A REFINED PRODUCT MODEL FOR SHIELD TUNNELS BASED ON A GENERALIZED APPROACH FOR ALIGNMENT REPRESENTATION

Julian Amann[1], André Borrmann[2], Felix Hegemann[3], Javier R. Jubierre[1], Matthias Flurl[4], Christian Koch[5], Markus König[6]

1) Ph.D. student, Chair of Computational Modeling and Simulation, Technische Universität München, Munich, Germany.
2) Dr.-Ing., Prof., Chair of Computational Modeling and Simulation, Technische Universität München, Munich, Germany.
3) Ph.D. student, Chair of Computing in Engineering, Ruhr Universität Bochum, Bochum, Germany.
4) Ph.D. student, Chair of Computation in Engineering, Technische Universität München, Munich, Germany.
5) Dr.-Ing., Chair of Computing in Engineering, Ruhr Universität Bochum, Bochum, Germany.
6) Dr.-Ing., Prof., Chair of Computing in Engineering, Ruhr Universität Bochum, Bochum, Germany.

Abstract: For realizing data exchange in the context of the planning and realization of large infrastructure projects, a comprehensive neutral data model capable of presenting both semantic as well as geometric aspects is necessary. The Industry Foundation Classes (IFC) provide a full-grown and standardized product model for the design and engineering of buildings. In the infrastructure sector, a comparably powerful data exchange solution is still missing. To fill this gap, this paper presents an alignment model which is based on the IFC data model and can be used as a data exchange standard for the design and maintenance of linear infrastructure facilities such as roads, bridges and tunnels. This paper presents a detailed account of the results of our data modeling activities. In particular, we demonstrate the use of the alignment sub-model by integrating it with a refined version of an existing shield tunnel product model. The proposed product model provides semantic entities, models the relationships between the physical objects and makes extensive use of the space aggregation concept inherited from standard IFC.

Keywords: Infrastructure, data exchange, alignment, IFC, tunnel product model

1. INTRODUCTION & MOTIVATION

There is a great need for data exchange in the architecture, engineering and construction (AEC) industry. A study from NIST in 2004 showed that more than 15.8 billion dollars are wasted each year due to insufficient data exchange in building projects (Venugopal et al. 2012). A proportionally similar amount of money wastage is likely in the field of infrastructure projects.

At present, there is very limited data exchange support for infrastructure elements such as roads, bridges or tunnels. For building construction, however, the Industry Foundation Classes (IFC) (buildingSMART 2013) have already become established as a mature standard. Something similar is currently lacking in the infrastructure sector. In this paper we describe, an alignment model that supports the well-established approach of aligning design based on vertical and horizontal alignments. The model supports also 3D space curves for alignments, which can be used, for instance, for alignment recording by GPS. We show how the IFC 4 standard can be extended with our alignment model. Furthermore, we describe how the alignment data model can be interconnected with an extended shield tunnel product model.

2. BACKGROUND

2.1 Related work

IFC-Bridge is an extension of the IFC Standard which is currently in ongoing development (Yakubi et al. 2006; Lebegue et al. 2012). It extends the current IFC data product model to include items of bridge design. A crucial part of the bridge design is the alignment curve. For this, the authors introduce an *IfcReferenceCurveAlignment2D* element which references a horizontal and a vertical alignment curve. For both the horizontal and vertical alignment, an *IfcCurve* element is used. Since *IfcCurve* is very general, it allows many different types of descriptions of curves, which makes it hard for software application implementers to handle all types and combinations of curve types. For example, editing the start or end radius of a clothoid is also difficult if it is described by arbitrary curve elements.

LandXML (Harrison & Ziering, 2007) is a standard that is targeted at road design. It is based on the Extensible Markup Language (XML) (Bray et al. 2006). Besides the alignment, it also offers the ability to store cross-sections of the corresponding road and offers a digital elevation model in the form of a triangle net. LandXML instance files can be validated against the LandXML schema definition (XSD). With the help of LandXML XSD, LandXML instance files can be checked for syntax errors. With the change from LandXML schema 1.1 to 1.2, it has been a main target to preserve downwards compatibility; meaning that a LandXML 1.1 reader can still read LandXML 1.2 instance files. As a consequence, the introduced modifications appear only on the content level of the XML instance file and do not have an impact on the structural level of the XML schema. However, this approach opened the door for misuse of the standard and results, for instance, in several different interpretations and uses of various variants for defining cross sections.

IFC-Tunnel is supposed to become an extension of the IFC Standard which provides data structures for tunnel buildings. It is mainly driven by German IFC Tunneling Project (Hegemann et al. 2012) and the Japanese Shield-Tunnel Project (Yabuki et al. 2007, Yabuki 2008). Neither of these works focuses on an alignment model. This paper closes this gap.

2.2 Objectives

This paper describes a generalized IFC 4 based alignment model that can be used in the field of infrastructure to describe road, tunnel and bridge alignments. In addition, we show how the proposed alignment model can be integrated in a refined version of an existing shield tunnel product model. The demonstrated model can be easily extended and used in other infrastructure domains such as bridge or road design. A proposal of an extended IFC 4 standard is also given and it is shown how it can be mapped to existing IFC 4 elements.

3. AN IMPROVED ALIGNMENT DATA STRUCTURE

3.1 Requirements for alignment data structure

An alignment data structure has to fulfill several requirements to be useful for infrastructure modeling. Since the user should be provided with the possibility to apply the well-established approach of using horizontal and vertical alignments besides directly storing a 3D curve, the corresponding vertical and horizontal alignment elements need to be stored.

The data structure must be able to support the drawing and modification of horizontal and vertical alignments. Moreover, it needs to be possible to generate a 3D space curve based on these alignment representations. Furthermore, it needs to be simple for software developers to adapt and integrate the alignment product data model to their software products. Since the presented alignment data model is based on the IFC 4 standard, it should re-use as many data structures from the existing standard as possible and should not duplicate data structures already present. The alignment data product model should also contain only necessary data and not data that can be derived. Additional data such as the current kilometrage also need to be included in the data model.

Figure 1 shows the data that is sufficient to reconstruct and edit different horizontal alignment segments of our test data. A horizontal alignment consists of horizontal alignment segments like straight line segments, circle segments (arcs) and transition curves. Since only clothoids are used as transition curves, only clothoids are provided in the data model. In this view (Figure 1), only the data that is needed for one specific horizontal alignment segment is considered without the knowledge of other horizontal alignment segments. It will be shown later how horizontal alignment segments can be connected. It is important for the requirements of an alignment model that it needs to be possible to connect different alignment segments and it also needs to be possible to provide a unique chainage.

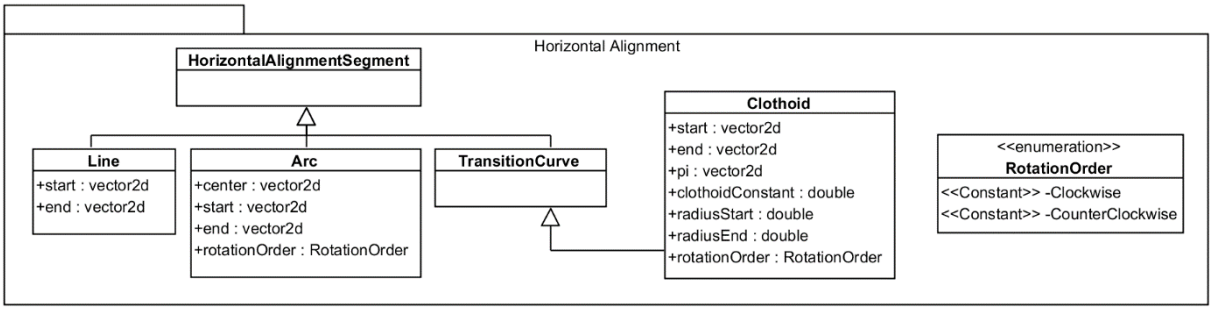

Figure 1. UML class diagram showing the requirements for the horizontal alignment data model.

For straight lines, the start and corresponding endpoints need to be stored. A circle segment (arc) can be described by its start, end and center point. Additionally, the rotation order (clockwise or counter clockwise) needs to be included. For clothoids, the start, end and point of intersection must be stored. The length of the clothoid, the start and end radius and rotation order also have to be stored. The length of the clothoid can be used to calculate the clothoid constant. Given this information, a horizontal alignment can be constructed as can be seen in Figure 2.

The line segments of the horizontal alignment can be used to compute intersection points. Thereby two successive line segments become prolonged and their point of intersection (PI) gets computed. This point is usually used to modify a track model. The PI can be moved. Afterwards, the lines are modified according to the movement of the PI. Eventually, the parameters of interjacent non line segment are computed so they match the new start and end point of the moved lines segments.

The vertical alignment is constructed by line segments which describe the gradient in the height field of

the underlying terrain (elevation model), vertical intersection points which describe the intersection of these line segments and roundings between the vertical alignment segments. Figure 3 shows which data is needed to store a vertical intersection. For roundings, only parabolas are used.

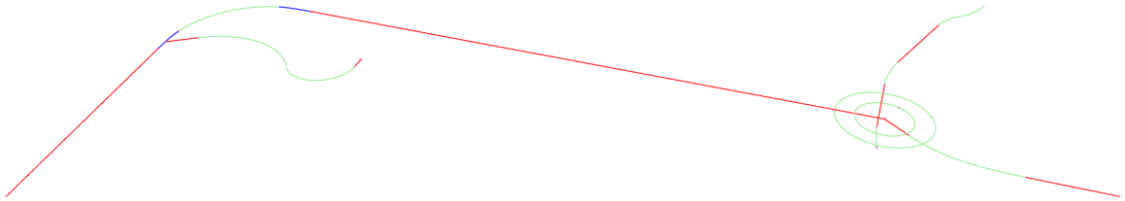

Figure 2. Line segments (red), arcs (green) and clothoids (blue)

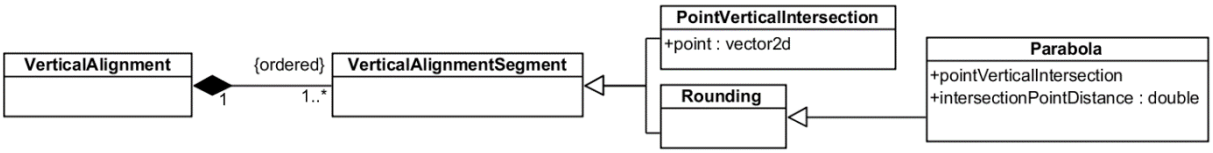

Figure 3. UML class diagram showing the requirements for the vertical alignment data model.

Figure 4 demonstrates how a vertical alignment can be constructed from the previously described data model. From the intersection point distance (distance between the point PVC and PVT) and the vertical intersection point of the parabola, the corresponding points PVC and PVT can be computed. The final computed vertical alignment consists of line segments and parabolas. The vertical alignment is a development drawing of a corresponding horizontal alignment. This requires the alignment data model to store also the correlation of the horizontal and vertical alignment. So another requirement is the correlation of the two alignments and that it is possible to compute a 3D space curve from both alignments.

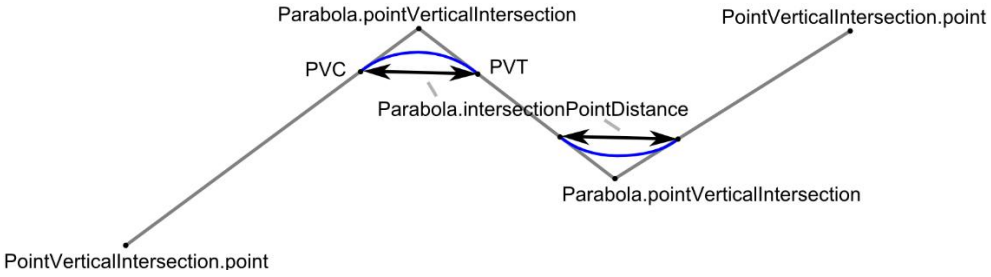

Figure 4. A vertical alignment.

3.2 A proposal for a new alignment data structure

In the previous section we discussed which requirements an alignment model has to fulfill. In this section we present the realization of an alignment model that actually meets those requirements.

First we consider an alignment in its top-level view. An alignment can be a 3D reference curve or a 2D alignment consisting of a horizontal and a vertical alignment. For the latter, some implementation defined restrictions have to be characterized. A horizontal alignment does not contain any junctions or gaps. Junctions are forbidden as a connection with a vertical alignment would thus be impossible. In the case of a junction, it is not clear how a proper vertical alignment can be developed. Gaps in a horizontal alignment are also not allowed because in this case we would also have to store this gap in the vertical alignment which would only make the data model more complicated. For junctions and gaps, another alignment with its own horizontal and vertical alignment has to be created. So instead of creating an alignment with a horizontal and a vertical alignment with a gap, we create two alignments each with its own junction and gap free horizontal and vertical alignment. This restriction simplifies the presented data model and does nevertheless support gaps and junctions. Since the introduced horizontal alignment and also the vertical alignment are gap free, the end point of an alignment segment is always the start point of a following alignment segment. So the same point is referenced two times, one time as start- and another time as endpoint. But this is acceptable considering the memory consumption (references are cheap). Moreover, storing just one reference would make the model cumbersome to use.

From the horizontal and vertical alignment, a 3D space curve can be computed. In the simple case where the horizontal and vertical alignments have the same length, the corresponding chainage in the horizontal and

vertical alignment just have to be found. If the vertical alignment has a different length, it is not invalid. A proportionality factor will be computed and the vertical alignment will be sized so it has the same length as the horizontal alignment. The horizontal alignment stores also the start chainage. The following chainages can be computed by summing up the different lengths of the horizontal alignment segments.

Figure 5 shows an overview of the proposed alignment data structure. On the top level there is the *IfcReferenceCurve* element. The model supports a 3D space curve (*IfcReferenceCurve3D*) as well as the traditional approach of horizontal and vertical alignments (*IfcReferenceAlignment2D*). The IfcReferenceAlignment2D consist of a gap and junction free horizontal (*IfcHorizontalAlignment*) and vertical alignment (*IfcVerticalAlignment*). The *IfcHorizontalAlignment* consist of an order list of *IfcHorizontal-AlingmentSegments*. An *IfcHorizontalAlingmentSegment* is a superclass of *IfcHorizontalAlignmentLine* for line segments, *IfcHorizontalAlignmentCircularSegment* for circle segments and *IfcHorizontalAlignmentTransiton-Curve* for transition curves. The only supported transition curve is the *IfcHorizontalAlignmentClothoid* for a clothoid.

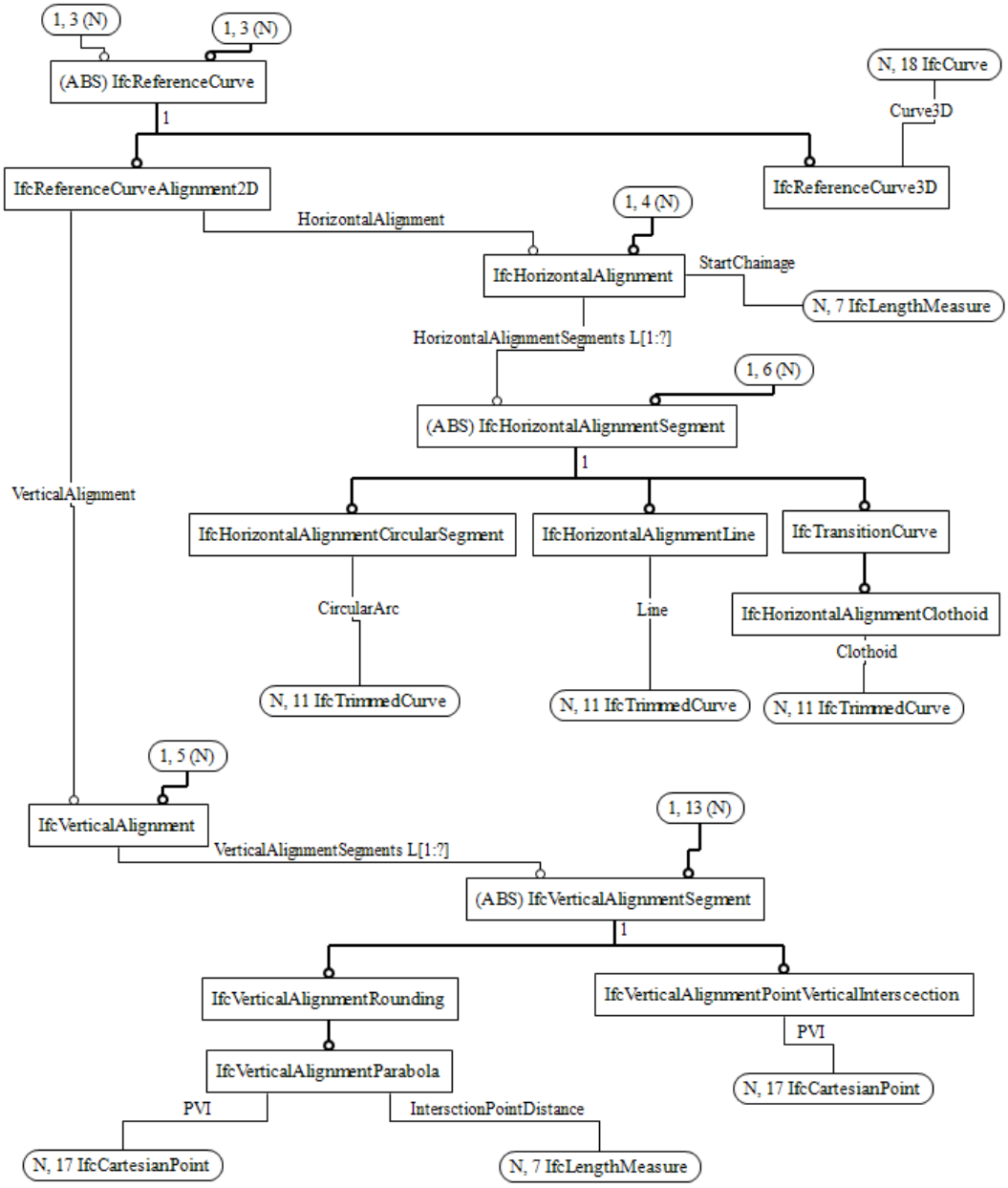

Figure 5. EXPRESS G-Diagram giving an overview of the proposed alignment data structure.

The vertical alignment is stored in the *IfcVerticalAlignment* element. It consists of an ordered list *IfcVerticalAlignmentSegments* such as *IfcVerticalAlignmentPointVerticalIntersection* and *IfcVertical-AlignmentRounding*. An *IfcVerticalAlignmentRounding* has only one subclass (*IfcVerticalAlignmentParabola*).

A horizontal alignment line segment can be described by a trimmed curve (*IfcTrimmedCurve*) that is using an *IfcLine* as a basis curve. The start point is used as the origin of the *IfcLine* and the difference vector between the end- and start point is used as direction vector. The first and second trim points can be set to the start and end of the corresponding line. The horizontal alignment circular segment also uses a trimmed curve. *IfcCircle* is used as a basis curve for the trimmed curve. The first and the second trim point are again set to the start and end point of the arc. The *IfcTrimmedCurve* offers a sense agreement which can be used to store if the circle is clockwise or counterclockwise. A horizontal alignment clothoid curve segment also has a start and an end point. These points are stored again in the first and the second trim point of the *IfcTrimmedCurve*. The sense argument of the trimmed curve is again used to store the rotation sense of the clothoid. Currently, there is no *IfcClothoid* curve in the IFC standard which can be used as a basis curve. An *IfcClothoid* element can easily be introduced in the IFC standard to circumvent this problem. Table 1 shows in detail the mapping of the horizontal alignment line and circle segment to an ASCII STEP file.

Table 1. Mapping of alignment segments to IFC4/STEP

IfcHorizontalAlignmentLine	IfcHorizontalAlignmentCircularSegment
#2=IFCHORIZONTALALIGNMENTLINE(#3); #3=IFCTRIMMEDCURVE(#4,(#8),(#9),,$); #4=IFCLINE(#5,#6); #5=IFCCARTESIANPOINT((1031.95,1177.96)); #6=IFCVECTOR(#7,1); #7=IFCDIRECTION((76.1796,252.095)); #8=IFCCARTESIANPOINT((1031.95,1177.96)); #9=IFCCARTESIANPOINT((1108.13,1430.06));	#11=IFCHORIZONTALALIGNMENT CIRCULARSEGMENT(#12); #12=IFCTRIMMEDCURVE(#13,(#16),(#17),.T.,$); #13=IFCCIRCLE(#14,86.6106); #14=IFCAXIS2PLACEMENT2D(#15,$); #15=IFCCARTESIANPOINT((1254.2,1394.62)); #16=IFCCARTESIANPOINT((1113.24,1445.92)); #17=IFCCCARTESIANPOINT((1165.1,1515.29));

In contrast to the proposal contained in IFC-Bridge (Arthaud and Lebegue, 2012), the 2D horizontal alignment does not only reference to an *IfcCurve* element. Instead, we decided to introduce a semantic level. Usually horizontal aligning design is based on line segments, circular arcs and transition curves (clothoids). The presented model reflects this semantic level. The advantage of this model is that it is more practical for modification. In a typical use case where a construction engineer modifies an alignment, he or she wants to select the intersection points of the alignment and move them by drag and drop as shown in Figure 6.

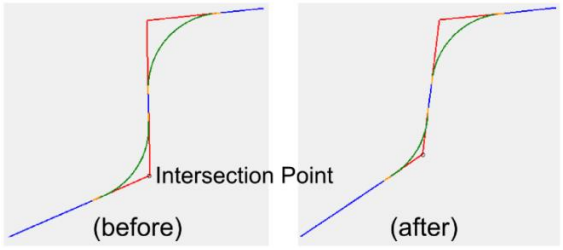

Figure 6. Modifying a horizontal alignment.

Modifying an alignment can be easily achieved with the data model presented here. Given the horizontal alignment segments, the intersection points can be easily computed. With IFC-Bridge, the alignment is described by a complex *IfcCurve* and a corresponding software implementation has therefore be able to handle all different types and kinds of *IfcCurves* configurations. Because there are different ways to describe a semantic line segment with the help of *IfcCurve*, the software needs to be able to handle each different way. Things get more difficult if a spline is used in the horizontal alignment of IFC-Bridge that has a straight part. This straight part is in fact a line segment with intersection points. This makes the IFC-Bridge alignment model impractical for modifications.

3.3 Computation of vertical alignments

The start point of a vertical alignment is a so called point of vertical intersection. This name is chosen because connected points of vertical intersection describe gradient lines. Typically, a construction engineer drafts these lines first and then replaces the discontinuous transitions with roundings, which are typically parabolas. From the intersection points and the distance of the left PVC and right PVT point (see Figure 4) the PVT and PVC can be calculated. First the slope *m* of the first line segment (PointVerticalIntersction.point to Parabola.pointVerticalIntersection) is computed. Given the PVI point of the parabola and the slope *m*, the y-intercept can be calculated. The length between the PVT and PVC is called the parabola span. The x-value of PVT can be computed by Equation (1). The corresponding y-coordinate can then also be calculated.

$$PVT.x = PVI.x - 0.5 * parabolaSpan \quad (1)$$

After computing these points, the vertical alignment can be viewed in terms of line segments and parabolas instead of points of vertical intersections and roundings. While the latter view is effectively more useful when a 3D space curve needs to be computed from the horizontal and vertical alignment, the intersection points are very important for editing a vertical alignment. For this reason, this approach has been chosen to store the vertical alignment data.

4 INTEGRATION INTO A SHIELD TUNNEL PRODUCT MODEL

4.1 Overview

The tunnel product data model proposed here (see Figure 7) is a refined version of the shield tunnel product model originally introduced by Yabuki et al. (2007). The model provides a fine-grained semantic model that precisely models the relationships between the physical objects and makes extensive use of the space aggregation concept inherited from standard IFC.

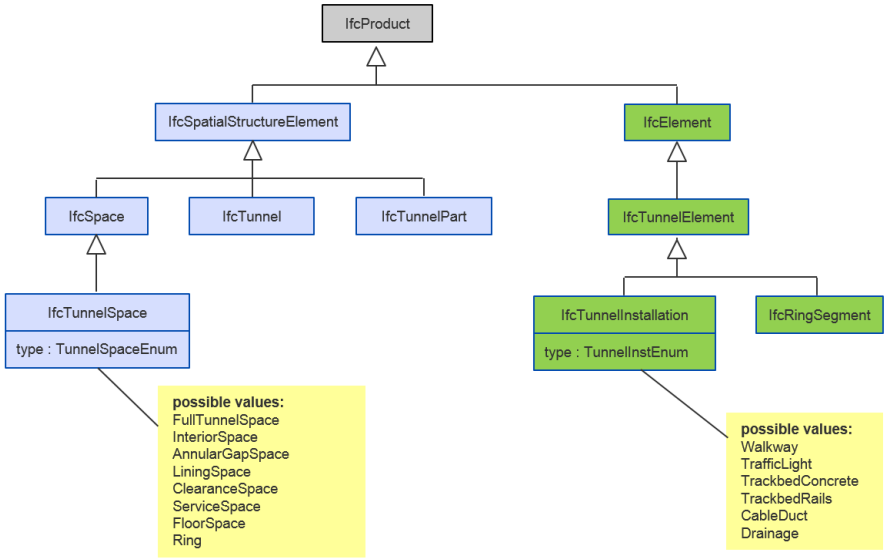

Figure 7. Proposed shield tunnel product model

In order to show the applicability of the proposed alignment model for modelling linear infrastructure facilities, we discuss its integration with a semantic shield tunnel product model. In contrast to the very fine grained class model presented in (Yabuki 2008), which introduces a large number of new classes, we restrict the extensions in the model proposed here to only six new classes. This is in accordance with the general principles of product modeling which aims to achieve maximum expressivity with minimum modeling effort.

In analogy to the space structure capabilities provided by standard IFC for buildings, we introduce the classes *IfcTunnel* (which corresponds to *IfcBuilding*) and *IfcTunnelPart* (which corresponds to *IfcBuildingStorey*), both modeled as subclasses of *IfcSpatialStructureElement*. In addition, the IFC class *IfcSpace* is sub-classed by the more specific *IfcTunnelSpace*. However, instead of undertaking a further sub-classing of *IfcTunnelSpace*, we make use of the enumeration attribute *type* to specify the particular space type present (*FullTunnelSpace, InteriorSpace, AnnularGapSpace, LiningSpace, ClearanceSpace, ServiceSpace, FloorSpace, Ring*). In a concrete tunnel model, instances of *IfcTunnel*, *IfcTunnelPart* and *IfcTunnelSpace* are arranged in an aggregation hierarchy as depicted in Figure 8.

For representing physical elements of the shield tunnel, we introduce the class *IfcTunnelElement* as a subclass of *IfcElement*. *IfcTunnelElement* is in turn subclassed by *RingSegment* and *IfcTunnelInstallation*. For the latter, we again provide the enumeration attribute *type* to specify the particular installation described (*Walkway, TrafficLight, TrackbedConcrete, TrackbedRails, CableDuct, Drainage*). The physical objects are associated with the respective space objects via relationship objects of type *IfcRelContainedInSpatialStructure* (see Figure 8). The semantics of the individual elements are illustrated in Figure 9.

Due to the tunnel's nature of being a linear infrastructure facility, the geometry of large parts of its comprising objects can be represented by an extrusion of a profile along the tunnel axis. In particular, this applies to the *IfcTunnelSpace* objects and the *IfcTunnelElement* objects. (Please note that *IfcRingSegment* cannot be modeled by means of an extruded geometry.)

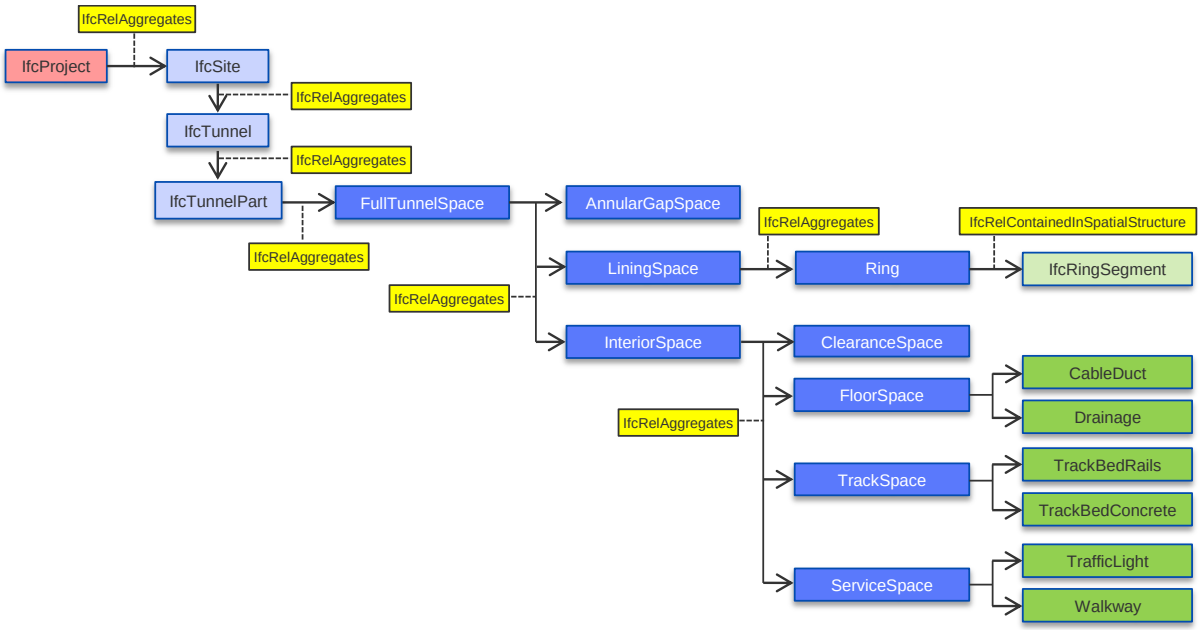

Figure 8: Instance diagram presenting the space aggregation hierarchy. Elements depicted in dark blue represent instances of *IfcTunnelSpace* using the type attribute for specifying the respective type. Elements depicted in dark green are instances of *IfcTunnelInstallation*.

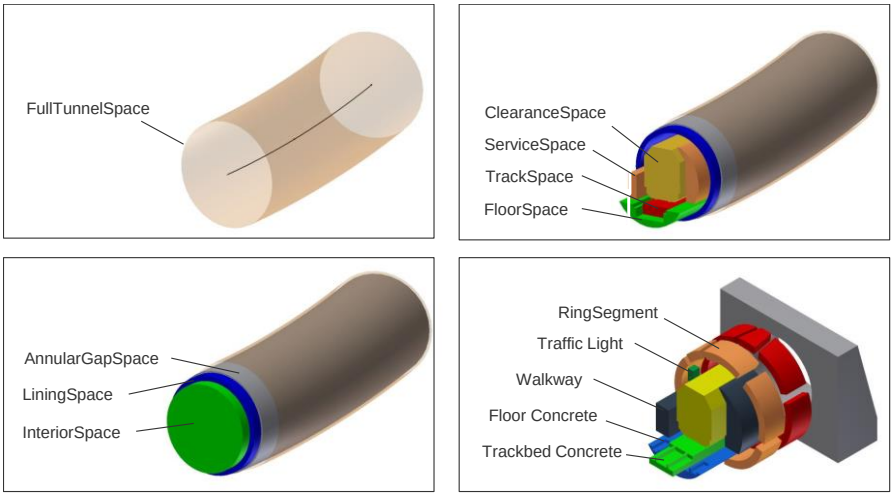

Figure 9. Illustration of the semantics of the individual elements of the proposed shield tunnel product model.

For representing the geometry of the respective space and tunnel element objects, it is made use of the entity *IfcSurfaceCurveSweptAreaSolid* provided as part of the standard IFC. Using the attribute Directrix it is possible to assign an *IfcCurve* object, which is used as sweeping path. In this case, the tunnel's axis is used for this purpose. The swept area is provided by a subtype of *IfcProfileDef* via the attribute *ReferenceSurface*. For most objects, *IfcArbitraryClosedProfileDef* is most appropriate for modelling the cross-section profile. Its *OuterCurve* attribute allows association of any *IfcCurve* object.

4.2 Tunnel alignment

To store the relationship between the *IfcTunnel* entity and the alignment model, an *IfcTunnelAxis* is introduced. The *IfcTunnel* entity references this *IfcTunnelAxis*. Again *IfcTunnelAxis* references to an *IfcReferenceCurve* which describes the alignment of the tunnel using the traditional approach of a horizontal and a vertical alignment (*IfcReferenceCurveAlignment2D*) or a 3D space curve (*IfcReferenceCurve*). The tunnel axis is used as an extrusion path for the tunnel profile. To find out how long a specific element of a tunnel profile has to be extruded, we store the stationing of each tunnel element in the *IfcTunnelAxis* entity. This enables us to automatically generate a new tunnel from the description, including internal elements such as traffic lights, when the alignment of the tunnel is changed.

5 VALIDATION

To validate the proposed approach, the tunnel product model is implemented in the IFC-compatible visualization tool Open IFC Tools. Open IFC Tools is a fully object-oriented Java-based application which can be used to read and write IFC STEP files. Figure 10 shows a part of a generated tunnel model imported by means of an IFC STEP file. To model the tunnel, the presented IFC-based modeling approach has been applied. On the right side of the figure, the structure and the hierarchy of the tunnel model can be seen. On the left side, the visualization of the model is presented. This shows that the approach presented here can be applied in IFC compatible software tools which implement the IFC Tunnel extension presented here.

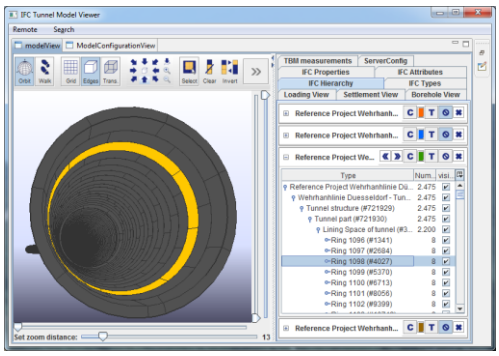

Figure 10: Tunnel model loaded into Open IFC Tools

6 CONCLUSION & FUTURE WORK

In this paper we demonstrated an approach for storing alignment data that is based on an extension of the IFC4 standard. The proposed alignment structure can be easily extended with new transition curves or roundings if needed. We also discussed how to integrate the suggested alignment structure in a tunnel product model. The alignment model shown is very versatile and can also be integrated in other IFC-based infrastructure extensions such as IFC-Bridge.

In a future development, other aspects of alignment design can also be taken into consideration such as the underlying terrain or the required ground work. Even though there are existing standards such as LandXML or GroundXML (Obergrießer 2009), an IFC extension targeted for infrastructure projects seems very promising.

REFERENCES

buildingSMART. (2013). Industry Fondation Classes IFC4 Official Release, buildingSMART http://www.buildingsmart-tech.org

Bray, T., Paoli, J. Sperberg-MCQueen, C. M., Maler, E. and Yergau, F. (2013): Extensible Markup Language (XML) 1.0 (Fourth Edition). World Wide Web Consortium, *http://www.w3.org/TR/2006/REC-xml-20060816/#sec-origin-goals*. 2006.

Harrison F. D. and Ziering E. (2007) National Research Council. *NCHRP Report 576: TransXML: XML Schemas for Exchange of Transportation Data*. Washington, DC: The National Academies Press, 2007.

Hegemann, F., Lehner K., König, M. (2012), IFC-based product modeling for tunnel boring machines, Proc. of European Conference on Product and Process Modeling, Reykjavik, Island

Lebegue, E., Fiês, B. and Gual, J., (2012). IFC-Bridge V3 Data Model – IFC 4, Edition R1.

Obergrießer M., Ji Y., Baumgärtel T., Euringer T., Borrmann A., and Rank E. (2009). GroundXML - An addition of alignment and subsoil specific cross-sectional data to the LandXML scheme *In: Proc. of the 12th International Conference on Civil, Structural and Environmental Engineering Computing*. Madeira, Portugal.

Venugopal, M., Eastman, C., and Teizer, J. (2012) An Ontological Approach to Building Information Model Exchanges in the Precast/Pre-Stressed Concrete Industry. *Construction Research Congress*.

Yabuki, N., Lebeque, E., Gual, J., Shitani, T. and Li, Z. T., (2006). International Collaboration for Developing the Bridge Product Model IFC-Bridge, *In Proc. Of the International Conference on Computing and Decision Making in Civil and Building Engineering.*

Yabuki, N., Azumaya, Y., Akiyama, M., Kawanai Y., Miya, T. (2007): Fundamental Study on Development of a Shield Tunnel Product Model. Journal of Civil Engineering Information Application Technology 16, pp. 261-268

Yabuki, N. (2008). Representation of caves in a shield tunnel product modeling. *In Proc. of the 7th European Conference on product and Process Modeling.* Sophia Antipolis, France.