A Practical Approach for Verification of Graph Transformation with Description Logic

Mohamed Chaabani
Department of Computer Science
LIMOSE Laboratory
University of Boumerdes, Algeria.
chaabani@univ-boumerdes.dz

Mohamed Mezghiche
Department of Computer Science
LIMOSE Laboratory
University of Boumerdes, Algeria.
mohamed.mezghiche@gmail.com

***Abstract*— Graphs and visual models play a central role in the modeling and meta-modeling of software systems, these models are specified using a modeling formalism, in a high-level abstraction independent of the platform, in which the focus is on the concepts rather than the implementation. This allows keeping the model, transporting it, and then transforming it into code. Several graph transformation tools have been developed to ensure efficient transformations. This transformation requires a process of verification and validation to guarantee the correction of this transformation process, of which there are different ways to checking that a software system achieves its goal. In computer science, formal methods are techniques that allow rigorous reasoning, using semantic and formal methods, to prove their validity with respect to a certain set of properties. In this sense, description logics are promising candidates for encoding graph structures and reasoning about graph transformations, they are privileged target to operationalize graph transformation tools because they have the mechanisms of reasoning or inference.**

Keywords-graph transformation; verification; Description logics; knowledge base.

I. INTRODUCTION

The main focus of this article is to introduce a new approach for verification of graph transformation with description logics.

Graphs play a central role in the simulation of different fields, covering many areas of application such as software engineering and visual languages. Moreover, graph transformation [24] or graph rewriting is a mechanism for specifying and applying transformations to graphs. The main idea behind this transformation is rule-based graph modification, to accomplish these goals, several tools are developed and used, such as Attributed Graph Grammar (AGG) [13] and ATOM3 [12].

In order to ensure a valid transformation by these applications, it is necessary to prove the correctness of the transformation, i.e, if the initial graph satisfies a given set of conditions, the graph obtained must also satisfy the same conditions.

Description Logics (DLs)[1,18,19,9] offer powerful formalisms for specifying and reasoning about graphs, most of which are decidable fragments of first-order logic. They have a formal semantics which is the basis of the reasoning service.

The approach presented in this article consists in translating or rewriting the definition of the graph represented by graph transformation tools in the description logic. This translation is provided by a transformation engine, which takes as input the file represents a graph in the tool and translates it into the syntax of the description logic. This new representation is known as a knowledge base. Then, the verification is ensured by the reasoning mechanisms of logic.

The amount of existing literature in the field of research in verification of graph transformation [28] is vast. Therefore, most of them use the model checking approach [25,26,27], the aim is to carry out a symbolic exploration of the state space, in order to determine out whether certain invariants are preserved or certain states are reachable.

Our interest in the subject stems from previous work on the formalization of the description logic [7,8,9] and the verification of graph transformations [10]. In this last article, we defined an imperative programming language for the transformation of the knowledge base which is seen as a graph structure, made up of nodes and binary relations between these nodes. A more in-depth investigation is carried out in [5].

This approach is based on logical reasoning for a rigorous and complete verification, the other approaches use partial verifications based on model checking or heuristic approaches. Another fundamental characteristic of this proposal is the feasibility of the implementation and its association with graph transformation tools such as AGG [13] .

The paper is organized as follows. Section II provides definitions of graphs and basic concepts of graph transformation. Section III presents useful notions of description logics. In section IV, we define an approach for verification of graph transformation. In Section V the approach is implemented for the AGG tool. Section VI concludes the paper.

II. GRAPHS TRANSFORMATION

Graphs are a practical, intuitive and simple way to visualize and model complex systems, examples include UML diagrams, Petri nets or Automata. Graph transformation [17] can be used to specify how these models can evolve. They have evolved as a consequence of the weakness of expressiveness in classical rewriting approaches such as Chomsky's grammars and the rewriting of terms to deal with nonlinear structures. A graph transformation consists of applying rules to a graph and iterating this process. Each rule application transforms a graph by replacing one of its parts with another graph. In other words, the graph transformation is the process of choosing a rule from a specified set, applying this rule to a graph and repeating the process until no rule can be applied. The graph transformation is specified as a graph grammar model. These are a generalization of Chomsky's grammars for graphs. They are composed of rules. A rule consists of two parts, the Left Hand Side (LHS) and the Right Hand Side (RHS). The LHS intended to be matched with the parts of the graph (called host graph) where the rule is applied. The right part of the rule, the RHS describes the modification that will be made on the host graph, it substitutes in the host graph the part identified by the left part of the rule.

A. *Principle of the transformation graphs*

The principle of a graph transformation is schematized in the Figure 1. The idea is to modify the structure of a graph by a transformation rule (or derivation). A transformation rule *p* is a pair (L, R) where L and R are graphs. More precisely, *p* is a morphism of graph from L to R. Applying the rule to a host graph G is equivalent to finding an occurrence (or match) of L in G that is replaced by the graph R to arrive at a graph H.

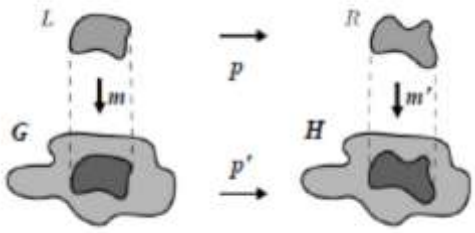

Figure 1. Principle of Graph Transformation.

B. *Approaches and Tools for Graph Transformation*

Currently, several approaches and tools have been developed for the graph transformations, we quote: VIATRA VIsual Automated model TRAnsformations [11] ATOM3 (A Tool for Multi-formalism and Meta-Modeling)[12], PROGRES (PROgrammed GRaph REwriting Systems) [16], GreAT (Graph Rewriting and Transformation) [3], Eclipse Modeling Framework (EMF) [6], Attributed Graph Grammar (AGG) [13], etc. They all allow the generation of a target graph from another host graph using a well-defined grammar.

III. DESCRIPTION LOGICS

The Description Logics DL [1, 18, 19, 9] are a family of knowledge representation and reasoning languages most of which are decidable fragments of First Order Logic "FOL". This allows for formal reasoning.
They are used for many applications. Among them we can mention: The representation of ontology languages used in the context of the Semantic Web such as OWL [2], the automatic language processing [20] and the representation of the semantics of UML class diagrams [4].

A. *Syntax*

The basic elements that are defined and manipulated by DL are concepts and roles. Description logics allow the representation of the knowledge of a domain by the mean of individuals (instances), concepts which are classes of individuals and roles that model relations between concepts.
For example, describing the domain of people and their family relationships could use concepts such as *Parent* to represent the set of all parents and *Female* to represent the set of all female persons, roles such as *parentOf* to represent the (binary) relationship between parents and their children, and individual names such as *julia* and *john* to represent the individuals *Julia* and *John*.

- Concepts : Male, Femal, Person …
- Roles: haschild, ParentOf, haswife…
- Individuals: julia, jhon, .,...

Example 1. Concepts and Roles

We recall that description logics have as a common basis *AL* (Attribute Language) enriched with different extensions: The description logic *ALC* (Attribute Language with general Complement), adds negation to *AL*. The most useful extensions of *ALC* are *ALCN* and *ALCQ*.

ALCQ adding qualified number restrictions, other logics, add the notion of sub-roles etc...
ALCQ concepts are inductively defined from a set of constructors, starting with a set *nc* of concept names, a set *nr* of role names, and (possibly) a set *ni* of individual names (all countably infinite).

Let *c* concept name, *r* role name and *n* a natural number, the data type *C* of concepts can be defined inductively by:

C ::=	$\top$	(universal concept)
	$\mid \bot$	(empty concept)
	$\mid$ c	(atomic concept)
	$\mid \neg$ C	(negation)
	$\mid$ C $\sqcap$ C	(conjunction)
	$\mid$ C $\sqcup$ C	(disjunction)
	$\mid (\geq$ n r C)	(at least)
	$\mid (<$ n r C)	(no more than)
	$\mid (\forall$ r C)	(universal quantifier)
	$\mid (\exists$ r C)	(existential quantifier)

For example, concept inclusions allow us to state that all women are female and that all females are persons, then the description of the concept Woman is *Female* $\sqcap$ *Person*. The Union (also called disjunction) is the dual of intersection. For example, the concept *Parent* can be defined as *Father* $\sqcup$ *Mother*, wish describe those individuals that are either fathers or mothers. The Top concept $\top$ is a special concept with every individual as an instance, it can be viewed as an abbreviation for $C \sqcup \neg C$ for an arbitrary concept *C*, where the Bottom concept $\bot$ is the dual of $\top$, that is the special concept with no individuals as instances; it can be seen as an abbreviation for $C \sqcap \neg C$ for an arbitrary concept *C*.
The concept *Parent* can be defined using the role *parentOf*, parent is someone who is a parent of at least one individual. In DLs, this relationship can be represented by the concept $\exists parentOf.T$. More, if we want to represent the concept parent that all his children are females, we use the universal restriction $\forall$ parentOf.Female.

B. Semantics

A semantics is provided by an interpretation *I* is essentially a couple $(\Delta_I, .^I)$ where Δ_I is called the domain of interpretation and $.^I$ is an interpretation function that maps an atomic concept A to subset A_I of a domain of interpretation Δ_I and a role *r* to subset r^I of the product $\Delta_I \times \Delta_I$. Its extension to other concept constructors is defined, in mathematical notation, as follows:

$$\top^I = \Delta^I$$
$$\bot^I = \emptyset$$
$$(C \sqcap D)^I = C^I \cap D^I$$
$$(C \sqcup D)^I = C^I \cup D^I$$
$$(\neg C)^I = \Delta^I - C^I$$
$$(\forall r.C)^I = \{x \in \Delta^I \mid \forall y : (x, y) \in r^I \rightarrow y \in C^I\}$$
$$(\exists r.C)^I = \{x \in \Delta^I \mid \exists y : (x, y) \in r^I \wedge y \in C^I\}$$
$$(\geq n\ r\ C)^I = \{x \in \Delta^I \mid \#\{y \in C^I \mid (x, y) \in r^I\} \geq n\}$$
$$(< n\ r\ C)^I = \{x \in \Delta^I \mid \#\{y \in C^I \mid (x, y) \in r^I\} < n\}$$

Table 1. Description logic semantics.

C. Knowledge Representation

Domain knowledge representation with DLs is done in two levels. The first, the terminology level or TBox, describes the general knowledge of a domain while the second, the factual level or ABox, represents a precise configuration. A TBox includes the definition of concepts and roles, while an ABox describes individuals by naming and specifying in terms of concepts and roles, assertions that relate to these named individuals. Several ABoxs can be associated with the same TBox, each represents a configuration made up of individuals, and uses the concepts and roles of the TBox to express it. DLs offer an extra feature that permit to attribute names to the complex concepts and describe relationships between them. These relationships are presented in the form of axioms called terminological axioms. More specifically, if C and D are DL concepts then the terminological axioms have the form, $C \sqsubseteq D$ or $C \equiv D$. The first is called subsumption or inclusion axiom, while the second is called equivalence axioms. Intuitively, an entry form $C \sqsubseteq D$ denotes that the concept D is more general than C (otherwise C is a subconcept of D). The equivalent $C \equiv D$ denotes that the two concepts are equivalent. A TBox is simply a finite set of subsumption or equivalence axioms.

- *Woman* $\equiv$ *Female* $\sqcap$ *Person*
- *Woman* $\sqsubseteq$ *Female*
- $\top \sqsubseteq$ *Male* $\sqcup$ *Female*
- *Parent* $\equiv$ *Father* $\sqcup$ *Mother*
- *Male* $\sqcap$ *Female* $\sqsubseteq \bot$

Example 2. Terminological axioms.

An assertional axiom (called also *Facts*) makes assertions about an instance being an element of a concept, and about being in a relation. In DL, facts are elements of an A Box. For a concept *C*, a role name *r* and *x* and *y* are individual's variable names, the type of *facts* is defined as follows:

fact ::=		
	$x : C$	(instance of concept)
	$\mid x\ r\ y$	(instance of role)
	$\mid x = y$	(equality of instances)
	$\mid x \neq y$	(inequality of instances)

We extend the interpretation of concepts to construct the interpretation of fact. The interpretation of fact is boolean value, defined in according with the syntax of fact:

$(x : C)^I = x^I \in C^I$
$(x\ r\ y)^I = (x^I, y^I) \in r^I$
$(x = y)^I = (x^I = y^I)$
$(x \neq y)^I = (x^I \neq y^I)$

- *julia* : Mother or (*Mother*(*julia*))
- *john parentOf julia* or (*parentOf*(*julia*,*john*))
- *julia* ≠ *john*

Example 3. Assertional axioms

The reasoning in DL makes it possible to infer knowledge represented implicitly from other explicit contents in knowledge bases. Two paths have been mainly explored so far: normalization-comparison algorithms, and a method derived from the method of semantic tableau in classical logic, a semantic tableau is a procedure that allows building an interpretation that satisfies the assertion of a given concept. Derivations can be established by applying a set of decomposition rules.

IV. Verification of Graph Transformations Approach

The aim of this study consists of modeling a graph transformation system (host graph, target graph) in the description logic by creating a KB (Knowledge Base); using a description logic reasoning to verify that the properties are preserved during the transformation process.

Figure 2. Components of the Approach

The main concepts of our approach are summarized by a scenario shown in Figure 3. We have an input model which represents the graph transformation system environment and an output model which represents the knowledge base in description logic. A transformation is performed using a transformation engine, which is defined by different functions. Most tools of graph transformations system structure and organize a set of graph information in a file in well-defined structures of XML family, GraphML, Ggx, XGMML, GraphXML and GXL. This representation allows the elements that represent the graph to be easily extracted.The transformation engine is built in three stages, data extraction, transformation and loading.

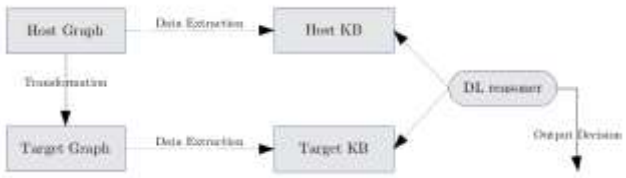

Figure 3. Approach Architecture

In the following, we will detail our procedure.

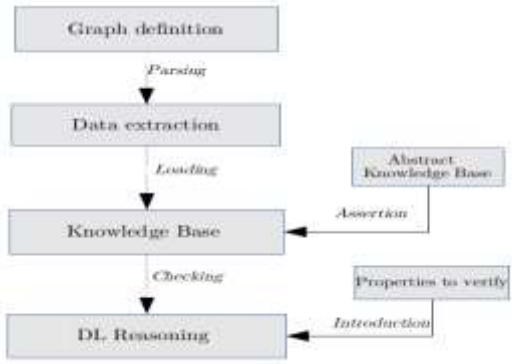

Figure 4. Verification procedure

A. Data Extraction

Data extraction is a process of collecting or retrieving data from the source representation, which defines the graph structure. it can consolidate, process and refine data, then store it before using it.
This data extraction process is carried out by translation tools using a translator or a parser; it allows to browse the graph file.

B. Construction of Knowledge Base

Knowledge bases (ABox and TBox) are made up of two fragments, a static fragment and a dynamic fragment.

A static fragment (also called signature) specifies the graph at the abstract level, (Meta specification), it defines valid axioms for any definition of a graph, this fragment is predefined and can't be changed during runtime.

A TBox can contain these axioms

- Graph ⊑ ⊤
- Host_Graph ⊑ Graph
- Target_graph ⊑ Graph
- Node ⊑ Graph

On the other hand, the dynamic part is inserted from the definition of the graph provided by the parser.

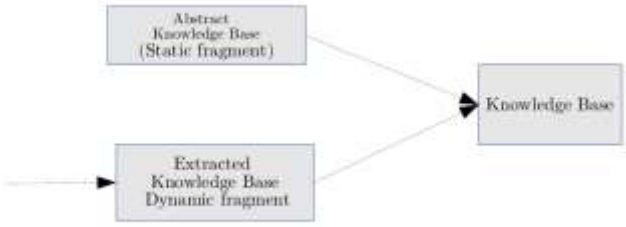

Figure 5. Construction of KB

C. *Reasoning*

Once the construction of the knowledge base is completed, the process of logical reasoning carried out the verification, according to the properties required for the correction of the transformation.
It is therefore trivial to write these requests in the syntax of DL, this step is ensured by tools used for this purpose, such as Fact ++, Pellet or Racer.

V. IMPLANTATION FOR AGG TOOL

AGG (Attributed Graph Grammar) is a tool for transforming typed and attributed graphs. It is considered one of the general purpose tools for transforming attributed graphs and the most widely used and cited tool in the field. AGG was developed for the purpose of implementing the Single-PushOut (SPO) approach and Double-PushOut (DPO) approach. It offers a visual framework for defining rewrite rules in a graphic and simple way. It also defines strategies for implementation of these rules with priority levels mechanism (layers). AGG saves the typed graph, the host graphs and the rewrite rules in the same GGX format file. It is important to note that AGG offers a Java API allowing its integration into Java applications.

In the following, we will detail our procedure written with the AGG tool.

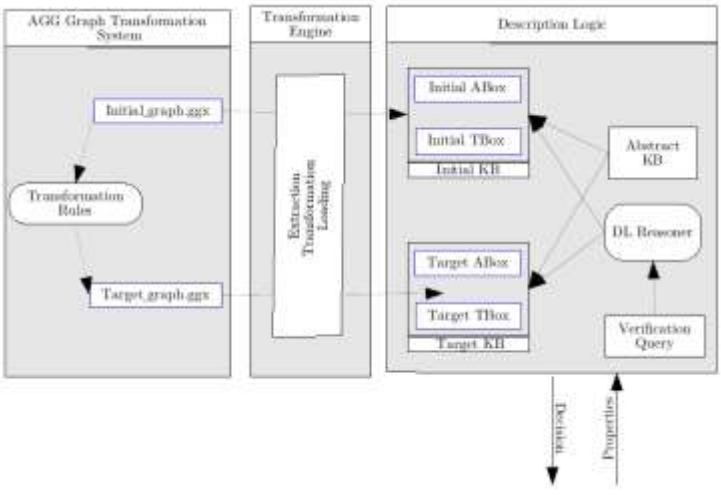

Figure 6. Structure of Verification Tool

A. *Data Extraction*

The first step is to extract the constituent elements of our transformation system using a parser,
The knowledge base is extracted from XML data from AGG file using DOM.

The Document Object Model (DOM) is W3C specification for proposing an API for HTML and XML documents. It determines the logical structure of documents and allows to model, browse and manipulate an XML document. In the DOM specification, XML is increasingly used to represent any type of information stored on any type of system. Most of them are traditionally seen as data rather than documents. However, XML represents this data as documents, and the DOM can be used to manage this data.
The main role of DOM is to provide a memory representation of an XML document in the form of a tree of objects and to allow its manipulation (browsing, search and update).

B. *Reasoning*

The verification of the correctness of the graph transformation is guaranteed by the reasoning mechanism of the DL reasoner, the knowledge base which is made up of assertional and terminological axioms is specified in the language of the description logic according to the syntax of this reasoner.
Pellet is among the most widely used reasoners. Open source Java-based reasoner for *SROIQ* with simple datatypes. It implements a tableau-based decision procedure and has an interface that allows their connection with this application.

VI. CONCLUSION

This document introduced an approach and a tool for the verification of graph transformations, this approach is considered as a logical verification layer based essentially on the extraction of the definition of the graph in the description logic.
The implemented tool based on this approach can extract and specify data from the XML file of the AGG graph transformation system in a knowledge base. The graph is therefore represented in the form of a knowledge base and verification is ensured by the reasoning mechanism of the description logic, which is the most suitable formalism for the representation and reasoning on knowledge. Several reasoners are implemented such as *FaCT++ [21]*, *RACER [22]* and *Pellet[23]*, with such reasoners we check the preservation of properties during the graph transformation process, more precisely, if a property is checked in the host graph it must also be checked in the target graph.

VII. FUTURE WORK

We will consider several extensions of this work, among which we can mention is the representation of rewriting rules of the graph transformation system in the knowledge base of description logic. This representation makes it possible to check the correctness of the transformation

system independently of the instances. The generated base represents the functional part of the rewrite system. Another possible extension is to generalize the operation of this tool on other systems such as Groove and adapt it to be compatible with other formats.

REFERENCES

[1] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F., 2007. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[2] Baader, F., Horrocks, I., Sattler, U., 2005. Description Logics as Ontology Languages for the Semantic Web. Vol. 2605/2005 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 228–248.

[3] Balasubramanian, D., Narayanan, A., van Buskirk, C., & Karsai, G. (2007). The graph rewriting and transformation language: GReAT. *Electronic Communications of the EASST*, *1*.

[4] Berardi, D., Calvanese, D., & De Giacomo, G. (2001, September). Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics* (Vol. 44).

[5] Brenas, J. H., Echahed, R., & Strecker, M. (2018, June). Verifying graph transformation systems with description logics. In *International Conference on Graph Transformation* (pp. 155-170). Springer, Cham.

[6] Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., & Weiss, E. (2006, October). Graphical definition of in-place transformations in the eclipse modeling framework. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 425-439). Springer, Berlin, Heidelberg.

[7] Chaabani, M., Mezghiche, M., Strecker, M., Dec. 2009. Formalisation de la logique de description ALC dans l'assistant de preuve Coq. In: Bellatrache, L., Kassel, G., Thiran, P. (Eds.), Proc. 3es Journées francophones sur les ontologies. pp. 149–163.

[8] Chaabani, M., Mezghiche, M., Strecker, M., Jun. 2010. Vérification d'une méthode de preuve pour la logique de description ALC. In: Ait-Ameur, Y. (Ed.), Proc. 10ème Journées Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL). pp. 149–163.

[9] Chaabani, M., Mezghiche,M and Strecker M., "Formal verification of a proof procedure for the description logic ALC." *arXiv preprint arXiv:1307.8211* (2013).

[10] Chaabani, M., Echahed, R., Strecker, M.: Logical foundations for reasoning about transformations of knowledge bases. In: Eiter, T., Glimm, B., Kazakov,Y., Krötzsch, M. (eds.) DL–Description Logics. CEUR Workshop Proceedings,vol. 1014, pp. 616–627.

[11] Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., & Varró, D. (2002, September). VIATRA-visual automated transformations for formal verification and validation of UML models. In *Proceedings 17th IEEE International Conference on Automated Software Engineering*, (pp. 267-270). IEEE.

[12] De Lara, J., & Vangheluwe, H. (2002, April). AToM 3: A Tool for Multi-formalism and Meta-modelling. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 174-188). Springer, Berlin, Heidelberg.

[13] Ermel, C., Rudolf, M., & Taentzer, G. (1999). The AGG approach: Language and environment. In *Handbook Of Graph Grammars And Computing By Graph Transformation: Volume 2: Applications, Languages and Tools* (pp. 551-603).

[14] Ehrig, H., Engels, G., Kreowski, H. J., & Rozenberg, G. (1999). *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*.

[15] Strecker, M. (2008). Modeling and verifying graph transformations in proof assistants. *Electronic Notes in Theoretical Computer Science*, *203*(1), 135-148.

[16] Zündorf, A. (1994, November). Graph pattern matching in PROGRES. In *International Workshop on Graph Grammars and Their Application to Computer Science* (pp. 454-468). Springer, Berlin, Heidelberg.

[17] Andries, M., Engels, G., Habel, A., Hoffmann, B., Kreowski, H. J., Kuske, S., ... & Taentzer, G. (1999). Graph transformation for specification and programming. *Science of Computer programming*, *34*(1), 1-54.

[18] Baader, F., Horrocks, I., Lutz, C., & Sattler, U. (2017). *Introduction to description logic*. Cambridge University Press.

[19] Krötzsch, M., Marx, M., Ozaki, A., & Thost, V. (2017, October). Attributed description logics: Ontologies for knowledge graphs. In *International Semantic Web Conference* (pp. 418-435). Springer, Cham.

[20] Gyawali, B., Shimorina, A., Gardent, C., Cruz-Lara, S., & Mahfoudh, M. (2017, May). Mapping natural language to description logic. In *European Semantic Web Conference* (pp. 273-288). Springer, Cham.

[21] Tsarkov, D., & Horrocks, I. (2006, August). FaCT++ description logic reasoner: System description. In *International joint conference on automated reasoning* (pp. 292-297). Springer, Berlin, Heidelberg.

[22] Haarslev, V., & Möller, R. (2001, June). RACER system description. In *International Joint Conference on Automated Reasoning* (pp. 701-705). Springer, Berlin, Heidelberg.

[23] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, *5*(2), 51-53.

[24] König, B., Nolte, D., Padberg, J., & Rensink, A. (2018). A tutorial on graph transformation. In *Graph Transformation, Specifications, and Nets* (pp. 83-104). Springer, Cham.

[25] Rensink, A., Schmidt, Á., & Varró, D. (2004, September). Model checking graph transformations: A comparison of two approaches. In *International Conference on Graph Transformation* (pp. 226-241). Springer, Berlin, Heidelberg.

[26] Varró, D. (2004). Automated formal verification of visual modeling languages by model checking. *Software & Systems Modeling*, *3*(2), 85-113.

[27] Padberg, J., & Schulz, A. (2016, July). Model checking reconfigurable Petri nets with Maude. In *International Conference on Graph Transformation* (pp. 54-70). Springer, Cham.

[28] Schneider, S., Dyck, J., & Giese, H. (2020, June). Formal verification of invariants for attributed graph transformation systems based on nested attributed graph conditions. In *International Conference on Graph Transformation* (pp. 257-275). Springer, Cham.