Advanced Engineering Informatics 33 (2017) 502–515

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Full length article

Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models

Simon Vilgertshofer *, André Borrmann

Chair of Computational Modeling and Simulation, Leonhard Obermeyer Center, Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany

ARTICLE INFO

Article history:
Received 3 November 2016
Received in revised form 8 June 2017
Accepted 26 July 2017
Available online 8 August 2017

Keywords:
Multi-scale modeling
Graph rewriting
Parametric modeling

ABSTRACT

For the design of large infrastructure projects such as inner-city subway tracks, it proves necessary to consider differing model scales, ranging from the scale of several kilometers down to a few millimeters. This challenge 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 procedural and parametric modeling techniques while creating the model. This results in a flexible multi-scale model that can be easily modified on one scale while other scales are automatically updated. However, the correct application of parametric constraints and procedural dependencies has shown to be a very complex and time-consuming process. To address this issue, this papers presents a semi-automated detailing mechanism, which is based on formal procedures based on graphs and graph transformations. This paper discusses how procedural parametric models based on two-dimensional sketches can be represented by graphs and how detailing steps in the form of parametric modeling operations can be formalized by using rule-based graph rewriting.

1. Introduction

The ongoing digitization of planning processes in the building sector has become a major improvement for the AEC industry and is being expanded into the infrastructure domain [10]. In the planning of large infrastructure facilities ranging over several kilometers, such as tunnels or roads, new requirements on the underlying models have become evident. These include the capability to represent substantially diverging scales as well as the applicability of the models to different or varying circumstances [9]. To decisively assist designers and engineers, those digital models need to support the efficient and consistent modeling and management of geometric objects on diverging 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) [28]. Previous research extends the LoD concept used in the GIS domain towards multi-scale representations of building information models, particularly used for the modeling of shield tunnels [9]. The main feature of the proposed concept is the preservation of the consistency across the different LoDs by applying parametric design techniques in order to define relationships and dependencies between geometric entities on different LoDs. However, one conclusion of this research is that the manual definition of the required parametric dependencies is a very complex, time-consuming and error-prone task, which could strongly benefit from automation mechanisms [6].

To address this challenge, this paper presents an approach to realize these automation mechanisms. It describes an automated detailing approach, which is based on the formal use of graphs and graph rewriting mechanisms. It further shows how procedural parametric geometric 3D-models based on two-dimensional sketches can be represented by graphs and how refinement steps can be realized through rule-based rewriting of these graphs.

The prospected benefit of this approach for end users is the reduction of the effort to manually define the consistency preservation dependencies. Doing so, it allows them to focus on the creative, conceptual and engineering aspects of the design process instead of time-consuming and repetitive modeling operations. Single-scale models can benefit likewise from the application of parametric modeling, since parameterized models can be easily adapted to different use-case scenarios and boundary conditions. The formal computer-interpretable description of a parametric design preserves the knowledge embodied in the respective manual tasks to build-up the model and allows their reuse in similar design scenarios. Furthermore, the modeling knowledge formal-

* Corresponding author.
E-mail address: simon.vilgertshofer@tum.de (S. Vilgertshofer).

http://dx.doi.org/10.1016/j.aei.2017.07.003
1474-0346/

ized in graph rewrite rules may be applied to new scenarios with only minor alterations or even be generalized in a way that the rules themselves are intrinsically adaptable to different scenarios. Accordingly, we see the proposed approach as a contribution to the fields of knowledge-based engineering [43,31,1] and knowledge-driven design synthesis [35,26,46]. This proposed methodology is demonstrated by applying it to the modeling process of a shield tunnel. The model is detailed step-wise (as shown in Fig. 1) from the basic layout of the alignment up to a LoD containing several space objects of the tunnel as described in Borrmann et al. [9]. Based on the general introduction of a graph rewriting system designed to represent and manipulate elementary parametric and procedural modeling operations as well as the corresponding geometric objects, a set of exemplary rules describing the detailing process of the tunnel model is presented as proof of concept. In addition, the process of interpreting the graph for creating the *evaluated model* in a commercial parametric modeling system is described. A software prototype was developed to prove the feasibility of the developed approach.

The paper is structured as follows: In Section 2 related works in the scope of design support and automation as well as the theoretical background of the research at hand 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. The Section further describes a prototypical software tool, which is able to interpret the graph-based representation and create an actual 3D model. In Section 5, the general applicability of the approach is discussed in terms of its prospects and limitations at the current state of development. The paper ends with a conclusion discussing the development of our research, the major findings including known limitations and its possible generalization as well as topics for future research.

2. Related work

Benefits of the computer-aided generation of designs or models have been addressed by researchers in various aspects. This section gives a short overview of existing approaches and puts the presented approach in their context. It further presents the theoretical background of the proposed methodology in terms of parametric and procedural modeling, multi-scale modeling and graph rewriting.

2.1. Computer-aided model and design synthesis

Computers have been successfully used to support, accelerate and simplify the process of generating technical drawings and product models. CAD software is widely used and enormously valuable in the building sector and in mechanical engineering [11]. Its main purpose is to assist an engineer in his creative design work by resolving the disadvantages of paper based drawing, though. Design, however, is one of the most complex human tasks, as it requires the consideration of various constraints and conditions to obtain a satisfactory solution [3]. Therefore, a further step is the development of methods and tools, which actively support a designer by automatically generating whole sets of design variants or by the automation of repetitive and trivial tasks in the design process. As the concept presented in this paper contributes to this field of research, similar approaches, which also utilize graph representations, are discussed here.

In the field of Computational Design Synthesis (CDS), Helms [20] uses a graph grammar for the computational synthesis of product architectures. Design knowledge is captured in a port-based metamodel and the procedural design rules of the grammar. The dissertation and corresponding publications [21,20] show how the computational synthesis of a design solution space for automotive hybrid powertrains and for the generation of aircraft cabin layouts can be realized. Hoisl [23] presents an approach for creating a general spatial grammar system that introduces interactive definition and application of grammar rules in the scope of CDS. It aims at actively supporting a designer in the modeling process using mechanical CAD systems. The approach by Kniemeyer [27] in the domain of biology makes use of a graph grammar to design and implement a language to support the functional-structural modeling of plants.

Knowledge-Based Engineering targets the formalization of engineering knowledge to assist or automate routine design tasks, which are repetitive and time-consuming. Main contributions have been made by Cooper and La Rocca [13] as well as Stokes [42]. A comprehensive literature and research review that analyses 50 contributions is presented in Verhagen et al. [43]. With regard to the use of graphs, Chein et al. [12] introduced a knowledge representation and reasoning language based on conceptual graphs.

A method for using graphs in order to represent the shape and dimension of design variants was presented by Borkowski and Grabska [5]. Here, the conceptual design of bridges is used as an

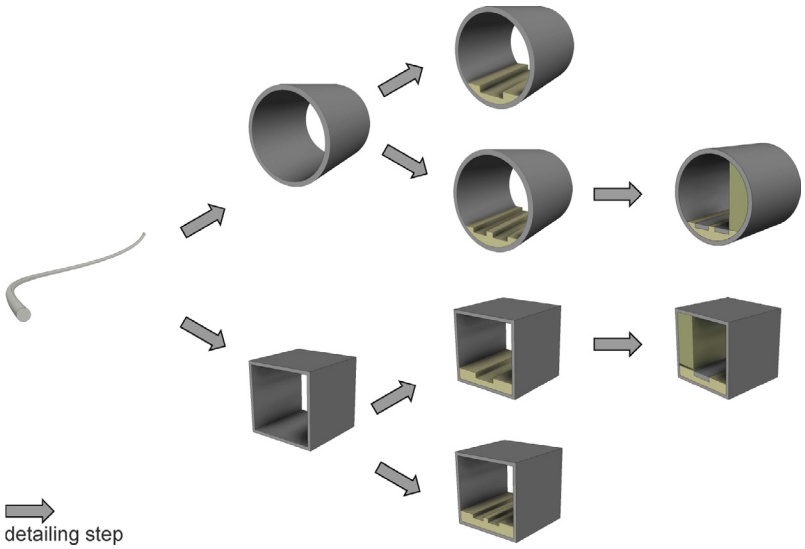

Fig. 1. Conceptual illustration of several detailing steps in a tunnel planing process. Dependent on the chosen rule different designs can be generated.

example for showing how different design alternatives can be generated by computationally manipulating the graph based representations. Grabska et al. [17] use hierarchical hypergraphs to represent spatial design solutions and formalize design knowledge. A related approach to support the conceptual building design phase was developed by Kraft and Nagl [30]. It is implemented in form of a CAD tool for conceptual design and a knowledge specification that formalizes various design rules in a graph-based form. In Langenhan et al. [32] the topology of spatial floor configurations is indexed by using a graph-based representation. The method supports the iterative process of searching for plausible solutions in early design phases by referring to preexisting examples. Also shape grammars have been used for architectural designs to automatically generate large varieties of two-dimensional layout schemes [38].

The approach presented in this paper as well as most of the methods listed above use the advantages of graph based representations and graph rewriting, but these methods aim at assisting a designer by generating sets of possible design variants, from which he can choose. The key difference of the approach presented here is that it does not focus on this automatic generation of multiple design variants. Instead, this paper targets an interactive process of semi-automated detailing of designs and the generation of consistent multi-scale models. More precisely, the designer can choose from a set of adaptable rules, while refining a model. The application of a rule creates a more detailed version of the model at hand by applying formalized parametric modeling knowledge. For this reason, graph rewriting techniques and explicitly not a graph grammar is used.

2.2. 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 [29]. The general concept is to provide different geometric representations of a spatial object. Which of these representations is actually used to display the object, depends on the particular context, for example in regard of the size of the model extent, the available hardware and the user's requirements. One of the most important examples is CityGML, an XML-based data model for the representation of mainly static 3D city models [28] which defines five different LoDs. In most cases, representations on coarser LoDs are generated from finer ones by a computational process called abstraction [40]. In contrast, the design process of construction projects starts with a very rough representation that gets gradually refined and more detailed as planning evolves [9]. Additionally, CityGML does not include any automated consistency preservation mechanisms to ensure that changes in one LoD are propagated to the other LoDs, as an object's geometry stored one LoD is completely independent from that on any other LoD. Nevertheless, a method to ensure the consistency across the LoDs was presented by Gröger and Plümer [18].

In order to fulfill the needs of consistency-preservation and multi-LoD representation in the infrastructure domain, a multi-scale product model for shield tunnels has been developed by Borrmann and Jubierre [8], which is based on a single scale model introduced by Yabuki et al. [47]. This multi-scale model uses procedural geometry description and parametric modeling techniques to provide mechanisms for automated consistency preservation across the different LoDs. A 3D representation of the LoDs that were defined for the tunnel product model is depicted in Fig. 2 without the first LoD that encompasses solely the alignment.

2.3. Parametric and procedural modeling

The concept of parametric modeling was developed in the 1990s Shah and Mäntylä [41] and is by now well established and used in many commercial and open source CAD applications such as Autodesk Inventor, Siemens NX and FreeCAD. While applied primarily in mechanical engineering, the concept is also increasingly used to create easily adaptable models of infrastructure facilities [25].

Parametric 2D models (sketches) are composed of geometric objects and parametric constraints. In the course of creating a sketch in a parametric CAD application, a system of geometry objects and geometric-topological constraints is defined. It forms a constraint problem which can be solved by a geometric constraint solver (GCS) Fudos and Hoffmann [15]; Owen [36]. Using this technique, the designer may define particular dimensions (positions, heights, widths) by using variables instead of fixed numerical values and is thereby able to quickly alter a design or to explore different variants by adjusting the valuables. The set of parametric constraints that is implemented by all major constraint solvers is defined as the standard geometric constraint language [39]. 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, which is called the construction history of the model. 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 [7,34]. To define constraints between elements in different sketches, so called projected geometry is used. This means that an source element from one sketch is projected into a second sketch, where a corresponding dependent element is created. If the source element is then changed, the dependent element is altered accordingly.

The presented approach uses parametric 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. [6]. Thus, the consistency of the model across multiple LoDs can be preserved.

2.4. Graph rewriting

The proposed concept for automating the detailing process and formalizing modeling operations relies on graph theory and graph rewriting. The underlying theory is extensively discussed in Rozenberg [37]. 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 solely parametric sketches has been presented in Vilgertshofer and Borrmann [44].

Generally, graphs are well suited to describe the relationships between different elements or entities. Graph nodes represent the elements, while the graph edges represent the relationships between these elements. In the presented approach the graph nodes are used to describe the geometric elements and the procedural modeling operations, while the edges represent parametric constraints and procedural dependencies. A graph consisting of these nodes and edges then represents a parametric procedural model.

Formally, the graph used here is a *labeled, typed, attributed, directed multigraph with loops*. This allows the definition of different types of node and edges, that in turn posses individual sets of attributes.

Graph rewriting operations are used to create a new graph out of an existing graph by altering, deleting or replacing parts (sub-

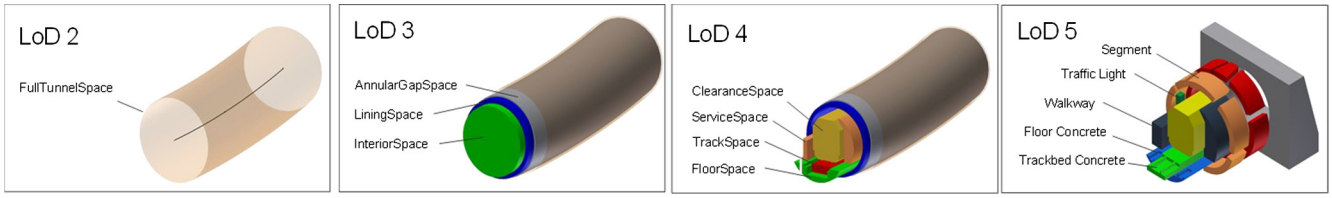

Fig. 2. A 3D representation of the LoDs 2–5 of the multi-scale shield tunnel product model [8].

graphs) of the existing graph. The changes are formalized through graph rewrite rules written as $p : L \xrightarrow{r} R$. A graph rewrite rule is defined by a pattern graph L and a replacement graph (also called rewrite graph) R. The left-hand side L defines the pre-conditions, while the right-hand side R describes the post-conditions of a rule. When a rule is applied to a graph (called the host graph H), this graph is searched to find a subgraph that matches the graph pattern L, more formally this match is an isomorphism of L to a subgraph found in H. If the matching succeeds L is replaced with R under the consideration of a preservation morphism r that determines, how an instance of L in the host graph is replaced or altered by R Heckel [19]. The outcome of this rule application is called the result graph H' as illustrated in Fig. 3.

There are several different approaches to graph rewriting. The examples given above characterizes the Single-Pushout Approach (SPO). Further methods are node replacement, hyperedge replacement or the Double-Pushout Approach (DPO). Productions of those two approaches are basically the same except for representation differences. A SPO production is a partial graph morphism $L \xrightarrow{p} R$ while a DPO production is a span of total injective graph morphisms $L \xleftarrow{l} K \xrightarrow{r} R$ [37].

The usage of formal graph representation and graph rewriting techniques allows to capture design knowledge, or more precisely detailing knowledge, in a persistent and CAD-system independent manner. This provides the design and engineering companies for which this approach is being developed the opportunity to safely maintain their valuable engineering know-how and make it available for the entire company staff and over long periods of time.

For the proof-of-concept of the proposed research approach the general-purpose graph rewriting system GrGen.NET was employed as part of the authors' software prototype [16]. GrGen.NET uses an SPO-based approach and understands graph rewriting as a method for *"declaratively specifying "changes" to a graph"* [4].

3. Formalization of modeling operations in a graph rewriting system

Parametric modeling systems use geometric elements such as points or lines as primitive planar entities. The topology of these entities is defined by parametric constraints as described in Section 2.3. Sets of geometric elements and corresponding parametric constraints define sketches that are the basis for further procedural modeling operations, which create spatial objects. To represent a procedural parametric model by means of a graph, it is necessary to define, which types of geometric elements, parametric constraints and procedural modeling operations may be used. In the scope of this research, the following types are generally considered:

- geometric elements: *point, line, spline circle, arc*
- parametric constraints
  - geometric constraints: *coincident, collinear, equal, concentric, horizontal, vertical, parallel, perpendicular, fixed*
  - dimensional constraints: *dimensions of one geometric element, distances between two geometric elements*
- procedural modeling operations: *workplane, extrusion, sweep*

This selection closely reflects the standard geometric constraint language defined in Schultz et al. [39] comprising the most common operations provided by any parametric CAD system.

To represent theses items, corresponding attributed types of graph nodes and edges need to be introduced. They are formally described in a *graph metamodel* and can then be instantiated during the generation of a graph. The metamodel also forms the basis for the definition of graph rewrite rules. These rules formally describe detailing steps that an end user can apply instead of manually executing the underlying procedural or parametric modeling operations.

Fig. 4 conceptually illustrates how a model is represented by a graph and how this graph is transformed by applying predefined rewrite rules. The application of a rewrite rule is indicated by an arrow marked *rule*. At each stage of this process, the graph can then be interpreted and processed by a parametric CAD system to create an editable parametric model. This model is further called the *evaluated model*. The generation of the *evaluated model* out of the graph is illustrated by an arrow marked *evaluation*.

Note, that the graphs in Fig. 4 do not actually represent the respective geometry. They merely illustrate the presented concept. Concrete examples of graph instances that represent geometry and corresponding definitions of rewrite rules are given in Section 3.3.

In the following Subsections, the metamodel is formally described and some exemplary graph-based representations with the corresponding models are presented. Furthermore, the definition of the rewrite rules is explained in general and also illustrated by an example. Additionally, the process of developing the graph metamodel and the requirements on the graph-based representation to be unambiguously translatable into an actual geometric 3D model is discussed.

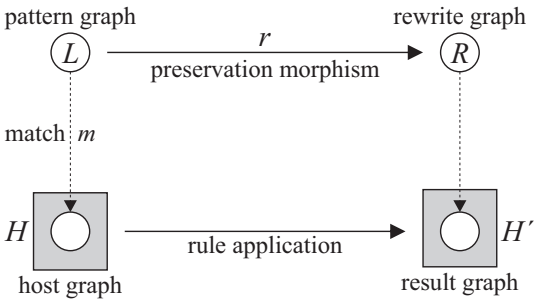

Fig. 3. Graph rewriting via the Single-Pushout approach (inspired by Blomer et al. [4]).

3.1. Definition of the graph and the 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.

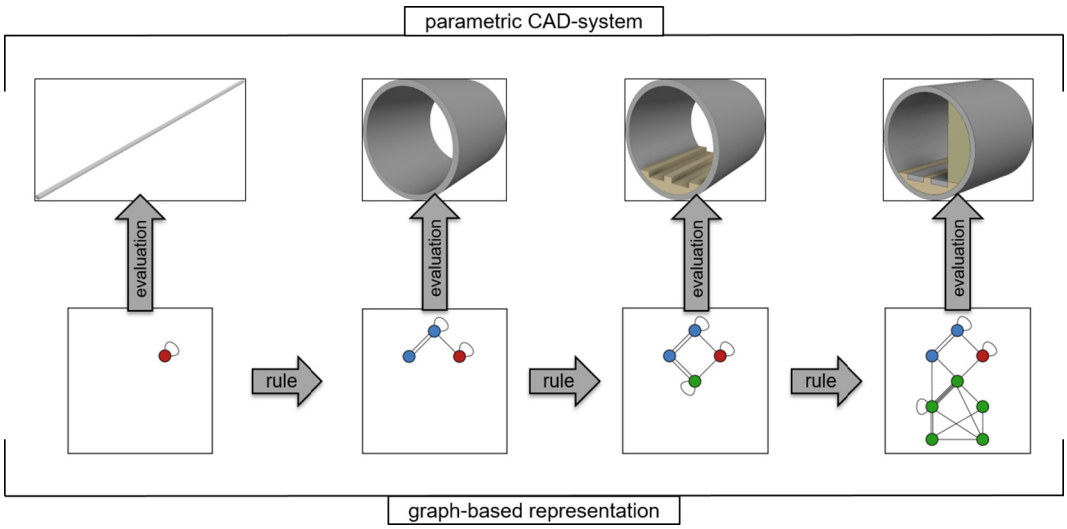

Fig. 4. Transforming the graph by using rewrite rules results in a more detailed model.

The graph representing a procedural geometry model is a directed multigraph with loops $G = (V, E, T^v, T^e, s, t, lb, ty^v, ty^e, att)$. It is defined as follows:

- $V = V_P \cup V_S$ is a nonempty finite set of vertices. Elements of V_P are vertices that represent procedural modeling operations, while elements of V_S represent geometric objects in a sketch.
- $E = E_P \cup E_S$ a is a nonempty finite set of edges. Elements of E_P are used to represent general relations or dependencies between the procedural operations and allocate geometric elements to a specific sketch. Elements of E_S represent parametric constraints of a geometric element or between two geometric elements.
- $V_P \cap V_S = \varnothing$.
- $E_P \cap E_S = \varnothing$.
- $s : E \to V$ is a mapping that indicates the source node of all edges.
- $t : E \to V$ is a mapping that indicates the target node of all edges.
- $\sum$ is an alphabet of labels of vertices and edges.
- $lb : E \cup V \to \sum$ is a labeling function.
- $T^v = T^v_P \cup T^v_S$ is a set of types for the vertices in V. T^v_P and T^v_S are sets of types for nodes in V_P and V_S respectively.
- $T^e = T^e_P \cup T^e_S$ is a set of types for the edges. T^e_P and T^e_S are sets of types for nodes in E_P and E_S respectively.
- $ty^v : V \to T^v$ is a typing function for the vertices, such that $ty^v(V_P) \cap ty^v(V_S) = \varnothing$.
- $ty^e : E \to T^e$ is a typing function for the edges, such that $ty^e(E_P) \cap ty^e(E_S) = \varnothing$.
- At is a set of attributes of vertices and edges.
- $att : E \cup V \to At$ is an attributing function.

The metamodel describes the possible set of types T^v and T^e of the graph entities V and E that can be instantiated when a rewrite rule is executed to create or alter a graph. Besides defining the available types, the metamodel furthermore defines the attributes of a certain type as well as conditions that determine which nodes and edges may be incident or which node types can be adjacent. As the type of a graph entity clearly determines which attributes that entity has, an attributing function is not given.

The metamodel of the graph is defined in an object-oriented manner that allows the inheritance of attributes. Fig. 5 gives an overview of the node and edge types defined in the metamodel.

3.2. Graph-based representations

By instantiating the metamodel, it is possible to define a graph that represents a specific model or more precisely the modeling process from which this model results. As a first basic example, a sketch depicting a tunnel cross-section in LoD 2 is given. Fig. 6 shows the graph as well as the corresponding model. Furthermore the necessary manual modeling operations that a user would have to execute in order to create this model are described: First, a *Workplane*, which is in the *xy*-plane is constructed. Next, a new empty 2D-*Sketch* that lies in this *Workplane* is created. In this *Sketch* a *Point* is drawn and fixed to its position by applying a *Fixed*-constraint. Additionally, a circle is drawn whose radius is defined by a *Dimensional Constraint (dc1)*. Finally, the center point of the *Circle* is constrained to be *Coincident* with the previously created *Point*.

Note, that the nodes that conceptually belong to the sketch are grouped in the *Sketch* node for readability. Formally, the nodes *Point* and *Circle* are respectively connected to the *Sketch* node by a *depend* edge with the *Sketch* node as its source. This is further explained in SubSection 3.4.

The second example (Fig. 7) shows how a 3D model can be defined based on the representation of the 2D sketch in the first example. Therefore, an alignment (defined by a spline) is added, so that the original cross-section can be sweeped along an alignment defined in the second sketch. Furthermore, the workplane that the sketch is drawn on now depends on the alignment. Thereby, the model is fully constrained, as it can only be edited by changing parametric values.

A further extension of this example scenario is the detailing of the cross-section to part of a LoD 3 model, which is shown in Fig. 8. Therefore, a new sketch needs to be created on the existing workplane to define a ring profile, which is then also sweeped along the alignment. The outer bound of this ring is created by a projection of the circle in the existing sketch. Thereby the consistency between LoD 2 and 3 is ensured. Note, that the graph now contains the model in the LoDs 1 and 2 as well as part of the 3rd LoD. For clarity reasons, the model on the right side of Fig. 8 only shows this part of LoD 3. LoD 1 and 2 are hidden.

These basic examples were created by executing graph rewrite rules. The presented graphs were then interpreted to generate the *evaluated models* shown here. How those rewrite rules were defined is shown in the next Subsection.

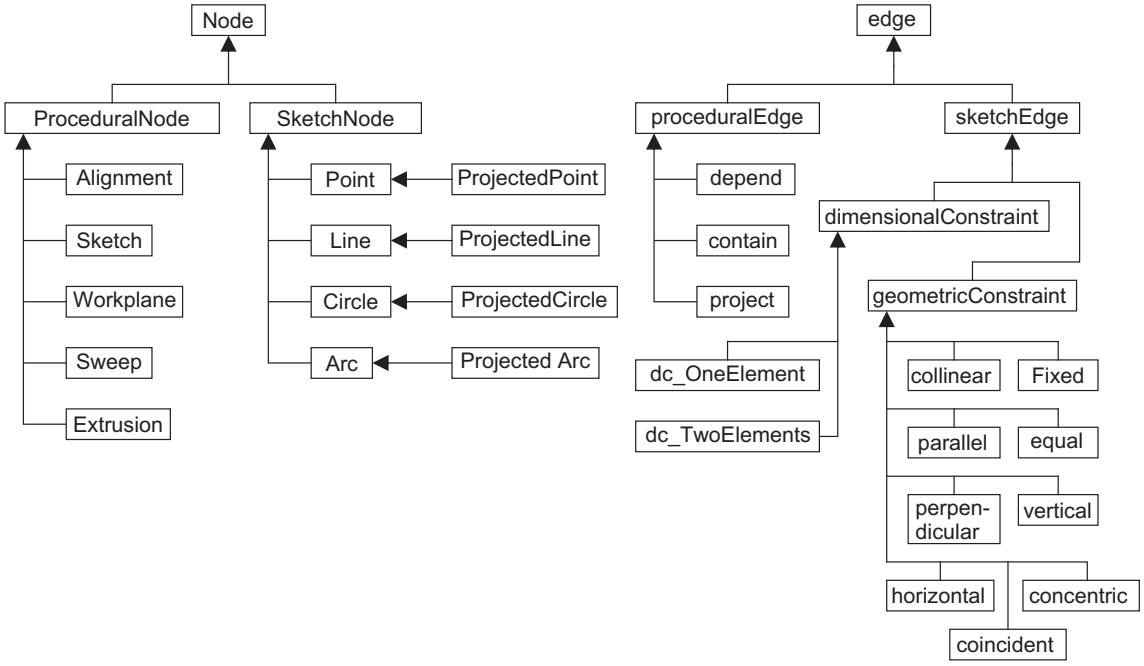

Fig. 5. General structure of the graph metamodel.

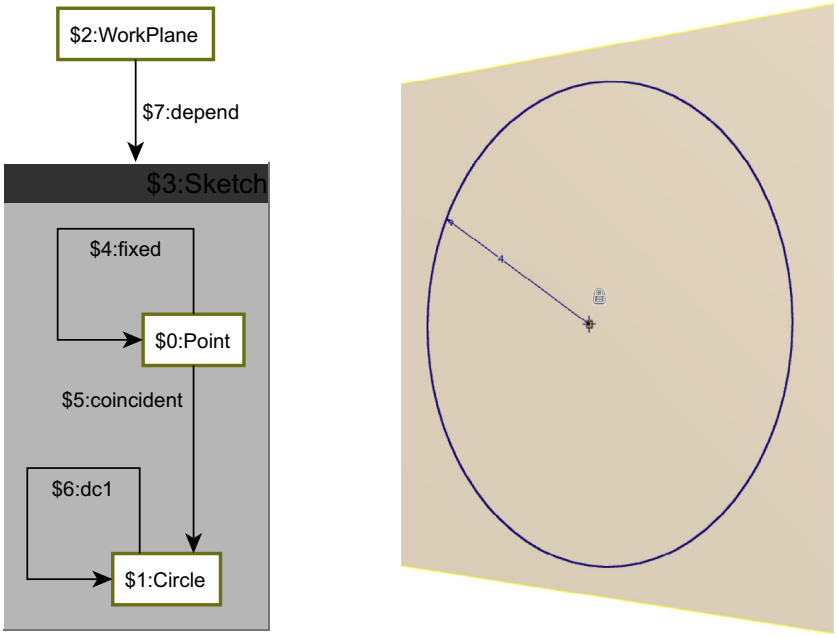

Fig. 6. Graph-based representation of a parametric tunnel cross-section.

3.3. Formalizing modeling operations as graph rewrite rules

With the given definition of the graph and the metamodel, it is now possible to formulate graph rewrite rules that formalize the process of detailing or changing (parts of) the parametric 3D-model. The graph rewrite rules are used to create different instances of the graph that represent the product model based on the graph entity types specified in the metamodel. However, the abstract task of a rewrite rule is to formally represent the execution of modeling operations (procedural or sketch related), so that the rule can be used instead of carrying out the operations manually. Thus, to conceptually define a rule, it needs to be determined, which existing model parts are to be used or altered. For example, the creation of a sketch requires a work plane that the sketch is drawn upon and the creation of a projected geometric element needs a source element. Thereby, the pattern part of the rule is formalized by a subgraph containing the nodes and edges that represent the existing model entities on which the new entities rely upon. 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 result in another valid representation. This means that the graph that is the result of

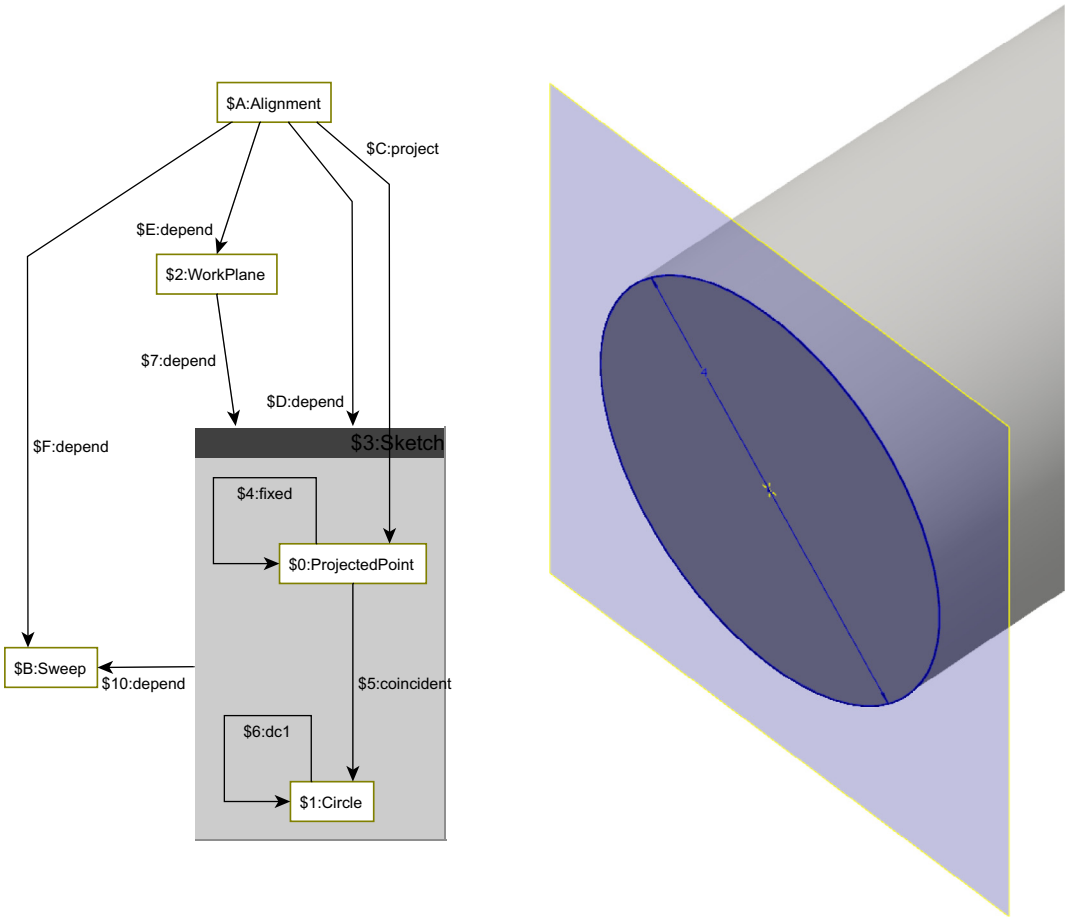

Fig. 7. Graph-based representation of a parametric tunnel model in LoD 1.

a rewrite operation correctly represents the construction history of a parametric procedural model and can be used to successfully create the corresponding evaluated model.

The execution of the rules is to be initiated by the end user. In the proposed concept, he can choose among a set of predefined rules that are part of the graph rewrite system. This practice allows to apply different detailing patterns on a given design and provides the desired flexibility for real-world model construction. However, the definition of rewrite rules by the user is not intended at the moment. This is caused by the fact that the creation of a rule requires 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 the rules that lead to the model geometry he requires. In the proposed concept, the task of creating the graph rewriting rules is rather supposed to be carried out by experts with the necessary modeling knowledge and understanding of graph rewriting. We therefore explicitly hide the process of formally defining the graph rewrite rules from the end user as well as the actual execution of a rewrite rule when it is triggered. The user only gets visual feedback in form of an updated evaluated model while the changes to the underlying graph are executed and stored in the graph rewrite system.

To show how the graph rewrite rules in the proposed graph rewrite system are developed, the following examples are given, which are based on the examples given in the previous subsection.

The first example shows a rule that creates the first example shown in Fig. 6 in an empty graph. As the modeling operations represented by this rule do not have any prerequisites the pattern part of the rule is empty. The rewrite part of the rule on the other hand contains all the elements that are created when the rule is executed. Fig. 9 shows the result of the application as well as the definition of the rule in the corresponding language of GrGen.Net. In this formal definition nodes of the type *WorkPlane*, *Sketch*, *Point* and *Circle* are instantiated. Afterwards the edges connection these nodes are created: An edge representing a *fixed* constraint is connected to the Point *node*. The *coincident* node connects the *Point* and *Circle* node, to make the center of the circle coincident to the fixed point. The edge *dc1* represents the dimensional constraint determining the radius of the circle. As the sketch needs to be drawn on a workplane, the node *depend* assigns the sketch to this specific workplane. The *contain* edges assign the circle and the point to be part of the sketch.

In a second example is it shown how the graph and the model shown in Fig. 7 are changed into those of Fig. 8. Fig. 10 shows the pre-state and post-state of the graph transformed by the rewrite operation. The graph entities defined in the pattern part and in the rewrite part are marked green. Fig. 11 shows the definition of the rule.

3.4. Conceptual development of the graph-based representation

In a preliminary concept introduced in [44], 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

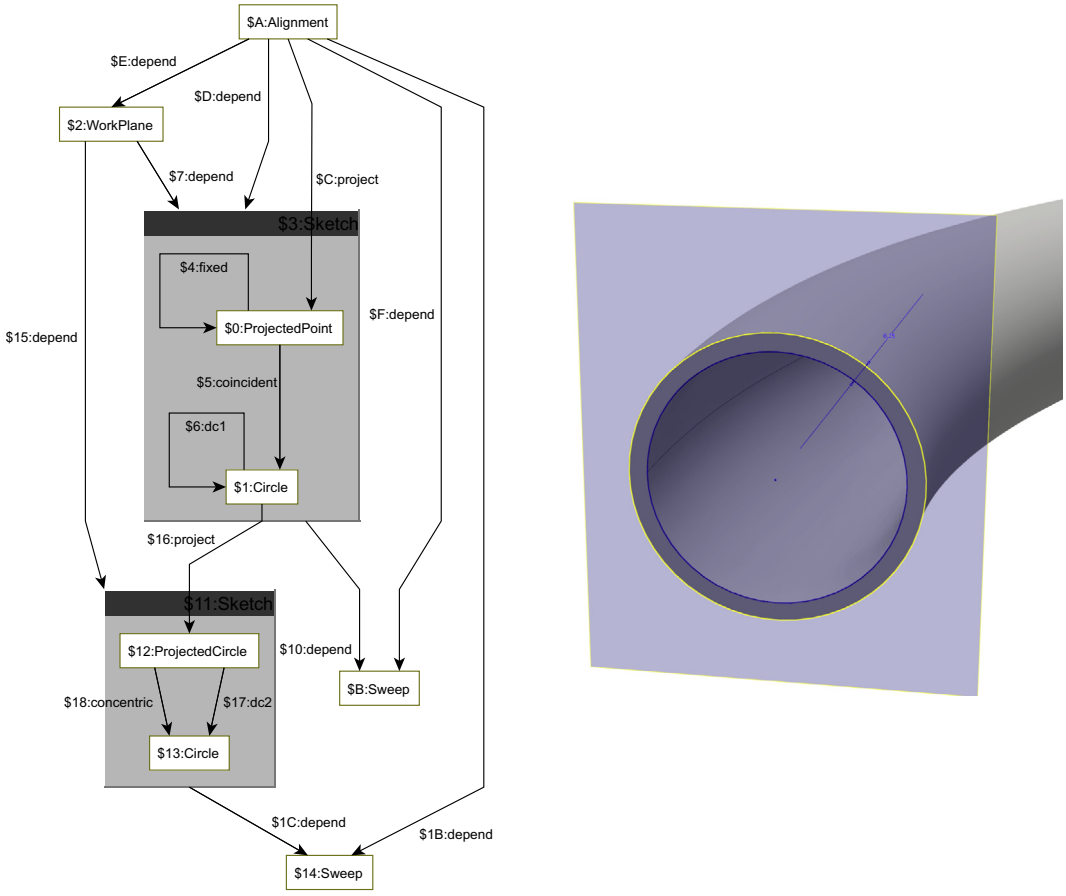

Fig. 8. Graph-based representation of the LiningSpace in LoD 3.

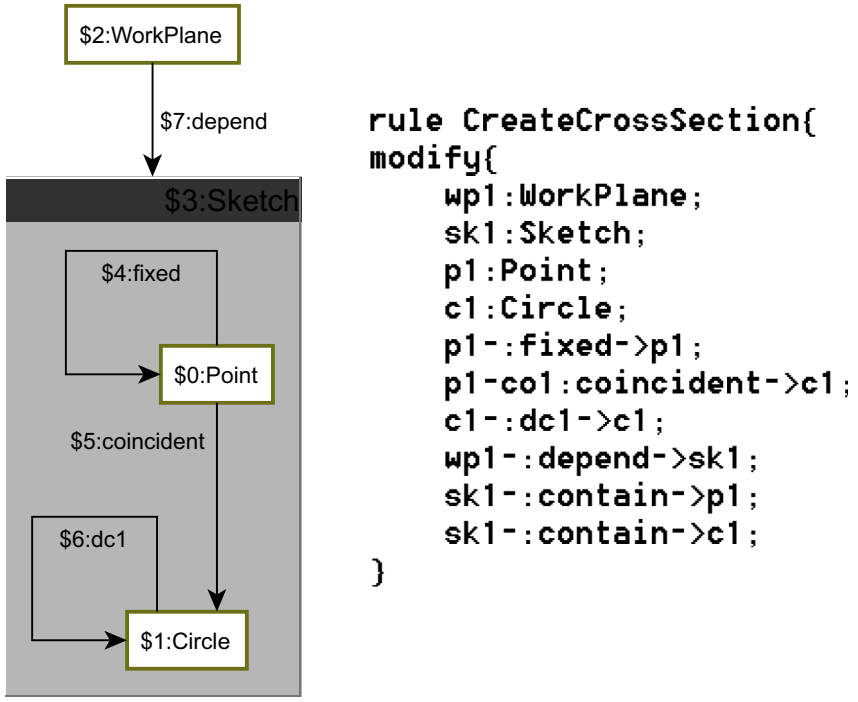

Fig. 9. Graphical illustration of a rule's rewrite part and the corresponding formal definition.

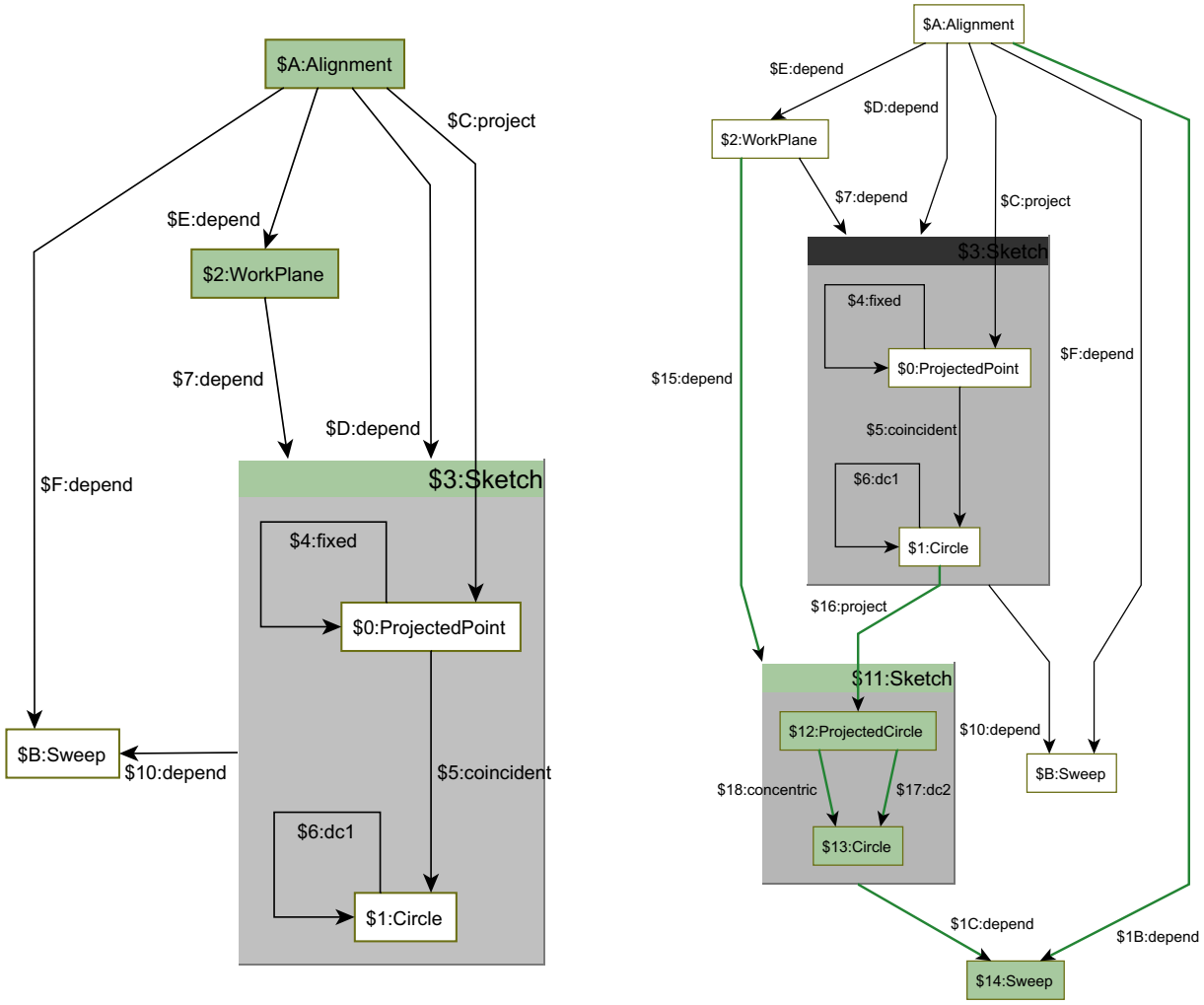

Fig. 10. Pre-state and post-state of a graph transformed by the rewrite rule depicted in Fig. 11. Note, that the pattern part of the rule in the host graph (left) and the rewrite part in the result graph (right) are highlighted with green color.

```
rule DetailCrossSection{
a:Alignment;
sk:Sketch-:contain->c:Circle;
if (sk.name == "FullTunnelSpaceSketch");
wp:WorkPlane-:depend->sk;
     modify{
     sk1:Sketch;
     eval{sk1.name="LiningSpaceSketch";}
     pc1:ProjectedCircle;
     c1:Circle;
     sw1:Sweep;
     wp-:depend->sk1;
     c-:project->pc1;
     pc1-:dc2->c1;
     pc1-:concentric->c1;
     sk1-:contain->c1;
     sk1-:contain->pc1;
     a-:depend->sw1;
     sk1-:depend->sw1;
}
```

Fig. 11. Formal definition of the rule depicted in Fig. 10. Lines 2–5 describe the pattern part while the rewrite part is defined after the initiating keyword *modify*.

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. [24] 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*, in which a sketch graph is stored inside a graph entity representing a sketch, without connecting it to the procedural graph. This would result in the use of hierarchical graphs, which are technically supported by GrGen.NET. 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. Additionally, GrGen.Net does not allow the definition of edges between nodes belonging to different (sub) graphs in hierarchical graphs.

When using the third option on the other hand, relationships between sketch graphs can be modeled much more straightforward. 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 Fig. 12 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, which is depicted on the right side. 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.

The integration of the graphs is not defined as an actual graph transformation. In fact, it is the conceptual basis for the definition of the overall graph and the rewrite rules, as the rules can 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 are used in order to indicate which part of the graph is referred to in a particular context.

3.5. 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 in an unambiguous way (i.e. without conflicts and inconsistencies) and result in a usable procedural geometry model. For this reason, the graph must 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 or *well constrained* sketch [22]. Otherwise the interpretation of the graph will not lead to an unambiguous solution. The aspects of sketch graphs in this regard are discussed in detail in Vilgertshofer and Borrmann [44]. There, the structure of sketch graphs is defined and the development of and requirements on rewrite rules that produce an interpretable graph are shown. The main tools for achieving an unambiguous representation discussed 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 geometric constraint solver (GCS) of the CAD-system that the *evaluated sketch* is created in.

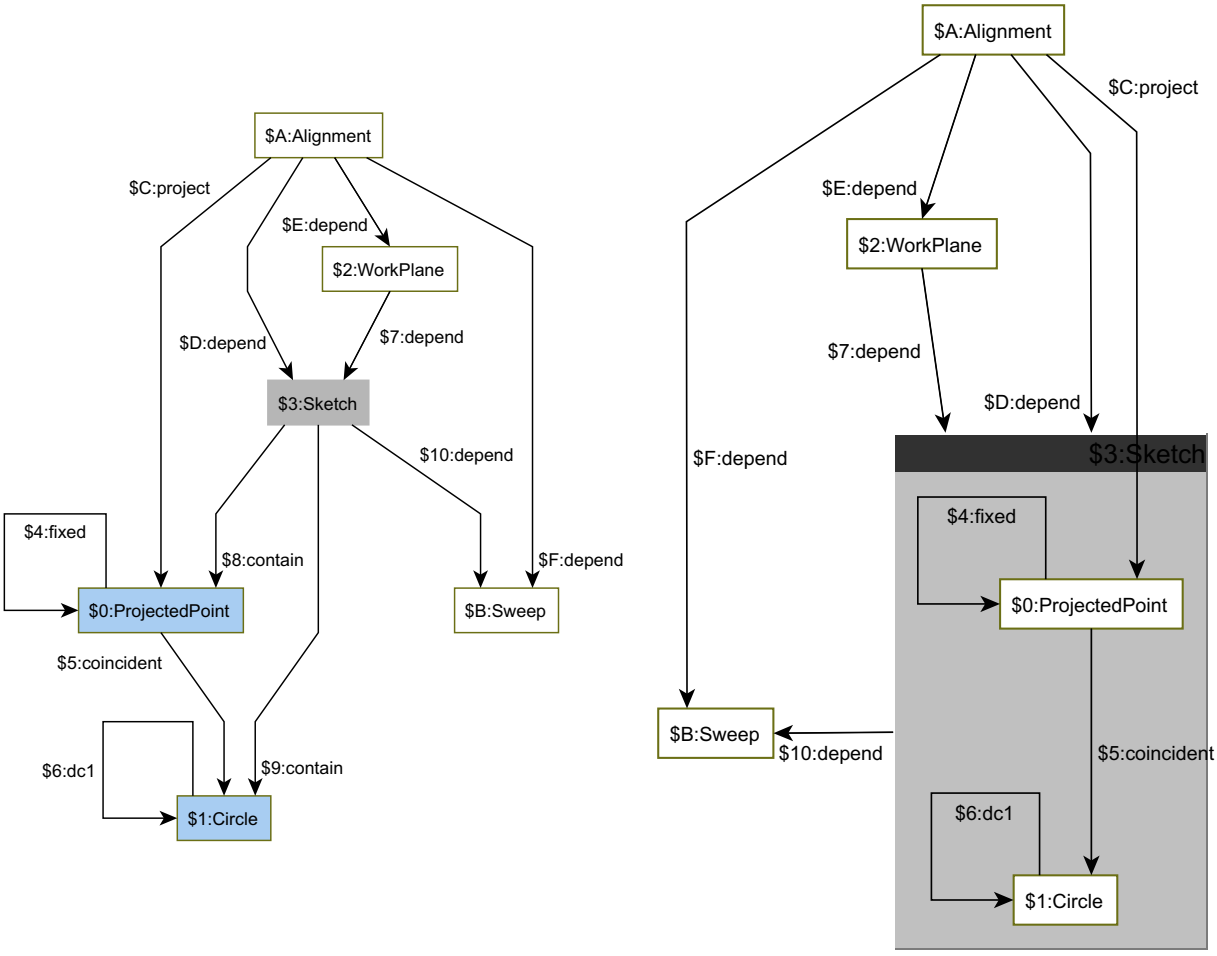

Fig. 12. Grouping of nodes representing geometric elements (blue) belonging to a sketch (right). A graph without the grouping of nodes is depicted on the left. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The latter refers to the particularity of interactive parametric systems, that the GCS chooses the solution, which is most similar to the sketched geometry, if there are multiple solutions to a given constraint problem. This behavior must be emulated by the graph-based generation method, because otherwise the GCS may produce a solution that is formally correct, but not what the user intended. Fig. 13 illustrates this problem. Here, the points *A* and *B* are grounded to their positions. The start- and endpoint of line *c* are defined as coincident with these two points. The startpoints of the lines *a* and *b* are defined as coincident with these points respectively. Additionally, the endpoints of the lines *a* and *b* are defined as coincident with point *C*. The lengths of the lines *a* and *b* are constrained to be of equal length.

The interpretation of this constraint problem may lead to the two solutions depicted in red and green color in Fig. 13 and the interpretation of a graph representing this sketch is not unambigous. If temporary coordinates are used when drawing the point *C* and the endpoints of the lines *a* and *b*, the solution most similar to this temporary coordinates is chosen by the GCS and the outcome in terms of the evaluated model matches the user's intention.

For this reason, temporary coordinates are assigned to geometric elements which form an approximation of the relative positioning of the geometric elements. With their help, the interpretation of the graph-based representation by the GCS will always result in the intended outcome. The temporary coordinates are stored in attributes of the nodes representing the geometric elements. The described method of prearranging the geometry does now result in the intended solution.

4. Case study and 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 (see Section 2.4) has been used while the generation of the evaluated sketch is performed with the commercial parametric CAD application Autodesk Inventor [2]. Inventor contains a geometrical constraint solver, which interprets the constraint problem defined by a graph. A software prototype was developed in order to combine the functionalities of GrGen.NET with that of Autodesk Inventor to illustrate the straight-forward application of rewrite rules and the consecutive creation of the evaluated model. The case study chosen is a knowledge intensive case-study, as the design of a multi-scale tunnel model requires the knowledge of the designing engineers not only in the field of tunnel design, but also in terms of applying parametric and procedural modeling. Indeed, the graph formalizes the knowledge of the engineers regarding detailing procedures, which are currently preformed manually.

The research presented here was undertaken as part of the research group 3DTracks "Computer-Aided Collaborative Subway Track Planning in Multi-Scale 3D City and Building Models", which is funded by the German Reserach Foundation (DFG). In scope of the 3DTracks group, the second main subway track in Munich that is currently under planning, was chosen as a case study project. Based on conventional 2D plans from the project, the multi-scale model for shield tunnels [9] was developed. The data of this subway track project was further used as a conceptional basis for the definition of the graph rewrite system to semi-automatically create consistent multi-scale models, as introduced in Section 2.2 to proof the feasibility of the developed methodology.

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 [16]. More concretely, it provides the possibility to create a graph metamodel and respective graph rewrite rules implementing (as default) an Single-Pushout Approach [4,14]. Further informations regarding the characteristics of the graph transformation process can be found in Section 2.4. As proof-of-concept, a graph rewrite system was implemented on the basis of GrGen.NET. It allows to create the graph-based representation of the shield tunnel model in a step-wise manner up to LoD 4 without any manual modeling operations. For the first evaluation of the approach, rather comprehensive rewrite rules were defined, as the primary focus was to proof that a graph created by these rules can indeed be used for the creation of the corresponding evaluated model. An overview of the underlying graph metamodel is given in the previous section. It shows the different types of the nodes and edges that were used as well as whether they conceptually belong to a sketch graph or to the procedural graph. A graph representing a LoD 3 model is shown in Fig. 14, the corresponding geometric model is illustrated in Fig. 15.

Besides the geometric representation, this model also contains low-level semantics. The rewrite rules create a graph in which the names of the tunnel spaces are stored in the nodes representing the procedural modeling operations that generate the geometric representations of the semantic objects. This enables the possibility to export a geometric-semantic model from Autodesk Inventor, which can then be used to write IfcTunnel files as presented in Vilgertshofer et al. [45].

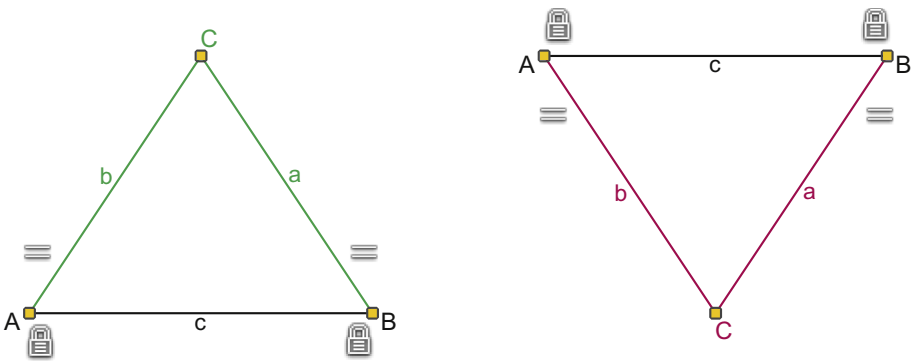

Fig. 13. Two formally correct solutions (green and red) generated by the interpretation of a constraint problem: The points *A* and *B* are grounded to their positions. The start- and endpoint of line *c* are defined as coincident with these two points. The startpoints of the lines *a* and *b* are defined as coincident with these points respectively. Additionally, the endpoints of the lines *a* and *b* are defined as coincident with point *C*. The lengths of the lines *a* and *b* are constrained to be of equal length. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

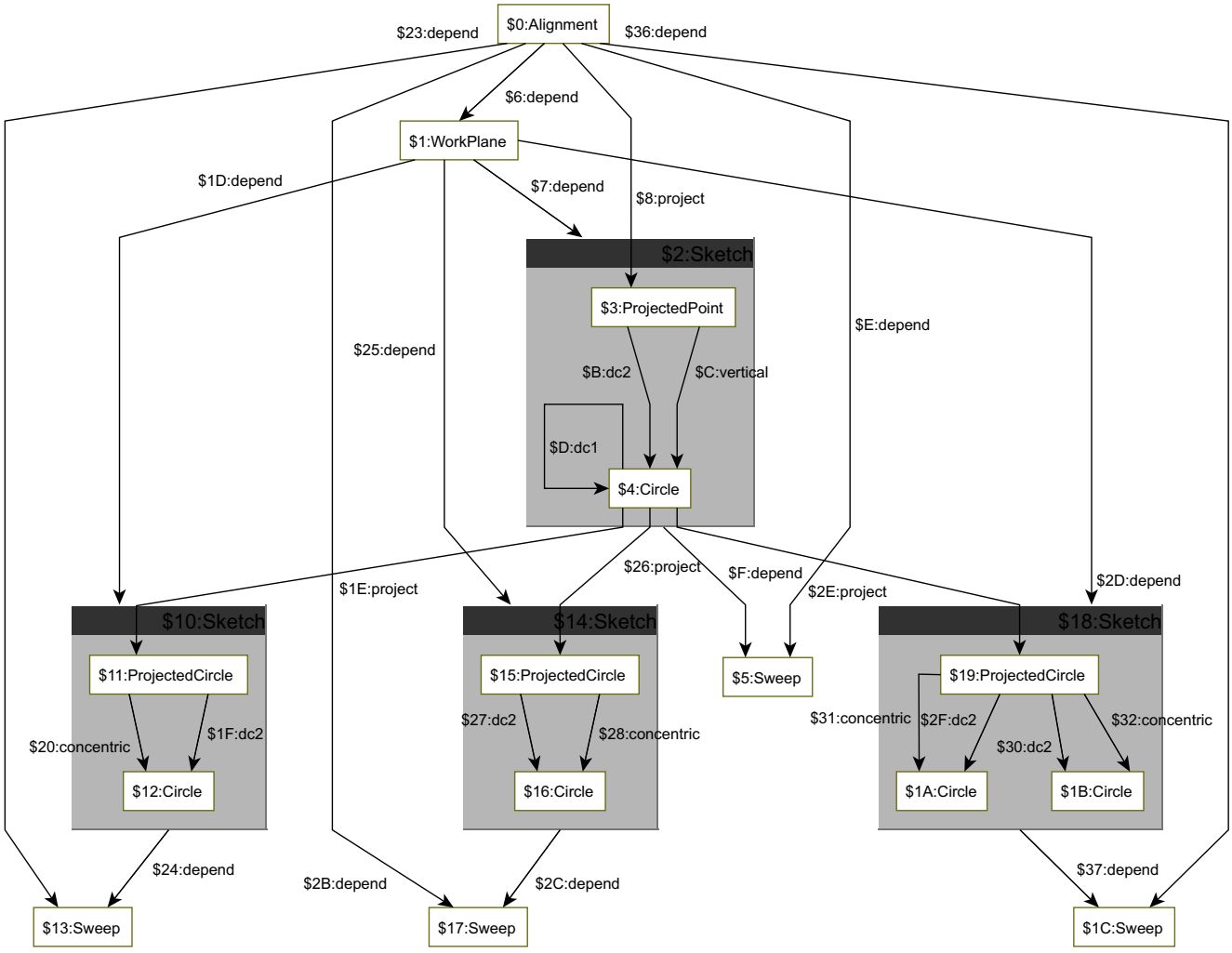

Fig. 14. Graph representing the LoD 3 model depicted in Fig. 15.

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 parametric 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 parametric modeling system Autodesk Inventor. A short overview of the functionality follows.

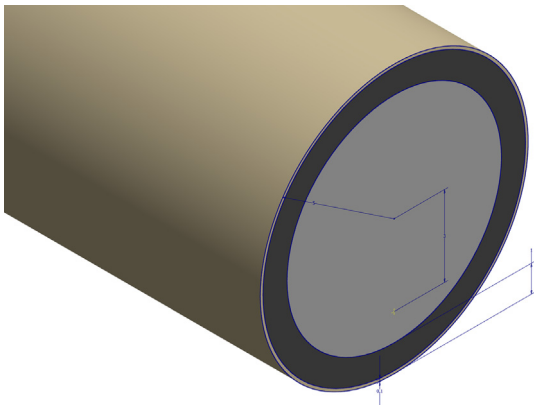

Fig. 15. Evaluated model of the graph depicted in 14.

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 corresponding 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 the parametric CAD system 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.

We make use of the mechanical engineering modeling system Inventor thanks to its advanced support of parametric design, in particular with respect to defining dependencies between geometric entities in a very flexible and powerful manner. So far, we do not work with currently available BIM modelers as they do not support this kind of flexibility in defining parametric dependencies - in most cases parametric functionalities are restricted to rather simple pre-defined dependencies.

However, Inventor lacks the possibility to define semantics for the created geometric objects, which is a standard feature in state-of-the art BIM modeling tools. As the proposed methodology does include semantics as part of the graph notation, the developed system can be interpreted as a BIM system that makes use of the advantages of parametric modeling.

5. Discussion

The original concept of this research is the presented method of capturing parametric modeling knowledge by formal graph rewrite operations. As laid out in Section 2.1 this contributes to the field of Knowledge-based Engineering or Computational Design Synthesis. With the presented definition of a graph metamodel, the approach enables the representation and storage of parametric procedural models including its construction history by using a vendor-neutral graph-based data format. Doing so, the proposed method is generic and can easily support other parametric CAD-systems besides Autodesk Inventor, which was chosen solely for demonstration purposes. One of the key features is that the evaluated model remains modifiable and can be used as basis for further modeling in the respective CAD system. As the metamodel itself is also easily extendable to support additional geometric elements and procedural modeling operations, the method may also be applied to generate much more complex parametric models and to capture the modeling knowledge used for various planning scenarios.

Additionally, the comparison of the application of a rule with the manual task of creating a parametric model shows that the use of rules is faster and prevents modeling errors, if the rules are defined correctly. The definition of the rules, on the other hand, is fairly complex and must be done very thoroughly. As the rules may then be applied many times and in different projects, this process of rule definition is worth its effort. In addition, the method opens the way for capturing and persistently storing a design office's best practices of model creation and development in a software-independent manner, which provides a value on its own.

While the general principle has been successfully applied, as shown in Section 4, certain limitations apply at the current state of the research project. Depending on the desired modeling outcome, a large set of rules may be required and needs to be defined before the method can be productively used by designers. Furthermore, it is nor clear yet, what size in terms of the represented modeling operations is reasonable for a single rule. Here, further research is required. Another limitation appears when defining rules that rely on existing 3D geometry. Here, the so called persistent naming problem [33] may occur, as parts of 3D geometric elements, such as faces or edges, which are created while the evaluated model is generated, cannot be referenced in the graph at the current stage. As this is a major limitation to the rule-based generation of complex 3D models, this problem is also focused on in subsequent research.

6. Conclusion

The presented research introduces a concept for the graph-based representation of product models and their automatic detailing by performing graph rewrite operations based on formal rules defined in a graph rewriting system. It focuses on product models of shield-tunnels and the automated creation of consistency-preserving multi-scale versions of such models. The main contribution is the development of a graph rewriting system that enables the generation of graphs representing those product models.

As stated in Sections 1 and 2.2 the motivation for this approach is based on previous research by Borrmann et al. which came to the conclusion that the manual creation of consistent multi-scale product models is a "complex, time-consuming and error-prone task which could strongly benefit from automation mechanisms". From this starting point, we chose the application of graph rewriting techniques as a means for realizing such an automation mechanism. Despite this chronology of problem identification and solution development, our findings have led us to believe that the proposed methodology is generic and applicable to a wide field of modeling tasks which either require consistency preservation or the application of complex parametric and procedural modeling operations.

However, in the moment, we are not yet ready to prove that the developed theory is applicable in a generic manner. We chose to apply it to the shield-tunnel scenario first in order to prove its applicability in this specific context. The next step in our research will be the further development of the theory towards a more general level. Nonetheless, we also aim at a further extension and refinement in scope of the presented scenario. More precisely, we target on the definition of a larger set of rewrite rules, which enables end users to create more diversified models. Additionally, we will extend the graph metamodel and work on overcoming the limitations described in the previous section.

In Section 2.1 we laid out how designers can benefit from a range of existing methods, approaches and tools that support them in their challenging work. We believe that our approach adds to the variety of instruments that designers can benefit from during the iterative process of determining good design solutions for engineering problems.

Acknowledgments

We gratefully acknowledge the support of the German Research Foundation (DFG) for funding the project under grant FOR 1546. We also want to thank the members of the 3DTracks research group for their support and the productive discussions.

References

[1] K. Amadori, M. Tarkian, J. Ölvander, P. Krus, Flexible and robust CAD models for design automation, Adv. Eng. Inf. 26 (2012) 180–195, http://dx.doi.org/10.1016/j.aei.2012.01.004.
[2] Autodesk, Autodesk Inventor <http://www.autodesk.de/products/inventor/>.
[3] M. Bhatt, A. Borrmann, R. Amor, J. Beetz, Architecture, computing, and design assistance, Autom. Construct. 32 (2013) 161–164, http://dx.doi.org/10.1016/j.autcon.2013.01.001.
[4] J. Blomer, R. Geiß, E. Jakumeit, The GrGen.NET User Manual, 2014 <http://www.info.uni-karlsruhe.de/software/grgen/GrGenNET-Manual.pdf>.
[5] A. Borkowski, E. Grabska, Representing designs by composite graphs, in: I. Smith, (Ed.), Proceedings of the IABSE Colloquium on Knowledge Support Systems in Civil Engineering, Bergamo, March 1995, IABSE Reports 72, 1995, pp. 27–36.
[6] A. Borrmann, M. Flurl, J.R. Jubierre, R.P. Mundani, E. Rank, Synchronous collaborative tunnel design based on consistency-preserving multi-scale models, Adv. Eng. Inf. 28 (2014) 499–517, http://dx.doi.org/10.1016/j.aei.2014.07.005.
[7] A. Borrmann, Y. Ji, J.R. Jubierre, M. Flurl, Procedural modeling: a new approach to multi-scale design in infrastructure projects, in: Proceedigns of the EG-ICE workshop on intelligent computing in civil engineering, Herrsching, Germany, 2012.
[8] A. Borrmann, J.R. Jubierre, A multi-scale tunnel product model providing coherent geometry and semantics, in: Proceedings of the 2013 ASCE international workshop on computing in civil engineering, Los Angeles, 2013, pp. 291–298, http://dx.doi.org/10.1061/9780784413029.03.
[9] A. Borrmann, T.H. Kolbe, A. Donaubauer, H. Steuer, J.R. Jubierre, M. Flurl, Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications, Comp-Aided Civil Infrastruct Eng 30 (2014) 263–281, http://dx.doi.org/10.1111/mice.12090.
[10] A. Bradley, H. Li, R. Lark, S. Dunn, BIM for infrastructure: an overall review and constructor perspective, Autom. Construct. 71 (2016) 139–152, http://dx.doi.org/10.1016/j.autcon.2016.08.019.
[11] J.D. Camba, M. Contero, P. Company, Parametric CAD modeling: an analysis of strategies for design reusability, Comp-Aided Des 74 (2016) 18–31, http://dx.doi.org/10.1016/j.cad.2016.01.003.

[12] M. Chein, M.L. Mugnier, M. Croitoru, Visual reasoning with graph-based mechanisms: the good, the better and the best, Knowl Eng Rev 28 (2013) 249–271, http://dx.doi.org/10.1017/S0269888913000234.
[13] D. Cooper, G. La Rocca, Knowledge-based techniques for developing engineering applications in the 21st century, in: 7th AIAA ATIO conference, Belfast, Northern Ireland, 2007.
[14] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner, Algebraic approaches to graph transformation: part II: single pushout approach and comparison with double pushout approach, in: G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, 1997.
[15] I. Fudos, C.M. Hoffmann, A graph-constructive approach to solving systems of geometric constraints, ACM Trans Graph 16 (1997) 179–216, http://dx.doi.org/10.1145/248210.248223.
[16] R. Geiß, G.V. Batz, D. Grund, S. Hack, A. Szalkowski, Grgen: a fast SPO-based graph rewriting tool, Proc. ICGT 4178 (2006) 383–397, doi:10.1.1.64.4131.
[17] E. Grabska, A. Łachwa, G. Ślusarczyk, New visual languages supporting design of multi-storey buildings, Adv. Eng. Inf. 26 (2012) 681–690, http://dx.doi.org/10.1016/j.aei.2012.03.009.
[18] G. Gröger, L. Plümer, How to achieve consistency for 3D city models, GeoInformatica 15 (2011) 137–165, http://dx.doi.org/10.1007/s10707-009-0091-6.
[19] R. Heckel, Graph transformation in a nutshell, Electron. Notes Theoret. Comp. Sci. 148 (2006) 187–198, http://dx.doi.org/10.1016/j.entcs.2005.12.018.
[20] B. Helms, Object-Oriented Graph Grammars for Computational Design Synthesis, Ph.D. thesis, TU München, 2013.
[21] B. Helms, K. Shea, Computational synthesis of product architectures based on object-oriented graph grammars, J. Mech. Des. 134 (2012) 021008, http://dx.doi.org/10.1115/1.4005592.
[22] M. Hidalgo, R. Joan-Arinyo, Computing parameter ranges in constructive geometric constraint solving: Implementation and correctness proof, Comp.-Aided Des. 44 (2012) 709–720, http://dx.doi.org/10.1016/j.cad.2012.02.012.
[23] F.R. Hoisl, Visual, Interactive 3D Spatial Grammars in CAD for Computational Design Synthesis, Ph.D. thesis, TU München, 2012.
[24] E. Jakumeit, S. Buchwald, M. Kroll, GrGen.NET, 2010, doi:http://dx.doi.org/10.1007/s10009-010-0148-8.
[25] Y. Ji, A. Borrmann, J. Beetz, M. Obergrießer, Exchange of parametric bridge models using a neutral data format, J. Comput. Civil Eng. 27 (2013) 593–606, http://dx.doi.org/10.1061/(ASCE)CP.1943-5487.0000286.
[26] K. Kim, J. Teizer, Automatic design and planning of scaffolding systems using building information modeling, Adv. Eng. Inf. 28 (2014) 66–80, http://dx.doi.org/10.1016/j.aei.2013.12.002.
[27] O. Kniemeyer, Design and Implementation of a Graph Grammar Based Language for Functional-Structural Plant Modelling (Ph.D. thesis), BTU Cottbus, 2008.
[28] T.H. Kolbe, Representing and exchanging 3D city models with CityGML, in: J. Lee, S. Zlatanova (Eds.), Proceedings of the 3rd International Workshop on 3D Geo-Information, Seoul, Korea, Lecture Notes in Geoinformation and Cartography, Springer, Berlin Heidelberg, 2009, http://dx.doi.org/10.1007/978-3-540-87395-2.
[29] A. Köninger, S. Bartel, 3D-GIS for urban purposes, Geoinformatica 2 (1998) 79–103, http://dx.doi.org/10.1023/A:1009797106866.
[30] B. Kraft, M. Nagl, Visual knowledge specification for conceptual design: definition and tool support, Adv. Eng. Inf. 21 (2007) 67–83, http://dx.doi.org/10.1016/j.aei.2006.10.001.
[31] G. La Rocca, Knowledge based engineering: between AI and CAD. Review of a language based technology to support engineering design, Adv. Eng. Inf. 26 (2012) 159–179, http://dx.doi.org/10.1016/j.aei.2012.02.002.
[32] C. Langenhan, M. Weber, M. Liwicki, F. Petzold, A. Dengel, Graph-based retrieval of building information models for supporting the early design stages, Adv. Eng. Inf. 27 (2013) 413–426, http://dx.doi.org/10.1016/j.aei.2013.04.005.
[33] D. Marcheix, G. Pierra, A survey of the persistent naming problem, in: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications - SMA '02, ACM Press, New York, New York, USA, 2002, p. 13, http://dx.doi.org/10.1145/566282.56628.
[34] D. Mun, S. Han, J. Kim, Y. Oh, A set of standard modeling commands for the history-based parametric approach, Comp.-Aided Des. 35 (2003) 1171–1179, http://dx.doi.org/10.1016/S0010-4485(03)00022-8.
[35] R. Niemeijer, B. de Vries, J. Beetz, Freedom through constraints: user-oriented architectural design, Adv. Eng. Inf. 28 (2014) 28–36, http://dx.doi.org/10.1016/j.aei.2013.11.003.
[36] J.C. Owen, Algebraic solution for geometry from dimensional constraints, in: Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM, 1991, pp. 397–407.
[37] G. Rozenberg, Handbook of Graph Grammars and Computing by Graph Transformation, vol. 1, World Scientific, 1997, http://dx.doi.org/10.1142/9789812384720.
[38] M. Ruiz-Montiel, J. Boned, J. Gavilanes, E. Jiménez, L. Mandow, J.L. Pérez-De-La-Cruz, Design with shape grammars and reinforcement learning, Adv. Eng. Inf. 27 (2013) 230–245, http://dx.doi.org/10.1016/j.aei.2012.12.004.
[39] C. Schultz, M. Bhatt, A. Borrmann, Bridging qualitative spatial constraints and feature-based parametric modelling: expressing visibility and movement constraints, Adv. Eng. Inf. 31 (2017) 2–17, http://dx.doi.org/10.1016/j.aei.2015.10.004.
[40] M. Sester, Optimization approaches for generalization and data abstraction, Int. J. Geograph. Inf. Sci. 19 (2005) 871–897, http://dx.doi.org/10.1080/13658810500161179.
[41] J.J. Shah, M. Mäntylä, Parametric and Feature-Based CAD/CAM: Concepts, Techniques and Applications, John Wiley & Sons, New York, 1995.
[42] M. Stokes, Managing Engineering Knowledge-MOKA: Methodology for Knowledge Based Engineering Applications, Professional Engineering Publishing, 2001.
[43] W.J. Verhagen, P. Bermell-Garcia, R.E. van Dijk, R. Curran, A critical review of knowledge-based engineering: an identification of research challenges, Adv. Eng. Inf. 26 (2012) 5–15, http://dx.doi.org/10.1016/j.aei.2011.06.004.
[44] S. Vilgertshofer, A. Borrmann, Automatic detailing of parametric sketches by graph transformation, in: Proceedings of the 32nd ISARC 2015, Oulu, Finnland, 2015.
[45] S. Vilgertshofer, J.R. Jubierre, A. Borrmann, IfcTunnel - a proposal for a multi-scale extension of the IFC data model for shield tunnels under consideration of downward compatibility aspects, in: Proceedings of the European Conference on Product and Process Modeling (ECPPM), Limassol, Cyprus, 2016.
[46] A. Weissman, M. Petrov, S.K. Gupta, A computational framework for authoring and searching product design specifications, Adv. Eng. Inf. 25 (2011) 516–534, http://dx.doi.org/10.1016/j.aei.2011.02.001.
[47] N. Yabuki, T. Aruga, H. Furuya, Development and application of a product model for shield tunnels, in: Proceedings of the 30th ISARC, Montréal, 2013.