A Graph Transformation Based Method for the Semi-Automatic Generation of Parametric Models of Shield Tunnels

S. Vilgertshofer, A. Borrmann
Chair of Computational Modeling and Simulation, Leonhard Obermeyer Center
Technical University of Munich, Germany
simon.vilgertshofer@tum.de

Abstract. During the design of large infrastructural projects like inner-city subway tracks, it proves necessary to consider differing model scales, ranging from kilometers to centimeters. This problem can be addressed by using multi-scale product models comprising multiple levels of detail (LoD). Ensuring consistency across the different LoDs can be achieved by applying parametric modeling techniques while creating the model. However, the correct application of constraints and dependencies has shown to be too complex to be handled manually. To address this issue, this papers presents an automated detailing mechanism, which is based on the use of graphs and graph transformations. We show how procedural parametric models based on two-dimensional sketches can be represented by graphs and how refinement steps can be realized through the rule-based transformation of such a graph.

1. Introduction

The ongoing digitalization of planning processes in the building sector has led to increasing demands on the complexity of the underlying digital models. One requirement on such a digital model is its capability to represents substantially diverging scales and levels of details. In particular, this is the case in the planning of large infrastructure facilities ranging over several kilometers (tunnels or roads). To efficiently assist planners, the corresponding models need to support the efficient modeling and management of geometric objects on both, large and small scales.

One approach to representing spatially extended facilities with an adaptive semantic and geometric resolution is the use of multiple levels of detail (LoDs). This approach is well established in the domain of Geographic Information Systems (GIS). Previous research extends the LoD concept used in the GIS domain towards consistent multi-scale representations of building information models, particularly used for the modeling of shield tunnels. One conclusion of this research is that the manual creation of these consistency preserving models is very complex, time consuming and error-prone and should be supported by automation mechanisms (Borrmann et al., 2014b).

This paper presents one possible approach to realize these automation mechanisms. It describes a detailing automation approach, which is based on the use of graphs and graph transformations. It further shows how procedural parametric three-dimensional geometric models based on two-dimensional sketches can be represented by graphs and how refinement steps can be realized through the rule-based transformation of these graphs.

The prospected benefit of this approach for end users is the reduction of the effort to manually create the consistency preserving models. Doing so, it allows them to focus on the conceptual and engineering aspects of the design process instead of time-consuming and repetitive modeling operations.

This paper is structured as follows: In Section 2 related works and the theoretical background of the research are outlined. Section 3 discusses the general approach of using graphs to represent a parametric 3D model. Section 4 focuses on the development of a specific graph rewriting system as a case study and describes the prototypical software tool that is used to interpret the graph-based representation and creates the actual 3D model. The paper concludes with a summary and a discussion of future work.

2. Related Work and Theoretical Background

2.1 Multi-Scale Modeling of Shield Tunnels

The idea of modeling and visualizing buildings and infrastructure facilities in different LoDs has been well established in the GIS domain for several years. The general concept is to provide different geometric representations of a semantic object. Which of these representations is actually used to display the object depends on the particular context, for example with respect to the user's requirements. An important example is CityGML, an XML-based data model for the representation of mainly static 3D city models (Kolbe 2009). CityGML defines five LoDs where representations in coarser LoDs are generated from finer ones by abstraction. In contrast, the design process of construction projects starts with a very general representation that gets gradually refined and more and more detailed as planning evolves (Borrmann et al. 2014). Additionally, the concept of CityGML does not include any automated consistency preservation mechanisms to ensure that changes in one LoD are propagated to the other LoDs due to the independent storage of an object's geometry in each LoD.

To fulfill the needs of consistency-preservation and multi-LoD representation, a multi-scale product model for shield tunnels has been developed by Borrmann and Jubierre (2013), which is based on a single scale model introduced by Yabuki et al. (2013). This multi-scale model uses procedural geometry description and parametric modeling techniques to provide mechanisms for automatic consistency preservation across the different LoDs. A 3D representation of the five LoDs of the tunnel model is depicted in Figure 1.

Figure 1: A 3D representation of the LoDs of the multi-scale shield tunnel product model (Borrmann and Jubierre, 2013). The first LoD, which only represents the alignment, is not depicted.

2.2 Parametric and Procedural Modeling

The concept of parametric modeling was developed in the 1990s (Shah and Mäntylä 1995) and is by now well established and used in many commercial and open source CAD applications e.g. Autodesk Inventor, Siemens NX and FreeCAD. While mainly applied in mechanical engineering, the concept has also been used to create flexible models of infrastructure facilities (Ji et al., 2013).

Parametric geometric 2D models (sketches) are composed of geometric objects and parametrical constraints. A system of constraints and objects is defined during the creation of a sketch in a parametric CAD application and forms a constraint problem, which can be solved

by an implemented geometric constraint solver (GCS) (Fudos and Hoffmann, 1997; Owen, 1991). The set of parametric constraints that is implemented by all major constraint solvers is defined as the *standard geometric constraint language* by (Schultz et al., 2015). It comprises the dimensional constraints for distances and angles as well as the following geometric constraints: *coincident, collinear, tangential, horizontal, vertical, parallel, perpendicular* and *fixed*.

The core concept of procedural modeling is to store not only the final outcome of a modeling process, but instead the sequence of single sketching and modeling operations. Models created this way are called procedural models or construction history models. They use the concept of parametric modeling to create flexible 2D sketches. These sketches form the basis for the procedural operations that generate 3D geometry by extrusions, sweeps, lofts or Boolean operations (Borrmann et al., 2012; Mun et al., 2003).

The presented approach uses parametrical constraints as listed above in combination with the procedural modeling of geometry to define dependencies between geometric objects belonging to different LoDs as described in (Borrmann et al., 2014a). Thus, the consistency of the model across multiple LoDs can be ensured.

2.3 Graph Rewriting

The proposed concept for automating the detailing process is based on graph theory and graph rewriting methodology (Rozenberg 1997). Graphs and graph rewriting mechanisms are employed to enable the representation and the modification of the procedural parametric models. An application of graph rewriting to semi-automatically create and alter parametric sketches has been presented in Vilgertshofer and Borrmann (2015).

Graph rewriting operations are used to create a new graph out of an existing graph by altering, deleting or replacing parts (subgraphs) of the existing graph. The changes are formalized through graph rewrite rules written as $L \rightarrow R$. A graph rewrite rule is defined by a pattern graph L and a replacement graph R. When a rule is applied to a graph (called the host graph), this graph is searched to find a subgraph that matches the graph pattern defined by L. If the matching succeeds L is replaced with R under the consideration of a preservation morphism that determines how an instance of L in the host graph is replaced or altered by R. There are several different approaches to graph rewriting. Two main examples are the Single-Pushout Approach (SPO) and the Double-Pushout Approach (DPO) (Heckel, 2006).

3. Conceptual Approach

To represent and detail a procedural parametric model, three major challenges are discussed after the general design of the graph representing a model is addressed. First, the so called *graph metamodel* is introduced. The metamodel defines a library of the node and edge types that the graph may consist of as well as possible attributes they may have. Secondly, to create and alter a graph based on this metamodel the requirements on graph rewrite rules are evaluated. These rules formally describe detailing steps that an end user can apply instead of manually executing the underlying procedural or parametric modeling operations. Last, we discuss the process of developing the graph metamodel and rewrite rules. These must meet requirements concerning the representation to be unambiguous and translatable into an actual geometric 3D model by interpreting the graph-based representation.

3.1 Definition of the Graph Metamodel

To create and detail a multi-scale geometric model by using graph rewrite operations, it is necessary to determine the composition of the graph which is used to represent this model. This is achieved by the definition of the graph metamodel. The metamodel defines the types of graph entities (nodes and edges) that may be used by a graph rewriting system to assemble a graph that represents a specific model. Besides the mere types of graph entities, the metamodel furthermore defines their attributes as well as conditions that determine which nodes and edges may be incident or which node types can be adjacent.

Formally, we define the graph representing a procedural geometry model as a directed multigraph with loops $G = \{V, E\}$. V is the set of vertices that represent either procedural modeling operations in the context of the procedural graph or geometric objects in the sketch graph. E on the other hand is a set of edges that connect these vertices. The edges are used to represent the general relations between the procedural operations or specific parametric constraints between geometric objects in a sketch graph. Thus, we define V_P and E_P as vertices and edges conceptually belonging to the procedural graph G_P. Further V_S and E_S are defined as the vertices and edges conceptually belonging to the sketch graph: The complete graph representing a model is described as $G = \{V_P + V_S, E_P + E_S\}$. The metamodel describes the sets of possible objects that V and E may contain.

The graph metamodel that was successfully used to generate models up to LoD 3 and parts of a LoD 4 model of a shield tunnel is described in Section 4.

3.2 Design of the Graph used for Model Representation

In a preliminary concept (Vilgertshofer and Borrmann, 2015) two individual types of graphs were considered necessary to represent two-dimensional sketches (*sketch graph*) and the procedural operations (*procedural graph*). While the conceptual separation of these graphs, which is caused by the different nature of the relations in parametric sketches and procedural operations, persists, the sketch graph needs to be integrated into the procedural graph. This is due to the fact that sketches are the basis for procedural operations used to create 3D objects by extrusion or sweeping. For this reason, the evaluation of a procedural operation will always need a reference to the sketch that this operation is based on.

To handle this interlinkage of the two graphs, three different possibilities supported by the used graph rewriting tool GrGen.NET (Jakumeit et al., 2010) were considered. In the case of an *Integration by reference*, a graph entity representing a sketch would only store a reference to an independent graph which represents the actual sketch (described by its geometry and constraints). The second possibility is an *Integration by using subgraphs*, where a sketch graph is stored inside a graph entity representing a sketch, without connecting it to the procedural graph. Last, an *Integration by combination of the graphs* was considered. In this case each graph that represents a sketch is an actual subgraph of the procedural graph and thereby a part of it.

While each of these solutions is generally possible, the third option proved to be the most advantageous. This is caused by the requirements on the necessary relations between the two graphs. The first two possibilities induce a conceptual separation of the sketch graphs from the procedural graph. This increases the complexity of rewrite rules that need to transform a sketch graph as well as the procedural graph, but does not yield any benefits.

When using the third option on the other hand, relationships between sketch graphs can be modeled much more straight-forward. The combination of the graphs is realized by connecting all nodes belonging to a sketch graph with the respective sketch node of the procedural graph, thereby turning it into a subgraph of the procedural graph. The application of this concept is

shown in Figure 2 exemplarily for a graph representing a LoD 2 tunnel model. Here the edges *$9* and *$A* are used to assign the nodes *ProjectedPoint* and *Circle* to the *Sketch* node and thereby create an integrated graph. The edge *$8:project* is introduced for defining the location of the *ProjectedPoint* to be derived from the alignment. This information would otherwise have to be kept as an attribute of the *ProjectedPoint* node.

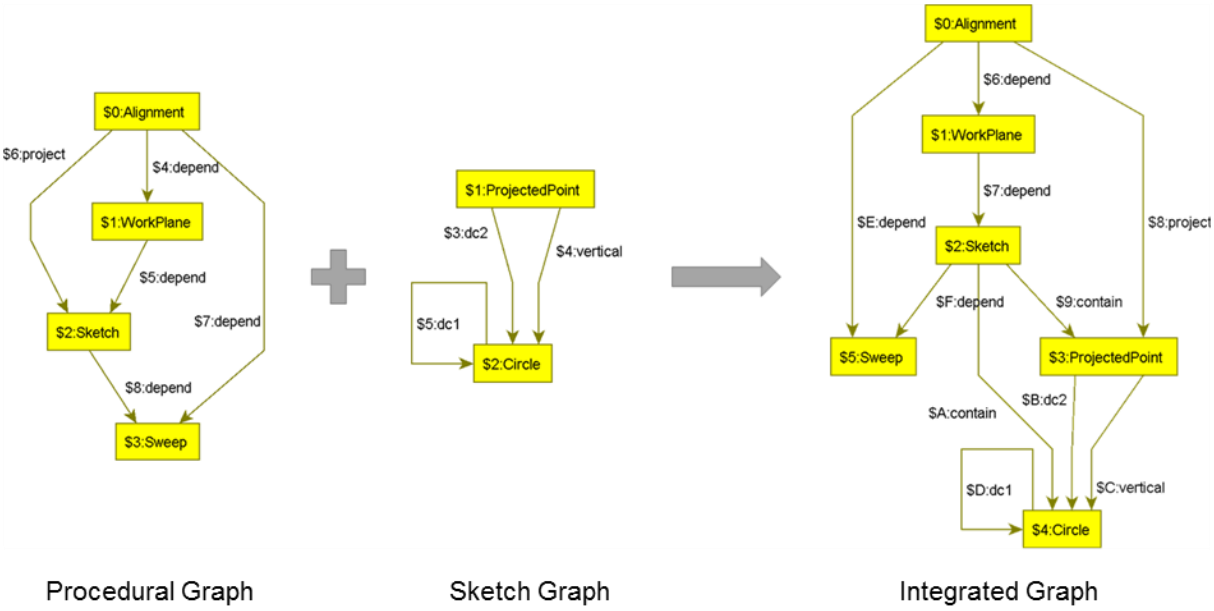

Figure 2: Integration of sketch and procedural graph.

The integration of the graphs is not defined as an actual graph transformation. In fact, it is the conceptual basis for the definition of all graph rewrite rules, as they should only create and transform an integrated graph.

Although only the integrated graph is used for the representation of a complete model, the terms sketch graph and procedural graph will continue to be used in order to indicate which part of the graph is referred to in a particular context.

3.3 Formalizing Modeling Operations as Graph Rewrite Rules

Graph rewrite rules are used to create different instances of the graph that represents the product model based on the graph entities available from the metamodel. The more abstract task of a rewrite rule is to combine a set of modeling operations (procedural or sketch related) that would otherwise have to be carried out manually. Thus, to formally define a rule, it needs to be determined which model parts are to be used or altered (e.g. the creation of a sketch requires a work plane that the sketch is drawn upon). Thereby, the pattern part of the rule is constructed out of the nodes and edges that represent the respective model entities. Next, the designated result of the represented modeling operation needs to be formalized in the rewrite part of the rule by adding, altering or removing new or existing nodes and edges. Additionally, we need to ensure that the application of any rule on the graph-based representation of a model will lead to another valid representation. This means that the representation is interpretable and can be used to successfully create an evaluated model.

The execution of the rules is to be initiated by the end user. To this end, he can choose among a set of predefined rules that are integrated in the graph rewrite system. Currently the definition of rewrite rules by the user is not intended. This is caused by the fact that the creation of a rule requires deep insight in parametric and procedural modeling as well as in the process of graph rewriting and should remain hidden from the end user. He should rather be only concerned with

the execution of those rules that lead to the model geometry he requires - which is naturally limited by the set of the predefined rewrite rules.

3.4 Requirements on the Graph to Allow Successful Interpretation

The main requirement on any graph describing a certain model is its validity, insofar as that the interpretation of the graph must be possible (i.e. without conflicts and inconsistencies) and result in a usable procedural geometry model. Therefore, the graph must always represent the result of a modeling process which could also have been performed manually. Procedural modeling applications support this manual process by preventing user actions that would destroy the procedural or parametric structure of the model. For example, they do not to allow the deletion of a sketch while keeping a dependent extrusion. As this is not a-priori assured by automated graph generation, special care has to be taken regarding the consistency of the graph.

In the context of the procedural graph this means, that every node representing a procedural operation has to have all necessary preceding operations (or input parameters) present in the graph in form of the respective nodes connected by proper edges. For example, a sketch node *S* always needs a work plane *WP* node connected by an incoming *depend* edge *d*: *d=(WP, S)*.

Additionally, a subgraph representing the 2D geometry of a sketch must always describe a fully constrained sketch. Otherwise the interpretation of the graph would not lead to an unambiguous solution. The aspects of sketch graphs in this regard are discussed in detail in Vilgertshofer and Borrmann (2015). There, the structure of sketch graphs is defined and the development of and requirements on rewrite rules, which always produce an interpretable graph are shown. The main tools for achieving an unambiguous representation shown there are the definition of ports (definition of the part of a geometric element that a constraint applies to) and the use of temporary coordinates to support the GCS.

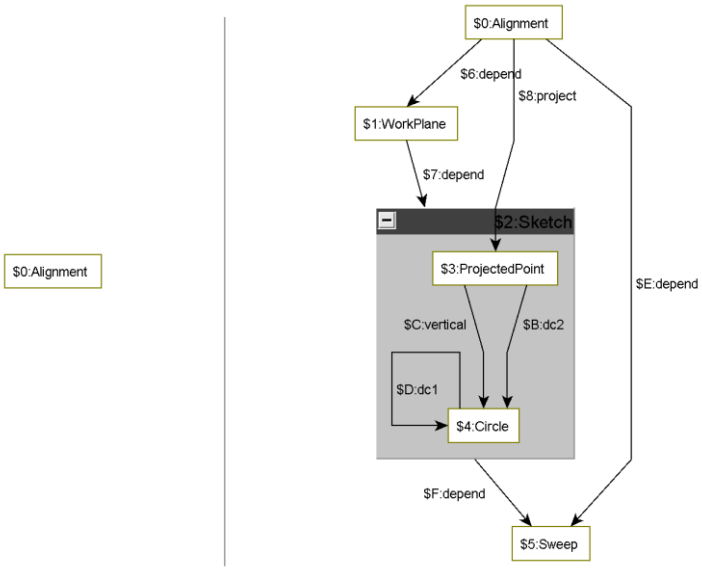

Figure 3: Pattern (left) and replacement (right) part of a rewrite rule that details a model from LoD 1 to LoD 2.

4. Prototypical Implementation

The developed concept of using graph transformation to automatically create consistent multi-scale product models has been implemented as a case study. For the definition of a graph rewrite system consisting of a metamodel and appropriate graph rewrite rules, the graph rewrite generator GrGen.NET has been used while the generation of the evaluated sketch is performed with the commercial parametric CAD application Autodesk Inventor.

4.1 A Graph Rewrite System for the Creation of a Shield Tunnel

GRGEN.NET (Graph Rewrite GENerator) is an open source software development tool that provides programming languages optimized for graph structured data. More concretely, it provides the possibility to create a graph metamodel and respective graph rewrite rules implementing (as default) an SPO-based approach (Blomer et al., 2014).

As a proof-of-concept we implemented a graph rewrite system with GRGEN.NET. It allows us to create the graph based representation of the shield tunnel model stepwise up to LoD 4 without any manual modeling operations. We defined rather comprehensive rewrite rules for this first evaluation of the approach as our primary focus was to proof that a graph created by these rules could actually be used for the creation of an evaluated model.

An overview of our graph metamodel is given in Table 1. It shows the different types of the nodes and edges that we use as well as whether they conceptually belong to a sketch graph or to the procedural graph. In Figure 3 an exemplary rewrite rule consisting of the pattern and the replacement graph is depicted. A graph representing a LoD 3 model is shown in Figures 4.

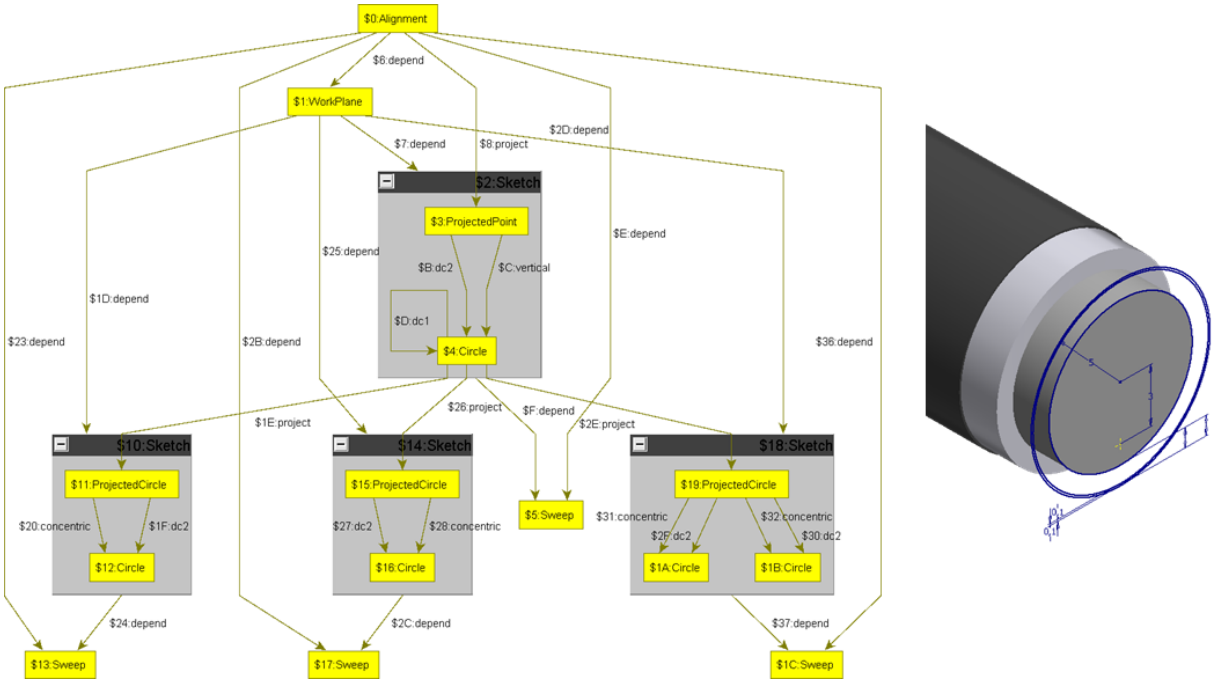

Figure 4: Graph representing the depicted LoD 3 model
(the sketch graphs have been grouped to improve readability)

Table 1: The Metamodel of the Developed Graph Rewriting System

	sketch graph	**procedural graph**
nodes	V_S: – point / projected point – line / projected line – circle / projected circle – arc / projected arc *projected geometric elements are used to create relationships between different sketches*	V_P: – alignment – sketch – work plane – sweep
edges	E_S: dimensional constraints: – dimension of one element (dc1) *(e.g. radius, length)* – dimension between two elements (dc2) *(e.g. distance, offset)* geometric constraints: – concentric – equal – horizontal – vertical – parallel – perpendicular – collinear – coincident	E_P: – contain *relates geometric elements to a certain sketch node* – depend *defines procedural dependencies* – project *defines the source geometry of a projected element*

4.2 Application of Rewrite Rules and Generation of the Evaluated Model

While a graph is used for the representation and alteration or detailing of the model, it is not particularly useful for engineering purposes. Therefore, an actual three-dimensional model, the *evaluated model*, needs to be generated from the graph-based representation for display and further use in a parametrical modeling system. A software tool to enable this generation has been prototypically developed. It combines the functionalities to create and transform the graph with the predefined metamodel and rewrite rules as well as the evaluation of the graph-based representation. Thereby, it generates the geometry and thus the evaluated model in the commercial parametrical modeling system Autodesk Inventor. A short overview of the functionality follows.

The developed program uses the API of the graph rewrite tool GrGen.Net to access the metamodel and rewrite rules predefined in the syntax of this tool. While the user triggers the execution of a rule the actual rewrite operation is performed by GrGen.Net. After the consecutive execution of any number of desired rules the current state of the graph is used to create the evaluated model. Therefore, the nodes and edges are interpreted and their equivalent objects are sequentially created by calling the respective methods of the API of Autodesk Inventor. During this process, the connections and relationships defined by the graph are used to determine the correct order of the construction operations and the necessary dependencies of the objects to be created within Inventor.

5. Conclusion

This paper presents a concept for the graph-based representation of product models and their automatic detailing by performing graph transformation operations based on formal rules defined in a graph rewriting system. It focuses on product models of shield-tunnels as the

automated creation of multi-scale versions of such models is the overall purpose of this research.

The main contribution is the description of the development of a graph rewriting system that enables the generation of graphs representing those product models. We give an overview of the main requirements on the graph in terms of its ability to be evaluated in order to create the actual 3D procedural model in a parametric CAD application. We further described how we improved our approach by combining the graphs representing 2D sketches and the procedural operations by integrating them. Using only one graph to represent the complete procedural geometry of the product model allows a much better handling of the graph and makes the definition of rewrite rules and the interpretation of the graph much easier. To prove the feasibility of our approach, the graph rewriting system has been implemented in the graph rewriting tool GRGEN.NET. Additionally, a software prototype was developed that enables to automatically create an evaluated model in the parametric modeling system Autodesk Inventor.

Further research is focusing on creating a larger set of rewrite rules, which enables end users to create more diversified models. Additionally, we will further extend the graph to realize the affiliation of the created objects to a certain LoD.

Acknowledgements

We gratefully acknowledge the support of the German Research Foundation (DFG) for funding the project under grant FOR 1546.

References

Blomer, J., Geiß, R., Jakumeit, E., 2014. The GrGen.NET User Manual.

Borrmann, A., Flurl, M., Ramos, J., Mundani, R., Rank, E., 2014a. Synchronous collaborative tunnel design based on consistency-preserving multi-scale models. Advanced Engineering Informatics 28, 499–517.

Borrmann, A., Ji, Y., Jubierre, J.R., Flurl, M., 2012. Procedural Modeling: A new approach to multi-scale design in infrastructure projects. In: Proc. of the EG-ICE Workshop on Intelligent Computing in Civil Engineering. Herrsching, Germany.

Borrmann, A., Jubierre, J.R., 2013. A multi-scale tunnel product model providing coherent geometry and semantics. In: Proc. of the 2013 ASCE International Workshop on Computing in Civil Engineering. Los Angeles, pp. 291–298.

Borrmann, A., Kolbe, T.H., Donaubauer, A., Steuer, H., Jubierre, J.R., Flurl, M., 2014b. Multi-Scale Geometric-Semantic Modeling of Shield Tunnels for GIS and BIM Applications. Computer-Aided Civil and Infrastructure Engineering 30, 263–281.

Fudos, I., Hoffmann, C.M., 1997. A Graph-constructive Approach to Solving Systems of Geometric Constraints. ACM Transactions on Graphics 16, 179–216.

Heckel, R., 2006. Graph Transformation in a Nutshell. Electronic Notes in Theoretical Computer Science 148, 187–198.

Jakumeit, E., Buchwald, S., Kroll, M., 2010. GrGen.NET. International Journal on Software Tools for Technology Transfer.

Ji, Y., Borrmann, A., Beetz, J., Obergrießer, M., 2013. Exchange of Parametric Bridge Models Using a Neutral Data Format. Journal of Computing in Civil Engineering 27, 593–606.

Kolbe, T.H., 2009. Representing and Exchanging 3D City Models with CityGML. In: Lee, J., Zlatanova, S. (Eds.), Proceedings of the 3rd International Workshop on 3D Geo-Information, Seoul, Korea, Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg.

Mun, D., Han, S., Kim, J., Oh, Y., 2003. A set of standard modeling commands for the history-based parametric approach. Computer-Aided Design 35, 1171–1179.

Owen, J.C., 1991. Algebraic solution for geometry from dimensional constraints. In: Proceedings of the First ACM

Symposium on Solid Modeling Foundations and CAD/CAM Applications. ACM, pp. 397–407.

Rozenberg, G., 1997. Handbook of Graph Grammars aand Computing by Graph Transformation. World Scientific.

Schultz, C., Bhatt, M., Borrmann, A., 2015. Bridging qualitative spatial constraints and feature-based parametric modelling: Expressing visibility and movement constraints. Advanced Engineering Informatics.

Shah, J.J., Mäntylä, M., 1995. Parametric and feature-based CAD/CAM: Concepts, Techniques and Applications. ohn Wiley & Sons, New York.

Vilgertshofer, S., Borrmann, A., 2015. Automatic Detailing of Parametric Sketches by Graph Transformation. In: Proc. of the 32nd ISARC 2015. Oulu, Finnland.

Yabuki, N., Aruga, T., Furuya, H., 2013. Development and application of a product model for shield tunnels. In: Proceedings of the 30th ISARC. Montréal.