Detailed Proposal for Master Thesis 01

19 May 2018

William Bombardelli da Silva

Properties of Triple Graph Grammars with Non-Terminal Nodes

Introduction

In the Model-driven Engineering (MDE) approach, models from one domain often need to be transformed to another domain consistently. A promising approach to solve this problem is the triple graph grammar (TGG), which aims to describe the consistency relation between two domains of models in terms of a grammar that produces all consistent pairs of models in these domains. Even though TGGs have shown some positive results, the expressiveness and usability of the formalism could be enhanced (Anjorin *et al.* , 2016). We believe that TGGs could profit from the concept of non-terminal symbols and context-freeness of the theory of formal languages.
More specifically, we propose an extension of the TGG formalism that includes non-terminal nodes and restricts the syntax of the grammar production rules so to create a family of grammars, that is potentially a proper subset of the general TGG. This restriction on the form of the rules is analogous to the context-free grammars of Chomsky. For this extended family of TGGs we aim to devise a transformation algorithm with a polynomial worst-case time and spatial complexity.

Motivation. The restriction on the form of the rules of TGGs may increase significantly its usability, as it reduces the minimum amount of rules necessary to describe a language, and increase the efficiency of the transformation algorithm, like it happens in the context of compilers with the LL and LR languages. This can, in turn, encourage more the use of triple graph grammars in practice and ease the solution of model transformation problems.

Key-words. Graph Grammars, Triple Graph Grammars, Parsing, Context-free, Graph Transformation, Model Transformation, Model-driven Engineering, Software Engineering.

Methodology

To accomplish this work, we first present the concept of boundary node-label controlled (B-NLC) graph grammar derived from Rozenberg & Welzl (1986); Flasiński & Flasińska (2014); Flasiński (1993) accompanied of examples. Then we extend B-NLC graph grammars to create B-NLC triple graph grammars (B-NLC-TGG), defining clearly the syntax and semantics of this new formalism also accompanied by examples.
Second, we devise a specialized transformation algorithm from BN-NLC-TGG, based on the CYK-like bottom-up parsing algorithm for B-NLC grammar, and analyze its computational complexity and termination. Subsequently, we study the expressiveness BN-NLC-TGG in comparison with the standard TGG. Finally, we want to analytically evaluate the practical use of our approach by comparing the amount of rules required by B-NLC-TGG and the standard TGG for the most reported transformations of the literature and by comparing the efficiency of the transformation algorithms in terms of run-time.
A discussion of the currently very researched topics of incremental transformation and model synchronization and its relation with our new approach shall also be included in this work, as well as, the proposal of possible extensions of it, like the characterization of an equivalence between B-NLC-TGG and a theoretical computational model (e.g. pushdown automaton or turing machine), similarly to what exists in the classical

theory of formal languages, or also the application of B-NLC graph grammars for the model-driven automatic generation of test cases.

Related Work

There exists an extensive literature on graph transformation and rewriting systems, specially on the algebraic approach to graph grammars (Ehrig *et al.* , 2015), and triple graph grammars (Schürr, 1994). Moreover, there are several proposals of families of graph grammars applied to the context of visual languages, including regular (Gilroy *et al.* , 2017), context-free (Rekers & Schürr, 1997), and context-sensitive grammars (Zhang *et al.* , 2001; Adachi *et al.* , 1999; Drewes *et al.* , 2010). Marriott & Meyer (1997) proposed a hierarchy for classes of graph grammars, Flasiński in (1998) associated several types of graph grammars with the respective complexity to parse and in (1993) presented a parsing algorithm with complexity $O(n^2)$.
Shi *et al.* (2016) proposed a method for simplifying graph grammars. And several authors report the use of TGGs to solve model transformations (Gottmann *et al.* , 2016). But we have not found any publication that tries to enhance TGGs' efficiency and expressiveness/ usability by means of restrictions on the form of the grammar rules.

Boundary Node-label Controlled Graph Grammar

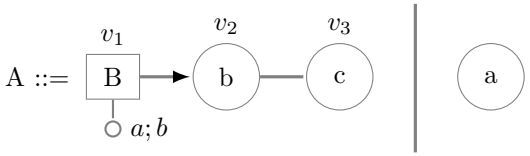
Definition. A graph G over Σ is a triple (V, E, ϕ), where Σ is alphabet of labels, V is the finite set of vertices, $E \subseteq V \times V$ the set of edges and $\phi : V \to \Sigma$ the total labeling function for the vertices. For a fixed graph G we will often refer to its components as V_G, E_G and ϕ_G. Moreover, we denote the set of all graphs over Σ as $\mathcal{G}_\Sigma$ and define the special empty graph $\varepsilon := (\emptyset, \emptyset, \emptyset)$.

Definition. A Γ-boundary graph (V, E, ϕ) is a weakly connected graph where nodes labeled with any symbol from Γ are not neighbors, that is $\nexists (v, w) \in E.\ \phi(v) \in \Gamma \wedge \phi(w) \in \Gamma$.

Definition. A NLC graph grammar $GG = (\Sigma, \Delta \subset \Sigma, P, S \in \Sigma)$ consists of a finite set of symbols (the alphabet) Σ, a set of terminal symbols Δ that is a subset of Σ (we will name the other partition of Σ as $\Gamma := \Sigma \setminus \Delta$), a set of production rules P of the form $A \to G, w$ where $A \in \Gamma$ is the left-hand side, and $G \in \mathcal{G}_\Sigma$ together with the embedding function $w : V_G \to 2^\Sigma$ is the right-hand side.

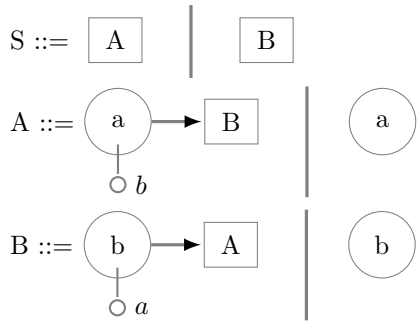
Definition. A boundary NLC (B-NLC) graph grammar GG is a NLC graph grammar where all graphs at the right-hand side of the productions are Γ-boundary graphs.

In the following, we present our concrete syntax, inspired by the well-known backus-naur form, to denote B-NLC graph grammar rules. Let $p = A \to G, w$ and $q = A \to H, z$be a production rule with $G = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_3),\ (v_3, v_2)\}, \{v_1 \mapsto B, v_2 \mapsto b, v_3 \mapsto c\})$, $w = \{v_1 \mapsto \{a, b\}\}$, $H = (\{u_1\}, \emptyset, \{u_1 \mapsto a\})$ and $z = \emptyset$, we denote p and q together as

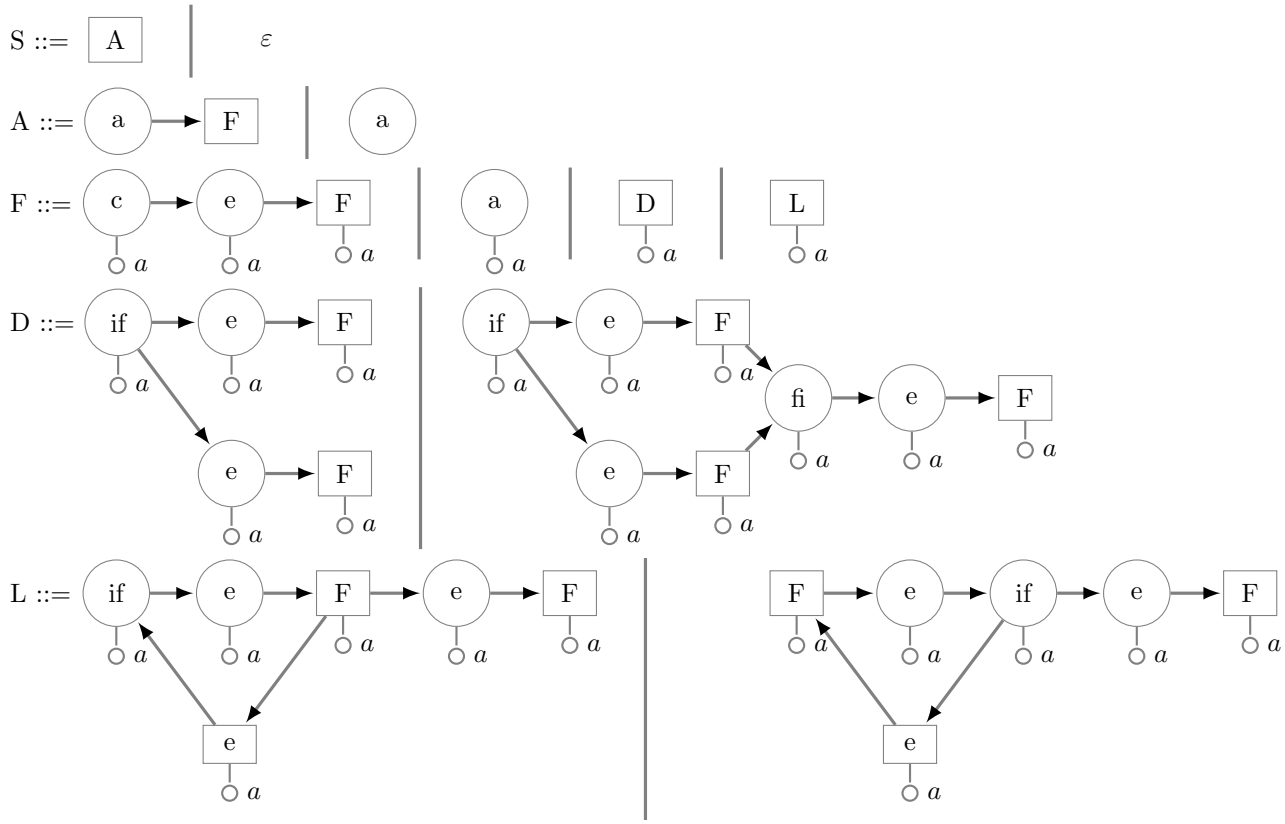


Note that, we use squares for nodes labeled with non-terminal symbols, circles for nodes labeled with terminal symbols, position the respective label inside the shape and the (possibly omitted) identifier over it. To depict the embedding function, we place below the respective node a small circle labeled with the image of the embedding function for this node.

Example. Chains of a's and b's. $GG = (\{S, A, B, a, b\}, \{a, b\}, P, S)$, where P is



Example. Activity diagrams. $GG = (\{S, A, F, D, L, a, c, e, if\}, \{a, c, e, if\}, P, S)$, where P is



The semantics of a B-NLC grammar is presented in the following.

Definition. Let G be a graph over Σ, G is derived into H with rule r and vertex v, we write $G \overset{r,v}{\Rightarrow} H$ and

call it a derivation step, if and only if,

$$\begin{aligned}
&\exists r = A \to R, w \in P.\ \exists v \in V_G. \\
&\phi_G(v) = A \\
&V_H = V_G \setminus \{v\} \cup V_R \\
&E_H = E_G \setminus (\{(v,u) \mid u \in V_G\} \cup \{(u,v) \mid u \in V_G\}) \\
&\quad \cup E_R \\
&\quad \cup \{(u,t) \mid (u,v) \in E_G \wedge \phi_G(u) \in w(t)\} \\
&\quad \cup \{(t,u) \mid (v,u) \in E_G \wedge \phi_G(u) \in w(t)\} \\
&\phi_H = (\phi_G \setminus \{(v,x)\}) \cup \phi_R
\end{aligned}$$

Note that, without loss of generalization, we suppose that $V_G \cap V_R = \emptyset$, that is, the vertex sets are disjoint. Moreover, we denote with $\Rightarrow^*$ the transitive reflexive closure of $\Rightarrow$.

Definition. A derivation D is a sequence of derivation steps and is written as

$$D = (G_0 \overset{r_0,v_0}{\Rightarrow} G_1 \overset{r_1,v_1}{\Rightarrow} G_2 \overset{r_2,v_2}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} G_n)$$

Definition. The language $L(GG)$ generated by the graph grammar $GG = (\Sigma, \Delta, P, S)$ is

$$L(GG) = \{G \mid S \Rightarrow^* G \wedge \forall v \in V_G.\ \phi_G(v) \in \Delta\}$$

Theorem 1. *For every $G \in L(GG)$, there is at least one finite derivation*

$$S \overset{r_0,v_0}{\Rightarrow} H_1 \overset{r_1,v_1}{\Rightarrow} H_2 \overset{r_2,v_2}{\Rightarrow} \dots \overset{r_{n-1},v_{n-1}}{\Rightarrow} H_n \cong G$$

Proof. ... □

Remark. It is not guaranteed that the derivation from Theorem 1 is unique. In the case that there are more than one derivation for a graph, we say that the grammar is ambiguous. Nevertheless, the grammar's ambiguity does not compromise the correctness of our algorithms presented in the following.

Given a certain B-NLC graph grammar GG and a graph G, we are interested in the problem of deciding if $G \in L(GG)$. This is sometimes called the *membership* problem and can be solved through a recognizer algorithm that always finishes answering yes if and only if $G \in L(GG)$ and no otherwise. A slight extension of this problem is the *parsing* problem, which consists of deciding if $G \in L(GG)$ and finding a derivation $S \Rightarrow^* G' \cong G$.
The Algorithm 1 is a correct (complete and consistent) parsing algorithm for any B-NLC graph grammar that runs in polynomial time and logarithmic space complexity. It has been taken from Rozenberg & Welzl (1986) and works in a bottom-up CYK-like (Cocke-Young-Kassami) manner.

Boundary Node-label Controlled Triple Graph Grammar

In the following, based on the definitions of B-NLC graph grammar, we define the extension of the standard TGG (Schürr, 1994) specification, the B-NLC TGG.

Definition. A triple graph $TG = G_s \overset{m_s}{\leftarrow} G_c \overset{m_t}{\rightarrow} G_t$ over Σ consists of three graphs G_s, G_c, G_t over Σ, respectively, the source, correspondence and target graphs, together with two total morphisms $m_s : G_c \to G_s$ and $m_t : G_c \to G_t$. We denote the set of all triple graphs over Σ as $\mathcal{TG}_\Sigma$.

Definition. A Γ-boundary triple graph $TG = G_s \leftarrow G_c \rightarrow G_t$ is such that G_s, G_c and G_t are Γ-boundary graphs.

Algorithm 1 Parsing Algorithm for B-NLC Graph Grammars

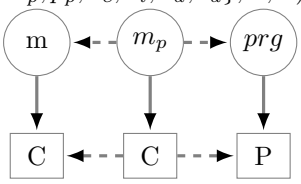
function $parse(GG = (\Sigma, \Delta, P, S), G = (V_G, E_G, \phi_G))$: $Derivation$
  $bup \leftarrow \{(\phi_G(x), x) \mid x \in V_G\}$
  $R_s \leftarrow 2^{bup}$
  repeat
    $R \leftarrow$ any $\{r \in R_s \mid V(r)$ is disjoint$\}$
    $T \leftarrow T \cup \{R\}$
    for all $d \in \Sigma$ **do**
      if $Z(\{(d, V(R))\}) \Rightarrow Z(R)$ **then**
        $bup \leftarrow bup \cup \{(d, V(R))\}$
        $R_s \leftarrow 2^{bup} \setminus T$
      end if
    end for
  until $R_s \neq \emptyset$
  return $(S, V_G) \in bup\,?\,true : false$
end function

Definition. A NLC triple graph grammar $TGG = (\Sigma, \Delta \subset \Sigma, P, S)$ consists of a finite set of symbols Σ, a set of terminal symbols Δ (define $\Gamma := \Sigma \setminus \Delta$), a set of production rules P of the form $(A, B, C) \to TG, w_s, w_t$ with $A, B, C \in \Gamma$, $TG \in \mathcal{TG}_\Sigma$ and two embedding functions $w_s : V_{G_s} \to 2^\Sigma$ and $w_t : V_{G_t} \to 2^\Sigma$.

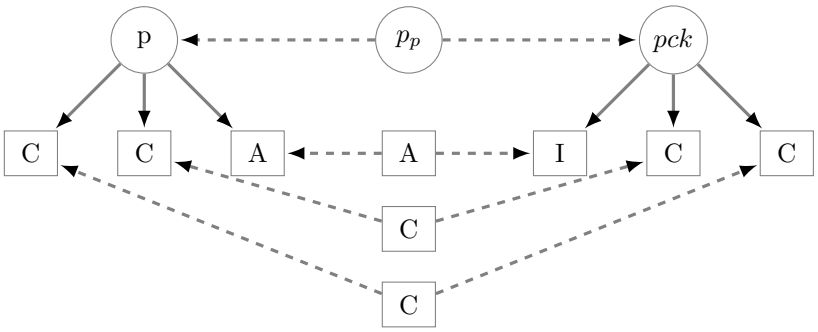
Definition. A B-NLC TGG is a NLC TGG where for all rules $(A, B, C) \to TG, w_s, w_t \in P$, TG is a boundary triple graph.

Example. Class Diagram to Java $TGG = (\{S, C, A, I, m, p, k, a, prg, pck, cls, imp, att\}, \{m, p, k, a, prg, pck, cls, imp, att, m_p, p_p, k_c, a_i, a_a, b_a\}, P, S)\}$, where P is

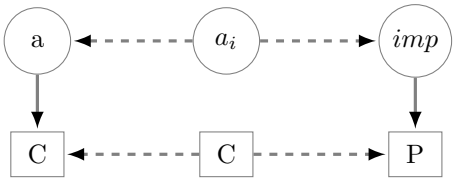
S-S-S ::=

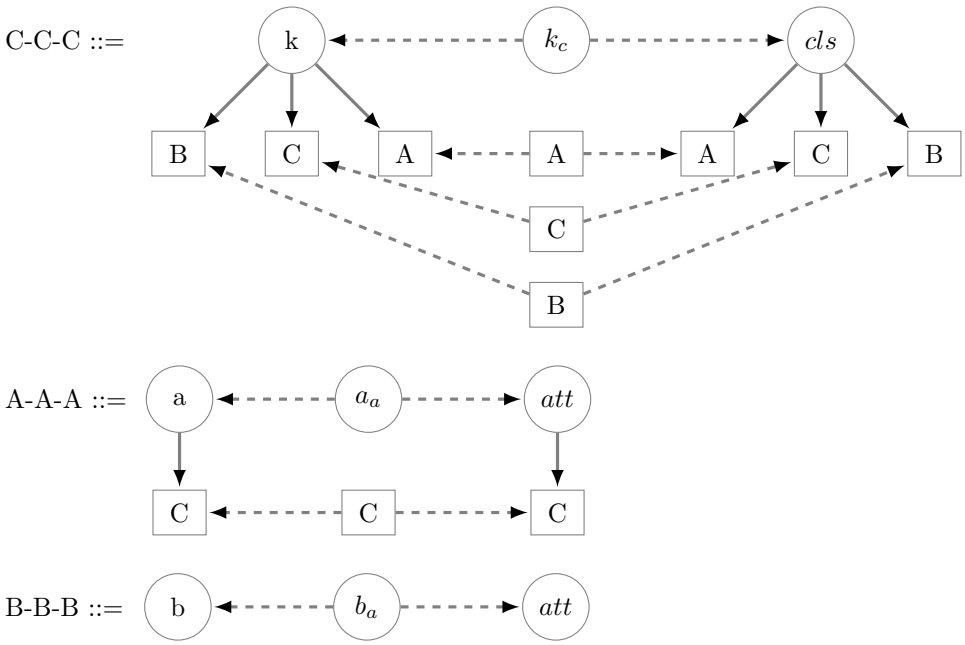


C-C-P ::=



A-A-I ::=





A consensus between the researchers of TGG is that the formalism aims at describing a consistency relation between two metamodels. In order to use it for transformation, it is necessary to operationalize it. In our case, the operationalization of the B-NLC TGG consists of deriving source and target rules from the production rules, which allows us to parse our input model and find a respective derivation using the Algorithm 1, to, then, generate the consistent triple graph containing the input and output models.

Definition. The derivation of source (target) rules from a B-NLC triple graph grammar $TGG = (\Sigma, \Delta, P, S)$ is done by creating a new B-NLC graph grammar $GG = (\Sigma, \Delta, \overline{P}, S)$, where for all rule $r_i = (A, B, C) \rightarrow G_s \leftarrow G_c \rightarrow G_t, w_s, w_t \in P$, there is a correspondent rule $\overline{r_i} = A \rightarrow G_s, w_s$ ($\overline{r_i} = C \rightarrow G_t, w_t$) in $\overline{P}$ and there is no other rule in $\overline{P}$.

The Algorithm 2 transforms a source graph into a target graph according to a triple graph grammar.

Algorithm 2 Forward Transformation Algorithm for B-NLC Triple Graph Grammars

function $fwd - transform(TGG = (\Sigma, \Delta, P, S), G = (V_G, E_G, \phi_G))$: $Graph$
  $GG \leftarrow$ source rules derived from TGG
  $D \leftarrow parse(GG, G)$
  if D **is** $S \overset{\overline{r_0}, v_0}{\Rightarrow} H_0 \overset{\overline{r_1}, v_1}{\Rightarrow} \dots \overset{\overline{r_n}, v_n}{\Rightarrow} H_n \cong G$ **then**
    let $S \leftarrow S \rightarrow S \overset{r_0, v_0}{\Rightarrow} TG_0 \overset{r_1, v_1}{\Rightarrow} \dots \overset{r_n, v_n}{\Rightarrow} TG_n = G_s \leftarrow G_c \rightarrow G_t$
    return G_t
  else
    return $fail$
  end if
end function

Evaluation

To evaluate our extension of the TGG approach we compare the amount of rules necessary to describe the most researched transformations of the literature in B-NLC TGG and standard TGG. Furthermore, we also report on the runtime for forward and backward batch transformations in a Intel Core i3 2.3GHz 4x 64bit

with 4GB RAM. The standard TGG version of the transformations were executed using the eMoflon Tool Leblebici *et al.* (2014). Table 1 presents these results.

Transformation	TGG Rules	B-NLC Rules	TGG Fwd	B-NLC Fwd	TGG Bwd	B-NLC Bwd
Class2ER						
Activity2ControlFlow						
StateMachine2PetriNets						
UMLClass2SDLBlock						
(Giese & Wagner, 2009)						
Code2ControlFlow						
(Pratt, 1971)						
BPMN2BPEL						
(Shi *et al.* , 2014)						
Activity2CSP						
(Bisztray *et al.* , 2009)						
StateMachine2Java						
(Striewe, 2008)						
Tree2XTree						
Star2Wheel						
LogicExpr2LogicCircuit						
NeuralNet2Function						

Table 1: Results of the empirical evaluation of the B-NLC TGG in comparison with standard TGG

References

Adachi, Yoshihiro, Kobayashi, Suguru, Tsuchida, Kensei, & Yaku, Takeo. 1999. An NCE context-sensitive graph grammar for visual design languages. *Pages 228–235 of: Visual Languages, 1999. Proceedings. 1999 IEEE Symposium on*. IEEE.

Anjorin, Anthony, Leblebici, Erhan, & Schürr, Andy. 2016. 20 Years of Triple Graph Grammars: A Roadmap for Future Research. *Electronic Communications of the EASST*, **73**.

Bisztray, Dénes, Heckel, Reiko, & Ehrig, Hartmut. 2009. Compositionality of Model Transformations. *Electronic Notes in Theoretical Computer Science*, **236**(C), 5–19.

Drewes, Frank, Hoffmann, Berthold, Janssens, Dirk, & Minas, Mark. 2010. Adaptive star grammars and their languages. *Theoretical Computer Science*, **411**(34-36), 3090–3109.

Ehrig, Hartmut, Ermel, Claudia, Golas, Ulrike, & Hermann, Frank. 2015. Graph and Model Transformation.

Flasiński, M, & Flasińska, Z. 2014. Characteristics of Bottom-Up Parsable edNLC Graph Languages for Syntactic Pattern Recognition. *Computer Vision and Graphics*, 195–202.

Flasiński, Mariusz. 1993. On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition*, **26**(1), 1–16.

Flasiński, M. 1998. Power properties of NLC graph grammars with a polynomial membership problem. *Theoretical Computer Science*, **201**(8), 189–231.

Giese, Holger, & Wagner, Robert. 2009. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, **8**(1), 21–43.

Gilroy, Sorcha, Lopez, Adam, & Maneth, Sebastian. 2017. Parsing Graphs with Regular Graph Grammars. *Pages 199–208 of: Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (* SEM 2017)*.

Gottmann, Susann, Nachtigall, Nico, Engel, Thomas, Ermel, Claudia, & Hermann, Frank. 2016. Towards the propagation of model updates along different views in multi-view models. *CEUR Workshop Proceedings*, **1571**(Bx), 45–60.

Leblebici, Erhan, Anjorin, Anthony, & Schürr, Andy. 2014. Developing eMoflon with eMoflon. *Pages 138–145 of: International Conference on Theory and Practice of Model Transformations*. Springer.

Marriott, Kim, & Meyer, Bernd. 1997. On the classification of visual languages by grammar hierarchies. *Journal of Visual Languages and Computing*, **8**(4), 375–402.

Pratt, Terrence W. 1971. Pair grammars, graph languages and string-to-graph translations. *Journal of Computer and System Sciences*, **5**(6), 560–595.

Rekers, Jan, & Schürr, Andy. 1997. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages & Computing*, **8**(1), 27–55.

Rozenberg, Grzegorz, & Welzl, Emo. 1986. Boundary NLC graph grammars-Basic definitions, normal forms, and complexity. *Information and Control*, **69**(1-3), 136–167.

Schürr, Andy. 1994. Specification of graph translators with triple graph grammars. *Pages 151–163 of: International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer.

Shi, Z, Zeng, X, Huang, S, Li, H, Hu, B, Lei, X, & Wang, Y. 2014. Transformation between BPMN and BPEL based on graph grammar. *Computing, Communication and Networking Technologies (ICCCNT), 2014 International Conference on*, 1–6.

Shi, Zhan, Zeng, Xiaoqin, Huang, Song, Qi, Zekun, Li, Hui, Hu, Bin, Zhang, Sainan, Liu, Yanyun, & Wang, Cailing. 2016. A method to simplify description and implementation of Graph Grammars. *6th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2015*, 2–7.

Striewe, Michael. 2008. Using a triple graph grammar for state machine implementations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **5214 LNCS**, 514–516.

Zhang, Da Qian, Zhang, Kang, & Cao, Jiannong. 2001. A context-sensitive graph grammar formalism for the specification of visual languages. *Computer Journal*, **44**(3), 186–200.