are components which clash or give a failure to the rule. Especially when two or more models from different expertise's are collaborated, this is an very useful method to determine failures. The checking is based on topological relationships and Boolean algebra. These rules can also be implemented parametrically, allowing the user to adjust the rule by changing the min / max tolerances the components are checked against (Borrmann & Rank, 2009).

Model content checking or pre-checking:
The purpose is to analyze the professional content of a BIM model for a specific use. It can be focused on the content of information compared to a requirement, comparing client demands, or on correctness of the model. If a model has the correct elements, naming, conventions, properties and other structures needed for full checking.

Guiding model checking:
The purpose of guiding is to guide the designer to consider a large set of solutions for a problem. It is typically used in professional fields, where the designer is not an expert in. The checking is based on two elements: Identify rules for the situations where problems occur, and the presentation of a list of possible actions. The rules are activated on model view definitions, which provides a set of possibilities (Hjelseth & Nisbet, 2010). This can be presented in a decision tree.

Adaptive model checking:
This type of model checking is related to artificial intelligent. It requires predefined rules and an object itself analysis its environment and acts on it. An adaptive object can be an increasing or decreasing floor thickness and its related compressive strength and reinforcements.

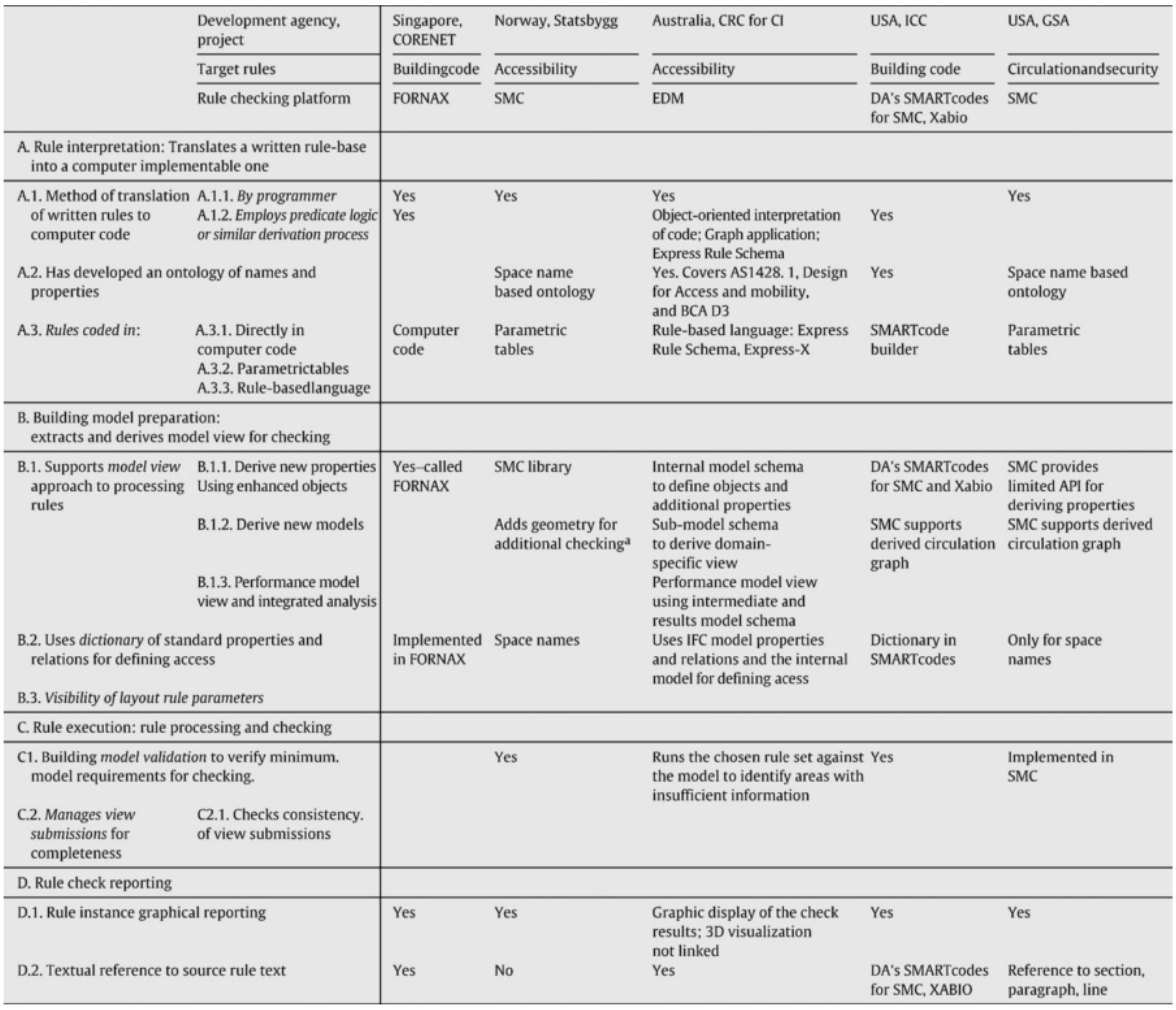
There are several different software platforms that have been developed, which vary in their capability, flexibility of modelling, flexibility of encoding building codes and domain knowledge, reporting and visualization systems and the integration with other applications (Nawari, 2012). These platforms are all specified to the Architecture, Engineering and Construction industry and there are still no platforms specific for the Infrastructure industry. There are four commonly used rule-based checking platforms, all applying rules to IFC models. These will be briefly described and presented in an overview in Table 1:

Solibri Model Checker (SMC):
Solibri Model Checker is a stand-alone, JAVA-based platform that reads a IFC model and maps it to an internal structure facilitating access and processing (Solibri, 2015). It contains a library of capabilities for pre-checking, such as shape overlaps, name and attribute conventions, object existence, fire code exit, path distance checking, space program checking against the actual spaces in a building and others. SMC also offers an automatic viewing of checking issues for reporting in the free Solibri Model Viewer. Rules can be parametrically varied through table-set control parameters. However, entirely new rules are added in java using the SMC application programming interface (API) (Eastman et al., 2009; Nawari, 2012).

Jotne EDModelChecker (EDM):
EDModelChecker provides an object database and supports the open development of rule checking (Jotne, 2015) using the EXPRESS language, which is the language in which the IFC model schema is written. New model views can be developed using EXPRESS and EXPRESS-X, which is a language for mapping instance data from one EXPRESS schema to another and supports extensive queries and reports. These facilities make EDM open to sophisticated user extensions. EDM also provides textual reporting and server services. It is supported by EDMModel Server, an object-based backend database server, that allows EDM to deal with large building models and potentially several of them at a time (Eastman et al., 2009).

FORNAX:
The first large effort in building rules checking, the Singapore CORENET effort developed its own platform, called FORNAX, developed by novaCITYNETS Pte. Ltd on top of EDM Model Checker (Khemiani, 2005). FORNAX is a C++ object library that derives new data and generates extended views of IFC data. FORNAX objects carry rules for assessing themselves, providing good object-based modularity (Eastman et al., 2009; Nawari, 2012).

SMARTcodes:
A new platform for rule checking is being developed by the International Code Counsel in coordination with buildingSmart Alliance (Conover, 2007). The SMARTcodes is a concept of intelligent codes, which provides methods of converting codes and standards from textual rigid format into computer code. This is done by using a powerful semantic-oriented representation of a dictionary of domain-specific terms and semi-formal mapping methods.



	Development agency, project	Singapore, CORENET	Norway, Statsbygg	Australia, CRC for CI	USA, ICC	USA, GSA
	Target rules	Buildingcode	Accessibility	Accessibility	Building code	Circulationandsecurity
	Rule checking platform	FORNAX	SMC	EDM	DA's SMARTcodes for SMC, Xabio	SMC
A. Rule interpretation: Translates a written rule-base into a computer implementable one						
A.1. Method of translation of written rules to computer code	A.1.1. *By programmer*	Yes	Yes	Yes		Yes
	A.1.2. *Employs predicate logic or similar derivation process*	Yes		Object-oriented interpretation of code; Graph application; Express Rule Schema	Yes	
A.2. Has developed an ontology of names and properties			Space name based ontology	Yes. Covers AS1428. 1, Design for Access and mobility, and BCA D3	Yes	Space name based ontology
A.3. *Rules coded in*:	A.3.1. Directly in computer code A.3.2. Parametrictables A.3.3. Rule-basedlanguage	Computer code	Parametric tables	Rule-based language: Express Rule Schema, Express-X	SMARTcode builder	Parametric tables
B. Building model preparation: extracts and derives model view for checking						
B.1. Supports *model view* approach to processing rules	B.1.1. Derive new properties Using enhanced objects	Yes–called FORNAX	SMC library	Internal model schema to define objects and additional properties	DA's SMARTcodes for SMC and Xabio	SMC provides limited API for deriving properties
	B.1.2. Derive new models		Adds geometry for additional checking[a]	Sub-model schema to derive domain-specific view	SMC supports derived circulation graph	SMC supports derived circulation graph
	B.1.3. Performance model view and integrated analysis			Performance model view using intermediate and results model schema		
B.2. Uses *dictionary* of standard properties and relations for defining access		Implemented in FORNAX	Space names	Uses IFC model properties and relations and the internal model for defining acess	Dictionary in SMARTcodes	Only for space names
B.3. *Visibility of layout rule parameters*						
C. Rule execution: rule processing and checking						
C1. Building *model validation* to verify minimum. model requirements for checking.			Yes	Runs the chosen rule set against the model to identify areas with insufficient information	Yes	Implemented in SMC
C.2. *Manages view submissions* for completeness	C2.1. Checks consistency. of view submissions					
D. Rule check reporting						
D.1. Rule instance graphical reporting		Yes	Yes	Graphic display of the check results; 3D visualization not linked	Yes	Yes
D.2. Textual reference to source rule text		Yes	No	Yes	DA's SMARTcodes for SMC, XABIO	Reference to section, paragraph, line

Table 1: Overview of rule-checking platform (Eastman et al., 2009)

3.3.2 PROCESS

Literature stated, based on early efforts and work, that there is a structure necessary for implementing a functionally complete rule checking and reporting system. It can be structured into four stages: (1) rule interpretation and logical structuring of rules for their application; (2) building model preparation, where the necessary information required for checking is prepared; (3) the rule execution phase, which carries out the checking, and (4) the reporting of the checking results. These stages will be further explained and are shown in Figure 7.

The data models and the rule-based checking system must have conventions regarding the properties and the structures of the data. These conventions can be managed by a mixture of three strategies: (1) the designer must provide information in the building model which is needed for the rule-checking, (2) the application provide new data or generate model views that explicitly derive the lacking data, and (3) the application generate model views and applies simulations or analysis to generate analytically derived data.

Rule interpretation and logical structuring of rules

Building codes and regulations are defined by governments and represented in human languages, written text, tables, equations and figures. To translate these codes into computer interpretable rules, the rules are formally interpreted and translated. A common intermediate language for mapping rules from a human language to a computer interpretable language is First Order Predicate Logic (Robbin, 2006). First Order Predicate Logic brakes down a human interpreted rule into a symbolic formal system which contains variables witch can be quantified.

This rule can be evaluated to TRUE, FALSE or UNDEFINED. Due to the quantification of these rules, First Order Predicate Logic can also determine if a rule applies to either all instances, or that it applies to at least to one of the instances. Defining a rule always relies on two aspects. The first aspect is the condition or context where the rule applies, such as the changing curvature of a transition curve from a straight to a circular curve of specific road. The second aspect is properties upon which the rule applies, in this case the specific changing curvature from zero to a finite value. These steps rely on classifications, and defined methods for measuring lengths and curvature. The classifications and properties are extracted and expanded with new data. The interpretation of where a rule applies and how many instances of the rules must be applied are based on these classifications and standards. The evaluation and assessment of these rules, rely heavily on standards. A defined rule can be implemented in a computer interpretable rule in two ways: Directly in computer language encoded rules or in parametric tables and then translated in computer language encoded rules. Computer language code uses parameterization and branching, where parametric tables defines classes of rules of these parameters, branches and other logical constructs. Defining the parameters provides an easy but limited method for defining rules.

Building model preparation
The manual evaluation and checking of 2D designs has the "visual correctness" as primary requirement. With the object orientated building models the requirements are more detailed and stricter, with e.g. types and properties. Designers have to define the building models in such a way that the models provide the information needed in well-defined agreed upon structures. This information must be properly encoded in data models to allow proper translation and testing. To ensure the issue of erroneous data, the data will be automatically derived for the required rule checking wherever possible, either within the design program or the rule checking program (Eastman et al., 2009). Separate model views can be used to both derive the needed data required for a specific type of rule checking and to extract subsets of an overall building model to allow more efficient processing (Han et al., 1998). Most efforts have followed this approach, if only to partition the development effort. Definition of such model views goes hand-in-hand with the preparation of rule checking functions.

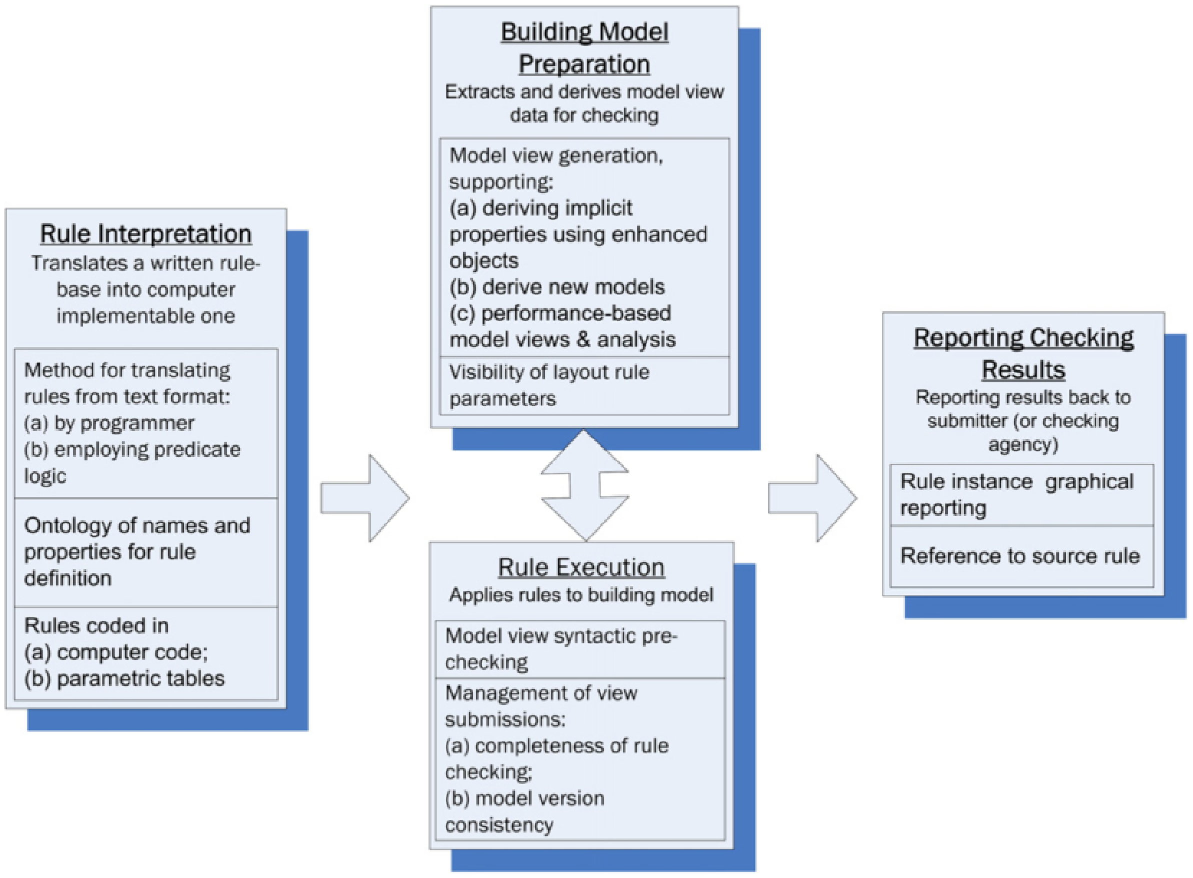
Some rules need implicit information to be checked. Several rule checking systems have developed enhanced building object implemented using object-oriented programming principles. The enhanced object include methods to derive new information and compute complex properties. However, this may not be sufficient for properties that are part of complex spatial configurations made up of multiple nested and bounding objects. Some of these limitations would disappear if fully and accurately defined space objects in buildings could be derived, allowing a rich set of assessments of spaces to be undertaken (Eastman et al., 2009). Another solution can be an automatically derive a new building model with certain attributes to facilitate assessment of the implicit properties or relations or to use performance-based rules. These rules also need a new derived building model, with mostly its own geometry, material or other parameters properties and assumed loads, as input for executing the analysis/simulation.

Rule execution
The rule execution stage consist of combining the computer interpretable rule and the prepared building model. Before the rules can be applied to the model view, the syntax of the model view must be checked. This pre-checking is needed to validate if the model view carriers the right information, such as properties, names, objects, etc. If new model views are generated, the pre-checking is carried out before the derivation.
When the pre-checking is executed and the model complies, the general rule checking can be performed. General rule checking will require a management system to coordinate and oversee the application of the multiple rule modules and their results (Eastman et al., 2009). This management system checks two issues: (1) the completeness of the rule checking and, (2) the model version consistency. The completeness of the rule checking checks if the right set of rules is selected and if the right model view is submitted. This is done by every ruleset and every model view, until the complete rule checking system is completed.

Reporting of the checking results
When the rule execution stage is done, the last part is to report the results. Both the results that PASS and FAIL need to be reported into the results. The rules that pass need to be reported as part of an audit trial that validates the completeness of the check. The rules that fail the rule checking has to be reported to address the problem. An intuitive way of presenting the report is a screenshot of the problem addressed, by using the project coordinate system. This is normally done for spatial conflict testing (Solibri, 2015). In addition to the screenshot it is important to conduct the applicable rule and how this rule has failed. This requires the reverse mapping from

the computer interpretable rule to the original natural language of the rule. Additional reporting may include the description of how the rule fails, with the parameters involved and possible actions to correct the model view.

These four stages are needed for a working rule-based checking system. The stages are shown in Figure 7, with the aspect of each stage.



Rule Interpretation
Translates a written rule-base into computer implementable one
Method for translating rules from text format: (a) by programmer (b) employing predicate logic
Ontology of names and properties for rule definition
Rules coded in (a) computer code; (b) parametric tables
Building Model Preparation
Extracts and derives model view data for checking
Model view generation, supporting: (a) deriving implicit properties using enhanced objects (b) derive new models (c) performance-based model views & analysis
Visibility of layout rule parameters
Rule Execution
Applies rules to building model
Model view syntactic pre-checking
Management of view submissions: (a) completeness of rule checking; (b) model version consistency
Reporting Checking Results
Reporting results back to submitter (or checking agency)
Rule instance graphical reporting
Reference to source rule

Figure 7: The four stages of a rule-based checking system with their aspects (Eastman et al., 2009)

3.3.3 GEOMETRY

Within validation based checking there are two kinds of checking: Compliance checking and geometry based checking. Compliance checking is to check if the model is in accordance with building codes, regulations and so on. Automatic management of building permit applications has long been a beacon for model checking. One reason is that permitting is a critical point that all facilities have to pass (Hjelseth & Nisbet, 2010). Geometry based checking is to check if there are components which clash or give a failure to the rule. Especially when two or more models from different disciplines are collaborated, this is an useful method to determine failures. The checking is based on topological relationships and Boolean algebra. These rules can also be implemented parametrically, allowing the user to adjust the rule by changing the min / max tolerances the components are checked against (Borrmann & Rank, 2009).

For geometry based checking for Infrastructure projects, the highest level of abstraction of these projects is the alignment model. This defines the course of the linear project and is defined by the superposition of two two-dimensional curves, the horizontal and the vertical alignment. Usually, the vertical alignment consists of line segments and parabolic arcs and defines the corresponding z-coordinates as a function of the length s of the horizontal alignment curve up to a certain point. The horizontal alignment usually consists of line segments, arcs and transition curves and describes the course of an alignment in the XY plane, (Amann et al., 2014). In the following section these alignment components will be further explained.

Horizontal alignment
The horizontal alignment is the profile of connected segments, which describes the course of the project. The horizontal alignment consist of line segments and of curve segment, but also consist of transition curves. In the Infrastructure industry the clothoid is the common used transition curve, but there are several others. In other linear industries, such as rail, other transition curves are used. Based on the context of the project, the segments are geo-referenced and convertible into Northing and Easting values.

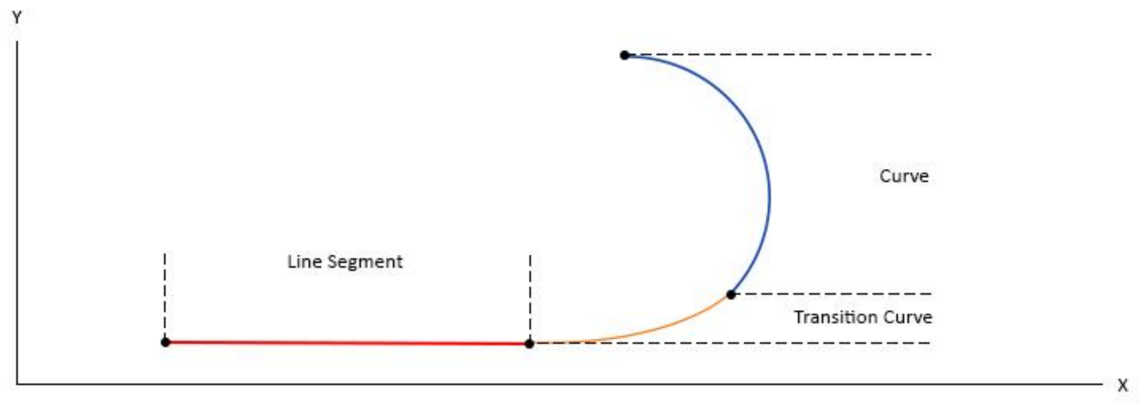


Figure 8: Horizontal alignment segments

Vertical alignment
The vertical alignment is a height profile along the horizontal alignment and gives therefore the height according to the project engineering coordinate system. The vertical alignment consist of segments, which are usually defined as a line segment, a circular arc segment, a parabolic arc segment and sometimes as a unsymmetrical parabolic arc segments. The segments are linked into a wire to create the total vertical alignment.

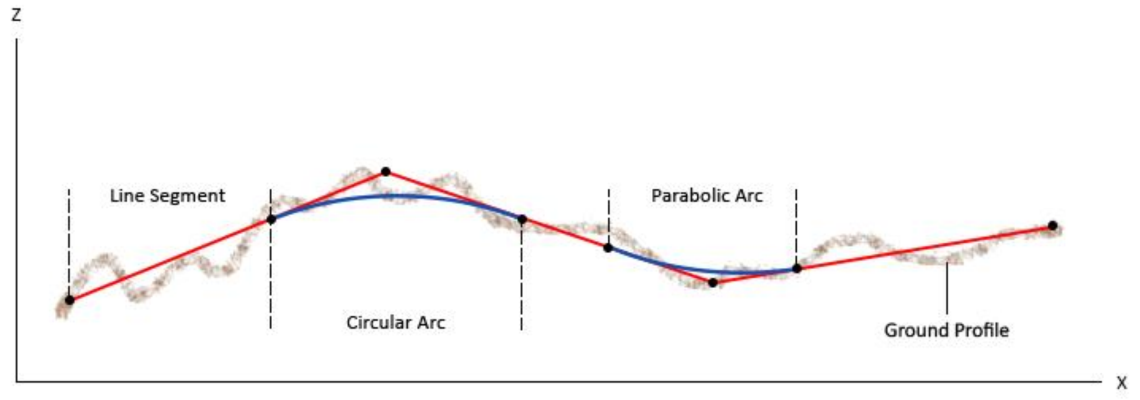


Figure 9: Vertical alignment segments

3.4 *INTEROPERABILITY OF DATA*

Starting in the late 1980s, data models were developed to support product and object model exchanges within different industries. These data models distinguish the schema used to organize the data and the schema language to carry the data (Eastman et al., 2011). From that time till now, the files evolved from modeling of shapes and geometry to modeling of object. While shapes and geometry was the main focus, with BIM this shifted to multiple kinds of geometry, attributes, and properties for different behaviors. The advanced features of building information modeling software have contributed to a shift in the way IT can be used in the construction industry, going beyond simple visual representation of the building to an integrated semantic product and process model (Laakso & Kiviniemi, 2012).

3.4.1 INFORMATION EXCHANGE

The building information modeling software applications should allow for the import of relevant data (for creating and editing a design) and export of data in various formats (to support integration with other application and workflows) (Eastman et al., 2011). This can be done by staying within one software vendor’s products or to use software products which can exchange data between different software applications, using standards. These standards provide a mechanism for interoperability among applications with different internal formats. Interoperability is defined as “The ability of two or more systems or components to exchange information and to use the information that has been exchanged” (The Institute of Electrical and Electronics Engineers, 1990). This can be achieved by mapping parts of each participating application's internal data structure to a universal data model and vice versa. If the universal data model employed is open (i.e. not proprietary), any application can participate in the mapping process and thus become interoperable with any other application that also participated in the mapping (Grilo & Jardim-Goncalves, 2010). In simple cases, the mapping of the internal data structure is syntactical and does not involve changes in meaning. However, many exchanges require embedded expertise that interprets the design information with one meaning to other information with other meanings. These meanings are defined by the field that use the data (Eastman et al., 2011).

Building Information Modeling is a developing process supported by toolsets and data standards for the use of project information. Due to this development, the tools support new processes allowing professionals to integrate intelligent and standardized data, graphics, databases, web services, and decision support methodologies changing the human-computer-interaction and richness of data supported in the process (BuildingSMART, 2007). The development in information modeling supports information exchange across the industry and offers the possibility of knowledge bases for building data aggregation. Standards contains these knowledge bases for building data aggregation. The English dictionary states that a stand is a document recognized agreements, specification or criteria about a product, service or method. De Vries, (2005) states an IT standard: “A standard is an approved specification of a limited set of solutions to actual or potential matching problems, prepared for the benefits of the party or parties involved, balancing their needs, and intended and expected to be used repeatedly or continuously, during a certain period, by a substantial number of the parties for whom they are meant.“

There are two main types of standards, namely proprietary exchange formats (closed) and standard formats (open). A proprietary exchange format is a data format developed by a commercial organization for interfacing with that companies application. The encoding-schema may be published or confidential (Eastman et al., 2011). The proprietary exchange formats can be further organized in two subcategories and the standard formats in three subcategories (Cerri & Fuggetta, 2007):

Proprietary undisclosed standards: are standards whose structure is kept undisclosed. They can be used/exploited by other companies through licensing ruled by specific NDAs (Non-Disclosure Agreements).

Proprietary disclosed standards: are created by a company and then made public. They can be restricted (nobody can use them) or licensable.

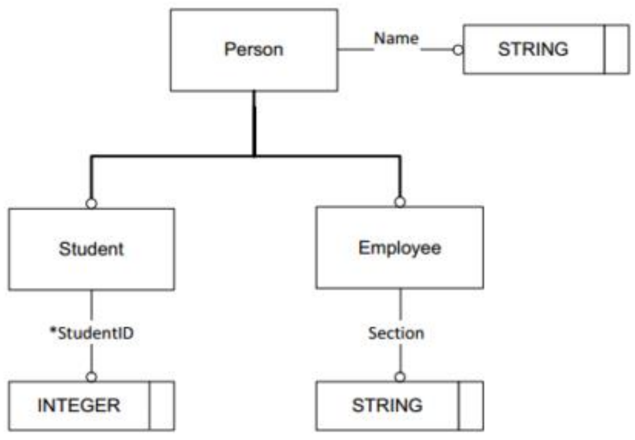
Concerted disclosed standards: ‘’de facto’’ standards are defined by closed or controlled groups of organizations that exploit a consultation mechanisms to collect feedback and suggestions about the evolution of the standard.

Open standards (concerted): they are defined by open consortia or group of companies, universities, and research institutions.

Open standard (de jure): are defined by official national and international standardization bodies such as ANSI and ISO.

3.4.2 DATA MODELING

Nowadays, there are two main data modeling techniques to integrate intelligent into the data standards, generic data modeling and semantic data modeling. Generic data modeling generalizes the conventional data models, which define standardized general relation types. These standardized general relation types ensure the data of an unlimited kind of facts to represent the data which is required. Semantic data modeling represent the object of interest in an application and their relationships in a way that more closely resembles the view the user has of these objects and relationships (Potter et al., 1988). This is used to structure and organize domain knowledge about an object or a phenomenon in such a way that software can automatically process and integrate large amount of information without a predefined interface or human intervention (Karan & Irizarry, 2015). The technique used to capture road information is still the generic data modeling technique, which are written in a data modeling language i.e. a programming language. A language is usually defined as consisting of all the sentences that can be formed according to its grammar. In other words, languages should include a theory of structural description of how the language is composed and what its constituents are (Venugopal et al., 2012). The subset of real information or so called entities are presentations of real world places, persons, things, processes, etc. The entities can be structured a certain way depending on the purpose of the data standard. This data structure can be described in a set of symbols and text or a modeling language (Figure 10).



```
ENTITY Person
  SUPERTYPE OF (Student ANDOR Employee);
  Name : STRING;
END_ENTITY;

ENTITY Person
  Name : STRING;
END_ENTITY;


ENTITY Student
  SUBTYPE OF (Person);
  StudentID : INTEGER;
UNIQUE
  OneIDPerStudent : StudentID;
END_ENTITY;

ENTITY Employee
  SUBTYPE OF (Person);
  Section : STRING;
END_ENTITY;
```

Figure 10: Graphical and Textual representation of entity Person (TU/e, 2014)

The data structure represents a main entity Person, which can be divided into a Student or an Employee. It is presented in a graphical way and a textual way, which have both their advantages and disadvantages (Table 2).

	Advantages	Disadvantages
Graphical	- Group communication - Association easy to read	- Big models - Hard to layout
Textual	- Computer processable - Formal syntax - Complex constraints	- Hard to read

Table 2: Graphical versus Textual structures (Heinen, 2015)

There are hundreds of different modeling languages worldwide, which structure data in all kind of different ways. In the AEC industry and Infrastructure industry also a lot of languages are used. Import languages for these industries are EXPRESS, EXPRESS-G, UML, XML, HTML, etc.

3.4.3 ROAD DATA MODELS

Product data models developed rapidly in the late eighties for construction industry and the standardization of these models started (Rebolj et al., 2008). This standardization focused on the AEC industry and not on Infrastructure, resulting in in-house road data models. In spite of this lacking development, there are several standardized road data models. Details of the evolution of road product models can be found in Rebolj et al. (2008). The Industry Foundation Classes (IFC) (steered by the buildingSMART organization) provides a standardized product model for the AEC industry that is been highly adopted by the industry and is updated with an alignment model. Next to IFC, there are several existing standards for storing and exchanging alignment data which are still being used and updated, such as LandXML, RoadXML and OKSTRA.

LandXML:
LandXML started as one of the earliest efforts, in 2000 with an effort to develop an open source XML-based data model for Infrastructure. Many data modeling efforts are starting to use the World Wide Web Consortium XML schema for data modeling, encoding, and exchange. The XML schema standards have proved to be particularly useful in supporting the development of semantic-rich object-oriented data models (Halfawy et al., 2006).

The LandXML standard is worldwide, the most frequently used open data model for representing and exchanging alignment data (Rebolj et al., 2008). The data schema includes a design data model and a surveying data model. The design data model covers entities such as roadways, alignments (e.g., road centerline, horizontal and vertical alignment curves, and cross sections), a 3-D terrain model, and pipelines networks. The survey data model includes entities such as coordinate geometry point elements, coordinate systems, land parcels, and raw data collection parameters and measurements from surveys (Halfawy et al., 2006). The development of LandXML stopped in 2009 for five years and in July 2015 a new version, LandXML-2.0 proposed. The alignment model consist of five elements, *StaEquation, CrossSects, CoordGeom, Superelevation* and *Profile* (Figure 11). The CoordGeom defines the horizontal alignment, consisting of a sequential chain of elements and the Profile is the vertical alignment. This is built from curves and Point of Vertical Intersections. Detailed description can be found at LandXML.org.

An assessment is conducted on the LandXML-1.2 and there are several problems discovered, where some of them were addressed in the proposed LandXML-2.0. It has key improvements for road design, waterway systems and innovations in interoperability and project life-cycle. The documentation is still minimal. Nevertheless, the interviewees stated the main applications in Infrastructure are still using LandXML-1.2. A major problem is that LandXML is not supported by a standard organization that guarantees its longevity and there are some legal and organizational issues (Amann et al., 2008). The assessment of (Scarponcini, 2013) stated several technical issues: minimally documented, syntax errors and structure errors in the schema, weak point typing, case inconsistencies, name optionality inconsistency, unique identifiers inconsistency, etc.

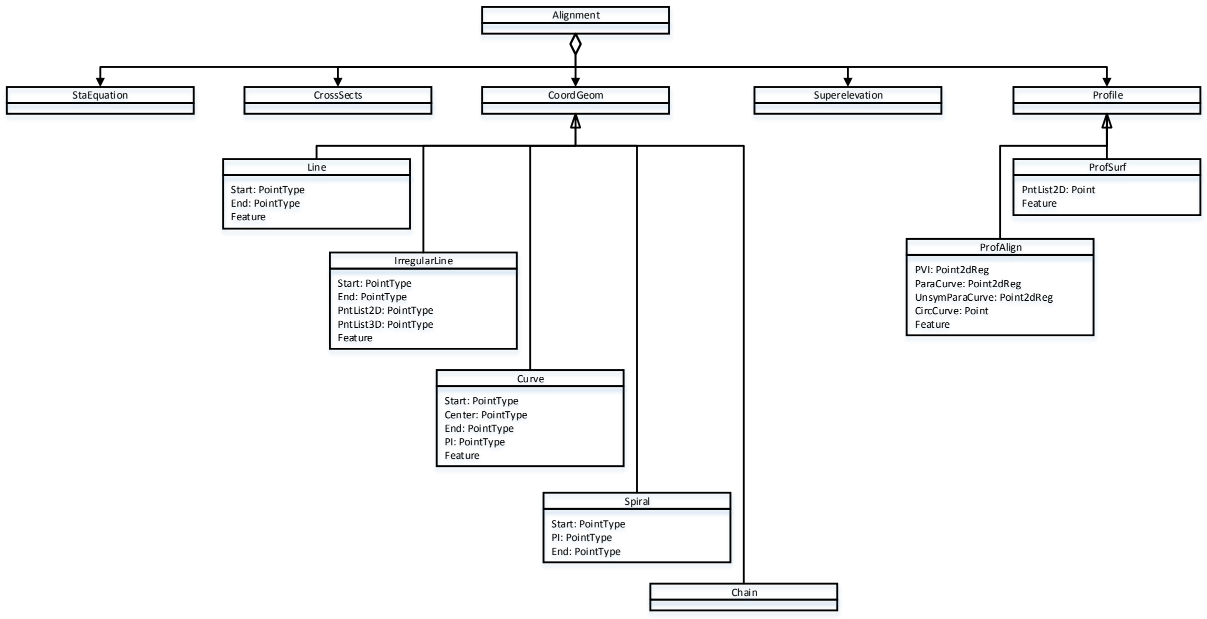


Figure 11: LandXML data model

RoadXML:

The RoadXML is an open data format since 2009 and started from 2006 as a proprietary format developed by different driving simulation actors. RoadXML first concept was to have one format for all the driving simulator modules and to give a unique access to the network description. Even when for some reasons another file format is needed, the access is done through the RoadXML file. The second main concept of the format is to be flexible enough to answer future or proprietary driving simulator needs (Chaplier et al., 2012).
The data standard is a XML based format especially designed for simulation. The elements and attribute are fully readable and understandable. Because it is XML based, data can easily be added at any level. A road network in the RoadXML file format is made of a patchwork of Sub-Network. Each Sub-Network is a collection of Tracks linked by Intersections. Each of these Intersections and Tracks are then enhanced with different layers of data (Ducloux & Millet, 2009):

- The road profile is added on the track to define the pavement surface.
- Road Signs and other local cognitive elements are attached to the track.
- Traffic and 3D description is carried by the road profiles.
- The geographic location according to a spatial coordinate system.

RoadXML, previously named RND is an open data format has as main objective vehicle driving simulations. It is an easy to read and use standard and is designed to be flexible and extendable to enhance the road network. RoadXML offers a multi-layer description of the environment for fast data access for real time applications, topological, logical, physical and visual (Chaplier et al., 2012). RoadXML alignment model is the *Track* model, consisting of the *BankingCurve*, *Portions*, *XYCurve*, *SZCurve* and the *TrackClippedData* (Figure 12). The XYCurve describes the horizontal alignment and the SZCurve describes the vertical alignment. Both alignments consist of a combination of Segments, CircleArcs, Clothoïds and PolyLines (the type attribute set the 3D representation of the PolyLine: sequence of segments or spline interpolation) (Ducloux & Millet, 2009).

Currently RoadXML is maintained by the international RoadXML Board, containing INRETS (French National Institute for Transport and Safety Research), Oktal (French company that develops simulation software and systems for vehicles), PSA Peugeot Citroën (French vehicle manufacturer), Renault (French vehicle manufacturer), Thales (French provider for electrical systems) and an additional member TRL (French Transport Research Laboratory) (Board of RoadXML, 2013).

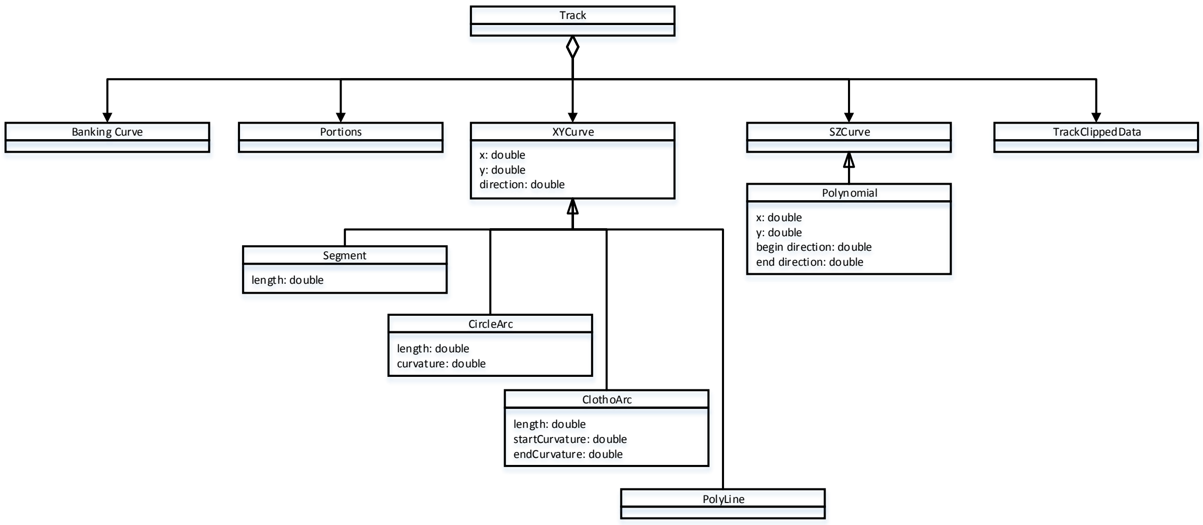


Figure 12: RoadXML data model

OKSTRA:
The OKSTRA Object Catalogue for the Road Sector is a standardized, conceptual data model for various areas of roads and transport. The objective is "ensuring a consistent object representation and a unified data exchange of graphical/geometric data in the road and transport sector" (Erstling & Portele, 1996). The data model is strongly orientated in line with existing regulations and standards, it uses several packages of the ISO standards. OKSTRA is owned and maintained by BASt (German Federal Highway Research Institute) and distributed under a free license.

The alignment model of OKSTRA (Figure 13) references an element of type Axis, which defines a ordered set of elements of type *AxisElement*. This AxisElement is the horizontal alignment and uses elements *Geralde* (line), *Klothoide* (clothoid) and *Kreisbogen, tangential* (arc, tangential). The *Leangsschnitt* (Longitudinal section) describes the vertical alignment using a vertical point of intersection approach.

The major problem of the OKSTRA model is the focus on the German market. The model and the documentation is provided in German, hindering the adoption internationally.

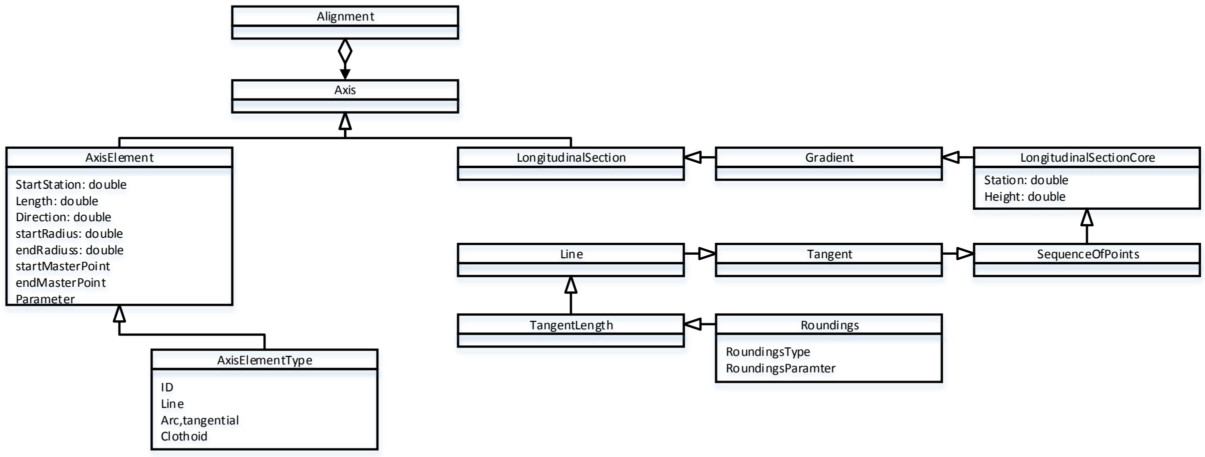


Figure 13: OKSTRA data model

IfcAlignment:
The Industry Foundation Class (IFC) data model is open de-jure data standard, originally for exchange of BIM data in the Architecture, Engineering and Construction industry. Since 1994 the International Alliance for Interoperability (IAI) began the development of an international data standard. It is not owned by a software vendor, but is currently maintained by the buildingSMART organization, former IAI. In October 2005 is was officially accepted as ISO 16739 standard. The data schema's of IFC are defined in four conceptual layers, the domain layer, the interoperability layer, the core layer and the domain layer. The elaboration can be found at buildingSMART IFC4 documentation. The IFC data model is based on entity relationship model, meaning a data model describing information and processes which can be implemented in a database. An entity can be anything, it is a certain class of element e.g. IfcWall, IfcCartersionPoint, IfcAlignment, etc. An entity can exist multiple times in a project, such as doors in a house. A single entity is an instance. An instance has its own identity and values for attributes.

The latest version of the International Foundation Class is IFC4. IFC4 is still not yet suited for Infrastructure projects, although there are new entities introduced, IfcCivilElement and IfcCivilElementType. These entities are only stubs for future work. The IfcAlignment is developed by buildingSMART and is an extension of the IFC4 schema to capture semantic and geometric information (Figure 14). The alignment is the baseline for further projects, such as IfcRoad and IfcBridge. All entities are related through an inheritance hierarchy. This is a hierarchical relationship structure between entity types formed through their inheritance relationships (Is-a). Where a relationship between entity types, the supertype and subtype(s), by which the subtype inherits all the attributes and constraints from the supertype. Additionally subtype may have more specific attributes and constraints (International, 2007).

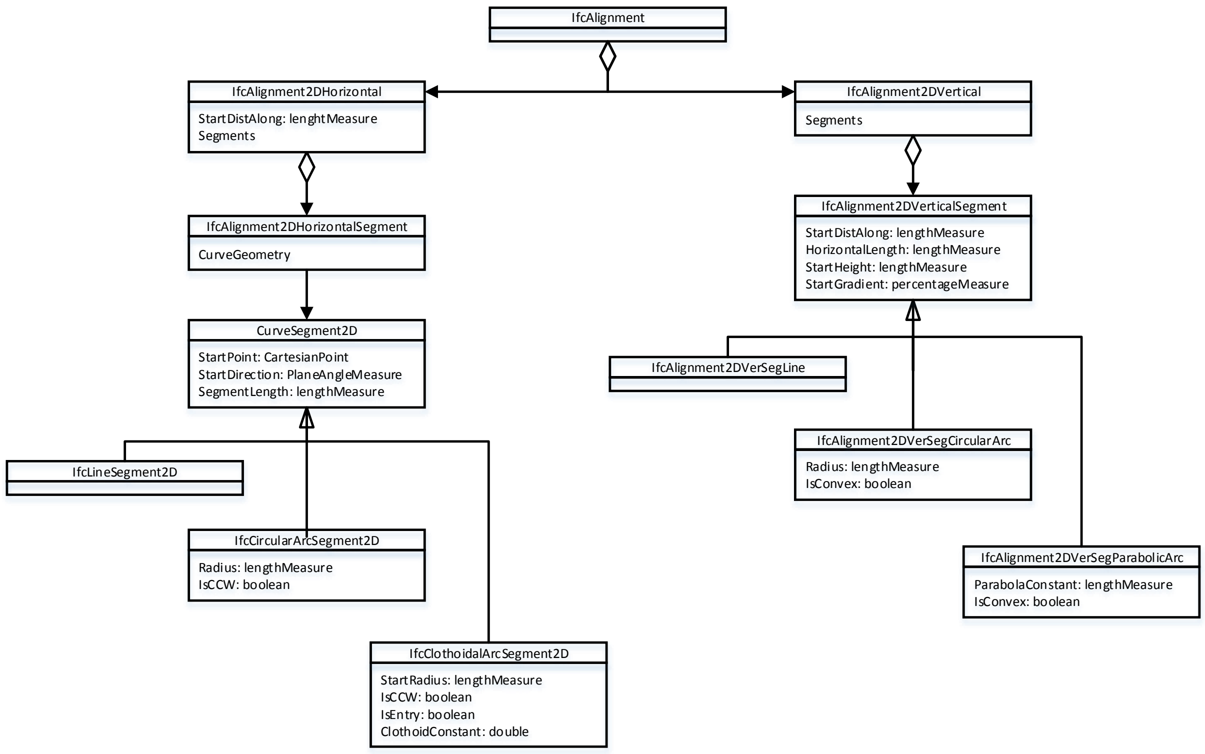

IfcAlignment
IfcAlignment2DHorizontal
StartDistAlong: lenghtMeasure
Segments
IfcAlignment2DVertical
Segments
IfcAlignment2DHorizontalSegment
CurveGeometry
IfcAlignment2DVerticalSegment
StartDistAlong: lengthMeasure
HorizontalLength: lengthMeasure
StartHeight: lengthMeasure
StartGradient: percentageMeasure
CurveSegment2D
StartPoint: CartesianPoint
StartDirection: PlaneAngleMeasure
SegmentLength: lengthMeasure
IfcAlignment2DVerSegLine
IfcLineSegment2D
IfcAlignment2DVerSegCircularArc
Radius: lengthMeasure
IsConvex: boolean
IfcCircularArcSegment2D
Radius: lengthMeasure
IsCCW: boolean
IfcAlignment2DVerSegParabolicArc
ParabolaConstant: lengthMeasure
IsConvex: boolean
IfcClothoidalArcSegment2D
StartRadius: lengthMeasure
IsCCW: boolean
IsEntry: boolean
ClothoidConstant: double

Figure 14: IfcAlignment data model

The alignment model consist of an IfcAlignment2DHorizontal and an IfcAlignment2DVertical. The horizontal alignment consist of line segments (IfcLineSegment2D), circular arcs (IfcCircularArcSegment2D) and a transition curve (IfcClothoidalArcSegment2D). A detailed description of the alignment model is presented by (Amann et al., 2014). All these segments have three common attributes inherited from IfcCurveSegment2D, *StartPoint*, *StartDirection* and *SegmentLength* (Figure 15). In addition IfcCircularArcSegment2D provides attribute for the radius and its orientation, *Radius* and *IsCCW*. The IfcClothoidalArcSegment2D also has the orientation attribute, *IsCCW* and provides the radius at its start point, *StartRadius*. This has a value when the clothoid starts following a circular arc and is 'NIL' when following a line segments, where the radius is interpreted as infinite. The clothoid provides the increasing or decreasing curvature towards the end point with *isEntry* and the clothoid constant parameter A in *ClothoidConstant*, to determine the rate of curvature.

The vertical alignment is defined by vertical line segments (IfcAlignment2DVerSegLine), vertical circular arc segments (IfcAlignment2DVerSegCircularArc) and vertical parabolic arc segments (IfcAlignment2DVerSegParabolicArc). All the three segments have common attributes *StartDistAlong, HorizontalLength, StartHeight and StartGradient*, as illustrated in Figure 15. The StartDistAlong and the HorizontalLength are measured along the horizontal alignment. The vertical circular arc has additionally the *Radius* attribute and the *IsConvex* attribute. The parabolic arc provides the steepness of the arc in the *ParabolaConstant* and also has the IsConvex attribute. The IsConvex describes if the arc is a concave or convex arc (-R or +R).

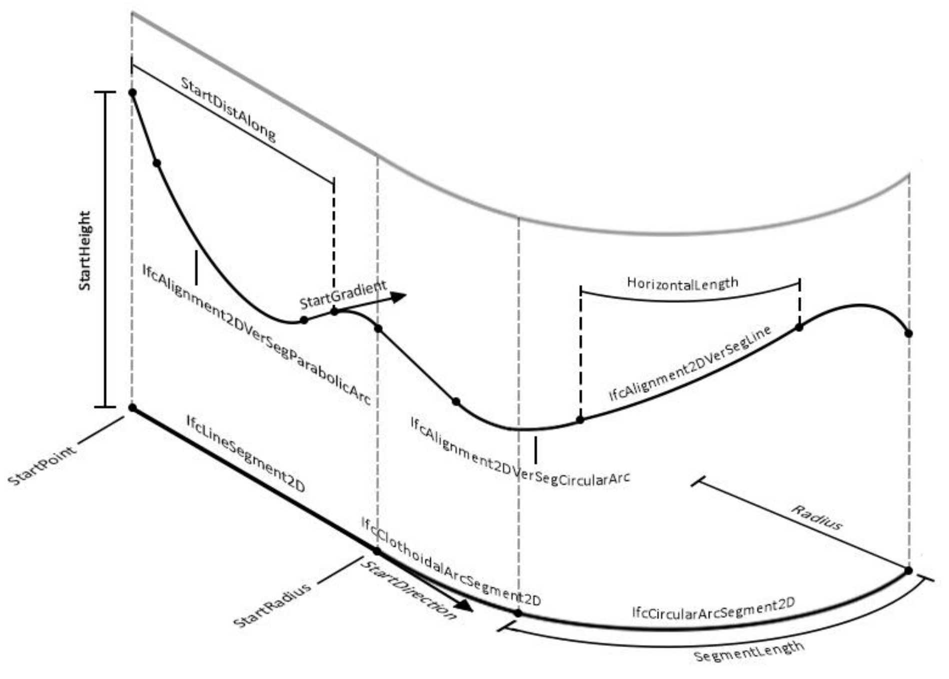


Figure 15: Entities and general attributes IfcAlignment data model

3.4.4 DISCUSSION ON ROAD DATA MODELS

Complexity of data models:
The road data models differ from each other and therefore have their own advantages and disadvantages. So is the RoadXML data model specified in driving simulations and to give unique access to the network description. This is done in an easy to modify and understand XML based data model, which therefore is not made that complex and big. On the other hand LandXML is a very big data model, which consist of several more elements than solely road design and consist of a lot of optional attributes. The OKSTRA model is the biggest of them all and with the German language not the most accessible. The well documented IFC schema is, with its entities for the AEC industry, the second biggest data model and very well comprehensible. Table 3 shown an overview of the different models with their last updates.

	XSD Lines of Code	Last Update
LandXML 2.0	5528	2014
LandXML 1.2	4821	2008
RoadXML 2.4	594	2013
OKSTRA 1.3.0.31	28400	2013
IFC4	13928	2015
IfcAlignment	2348	2015

Table 3: Lines of Code data models

Geometry within data models:
Within the data models the geometry of the segments are defined by specific parameters, such as start point, start direction, segment length, radius, etc. The parameters of straight line segments are clearly defined, in contrast to these of the curves and especially the transition curves. Transition curves allow smooth transitions between segments with different curvature in the horizontal alignment, as shown in Figure 16.

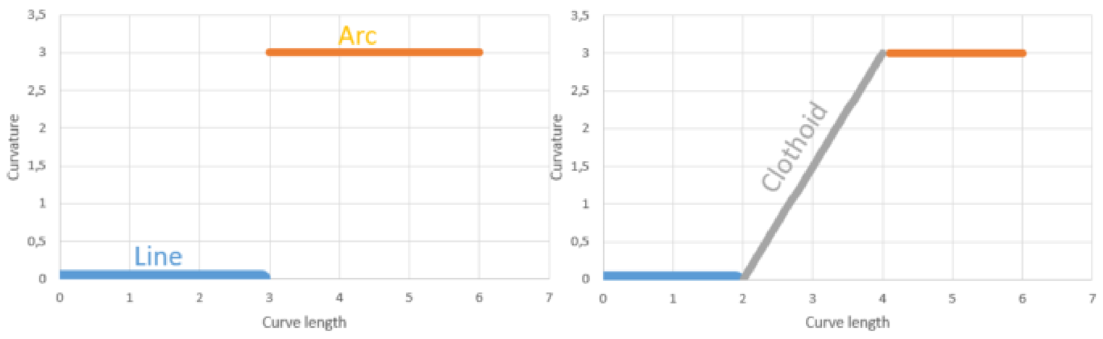


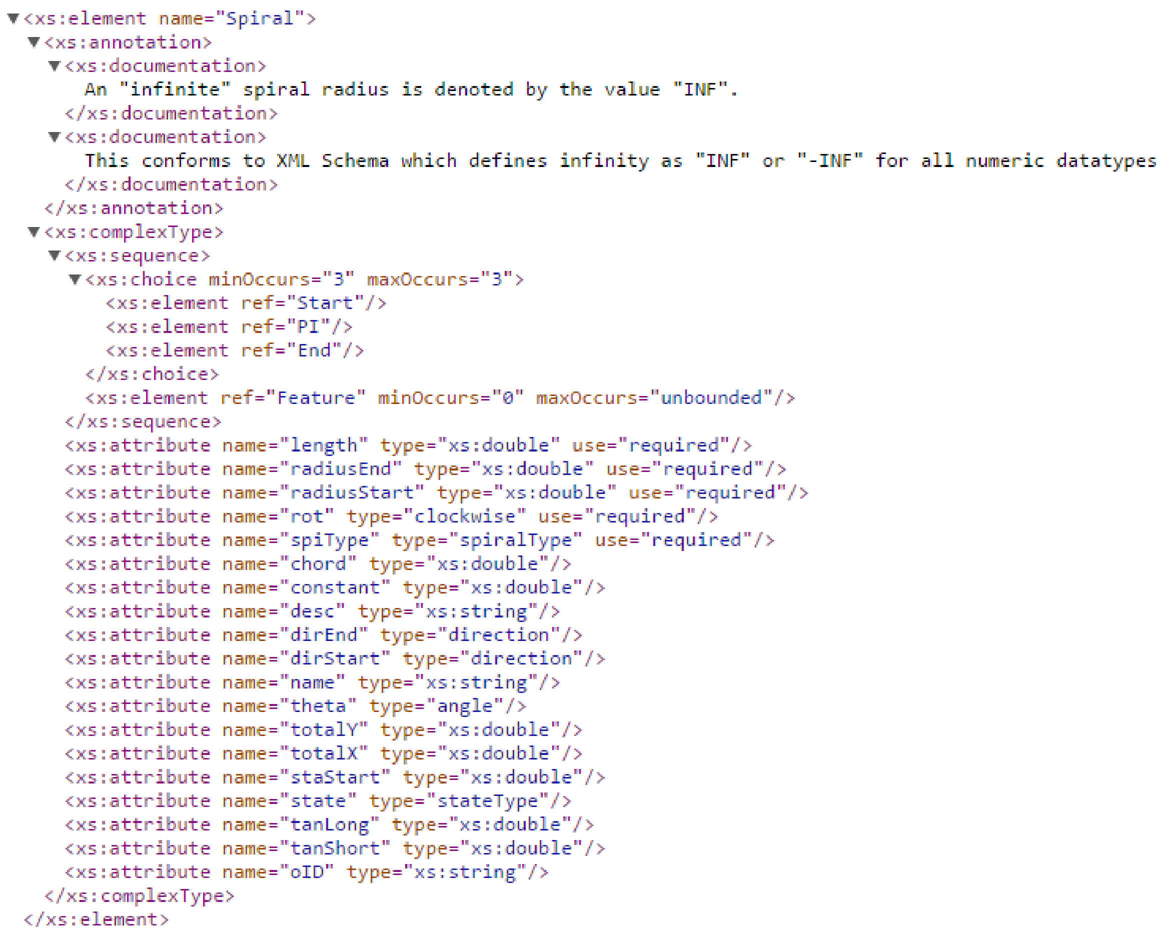
Figure 16: Smooth transitions between lines and arcs (Amann et al., 2014)

Due to specific domains in Infrastructure and country regulations, none of the existing alignment models support all types of transition curves used in Infrastructure. For instance, LandXML is missing a C-Clothid curve type and RoadXML and the IfcAlignment Proposal lack the support of a Bloss transition curve type (Amann et al., 2014). LandXML is the model which supports the most transition curves, as shown in Table 4.

Curve Type	LandXML	RoadXML	OKSTRA	IfcAlignment
Clothoid	Yes	Yes	Yes	Yes
Bloss	Yes	No	No	No
Sinusoid	Yes	No	No	No
Wiener Bogen	Yes	No	No	No
C-Clothoid	No	No	No	No

Table 4: Support of different transition curves across different standards (Amann et al., 2014)

The transition curves of LandXML are bloss, clothoid, cosine, cubic, sinusoid, reverse biquadratic, reverse bloss, reverse cosine, reverse sinusoid, sine half wave, biquadratic parabola, cubic parabola, Japanese cubic, radioed and Wiener Bogen. These transition curves are defined by this 'Spiral' type and have several flaws. These flaws are insufficient documentation, syntax errors in the LandXML 1.2 schema, weak point typing, case inconsistencies, or name inconsistencies (Scarponcini, 2013). For example a clothoid can be described unambiguously by specifying only six parameters (six double values), while LandXML allows to over determine the spiral type, for instance by specifying a clothoid with more than six values. The spiral type in LandXML has 19 different XML attributes (Figure 17). Unfortunately, that also allows to specify impossible transition curves, which violates the principle of data integrity (Amann et al., 2014). In RoadXML, exclusively clothoids are used for transition curves. Clothoids are described by six parameters. This allows a very compact and at the same time unique definition of a clothoid. Other transition curve types are missing in RoadXML (Chaplier et al., 2012). Within the IfcAlignment, also exclusively clothoids are defined for transition curves. All these curves have seven parameters in total. The inherited start point, start direction and segment length and the specific defined parameters for curves. The radius, the orientation (clockwise or counter-clockwise), the increasing or decreasing curvature to the endpoint and the clothoid constant A. This constant determines the rate of curvature change along the clothoid.



```
▼<xs:element name="Spiral">
  ▼<xs:annotation>
    ▼<xs:documentation>
       An "infinite" spiral radius is denoted by the value "INF".
     </xs:documentation>
    ▼<xs:documentation>
       This conforms to XML Schema which defines infinity as "INF" or "-INF" for all numeric datatypes
     </xs:documentation>
   </xs:annotation>
  ▼<xs:complexType>
    ▼<xs:sequence>
      ▼<xs:choice minOccurs="3" maxOccurs="3">
         <xs:element ref="Start"/>
         <xs:element ref="PI"/>
         <xs:element ref="End"/>
       </xs:choice>
       <xs:element ref="Feature" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="length" type="xs:double" use="required"/>
     <xs:attribute name="radiusEnd" type="xs:double" use="required"/>
     <xs:attribute name="radiusStart" type="xs:double" use="required"/>
     <xs:attribute name="rot" type="clockwise" use="required"/>
     <xs:attribute name="spiType" type="spiralType" use="required"/>
     <xs:attribute name="chord" type="xs:double"/>
     <xs:attribute name="constant" type="xs:double"/>
     <xs:attribute name="desc" type="xs:string"/>
     <xs:attribute name="dirEnd" type="direction"/>
     <xs:attribute name="dirStart" type="direction"/>
     <xs:attribute name="name" type="xs:string"/>
     <xs:attribute name="theta" type="angle"/>
     <xs:attribute name="totalY" type="xs:double"/>
     <xs:attribute name="totalX" type="xs:double"/>
     <xs:attribute name="staStart" type="xs:double"/>
     <xs:attribute name="state" type="stateType"/>
     <xs:attribute name="tanLong" type="xs:double"/>
     <xs:attribute name="tanShort" type="xs:double"/>
     <xs:attribute name="oID" type="xs:string"/>
   </xs:complexType>
 </xs:element>
```

Figure 17: Attributes of transition curves 'Spiral' type (LandXML.org, 2015)

Development of arbitrary transition curves:
All the data models define parameters to describe the transition curves. If a software vendor wants to implement a data model, it has to understand and interpret the defined parameters. This can lead to problems resulting from missing, informal, incomplete or flawed documentation (Amann et al., 2014). Additional to these problems it is time-consuming to interpret all parameters into an application. When a rich set of parameters is included, it often not clear which parameters are used for certain specific transition curves. This gives a problem of including a minimal, but sufficient set of parameters. Due to these problems the Technical University of München propose an inversion of control in the design of an alignment model standard. The main idea in the inversion of control approach is not to store parameters and their values solely, but to additionally exchange functions to interpret these values in order to visualize or analyze curves (Amann et al., 2014). This approach can be used for all types of curves. To correctly interpret a curve, a general set of parameters are defined and additionally an inclusion of an algorithm is proposed within an alignment model. This algorithm defines a set of functions necessary to visualize and analyze curves.

- This approach offers several advantages (Amann et al., 2014):
- The user can choose its own parameters for a curve;
- There is no misinterpretation of the parameters of a curve by the user;
- There is no limitation of types of curves;

- There is no more documentation of parameters of curves necessary;
- No time is needed to interpret different curve types defined by data models.

Relative stationing and Absolute stationing:
Within a data model there a two types of stationing, relative and absolute stationing. Relative stationing implies that an alignment element is described referring to a specific element. A line can be defined by its start point, its direction and length and from these attributes, other attributes can be derived such as an endpoint. Absolute stationing capture the information of both the start point and end point and compute a line connected to these points.

When relative stationing is used and the computed endpoint is used for the start point of the next segment, inaccuracies of computed coordinates of an alignment can occur. Using absolute stationing and storing the attributes of the start and endpoint, these inaccuracies shift to the computed length attribute. Considering these two approaches, one has to weigh the importance of the endpoint and length.

4 SCIENTIFIC ARTICLE

For this research I was asked to present this research in a scientific paper to contribute to the academic community. The journal 'Automated in Construction' is a good candidate to present this paper to. It is an international journal for the publication of original research papers.

The journal publishes refereed material on all aspects pertaining to the use of Information Technologies in Design, Engineering, Construction Technologies, Maintenance and Management of Construction Facilities. The scope of Automated in Construction is broad, encompassing all stages of the construction life cycle from initial planning and design, through construction of the facility, its operation and maintenance, to the eventual dismantling and recycling of buildings and engineering structures (Skibniewski, 2016).

AUTOMATED GEOMETRY CHECKING IN THE PLANNING PHASE OF THE SYSTEM ENGINEERING PROCESS BASED ON OPEN SOURCE

L.I.H. Wijnholts, 0786166
Eindhoven University of Technology, Student Construction Management and Engineering, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

ARTICLE INFO

Article history:
V 1.0 - 12 January 2016

Keywords:
BIM
Infrastructure industry
Road data models
Alignment models
IfcAlignment
Model checking

ABSTRACT

This research reports on a prototypical implementation of an Automated Geometry Checker for the Planning phase of the System Engineering process, based on open source. The checker is based on the new Industry Foundation Classes (Ifc)Alignment data model, the baseline for further developments in infrastructural and civil engineering. The research presented has two aims: (1) to develop an easy-to-use prototypical validation geometry checker, (2) to identify issues and capabilities of the IfcAlignment schema based on real-world scenarios. Geometric guidelines for road design are used to identify a classification of these rules applicable on the alignment model and translated in formal code. There are four classes identified by complexity of the rules to be checked. For all four classes an example is presented and implemented in the prototypical Geometry Checker. The alignment model is parsed in a viewer and issues from the rulesets are displayed. Based on these experiences, a detailed conclusion, discussion and starting points for future research is provided.

4.1 INTRODUCTION

Due to the widely acknowledged economic and environmental benefits of Building Information Modeling (BIM) in the Architecture, Engineering and Construction (AEC) industry there is a growing interest and usage of BIM in Infrastructure industry. The National Building Information Modeling Standard (NBIS) vision for BIM is 'an improved planning, design, construction, operation, and maintenance process using a standardized machine-readable information model for each facility, new or old, which contains all appropriate information created or gathered about that facility in a format useable by all throughout its lifecycle'. With this useable information automated compliance checking of designs using model checking software is becoming a more realistic prospect (Choi & Kim, 2008).

4.1.1 BIM IN INFRASTRUCTURE

The advanced features of Building Information Modeling software have contributed to a shift in the way IT can be used in industry, going beyond simple visual representation of the building to an integrated semantic product and process model (Laakso & Kiviniemi, 2012). The forecasted growth of BIM use for Infrastructure is no surprise given the expertise available in the AEC industry, the high level of complexity involved in large infrastructure projects, the increased use of prefabrication in Infrastructure, and the growing need for greater efficiency and effectiveness on all aspects of Infrastructure projects (Jones & Bernstein, 2012). The growing need for greater efficiency is due to the scarcer financing and the increasing demand for Infrastructure. Therefore the industry develops alternatives for financing and development methods, such as Public Private Partnerships (PPPs). In such alternatives, collaboration is of high importance and Building Information Modeling is recognized as a process supporting it.

Many benefits of the usage of Building Information Modeling have been identified in the past regarding visualization, design consistency, interoperability, analysis, clash detection, simulation, planning, cost estimations, monitoring and life cycle data (Azhar et al., 2007b; Czmoch & Pękala, 2014; Strafaci, 2008; Volk et al., 2014). Most of these benefits can be extrapolated to the Infrastructure industry (Jones & Bernstein, 2012; Royal Institution of Chartered Surveyors, 2013). Opposing the potential benefits of BIM are the challenges that need to be overcome if effective multi-disciplinary collaborative team work is to be achieved (Bryde et al., 2013). One of the biggest challenges is the lack of interoperability between BIM systems and third party applications of choice (Royal Institution of Chartered Surveyors, 2013). This is one of the three major technical challenges to adopt BIM in the industry: (1) transactional construction process models to eliminate data interoperability; (2) requirements for computable digital design data and (3) practical strategies for the purposeful exchange and integration of information (Azhar et al., 2007b).

Next to these technical challenges, the barriers can be grouped into three main categories: product, process and people (Ning et al., 2008). A number of challenges have been identified, which can be divided into these groups (Eadie et al., 2013; Heinen, 2015; Muz, 2014; Ning et al., 2008; Royal Institution of Chartered Surveyors, 2013; Volk et al., 2014).

4.1.2 VALIDATING COMPLIANCE CHECKING

Guidelines and contract requirements require the industry to check their designs for compliance. Manual construction compliance checking is time-consuming and error-prone, due to a lot of reasons. Examples of these reasons are unfamiliarity with or even lack of the guideline expertise knowledge, or being overwhelmed of the amount of guideline text, engineers own way of quality check based on experience and complexity of the regulations (Nawari, 2012; Zhong et al., 2012). For Infrastructure, the geometry within the topology is the basis and the end result, which is formed in the Planning phase of a project. This early design determines the eventual success and impact of a project in terms of planning, construction, costs and maintenance aspects. Within the Dutch guidelines for roads, there are three guidelines which apply for the geometric infrastructure of roads, the Richtlijn Ontwerp Autosnelwegen (ROA) 2014, the Handboek Wegontwerp Regionale Stroomwegen (HWO) 2013 and the Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom (ASVV) 2012. Together these state the guidelines only for geometrics of roads and consist together of hundreds of rules, which also can exist of sub rules. The ASVV is projected on smaller road within the urban area and is therefore not included in this research. A brief overview of the geometric rule types is presented in Table 5.

To address the issues mentioned above, this paper reports on an open source Automated Geometry Checker for the Planning phase, based on open standards. A prototypical implementation of several rules with example use-cases are presented. In section 2, a brief overview of related work is given. Section 3 presents the main road models. The categorization of the rules and the essential implementation details of the checker are proposed in section 4. In section 5, use-cases from the main requirements are presented. The unit tests are shown in section 6. The conclusion, limitation and future research are presented in section 7 and 8.

RULE TYPES	REQUIREMENTS OF GOVERNMENT DOCUMENTS
Horizontal alignment	ROA §5.2.1 §5.2.2 §5.2.3 HWO §4.3.1 §4.3.2 §4.3.3 §4.3.4 §4.3.5 §4.3.6
Vertical alignment	ROA §5.3.1 §5.3.2 §5.3.3 HWO §4.4.1 §4.4.2 §4.4.3
Cross sections	ROA §5.4 §5.5 HWO §5.1 §5.2 §5.3 §5.4
Discontinuity	ROA §6.1 §6.2 §6.3 §6.4 §6.5 §6.6 §6.7 §6.8 HWO §6.1 §6.2 §6.3 §6.4 §6.5 §6.6 §6.7 §6.8 §6.9 §6.10 §6.11 §6.12 §6.13 §6.14
Line of Sights	ROA §5.1.1 §5.1.2 §5.1.3 §5.6.5 HWO §7.4.2 §7.4.3 §7.4.4 §7.4.5 §7.4.6 HWO §4.2.1 §4.2.2 §4.2.3 §4.2.4 §4.2.5 §4.2.6 §4.2.7

Table 5: Rule categories of government documents

4.2 *RELATED WORK*

Model checking is a way to share and utilize knowledge and cannot be interpreted, when validation passed, as a good design. It presents a way to validate regulations and can therefore rule out the possibility of a bad design. Model checking is a general term of several types of checking, but always performed on a model and the information it contains. This section presents the types of model checking and a brief review of the rule based applications. A detailed review of these applications can be found in (Eastman et al., 2009).

4.2.1 TYPES OF MODEL CHECKING

Validating model checking:
Within validation based checking there are two kind of checking, compliance checking and geometry based checking. Compliance checking is to check if the model is in accordance with building codes, regulations and so on. Geometry based checking is to check if there are components which clash or give a failure to a rule. Especially when two or more models from different disciplines are collaborated, this is an very useful method to determine failures. The checking is based on topological relationships and boolean algebra. These rules can also be implemented parametrically, allowing the user to adjust the rule by changing the min / max tolerances the components are checked against (Borrmann & Rank, 2009).

Model content checking or pre-checking:
The purpose is to analyze the professional content of a BIM model for a specific use. It can be focused on the content of information compared to a requirement, compared to client demands, or on correctness of the model. Correctness of a model contains the correct elements, naming, conventions, properties and other structures needed for full checking.

Guiding model checking:
The purpose of guiding is to provide the designer a large set of solutions for a problem to consider. It is typically used in professional fields, where the designer is not an expert in. The checking is based on two elements: identify rules for the situations where problems occur, and the presentation of a list of possible actions. The rules are activated on model view definitions, which provides a set of possibilities (Hjelseth & Nisbet, 2010). This can be presented in a decision tree.

Adaptive model checking:
This type of model checking is related to artificial intelligent. It requires predefined rules and an object, which itself analysis its environment and acts on it. An adaptive object can be an increasing or decreasing floor thickness and its related compressive strength and reinforcements.

4.2.2 RULE CHECKING PLATFORMS

Solibri Model Checker (SMC):
A stand-alone, JAVA-based platform that reads an Industry Foundation Classes (IFC) model and maps it to an internal structure facilitating access and processing (Solibri, 2015). It contains a library of capabilities for pre-checking, such as shape overlaps, name and attribute conventions, object existence, fire code exit, path distance checking, space program checking against the actual spaces in a building and others. SMC also offers an automatic viewing of checking issues for reporting in the free Solibri Model Viewer. Rules can be parametrically varied through table-set control parameters. However, entirely new rules are added in JAVA using the SMC Application Programming Interface (API) (Eastman et al., 2009; Nawari, 2012).

Jotne EDModelChecker (EDM):
EDM is an object database which supports the open development of rule checking using the EXPRESS language, which is the language in which the IFC model schema is written (Jotne, 2015). New model views can be developed using EXPRESS and EXPRESS-X, which is a language for mapping instance data from one EXPRESS schema to another and supports extensive queries and reports. These facilities make EDM open to sophisticated user extensions. EDM also provides textual reporting and server services. It is supported by EDMModel Server, an

object-based backend database server, that allows EDM to deal with large building models and potentially several of them at a time (Eastman et al., 2009).

FORNAX:
The Singapore CORENET effort developed its own platform, called FORNAX, developed by novaCITYNETS Pte. Ltd on top of EDM Model Checker (Khemiani, 2005). FORNAX is a C++ object library that derives new data and generates extended views of IFC data. FORNAX objects carry rules for assessing themselves, providing good object-based modularity (Eastman et al., 2009; Nawari, 2012).

SMARTcodes:
A concept of intelligent codes, which provides methods of converting codes and standards from textual rigid format into computer code. This is done by using a powerful semantic-oriented representation of a dictionary of domain-specific terms and semi-formal mapping methods (Nawari, 2012).

4.3 ROAD DATA MODELS

Product data models developed rapidly in the late eighties for construction industry and the standardization of these models started (Rebolj et al., 2008). This standardization focused on the AEC industry and not on Infrastructure, resulting in in-house road data models. In spite of this lacking development, there are several standardized road data models. Details of the evolution of road product models can be found in (Rebolj et al., 2008). The Industry Foundation Classes (IFC) (steered by the buildingSMART organization) provides a standardized product model for the AEC industry that is highly adopted by the industry and is updated with an alignment model. Next to IFC, there are several existing standards for storing and exchanging alignment data which are still being used and updated, such as LandXML, RoadXML and OKSTRA.

4.3.1 QUALITY OF ALIGNMENT MODELS

An infrastructure data model is defined by spatial and non-spatial data. Spatial data consist of geometry and positioning of elements and segments. The highest level of geometric abstraction of linear projects is the alignment model. This is the start of the design from almost every linear project and is defined by the superposition of two two-dimensional curves, the horizontal alignment and the vertical alignment. This is to allow engineers to focus on the relevant aspects, such as curvature and gradient of these two two-dimensional curves (Jubierre & Borrmann, 2013). The non-spatial data include all other aspects such as the physical, functional, and performance attributes of the infrastructure assets, as well as the operational, maintenance, rehabilitation, and cost attributes (Halfawy et al., 2006). These attributes have in most cases a relationship with the spatial data and therefore enhance the efficiency in planning, constructing and operating infrastructure assets.

The basic qualities to which a good alignment model has to comply are defined (Amann et al., 2008). On technical matters, there are four qualities. The first quality is to avoid redundant data, since this can lead to data inconsistency. The second is query complexity, which expresses the difficulties for implementation of a data model while querying the model for specifics data. The third is if an alignment model support relative stationing or absolute stationing. Relative stationing data can be derived from the explicit data, such as an endpoint from a start point, direction and length. Absolute stationing stores all the explicit data. The last technical matter is the domain mapping, assessing how good the model maps to road construction. Additionally, there are two general matters to which a good alignment model has to comply, the tool / technology support and the internationalization support.

4.3.2 LANDXML

The LandXML standard is worldwide, the most frequently used open data model for representing and exchanging alignment data (Rebolj et al., 2008). It captures land development, civil, survey and other infrastructure related data. The development of LandXML stopped in 2009 for five years and in July 2015 a new version is proposed, the LandXML-2.0 schema. The alignment model consist of five elements, *StaEquation, CrossSects, CoordGeom, Superelevation* and *Profile* (Figure 18). The CoordGeom defines the horizontal alignment, consisting of a sequential chain of elements and the Profile is the vertical alignment. This is built from curves and Point of Vertical Intersections (PVI). Detailed description can be found at LandXML.org.

An assessment on the LandXML-1.2 discovered several problems, where some of them were addressed in the proposed LandXML-2.0. Nevertheless, the main applications used in Infrastructure are still using LandXML-1.2. A major problem is that LandXML is not supported by a standard organization that guarantees its longevity and there are some legal and organizational issues (Amann et al., 2008). The assessment of the LandXML-1.2 discovered several technical issues (Scarponcini, 2013): minimally documented, syntax errors and structure errors in the schema, weak point typing, case inconsistencies, name optionality inconsistency, unique identifiers inconsistency, etc.

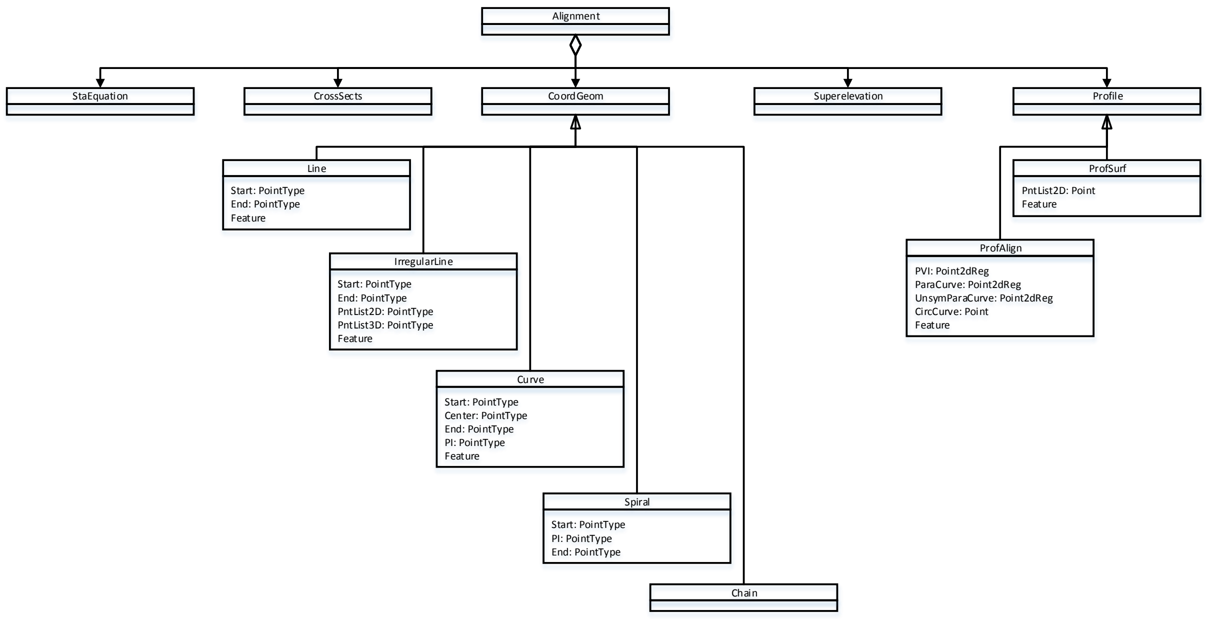


Figure 18: LandXML data model

4.3.3 ROADXML

RoadXML, previously named RND is an open data standard with main objective vehicle driving simulations. It is an easy to read and use standard and is designed to be flexible and extendable to enhance the road network. RoadXML offers a multi-layer description of the environment for fast data access for real time applications, topological, logical, physical and visual (Chaplier et al., 2012). RoadXML alignment model is the *Track* model, consisting of the *BankingCurve*, *Portions*, *XYCurve*, *SZCurve* and the *TrackClippedData* (Figure 19). The XYCurve describes the horizontal alignment and the SZCurve describes the vertical alignment. Both alignments consist of a combination of segments, CircleArcs, Clothoïds and PolyLines (the type attribute set the 3D representation of the PolyLine: sequence of segments or spline interpolation) (Ducloux & Millet, 2009).

Currently RoadXML is maintained by the international RoadXML Board, containing INRETS (French National Institute for Transport and Safety Research), Oktal (French company that develops simulation software and systems for vehicles), PSA Peugeot Citroën (French vehicle manufacturer), Renault (French vehicle manufacturer), Thales (French provider for electrical systems) and an additional member TRL (French Transport Research Laboratory) (Board of RoadXML, 2013).

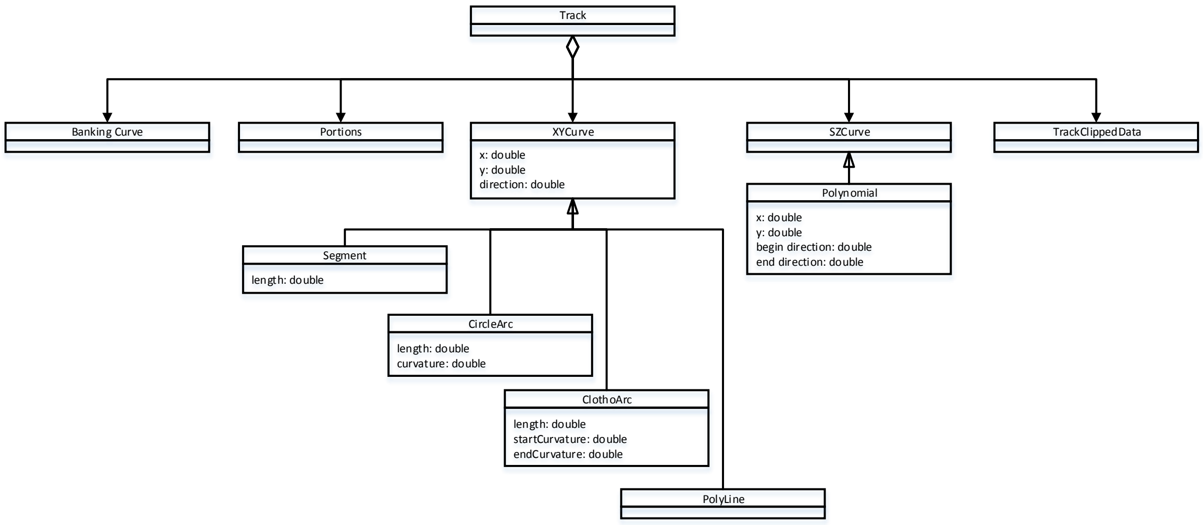


Figure 19: RoadXML data model

4.3.4 OKSTRA MODEL

The OKSTRA Object Catalogue for the Road Sector is a standardized, conceptual data model for various areas of roads and transport. Their objective is "ensuring a consistent object representation and a unified data exchange of graphical / geometric data in the road and transport sector" (Erstling & Portele, 1996). The data model is strongly orientated in line with existing regulations and standards, it uses several packages of the ISO standards. OKSTRA is owned and maintained by BASt (German Federal Highway Research Institute) and distributed under a free license.

The alignment model of OKSTRA (Figure 20) references an element of type Axis, which defines a ordered set of elements of type *AxisElement*. This AxisElement is the horizontal alignment and uses elements *Geralde* (line), *Klothoide* (clothoid) and *Kreisbogen, tangential* (arc, tangential). The *Leangsschnitt* (Longitudinal section) describes the vertical alignment using a Vertical Point of Intersection approach.

The major problem of the OKSTRA model is the focus on the German market. The model and the documentation is provided in German, hindering the adoption internationally.

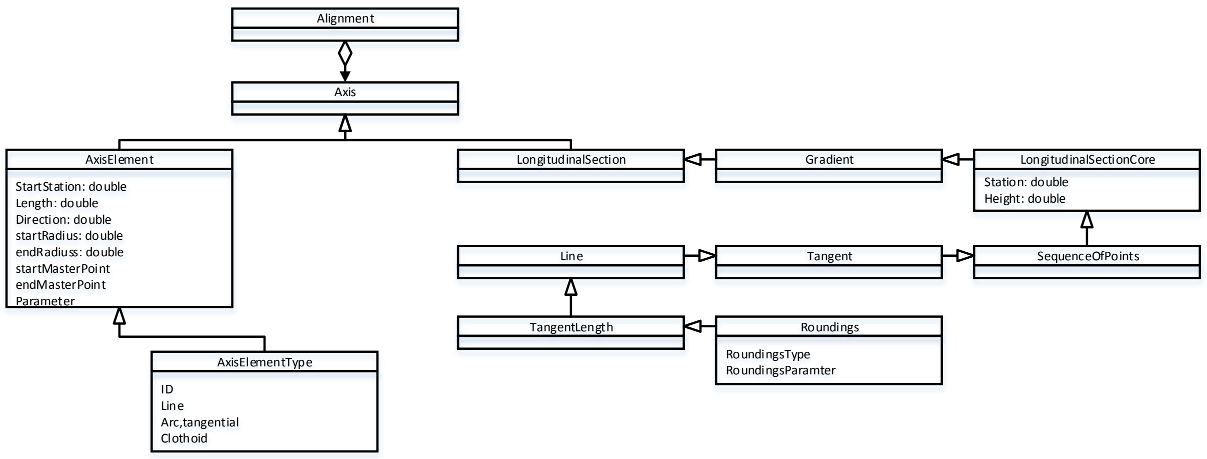


Figure 20: OKSTRA data model

4.3.5 IFCALIGNMENT MODEL

In the AEC industry, the data exchange of product models is mainly achieved using the Industry Foundation Classes (IFC) (Eastman et al., 2011). Since the end of 2014, buildingSMART extended the IFC model with an alignment model (Figure 21).

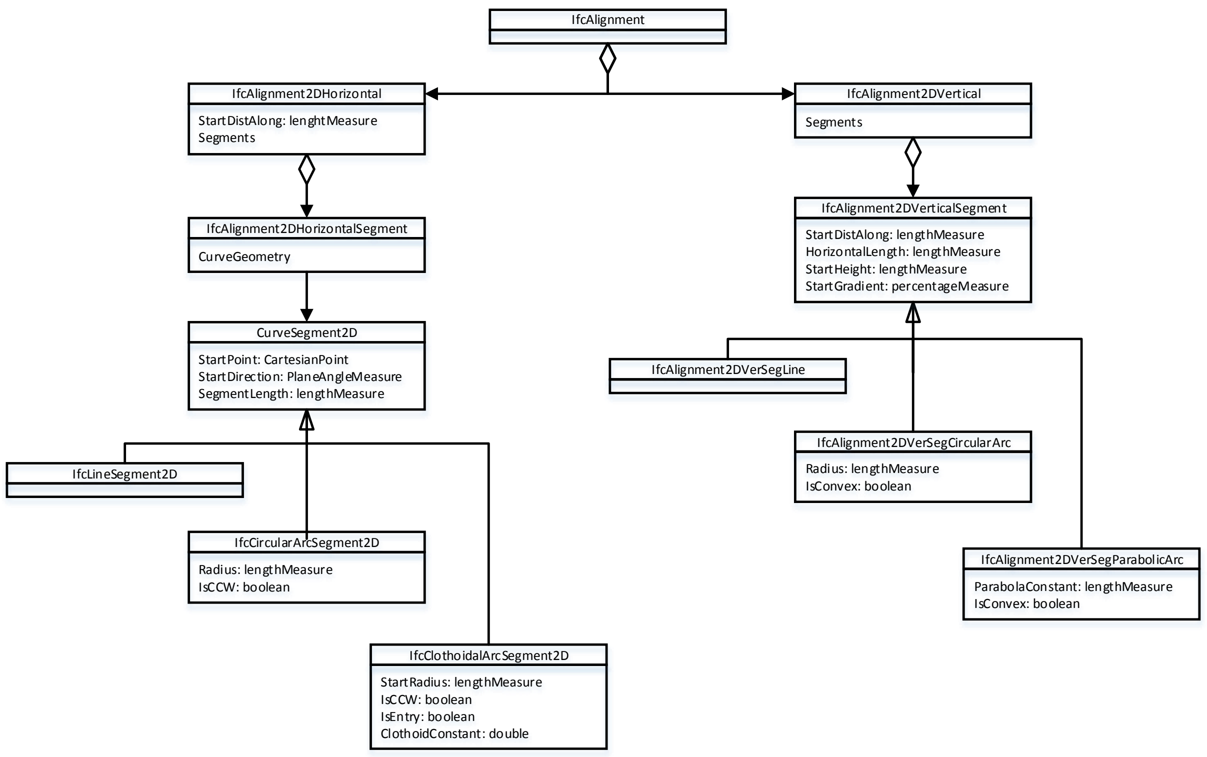

IfcAlignment
IfcAlignment2DHorizontal
StartDistAlong: lenghtMeasure
Segments
IfcAlignment2DVertical
Segments
IfcAlignment2DHorizontalSegment
CurveGeometry
IfcAlignment2DVerticalSegment
StartDistAlong: lengthMeasure
HorizontalLength: lengthMeasure
StartHeight: lengthMeasure
StartGradient: percentageMeasure
CurveSegment2D
StartPoint: CartesianPoint
StartDirection: PlaneAngleMeasure
SegmentLength: lengthMeasure
IfcAlignment2DVerSegLine
IfcLineSegment2D
IfcAlignment2DVerSegCircularArc
Radius: lengthMeasure
IsConvex: boolean
IfcCircularArcSegment2D
Radius: lengthMeasure
IsCCW: boolean
IfcAlignment2DVerSegParabolicArc
ParabolaConstant: lengthMeasure
IsConvex: boolean
IfcClothoidalArcSegment2D
StartRadius: lengthMeasure
IsCCW: boolean
IsEntry: boolean
ClothoidConstant: double

Figure 21: IfcAlignment data model

The alignment model consist of an IfcAlignment2DHorizontal and an IfcAlignment2DVertical. The horizontal alignment consist of lines (IfcLineSegment2D), circular arcs (IfcCircularArcSegment2D) and transition curves (IfcClothoidalArcSegment2D). A detailed description of the alignment model is presented by (Amann et al., 2014). All these segments have three common attributes inherited from IfcCurveSegment2D, *StartPoint*, *StartDirection* and *SegmentLength* (Figure 22). In addition IfcCircularArcSegment2D provides attributes for its radius and its orientation, *Radius* and *IsCCW*. The IfcClothoidalArcSegment2D also has the orientation attribute, *IsCCW* and provides the radius at its start point, *StartRadius*. This has a value when the clothoid starts following circular arc segments and is 'NIL' when following line segments, where the radius at its start point is interpreted as infinite. The clothoid provides the increasing or decreasing curvature towards the end point with *isEntry* and the clothoid constant parameter A in *ClothoidConstant*, to determine the rate of curvature.

The vertical alignment is defined by vertical lines (IfcAlignment2DVerSegLine), vertical circular arcs (IfcAlignment2DVerSegCircularArc) and vertical parabolic arcs (IfcAlignment2DVerSegParabolicArc). All the three segments have common attributes *StartDistAlong, HorizontalLength, StartHeight and StartGradient*, as illustrated in Figure 22*.* The StartDistAlong and the HorizontalLength are measured along the horizontal alignment. The vertical circular arcs have additionally the *Radius* attribute and the *IsConvex* attribute. The parabolic arcs provides the steepness of the arc in the *ParabolaConstant* and also has the IsConvex attribute. The IsConvex describes if the arc is a concave or convex arc (-R or +R).

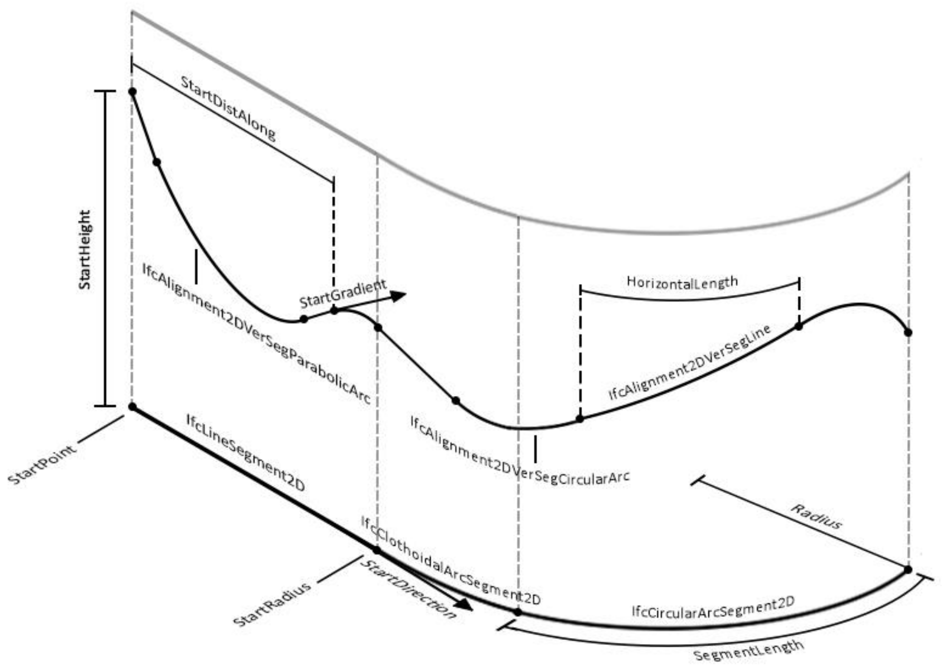


Figure 22: Entities and general attributes IfcAlignment data model

4.3.6 REVIEW ALIGNMENT MODELS

The complexity of the data models is not solely presented in the lines of code, but Table 6 shows the size of the data models. LandXML, OKSTRA and IFC are not limited to alignment data and have also other use-cases.

	XSD Lines of Code	Last Update
LandXML 2.0	5528	2014
LandXML 1.2	4821	2008
RoadXML 2.4	594	2013
OKSTRA 1.3.0.31	28400	2013
IFC4	13928	2015
IfcAlignment	2348	2015

Table 6: Lines of Code data models

Within the data models, the geometry of the segments are defined by specific parameters, such as start point, start direction, segment length, radius, etc. The parameters of straight line segments are clearly defined, in contrast to these of the curves and especially the transition curves. Transition curves allow smooth transitions between segments with different curvature in the horizontal alignment. Due to specific domains in Infrastructure and country regulations, none of the existing alignment models support all types of transition curves used in Infrastructure. LandXML is the model which supports the most transition curves, as shown in Table 7.

Curve Type	LandXML	RoadXML	OKSTRA	IfcAlignment
Clothoid	Yes	Yes	Yes	Yes
Bloss	Yes	No	No	No
Sinusoid	Yes	No	No	No
Wiener Bogen	Yes	No	No	No
C-Clothoid	No	No	No	No

Table 7: Support of different transition curves across different standards (Amann et al., 2014)

The transition curves of LandXML are bloss, clothoid, cosine, cubic, sinusoid, reverse biquadratic, reverse bloss, reverse cosine, reverse sinusoid, sine half wave, biquadratic parabola, cubic parabola, Japanese cubic, radioed and Wiener Bogen. These transition curves are defined by the 'Spiral' type and have several flaws. These flaws are insufficient documentation, syntax errors in the LandXML 1.2 schema, weak point typing, case inconsistencies, or name inconsistencies (Scarponcini, 2013). For example a clothoid can be described unambiguously by specifying only six parameters (six double values), while LandXML allows to over determine the spiral type, for instance by specifying a clothoid with more than six values. The spiral type in LandXML has 19 different XML attributes. Unfortunately, that also allows to specify impossible transition curves, which violates the principle of data integrity (Amann et al., 2014). In RoadXML, exclusively clothoids are used for transition curves. Clothoids are described by six parameters. This allows a very compact and at the same time unique definition of a clothoid. Other transition curve types are missing in RoadXML (Chaplier et al., 2012). Within the IfcAlignment, also exclusively clothoids are defined for transition curves. All these curves have seven parameters in total and is therefore also unique in the definition of the clothoid.

Within a data model there a two types of stationing, relative and absolute stationing. Relative stationing implies that an alignment element is described referring to a specific element. A line can be defined by its start point, its direction and length and from these attributes, other attributes can be derived such as an endpoint. Absolute stationing captures the information of both the start point and end point and compute a line connected to these points. When relative stationing is used and the computed endpoint is used for the start point of the next segment, inaccuracies of computed coordinates of an alignment can occur. Using absolute stationing and storing the attributes of the start and endpoint, these inaccuracies shift to the computed length attribute. Considering these two approaches, one has to weigh the importance of the endpoint and length.

4.4 IFCALIGNMENT CHECKING APPROACH

The IfcAlignment checker contains generally four steps to compute the geometry validation: (1) RuleSet interpretation of rules from the general guidelines and contract requirements, (2) parsing the IfcAlignment file, based on IfcAlignment data schema (Technical Universität München, 2015), (3) the execution of the checks and (4) the reporting in the BIM Collaboration Format (BCF) (Stangeland, 2011) For this research the first three steps are implemented (Figure 23).

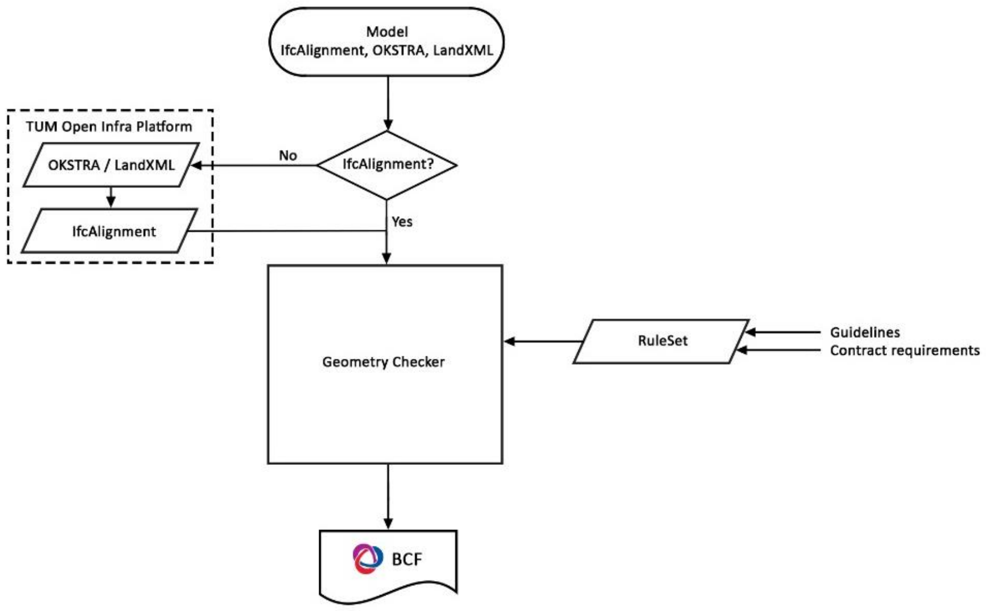


Figure 23: General architecture Automated Geometry Checker

4.4.1 DEVELOPMENT OF RULE-SETS

Rules from guidelines and contracts are represented in human language formats, which have to be formally interpreted and translated into computer processable code. There are several formal interpretation to map this human language rules into computer formal language rules, such as the first order predicate logic, higher order predicate logic, etc. For this research the informal high-level description method pseudocode is used to elaborate on the algorithms of the Automated Geometry Checker.

Considering the computation into formal code, the fundamentals of data checking is reading the attribute values of the model. This explicit data checking is not enough to fulfill all rules, therefore implicit information checks are required. Implicit information is data which is inferred from the explicit data. This data can be computed from the geometry of the alignment model and therefore can fulfill complex rules. This research classifies four types of rules, from the complexity of the rules processing.

Class 1 – Rules checked with one explicit attribute:
This class is based on rule checking of explicit attributes of the geometry of the alignment model. The explicit data is available from the model either directly from the entities or its associated properties with other entities using the explicit relationship entities. Within the IfcAlignment data schema the properties are not on geometric segment level but on Alignment level, which can explicitly checked. Uses of this class of rules are:

- Attributes settings and attributes values for correctness checking.
- Attributes and attributes values that are essential for derived checks.
- Simple geometric compliance checking, such as rules for minimum or maximum segment length, radius, direction, gradient, etc.

Class 2 – Rules checked with multiple explicit attributes:
Within this rule class, the rules consist of multiple derivations of explicit data of a dataset. It derives data with implicit relationships, but does not generate new data. The alignment model does not explicitly captures this relationship, wherefore the program needs to derive the values of these attributes in a generic order. The rule to be checked can have conditions in relationship with other attributes, but also these attributes can have conditions. These sub-rules need to be checked independently, whereupon the main rule can be checked.

Class 3 – Rules checked with computed data from geometry:
More complex rules, which cannot be checked by the derivation of data, extensive computations of the geometry is necessary. Also the derivation and relationships of attributes can play a part for these rules. To compute geometric calculations, software engines on geometrical and topological algorithms are required. Modeling algorithms to analyze the geometrics of the model are used, such as minimal and maximal distances, intersections and others.

Class 4 – Rules checked with external data structures:
Alignment models have a close relation to its surroundings and therefore have a link with Geographic Information Systems (GIS). This class of rules are topological based and checks the alignment model to its geometrical surroundings, such as ground surface, buildings, shielding facilities, signage, etc.

4.4.2 PARSING IFCALIGNMENT FILE

This research uses the Technical University of München’s Open Infra Platform (TUM OIP) to compute a LandXML or OKSTRA alignment model into a IfcAlignment model. This can also be done within the Automated Geometry Checker, to load an alignment model and by use of the CommandLineUtilities a new temporary IfcAlignment file is generated. After the file is loaded, the converted IFC file is parsed and displayed in the 3D view, the 2D horizontal view and the 2D vertical view tabs (Figure 24). The parsing of the IFC file is done with a mix of geometric and topological segments, for inferences and viewing uses. The geometric segments are points, lines, curves, surfaces, etc. For viewing purposes geometric elements are converted in shapes, edges, wires, faces, shells, solids, etc. Geometric line and curve segments are converted in edges and when these edges are adjoined, these can be connected in a wire. This way one alignment can be one shape.

The horizontal alignment is a 2D wire built from all curve and line segments and is displayed in the XY-plane. The vertical alignment is also a 2D wire built from all curve and line segments and is displayed in the XZ-plane. The 3D view combines the horizontal and vertical alignments and is a continued wire, which means calculations can be done to this geometry.

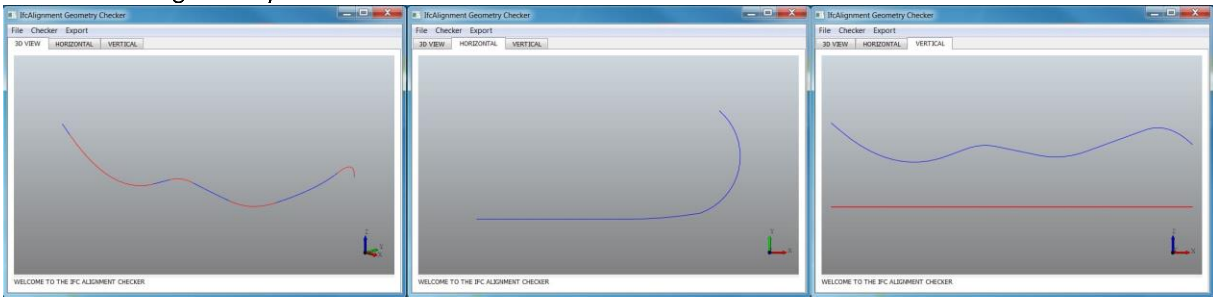

IfcAlignment Geometry Checker
File Checker Export
3D VIEW HORIZONTAL VERTICAL
WELCOME TO THE IFC ALIGNMENT CHECKER

Figure 24: Parsed IfcAlignment file in 3D view, 2D horizontal view and 2D vertical view

Due to the small differences of the relative stationing approach of the IfcAlignment data model, horizontal segments are created with the start point of the next segment. The last segment of the wire is created using the direction and length of the segment. In (1) an example of the parsing of a line segment is shown. Therefore the horizontal wire is an continues curve.

```
PROCESS LINESEGMENT                                                          (1)

INPUT horizontal_line_segment
INPUT the next_segment of the horizontal_line_segment
OUTPUT an edge of the horizontal_line_segment


IF next_segment is not None THEN
        line_segment = MakeLine( horizontal_line_segment.StartPoint, next_segment.StartPoint )
        edge = MakeEdge( line_segment )

ELSE THEN
        line = GeomLine( horizontal_line_segment.StartPoint, horizontal_line_segment.StartDirection )
        edge = MakeEdge( line, horizontal_line_segment.SegmentLength )

END IF
```

The vertical segments are built from their direction and length, which are computed from the HorizontalLength and the StartGradient (2). If a vertical segment is a circular arc or a parabolic arc, it has a convex and a radius or parabola constant, which is needed to compute the curve. The vertical segments are projected on the surface in the Z-direction of the horizontal wire. The attributes StartDistAlong and StartHeight are the parametrized values of the surface.

```
ENTITY IfcAlignment2DVerticalSegment                                          (2)
 ABSTRACT SUPERTYPE OF (ONEOF
        (IfcAlignment2DVerSegCircularArc
        ,IfcAlignment2DVerSegLine
        ,IfcAlignment2DVerSegParabolicArc))
 SUBTYPE OF (IfcAlignment2DSegment);
        StartDistAlong : IfcLengthMeasure;
        HorizontalLength : IfcPositiveLengthMeasure;
        StartHeight : IfcLengthMeasure;
        StartGradient : IfcRatioMeasure;
 INVERSE
        ToVertical : SET [1:1] OF IfcAlignment2DVertical FOR Segments;
END_ENTITY;
```

The general pseudocode for vertical line segments and curve segments is presented (3). For the parabolic arc segment and the circular arc segment a generalization is made.

```
PROCESS VERTICALSEGMENTS                                                          (3)

INPUT the horizontal_wire of Alignment[n]
INPUT the vertical_segments of Alignment[n]
OUTPUT the edges of the vertical_segments


surface = SurfaceOfLinearExtrusion( horizontal_wire, Direction( 0,0,1 ) )

IF vertical_segment is a ("IfcAlignment2DVerSegLine")
    line = GeomLine( horizontal_segment.StartPoint, horizontal_segment.StartDirection )

END IF

line_on_surface = BuildCurve3D( surface, line, start_parameter, end_parameter )
edge = MakeEdge( line_on_surface )



IF vertical_segment is a ("IfcAlignment2DVerSegParabolicArc" or "IfcAlignment2DVerSegCircularArc")

    IF vertical_segment.IsConvex == True THEN
        curve = MakeCurve( vertical_segment.StartPoint, vertical_segment.StartGradient,
            vertical_segment.Constant, True )

    IF vertical_segment.IsConvex == False THEN
        curve = MakeCurve( vertical_segment.StartPoint, vertical_segment.StartGradient,
            vertical_segment.Constant, False )

    END IF
END IF

line_on_surface = BuildCurve3D( surface, curve, start_parameter, end_parameter )
edge = MakeEdge( line_on_surface )
```

The parsing of the 3D wire of one Alignment is done by connecting all vertical edges which are projected on the surface of the horizontal alignment to a wire (4). Due to the small differences of the relative stationing a tolerance is needed to create a continued line.

```
PARSING THE 3D ALIGNMENT WIRE                                                     (4)

INPUT an IfcAlignment file
INPUT vertical_segments
OUTPUT 3Dwires

FOR each IfcAlignment in IfcAlignments
    edges = array of edges

    FOR each segment in vertical_segments
        IF segment is a 'IfcAlignment2DVerSegLine' THEN
            edge = process_VerSegLine
        IF segment is a     'IfcAlignment2DVerParabolicArc'
            edge = process_VerParabolicArc
        IF segment is a 'IfcAlignment2DVerCircularArc'
            edge = process_VerCircularArc

        ADD edge to array of edges
    END FOR

    3Dwires = ConnectAllEdgesToWire( edges, Tolerance )
END FOR
```

The alignment model has a close relationship to its surroundings. By using IfcOpenShell (Krijnen, 2013), the surroundings can be modeled as geometry and displayed in the viewers. The surroundings can be implemented from Geographical Information Systems and loaded in the Automated Geometry Checker.

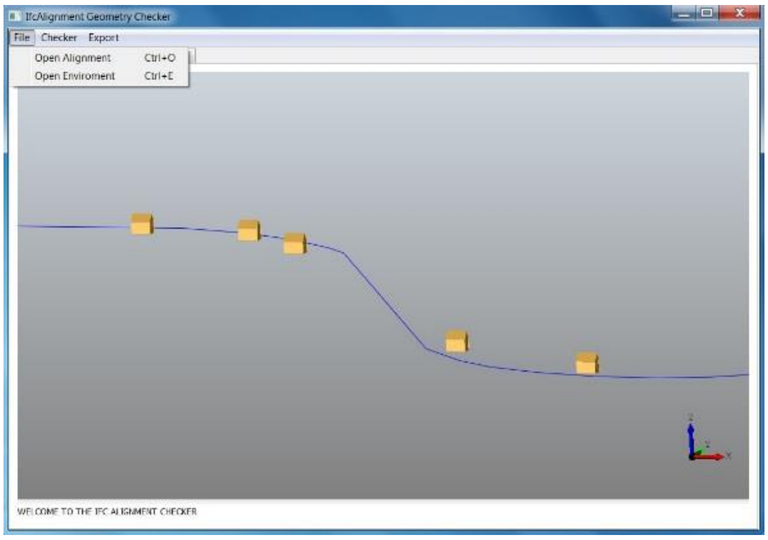

IfcAlignment Geometry Checker
File
Checker
Export
Open Alignment
Ctrl+O
Open Enviroment
Ctrl+E
WELCOME TO THE IFC ALIGNMENT CHECKER

Figure 25: IfcAlignment model with surroundings

4.4.3 CHECK EXECUTION

When the IfcAlignment file is parsed and displayed in the viewer, the checking of the ruleset can be processed. This is done by the Checker and the RuleSet as illustrated in Figure 23. The Checker defines what and how to display and the RuleSet checks the data for errors. When the RuleSet identifies the failed segments, the Checker has to color this segment red. To determine the position of the failed segment in the Alignment, the Checker defines an array of all sequential segment's id's (the #number) of the Alignment and when the failed segment is determined, the id of this failed segment is matched to the position in the array. By using this lookup table method, the right segment in the alignment is determined. To specify the segment of the right Alignment, an offset of all segments is used. The general code of the Checker is proposed in (5). For the four classes of RuleSets, there are use-cases presented in section 5.

```
CHECK EXECUTION                                                              (5)

INPUT an Alignment file
OUTPUT an array of errors

errors = array of errors

FOR each IfcAlignment in IfcAlignments
    error = IfcAlignment_RuleSet.Rule( INPUT )
    ADD error to array of errors

END FOR

IF errors is not None THEN
    FOR each error in errors
        segment = errors[i]
        index = [ segment.id() FOR segment in IfcAlignment.Segments ].index( segment.id() )

        DISPLAY in Viewer segment with color[index] = 'RED'

    END FOR
END IF
```

4.4.4 REPORT GENERATION

In this implementation, the BIM Collaboration Format (BCF) is proposed for the IfcAlignment Checker to report identified issues. BCF is an open standard, initially proposed by DDS, Solibri and Tekla in 2010, to enable workflow communication capability connected to IFC models (Stangeland, 2011). This way of communication is easily implemented by visualization applications and the basic content is creating an issue, add comments and refer to an object. Referring to an object is normally done by using the Global Unique ID's (GUIDs), but for the alignment model this has to be done at geometry level. Only an IfcAlignment entity has an GUID, and the element in IfcAlignment2DHorizontal and IfcAlignment2DVertical has the inherited GUID.

When the IfcAlignment file is imported, every issues is captured in a BCF report, which contains a markup file and a viewpoint file. However the IfcAlignment is not implemented in the BCF standard, so this has to be developed.

4.5 USE-CASES

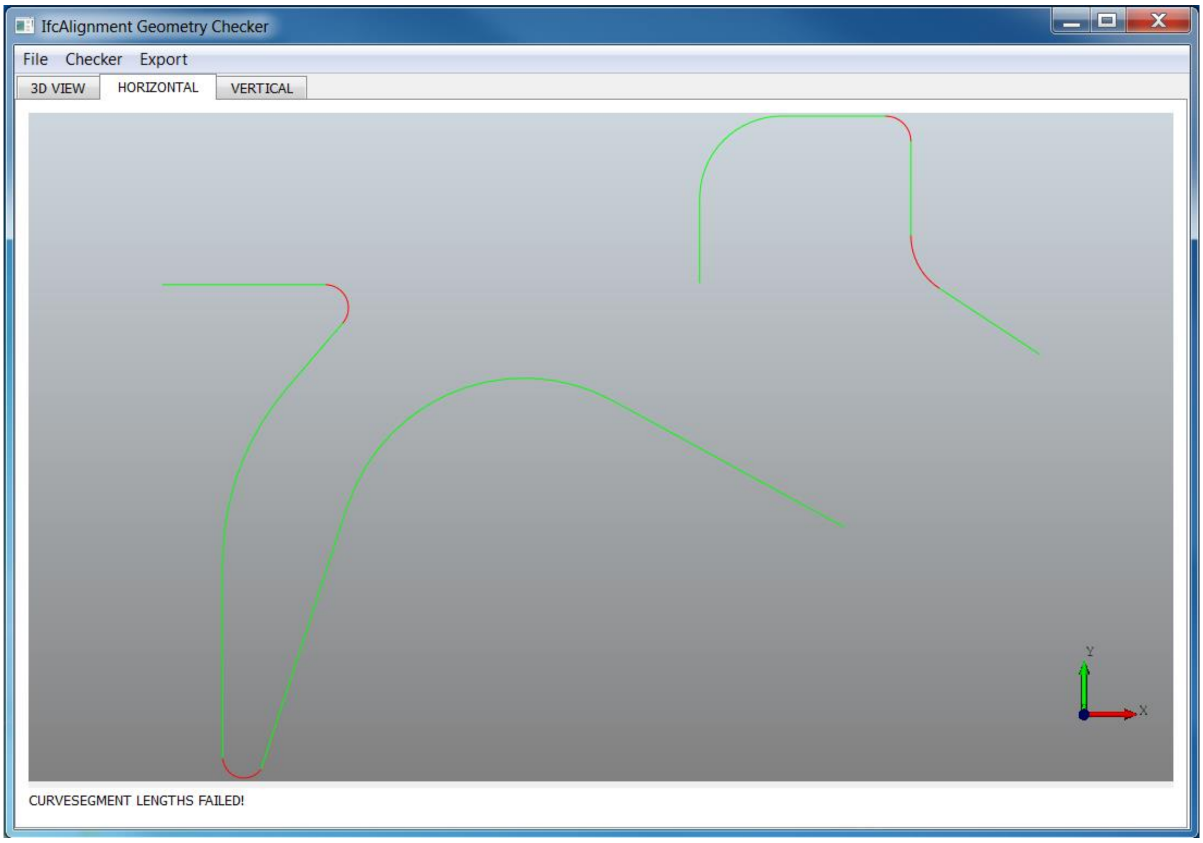
Real-world use-cases from the Richtlijn Ontwerp Autosnelwegen (ROA) 2014 are used to define the RuleSets to test the Automated Alignment Checker. The ROA specifies different categories for guidelines, as mentioned in Table 5. For this research the categories horizontal, vertical and the line of sight are within the scope. The rules for cross sections are focused on the road body and therefore cannot be tested solely to the alignment model. A discontinuity in road design is the convergence and divergence of traffic lanes, which are also not included in the alignment model.

4.5.1 CLASS 1 – RULES CHECKED WITH ONE VARIABLE OF EXPLICIT DATA

This class is based on rule checking of explicit attributes of the geometry of the alignment model. Therefore in the horizontal alignment the minimal length of a curve is given for each design speed (Table 8). Two alignments are modelled with multiple curves, including three curves with a length smaller than the given length at 120 km/h. In Figure 26, this model is presented with the green curves passed and the red curves failed.

Design Speed	Minimal curve length
120 km/h	100 m
90 km/h	75 m
70 km/h	60 m
50 km/h	40 m

Table 8: Minimal curve length (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015)



CURVESEGMENT LENGTHS FAILED!

Figure 26: Checked curves (green = passed, red = failed)

This rule is represented in pseudocode as (6).

```
CHECKING CURVE LENGTH                                                    (6)

INPUT horizontal_wire of Alignment[n]
OUTPUT an array of errors

RuleSet = {
        '120': { 'minimum_length': 100 },
        '90' : { 'minimum_length': 75 },
        '70' : { 'minimum_length': 60 },
        '50' : { 'minimum_length': 40 }
        }

FOR each segment in horizontal_wire
        IF segment is a ("IfcCircularArcSegment2D") THEN
                IF segmentLength < RuleSet[ RoadType ][ 'minimum_length' ] THEN
                        ADD segment to array of errors

                END IF
        END IF
END FOR
```

4.5.2 CLASS 2 – RULES CHECKED WITH MULTIPLE VARIABLE OF EXPLICIT DATA

An example of a main rule with relationship conditions is the minimum and maximum length of horizontal line segments. These segments have different conditions when the adjoined curve segments have the same or opposite direction, shown in Figure 27. The minimum length at 120 km/h of a horizontal line segment with adjoined curve segments with the same direction is 480 meter and curve segments with the opposite direction is 240 meter.

Rules were the relationships also have conditions are for example the Steps Theory (Figure 28) of the Dutch Road Authorities (Rijkswaterstaat). This rule checks if the speed deceleration of connections and nodes of highway are correctly and safely done. By stepwise decreasing the design speed of an exit, the motorist can adjust his speed in a natural way to the following section of the road and therefore the safety is ensured. The following segments have a decreasing design speed (condition 1) and the associated minimum curve length (condition 2), radius (condition 3) and superelevation (condition 4).

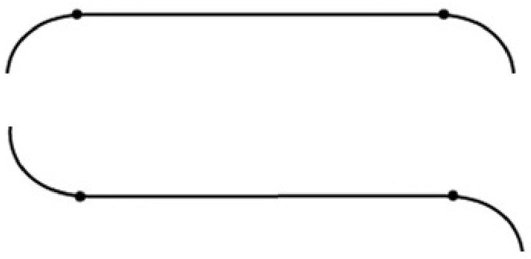


Figure 27: Line segment with adjoined curve segments with same and opposite directions

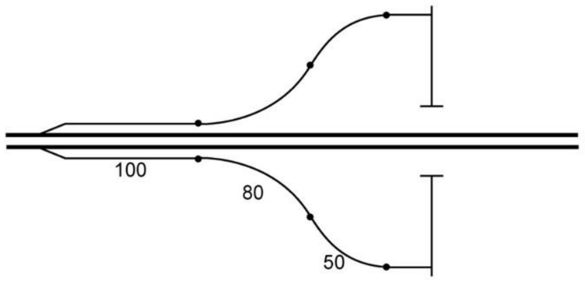


Figure 28: Stepwise decreasing design speed with associated curve lengths

The pseudocode of the use-case, line segment length, is presented as (7).

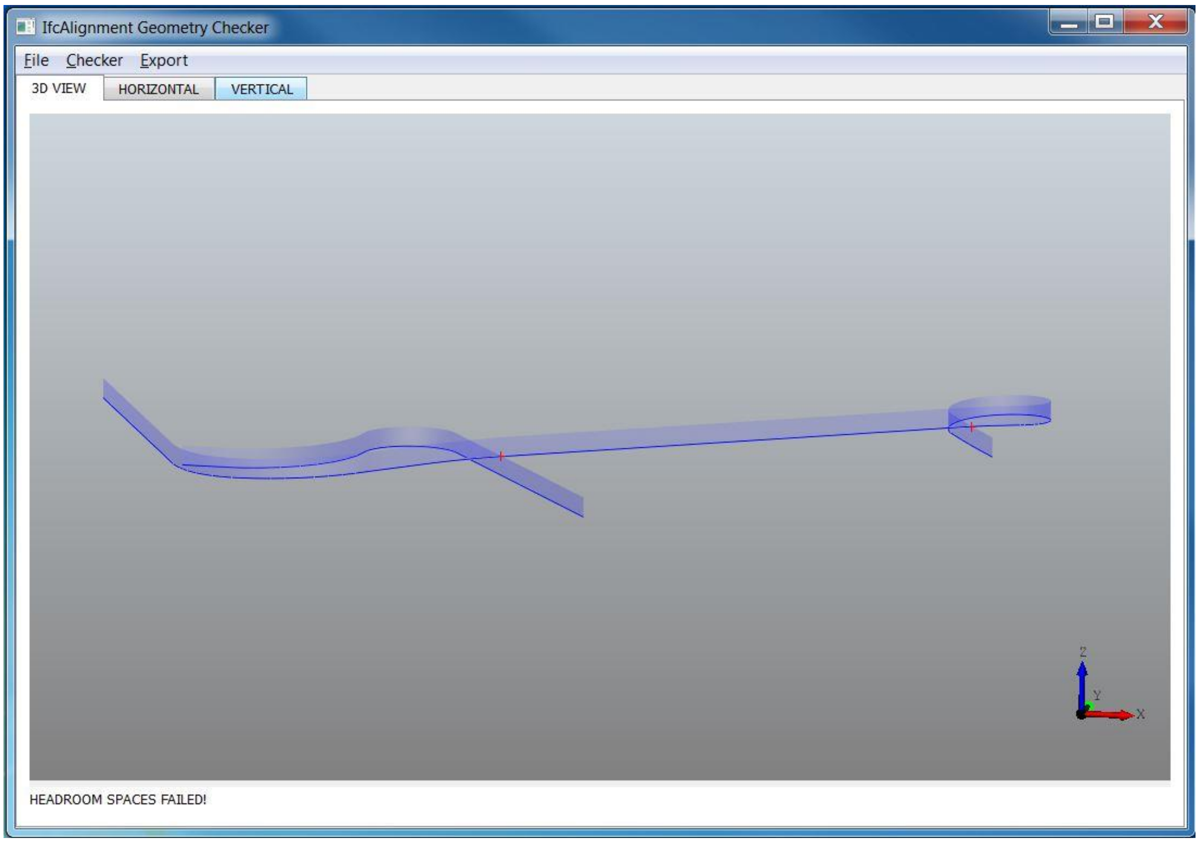
```
CHECKING LINE SEGMENT LENGTH                                                    (7)

INPUT horizontal_segments of Alignment[n]
OUTPUT an array of errors

RuleSet = {
        '120': {    'same_direction': 480,
                    'opposite_direction': 240,
                    'max_length': 2400 },
        '90' : {     'same_direction': 480,
                    'opposite_direction': 240,
                    'max_length': 2400},
        '70' : {     'same_direction': 480,
                    'opposite_direction': 240,
                    'max_length': 2400 },
        '50' : {     'same_direction': 480,
                    'opposite_direction': 240,
                    'max_length': 2400 }}

line_segments = array of line_segments

FOR each segment in horizontal_segments
        IF segment is ("IfcLineSegment2D") THEN
                    ADD line_segment to array of line_segments

        END IF
END FOR
```

```
FOR each line_segment in line_segments
    index = horizontal_segments.index( line_segment )
    IF index > 0 AND index < COUNT( horizontal_segments ) - 1 THEN
        previous_segment = horizontal_segments[ index - 1 ]
        next_segment = horizontal_segments[ index + 1 ]

        IF previous_segment.IsCCW == next_segment.IsCCW THEN
            IF line_segment.SegmentLength < RuleSet[ RoadType ][ 'same_direction' ] THEN
                ADD line_segment to array of errors
            IF line_segment.SegmentLength > RuleSet[ RoadType ][ 'max_length' ] THEN
                ADD line_segment to array of errors

            END IF

        IF previous_segment.IsCCW != next_segment.IsCCW THEN
            IF line_segment.SegmentLength < RuleSet[ RoadType ][ 'opposite_direction' ] THEN
                ADD line_segment to array of errors
            IF line_segment.SegmentLength > RuleSet[ RoadType ][ 'max_length' ] THEN
                ADD line_segment to array of errors

            END IF
        END IF
    END IF
END FOR
```

4.5.3 CLASS 3 – RULES CHECKED WITH GENERATED DATA FROM GEOMETRY

Within this class, the rules of the profile of free spaces in the horizontal and in the vertical direction are contained. The ROA states for the free vertical space a safety zone, consisting of crash safe and / or shielded objects. This space provides space for stranded vehicles and emergency- and maintenance services. These rules are constructed from the cross sections of a road body and cannot be tested solely on the alignment model. The free horizontal space is the zone were in the Z-direction there are no object. This applies for underpasses, fly-overs, crossovers, etc. This rule is measured from top asphalt to the bottom of the construction of the passage, but these spaces can be translated into the free spaces of the alignment data. For this rule the geometric model is needed to compute a surface in the Z-direction of this free space. When this surface is generated, it can be checked on intersections of the geometric model (Figure 29) and the code is presented as (8).

```
CHECKING HEADROOM                                                                   (8)

INPUT all 3DWires of Alignment file
OUTPUT an array of errors

headroom_spaces = array of spaces

FOR each 3DWire in 3DWires
    headroom_space = MakePrism( 3DWire, Vector( 0,0,4.8 ) )
    ADD headroom_space to array of spaces

FOR each headroom_space in headroom_spaces
    LOAD ShapeIntersector( headroom_space )

    FOR each 3DWire in 3DWires
        CHECK ShapeIntersector( 3DWire )
        FOR each intersection in ShapeIntersector.NbPnt
            ADD intersection to array of errors

        END FOR
    END FOR
END FOR
```



HEADROOM SPACES FAILED!

Figure 29: Checked HeadRoom with two intersections

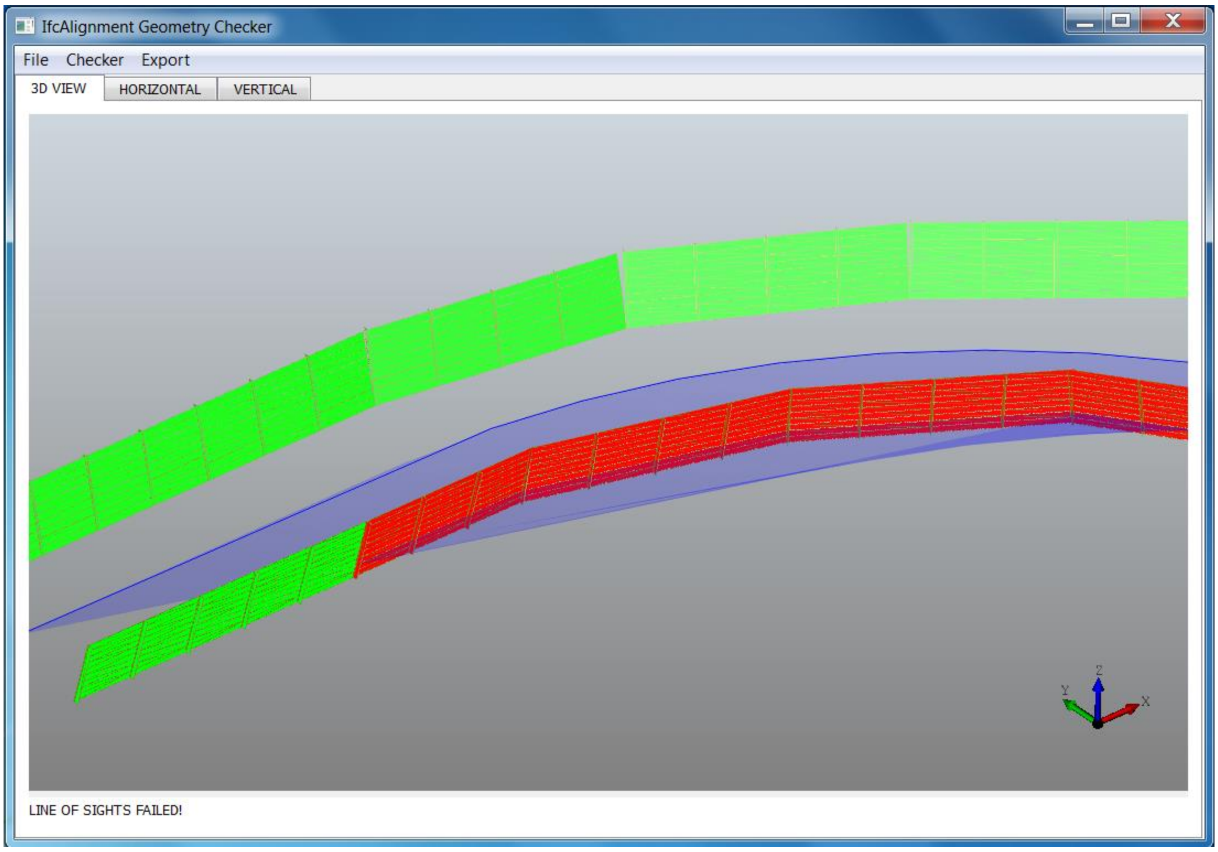
4.5.4 CLASS 4 – RULES CHECKED WITH EXTERNAL DATA STRUCTURES

Alignment models have a close relation to its surroundings and therefore have a link with Geographic Information Systems. This class of rules are topological based and checks the alignment model to its geometrical surroundings, such as ground surface, buildings, shielding facilities, signage, etc.

One of the most important rules within this class is the line of sight. There are three types with their own sub-rules, anticipation sight, road course sight and stop sight. These line of sights can differ in the strictness, for stop sight the sight cannot be interrupted and for the road course sight it can for two seconds (depending on design speed). For anticipation sight it is necessary to have small elements in the line of sight to ensure the motorist can anticipation of the course of the road. To check these line of sights the geometric alignment model and the geometric environment model has to be situated exactly according the coordinate reference system. When these models are situated, the Alignment can be offsetted with parameterized values according the rule specification and a surface can be created from the Alignment and the offsetted Alignment (9). This surface then can be checked on intersections with geometric objects from the environment file (Figure 30).

```
CHECKING LINE OF SIGHT                                                                                  (9)

INPUT all 3DWires of Alignment file
INPUT all shapes of Alignment file
OUTPUT an array of errors


ruled_surfaces = array of surfaces

FOR each 3DWire in 3DWires
        3DWireParameterized = MakeWire( 3DWire, 3DWire.FirstParameter + 100, 3DWire.LastParameter )
        ruled_surface = MakeFace( 3DWire, 3DWireParameterized )
        ADD ruled_surface to array of surfaces
```

```
END FOR

FOR each surface in ruled_surfaces
        IF surface intersects shapes THEN
                ADD intersection to array of errors

        END IF
END FOR
```



LINE OF SIGHTS FAILED!

Figure 30: Alignment with sound barriers in the line of sight

4.6 UNIT TESTS

To verify the Automated Geometry Checker, unit tests are performed to check if the prototypically implementation of the geometry checking works. Typically, this is done to existing implementations, but there are no open-source infrastructure geometry checkers. Therefore this sections presents two examples of the checker and their corresponding data and calculations if necessary.

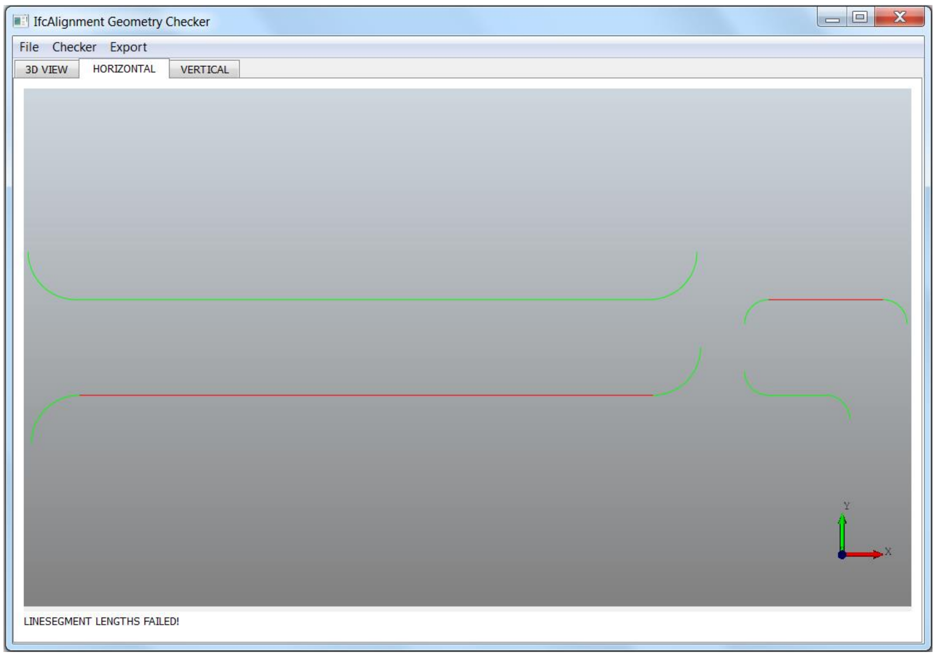
4.6.1 LINE_SEGMENT

For the use-case of the length of a line segment, four line segments are created with adjoined circular arc segments with the same or the opposite direction. The design speed 120 km/h is selected and therefore the minimum and maximum length are presented in Table 9.

Design Speed	Min same direction	Min opposite direction	Max length
120 km/h	480 m	240 m	2.400 m

Table 9: Segment length presented as the ROA (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015)

Figure 31 shows the checked line segments in the Automated Geometry Checker. From the top left, to bottom right the line segments are numbered 1 to 4. The IfcCircularArcSegment2D fifth attribute is the IsCCW which can be True or False. When this attribute of both adjoined curve is the same the direction is also the same and vice versa. The IfcLineSegment2D third attribute is the SegmentLength.



LINESEGMENT LENGTHS FAILED!

Figure 31: Checked line segments with adjoined arcs with the same or opposite directions

The data below shows the first and fourth line segment pass, with a SegmentLength respectively of 2400 and 240 meters. The second and the third line segment fail, both with 1 meter outside its range.

Line segment 1:
```
#26=IFCCIRCULARARCSEGMENT2D(#27,4.7123889803846897000,314.15926535897933000,200.00000000000000000,.F.);
#29=IFCLINESEGMENT2D(#30,3.1415926535897931000,2400.0000000000000000);
#32=IFCCIRCULARARCSEGMENT2D(#33,3.1415926535897931000,314.15926535897933000,200.00000000000000000,.F.);
```

Line segment 2:
```
#37=IFCCIRCULARARCSEGMENT2D(#38,1.5707963267948968000,157.07963267948966000,100.00000000000000000,.F.);
#40=IFCLINESEGMENT2D(#41,0.00000000000000000000,479.00000000000000000);
#43=IFCCIRCULARARCSEGMENT2D(#44,0.00000000000000000000,157.07963267948958000,99.999999999999943000,.F.);
```

Line segment 3:
```
#48=IFCCIRCULARARCSEGMENT2D(#49,4.7123889803846897000,314.15926535897984000,200.00000000000091000,.F.);
#51=IFCLINESEGMENT2D(#52,3.1415926535897931000,2401.0000000000000000);
#54=IFCCIRCULARARCSEGMENT2D(#55,3.1415926535897931000,314.15926535897933000,200.00000000000000000,.T.);
```

Line segment 4:
```
#59=IFCCIRCULARARCSEGMENT2D(#60,4.7123889803846897000,157.07963267948966000,100.00000000000000000,.T.);
#62=IFCLINESEGMENT2D(#63,0.00000000000000000000,240.00000000000000000);
#65=IFCCIRCULARARCSEGMENT2D(#66,0.00000000000000000000,157.07963267948966000,100.00000000000000000,.F.);
```

4.6.2 LINE_OF_SIGHT

For the use-case line of sight, the design speed of 50 km/h is chosen and therefore a line of sight for stopping is a minimum of 40 m. Therefore two arcs are modeled, one with a radius of 12.73 meters and one with a radius of 25.46 meters. This creates the first half circle with an outline of 40 meters and the second with an outline of 80 meters, as shown in Figure 32. With these parameters the surface of the first arc is exactly the surface of half a circle and the second is small surface track along the bigger arc.

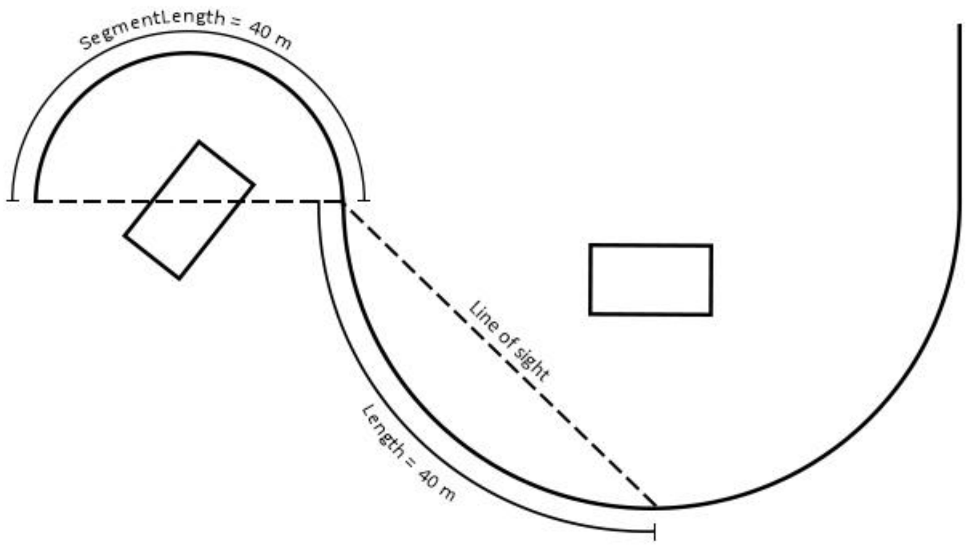
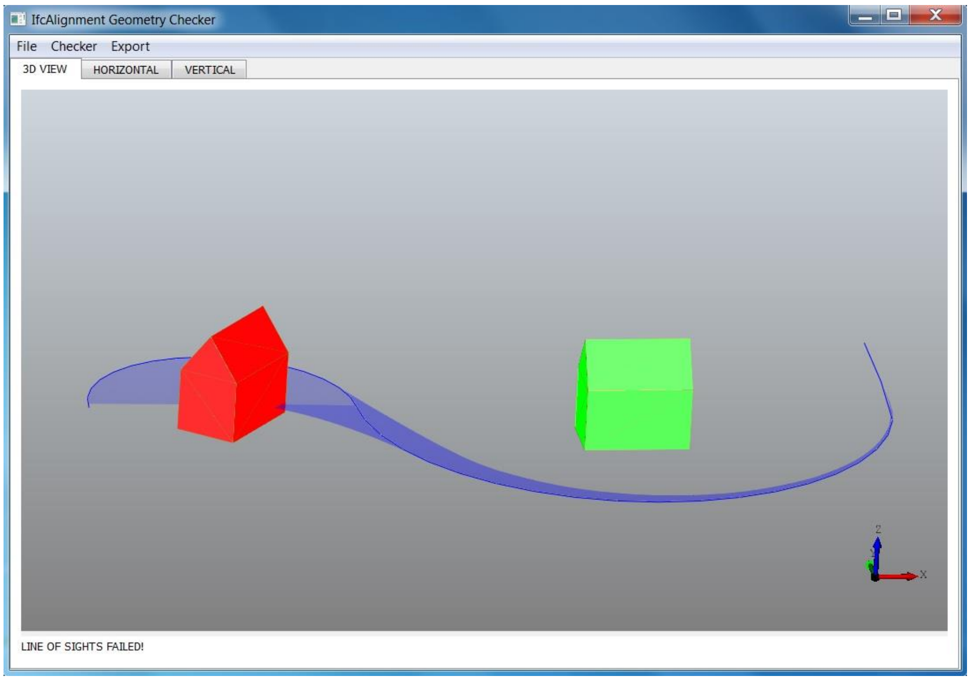


Figure 32: Calculation of surface of line of sight

The Automated Geometry Checker checks the surface with all geometry of its surroundings and will color the object red if it intersects the surface and green if it does not intersect as shown in Figure 33. #26 and #27 has as third attribute their SegmentLength.

```
#26=IFCCIRCULARARCSEGMENT2D(#27,1.5707963267948253000,40.000000000000000000, 12.732395447351626861,.F.);
#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,80.000000000000000000,25.464790894703253723,.T.);
```



LINE OF SIGHTS FAILED!

Figure 33: Checked line of sight with one object clash

4.7 *CONCLUSSION AND DISCUSSION*

This research shows the value of BIM in the Infrastructure and the role of compliance checking within it. There are numerous amount of guidelines and requirements, starting in the Planning phase of a Infrastructure project. Manual compliance checking is time-consuming and error-prone, examples are: unfamiliarity with or even lack of the guideline expertise knowledge, or being overwhelmed of the amount of guideline text, engineers own way of quality check based on experience, complexity of the regulations, etc. Therefore a prototypically Automated Geometry Checker for the Planning phase based on open source is proposed to address these issues.

4.7.1 CONCLUSION

The overview of types of model checking and rule checking platforms are based on an ontology foundation with a four level taxonomy as shown in Figure 34. These different taxonomies are included in the prototypically Alignment Checker and therefore prove the conceptual framework.

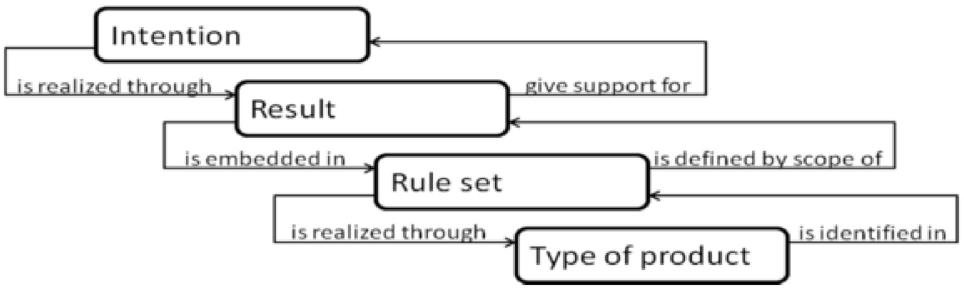


Figure 34: Ontology of types of model checking (Hjelseth & Nisbet, 2010)

The majority of existing infrastructure data standards focus on modeling linear assets and, therefore, do not adequately address the modeling of nonlinear assets, such as facilities. During the past decade, several data standards have been developed to support facilities design, construction, and management. Within the open road data models, LandXML is the most widely used model. This model has several issues, organizational and technical. Technical issues are of relevance for this research, existing of minimally documented, syntax errors and structure errors in the schema, weak point typing, case inconsistencies, name optionality inconsistency, unique identifiers inconsistency, etc. The new IfcAlignment shows also some shortcomings;

- No unique identifier of geometric elements;
- No addition information, such as design speed or superelevation;
- Insufficient specific of relative stationing.

When a road is designed, elements of a single Alignment can differ from each other. Therefore these elements should have an GUID to support exchange purposes and identification. In a latter phase of a project, information will be added and identification of an element will be necessary. For the alignment there are two properties of high importance, the design speed and the superelevation. Both are of importance to define the geometrics of road design and therefore are needed to check guidelines and requirements. Next to this the IfcAlignment uses relative stationing, which gives an error at the endpoint of an element. Within the building construction this is not a big error, due to the fact the scale of a project is much smaller compared to Infrastructure. This will be further explained in section 4.8.

However, the IFC standard represents one of the largest scale and most mature efforts to standardize facilities design and construction data. The IFC model defines a multilayer, integrated schema that represents the structure and organization of data in the form of a class hierarchy. The hierarchy covers the core project information such as building elements, the geometry and material properties of building products, project costs, schedules, and organizations (Halfawy et al., 2006). The IfcAlignment is the first step into the Infrastructure industry and there are more extensions in development, such as IfcRoad and IfcBridge. Many applications in the building industry have implemented this data schema and this is promising for the Infrastructure industry.

4.7.2 DISCUSSION

These automated model checking technologies demonstrate a great improvement to achieve an optimal design, starting from the Planning phase to realization. Examples of analyzing models include more complete and accurate performance estimates earlier in the design process, improved life-cycle costing analysis, increased opportunities for measurement and verification during operations, and improved processes for gathering lessons learned in high performance projects (Nawari, 2011).

There are some difficulties concerning these technologies. The applications interpret codes, guidelines and contract requirements into ruleset to verify the models. These rulesets are formats based on documents, written in human language with a legal status and therefore the interpretation into computer processable code is arguable. The responsibility of formalizing these documents relies on software developers, or directly into computer processable code or into formal code and then translated into computer processable code.

Another vulnerability is the derivation of data. When a model requires complex calculations or analyses on derivate data, or the application derive new data this can lead to vulnerability and legal risks.

4.8 LIMITATIONS AND FURTHER RESEARCH

From this implementation, the IfcAlignment schema is a clear data schema and easy to check. However there are some general questions and limitations on model checking and on the schema. The following sections will present these limitations.

4.8.1 MODEL CHECKING

Reusability and efficiency:
The overall time-consuming and vulnerability, as mentioned in the discussion, is the interpretation of the human language rules. This will only increase when the IfcAlignment schema will include cross section and eventually when IfcRoad will develop, due to the more complex rules which than can be checked. There will be agreements and disambiguation's that have to be made.

In the ROA 2014, there are several guidelines which are conceptual overlapping with the rules from the Handboek WegOntwerp 2013 and ASVV 2012. Therefore, there is a possibility of reusing the rules across different standards to improve efficiency. Additionally, if one has a good understanding of the IFC specifications, the checker is more easy to use than full-fledged programming languages and thus lower the threshold for usage.

Reporting:
As mentioned in section 4.4.4 the reporting should be in the BIM Collaboration Format (BCF). Referring to an object is normally done by using the Global Unique ID's (GUIDs), but for the alignment model this has to be done at geometry level. This because only the IfcAlignment entity has an GUID.

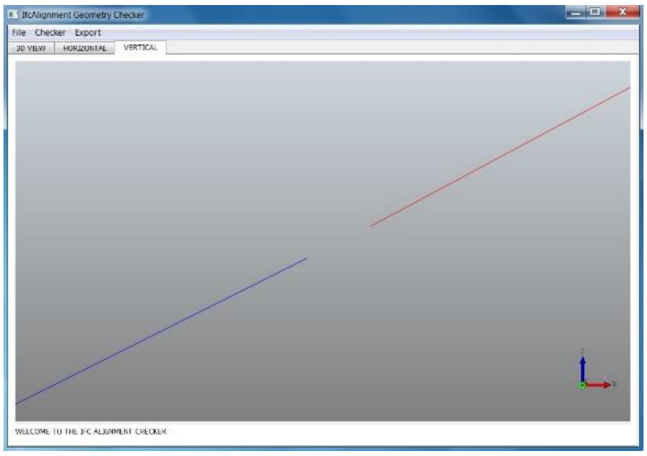
4.8.2 IFCALIGNMENT SCHEMA

The IFC schema for the building industry is based on relative stationing, which means a line segment is built from its start point, direction and length. For the AEC industry, this is a sufficient way to capture the information and to derive attributes such as its endpoint, due to the smaller geometric elements. The Infrastructure industry has to deal with large geometric elements, such as a roads of several kilometers. Due to computer processable code, there is a rounding of numbers at 12 digits. With 12 digits the rounding error is 1 : 0.00085 meters, this is calculated in (10) and shown in Figure 35. This means at a 1000 meters there is an error of 0.85 meters. If we use the 20 digits of the data schema the error would be 1 : 0.0000000000000000025. Due to processable code it could be considered to change or add absolute stationing to the IfcAlignment schema and store the end point within the geometric element.

Additionally, within the Infrastructure industry there are many transition curves to determine, see section 4.3.6. The IfcAlignment schema uses the most used transition curve, the clothoid. When considered expanding the IfcAlignment schema to other linear construction industries, such as Rail, other transition curves should be considered. A research from (Amann et al., 2014) about arbitrary transition curves could be a solution.

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,1.00000000000000000000,0.00000000000000000000,0.500000000..);
#30=IFCALIGNMENT2DVERSEGLINE($,$,$,1.00000000000000000000,1.00000000000000000000,0.50000000000000000000,0.5000000000..);
```

```
Exact                                                                    (10)
VerticalLength =   HorizontalLength * dy / dx
                   1 * sin( tan^-1(0.5) ) / cos( tan^-1(0.5) )
                   0.5

Length =           sqrt( HorizontalLength^2 + VerticalLength^2 )
                   sqrt( 1^2 + 0.5^2 )
                   1.11803398874 or 1.1180339887498948482

12 digits
VerticalLength =   HorizontalLength * dy / dx
                   1 * sin( tan^-1(0.5) ) / cos( tan^-1(0.5) )
                   0.49808839664

Length =           sqrt( HorizontalLength^2 + VerticalLength^2 )
                   sqrt( 1^2 + 0.49808839664^2 )
                   1.11718040211


20 digits
VerticalLength =   HorizontalLength * dy / dx
                   1 * sin(0.46364760900080611621) / cos(0.46364760900080611621)
                   1 *    0.44721359549995793928  /    0.89442719099991587856
                   0.49999999999999999944

Length =           sqrt( HorizontalLength^2 + VerticalLength^2 )
                   sqrt( 1^2 + 0.49999999999999999944^2 )
                   1.1180339887498948457
```


IfcAlignment Geometry Checker
File Checker Export
3D VIEW HORIZONTAL VERTICAL
WELCOME TO THE IFC ALIGNMENT CHECKER

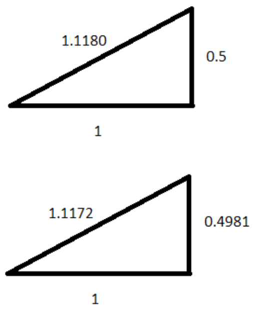


Figure 35: Rounding error 1 : 0.00085 shown in the Checker and schematic

4.9 REFERENCES

The references of the scientific article are listed in the references of the thesis.

5 CONCLUSION AND RECOMMENDATIONS

The Infrastructure and the Architecture, Engineering and Construction industry are both not characterized as innovative industries. However, Building Information Modeling has introduced a significant change in the construction process of the AEC industry. The potential benefits are now acknowledged and therefore the Infrastructure industry is adopting the process. This research provides a sufficient description of benefits, barriers, opportunities and limitations of adopting Building Information Modeling in the Infrastructure industry and specifies on Alignment Model Checking. The research's main objective is to explore how the validation of official regulations of roads can be automated in the Planning phase of the System Engineering process, based on open standards and software. However, before one investigates such possibilities, it is crucial to understand the current process of validation of these official regulations. When this knowledge was gathered a prototypical implementation of an Alignment Geometry Checker is developed to answer the main research question.

5.1 CONLUSION

Theoretical background on Building Information Modeling in the AEC industry is necessary, to identify the possible potentials and barriers for the Infrastructure industry. Literature states BIM is ready for bigger and more complex projects and therefore ready to be adopted by the Infrastructure industry. One of the most important processes within complex projects is the validation of its guidelines and contract requirements. Concluding from the conducting semi-structured interviews, most of this validation is still done by manual checking. Manual checking consist of stacks of drawing and excel sheets and looking them over for errors. There are some contractors who already use some sort of clash detection, mostly to clash models from several disciplines. However, the guidelines and contract requirements are still checked manually. This compliance checking is a time-consuming and error-prone tasks, which completely reflects the knowledge of the engineer. There are several issues arising from this:

- Unfamiliarity with or even lack of the guideline knowledge, or
- Being overwhelmed of the amount of guideline and contract requirement text,
- Engineers own way of quality check based on experience,
- Complexity of the regulations.

Due to the increasing size and complexity of project, the need for computerized compliance checking is confirmed by reviewing the literature and by the conducted interviews. It would improve quality compliance checking and reduce violations to guidelines and contract requirements. In terms of research objectives, the need for automated compliance checking is confirmed. Compliance checking should be done from the moment the first line is drawn onto a map, to create a solid basis for later stages.

In contrast to the AEC industry, the Infrastructure industry does not have official regulations for the geometrics of a road design. There are several guidelines for the geometrics, such as the Richtlijn Ontwerp Autosnelwegen (ROA) 2014, the Handboek Wegontwerp 2013 and the Aanbevelingen voor verkeervoorzieningen binnen de bebouwde kom (ASVV) 2012. These are included into the contracts and sometimes the boundaries of these guidelines are specifically adjusted for a project. The classification of these guidelines are broadly structured in Horizontal alignment, Vertical alignment, Cross sections, Discontinuity and Line of Sight. The geometric adjustments and geometric additions within the contract requirements are also classified according. These classification all have their level of detail and therefore these cannot all be checked to the alignment model in the Planning phase. Figure 36 represent where the guidelines have to be checked in the System Engineering process.

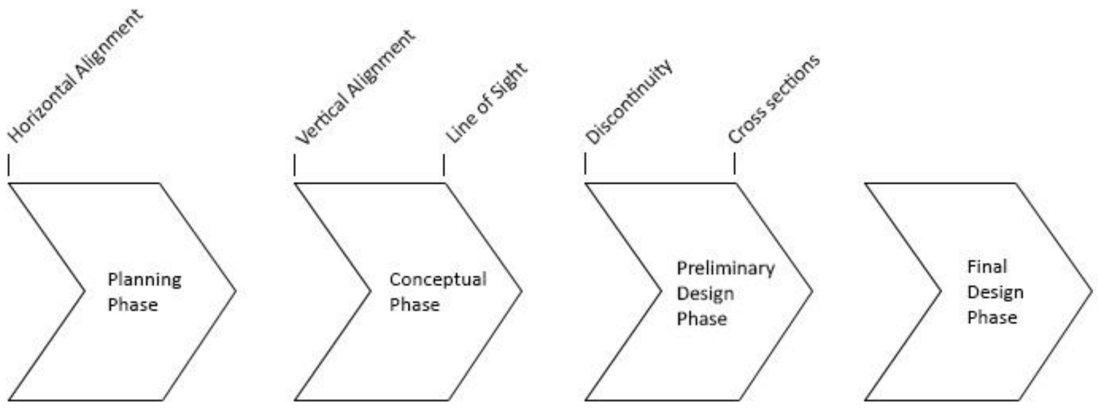


Figure 36: Guidelines checked in different phases of the SE process

The second part of this research implements a prototypically implementation of an Alignment Geometry Checker. Thorough literature review provided a better understanding of how to structure data so that this is meaningful to others and easy to use by multiple platforms. The information of road design is structured in Infrastructure data standards. The majority of existing Infrastructure data standards focusses on modeling linear assets and, therefore, do not adequately address the modeling of nonlinear assets, such as facilities. During the past decade, several data standards have been developed to support facilities design, construction, and management. The Industry Foundation Classes (IFC) is a data model which is a promising current development supporting open source initiative. For the AEC industry it is highly adopted and for the Infrastructure industry, an alignment model is added, IfcAlignment. There are also some initiatives to develop IfcRoad and IfcBridge.

In terms of research objectives, the requirements, which directly can be checked, are these of classes Horizontal alignment, Vertical alignment and Line of Sight. For these classes, most of the requirements can be checked. However, the IfcAlignment schema does not include design speed and superelavation. These properties are also of high importance for each element in the geometrics of road design, due to several rules in the guidelines. These attributes are specifically for the alignment model and included these should be considered. For the other two classes there is additional information needed, such as a road cross section and convergence and divergence of traffic lanes. These should be added into the development of IfcRoad. The classification of the requirements are converted into an computer processable classification, based on complexity of the rules processing. These are presented in 4.4.1. The calculations and inferences of the geometry contained, calculated surface, intersection, min and max calculations, etc. The formalization is done in a non-formal language pseudocode and is presented in 4.4 and 4.5.

The IFC standard represents one of the largest scale and most mature efforts to standardize facilities design and construction data. The IFC model defines a multilayer, integrated schema that represents the structure and organization of data in the form of a class hierarchy. The hierarchy covers the core project information such as building and elements infrastructure geometry, and material properties of products, project costs, schedules, and organizations (Halfawy et al., 2006). The IfcAlignment is the first step into the Infrastructure industry and there are more extensions in development, such as IfcRoad and IfcBridge. Many applications in the AEC industry have implemented this data schema and this is promising for the Infrastructure industry. This prototypical implementation could be a starting point in Infrastructure compliance checking.

5.2 RECOMMENDATIONS AND FUTURE RESEARCH

The research has shown general organizational and technical implementation of BIM and model checking for the Infrastructure industry. The barriers presented on product, process and people have to be addressed in order to achieve full implementation in the Infrastructure industry. In general the Infrastructure industry is an conservative industry, which requires exploration of actual (real, tangible, measurable) gains witnessed by those who have already adopted collaborative way of working before completely adopting BIM. For model checking there are some recommendations to be made.

There are some legal issues to address, such as the interpretation of guidelines and contract requirements into formal code. The guidelines and requirements are written on human language and have a legal status and therefore the interpretation into formal code is vulnerable for legal issues. The responsibility of formalizing these documents relies on software developers, or directly into computer processable code or into formal code and then translated into computer processable code. Another legal issue is the derivation of data. When a model requires complex calculations or analyses on derivate data, or the application derive new data this can lead to vulnerability and legal risks.

For the reporting of the checked data, the BIM Collaboration Format is proposed. Before this can be achieved the IfcAlignment schema has to be included as a standard into the BCF file format. Additionally the referring to an issues is normally done to an unique identifier of an object and within the IfcAlignment this has to be done at geometric level.

The Alignment Geometry Checker is based on the IfcAlignment standard and for this to fully work there are some issues to address. The properties design speed and superelevation are of high importance for checking according the guidelines and requirements. Therefore these properties has to be considered for adoption within the IfcAlignment schema. Also the relative stationing (line segment built from its start point, direction and length) and absolute station (line segment built from start point and endpoint) has to be considered, due to the small differences in computing other attributes. The last part for future research is the geometric attributes of transition curves. For road design the clothoid is the common used transition curve, but for other linear disciplines there are other transition curves used. The geometrics of these curves are challenging to compute and to capture in a data schema, without risking the quality of the data schema.

6 REFERENCES

Amann, J., Borrmann, A., Chipman, T., Lebeque, E., Liebich, T., & Scarponcini, P. (2014). P6 IFC Alignment Project – Conceptual Model. *buildingSMART*, (1), 1–22.

Amann, J., Flurl, J., Jubierre, R., & Borrmann, A. (2014). An Approach to Describe Arbitrary Transition Curves in an IFC Based Alignment Product Data Model.

Amann, J., Jubierre, J. R., & Borrmann, A. (2008). An alignment meta-model for the comparison of alignment product models.

Azhar, S., Hein, M., & Sketo, B. (2007a). Building Information Modeling (BIM): Benefits , Risks and Challenges.

Azhar, S., Hein, M., & Sketo, B. (2007b). Building Information Modeling (BIM): Benefits , Risks and Challenges. *Building Science*, *18*, 11. http://doi.org/10.1061/(ASCE)LM.1943-5630.0000127

Board of RoadXML. (2013). RoadXML file format, The open format for road networks. Retrieved from http://www.road-xml.org/

Borrmann, A., & Rank, E. (2009). Topological analysis of 3D building models using a spatial query language. *Advanced Engineering Informatics*, *23*(4), 370–385. http://doi.org/10.1016/j.aei.2009.06.001

Bryde, D., Broquetas, M., & Volm, J. M. (2013). The project benefits of building information modelling (BIM). *International Journal of Project Management*, *31*(7), 971–980. http://doi.org/10.1016/j.ijproman.2012.12.001

BuildingSMART. (2007). National BIM Standard - United States ™ Version 2, 182. Retrieved from www.nationalbimstandard.org/nbims-us-v2/pdf/NBIMS-US2_aB.pdf

Cerri, D., & Fuggetta, A. (2007). Open standards, open formats, and open source. *Journal of Systems and Software*, *80*, 1930–1937. http://doi.org/10.1016/j.jss.2007.01.048

Chaplier, J., That, T. N., Hewatt, M., Gallée, G., Sa, O., Unies, N., & France, M. (2012). Toward a standard : RoadXML , the road network database format. *Driving Simulation Conference 2010 Europe*, 211–220.

Choi, J., & Kim, I. (2008). An Approach to Share Architectural Drawing Information and Document Information for Automated Code Checking System, *13*(October), 171–178.

Conover, D. (2007). Development and Implementation of Automated Code Compliance Checking. *International Code Council*.

Czmoch, I., & Pękala, A. (2014). Traditional Design versus BIM Based Design. *Procedia Engineering*, *91*(TFoCE), 210–215. http://doi.org/10.1016/j.proeng.2014.12.048

De Vries, H. (2005). IT Standards Typology. *Advanced Topics in Information Technology Standards and Standardization Research*, *1*, 11–36.

Ding, L., Drogemuller, R., Rosenman, M., & Marchant, D. (2006). Automating code checking for building designs - DesignCheck, 1–16.

Ducloux, P., & Millet, G. (2009). Road Network Description XML Format Specification, 1–88.

Eadie, R., Browne, M., Odeyinka, H., McKeown, C., & McNiff, S. (2013). BIM implementation throughout the UK construction project lifecycle: An analysis. *Automation in Construction*, *36*, 145–151. http://doi.org/10.1016/j.autcon.2013.09.001

Eastman, C., Lee, J. M., Jeong, Y. S., & Lee, J. K. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, *18*(8), 1011–1033. http://doi.org/10.1016/j.autcon.2009.07.002

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM Handbook: A guide to Building Information Modeling for Owners, Managers, Designer,Engineers and Contractors*.

Erstling, R., & Portele, C. (1996). Standardisierung graphischer Daten im Straßen- und Verkehrswesen. Teil 1 – Studie. Forschung Straßenbau und Straßenverkehrstechnik 724, Bundesministerium für Verkehr, Abteilung Straßenbau, Bonn-Bad Godesberg.

Garrett, J. H., & Fenves, S. J. (1988). A knowledge-based standards processor for structural component design, *40*(1), 1988.

Grilo, A., & Jardim-Goncalves, R. (2010). Value proposition on interoperability of BIM and collaborative working environments. *Automation in Construction*, *19*(5), 522–530. http://doi.org/10.1016/j.autcon.2009.11.003

Halfawy, M. R., Vanier, D. J., & Froese, T. M. (2006). Standard data models for interoperability of municipal infrastructure asset management systems. *Canadian Journal of Civil Engineering*, *33*(2006), 1459–1469. http://doi.org/10.1139/l05-098

Han, C. S., Kunz, J. C., & Law, K. H. (1998). A Client / Server Framework for On-line Building Code Checking, 1–40.

Heinen, J. (2015). The added value of Building Information Models in the Operation & Maintenance processes . Mapping between Building Information Models and a Facility. *Graduation Thesis*.

Hjelseth, E., & Nisbet, N. (2010). Overview of concepts for model checking. *Proceedings of the CIB W78 2010: 27th International Conference –Cairo, Egypt*, 16–18.

Integrated Project Delivery. (2004). The MacLeamy Curve, 2004. Retrieved from http://www.msa-ipd.com/MacleamyCurve.pdf

International, B. (2007). The buildingSMART Glossary of Terms, (January).

Jones, S. a., & Bernstein, H. M. (2012). *The Business Value of BIM for Infrastructure Addressing America's Infrastructure Challenges with Collaboration and Technology*. *McGraw-Hill Construction SmartMarket Report* (pp. 1–64).

Jotne. (2015). EDM ModelChecker. Retrieved from http://www.epmtech.jotne.com/index.php?id=512200

Jubierre, J. R., & Borrmann, A. (2013). Flexible linking of semantic , procedural and logic models for consistent multi-scale infrastructure design.

Karan, E. P., & Irizarry, J. (2015). Extending BIM interoperability to preconstruction operations using geospatial analyses and semantic web services. *Automation in Construction*, *53*, 1–12. http://doi.org/10.1016/j.autcon.2015.02.012

Khemiani, L. (2005). CORENET e-PlanCheck: Singapore ' s Automated Code Checking System. *AECbytes, Building the Future*, 1–8.

Krijnen, T. (2013). IfcOpenShell. Retrieved from www.ifcopenshell.org

Laakso, M., & Kiviniemi, A. (2012). The IFC standard - A review of history, development, and standardization. *Electronic Journal of Information Technology in Construction*, *17*(May), 134–161.

Moody, D. (2002). Empirical research methods, 1–4.

Muz, R. (2014). " Extending Building Information Modeling beyond Design / Construction project phase ." *Graduation Thesis*.

Nawari, O. (2011). A framework for automating codes conformance in structural domain a framework for automating codes conformance in structural domain.

Nawari, O. (2012). Automated Code Checking in BIM Environment. *Proceedings of 14th International Conference on Computing in Civil and Building Engineering*, *1*(Aashto 1998), 1–8. http://doi.org/10.1061/9780784412343.0036

Ning, G. U., Vishal, S., Kerry, L., Ljiljana, B., & Claudelle, T. (2008). BIM : Expectations and a Reality Check.

Potter, W. D., Trueblood, R. P., & Carolina, S. (1988). Hyper-Semantic Approaches to Data Modeling, (June).

Rebolj, D., Tibaut, A., Čuš-Babič, N., Magdič, A., & Podbreznik, P. (2008). Development and application of a road product model. *Automation in Construction*, *17*, 719–728. http://doi.org/10.1016/j.autcon.2007.12.004

Rijkswaterstaat GPO m.m.v Witteveen + Bos. (2015). Richtlijn Ontwerp Autosnelwegen 2014.

Robbin, J. (2006). *Mathematical Logic: A First Course*. Dover Publicaitons.

Royal Institution of Chartered Surveyors. (2013). RICS 2013 Building Information Modelling Survey Report, *44*(0), 1–31.

Scarponcini, P. (2013). InfraGML Proposal (13-121), 2–7.

Skibniewski, M. J. (2016). Automated in Construction. *Automated in Construction*, 2016. Retrieved from http://www.journals.elsevier.com/automation-in-construction/

Solibri. (2015). Solibri Model Checker. Retrieved from http://www.solibri.com

Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction*, *53*, 69–82. http://doi.org/10.1016/j.autcon.2015.03.003

Stangeland, B. K. (2011). BIM Collaboration Format, 1–3.

Strafaci, A. (2008). What does BIM mean for civil engineers? Road and highway project can benefit from design using building information modeling, 5–9.

Technical Universität München. (2015). TUM Open Infra Platform, Lehrstuhl für Computergestützte Modellierung und Simulation, 2015. Retrieved from https://www.cms.bgu.tum.de/de/forschung/projekte/31-forschung/projekte/397-tum-open-infra-platform

The Institute of Electrical and Electronics Engineers. (1990). *IEEE Standard Computer Dictionary: A Compilation OF IEEE Standard Computer Glossaries*.

Venugopal, M., Eastman, C. M., Sacks, R., & Teizer, J. (2012). Semantics of model views for information exchanges using the industry foundation class schema. *Advanced Engineering Informatics*, *26*(2), 411–428. http://doi.org/10.1016/j.aei.2012.01.005

Volk, R., Stengel, J., & Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings - Literature review and future needs. *Automation in Construction*, *38*, 109–127. http://doi.org/10.1016/j.autcon.2013.10.023

Werkgroep Leidraad SE. (2013). Leidraad voor System Engineering binnen de GWW-sector.

Wilson, C. (2014). Interview Techniques for Ux Practitioners. *Interview Techniques for Ux Practitioners*, 23–41. http://doi.org/10.1016/B978-0-12-410393-1.00002-8

Zhong, B. T., Ding, L. Y., Luo, H. B., Zhou, Y., Hu, Y. Z., & Hu, H. M. (2012). Ontology-based semantic modeling of regulation constraint for automated construction quality compliance checking. *Automation in Construction*, *28*, 58–70. http://doi.org/10.1016/j.autcon.2012.06.006

7 APPENDICES

APPENDIX A – INTERVIEW QUESTIONNAIRE

1. GOAL

- Gathering knowledge about BIM usage in Infrastructure.
- Gathering knowledge about the need of automated design checking.

2. SUMMARY

Regulations together with the requirements of a contract, play a major role in assuring the quality within the Infrastructure. It is of high importance to inspect the construction process according to these regulations and requirements. Manual construction quality compliance checking and final inspection are time-consuming and error-prone, due to a lot of reasons. Examples of these reasons are: unfamiliarity with or even lack of the regulations, being overwhelmed by the amount of regulatory text, inspectors own way of quality checking based on experience or complexity of the regulations. (Nawari, 2012; Zhong et al., 2012).

The need for computerizing the construction regulations and automating the compliance checking is becoming more critical. When implemented it will reduce quality inspection errors, which consequently improves quality compliance and reduces violations to the regulations. The Dutch Directorate General for Public Works and Water Management stated that from the moment the first line is drawn onto a map, there has to be compliance checking according to the construction regulations to avoid problems in a later stage. In addition to this compliance checking of construction regulations, there is also a set of requirements within the contract which need to be checked. These requirements, when met, also improve the quality of the construction and should be checked. Some of the compliance checks of construction regulations and contract requirements should be checked in an earlier stage than other, due to the specifics of these regulations and requirements.

Within the Infrastructure the alignment is developed, the baseline for further projects, such as IfcRoad and IfcBridge. The alignment provides alignment data for spatial location of Infrastructure assets. Most of the compliance rules, which are of geometric nature, can be checked with data from IfcAlignment. The manual compliance checks are time-consuming and error-prone, so the need for computerized compliance checks is high. To computerize the compliance checking process, it has to be analyzed whether the data from IfcAlignment can directly be extracted from the model or whether any calculations or rationalizations are necessary. If there are compliance rules which cannot be extracted from the IfcAlignment, it should be possible to appoint these to future IFC related extensions. After all data is analyzed and the code is generated, the code needs to be formalized so it can be used by other software programs.

Nawari, O. (2012). Automated Code Checking in BIM Environment. *Proceedings of 14th International Conference on Computing in Civil and Building Engineering*, *1*(Aashto 1998), 1–8. http://doi.org/10.1061/9780784412343.0036

Zhong, B. T., Ding, L. Y., Luo, H. B., Zhou, Y., Hu, Y. Z., & Hu, H. M. (2012). Ontology-based semantic modeling of regulation constraint for automated construction quality compliance checking. *Automation in Construction*, *28*, 58–70. http://doi.org/10.1016/j.autcon.2012.06.006

3. INTERVIEW QUESTIONAIRE

3.1. GENERAL

a) What is your roll in the company?
b) What is your perception of BIM in the Infrastructure industry?
c) Have you used BIM in previous/ current projects; if so in what form?

d) What are the potential benefits of implementing BIM in a project?
e) What are the barriers to implementation of BIM in a project?
f) Where do the opportunities for the implementation of BIM in the Infrastructure industry lie?
g) What threats are there to the implementation of BIM in the Infrastructure industry?

3.2. REGULATIONS & REQUIREMENTS

a) What guidelines and codes can be applicable for road designs?
b) Are there guidelines and codes which have to be variable for checking?
   I. If so, which guidelines are this, in what form do these guidelines have to variable, and why?

c) What requirements can be extracted from a Design and Build contract?
d) Can these requirements be classified?

3.3. MODEL CHECKING

a) What is the current (traditional) method of design checking?
b) How would you rate the efficiency, of this method of design checking on a scale from one to ten, and why?
c) What kind of mistakes are made when using this method of design checking and what causes these mistakes?

d) Is the current (traditional) method of design checking divided in phases?
   I. If so, which requirements are checked in which phase?
e) What is the current way of presenting the checked design?
f) How can this way of presenting be improved?

g) What are the potential benefits of automated design checking?
h) What are the potential weaknesses of automated design checking?

3.4. STANDARDS

a) Which data models are you working with in the Infrastructure industry?
b) Are these data models interoperable with each other and in what form?
   I. Do you experience problems with data interoperability and if so which?

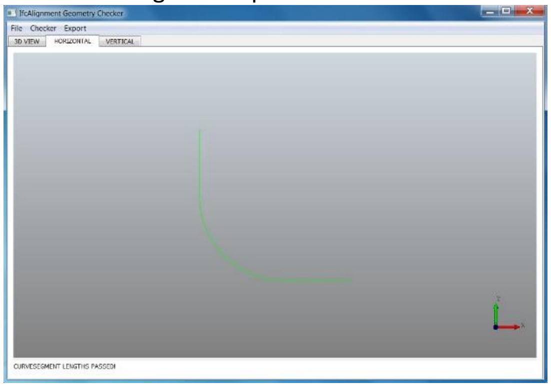
APPENDIX B – UNIT TESTS

CURVE SEGMENT LENGTH

Design Speed	Minimal curve length
120 km/h	100 m
90 km/h	75 m
70 km/h	60 m
50 km/h	40 m

Table 10: Minimal curve length (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015)

Curve segment tested at 120 km/h. Third attribute of IfcCircularArcSegment2D is the SegmentLength. The circular arc segment is passed with 100 meters.

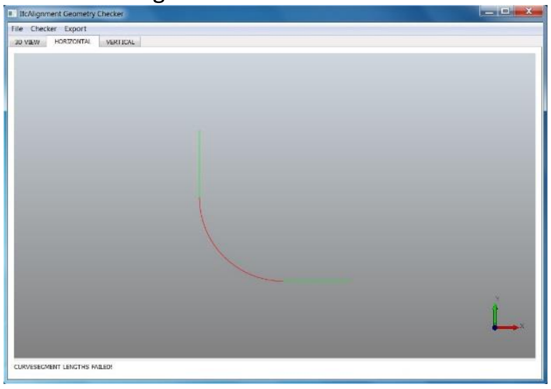


```
CURVESEGMENT LENGTHS PASSED!
```

Figure 37: Curve length of 100 meters at 120 km/h

```
#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,100.00100000000080000,63.662613856531010000,.T.);
```

Curve segment tested at 120 km/h. Third attribute of IfcCircularArcSegment2D is the SegmentLength. The circular arc segment is failed with 99 meters.

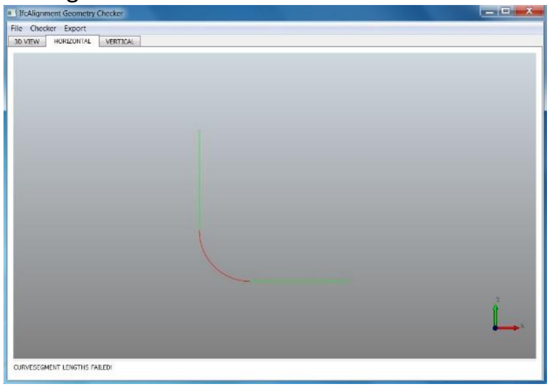


```
CURVESEGMENT LENGTHS FAILED!
```

Figure 38: Curve length of 99 meters at 120 km/h

```
#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,99.000000000000725000,63.025357464391021000,.T.);
```

Curve segment tested at 90 km/h. Third attribute of IfcCircularArcSegment2D is the SegmentLength. The circular arc segment is failed with 60 meters.

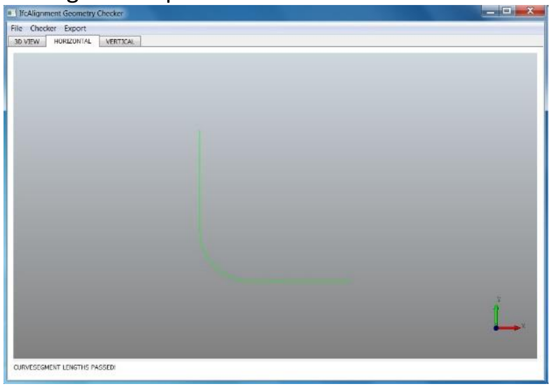


CURVESEGMENT LENGTHS FAILED!

Figure 39: Curve length of 60 meters at 90 km/h

#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,60.000000000000192000,38.197186342055005000,.T.);

Curve segment tested at 70 km/h. Third attribute of IfcCircularArcSegment2D is the SegmentLength. The circular arc segment is passed with 60 meters.



CURVESEGMENT LENGTHS PASSED!

Figure 40: Curve length of 60 meters at 70 km/h

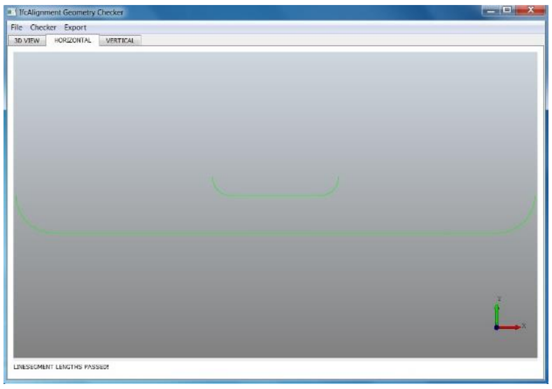
#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,60.000000000000192000,38.197186342055005000,.T.);

LINE SEGMENT LENGTH

Design Speed	Min same direction	Min opposite direction	Max length
120 km/h	480 m	240 m	2.400 m
90 km/h	360 m	180 m	1.800 m
70 km/h	280 m	140 m	1.400 m
50 km/h	200 m	100 m	1.000 m

Table 11: Segment length presented as the ROA (Rijkswaterstaat GPO m.m.v Witteveen + Bos, 2015)

Line segment tested on 120 km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are passed, respectively to the 480 meters and 2400 meters by adjoined same direction circular arcs.

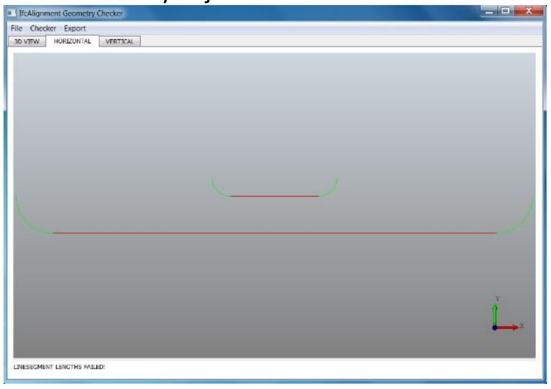


LINESEGMENT LENGTHS PASSED!

Figure 41: Passed line segments at 120 km/h

```
#26=IFCCIRCULARARCSEGMENT2D(#27,4.7123889803846799000,157.07963267949066000,100.00000000000000000,.T.);
#29=IFCLINESEGMENT2D(#30,0.00000000000000000000,480.00000000000023000);
#32=IFCCIRCULARARCSEGMENT2D(#33,6.2831853071795862000,157.07963267948966000,100.00000000000000000,.T.);

#37=IFCCIRCULARARCSEGMENT2D(#38,4.7123889803846897000,314.15926535897933000,200.00000000000000000,.T.);
#40=IFCLINESEGMENT2D(#41,0.00000000000000000000,2400.0000000000000000);
#43=IFCCIRCULARARCSEGMENT2D(#44,6.2831853071795862000,314.15926535897842000,200.00000000000000000,.T.);
```

Line segment tested on 120 km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are failed, respectively to the 479 meters and 2401 meters by adjoined same direction circular arcs.

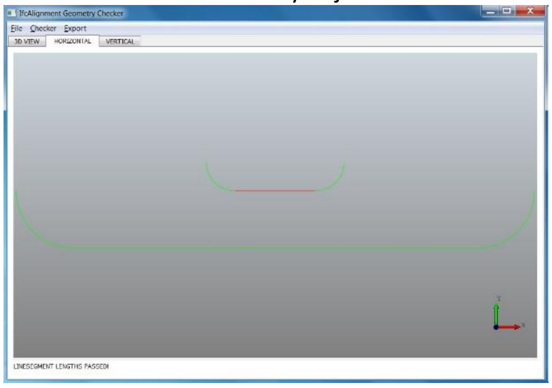


LINESEGMENT LENGTHS FAILED!

Figure 42: Failed line segments at 120 km/h

```
#26=IFCCIRCULARARCSEGMENT2D(#27,4.7123889803846897000,157.07963267949029000,100.00000000000114000,.F.);
#29=IFCLINESEGMENT2D(#30,3.1415926535897931000,479.00000000000000000);
#32=IFCCIRCULARARCSEGMENT2D(#33,3.1415926535897931000,157.07963267949023000,99.999999999999005000,.F.);

#37=IFCCIRCULARARCSEGMENT2D(#38,4.7123889803846897000,314.15926535897933000,200.00000000000000000,.T.);
#40=IFCLINESEGMENT2D(#41,0.00000000000000000000,2401.0000000000000000);
#43=IFCCIRCULARARCSEGMENT2D(#44,6.2831853071795862000,314.15926535897842000,200.00000000000000000,.T.);
```

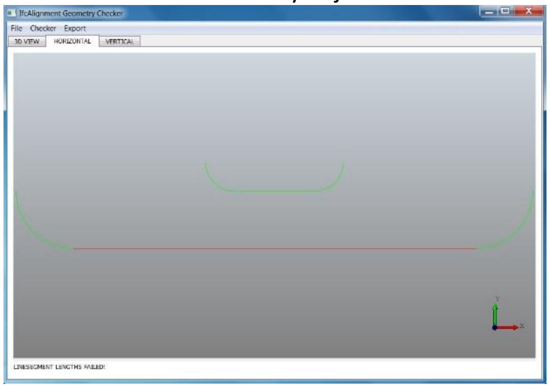
Line segment tested on 90 km km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are failed and passed, respectively to the 280 meters and 1401 meters by adjoined same direction circular arcs.



LINESEGMENT LENGTHS PASSED!

Figure 43: Failed and passed line segments at 90 km/h

Line segment tested on 70 km km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are passed and failed, respectively to the 280 meters and 1401 meters by adjoined same direction circular arcs.

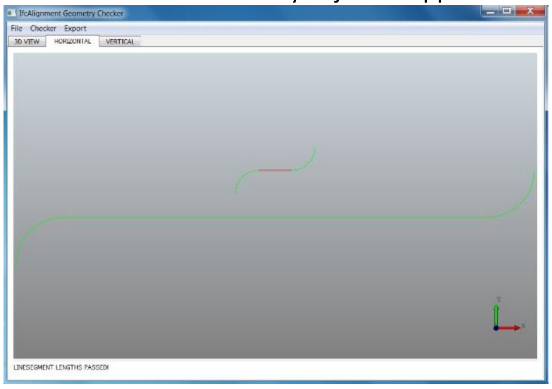


LINESEGMENT LENGTHS FAILED!

Figure 44: Passed and failed line segments at 70 km/h

```
#26=IFCCIRCULARARCSEGMENT2D(#27,4.7123889803846897000,157.07963267948966000,100.00000000000000000,.T.);
#29=IFCLINESEGMENT2D(#30,0.00000000000000000000,280.00000000000000000);
#32=IFCCIRCULARARCSEGMENT2D(#33,6.2831853071795862000,157.07963267948864000,100.00000000000000000,.T.);

#37=IFCCIRCULARARCSEGMENT2D(#38,4.7123889803846897000,314.15926535897933000,200.00000000000000000,.T.);
#40=IFCLINESEGMENT2D(#41,0.00000000000000000000,1401.0000000000000000);
#43=IFCCIRCULARARCSEGMENT2D(#44,6.2831853071795862000,314.15926535897933000,200.00000000000000000,.T.);
```

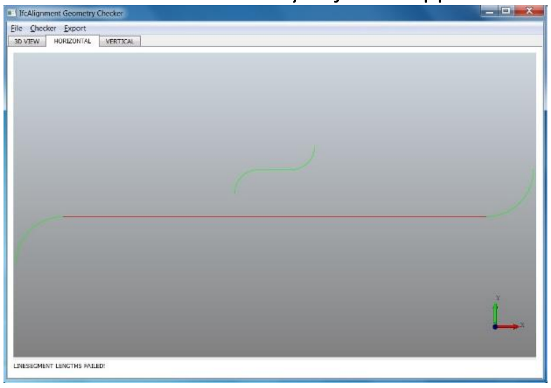
Line segment tested on 90 km km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are failed and passed, respectively to the 140 meters and 1800 meters by adjoined opposite direction circular arcs.



LINESEGMENT LENGTHS PASSED!

Figure 45: Failed and passed line segments with opposite direction circular arcs at 90 km/h

Line segment tested on 70 km km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are passed and failed, respectively to the 140 meters and 1800 meters by adjoined opposite direction circular arcs.

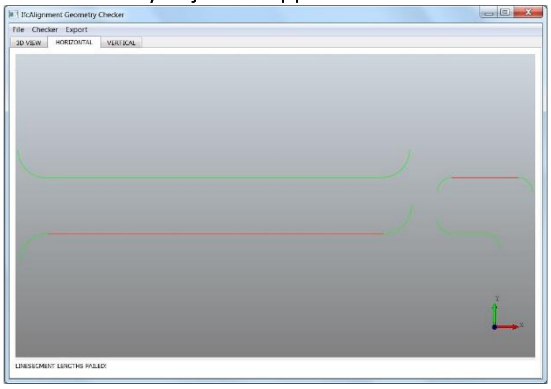


LINESEGMENT LENGTHS FAILED!

Figure 46: Failed and passed line segments with opposite direction circular arcs at 70 km/h

```
#26=IFCCIRCULARARCSEGMENT2D(#27,1.5707963267948968000,157.07963267948966000,100.00000000000000000,.F.);
#29=IFCLINESEGMENT2D(#30,0.00000000000000000000,140.0000000000000000);
#32=IFCCIRCULARARCSEGMENT2D(#33,6.2831853071795862000,157.07963267948961000,99.999999999999972000,.T.);

#37=IFCCIRCULARARCSEGMENT2D(#38,4.7123889803846897000,314.15926535897967000,200.00000000000023000,.F.);
#40=IFCLINESEGMENT2D(#41,3.1415926535897931000,1800.0000000000000000);
#43=IFCCIRCULARARCSEGMENT2D(#44,3.1415926535897931000,314.15926535897933000,200.00000000000000000,.T.);
```

Line segment tested on 120 km km/h. Third attribute of IfcLineSegment2D is the SegmentLength and the fifth attribute of IfcCircularArcSegment is the IsCcw. The line segments are numbered from top left to bottom right, 1 to 4. The first line segment passed, due to the 2400 meters by adjoined same direction circular arcs. The second line segment failed, due to the 479 meters by adjoined same direction circular arcs. The third line segment failed, due to the 2401 meters by adjoined opposite direction circular arcs. The fourth line segment passed, due to the 240 meters by adjoined opposite direction circular arcs.



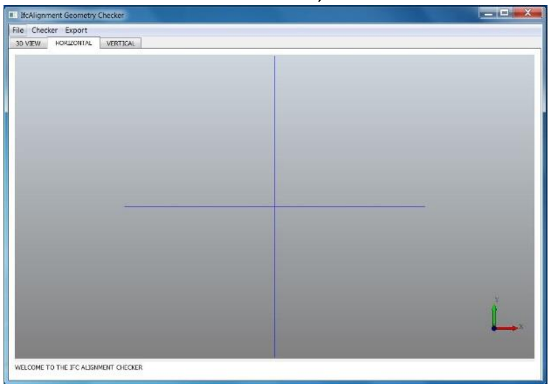
LINESEGMENT LENGTHS FAILED!

Figure 47: Mixed passed and failed line segments with same and opposite direction circular arcs at 120 km/h

```
#26=IFCCIRCULARARCSEGMENT2D(#27,4.7123889803846897000,314.15926535897933000,200.00000000000000000,.F.);
#29=IFCLINESEGMENT2D(#30,3.1415926535897931000,2400.0000000000000000);
#32=IFCCIRCULARARCSEGMENT2D(#33,3.1415926535897931000,314.15926535897933000,200.00000000000000000,.F.);

#37=IFCCIRCULARARCSEGMENT2D(#38,1.5707963267948968000,157.07963267948966000,100.00000000000000000,.F.);
#40=IFCLINESEGMENT2D(#41,0.00000000000000000000,479.00000000000045000);
#43=IFCCIRCULARARCSEGMENT2D(#44,0.00000000000000000000,157.07963267948958000,99.999999999999943000,.F.);

#48=IFCCIRCULARARCSEGMENT2D(#49,4.7123889803846897000,314.15926535897984000,200.00000000000091000,.F.);
#51=IFCLINESEGMENT2D(#52,3.1415926535897931000,2401.0000000000000000);
#54=IFCCIRCULARARCSEGMENT2D(#55,3.1415926535897931000,314.15926535897933000,200.00000000000000000,.T.);

#59=IFCCIRCULARARCSEGMENT2D(#60,4.7123889803846897000,157.07963267948966000,100.00000000000000000,.T.);
#62=IFCLINESEGMENT2D(#63,0.00000000000000000000,240.00000000000045000);
#65=IFCCIRCULARARCSEGMENT2D(#66,0.00000000000000000000,157.07963267948966000,100.00000000000000000,.F.);
```

HEADROOM FREE SPACE

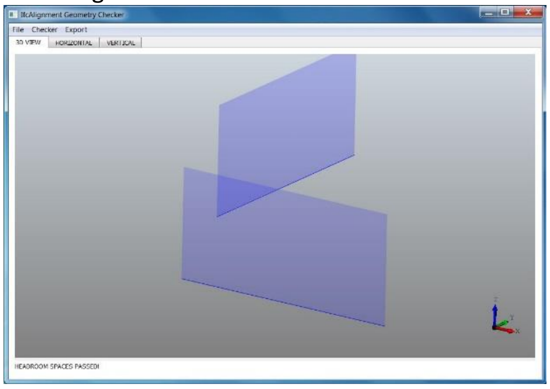
The free space of a headroom is 4.6 meters. For this unit test two line segments are modelled and positioned above each other. The sixth attribute value of the IfcAlignment2DVerSegLine is the StartHeight and the seventh attribute is the StartGradient, which is set to 0 and therefore is a horizontal line.



WELCOME TO THE IFC ALIGNMENT CHECKER

Figure 48: Top view unit test HeadRoom free space

HeadRoom tested to 4.6 meters free space. The HeadRoom passed, respectively to the 0 and 5 meters StartHeight.

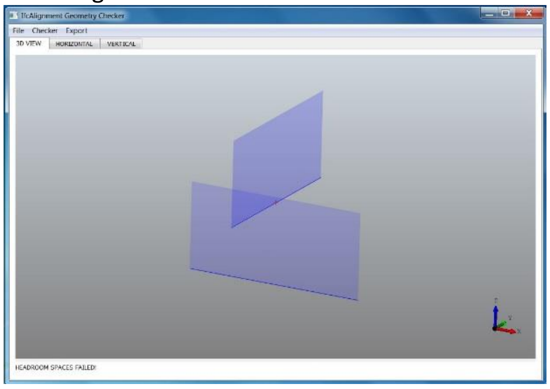


HEADROOM SPACES PASSED!

Figure 49: Passed HeadRoom free space of 5 meters

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,9.9999999999999998000e013,10.000000000001000000,0.00000000000000000000,0.00000000
000000000000);
#36=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,10.000000000000000000,5.0000000000000000000,0.000000000000
00000000);
```

HeadRoom tested to 4.6 meters free space. The HeadRoom failed, respectively to the 0 and 4.5 meters StartHeight.

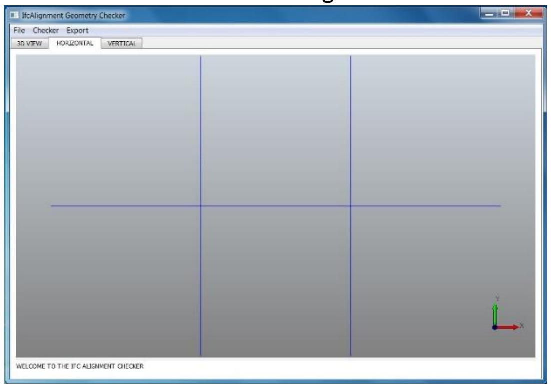


HEADROOM SPACES FAILED!

Figure 50: Failed HeadRoom free space of 4.5 meters

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,9.9999999999999998000e013,10.000000000001000000,0.00000000000000000000,0.00000000
000000000000);
#36=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,10.000000000000000000,4.5000000000000000000,0.000000000000
00000000);
```

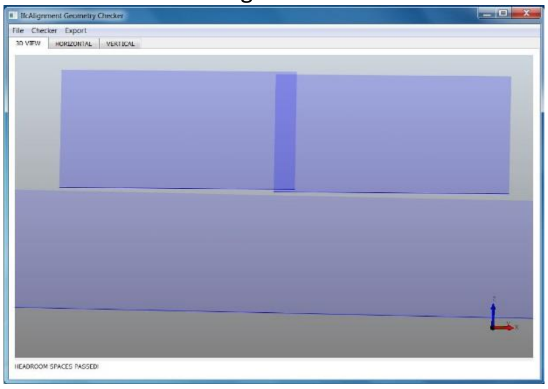
For this unit test three line segments are modelled and positioned above each other.



WELCOME TO THE IFC ALIGNMENT CHECKER

Figure 51: Top view unit test HeadRoom free spaces

HeadRoom tested to 4.6 meters free space. The HeadRoom passed on both crossings, respectively to the 0, 5 and 5 meters StartHeight.

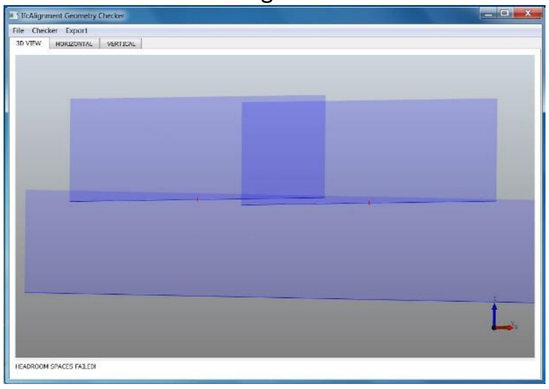


HEADROOM SPACES PASSED!

Figure 52: Passed HeadRoom free spaces of 5 meters

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,30.000000000000000000,0.00000000000000000000,0.00000000000
000000000);
#36=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,20.000000000000000000,5.0000000000000000000,0.000000000000
00000000);
#43=IFCALIGNMENT2DVERSEGLINE($,$,$,0.00000000000000000000,20.000000000000000000,5.0000000000000000000,0.000000000000
00000000);
```

HeadRoom tested to 4.6 meters free space. The HeadRoom failed on both crossings, respectively to the 0, 4.5 and 4.5 meters StartHeight.

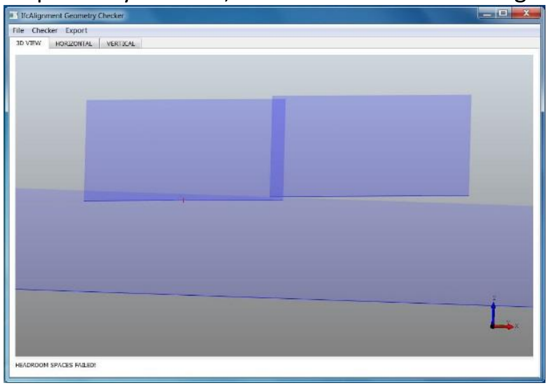


HEADROOM SPACES FAILED!

Figure 53: Failed HeadRoom free spaces of 4.5 meters

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,30.000000000000000000,0.0000000000000000000000,0.00000000000
000000000);
#36=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,20.000000000000000000,4.5000000000000000000,0.000000000000
00000000);
#43=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,20.000000000000000000,4.5000000000000000000,0.000000000000
00000000);
```

HeadRoom tested to 4.6 meters free space. The HeadRoom passed and failed at the first and second crossings, respectively to the 0, 4.5 and 5 meters StartHeight.



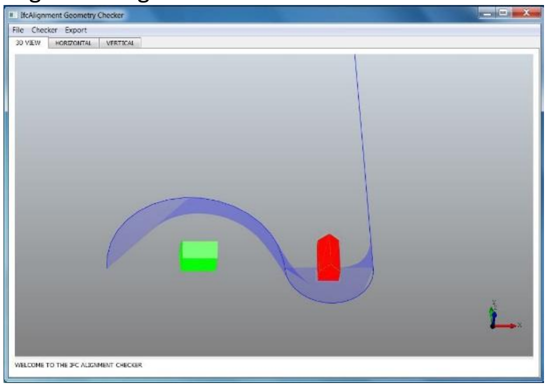
HEADROOM SPACES FAILED!

Figure 54: Failed and passed HeadRoom free spaces of 4.5 and 5 meters

```
#29=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,30.000000000000000000,0.0000000000000000000000,0.00000000000
000000000);
#36=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,20.000000000000000000,4.5000000000000000000,0.000000000000
00000000);
#43=IFCALIGNMENT2DVERSEGLINE($,$,$,0.0000000000000000000000,20.000000000000000000,5.0000000000000000000,0.000000000000
00000000);
```

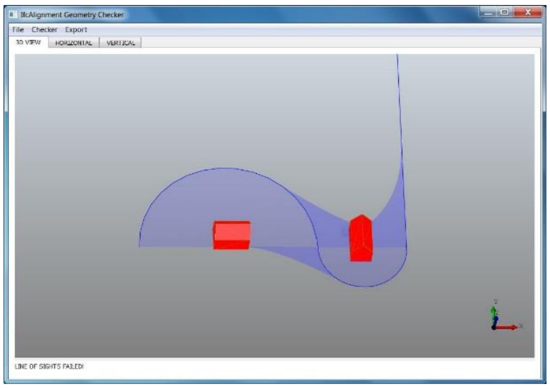
LINE OF SIGHT

The line of sight is 40 meters at 50 km/h and 80 meters at 70 km.h. For this unit test two circular arc segments of 80 meters and 40 meters are modelled. The third attribute value of IfcCircularArcSegment2D is the SegmentLength.



WELCOME TO THE IFC ALIGNMENT CHECKER

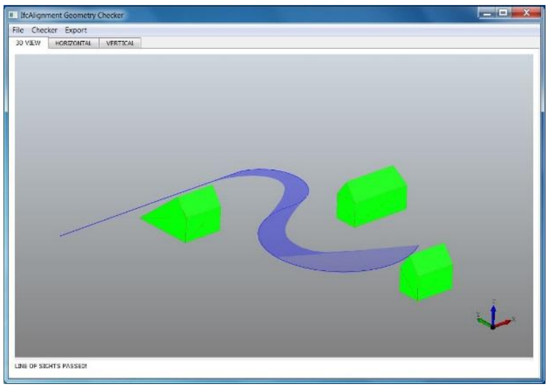
Figure 55: Line of sight of 40 meters



LINE OF SIGHTS FAILED!

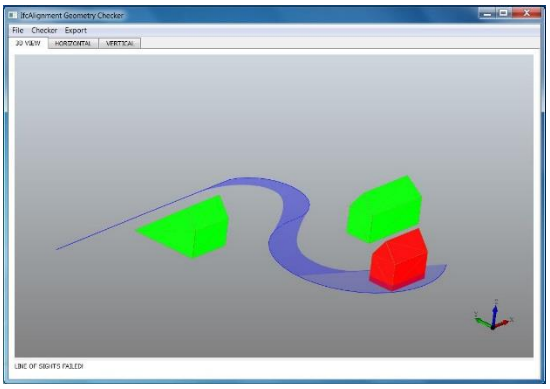
Figure 56: Line of sight of 80 meters

```
#26=IFCCIRCULARARCSEGMENT2D(#27,1.5707963267948968000,80.000028605133352000,25.464799999999997000,.F.);
#29=IFCCIRCULARARCSEGMENT2D(#30,4.7123889803846897000,40.000014302566719000,12.732400000000013000,.T.);
```



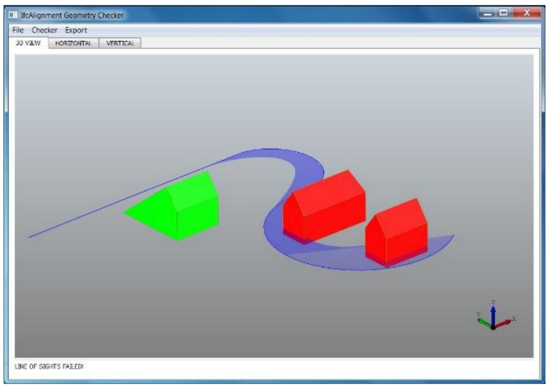
LINE OF SIGHTS PASSED!

Figure 57: Line of sight all passed



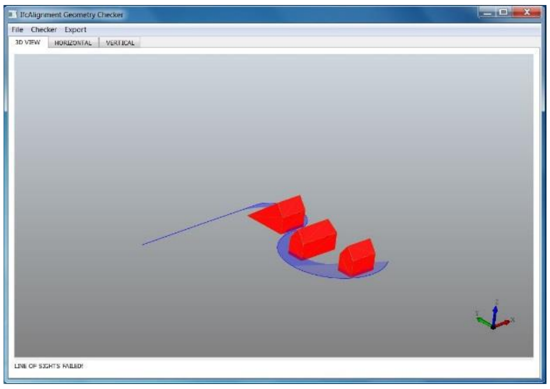
LINE OF SIGHTS FAILED!

Figure 58: Line of sight with one shape failed



LINE OF SIGHTS FAILED!

Figure 59: Line of sight with two shapes failed



LINE OF SIGHTS FAILED!

Figure 60: Line of sight with all shapes failed

APPENDIX C – NEEDED SOFTWARE

The installation procedure to get started with the development of the Automated Geometry Checker is outlined below and expects users to be on Windows.

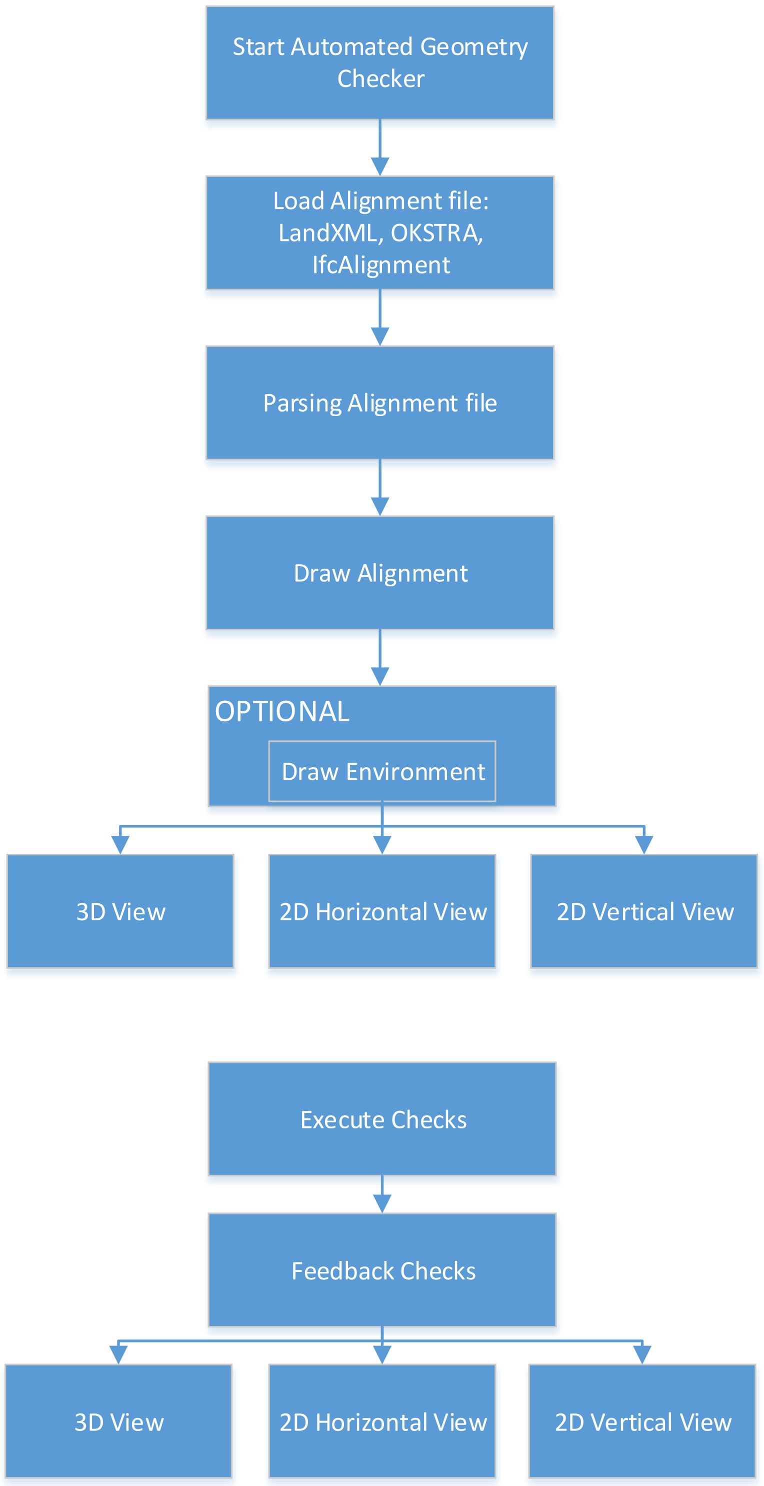
Mandatory:

- Python 2.7
- Qt OpenSource 4.8.6
- pyQt GPL v4.11.3 for Python 2.7 (x32)
- pythonOCC 0.16.0
- IfcOpenShell for Python 2.7 (extract into Lib\site-packages\ directory of Python installation folder)
- TUM Open Infra Platform 1335

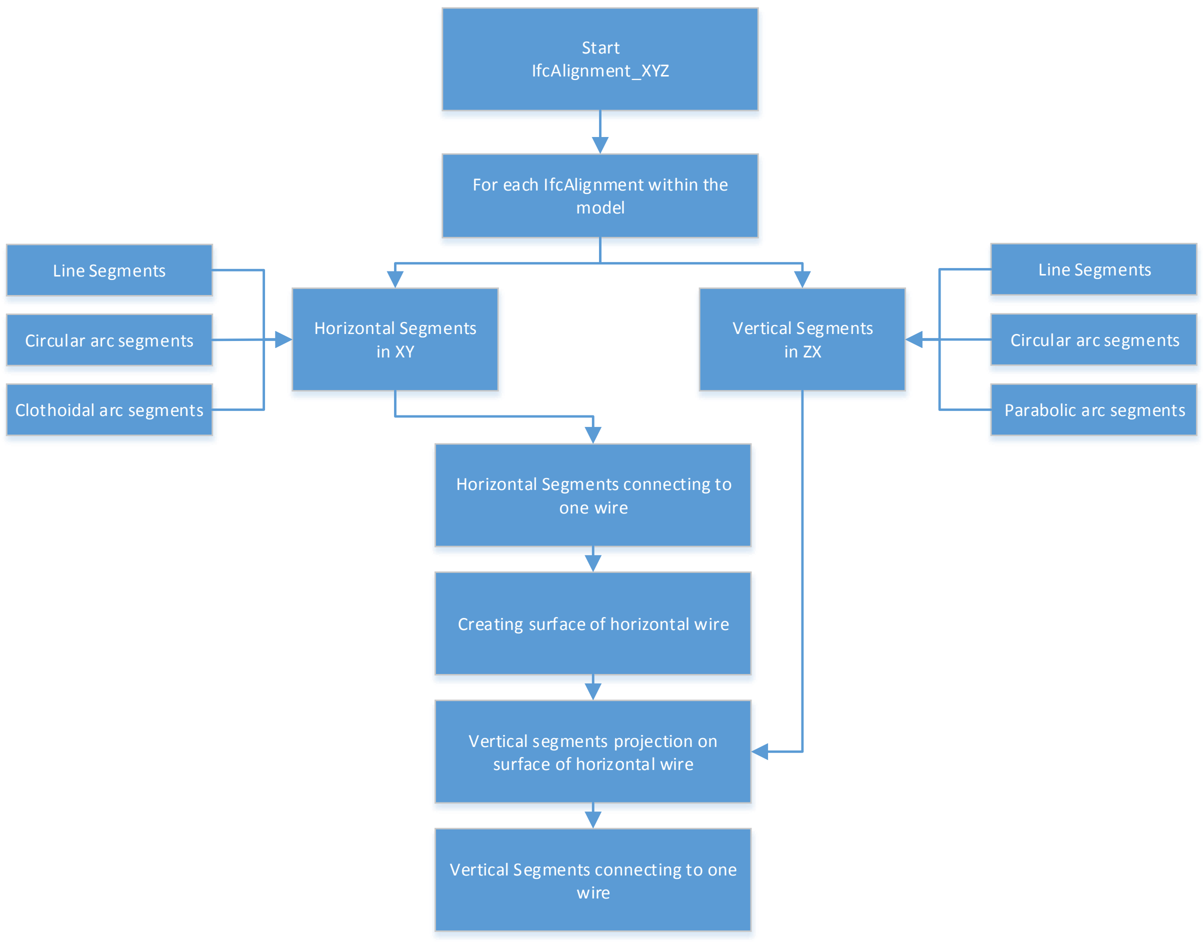
Optional:

- Autodesk Civil3D
- Autodesk Revit
- Bentley Microstation
- Bentley MX Road
- Sketchup
- Notepad++

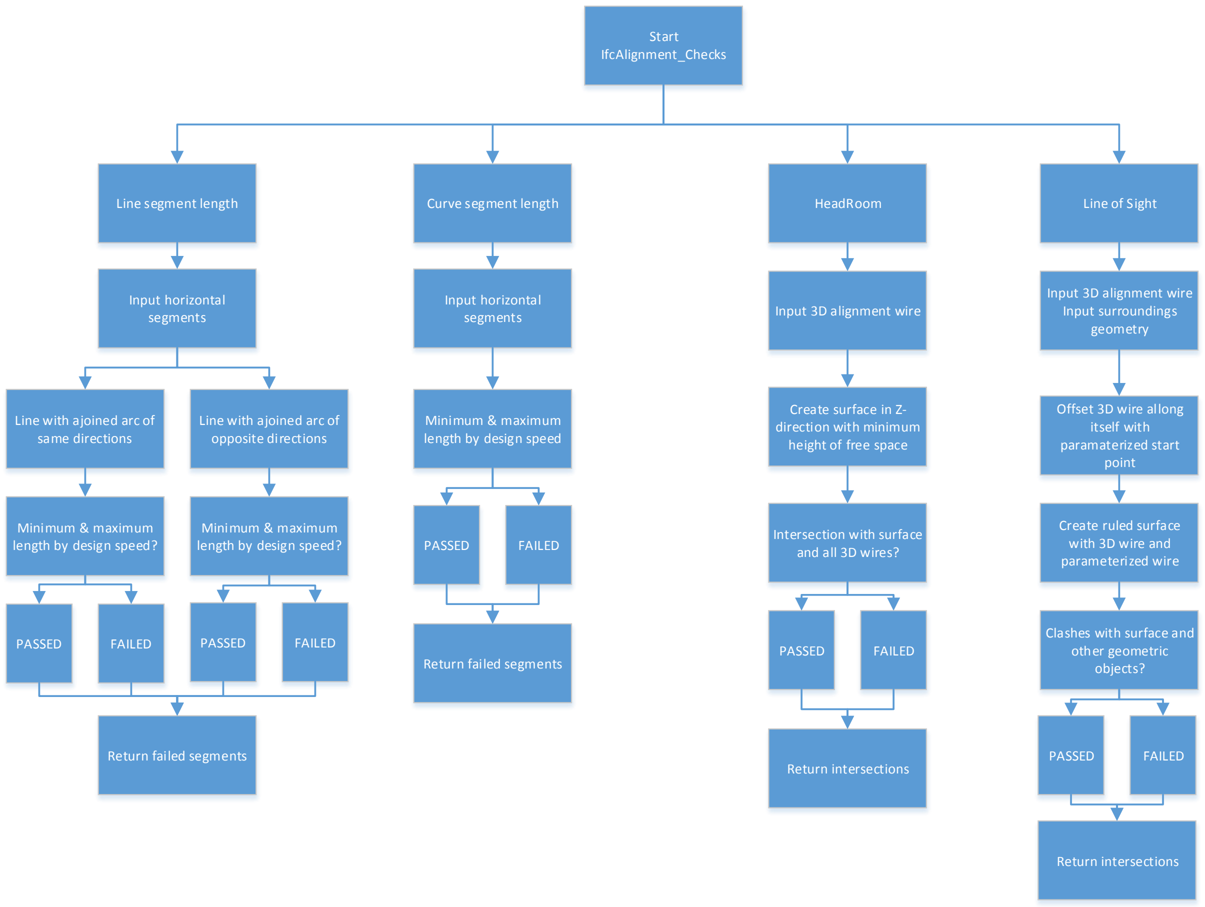
APPENDIX B – OVERVIEW ALIGNMENT CHECKER



APPENDIX C – FLOWCHART PARSING IFCALIGNMENT


Start
IfcAlignment_XYZ
For each IfcAlignment within the model
Line Segments
Circular arc segments
Clothoidal arc segments
Horizontal Segments in XY
Vertical Segments in ZX
Line Segments
Circular arc segments
Parabolic arc segments
Horizontal Segments connecting to one wire
Creating surface of horizontal wire
Vertical segments projection on surface of horizontal wire
Vertical Segments connecting to one wire

APPENDIX D – FLOWCHART CHECK EXECTUTION


Start IfcAlignment_Checks
Line segment length
Curve segment length
HeadRoom
Line of Sight
Input horizontal segments
Input horizontal segments
Input 3D alignment wire
Input 3D alignment wire Input surroundings geometry
Line with ajoined arc of same directions
Line with ajoined arc of opposite directions
Minimum & maximum length by design speed
Create surface in Z-direction with minimum height of free space
Offset 3D wire allong itself with paramaterized start point
Minimum & maximum length by design speed?
Minimum & maximum length by design speed?
PASSED
FAILED
Intersection with surface and all 3D wires?
Create ruled surface with 3D wire and parameterized wire
PASSED
FAILED
PASSED
FAILED
Return failed segments
PASSED
FAILED
Clashes with surface and other geometric objects?
Return failed segments
Return intersections
PASSED
FAILED
Return intersections

APPENDIX E – SCRIPT ALIGNMENT CHECKER

```
#######################################################
##                                                   ##
##               IFCALIGNMENT CHECKER                ##
##                                                   ##
#######################################################


## Import IfcAlignment_XYZ, alignment parser
## Import IfcAlignment_Checks, coded rulesets
## Import System Specific Parameters and Functions
## Import Subrpocess management
## Import IfcOpenShell
## Import Python OpenCascade
## Import PyQt4 display


import IfcAlignment_XYZ
import IfcAlignment_Checks

import sys
import subprocess

import ifcopenshell
import ifcopenshell.geom

import OCC.BRepPrimAPI
import OCC.gp
import OCC.TopoDS
import OCC.TopTools

from PyQt4 import QtGui,QtCore
from OCC.Display.qtDisplay import qtViewer3d
from OCC.Display.pyqt4Display import qtViewer3d
```

```
## SPECIFY TO RETURN PYTHONOCC SHAPES FROM IFCOPENSHELL
settings = ifcopenshell.geom.settings()
settings.set(settings.USE_PYTHON_OPENCASCADE, True)


## MAIN CLASS OF THE GRAPHICAL USER INTERFACE
class Main(QtGui.QMainWindow):

    ## INITIALIZING THE IFCALIGNMENT CHECKER
    def __init__(self, parent = None):
        self.parent = parent
        global filename
        filename = None

        ## Creating 3D, Hor and Ver viewing tabs
        self.tabs = QtGui.QTabWidget()
        self.tab_3dview = QtGui.QWidget()
        self.tab_horizontal = QtGui.QWidget()
        self.tab_vertical = QtGui.QWidget()

        self.tabs.addTab(self.tab_3dview, '3D VIEW')
        self.tabs.addTab(self.tab_horizontal, 'HORIZONTAL')
        self.tabs.addTab(self.tab_vertical, 'VERTICAL')

        ## Implementing the OCC viewer
        super(Main, self).__init__()
        self.canvas = qtViewer3d(self)
        self.horizontalcanvas = qtViewer3d(self)
        # self.horizontalcanvas.setDisabled(True)
        self.verticalcanvas = qtViewer3d(self)
        # self.verticalcanvas.setDisabled(True)

        ## Design of the common, 3D, Hor and Ver widgets
        self.setCentralWidget(self.tabs)
        self.setWindowTitle('IfcAlignment Geometry Checker')
        self.setGeometry(400, 200, 1080, 720)
        ## Implementing the Main (3D), Hor and Ver Viewers
        self.MainViewer()
        self.HorizontalViewer()
        self.VerticalViewer()

        ## Commands to display within the viewers
        self.canvas.InitDriver()
        self.display = self.canvas._display
        self.horizontalcanvas.InitDriver()
        self.Hdisplay = self.horizontalcanvas._display
        self.verticalcanvas.InitDriver()
        self.Vdisplay = self.verticalcanvas._display

        ## Implementing a menu bar
        self.menu_bar()


    ## CREATING THE MAIN (3D) VIEWER
    def MainViewer(self):
        ## Welcome text below the viewer
        self.console = QtGui.QLabel()
        self.console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")

        center = QtGui.QWidget()

        ## Creating boxlayout with textbox and canvas
        mainLayout = QtGui.QHBoxLayout(center)
        mainLayout.setContentsMargins(0, 0, 0, 0)

        splitter = QtGui.QSplitter(QtCore.Qt.Vertical)
        splitter.addWidget(self.canvas)
        splitter.addWidget(self.console)

        splitter.setStretchFactor(0, 14)
        splitter.setStretchFactor(1, 1)

        box_layout = QtGui.QVBoxLayout()
        box_layout.addWidget(splitter)
        self.tab_3dview.setLayout(box_layout)
```

```
## CREATING THE HORIZONTAL (2D) VIEWER
def HorizontalViewer(self):
    ## Welcome text below the viewer
    self.horizontal_console = QtGui.QLabel()
    self.horizontal_console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")

    center = QtGui.QWidget()

    ## Creating boxlayout with textbox and canvas
    mainLayout = QtGui.QHBoxLayout(center)
    mainLayout.setContentsMargins(0, 0, 0, 0)

    splitter = QtGui.QSplitter(QtCore.Qt.Vertical)
    splitter.addWidget(self.horizontalcanvas)
    splitter.addWidget(self.horizontal_console)

    splitter.setStretchFactor (0, 14)
    splitter.setStretchFactor (1, 1)

    box_layout = QtGui.QVBoxLayout()
    box_layout.addWidget(splitter)
    self.tab_horizontal.setLayout(box_layout)


## CREATING THE VERTICAL (2D) VIEWER
def VerticalViewer(self):
    ## Welcome text below the viewer
    self.vertical_console = QtGui.QLabel()
    self.vertical_console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")

    center = QtGui.QWidget()

    ## Creating boxlayout with textbox and canvas
    mainLayout = QtGui.QHBoxLayout(center)
    mainLayout.setContentsMargins(0, 0, 0, 0)

    splitter = QtGui.QSplitter(QtCore.Qt.Vertical)
    splitter.addWidget(self.verticalcanvas)
    splitter.addWidget(self.vertical_console)
```

```
        splitter.setStretchFactor (0, 14)
        splitter.setStretchFactor (1, 1)

        box_layout = QtGui.QVBoxLayout()
        box_layout.addWidget(splitter)
        self.tab_vertical.setLayout(box_layout)


    ## CREATING THE MENU BAR
    def menu_bar(self):
        ## Open Alignment file
        openAlignmentAction = QtGui.QAction(QtGui.QIcon('open.png'), '&Open Alignment', self)
        openAlignmentAction.setShortcut('CTRL+O')
        openAlignmentAction.setStatusTip('Open a .ifc file')
        openAlignmentAction.setStatusTip('Open a .xml file')
        openAlignmentAction.triggered.connect(self.open_alignment_file)

        ## Open environment file
        openEnvironmentAction = QtGui.QAction(QtGui.QIcon('open.png'), '&Open Enviroment', self)
        openEnvironmentAction.setShortcut('CTRL+E')
        openEnvironmentAction.setStatusTip('Open a .ifc file')
        openEnvironmentAction.triggered.connect(self.open_environment_file)

        ## Initialize checker functions
        LinesegmentLengthCheck = QtGui.QAction('Run LineSegment Length', self)
        LinesegmentLengthCheck.triggered.connect(self.Linesegment_Length_Check)

        CurvesegmentLengthCheck = QtGui.QAction('Run CurveSegment Length', self)
        CurvesegmentLengthCheck.triggered.connect(self.Curvesegment_Length_Check)

        HeadRoomCheck = QtGui.QAction('Run HeadRoom', self)
        HeadRoomCheck.triggered.connect(self.HeadRoom_check)

        LineOfSightCheck = QtGui.QAction('Run Line of Sight', self)
        LineOfSightCheck.triggered.connect(self.Line_Of_Sight_Check)

        ## Export to PDF (first setup, in development)
        ExportPDF = QtGui.QAction('Export to PDF', self)
        ExportPDF.setShortcut('CTRL+S')
        ExportPDF.triggered.connect(self.generate_pdf)

        ## Menu bar actions
        menubar = self.menuBar()
        file_menu = menubar.addMenu('File')
        file_menu.addAction(openAlignmentAction)
        file_menu.addAction(openEnvironmentAction)

        checker_menu = menubar.addMenu('Checker')
        checker_menu.addAction(LinesegmentLengthCheck)
        checker_menu.addAction(CurvesegmentLengthCheck)
        checker_menu.addAction(HeadRoomCheck)
        checker_menu.addAction(LineOfSightCheck)

        export_menu = menubar.addMenu('Export')
        export_menu.addAction(ExportPDF)


    ## LOADING ALIGNMENT FILE
    def open_alignment_file(self):
        ## Open alignment file
        self.filename = QtGui.QFileDialog.getOpenFileName(self, 'Open file',".","IFC, LandXML or OKSTRA (*.ifc *.xml
                                                          *.cte)")

        ## Load an LandXML or OKSTRA file, mapping through TUM OIP and export in IfcAlignment
        if self.filename.endsWith(".xml") or self.filename.endsWith(".cte"):
            subprocess.call('bin\OpenInfraPlatform.CommandLineUtilities.exe -i %s -o temp.ifc' % self.filename)
            self.filename = 'temp.ifc'

        ## Erase old file when new file is loaded
        if self.filename:
            self.display.EraseAll()
            self.Hdisplay.EraseAll()
            self.Vdisplay.EraseAll()

            self.console.clear()
            self.horizontal_console.clear()
            self.vertical_console.clear()
        ## Open IfcAlignment file by using IfcOpenShell to enable parsing
        self.IfcAligmentFile = ifcopenshell.open(self.filename)
        self.parse_IfcAlignment()


    ## LOADING ENVIRONMENT FILE
    def open_environment_file(self):
        ## Open file
        self.environment_filename = QtGui.QFileDialog.getOpenFileName(self, 'Open file',".","Industry Foundation
                                                                      Classes (*.ifc)")

        ## This if statement does not work, it should only erase the geometriics of the environmental file
        # if self.environment_filename:
        #     self.parent.display.EraseAll()
        #     self.console.clear()

        ## Open IfcAlignment file by using IfcOpenShell to enable parsing
        self.IfcEnvironmentFile = ifcopenshell.open(self.environment_filename)
        self.parse_IfcEnivironment()


    ## PARSING IFCALIGNMENT FILE
    def parse_IfcAlignment(self):
        ## Set welcome texts
        self.console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")
        self.horizontal_console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")
        self.vertical_console.setText("WELCOME TO THE IFC ALIGNMENT CHECKER")

        ## Creating display trees with color options
        self.horizontal_display_tree = []
        self.horizontal_display_tree_options = []

        self.vertical_display_tree = []
        self.vertical_display_tree_options = []

        self.projection3d_display_tree = []
        self.projection3d_display_tree_options = []
```

```
## Finding entities in dataset
self.IfcAlignments = self.IfcAligmentFile.by_type("IFCALIGNMENT")
self.IfcMapConversion = self.IfcAligmentFile.by_type("IFCMAPCONVERSION")
## Creating emty 3D Alignments shapes container
self.Projections = OCC.TopTools.TopTools_HSequenceOfShape()

## For each IfcAlignment the parsing is done
for IfcAlignment in self.IfcAlignments:
    ## Horizontal Alignment
    self.horizontal_wire, edges = IfcAlignment_XYZ.Alignment().HorizontalWire(IfcAlignment.Horizontal.
                                       Segments, self.IfcMapConversion)
    for i in range(0, edges.Length()):
        self.horizontal_display_tree.append(edges.Value(i+1))
        self.horizontal_display_tree_options.append('BLUE1')
    ## To color each even and odd element a differnt color (ADD ",2" in both ranges)
    # for i in range(1, edges.Length(),2):
    #     self.horizontal_display_tree.append(edges.Value(i+1))
    #     self.horizontal_display_tree_options.append('RED')

    ## Vertical Alignment
    if IfcAlignment.Vertical != None :
        self.wire_vertical_x, vert_edges = IfcAlignment_XYZ.Alignment().VerticalWire(self.horizontal_wire,
                                               IfcAlignment.Vertical.Segments)
        for i in range(0, vert_edges.Length()):
            self.vertical_display_tree.append(vert_edges.Value(i+1))
            self.vertical_display_tree_options.append('BLUE1')
        ## To color each even and odd element a differnt color (ADD ",2" in both ranges)
        # for i in range(1, vert_edges.Length(),2):
        #     self.vertical_display_tree.append(vert_edges.Value(i+1))
        #     self.vertical_display_tree_options.append('RED')

        ## 3D Alignment
        self.Projection, ver_edges = IfcAlignment_XYZ.Alignment().Alignment3D(self.horizontal_wire,
                                   IfcAlignment.Vertical.Segments)
        self.Projections.Append(self.Projection.GetHandle())
        for i in range(0, ver_edges.Length(),2):
            self.projection3d_display_tree.append(ver_edges.Value(i+1))
            self.projection3d_display_tree_options.append('BLUE1'
            ## To color each even and odd element a differnt color (ADD ",2" in both ranges)
            for i in range(1, ver_edges.Length(),2):
                self.projection3d_display_tree.append(ver_edges.Value(i+1))
                self.projection3d_display_tree_options.append('RED')

    ## Initialize drawing in viewers
    self.draw()


## PARSING AND DRAWING ENVIRONMENT FILE
def parse_IfcEnivironment(self):
    ## Finding entities in dataset
    self.products = self.IfcEnvironmentFile.by_type("IFCPRODUCT")
    ## Creating emty geometric shapes container
    self.shapes = OCC.TopTools.TopTools_HSequenceOfShape()

    ## The geometric elements in an IFC file are the IfcProduct elements
    ## For every product a shape is created if the shape has a Representation
    ## The shapes can be displayed in the 3D, 2D Hor and 2D Ver viewers
    for product in self.products:
        if product.Representation:
            self.shape = ifcopenshell.geom.create_shape(settings, product).geometry
            self.display.DisplayShape(self.shape)
            # self.Hdisplay.DisplayShape(self.shape)
            # self.Vdisplay.DisplayShape(self.shape)
            self.shapes.Append(self.shape)

    self.display.FitAll()


## DRAWING ALIGNMENT FILE
def draw(self):
    ## Erasing all when new drawing is initialized
    self.display.EraseAll()
    self.Hdisplay.EraseAll()
    self.Vdisplay.EraseAll()
```

```
        ## Drawing 3D view with edges and colors arrays
        for i in range(0, len(self.projection3d_display_tree)):
            shape = self.projection3d_display_tree[i]
            options = self.projection3d_display_tree_options[i]
            self.display.DisplayShape(shape, color=options)
        self.display.FitAll()
        self.display.Repaint()

        ## Drawing 2D Horizontal view with edges and colors arrays
        for i in range(0, len(self.horizontal_display_tree)):
            shape = self.horizontal_display_tree[i]
            options = self.horizontal_display_tree_options[i]
            self.Hdisplay.DisplayShape(shape, color=options)
            # self.display.DisplayShape(shape, color=options)
        self.Hdisplay.View_Top()
        self.Hdisplay.FitAll()
        self.Hdisplay.Repaint()

        ## Drawing 2D Vertical view with edges and colors arrays
        for i in range(0, len(self.vertical_display_tree)):
            shape = self.vertical_display_tree[i]
            options = self.vertical_display_tree_options[i]
            self.Vdisplay.DisplayShape(shape, color=options)
            ## Displaying a line along the x-axis
            self.Vdisplay.DisplayShape(self.wire_vertical_x)
        self.Vdisplay.View_Front()
        self.Vdisplay.FitAll()
        self.Vdisplay.Repaint()


    ## GENERATING PDF EXPORT (IN DEVELOPMENT)
    def generate_pdf(self):

        # scene = QtGui.QGraphicsScene()
        # scene.addWidget(QtGui.QWidget)
        # view = QtGui.QGraphicsView()
        # view.show()
```

```python
        # scene = QtGui.QGraphicsScene()
        # view = QtGui.QGraphicsView()
        # item = QtGui.QGraphicsTextItem()
        # item.setOpenExternalLinks(True)
        # item.setHtml('<a href="http://stackoverflow.com">Hello</a>')
        # item.setTextInteractionFlags(QtCore.Qt.LinksAccessibleByMouse)
        #
        # scene.addItem(item)
        # view.show()

        filename = QtGui.QFileDialog.getSaveFileName(self, "Export to PDF", "test.pdf", "PDF files (*.pdf)")

        pdf_printer = QtGui.QPrinter(QtGui.QPrinter.HighResolution)
        pdf_printer.setOutputFormat(QtGui.QPrinter.PdfFormat)
        pdf_printer.setOutputFileName(filename)

        pdf_painter = QtGui.QPainter()
        pdf_painter.begin(pdf_printer)
        pdf_painter.background()
        pdf_painter.setViewport( 0,0,100,100)
        # scene.render(pdf_painter)

        pdf_painter.end()

        self.tab_3dview.render(pdf_printer)

        # doc = QtGui.QTextDocument()
        # doc.setPlainText("HOI")

        # doc.print_(printer)
        # printer.newPage()


    ## CLASS 1: UNARY FUNCTION
    def Curvesegment_Length_Check(self):
        ## All segments are passed (GREEN) untill proven otherwise
        for i in range(0, len(self.horizontal_display_tree_options)):
            self.horizontal_display_tree_options[i] = 'GREEN'
```

```
## Offset to ensure display tree counting right elements
offset = 0

## Array of failed shapes
shapes = []

## For each IfcAlignment the check is done
for IfcAlignment in self.IfcAlignments:
    ## Input all horizontal segments of the IfcAlignment
    ## Output array of errors
    errors = IfcAlignment_Checks.Checks().Curvesegment_Length(IfcAlignment.Horizontal.Segments)

    ## If there are no errors, the IfcAlignment passed
    if len(errors) == 0:
        self.console.clear()
        self.console.setText("CURVESEGMENT LENGTHS PASSED!")
        self.horizontal_console.clear()
        self.horizontal_console.setText("CURVESEGMENT LENGTHS PASSED!")

    ## If there are errors, the IfcAlignment failed
    if len(errors) > 0:
        self.console.clear()
        self.console.setText("CURVESEGMENT LENGTHS FAILED!")
        self.horizontal_console.clear()
        self.horizontal_console.setText("CURVESEGMENT LENGTHS FAILED!")

    # print len(errors)


  ## Displaying the errors
    for i in range(0, len(errors)):
        segment = errors[i]
        index = offset + [y.id() for y in IfcAlignment.Horizontal.Segments].index(segment.id())
        shapes.append(self.horizontal_display_tree[index])
        self.horizontal_display_tree_options[index] = 'RED'
```

```
            ## Counting all segments to the offset to ensure next segments are counted correctly
            offset += len(IfcAlignment.Horizontal.Segments)

        ## Initialize drawing
        self.draw()


    ## CLASS 2: MULTIPLE FUNCTION
    def Linesegment_Length_Check(self):
        ## All segments are passed (GREEN) untill proven otherwise
        for i in range(0, len(self.horizontal_display_tree_options)):
            self.horizontal_display_tree_options[i] = 'GREEN'

        ## Offset to ensure display tree counting right elements
        offset = 0

        ## Array of failed shapes
        shapes = []

        ## For each IfcAlignment the check is done
        for IfcAlignment in self.IfcAlignments :
            ## Input all horizontal segments of the IfcAlignment
            ## Output array of errors
            errors = IfcAlignment_Checks.Checks().Linesegment_Length(IfcAlignment.Horizontal.Segments)

            ## If there are no errors, the IfcAlignment passed
            if len(errors) == 0:
                self.console.clear()
                self.console.setText("LINESEGMENT LENGTHS PASSED!")
                self.horizontal_console.clear()
                self.horizontal_console.setText("LINESEGMENT LENGTHS PASSED!")

            ## If there are errors, the IfcAlignment failed
            if len(errors) > 0:
                self.console.clear()
                self.console.setText("LINESEGMENT LENGTHS FAILED!")
                self.horizontal_console.clear()
                self.horizontal_console.setText("LINESEGMENT LENGTHS FAILED!")
            # print len(errors)
        ## Displaying the errors
        for i in range(0, len(errors)):
            segment = errors[i]
            index = offset + [y.id() for y in IfcAlignment.Horizontal.Segments].index(segment.id())

            shapes.append(self.horizontal_display_tree[index])
            self.horizontal_display_tree_options[index] = 'RED'

        ## Counting all segments to the offset to ensure next segments are counted correctly
        offset += len(IfcAlignment.Horizontal.Segments)

    ## Initialize drawing
    self.draw()


## CLASS 3: CALCULATIONS AND INFERENCES
def HeadRoom_check(self):
    ## Create an emty geometric shapes container for surfaces in z-direction
    headroom_spaces = OCC.TopTools.TopTools_HSequenceOfShape()

    ## For each 3D parsed alignment the check is done
    for i in range(0, self.Projections.Length()):
        ## Very small offset of the 3D alignment, otherwise it will have intersections along itself
        offset = OCC.BRepPrimAPI.BRepPrimAPI_MakePrism(OCC.TopoDS.topods_Wire(self.Projections.Value(i + 1)),
                                                      OCC.gp.gp_Vec(0,0,0.0001), True)
        ## Creating surface from extrusion of the 3D alignment in the z-direction
        space = OCC.BRepPrimAPI.BRepPrimAPI_MakePrism(OCC.TopoDS.topods_Wire(offset.LastShape()),
                                                     OCC.gp.gp_Vec(0,0,4.8), True).Shape()

        ## Adding the surface to the container of geometric shapes
        headroom_spaces.Append(space)
        ## Display the surface in the 3D view
        self.display.DisplayShape(space, transparency=0.8, color='BLUE1')

    ## Array of intersection points
    ## Input all parsed 3D alignments
    ## Output array of errors
    pnts = IfcAlignment_Checks.Checks().HeadRoom(self.Projections, headroom_spaces)
```

```
        ## If there are no points, the IfcAlignment passed
        if len(pnts) == 0:
            self.console.clear()
            self.console.setText("HEADROOM SPACES PASSED!")
            self.horizontal_console.clear()
            self.horizontal_console.setText("HEADROOM SPACES PASSED!")

        ## If there are errors, the IfcAlignment failed
        if len(pnts) > 0:
            self.console.clear()
            self.console.setText("HEADROOM SPACES FAILED!")
            self.horizontal_console.clear()
            self.horizontal_console.setText("HEADROOM SPACES FAILED!")

        ## Display the errors in the 3D view
        for i in range(0, len(pnts)):
            self.display.DisplayShape(pnts[i], color='RED')
            self.display.FitAll()


    ## CLASS 4: CALCULATIONS AND INFERENCES FROM EXTERNAL GEOMETRY
    def Line_Of_Sight_Check(self):
        ## Create the surfaces to test on intersections
        ## Input all parsed 3D alignments
        ## Output an container with geometric surfaces of the 3D alignments and its parameterized 3D alignments
        ruled_surfaces = IfcAlignment_Checks.Checks().Line_of_Sight_surface(self.Projections)

        ## All geometric shapes are passed (GREEN) untill proven otherwise
        for i in range(0, self.shapes.Length()):
            self.display.DisplayShape(self.shapes.Value(i+1), color='GREEN')

        ## Display the geometric surfaces
        for i in range(0, ruled_surfaces.Length()):
            self.display.DisplayShape(ruled_surfaces.Value(i+1), transparency=0.8, color='BLUE1')
            self.display.FitAll()
            # print "DISPLAYED SURFACE"
            ## Failed shapes (intersect with surfaces)
            ## Input geometric surfaces and geometric shapes
            ## Output an array of failed shapes
            shapes_fail = IfcAlignment_Checks.Checks().Line_of_Sight_shapes(ruled_surfaces.Value(i+1), self.shapes)

            # print len(shapes_fail)

            ## If there are no failed shapes, the line of sight passed
            if len(shapes_fail) == 0:
                self.console.clear()
                self.console.setText("LINE OF SIGHTS PASSED!")
                self.horizontal_console.clear()
                self.horizontal_console.setText("LINE OF SIGHTS PASSED!")
                self.display.FitAll()

            ## If there are errors, the line of sight failed
            if len(shapes_fail) > 0:
                self.console.clear()
                self.console.setText("LINE OF SIGHTS FAILED!")
                self.horizontal_console.clear()
                self.horizontal_console.setText("LINE OF SIGHTS FAILED!")

            ## Display the failed geometric shapes in the 3D view
            for i in range(0, len(shapes_fail)):
                self.display.DisplayShape(shapes_fail[i], color='RED')
                self.display.FitAll()


## CALLING THE IFCALIGNMENT CHECKER APPLICATION
if __name__ == "__main__":
    app = QtGui.QApplication(sys.argv)
    main = Main()
    main.show()
    sys.exit(app.exec_())
```

```
#######################################################
##                                                   ##
##         IFCALIGNMENT CHECKER - XYZ PARSER         ##
##                                                   ##
#######################################################


## Import Mathematical functions
## Import IfcOpenShell
## Import Python OpenCascade functions

import math

import ifcopenshell
import ifcopenshell.geom

import OCC.Adaptor2d
import OCC.Adaptor3d
import OCC.BRep
import OCC.BRepAdaptor
import OCC.BRepBuilderAPI
import OCC.BRepLib
import OCC.gce
import OCC.GCE2d
import OCC.Geom
import OCC.Geom2d
import OCC.Geom2dAdaptor
import OCC.GeomAdaptor
import OCC.GeomLib
import OCC.gp
import OCC.ProjLib
import OCC.ShapeAnalysis
import OCC.ShapeExtend
import OCC.ShapeFix
import OCC.ShapeUpgrade
import OCC.StdPrs
import OCC.TopAbs
```

```python
import OCC.TopExp
import OCC.TopoDS
import OCC.TopTools


## MAIN CLASS OF THE IFCALIGNMENT XYZ PARSER
class Alignment():

    ## INITIALIZING THE IFCALIGNMENT CHECKS
    def __init__(self,*args):
        self.object = 0

    ## 2D HORIZONTAL ALIGNMENT
    def HorizontalWire(self, horizontal_segments, mapconversion):
        ## The segment builders are functions which are called when the specific segment has to be parsed
        segment_builders = {
            'IfcLineSegment2D': process_LineSegment,
            'IfcCircularArcSegment2D': process_CircularArcSegment,
            'IfcClothoidalArcSegment2D': process_ClothoidalArcSegment}

        ## The mapconversion positions the right situation of the segments by Nothings and Eastings
        for conversion_value in mapconversion:
            conversion_x = conversion_value.Eastings
            conversion_y = conversion_value.Northings

        ## Creating emty Horizontal Alignment shapes containers for edges and wires
        horizontal_edges = OCC.TopTools.TopTools_HSequenceOfShape()
        horizontal_wire = OCC.TopTools.TopTools_HSequenceOfShape()
        wire_handle = OCC.TopTools.Handle_TopTools_HSequenceOfShape()

        ## For each segment in the Horizontal Alignment some general attributes are appointed
        for i, segment in enumerate(horizontal_segments):
            ## Startpoint with the movement of the Nothing and Easting values
            self.start_point_x = segment.CurveGeometry.StartPoint.Coordinates[0] + conversion_x
            self.start_point_y = segment.CurveGeometry.StartPoint.Coordinates[1] + conversion_y
            self.start_point = OCC.gp.gp_Pnt(self.start_point_x, self.start_point_y, 0)
    ## If the segment is not the last segment of the Horizonal Alignment,
    ## the startpoint of the next segment is the endpoint of the previous segment
    ## This is done to create continuity in the Horizontal Alignment
    if i < len(horizontal_segments) - 1:
        ## Endpoint with the movement of the Nothing and Easting values
        self.end_point_x = horizontal_segments[i+1].CurveGeometry.StartPoint.Coordinates[0] + conversion_x
        self.end_point_y = horizontal_segments[i+1].CurveGeometry.StartPoint.Coordinates[1] + conversion_y
        self.end_point  = OCC.gp.gp_Pnt(self.end_point_x, self.end_point_y, 0)

        ## The nextsegment is used as a variable input in the parsing of the segments
        Nextsegment = horizontal_segments[i+1]
    else:
        Nextsegment = None

    ## The builder calls the right segment builder in the sequential list of segments presented in the
       Horizontal Alignment
    ## The function is constructed by its segment and the nextsegment
    horizontal_builder = segment_builders[segment.CurveGeometry.is_a()]
    horizontal_builder(self, segment.CurveGeometry, Nextsegment)

    ## Each function in the segment builder returns an edge
    ## These edges are sequentially stored in the horizontal edges shape container
    horizontal_edges.Append(horizontal_builder(self, segment.CurveGeometry, Nextsegment).Edge())

## The edges are connected to a wire
## This way one Horizontal Alignment is one wire
OCC.ShapeAnalysis.ShapeAnalysis_FreeBounds.ConnectEdgesToWires(horizontal_edges.GetHandle(), 1e-2, False,
                                                                     wire_handle)
horizontal_wire = wire_handle.GetObject()

## The function returns the Horizontal wire and Horizontal edges
return horizontal_wire, horizontal_edges
```

```python
## 2D VERTICAL ALIGNMENT
def VerticalWire(self, horizontal_wire, vertical_segments):
    ## The segment builders are functions which are called when the specific segment has to be parsed
    segment_builders = {
            'IfcAlignment2DVerSegLine': process_VerSegLine,
            'IfcAlignment2DVerSegParabolicArc': process_VerSegParabolicArc,
            'IfcAlignment2DVerSegCircularArc': process_VerSegCircularArc
            }

    ## Create a Vertical wire along the x-axis
    ## Compute the Horizontal wire in a Adaptor Horizontal wire to get the start and end U-parameter
    ## Create a line along the x-axis
    ## Create a edge and wire from the line, with start and end parameters from the Adaptor Horizontal wire
    adaptor_horizontal_wire = OCC.BRepAdaptor.BRepAdaptor_CompCurve(OCC.TopoDS.topods_Wire(horizontal_wire.
                                                                    Value(1)), True)
    ax_vertical_x = OCC.Geom.Geom_Line(OCC.gp.gp_Pnt(0,0,0), OCC.gp.gp_Dir(1,0,0))
    edge_vertical_x = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(ax_vertical_x.GetHandle(),
                                                              adaptor_horizontal_wire.FirstParameter(),

    adaptor_horizontal_wire.LastParameter()).Edge()
    wire_vertical_x = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeWire(edge_vertical_x).Wire()

    ## Create a surface from the wire along the x-axis in the z-direction
    ## All vertical segments are projected onto this surface
    Comp_Curve = OCC.BRepAdaptor.BRepAdaptor_CompCurve(OCC.TopoDS.topods_Wire(wire_vertical_x), True)
    HComp_Curve = OCC.BRepAdaptor.BRepAdaptor_HCompCurve(Comp_Curve)
    self.surface_wires = OCC.Adaptor3d.Adaptor3d_SurfaceOfLinearExtrusion(HComp_Curve.GetHandle(),
                                                                         OCC.gp.gp_Dir(0,0,1))

    ## Creating emty Vertical Alignment shapes containers for edges and wires
    vertical_edges = OCC.TopTools.TopTools_HSequenceOfShape()
    vertical_wire = OCC.TopTools.TopTools_HSequenceOfShape()
    wires_handle = OCC.TopTools.Handle_TopTools_HSequenceOfShape()
```

```
## For each segment in the Vertical Alignment some general attributes are appointed
for i, segment in enumerate(vertical_segments):
    if i < len(vertical_segments) - 1:
        ## Conversion of all segments Start Distance Along the Horizontal Alignements, so that the first
           segment starts at 0
        ## Therefore also the StartDistAlong of the next segment is converted
        self.StartDistAlong = vertical_segments[i].StartDistAlong - vertical_segments[0].StartDistAlong
        self.NextStartDistAlong = vertical_segments[i+1].StartDistAlong -
                                                                      vertical_segments[0].StartDistAlong

        ## The nextsegment is used as a variable input in the parsing of the segments
        Nextsegment = vertical_segments[i+1]
    else:
        self.StartDistAlong = vertical_segments[i].StartDistAlong - vertical_segments[0].StartDistAlong
        self.NextStartDistAlong = None
        Nextsegment = None

    ## The vertical builder calls the right segment builder in the sequental list of segments presented in
       the Vertical Alignment
    ## The function is constructed by its segment and the nextsegment
    vertical_builder = segment_builders[segment.is_a()]
    vertical_builder(self, segment, Nextsegment)

    ## Each function in the segment builder returns an edge
    ## These edges are sequentially stored in the vertical edges shape container
    vertical_edges.Append( vertical_builder(self, segment, Nextsegment).Edge())

## The edges are connected to a wire
## This way one Vertical Alignment is one wire
OCC.ShapeAnalysis.ShapeAnalysis_FreeBounds.ConnectEdgesToWires(vertical_edges.GetHandle(), 1e-1, False,
                                                                      wires_handle)
vertical_wire = wires_handle.GetObject()

## The function returns the Horizontal wire and Horizontal edges
return wire_vertical_x, vertical_edges
```

```
## 3D ALIGNMENT
def Alignment3D(self, horizontal_wire, vertical_segments):
    ## The segment builders are functions which are called when the specific segment has to be parsed
    segment_builders = {
            'IfcAlignment2DVerSegLine': process_VerSegLine,
            'IfcAlignment2DVerSegParabolicArc': process_VerSegParabolicArc,
            'IfcAlignment2DVerSegCircularArc': process_VerSegCircularArc}

    ## Create a surface from the horizontal wire in the z-direction
    ## All vertical segments are projected onto this surface
    Comp_Curve = OCC.BRepAdaptor.BRepAdaptor_CompCurve(OCC.TopoDS.topods_Wire(horizontal_wire.Value(1)), True)
    HComp_Curve = OCC.BRepAdaptor.BRepAdaptor_HCompCurve(Comp_Curve)
    self.surface_wires = OCC.Adaptor3d.Adaptor3d_SurfaceOfLinearExtrusion(HComp_Curve.GetHandle(),
                                                                             OCC.gp.gp_Dir(0,0,1))

    ## Creating emty Alignment3D shapes containers for edges and wires
    alignment3D_edges = OCC.TopTools.TopTools_HSequenceOfShape()
    alignment3D_wire = OCC.TopTools.TopTools_HSequenceOfShape()
    wires_handle = OCC.TopTools.Handle_TopTools_HSequenceOfShape()

    ## For each segment in the Alignment3D some general attributes are appointed
    for i, segment in enumerate(vertical_segments):

        if i < len(vertical_segments) - 1:
            ## Conversion of all segments Start Distance Along the Horizontal Alignements, so that the first
               segment starts at 0
            ## Therefore also the StartDistAlong of the next segment is converted
            self.StartDistAlong = vertical_segments[i].StartDistAlong - vertical_segments[0].StartDistAlong
            self.NextStartDistAlong = vertical_segments[i+1].StartDistAlong -
                                                  vertical_segments[0].StartDistAlong

            ## The nextsegment is used as a variable input in the parsing of the segments
            Nextsegment = vertical_segments[i+1]
        else:
            self.StartDistAlong = vertical_segments[i].StartDistAlong - vertical_segments[0].StartDistAlong
            self.NextStartDistAlong = None
            Nextsegment = None
            ## The alignment3D builder calls the right segment builder in the sequental list of segments presented
               in the Vertical Alignment
            ## The function is constructed by its segment and the nextsegment
            alignment3D_builder = segment_builders[segment.is_a()]
            alignment3D_builder(self, segment, Nextsegment)
            ## Each function in the segment builder returns an edge
            ## These edges are sequentially stored in the vertical edges shape container
            alignment3D_edges.Append(alignment3D_builder(self, segment, Nextsegment).Edge())

        ## The edges are connected to a wire
        ## This way one Alignment3D is one wire
        OCC.ShapeAnalysis.ShapeAnalysis_FreeBounds.ConnectEdgesToWires(alignment3D_edges.GetHandle(), 1e-1, False,
                                                                                     wires_handle)
        alignment3D_wire = wires_handle.GetObject()

        ## The function returns the Alignment3D wire and Alignment3D edges
        return alignment3D_wire, alignment3D_edges


## PARSING HORIZONTAL LINE SEGMENTS
def process_LineSegment( self, IfcLineSegment2D, NextSegment ):
    ## Converting the StartDirection in a dx, dy and direction of a linesegment
    dx = math.cos(IfcLineSegment2D.StartDirection)
    dy = math.sin(IfcLineSegment2D.StartDirection)

    dir = OCC.gp.gp_Dir(dx, dy, 0)

    ## If the segment has a next segment the line will be parsed by absolute stationing from startpoint to endpoint
    ## This is done to ensure continuity and none interuptions in the segments
    if NextSegment != None:
        line = OCC.gce.gce_MakeLin(self.start_point, self.end_point).Value()
        edge_linesegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(line, self.start_point, self.end_point)
    ## If the segment has not a next segment, the line will be parsed by relative stationing from startpoint,
       direction and segment length
    else:
        line = OCC.Geom.Geom_Line(self.start_point, dir)
        edge_linesegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(line.GetHandle(), 0,
                                                                       IfcLineSegment2D.SegmentLength

    ## The function returns the edge of a line segment
    return edge_linesegment


## PARSING HORIZONTAL CIRCULAR ARC SEGMENTS
def process_CircularArcSegment(self, IfcCirculararcSegment2D, NextSegment):
    ## Converting the StartDirection in a dx, dy and direction of a linesegment
    dx = math.cos(IfcCirculararcSegment2D.StartDirection)
    dy = math.sin(IfcCirculararcSegment2D.StartDirection)

    dir = OCC.gp.gp_Dir(0, 0, 1)

    ## If the segment has a next segment the circular arc will be parsed by absolute stationing from three points,
       startpoint, endpoint, and another
    ## This is done to ensure continuity and none interuptions in the segments
    if NextSegment != None:
        if IfcCirculararcSegment2D.IsCcw == True:
            ## When the circular arc is counter clockwise, the middle point is on the left from its startpoint, so +
               pi / 2
            ## The middle point is needed to calculate a third point of the circular arc
            middle_pnt_x = self.start_point_x + (math.cos(IfcCirculararcSegment2D.StartDirection + (math.pi / 2)) *
                                IfcCirculararcSegment2D.Radius)
            middle_pnt_y = self.start_point_y + (math.sin(IfcCirculararcSegment2D.StartDirection + (math.pi / 2)) *
                                IfcCirculararcSegment2D.Radius)
            middle_pnt = OCC.gp.gp_Pnt(middle_pnt_x, middle_pnt_y, 0)
    point_3_x = middle_pnt_x - (math.cos(IfcCirculararcSegment2D.StartDirection) *
                    IfcCirculararcSegment2D.Radius)
    point_3_y = middle_pnt_y - (math.sin(IfcCirculararcSegment2D.StartDirection) *
                    IfcCirculararcSegment2D.Radius)
    point_3 = OCC.gp.gp_Pnt(point_3_x, point_3_y, 0)

    circulararcsegment = OCC.gce.gce_MakeCirc(self.start_point, self.end_point , point_3).Value()

    edge_circulararcsegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(circulararcsegment,
                    self.start_point, self.end_point)

if IfcCirculararcSegment2D.IsCcw == False :
    ## When the circular arc is not counter clockwise, the middle point is on the right from its startpoint,
       so - pi / 2
    ## The middle point is needed to calculate a third point of the circular arc
    middle_pnt_x = self.start_point_x + (math.cos(IfcCirculararcSegment2D.StartDirection - (math.pi / 2)) *
                    IfcCirculararcSegment2D.Radius)
    middle_pnt_y = self.start_point_y + (math.sin(IfcCirculararcSegment2D.StartDirection - (math.pi / 2)) *
                    IfcCirculararcSegment2D.Radius)
    middle_pnt = OCC.gp.gp_Pnt(middle_pnt_x, middle_pnt_y, 0)

    point_3_x = middle_pnt_x - (math.cos( IfcCirculararcSegment2D.StartDirection ) *
                    IfcCirculararcSegment2D.Radius)
    point_3_y = middle_pnt_y - (math.sin( IfcCirculararcSegment2D.StartDirection ) *
                    IfcCirculararcSegment2D.Radius)
    point_3 = OCC.gp.gp_Pnt(point_3_x, point_3_y, 0)

    circulararcsegment = OCC.gce.gce_MakeCirc(self.start_point, self.end_point , point_3).Value()

    edge_circulararcsegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(circulararcsegment,
                    self.start_point, self.end_point)
```

```
## If the segment has not a next segment, the circular arc will be parsed by relative stationing from
    startpoint, direction, radius and segment length
else :
    if IfcCircularArcSegment2D.IsCcw == True:
        ## When the circular arc is counter clockwise, the middle point is on the left from its startpoint, so +
           pi / 2
        middle_pnt_x = self.start_point_x + (math.cos(IfcCircularArcSegment2D.StartDirection + (math.pi / 2)) *
                            IfcCircularArcSegment2D.Radius)
        middle_pnt_y = self.start_point_y + (math.sin(IfcCircularArcSegment2D.StartDirection + (math.pi / 2)) *
                            IfcCircularArcSegment2D.Radius)
        middle_pnt = OCC.gp.gp_Pnt(middle_pnt_x, middle_pnt_y, 0)

        start_dir_arc = IfcCircularArcSegment2D.StartDirection - (math.pi / 2)
        end_dir_arc = start_dir_arc + (IfcCircularArcSegment2D.SegmentLength / IfcCircularArcSegment2D.Radius)

        circulararcsegment = OCC.gce.gce_MakeCirc(OCC.gp.gp_Ax2(middle_pnt, dir),
                            IfcCircularArcSegment2D.Radius).Value()
        edge_circulararcsegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(circulararcsegment, start_dir_arc,
                            end_dir_arc)

    if IfcCircularArcSegment2D.IsCcw == False:
        ## When the circular arc is not counter clockwise, the middle point is on the right from its startpoint,
           so - pi / 2
        middle_pnt_x = self.start_point_x + (math.cos(IfcCircularArcSegment2D.StartDirection - (math.pi / 2)) *
                            IfcCircularArcSegment2D.Radius)
        middle_pnt_y = self.start_point_y + (math.sin(IfcCircularArcSegment2D.StartDirection - (math.pi / 2)) *
                            IfcCircularArcSegment2D.Radius)
        middle_pnt = OCC.gp.gp_Pnt(middle_pnt_x, middle_pnt_y, 0)

        start_dir_arc = IfcCircularArcSegment2D.StartDirection + (math.pi / 2)
        end_dir_arc = start_dir_arc - (IfcCircularArcSegment2D.SegmentLength / IfcCircularArcSegment2D.Radius)
```

```
            circulararcsegment = OCC.gce.gce_MakeCirc(OCC.gp.gp_Ax2( middle_pnt, dir),
                                 IfcCirculararcSegment2D.Radius).Value()
            edge_circulararcsegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(circulararcsegment, end_dir_arc,
                                 start_dir_arc)

        ## The function returns the edge of a line segment
        return edge_circulararcsegment


    ## PARSING HORIZONTAL CLOTHOIDAL ARC SEGMENTS
    def process_ClothoidalArcSegment(self, IfcClothoidalArcSegment2D, NextSegment):
        ## The clothoid is geometrically difficult curve and out of scope for this research
        ## Due to the data sets with clothoidal arc segments, it has to be parced a certain way to create a continued
           horizontal wire
        ## Therefore a line is parsed by absolute stationing, because a clothoid is always a segment between segments
        ## It does not jeopaardize the basics of this research, but it is adapted in the future researches
            if NextSegment != None :
                clothoid = OCC.gce.gce_MakeLin(self.start_point, self.end_point).Value()

                edge_clothoidsegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(clothoid, self.start_point,
                                                                                    self.end_point)

            ## The function returns the edge of a clothoidal segment as a line segment
            return edge_clothoidsegment


    ## PARSING VERTICAL LINE SEGEMNTS
    def process_VerSegLine(self, IfcVerSegLine, NextSegment):
        ## Converting the StartGradient in a dx, dy and direction of a vertical line segment
        ## The gradient of the tangent of the vertical segment at the start point. It is provided as a ratio measure.
        ## The ratio is percentage/100 (0.1 is equal to 10%). It has a theoretical range of # - infinite to infinite
           using a ratio measure.
        ## The equivalent range measured in degree is -90 degree to 90 degree.
        dx = math.cos(math.tanh(IfcVerSegLine.StartGradient))
        dy = math.sin(math.tanh(IfcVerSegLine.StartGradient))
        direction = OCC.gp.gp_Dir2d(dx, dy)
```

```
## Calculating the vertical length
VerticalLength = (IfcVerSegLine.HorizontalLength * dy) / dx

## If the segment has a next segment the line segment will be parsed by absolute stationing from its startpoint
   and endpoint
if self.NextStartDistAlong != None:
    ## The startpoint is the 'converted till 0 StartDistAlong' and the z-positioning on the surface
    ## The endpoint is from the next segment
    end_point = self.surface_wires.Value(self.NextStartDistAlong, NextSegment.StartHeight)
    line_on_surface = OCC.GCE2d.GCE2d_MakeLine(OCC.gp.gp_Pnt2d(self.StartDistAlong, IfcVerSegLine.StartHeight),
                           OCC.gp.gp_Pnt2d(self.NextStartDistAlong, NextSegment.StartHeight)).Value()

## If the segment has not a next segment the line segment will be parsed by relative stationing from its
   startpoint and calculated endpoint
else:
    ## The startpoint is the 'converted till 0 StartDistAlong' and the z-positioning on the surface
    ## The endpoint is calculated from startpoint adding the HorizontalLength and the VerticalLength
    end_point = self.surface_wires.Value(self.StartDistAlong + IfcVerSegLine.HorizontalLength,
                   IfcVerSegLine.StartHeight + VerticalLength)
    line_on_surface = OCC.GCE2d.GCE2d_MakeLine(OCC.gp.gp_Pnt2d( self.StartDistAlong, IfcVerSegLine.StartHeight),
                   OCC.gp.gp_Pnt2d(self.StartDistAlong + IfcVerSegLine.HorizontalLength,
                   IfcVerSegLine.StartHeight + VerticalLength)).Value()

## Project the Adaptor line on the linear surface of the Horizontal Wire
adaptor_curve = OCC.Adaptor2d.Handle_Adaptor2d_HCurve2d(OCC.Geom2dAdaptor.Geom2dAdaptor_HCurve(line_on_surface))
adaptor_surface = OCC.Adaptor3d.Handle_Adaptor3d_HSurface(OCC.Adaptor3d.Adaptor3d_HSurfaceOfLinearExtrusion(
                                                               self.surface_wires))
curve_on_surface = OCC.Adaptor3d.Adaptor3d_CurveOnSurface(adaptor_curve, adaptor_surface)

## Create a geometric line
VerSegLineSegment = OCC.Geom.Handle_Geom_Curve()

## Build the geometric line from the Adaptor line with start and end parameters
Length = math.sqrt((VerticalLength**2) + (IfcVerSegLine.HorizontalLength**2))
a, b = OCC.GeomLib.geomlib_BuildCurve3d(1e-9, curve_on_surface, 0, Length, VerSegLineSegment)
```

```
    ## Create an edge from the geometric line
    Edge_VerSegLineSegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(VerSegLineSegment)

    ## The function returns the edge of a line segment
    return Edge_VerSegLineSegment


## PARSING VERTICAL PARABOLIC ARC SEGMENTS
def process_VerSegParabolicArc(self, IfcVersegParabolicArc, NextSegment):
    ## Converting the StartGradient in a dx, dy and direction of a vertical line segment
    ## The gradient of the tangent of the vertical segment at the start point. It is provided as a ratio measure.
    ## The ratio is percentage/100 (0.1 is equal to 10%). It has a theoretical range of # - infinite to infinite
        using a ratio measure.
    ## The equivalent range measured in degree is -90 degree to 90 degree.
    dx = math.cos(math.tanh(IfcVersegParabolicArc.StartGradient))
    dy = math.sin(math.tanh(IfcVersegParabolicArc.StartGradient))
    direction = OCC.gp.gp_Dir2d(dx, dy)

    if IfcVersegParabolicArc.IsConvex == True:
        ## When the parabolic arc is convex, the parabolaconstant is positive
        Vertex_StartDistAlong = (0 - IfcVersegParabolicArc.StartGradient) * IfcVersegParabolicArc.ParabolaConstant +
                                      self.StartDistAlong
        Vertex_StartHeight = (Vertex_StartDistAlong - self.StartDistAlong) * (0 +
                                      IfcVersegParabolicArc.StartGradient) / 2 + IfcVersegParabolicArc.StartHeight
        vertex_points = self.surface_wires.Value(Vertex_StartDistAlong, Vertex_StartHeight)

        parabola_on_surface = OCC.Geom2d.Geom2d_Parabola(OCC.gp.gp_Ax2d(OCC.gp.gp_Pnt2d(Vertex_StartDistAlong,
                                      Vertex_StartHeight), OCC.gp.gp_Dir2d(0, 1)),
                                      IfcVersegParabolicArc.ParabolaConstant/2, False)

        start_u = 0 - (Vertex_StartDistAlong - self.StartDistAlong)
        end_u = 0 - (Vertex_StartDistAlong - self.NextStartDistAlong)
        if IfcVersegParabolicArc.IsConvex == False:
            ## When the parabolic arc is convex, the parabolaconstant is negative
            Vertex_StartDistAlong = (0 - IfcVersegParabolicArc.StartGradient) * -IfcVersegParabolicArc.ParabolaConstant
                                       + self.StartDistAlong
            Vertex_StartHeight = (Vertex_StartDistAlong - self.StartDistAlong) * (0 +
                                       IfcVersegParabolicArc.StartGradient) / 2 + IfcVersegParabolicArc.StartHeight
            vertex_points = self.surface_wires.Value(Vertex_StartDistAlong, Vertex_StartHeight)

            parabola_on_surface = OCC.Geom2d.Geom2d_Parabola(OCC.gp.gp_Ax2d(OCC.gp.gp_Pnt2d(Vertex_StartDistAlong,
                                       Vertex_StartHeight), OCC.gp.gp_Dir2d(0, -1)),
                                       IfcVersegParabolicArc.ParabolaConstant/2, False)

            end_u = 0 - (self.StartDistAlong - Vertex_StartDistAlong)
            start_u = 0 - (self.NextStartDistAlong - Vertex_StartDistAlong)


        ## Project the Adaptor parabola on the linear surface of the Horizontal Wire
        adaptor_curve = OCC.Adaptor2d.Handle_Adaptor2d_HCurve2d(OCC.Geom2dAdaptor.Geom2dAdaptor_HCurve(
                                                                 parabola_on_surface.GetHandle()))
        adaptor_surface = OCC.Adaptor3d.Handle_Adaptor3d_HSurface(OCC.Adaptor3d.Adaptor3d_HSurfaceOfLinearExtrusion(
                                                                 self.surface_wires))
        curve_on_surface = OCC.Adaptor3d.Adaptor3d_CurveOnSurface(adaptor_curve, adaptor_surface)

        ## Create a geometric parabola curve
        VerSegParabolicArcSegement = OCC.Geom.Handle_Geom_Curve()

        ## Build the geometric parabola curve and edge from the Adaptor parabola with start and end parameters
        a, b = OCC.GeomLib.geomlib_BuildCurve3d(1e-9, curve_on_surface, start_u, end_u, VerSegParabolicArcSegement)
        edge_VerSegParabolicArcSegement = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(VerSegParabolicArcSegement)

        ## The function returns the edge of a parabolic arc segment
        return edge_VerSegParabolicArcSegement
```

```python
## PARSING VERTICAL CIRCULAR ARC SEGMENTS
def process_VerSegCircularArc(self, IfcVerSegCircularArc, NextSegment):
    ## Converting the StartGradient in a dx, dy and direction of a vertical line segment
    ## The gradient of the tangent of the vertical segment at the start point. It is provided as a ratio measure.
    ## The ratio is percentage/100 (0.1 is equal to 10%). It has a theoretical range of # - infinite to infinite
        using a ratio measure.
    ## The equivalent range measured in degree is -90 degree to 90 degree.
    dx = math.cos( math.tanh(IfcVerSegCircularArc.StartGradient))
    dy = math.sin( math.tanh(IfcVerSegCircularArc.StartGradient))
    direction = OCC.gp.gp_Dir2d(dx, dy)

    if NextSegment != None:
        if IfcVerSegCircularArc.IsConvex == True:
            start_point = self.surface_wires.Value(IfcVerSegCircularArc.StartDistAlong,
                                   IfcVerSegCircularArc.StartHeight)

            middle_pnt_x = self.StartDistAlong + (dx + (math.pi / 2) * IfcVerSegCircularArc.Radius)
            middle_pnt_y = IfcVerSegCircularArc.StartHeight + (dy + (math.pi / 2) * IfcVerSegCircularArc.Radius)
            middle_pnt = self.surface_wires.Value(middle_pnt_x, middle_pnt_y)

            point_3_x = middle_pnt_x - (dx  * IfcVerSegCircularArc.Radius)
            point_3_y = middle_pnt_y - (dy * IfcVerSegCircularArc.Radius)
            point_3 = self.surface_wires.Value(point_3_x, point_3_y)

            end_point = self.surface_wires.Value(NextSegment.StartDistAlong, NextSegment.StartHeight)

            VerSegCircularArc = OCC.gce.gce_MakeCirc(start_point, end_point , point_3).Value()

        if IfcVerSegCircularArc.IsConvex == False:
            start_point = self.surface_wires.Value(IfcVerSegCircularArc.StartDistAlong,
                                   IfcVerSegCircularArc.StartHeight)

            middle_pnt_x = self.StartDistAlong + (dx - (math.pi / 2) * IfcVerSegCircularArc.Radius)
            middle_pnt_y = IfcVerSegCircularArc.StartHeight + (dy - (math.pi / 2) * IfcVerSegCircularArc.Radius)
            middle_pnt = OCC.gp.gp_Pnt(middle_pnt_x, middle_pnt_y, 0)
        point_3_x = middle_pnt_x - (dx  * IfcVerSegCircularArc.Radius)
        point_3_y = middle_pnt_y - (dy * IfcVerSegCircularArc.Radius)
        point_3 = OCC.gp.gp_Pnt(point_3_x, point_3_y, 0)

        end_point = self.surface_wires.Value(NextSegment.StartDistAlong, NextSegment.StartHeight)

        VerSegCircularArc = OCC.gce.gce_MakeCirc(start_point, end_point , point_3).Value()

else:
    if IfcVerSegCircularArc.IsConvex == True :
        middle_pnt_x = self.StartDistAlong + (dx + (math.pi / 2) * IfcVerSegCircularArc.Radius)
        middle_pnt_y = IfcVerSegCircularArc.StartHeight + (dy + (math.pi / 2) * IfcVerSegCircularArc.Radius)
        middle_pnt = self.surface_wires.Value(middle_pnt_x, middle_pnt_y)

        start_dir_arc = IfcVerSegCircularArc.StartDirection - (math.pi / 2)
        end_dir_arc = start_dir_arc + (IfcVerSegCircularArc.SegmentLength / IfcVerSegCircularArc.Radius)

        VerSegCircularArc = OCC.gce.gce_MakeCirc(OCC.gp.gp_Ax2( middle_pnt, direction),
                                  IfcVerSegCircularArc.Radius).Value()

    if IfcVerSegCircularArc.IsConvex == False :
        middle_pnt_x = self.StartDistAlong + (dx - (math.pi / 2) * IfcVerSegCircularArc.Radius)
        middle_pnt_y = IfcVerSegCircularArc.StartHeight + (dy - (math.pi / 2) * IfcVerSegCircularArc.Radius)
        middle_pnt = self.surface_wires.Value(middle_pnt_x, middle_pnt_y)

        start_dir_arc = IfcVerSegCircularArc.StartDirection + (math.pi / 2)
        end_dir_arc = start_dir_arc - (IfcVerSegCircularArc.SegmentLength / IfcVerSegCircularArc.Radius)

        VerSegCircularArc = OCC.gce.gce_MakeCirc(OCC.gp.gp_Ax2(middle_pnt, direction),
                                  IfcVerSegCircularArc.Radius).Value()
```

```
## Project the Adaptor circular arc on the linear surface of the Horizontal Wire
adaptor_curve = OCC.Adaptor2d.Handle_Adaptor2d_HCurve2d(OCC.Geom2dAdaptor.Geom2dAdaptor_HCurve(
                                                        VerSegCircularArc.GetHandle()))
adaptor_surface = OCC.Adaptor3d.Handle_Adaptor3d_HSurface(OCC.Adaptor3d.Adaptor3d_HSurfaceOfLinearExtrusion(
                                                        self.surface_wires))
curve_on_surface = OCC.Adaptor3d.Adaptor3d_CurveOnSurface(adaptor_curve, adaptor_surface)

## Create a geometric circular arc
VerSegCircularArcsSegment = OCC.Geom.Handle_Geom_Curve()

## Build the geometric circular arc and edge from the Adaptor circular arc with start and end parameters
a, b = OCC.GeomLib.geomlib_BuildCurve3d(1e-9, curve_on_surface, 0, IfcVerSegCircularArc.HorizontalLength,
                                                        VerSegCircularArcsSegment)
Edge_VerSegCircularArcsSegment = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(VerSegCircularArcsSegment)

## The function returns the edge of a parabolic arc segment
return Edge_VerSegCircularArcsSegment
```

```
#######################################################
##                                                   ##
##           IFCALIGNMENT CHECKER - CHECKS           ##
##                                                   ##
#######################################################


## Import Mathematical functions
## Import Configuration file parser
## Import IfcOpenShell
## Import Python OpenCascade functions


import math

import ConfigParser

import ifcopenshell
import ifcopenshell.geom

import OCC.Adaptor2d
import OCC.Adaptor3d
import OCC.Approx
import OCC.BRep
import OCC.BRepAdaptor
import OCC.BRepAlgoAPI
import OCC.BRepBuilderAPI
import OCC.BRepClass
import OCC.BRepFill
import OCC.BRepLib
import OCC.BRepPrimAPI
import OCC.gce
import OCC.GCE2d
import OCC.Geom
import OCC.Geom2d
import OCC.Geom2dAdaptor
import OCC.GeomAdaptor
import OCC.GeomLib
```

```python
import OCC.gp
import OCC.IntCurvesFace
import OCC.ProjLib
import OCC.ShapeAnalysis
import OCC.ShapeExtend
import OCC.ShapeFix
import OCC.ShapeUpgrade
import OCC.StdPrs
import OCC.TColStd
import OCC.TopAbs
import OCC.TopExp
import OCC.TopoDS
import OCC.TopTools

## MAIN CLASS OF THE IFCALIGNMENT CHECKS
class Checks():

    ## INITIALIZING THE IFCALIGNMENT CHECKS
    def __init__(self,*args):
        self.object = 0

    ## CLASS 1: CURVE SEGMENT LENGTH
    def Curvesegment_Length(self, horizontalsegments):
        ## Ruleset which can be loaded by an ini file
        # config = ConfigParser.ConfigParser()
        # config.read ('RuleSet_CurveSegmentLengths.ini')

        ## RoadType should be implemented in the IfcAlignment data structure
        RoadType = '120'

        ruleset_curve_length = {
            '120' : {
                'minimum_length': 100
            },
            '90' : {
                'minimum_length': 75
            },
        '70' : {
            'minimum_length': 60
        },
        '50' : {
            'minimum_length': 40
        }
    }

    print config.get('curve_length','120')

    ## Array of failed segments
    errors = []

    ## Within all horizontal segments the IfcCircularArcSegments are determined
    for i, segment in enumerate(horizontalsegments) :
        if segment.CurveGeometry.is_a("IFCCIRCULARARCSEGMENT2D"):

            ## Each segment is tested to the RuleSet and if the segment failes, it is added to the array of
               failed segments
            if segment.CurveGeometry.SegmentLength < ruleset_curve_length[RoadType]['minimum_length']:
            # if segment.CurveGeometry.SegmentLength < config.get('curve_length', '120'):
                print segment.CurveGeometry.SegmentLength
                errors.append(segment)

            else:
                continue

    ## The function returns the array with failed segments
    return errors


## CLASS 2: LINE SEGMENT LENGTH
def Linesegment_Length(self, horizontalsegments):
    ## Ruleset which can be loaded by an ini file
    # config = ConfigParser.ConfigParser()
    # config.read ('RuleSet_LineSegmentLengths.ini')
```

```
## RoadType should be implemented in the IfcAlignment data structure
RoadType = '120'

ruleset_line_segment_length = {
    '120': {
        'curve_same': 480,
        'curve_opposite': 240,
        'max_length': 2400
    },
    '90': {
        'curve_same': 360,
        'curve_opposite': 180,
        'max_length': 1800
    },
    '70': {
        'curve_same': 280,
        'curve_opposite': 140,
        'max_length': 1400
    },
    '50': {
        'curve_same': 200,
        'curve_opposite': 100,
        'max_length': 1000
} }

## Array of all line segments
line_segments = []

## Within all horizontal segments the IfcLineSegments are determined
for line_segment in horizontalsegments :
    if line_segment.CurveGeometry.is_a("IfcLineSegment2D") == True:
        ## All line segments are added to the array of all line segments
        line_segments.append (line_segment)

## Array of failed segments
errors = []
```

```
## Each line segments is checked if it has a previous and a next segment
for line_segment in line_segments:
    index = horizontalsegments.index(line_segment)
    if index > 0 and index < len(horizontalsegments) - 1:
        prv = horizontalsegments[index - 1]
        nxt = horizontalsegments[index + 1]


        ## If the line segment has a previous and next segments, it checks if the Is Counter Clockwise is
           the same
        ## When the IsCcw is the same the segment is tested to the RuleSet and if the segment failes, it is
           added to the array of failed segments
        if prv.CurveGeometry.IsCcw == nxt.CurveGeometry.IsCcw:
            if line_segment.CurveGeometry.SegmentLength < ruleset_line_segment_length[
                                                                RoadType]['curve_same']:
            # if line_segment.CurveGeometry.SegmentLength < config.get('curve_same', '120'):
                errors.append(line_segment)
            if line_segment.CurveGeometry.SegmentLength > ruleset_line_segment_length[
                                                                RoadType]['max_length']:
            # if line_segment.CurveGeometry.SegmentLength > config.get('max_length', '120'):
                errors.append(line_segment)
            else:
                continue

        ## If the line segment has a previous and next segments, it checks if the Is Counter Clockwise is
           not the same
        ## When the IsCcw is not the same the segment is tested to the RuleSet and if the segment failes, it
           is added to the array of failed segments
        if prv.CurveGeometry.IsCcw != nxt.CurveGeometry.IsCcw:
            if line_segment.CurveGeometry.SegmentLength < ruleset_line_segment_length[
                                                                RoadType]['curve_opposite']:
            # if line_segment.CurveGeometry.SegmentLength < config.get('curve_opposite', '120'):
                errors.append(line_segment)
            if line_segment.CurveGeometry.SegmentLength > ruleset_line_segment_length[
                                                                RoadType]['max_length']:
```

```
    # if line_segment.CurveGeometry.SegmentLength > config.get('max_length', '120'):
                    errors.append(line_segment)
                else :
                    continue

    ## The function returns the array with failed segments
    return errors


## CLASS 3: HEADROOM FREE SPACE
def HeadRoom(self, projections, headroom_spaces):
    ## Function to check if and where two shapes intersects
    check_headroom = OCC.IntCurvesFace.IntCurvesFace_ShapeIntersector()

    ## Array of intersection point
    pnts = []

    ## For each surface from extrusion of the 3D alignment in the z-direction
    for i in range(0, headroom_spaces.Length()):
        ## Load the surface in the function to check on intersections
        ## Continue for all surfaces
        check_headroom.Load( headroom_spaces.Value(i+1), 1e-3 )

        ## For each parsed 3D Alignment
        for i in range(0, projections.Length()):
            ## Create an Adaptor curve to load in the function to check on interesections
            CompCurve = OCC.BRepAdaptor.BRepAdaptor_CompCurve( OCC.TopoDS.topods_Wire( projections.Value(i+1) ),
                                                                 True )
            HCompCurve = OCC.BRepAdaptor.BRepAdaptor_HCompCurve( CompCurve )

            ## Perform if the loaded surface and the loaded parsed 3D alignment have an intersection
            ## Continue for all parsed 3D alignments per surface
            check_headroom.Perform( HCompCurve.GetHandle(), -1000, 1000 )
            ## If there are intersections add these to the array of intersection points
            for i in range(1, check_headroom.NbPnt() + 1):
                pnts.append(check_headroom.Pnt(i))

    ## The function returns the array with intersection points
    return pnts


## CLASS 4: LINE OF SIGHT - SURFACE
def Line_of_Sight_surface(self, projections ):
    ## Create emty surface shapes container
    ruled_surfaces = OCC.TopTools.TopTools_HSequenceOfShape()

    ## For each parsed 3D Alignment
    for i in range(0, projections.Length()):
        ## Function which makes the surface has as input an edge
        ## Create the first Adaptor curve to convert the 3D Alignment in an edge
        complete_curve = OCC.BRepAdaptor.BRepAdaptor_HCompCurve( OCC.BRepAdaptor.BRepAdaptor_CompCurve(
                                  OCC.TopoDS.topods_Wire( projections.Value(i+1) ), True ))

        Comp_curve_1 = OCC.BRepAdaptor.BRepAdaptor_HCompCurve( OCC.BRepAdaptor.BRepAdaptor_CompCurve(
                                  OCC.TopoDS.topods_Wire( projections.Value(i+1)), True,
                                  complete_curve.FirstParameter(), complete_curve.LastParameter(), 1e-3 ) )
        Approx_wire = OCC.Approx.Approx_Curve3d( Comp_curve_1.GetHandle(), 1e-3, OCC.GeomAbs.GeomAbs_C0, 200, 1)
        Approx_edge = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge(Approx_wire.Curve())

        ## A line of sight is most likely to be 100 meters, so the 3D Alignment must be larger then 100 meters
        if complete_curve.LastParameter() - complete_curve.FirstParameter() > 100:
            ## Create the second Adaptor curve to convert the 3D Alignment with parameterized values in an edge
            ## Parameterization is done in the CompCurve third and fourth inputs:
            ## The parameters are the start and end value of the original complete curve added by the parameter
               from the guideline
            Comp_curve_2 = OCC.BRepAdaptor.BRepAdaptor_HCompCurve( OCC.BRepAdaptor.BRepAdaptor_CompCurve(
                                  OCC.TopoDS.topods_Wire( projections.Value(i+1)), True,
                                  complete_curve.FirstParameter() + 40, complete_curve.LastParameter() - 1, 1e-3))
                Approx_wire_2 = OCC.Approx.Approx_Curve3d( Comp_curve_2.GetHandle(), 1e-3, OCC.GeomAbs.GeomAbs_C0,
                                      200, 1)
                Approx_edge_2 = OCC.BRepBuilderAPI.BRepBuilderAPI_MakeEdge( Approx_wire_2.Curve() )

                ## The surface is made from the original 3D Alignment edge and the parameterized 3D Alignment edge
                ruled_surface = OCC.BRepFill.brepfill_Face( Approx_edge.Edge(), Approx_edge_2.Edge()  )
                ruled_surfaces.Append(ruled_surface)
                # print "SURFACE MADE"

            else:
                continue

        ## The function returns the filled container with surfaces
        return ruled_surfaces


    ## CLASS 4: LINE OF SIGHT - INTERSECTIONS
    def Line_of_Sight_shapes(self, ruled_surface, shapes ):
        ## Array of failed geometric shapes
        shapes_fail = []

        ## For each geometric shape
        for i in range(0, shapes.Length()):
            ## A ruled surface is inputted and checked if it intersect each individuel shape
            section = OCC.BRepAlgoAPI.BRepAlgoAPI_Section( ruled_surface, shapes.Value(i+1) ).Shape()

            ## If there is an intersection, the explorer checks if there are intersections
            ## If there are intersections, the shape is added to the array of failed geometric shapes
            exp = OCC.TopExp.TopExp_Explorer( section, OCC.TopAbs.TopAbs_EDGE )
            if exp.More():
                shapes_fail.append(shapes.Value(i+1))

        ## The function returns the filled container with failed geometric shapes
        return shapes_fail
```