On Synergies between Model Transformations and Semantic Web Technologies

Robert Bill[1], Simon Steyskal[1,2], Manuel Wimmer[1], and Gerti Kappel[1]

[1] Vienna University of Technology, Austria
[lastname]@big.tuwien.ac.at
[2] Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria

Abstract. The integration of heterogeneous data is a reoccurring problem in different technical spaces. With the rise of model-driven engineering (MDE), much effort has been spent in developing dedicated transformation languages and accompanying engines to transform, compare, and synchronize heterogeneous models. At the same time, ontologies have been proposed in the Semantic Web area as the main mean to describe the intension as well as the extension of a domain. While dedicated languages for querying and reasoning with ontologies have been intensively studied, specific support for integration concerns leading to executable transformations is rare compared to MDE.
Based on previous studies which relate metamodels and models to ontologies, we discuss in this paper synergies between transformation languages of MDE, in particular Triple Graph Grammars (TGGs), and Semantic Web technologies (SWTs), namely OWL/SPARQL. First, we show how TGGs are employed to define correspondences between ontologies and how these correspondences are expressed in SPARQL. Second, we show how reasoning support of SWTs is applied to allow for underspecified model transformation specifications as well as how the different assumptions on existing knowledge effect transformations. We demonstrate these aspects by a common case study.

Keywords: Model Transformation, Model Integration, Triple Graph Grammars, OWL, SPARQL

1 Introduction

The integration of heterogeneous data has first emerged in the database area [26]. However, data integration is a reoccurring problem, not only in the database area, but in different technical spaces [8, 11]. With the raise of model-driven engineering (MDE), much effort has been spent in developing dedicated transformation languages and accompanying engines to transform, compare, and synchronize heterogeneous models.

At the same time, ontologies have been proposed in Semantic Web to describe the intension as well as the extension of a domain. While dedicated languages for querying and reasoning with ontologies have been intensively studied (e.g., classification of individuals and consistency checking are provided by standard reasoner), specific support for integration concerns leading to executable transformations is rare.

In order to understand the differences and commonalities between MDE and Semantic Web technologies (SWTs), several studies have investigated about the languages used in both fields to describe the domain of discourse [28]. Thus, bridges are already available between these two worlds for transforming metamodels and corresponding models to ontologies and vice versa. Some studies go also beyond purely structural information [12], but bridges concerning dynamic information are still mostly unexplored. Moreover, for specific domains, such as configuration management [4], both technologies are applied, but mostly in an isolated manner as we currently explore in a recent project[3].

Based on previous studies which relate metamodels and models to ontologies [5,10], we discuss in this paper synergies between transformation languages of MDE, in particular Triple Graph Grammars (TGGs) [24], and SWTs, namely a combination of OWL/SPARQL. First, we show how TGGs are employed to define correspondences between ontologies visualized as metamodels and how these correspondences are operationalized by a compilation of TGGs to OWL/SPARQL. Second, we show how reasoning support of SWTs is applicable to allow for underspecified model transformation specifications, i.e., the concrete types of instances are assigned in a post-processing step using OWL reasoner. Third, we discuss how switching between the closed world assumption (CWA) to an open world assumption (OWA) is beneficial for particular integration scenarios where only partial knowledge of existing models is present. We demonstrate these aspects by a common case study.

The rest of this paper is structured as follows. In the next section, we introduce the running example for this paper as well as the technological prerequisites. In Section 3, we discuss the mapping between TGGs and OWL/SPARQL in a general form, whereas in Section 4 we demonstrate the compilation of TGGs to OWL/SPARQL by-example and discuss how features of ontologies may be exploited for model transformations. In Section 5 we discuss related work before we conclude in Section 6.

2 Preliminaries

2.1 Motivating Example

As an example we will illustrate how heterogeneous views on computer networks can be joined (cf. Fig. 1). The first view comprises the physical network structure including cables of various types and speed as well as computers. The second view contains the application structure of the network, i.e., various computers are running different services which might require each other. In that case, a connection to a matching service is required. Connections can be modeled by their physical structure, as well as their logical network structure. By using TGGs we are able to define correspondences between both structures, which can be used to either (i) express graph transformation rules to transform individuals from one schema to another or (ii) check whether or not such alignments hold for given models. Extending those correspondence definitions with SWTs, allows even more sophisticated reasoning, inferencing, and querying tasks.

[3] `http://cosimo.big.tuwien.ac.at`

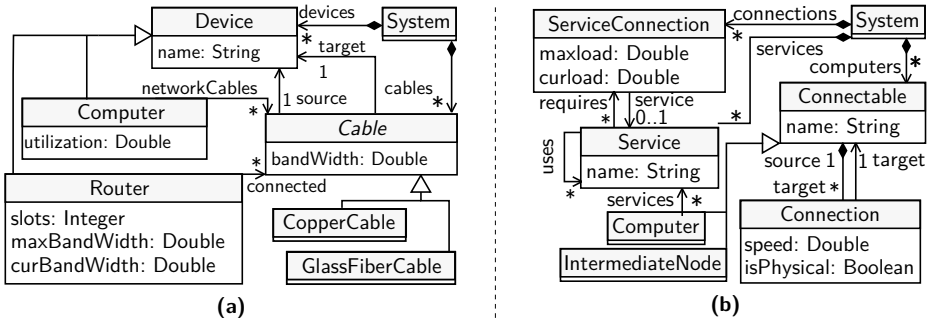


Fig. 1. Representations of networks: (a) physical and (b) logical

2.2 Triple Graph Grammars (TGGs)

TGGs have first been introduced by Andy Schürr [24]. A TGG rule combines elements from a left model (LM), a right model (RM) and a correspondence model (CM). Each TGG rule contains a left hand side graph (LG) conforming to LM, a right hand side graph (RG) conforming to RM and a correspondence graph (CG) conforming to CM which connects elements from LG and RG. Vertices and edges may not be deleted by any rule, they can only be preserved or created. In contrast to usual graph transformation rules, TGG rules are inherently bidirectional. A TGG engine searches for rule applications creating the required input graphs and creates elements of all other graphs during that process. For example, in a transformation scenario an input graph for LM would result in a graph for CM and RM. TGG rules might also have additional constraints, e.g., negative application conditions or attribute constraints, also restricting the value of an attribute depending on attribute values of other objects in the TGG rule.

The left-hand side of Fig. 5 shows a simple example of a TGG rule of a computer network that relates connections of the same speed without declaring them equal. Black elements denote elements which have been matched already, green elements are elements which are matched or created then. In this case, the difference between both models is only a syntactical one.

2.3 Semantic Web Technologies (SWTs)

RDF & OWL. The Resource Description Framework (RDF)[4] is a framework to describe and represent information about resources and is both human-readable and machine-processable, which enables the possibility to easily exchange information among different applications using RDF triples.

In RDF everything is a resource, uniquely identified by its URI and all data is represented as $(subject, predicate, object)$ triples, where subjects and predicates are URIs and objects can either be literals (strings, integers, ...) or URIs.

Since RDF itself does not contain sophisticated semantics to express characteristics of concepts or to define more expressive relationships among them, the Web Ontology

[4] `http://www.w3.org/TR/rdf-mt/`

Language (OWL)[5] was developed. Introducing OWL allows the usage of reasoning systems such as Pellet [27], FaCT++ [29] or HermiT [25] to (i) infer new knowledge and (ii) detect inconsistencies based on the modeled semantics.

SPARQL. The Protocol And RDF Query Language (SPARQL) is basically the standard query language for RDF[6]. Its syntax (cf. Figs. 2-4) is highly influenced by an RDF serialization format called Turtle [1] and SQL. In its current version, SPARQL allows besides basic query operations such as union of queries, filtering, sorting and ordering of results as well as optional query parts, the use of aggregate functions (`SUM, AVG, MIN, MAX, COUNT, ...`), the possibility to use subqueries, perform update actions via SPARQL Update and several other requested features as indicated in [18].[7]

Another important feature of SPARQL are CONSTRUCT queries, which allow the construction of new RDF graphs based on a previously matched (against one ore more input graphs) SPARQL graph pattern (cf. Fig. 3). Their main purpose lies in data integration scenarios, where data from one ore more data sources have to be normalized to fit a common schema [19, 23].

```
SELECT ?a ?name
WHERE {
  ?a a :Device .
  ?a :name ?name .
}
```

Fig. 2. SELECT Query

```
CONSTRUCT {
  ?a :hasName true .
}
WHERE {
  ?a a :Device .
  ?a :name ?name .
}
```

Fig. 3. CONSTRUCT Query

```
ASK WHERE {
  ?a a :Device .
  ?a :name ?name .
}
```

Fig. 4. ASK Query

3 Aligning TGGs and SWTs

In the following, we map basic TGG rules to SPARQL queries. The TGG rule definition is based on [2] and extended by labels. Advanced features of graph transformations like multi–nodes are not supported. Subsequently, we discuss two benefits of using SWTs: (i) automatic type inference and (ii) reasoning under CWA and OWA.

Definition 1 (E–Graph)**.** A labeled E–Graph $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (src_j, trg_j)_{j\in\{G,EA,EA\}}, l)$ consists of graph vertices V_G, data vertices V_D, graph edges, node attribute edges and edge attribute edges E_G, E_{NA}, E_{EA}, source and target edge functions src_j and trg_j mapping edges to corresponding vertices and a labeling function l defining a label for each vertex and edge. As shorthand we use $src(e)$ and $trg(e)$ to denote source and target edge functions operating on edges of any kind.

Definition 2 (Typegraph)**.** A typegraph TG is an E-Graph specifying the relation between types. A type morphism t maps nodes and edges of a graph to the corresponding nodes and edges in the type graph.

[5] `http://www.w3.org/TR/owl2-overview/`
[6] `http://www.w3.org/TR/sparql11-overview/`
[7] For a comprehensive overview on the semantics of SPARQL queries see [17, 22].

Definition 3 (Triple graph)**.** A triple graph $TRG = (LG \xleftarrow{m_s} CG \xrightarrow{m_t} RG)$ consists of three E–Graphs LG, CG and TG and morphisms m_s and m_t mapping corresponding nodes from the correspondence graph to other graphs.

Definition 4 (TGG rule)**.** A parametrized TGG rule $TGG = (SG, ZG, ac)$ consists of a source triple graph SG, a target triple graph $ZG \supset SG$, and application conditions ac. For the sake of simplicity, we consider application conditions as boolean formulas using $\wedge$ and $\vee$, over atomic positive application conditions (PACs) and negative application conditions (NACs) defined as triple graphs which might share vertices with each other and SG and TG.

Example 1. The TGG rule $r2n = (tr_s, tr_t, \text{true})$ represented in Fig. 5 could be specified as follows: The type graph of both models is derived from the metamodel. For example, a subset of the typegraph for `network1` could be specified as $G_{n1} = (\{d_{n1}, c_{n1}\}, \{double\}, \{s_{n1}, t_{n1}\}, \{b_{n1}\}, \{\}, \{(s_{n1}, c_{n1}), (t_{n1}, c_{n1})\}, \{(s_{n1}, d_{n1}), (t_{n1}, d_{n1})\}), \{(b_{n1}, c_{n1})\}, \{(b_{n1}, double)\}, \{\}, \{\})$ combined with a standard double DSIG algebra.

The source graph of TGG rule $tr_s = (LG_s \xleftarrow{m_{ss}} CG_s \xrightarrow{m_{ts}} TG_s)$ with $LG_s = (\{d_1, d_2\}, \emptyset, \ldots, \emptyset)$, $CG_s = (\{g2c_1, g2c_2\}, \emptyset, \ldots, \emptyset)$, $TG_s = (\{cb_1, cb_2\}, \emptyset, \ldots, \{\})$, $m_{ss} = \emptyset$, $m_{ts} = \emptyset$. The target graph $tr_t = (LG_t \xleftarrow{m_{st}} CG_t \xrightarrow{m_{tt}} TG_t)$ with $LG_t = LG_s \cup (\{c_1\}, \{n\}, \{s_1, t_1\}, \{b_1\}, \emptyset, \{(s_1, c_1), (t_1, c_1)\}, \{(s_1, d_1), (t_1, d_2)\}, \{(b, c_1)\}, \{(b, n)\}, \emptyset, \emptyset)$, $CG_t = CG_s \cup (\{\}, \emptyset, \ldots, \emptyset)$, $RG_t = RG_s \cup (\{c_1\}, \{n\}, \{s_1, t_1\}, \{b_1\}, \emptyset, \{(s_1, c_1), (t_1, c_1)\}, \{(s_1, cb_1), (t_1, cb_2)\}, \{(sp, c_1)\}, \{(sp, n)\}, \emptyset, \emptyset)$. The labeling function assigns the type names from the metamodel in Fig. 1 to elements of the type graph and lables equal to variable name to elements of other graphs.

Mapping TGG rules to SPARQL queries. Both transformation and correspondence construction can be expressed using SPARQL queries in the merged ontology using the labeling function. If a model should be synchronized, at first the maximum correspondence is searched, then for the unmatched elements a transformation might be conducted. Each vertex and edge in a TGG rule can be used as context, matched or created element, depending on the use of the TGG rule. In every case, the context nodes are exactly those occurring in the source model, but not in the target one. In a transformation, elements of source model of the transformation in the target rule, but not in the source, are used as matched elements while the others are used as created elements. In a corresponding search scenario, only the elements of the correspondence graph are created, all others are matched. According to [7], an attribute `hasMatch` is introduced to specify which elements have been matched already. For elements, it is initially unset and may be set to `true`. For edges, it has the same domain and range as the edge to match, but `hasMatch_` as prefix to the original name. Thus, it is not necessary to modify the ontology prior to using the converted TGG rules.

In many cases, TGG rules as defined previously can be transformed to corresponding SPARQL queries. Table 1 shows matching concepts. The general structure of a generated SPARQL query is `CONSTRUCT <transformed created elements> WHERE <context elements>`. The first two lines, e.g., indicate that the occurrence of a vertex n labeled *c1* with a type labeled *computer* in the TGG rule should

P.	Type	TGG	SPARQL
W	c,m	Vertex n	?$l(n)$ a $l(t(n))$.
W	c,m	Vertex n_1, n_2, $n_1 \neq n_2$, injective matching	FILTER (?$l(n_1)$!= ?$l(n_2)$).
C	cr	Vertex n	?$l(n)$ a $l(t(n))$.
W	c,m	Edge e	?$l(s(e))$ $dom(e)$:$l(e)$?$l(t(n))$.
C	cr	Edge e	?$l(s(e))$ $dom(e)$:$l(e)$?$l(t(n))$.
C	m,cr	Vertex n	?$l(n)$ tgg:hasMatch true.
C	m,cr	Edge e	?$l(s(e))$ tgg:hasMatch_$dom(e)$_$l(e)$?$l(t(e))$.
W	m	Vertex n	FILTER NOT EXISTS {?$l(n)$ tgg:hasMatch true.}
W	m	Edge e	FILTER NOT EXISTS {?$l(s(e))$ tgg:hasMatch_$dom(e)$_$l(e)$?$l(t(e))$}
W	c	Vertex n	FILTER EXISTS {?$l(n)$ tgg:hasMatch true.}
W	c	Edge e	FILTER EXISTS {?$l(s(e))$ tgg:hasMatch_$dom(e)$_$l(e)$?$l(t(e))$}
C	m,cr	Mapping $e_1 \mapsto e_2$ from m_{ss}, m_{st}, tt or ts	?$l(e_1)$ owl:sameAs ?$l(e_2)$
C	m,cr	Mapping $e_1 \mapsto e_2$ from m_{ss}, m_{st}, tt or ts	?$l(e_1)$ owl:sameAs ?$l(e_2)$
W	-	Atomic PAC ac	BIND (EXISTS {<expand ac acc. to Tab. 1>}) AS ?$gn(ac)$
W	-	Atomic NAC ac	BIND (NOT EXISTS {<expand ac acc. to Tab. 1>}) AS ?$gn(ac)$
W	-	Full AC $ac = ac_1(\wedge\|\vee)ac_2$	FILTER (?$gn(ac_1)$ (&&/||) ?$gn(ac_2)$) or rather applied recursively until the atomic operations.

Table 1. SPARQL patterns occuring in the WHERE (W) and CONSTRUCT (C) part for context (c), matching (m) and created (cr) elements

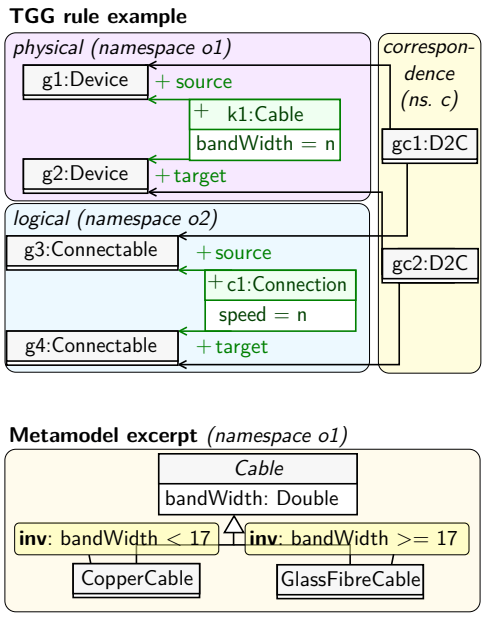
result in c1 a computer., which is placed in the CONSTRUCT part of the SPARQL query for context nodes and nodes to be matched and in the WHERE part for created nodes.

The helper function dom returns the graph for an element. It returns o1 for elements of the left graph, o2 for elements of the right graph and c for elements of the correspondence graph. The helper function gn assigns a unique name to each atomic application condition.

Many current TGG implementations allow the use of functions to set values for attributes. Existing approaches for converting OCL into SPARQL, could be used for functions requiring matched nodes, context nodes and, for created nodes, (other) created nodes. The result then can be assigned using BIND. Some constraints on created nodes might be formulated using SWRL expressions, e.g., the subset of OCL defined in [12]. In this case, the specific match of vertices in a TGG rule has to be stored to be able to subsequently apply the constraint. Thus, attributes :hasSwrlRule_$name$_$index$ might be set true to specify that an element is used as element nr. $index$ in the SWRL rule $name$. In such a way, not only conditions in the TGG can be formulated, but also invariants of each individual model or the merged model. Simple invariants may also be modeled directly in OWL. In the following, we will show an example to illustrate how this may help reducing the complexity of TGG rules significantly.

Automatic Type Inference. Considering the two views on an imaginary network, one might distinguish the creation of `Coppercables` and `Glassfibrecables` based on the `speed` of a correlating `Connection` between two previously aligned `Devices` and `Connectables` (e.g. creating a `Glassfibrecable` if the `speed` exceeds a certain threshold or a `Coppercable` otherwise). Although such distinctions can be modeled with TGGs, at least two TGG rules (in our case; one matching `Coppercable` and one matching `Glassfibrecable`) would be necessary.

A more convenient way to model such a behavior can be achieved by *out-sourcing* the type inference to OWL reasoners and only define one TGG rule, which describes the more general `Cable` and `Connection` correspondence as depicted in Figure 5 (with its corresponding SPARQL CONSTRUCT query). The constraints itself (i.e., defining the concept `Glassfibrecable` to be equivalent to an anonymous concept which is defined as `Cable` having a `speed` with a value over 17) can be directly modeled within the ontology using OWL axioms[8] and were generated during the initial model to ontology transformation .




```
CONSTRUCT {
 ?k1 a o1:Cable .
 ?k1 o1:source ?g1 .
 ?k1 o1:target ?g2 .
 ?k1 o1:bandWidth ?n .
 ?k1 tgg:hasMatch true .
 ?c1 tgg:hasMatch true .
} WHERE {
 ?gc1 a c:D2C. ?gc2 a c:D2C.
 ?g1 owl:sameAs ?gc1 .
 ?g2 owl:sameAs ?gc2 .
 ?g3 owl:sameAs ?gc1 .
 ?g4 owl:sameAs ?gc2 .
 ?c1 o2:source ?g3 .
 ?c1 o2:target ?g4 .
 ?c1 o2:speed ?n .
 FILTER (?gc1 != ?gc2) .
 FILTER (?g1 != ?g2) .
 FILTER (?g3 != ?g4) .
 FILTER EXISTS
   {?gc1 tgg:hasMatch true} .
 FILTER EXISTS
   {?gc2 tgg:hasMatch true} .
 FILTER NOT EXISTS
   {?c1 tgg:hasMatch true} .
}
```

Fig. 5. CONSTRUCT query which generates `Cables` for given `Connections`

Reasoning under Open and Closed World Assumption. One of the major benefits of SWTs for integration scenarios are their well defined semantics and the extensive reasoner support as already discussed previously. With OWL and OWL reasoners it is

[8] cf. [6] for a comprehensive list of OWL axioms

e.g., possible to describe cardinality constraints, perform automatic type inferencing as discussed above and to check for inconsistency in the given models.

While Semantic Web languages are based on OWA (i.e., if a statement is not explicitly stated, it does not mean that it does not exist), software engineering languages are mostly based on CWA (i.e., if a statement is not present, it does not exist) [20]. To deal with this issue, we translate parts[9] of the constraints expressed as OWL axioms into SPARQL queries and query for the presence of individuals which violate those constraints. E.g., consider the cardinality constraint `computers exactly 2 Computer` for concept `System` expressed in *OWL Manchester Syntax*[10]. The answer to the question whether or not a particular `System` has the right amount of `Computers`, would not be directly decidable for the OWA but for the CWA with the support of SPARQL as depicted in Listing 1.

```
ASK WHERE {
 { SELECT (count(?b) AS ?number) ?a WHERE {
     ?a a :System .
     ?a :computers ?b . } GROUP BY ?a }
 FILTER(?number != 2)}
```

Listing 1. ASK Query which returns `true` if a `System` has not exactly 2 `Computers`

4 Related Work

We discuss three lines of related work: (i) approaches for bridging models and ontologies, (ii) approaches for transforming transformations to SWTs, and (iii), approaches directly using SWTs to encode model transformations.

Bridging models and ontologies. Combining modeling approaches steaming from MDE with ontologies has been studied in the last decade [5]. There are several approaches to transform Ecore-based models to OWL and back, e.g., cf. [9, 31]. In addition, there exist approaches that allow for the definition of ontologies in software modeling languages such as UML by using dedicated profiles [13]. Moreover, there are also approaches which combine the benefits of models and ontologies such as done in [14, 16]. Not only the purely structural part of UML is considered, but some works also target the translations of constraints between these two technical spaces by using an intermediate format [3]. We build on these mentioned approaches, but we focus on correspondence definitions and their execution as transformations.

Transforming transformations to SWTs. Concerning the definition and execution of model transformations based on SWTs, we are aware of two approaches. First, [15] propose the usage of an ATL-inspired language for defining mappings between ontologies. Thus, uni-directional transformations are implementable for ontologies as it is known from model transformations. Another approach is presented in [30] which translates parts of ATL transformations to ontologies for checking the consistency of transformation rules, e.g., overlaps between rules in terms of overlapping matches. In our work, we follow this line of research, but we consider bi-directional transformations specified in TGGs. Thus, in our translations to ontologies we have to consider not only source to

[9] The decision, which constraints have to be translated, highly depends on the respective integration scenario.

[10] `http://www.w3.org/TR/owl2-manchester-syntax/`

target transformations, but we have to encode comparison and synchronization transformations as well in SPARQL.

Specifying transformations with SWTs. Finally, there are approaches which shift the definition of the model transformations to the SWTs. For instance, in [21] it is proposed to use SWRL to define the correspondences between models to allow for model synchronization. In [10], ontology matching tools are applied to search for correspondences between metamodels and to derive from these correspondences model transformations. In the context of this work, we have the assumption that correspondences are defined based on models using TGGs, but at the same time we explored which benefits from ontology reasoning may be transferred to model transformation approaches.

5 Conclusion and Further Work

In this paper we have outlined an initial mapping between TGGs and OWL/SPARQL. Especially, new features of the latest SPARQL version helped in defining a comprehensive mapping between these languages. Moreover, we also explored how reasoning capabilities can be leveraged for underspecified model transformations.

While the initial results of applying our approach seem promising, both from a mapping point of view and usage of reasoning capabilities for model transformations, further investigation are planned such as considering a mapping between TGGs and SWRL. Empirical studies are planned as well in the area of configuration management together with our industry partner Siemens AG Österreich. In particular, for performing distributed configuration management [4] where several different models and reasoners have to be connected, we plan to apply our approach to provide the necessary integration means.

Acknowledgment: This work has been funded by the Vienna Business Agency (Austria), in the programme ZIT13 plus, within the project COSIMO (Collaborative Configuration Systems Integration and Modeling) under grant number 967327.

References

1. David Beckett and Tim Berners-Lee. Turtle-terse RDF triple language. *W3C Team Submission*, 14, 2008.
2. Juan de Lara, Roswitha Bardohl, Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. Attributed graph transformation with node type inheritance. *Theoretical Computer Science*, 376(3):139–163, 2007.
3. Dragan Djuric, Dragan Gasevic, Vladan Devedzic, and Violeta Damjanovic. A UML Profile for OWL Ontologies. In *Proc. of MDAFA*, pages 204–219, 2004.
4. Andreas A. Falkner, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. Modeling and solving technical product configuration problems. *AI EDAM*, 25(2):115–129, 2011.
5. Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Engineering and Ontology Development (2. ed.)*. Springer, 2009.
6. OWL Working Group. OWL 2 Web Ontology Language. *W3C recommendation*, 2012.
7. Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, Yingfei Xiong, Susann Gottmann, and Thomas Engel. Model synchronization based on triple graph grammars: correctness, completeness and invertibility. *SoSyM*, pages 1–29, 2013.
8. Zhenjiang Hu, Andy Schürr, Perdita Stevens, and James F. Terwilliger. Dagstuhl seminar on bidirectional transformations (bx). *SIGMOD Record*, 40(1):35–39, 2011.

9. Gerti Kappel, Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, Wieland Schwinger, and Manuel Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In *Proc. of MODELS*, pages 528–542, 2006.
10. Gerti Kappel, Horst Kargl, Gerhard Kramler, Andrea Schauerhuber, Martina Seidl, Michael Strommer, and Manuel Wimmer. Matching metamodels with semantic systems - an experience report. In *Proc. of BTW Workshops*, pages 38–52, 2007.
11. Ivan Kurtev, Mehmet Aksit, and Jean Bézivin. Technical Spaces: An Initial Appraisal. In *Proc. of CoopIS*, 2002.
12. Sergey Lukichev. Defining a subset of OCL for expressing SWRL rules. In *RuleApps*, pages 1–3, 2008.
13. Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. Towards Sharing Rules Between OWL/SWRL and UML/OCL. *ECEASST*, 5, 2006.
14. Fernando Silva Parreiras and Steffen Staab. Using ontologies with UML class-based modeling: The TwoUse approach. *Data Knowl. Eng.*, 69(11):1194–1207, 2010.
15. Fernando Silva Parreiras, Steffen Staab, Simon Schenk, and Andreas Winter. Model driven specification of ontology translations. In *Proc. of ER*, pages 484–497, 2008.
16. Fernando Silva Parreiras, Steffen Staab, and Andreas Winter. On marrying ontological and metamodeling technical spaces. In *Proc. of FSE*, pages 439–448, 2007.
17. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. In *Proc. of ISWC*, pages 30–43, 2006.
18. Axel Polleres. SPARQL1. 1: New features and friends (OWL2, RIF). In *Web Reasoning and Rule Systems*, pages 23–26. Springer, 2010.
19. Axel Polleres, François Scharffe, and Roman Schindlauer. SPARQL++ for mapping between RDF vocabularies. In *Proc. of OTM*, pages 878–896, 2007.
20. Tirdad Rahmani, Daniel Oberle, and Marco Dahms. An adjustable transformation from OWL to Ecore. In *Proc. of MODELS*, pages 243–257, 2010.
21. Federico Rieckhof, Mirko Seifert, and Uwe Aßmann. Ontology-based model synchronisation. In *Proc. of TWOMDE Workshop*, 2010.
22. Simon Schenk. A sparql semantics based on datalog. In *Proc. of KI*, pages 160–174, 2007.
23. Simon Schenk and Steffen Staab. Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web. In *Proc. of WWW*, pages 585–594, 2008.
24. Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Proc. of WG Workshop*, pages 151–163, 1994.
25. Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proc. of OWLED Workshop*, 2008.
26. Nan C. Shu, Barron C. Housel, Robert W. Taylor, Sakti P. Ghosh, and Vincent Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM Trans. Database Syst.*, 2(2):134–174, 1977.
27. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.
28. Steffen Staab, Tobias Walter, Gerd Gröner, and Fernando Silva Parreiras. Model driven engineering with ontology technologies. In *Proc. of Reasoning Web*, pages 62–98, 2010.
29. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297, 2006.
30. Dennis Wagelaar. Towards using OWL DL as a metamodelling framework for ATL. In *Proc. of MtATL Workshop*, pages 79–85, 2010.
31. Tobias Walter, Fernando Silva Parreiras, Gerd Gröner, and Christian Wende. OWLizing: Transforming Software Models to Ontologies. In *Proc. of ODiSE*, pages 7:1–7:6, 2010.