Bridging Concrete and Abstract Syntax in Model Driven Engineering: A Case of Rule Languages

Milan Milanović1, Dragan Gašević2, Adrian Giurca3, Gerd Wagner3, Vladan Devedžić1

1 FON-School of Business Administration, University of Belgrade, milan@milanovic.org, devedzic@etf.rs
2 School of Computing and Information Systems, Athabasca University, Canada dgasevic@acm.org
3 Institute of Informatics, Brandenburg University of Technology at Cottbus, Germany {giurca|g.wagner}@tu-cottbus.de

Abstract. The paper covers the problem of bridging the gap between abstract and textual concrete syntax of software languages in the model-driven engineering (MDE) context. This problem is well-studied in the context of programming languages, but due to the obvious difference in the definitions of abstract syntax, MDE requires a new set of engineering principles. We first explore different approaches to defining abstract and concrete syntax in the MDE context. Next, we investigate, the current state of languages and techniques used for bridging between textual concrete and abstract syntaxes in the context of MDE. Finally, we report on lessons learned in experimenting with current technologies. In order to provide a comprehensive coverage of the problem under study, we have selected a case of Web rule languages. Web rule languages leverage various types of syntax specification languages; and they are complex in nature and large in terms of the language elements. Thus, they provide us with a realistic analysis framework based on which we can draw general conclusions. Based on the series of experiments that we conducted with the analyzed languages, we propose a method for approaching such problems and report on the empirical results obtained from the data collected during our experiments.

1. Introduction

Languages are everywhere [54]. In general, languages are used to communicate ideas and intentions between people, while software languages in particular have different purposes depending on a phase in software development lifecycle for which they are intended to be used. For example, certain languages are suitable for requirements specification and verification (e.g., Z), others are suitable for software design (e.g., UML), implementation (e.g., Java) and integration (e.g., BPEL). This wealth of software languages primarily calls for the development of software language engineering mechanisms. In the software language engineering context, the key concept is language descriptions. According to Anneke Kleppe [46] “a language description of language L is the set of rules according to which the linguistic utterances of L are structured, optionally combined with a description of the intended meaning of the linguistic utterances.” This definition identifies the two core components of language descriptions: syntax and semantics.

In this paper, our focus is on syntax of software languages where we distinguish between abstract syntax and concrete syntax. This distinction is based on their purpose where abstract syntax is independent of any machine data structure or physical representation of language utterances, while concrete syntax is primarily concerned with the concrete representations of language utterances. Concrete syntaxes of languages can be divided into graphical and textual. Typically, a language has one abstract syntax and can have more than one concrete syntax. For example, UML has one abstract syntax in the form of the UML metamodel, and two concrete syntaxes – one XML based one so-called UML XMI, and another one – UML graphical notation.

As there are different approaches to language engineering (e.g., grammar and metamodel engineering), there are also different techniques for defining syntax. In the well-established area of programming languages [47], abstract syntax is defined through the use of the EBNF notation. For bridging
between abstract syntax and concrete syntax, parsing is used where a concrete syntax-based representation of language utterances is processed and abstract syntax trees are created as a final result. On the other hand, in the emerging software engineering discipline of Model-Driven Engineering (MDE), metamodels are used for defining abstract syntax of languages. The use of metamodels also implies that the primary description of languages and relations between their entities is based on graphs. This implies that bridging between abstract syntax and concrete syntax in the MDE context has different needs comparing to grammar engineering.

The main research problem, which this paper looks at, is bridging the gap between abstract and textual concrete syntaxes of software languages in the MDE context. This problem is well-studied in the context of programming languages, but due to the obvious difference in the definitions of abstract syntax, MDE requires a new set of engineering principles. There have already been several proposals to bridge between concrete and abstract syntax in MDE. For example, Graphical Modeling Framework (GMF)\(^1\) is an Eclipse effort for graphical concrete syntax, while Textual Concrete Syntax (TCS) [23] and xText\(^2\) are proposals to defining textual concrete syntax of languages. However, to the best of our knowledge there has not been an extensive evaluation of the currently available techniques. Moreover, there has not been proposed an engineering that proposes best practices in dealing with the problem under study.

In this paper, we fully focus on bridging between textual concrete syntax and abstract syntax in the MDE domain. In fact, our major goals are:

- Investigate different approaches to defining abstract and concrete syntax in the MDE context;
- Investigate the current state of languages and techniques used for bridging between textual concrete and abstract syntax in the MDE context;
- Report on lessons learned in experimenting with current technologies, and define an engineering method for bridging between concrete and abstract syntaxes in the MDE context.

In order to provide a comprehensive coverage of the problem under study, we have selected a case of Web rule languages. Web rule languages leverage various types of syntax specification languages; and they are complex in nature and large in terms of the language elements. Thus, they provide us with a realistic analysis framework based on which we can draw general conclusions.

We start the paper by a short introduction into the background MDE concepts that we use in our study. In Section 3, we describe the abstract and contract syntaxes of the rule languages used in our study, including, REWERSE Rule Markup Language (R2ML), Semantic Web Rule Language (SWRL), and Object Constraint Language (OCL). In Section 4, we leverage a three-layered metamodeling architecture, typically used in MDE, to analyze relations between abstract and concrete syntax definition mechanisms. Section 5 summarizes the method that we propose to bridge between abstract and concrete syntaxes in the MDE context, while section 6 describes a concrete application of the proposed method to rule languages. In Section 7, we report on the results that we obtained by empirically examining the proposed method and developed software artifacts. Before concluding the paper, we highlight the related work in Section 8.

2 Background

This section introduces the basic concepts and technologies that are used in our analysis and experimental solution to bridging the gap between abstract and concrete syntax of the languages used in our study. Namely, we introduce fundamental concepts of MDE such as metamodeling, technical spaces, and model transformations.

2.1. Model Driven Engineering

MDE is a new software engineering discipline in which the process heavily relies on the use of models

\(^1\) http://www.eclipse.org/gmf/
\(^2\) http://wiki.eclipse.org/Xtext
while the OMG’s Model-Driven Architecture (MDA) is considered a possible metamodeling architecture enabling the use of the MDE principles. The core concept in MDE is model. A model defined is a set of statements about some system under study [41]. Models are usually specified by using modeling languages (e.g., UML), while modeling languages can be defined by metamodels. A metamodel is a model of a modeling language. That is, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language [41]. Relations between models and metamodel are described through metamodeling architectures where a metamodeling architecture typically consists of three layers (e.g., MDA), namely:

- M1 layer or model layer where models are defined by using modeling languages;
- M2 layer or metamodel layer where models of modeling languages (i.e. metamodels) are defined (e.g., UML or Ontology Definition Metamodel, ODM [34]) by using metamodeling languages such as MOF;
- M3 layer or metametamodel layer where the only metamodeling language is defined (i.e. MOF) by itself [32].

The relations between different MDE layers can be considered as instance-of or conformant-to, which means that a model is an instance of (i.e., conformant to) a metamodel, and a metamodel is an instance of (i.e., conformant to) a metametamodel. The rationale for having only one language on the M3 layer is to have a unique (grammar) space for defining various modeling languages on the M2 layer. Thus, various modeling language can be processed in the same way, for example, by using the same API or transformation language. An important feature of MOF is that it inherits the graphical concrete syntax of UML, that is, MOF is a subset of UML class models, so that an abstract syntax of a modeling language defined by MOF can be represented by using UML class diagrams. Of course, similar relations can also be leveraged to describe phenomena in the context of grammars [5].

We should mention also the well-formedness rules in terms of metamodeling architectures. Typically, they are defined on the metamodel level, by using a combination of a metamodeling language (MOF) and a constraint language (OCL). The constraints are defined over metamodels, most commonly, in a form of integrity constraints such as OCL invariants. These rules regulate a set of valid models (i.e., expressions of a modeling language), and only those models that are fully compliant with such well-formedness rules are considered well-formed. For example, well-formedness of R2ML is defined by using MOF and OCL, while all R2ML rules (i.e., models) must comply to the well-formedness rules of R2ML.

Note also that for each MOF-based metamodel and model, one can automatically generate their corresponding XML schema (so called XML Metadata Interchange–XMI) by following the OMG’s XMI specification [37]. This enables sharing MOF-based models and metamodels among different MOF-based model repositories.

### 2.2. Technical spaces

Although MDE principles for defining modeling languages seems quite promising, the reality is that languages related can be defining and represented by using various technologies such as XML, databases, and MOF. In fact, the MDE theory introduces a concept of technical spaces [3], where a technical space (TS) is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [25]. Although some technical spaces are difficult to define, they can easily be recognized (e.g., XML, databases, and Semantic Web). In the case of the problem analyzed in this paper, we have to bridge between two technical spaces – the MDE technical space (as we assume that abstract syntaxes of languages are defined by using metamodeling) and technical spaces, such as XML, RDF and EBNF, are used for defining concrete syntaxes of languages.

### 2.3. Model Transformations

Model transformations represent the central operation for handling models in MDE [40]. Model trans-
formations are the process of producing models from (a set of) model(s) of the same system [4]. In fact, a model transformation means converting an input model (or a set of input models), which conforms to one metamodel, to another model(s), which conforms to another metamodel (see Fig. 1). This conversion is done by defining rules that match and/or navigate elements of source models resulting in the production of elements of the target model. The transformation itself is a model, which conforms to some transformation metamodel [4]. Languages for defining model transformations are generally declarative, and include approaches based on relations [4], or those based on patterns of logical constraints [12]. In Fig. 1, we demonstrate a well-known metamodel-based model transformation principle, which means that input and output models of model transformations must be conformant to their corresponding metamodels (i.e., input and output models must satisfy all well-formedness rules, as otherwise the transformation will not be produce any output).

Although it is very important to have a standard such as OMG’s MOF2 Query View Transformation [36], it is equally important to use an appropriate tool that allows us to represent models and metamodels being transformed between different technical spaces (e.g., MOF and XML). In our research, we have decided to use ATLAS Transformation Language (ATL) [56] as the primary language and tool for model transformations based on the following arguments: an open-source software, the biggest user community, a solid developer support, a rich knowledge base of model transformation examples and projects, and a very mature support for technical spaces (e.g., XML, EMF, and MOF).

![Fig. 1. An overview of model transformations](image)

ATL is a hybrid (i.e., declarative and imperative) transformation language, and it is based on the OMG OCL norm [33] for both its data types and its declarative expressions. Table 1 summarizes basic ATL constructs.

<table>
<thead>
<tr>
<th>ATL Construct</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>matched rules</td>
<td>Declarative element. Triggered when an input pattern is matched in source models. Produces an output pattern that is created in target models.</td>
</tr>
<tr>
<td>called rule</td>
<td>Imperative element. Invoked (called) as a procedure/method.</td>
</tr>
<tr>
<td>action block</td>
<td>Imperative element. Represents a sequence of imperative constructs that can be used in both matched and called rules.</td>
</tr>
<tr>
<td>lazy rule</td>
<td>Declarative element. Explicitly invoked from some other rule. When called create a new element with the same content from the same input element. Suitable for one-to-many transformations.</td>
</tr>
<tr>
<td>unique lazy rule</td>
<td>Declarative element. Explicitly invoked from some other rule. Create an element of the target model once for a specific element of the source model, while all other executions of the same rule for that specific input element create references to the only target definition. Suitable for many-to-one transformations.</td>
</tr>
<tr>
<td>helpers</td>
<td>ATL equivalent to Java methods. They make it possible to define factorized ATL code that can be called from different points of an ATL transformation.</td>
</tr>
</tbody>
</table>

ATL is implemented as an Eclipse plug-in and integrates the notion of technical spaces. Although mainly intended to deal with modeling languages, this framework should also handle other kinds of
models from different technical spaces (e.g., Java programs, XML documents, and databases). The ATL Eclipse perspective provides tools for importing/exporting various XML formats into/from ATL’s MOF based model repository. This is implemented in the form of the following tools:

- **XML injection**: takes an XML file as the input and produces its equivalent model that is conformant to the XML metamodel defined in MOF or Ecore. Ecore is a part of Eclipse Modeling Framework (EMF) and it is equivalent and very similar to MOF.
- **XML extraction**: produces an XML file from the model conformant to the metamodel defined in either MOF or Ecore.
- **EBNF injection**: takes a textual file (written in a textual concrete syntax of a language) as the input and produces an equivalent model that is conformant to the language’s metamodel defined in either MOF or Ecore.
- **EBNF extraction**: produces a textual file (written in a textual concrete syntax of a language) from the model conformant to the language’s metamodel defined in either Ecore or MOF.

In addition, ATL toolkit also includes TCS (Textual Concrete Syntax) tools [23]. TCS represents a Domain Specific Language (DSL) for defining textual concrete syntax in MDE. As a part of the ATL toolkit, it can be used for parsing text-to-model and serialization model-to-text.

### 3. Syntaxes of the Analyzed Rule Languages

The proposed methodology and results reported in this paper are based on our experience in developing Web rule languages and their supporting tools. Namely, we have been contributing to the overall effort of the Web engineering community to develop an interchange language for rules between various types of rule languages on the Web. This is actually the main focus on the W3C effort called Rule Interchange Format (RIF) [17]. The most known research proposals for a RIF are R2ML (R2ML) [29],[44] and RuleML [7]. In this scope of our research activities, we have developed the R2ML language for which we developed transformations with nine other rule languages such as Semantic Web Rule Language [20], OCL [33], Jess [22], RuleML [7], and JBoss’ Drools [21]. While our primary subject of research was the translation between various rule languages, we faced another important challenge – bridging between abstract and concrete syntaxes of the same rule language [8]. In this paper, we use only three rule languages for our discussion where those languages are selected based on the size of their abstract syntax (i.e., metamodels) and based on variety of definitions of their textual concrete syntaxes. In the rest of the section, we describe these three rule languages under study by means of their abstract and concrete syntaxes.

#### 3.1. R2ML Metamodel and R2ML XML Schema

This section describes the R2ML abstract syntax and R2ML XML-based concrete syntax of the R2ML language [44],[45]. The R2ML language tries to address all the requirements specified in the RIF document [17] in order to provide a general markup language for sharing Web rules. Due to the size of the R2ML language, we only give an excerpt of the language related to integrity and derivation rules in this section, although the full language also defines reaction rules (also known as event-condition-action rules) and production rules as well as features for representing vocabularies similar to those of UML and the Web Ontology Language (OWL) [45]. For the complete definition of the R2ML metamodel and R2ML XML schema, we refer readers to [38].

#### 3.1.1. The R2ML Abstract Syntax: R2ML Metamodel

The R2ML metamodel is defined by using MOF. In Fig. 2, we give a UML class diagram depicting the MOF definition of integrity rules with the definition of RuleBase and RuleSet. An R2ML RuleBase contains IntegrityRuleSet-s or DerivationRuleSet-s, while either of these two RuleSet-s contains specific rules, integrity or derivation, respectively. An integrity rule, also known as (integrity) constraint, consists of a constraint assertion, which is a sentence (or formula without free variables) in a logical
language such as first-order predicate logic or OCL [33].

Example 1 (Integrity rule). If a rental is not a one way rental then the return branch of the rental must be the same as the pick-up branch of the rental. 

Example 2 (Derivation rule). The discount for a customer buying a product is 7.5 percent if the customer is premium and the product is luxury.

In order to be automatically verified, metamodel constraints could be written in OCL to fully specify well-formedness rules of the R2ML language. For defining these rules we use OCL Invariants. Fig. 2 also shows an OCL invariant stating that IntegrityRules must not contain any free variables.

3.1.2 R2ML XML Schema

The concrete syntax of the R2ML language is defined in a form of an XML schema. This XML schema is defined based on the R2ML MOF-based metamodel, while here we just briefly indicate those mapping rules:

1. Every metamodel class is represented by an XML element and a complexType in the XML schema. The corresponding XML element contains XML attributes for each data type attributes from the model class.

2. A MOF association is mapped to an XML attribute that is part of the content model of the XML complexType generated from the class referencing this association.

3. A composite association is mapped to an XML element that is a part of the content model of the XML complexType generated from the class referencing this association.

The full definition of the R2ML XML schema can be found in [38]. In Fig. 3, we give the integrity and derivation rules defined in Examples 1 and 2, respectively, as R2ML XML format.

3.2 OCL Metamodel and OCL EBNF-based Concrete Syntax

Here, we briefly describe the OCL language by explaining its MOF-based abstract syntax and EBNF-based textual concrete syntax, while their complete description could be found in [33].

3.2.1. The OCL Abstract Syntax: OCL Metamodel

The OCL metamodel (i.e., abstract syntax) is also defined by using MOF [32]. The basic structure of the OCL metamodel defines OCL expressions by OclExpression that is an abstract superclass for all OCL expressions. An excerpt of the OCL metamodel is given in Fig. 4.
Fig. 3. R2ML XML representation of the rules from Example 1 (a) and Example 2 (b)

Fig. 4. The basic structure of expressions in the OCL metamodel

3.2.2. The OCL Concrete Syntax
The concrete syntax of the OCL language is defined in a form of a full attribute grammar, where each production in the attribute grammar may have synthesized attributes attached to it [33]. Each production rule has one synthesized attribute called ast (short for abstract syntax tree), that holds the instance of the OCL Abstract Syntax that is returned by the rule. Each production rule also has one inherited attribute called env (short for environment), that holds a list of names that are visible from the expression. All names are referenced to elements in the model. In fact, env is a name space environment for the expression or expression part denoted according to the production rule [33].

The mapping from the concrete to abstract syntax is described as part of the grammar in [33]. It is described by adding a synthesized attribute ast to each production which has the corresponding metaclass from the abstract syntax as its type. This allows the mapping to be fully formalized within the attribute grammar formalism. Fig. 5 shows an example of production rules for OclModuleElement.
Fig. 5. Production rules for OCLModuleElement

The Abstract syntax mapping for OCLModuleElement is defined in Fig. 6.

OclModuleElementCS.ast : OclModuleElement

Fig. 6. Abstract syntax mapping of OclModuleElement

The full definition of the OCL concrete syntax with well-formedness rules can be found in [33].

3.3 SWRL Metamodel and SWRL Concrete Syntax

In this subsection, we present the abstract syntax and the XML-based and RDF-based concrete syntaxes of the SWRL language. SWRL tends to be a standardized reasoning layer built on top of Web Ontology Language (OWL). For a complete definition of the SWRL metamodel we refer readers to [10], and for description of the complete SWRL language concrete syntaxes to [20].

3.3.1. The SWRL Abstract Syntax: RDM Metamodel

In [10], authors, inspired by the OMG’s ODM specification [34], proposed a Rule Definition Metamodel (RDM) – a MOF-based metamodel for SWRL [10]. As SWRL includes OWL constructs, RDM represent an extension of the ODM metamodel [34], which defines a metamodel of ontology languages such as OWL and RDF Schema.

3.3.2. The SWRL Concrete Syntax

SWRL has two concrete syntaxes, namely, RDF/XML [2] and OWL/XML [19]. As SWRL is defined on top of OWL, while OWL is based on RDF [26], SWRL has an RDF-based concrete syntax. Thus, SWRL extends two existing concrete syntaxes of OWL. The RDF/XML concrete syntax is defined by an RDF Schema (RDFS) [9] available in [20]. In Fig. 7a, we give an example of a SWRL rule in a form of a RDF document following the SWRL RDF/XML schema. To notion of the rule is “if a person has a parent and the parent has a brother, then the parent’s brother is the person’s uncle.” The SWRL OWL/XML concrete syntax is a combination of the OWL Web Ontology Language XML Presentation Syntax [19] with the rule specific constructs previously defined in the RuleML XML concrete syntax [18]. In Fig. 7b, we give the same SWRL rule as in Fig. 7a, but now the rule is written in the OWL/XML concrete syntax.

Fig. 7. SWRL rule asserting that a combination of the hasParent and hasBrother properties implies the hasUncle property in the SWRL RDF/XML syntax
4. Analysis of the Representations of Language Syntaxes

In this section, we analyze mutual relations of the syntaxes of the rule languages used in our case study. Our goal is to make their systematic inventory and chart their mutual relations by positioning them with respect to the typical MDE reference model – three-layered metamodeling architecture (c.f. Section 2.1). The layered architecture-based approach allows us to clearly position the syntax definition mechanisms with respect to their corresponding technical spaces (TSs), and thus systematically compare them with the other syntax definition mechanisms used in other technical spaces.

For all the analyzed languages, we started from the fact that their abstract syntaxes are defined as metamodels in the MDE TS (see Fig. 8). In our particular case, the MDE TS is defined by its metamodel, that is, by MOF. It is important to note that rule language metamodels are defined on the M2 layer, while concrete real world rule models are defined on the M1 layer.

Fig. 8. MDE TS and abstract syntax of rule languages

4.1. Mappings between SWRL Concrete and Abstract Syntax

In Fig. 9, we show relations between different technical spaces used for abstract and concrete syntax of SWRL. In the center of Fig. 9, we show the RDM metamodel (from Fig. 8) in MDE TS, where RDM is an abstract syntax of SWRL in terms of MDE. The SWRL language has two concrete syntaxes defined in the RDF and XML technical spaces. In the right part of Fig. 9, we have the SWRL OWL/XML concrete syntax defined as an XML Schema. We observe the XML TS in terms of the W3C XML Schema recommendation where the M3 layer is a schema for XML schemas (i.e. XML meta-schema) – this schema defines the validity of XML Schema definition documents. Domain specific XML vocabularies (i.e. schemas) are defined on the M2 layer of the XML TS, while concrete XML documents are on the M1 layer. Epistemologically, one can say that these three layers are equivalent to the MDE layers. On the M2 level, where metamodels and XML schemas are, we have a mapping function between the RDM metamodel and SWRL OWL/XML Schema (i.e., between sets of expressions that can be asserted in the two syntaxes of the same language, as defined in Section 2). On the M1 level (i.e., MDE models and XML concrete rule), we have transformations which are based on the mapping function.

In the left part of Fig. 9, we show the RDF TS where on the M3 level, we have a similar case to that of the XML TS, that is, the RDF meta-schema, which defines the validity of the domain specific RDF schemas, which are defined on the M2 level. Finally, concrete RDF documents are on the M1 layer of the RDF TS. In terms the RDF TS, the SWRL RDF/XML schema is on the M2 level and concrete SWRL rules represented in the RDF/XML-based concrete syntax are on the M1 level. Similar to the XML TS, we define a mapping function between the SWRL RDF/XML Schema and RDM metamodel on the M2 level, while the concrete transformation which conforms to this mapping is defined and
executed on the level of models and concrete rules (i.e., M1 level).

Fig. 9. Bridges between the SWRL concrete and abstract syntax in different technical spaces

General consideration on how both bridges can be done is provided in Section 4.4, while an example of a concrete implementation in ATL is given in Section 6.1.

4.2. Mappings between R2ML Concrete and Abstract Syntax

For R2ML whose concrete syntax is defined in the XML TS, we have mappings and transformations defined in a similar way as for the SWRL OWL/XML Schema and the RDM metamodel (see Fig. 10).

Fig. 10. Bridges between R2ML concrete and abstract syntax in different technical spaces

General consideration on how the bridge can be done is provided in Sect. 4.4, while an example of a concrete implementation in ATL is given in Sect. 6.1.

4.3. Mappings between OCL Concrete and Abstract Syntax

In Fig. 8, we presented the OCL metamodel and its textual concrete syntax. An important difference to the previous two cases is that the OCL concrete syntax is defined in the EBNF TS (see Fig. 11). On the M3 level of the EBNF TS, we have the EBNF definition (meta-grammar) which is used to define the OCL grammar on the M2 level of the EBNF TS. This OCL grammar is then used to define rules for expressing OCL constraints that are used on the M1 layer. On the M2 level of Fig. 11, we define mapping function between concepts of the grammar and metamodel, while on the M1 level, we define a transformation that implements the mapping function to enable transforming between text-based
OCL constraints and MOF-based instances of the OCL metamodel. General consideration on how the bridge can be implemented is provided in Sect. 4.4, while an example of a concrete implementation in ATL is given in Sect. 6.1.

Fig. 11. Bridges between OCL concrete and abstract syntax in different technical spaces

4.4. Technologies for Implementation

Summarizing the relations between concrete and abstract syntax of the analyzed rule languages, we can say that for each of the three analyzed technical spaces (e.g., XML, EBNF, and RDF), we have a mapping function-transformation pair that determines a bridge between each particular concrete and abstract syntax. Table 2 gives guidelines on how to develop these bridges pairs of the discussed languages. In the table, we only indicate the transformations between each of the observed languages and the technical space in which the bridge can be developed by employing some of transformation techniques.

At the first look, we can see that all transformations between the languages’ concrete syntaxes (i.e., R2ML, SWRL and OCL) and their corresponding abstract syntax (i.e., metamodels) can be done in different technical spaces. From Table 2, the transformation between abstract and concrete syntax of every of the three languages can be done in the MDE TS, but regarding the type of languages’ concrete syntax it can be also done in some other technical spaces.

Table 2. An overview of possible transformations between abstract and concrete syntax of OCL, R2ML and SWRL with respect to what technical space their bridges can be developed in (XSLT, Query/View/Transformation – QVT, ATL’s Textual-Concrete Syntax (TCS), and TRIPLE3).

<table>
<thead>
<tr>
<th>Rule languages metamodels</th>
<th>EBNF</th>
<th>OCL</th>
<th>R2ML</th>
<th>SWRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical spaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MDE</td>
<td>EBNF</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>TS</td>
<td>TS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TCS</td>
<td>TXL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XML</td>
<td>MDE</td>
<td>XML</td>
<td>MDE</td>
</tr>
<tr>
<td></td>
<td>TS</td>
<td>TS</td>
<td>TS</td>
<td>TS</td>
</tr>
<tr>
<td></td>
<td>QVT</td>
<td>XSLT</td>
<td>QVT</td>
<td>XSLT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RDF</td>
<td>MDE</td>
<td>XML</td>
<td>RDF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TS</td>
<td>TS</td>
<td>TS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QVT</td>
<td>XSLT</td>
<td>TRIPLE</td>
</tr>
</tbody>
</table>

Looking at the first row, the bridge between OCL rules in the concrete EBNF-based and the OCL metamodel can be done in the MDE TS by using TCS and in EBNF TS by using the grammar-based transformation languages such as ASF+SDF [43] [52], TXL [48] or Stratego/XT [49]. In the second

row, R2ML and SWRL both have the XML-based concrete syntax, so that one can use either the MDE-based approach (QVT) or the XML-based approach (XSLT) for bridging between the abstract and concrete syntax of the two languages.

Additionally, SWRL also has an RDF-based syntax (the last row in Table 2), and for bridging this type of concrete syntax and the SWRL abstract syntax (metamodel), we can also use table to use the TRIPLE language [42], that is, this transformation can be done in the RDF TS. Some constrains with the use of TRIPLE are discussed in Section 8.

We should finally note that the table should not be considered a definite list of possible implementation technologies for the bridges between syntaxes of the analyzed languages. A further discussion on cons and pros of different bridging approaches is given in Section 8.

5. Proposed Method

In this section, we summarize the method that we used during our experiments in bridging abstract and concrete syntax of rule languages. The method is based on the technologies used in the MDE TS, which assumes the use of the ATL language and its accompanied TCS language.

5.1. General Bridging Steps

Here, we describe some general steps for the bridging of abstract and concrete and syntax.

5.1.1. Sharing Rules between Different Languages

The initial step is to provide a detailed analysis of the formalisms used for defining abstract and concrete syntax of the languages at hand. In Section 3, we demonstrated how the MDE layered architecture can be used in this step by allowing for drawing epistemological relations between different syntax definitions used in different technical spaces. Once those relations are drawn and syntax definitions are positioned in their relevant technical spaces, the next step is to identify the needed bridges between the analyzed concrete and abstract syntax. This activity is followed by defining conceptual mappings of the concepts used in the analyzed concrete and abstract syntaxes.

5.1.2. Making Unique Mappings for Bridging Abstract and Concrete Syntax

Defining conceptual mappings between corresponding elements of the syntaxes is a key activity in bridging an abstract and concrete syntax. In our approach, we used a tabular form such as Table 3. Based on the defined mappings, we can later develop transformations to bridge concrete models into one or another technical space (such as for R2ML in Section 6.1), conforming to the language’s metamodel, or concrete syntax definition, respectively.

5.1.3. Transformations based on Mappings for Bridging Abstract and Concrete Syntax

Once a technical space for the representation of the language’s abstract syntax is defined (in our case, that is MDE), and when such an abstract syntax is defined, we then need to choose a suitable solution to developing the transformation. This solution should be based on criteria such as potential to define needed transformations (Table 2), flexibility to change those transformations, and compatibility to work in needed technical spaces. Once the abstract or concrete syntax of the rule language is changed, we need to change mappings between their elements and implemented transformations, too.

5.2. Steps Specific for Different Definitions of Concrete Syntaxes

In this subsection, we describe some steps, which are specific for the particular type of concrete syntaxes, namely, textual and XML-based. The description of our approach is based on the ATL terminology introduced in Section 2.3, and assumption that the approach is used in the MDE TS, as per the objectives of our research.
5.2.1. XML-based Concrete Syntax
Here, we describe the activities needed to develop transformations of XML-based textual concrete syntaxes. We assume that such syntaxes are defined by means of an XML Schema.

5.2.1.1. XML Schema and Metamodel Mappings
XML Schema elements and metamodel elements are mapped in the following way: every metamodel class is mapped to an element and a complexType in the XML schema (or to RDF class if we map RDF/XML-based syntax). The names of the element and complex type are the same as the class’s name. If the class is abstract, the corresponding element is also abstract. The corresponding XML element contains attributes for each data type attributes from the model class. A MOF association is mapped to an XML attribute (or to an RDF Property in the case of RDF-based syntax). Such an XML attribute is a part of the content model of the complexType obtained from the class. Composite and n-ary MOF associations are always mapped into XML elements. A composite association is mapped to an element that is a part of the content model of the complexType generated from the class referencing that association. In the case of RDF, a MOF composite association is mapped to an RDF Property whose domain is a translated class referencing the association and whose range is the translated class referenced by the association.

5.2.1.2. Helper Functions for Searching through a XML-based concrete syntax
In the first step of the described transformation between the XML concrete syntax and abstract syntax, rules in the concrete syntax are injected into the XML models. For every input model element, in this case XML models’ Element, we create transformation rules to translate them into the output metamodel elements. But, such rules cannot always straightforwardly be defined, because often there is a need to search through the input XML model graph for some element(s). For this purpose, we have to create different helper functions in transformations, including, functions that return input elements by name, functions that return an attribute value for some XML element, and recursive functions for searching through a complete graph of XML elements, starting from the root element (actually flattening the complete XML model graph) in order to find a certain element such as a variable.

5.2.1.3. Sequence of Transformation Rules
For unidirectional transformation languages, such as ATL in our case, we need to create two transformations; one from the concrete to the abstract syntax and another one in the opposite direction. In either of transformations, a transformation rule should be created for each concrete syntax element and their corresponding metamodel elements, by following the mapping rules we defined earlier in section 5.2.1.1, and by using helper functions describer in section 5.2.1.2. A transformation rule should be created for every non-abstract metamodel class (e.g., RuleBase in Fig. 2), starting from the root element of the language to the other elements following the metamodel hierarchy. For example, the root element for the R2ML metamodel is RuleBase.

5.2.1.4. Transformations of Unique Elements in Languages
As model transformation languages transform every element of the input model into the corresponding element of the output model, this can be problematic if we want to have only one output element for multiple input elements (i.e., one-to-many transformation) such as the case for rule language variables as will be described in section 6.1.1. In order to define these types of transformation rules, we need to search through the input XML model to find all occurrences of a variable (by using helper functions described in section 5.2.1.2) and to create a unique element with the same name in the output model. This can be done by using unique lazy rules. In the opposite direction, the situation is a bit simpler. In the case we need to create multiple elements in the output model from one unique element of the input

---

4 It is important to notice that XML is represented in terms of instances of the MOF-based XML metamodel, and thus, it is a graph not a tree as it is in the regular XML.
model (as we described in section 6.1.2, i.e., many-to-one), we propose using lazy rules.

5.2.2. EBNF-based Concrete Syntax
Here, we describe the steps we identified for languages with EBNF-based concrete syntaxes. In our case, we considered TCS language [23] for defining bidirectional transformation between EBNF-based representation of a textual concrete syntax and a metamodel.

5.2.2.1. Sequence of Transformation Rules
For every metamodel element, a transformation rule (called template) should be defined, even for abstract classes such as OCLExpression in Fig. 4. The transformation rules are defined only once, and they are used for transformations in both directions. In fact, we generate a parser that parses and translates language constructs defined in the concrete syntax into a representation compliant with the metamodel. The definition of the transformation rules should also start from the main language elements (i.e., classes, in a language’s abstract syntax (i.e., metamodel), such as the metamodel of the OCL constraints in our case. However, before the definition of transformations rules for the main language elements, transformation rules for primitive (e.g., integer and string) and enumerated types, should be defined by using TCS primitiveTemplate and enumerationTemplate, respectively.

5.2.2.2. Transformations of Unique Elements
Unique elements occurring several times in an EBNF-based concrete syntax should be created only once in the metamodel for every instance in the concrete syntax (many-to-one transformation). We describe this transformation on an example of variables in OCL, where a variable expression (VariableExp element) points to its variable declaration (Variable element) via association referredVariable. Fig. 12 gives the corresponding TCS model excerpt.

5.2.2.3. Definition of Classes Connected to Operators
If we have elements of a language’s concrete syntax that could be connected with some operators (e.g., point, star, and bracket), then those elements should be considered separately in a bridge to the language’s abstract syntax. This is due to the fact that such operators play an important role during the transformation process of elements connected with them (i.e., such an operator must be taken into account during the transformation process). In TCS, we use a specific construct called operatorTemplate for describing transformation rules for elements of concrete and abstract syntax, which can have operators connected to them. The TCS construct operatorTemplate enables defining operators that precede a definition of a metamodel element. We show an example of using this operator in Fig. 13.

Fig. 12. OCL TCS model excerpt: Variable definition

We first put Variable in the current symbol table (addToContext keyword). That is, each time when a variable is encountered, it is added to the symbol table. Then, the representation of the VariableExp’s property referredVariable (from Fig. 12) is changed by adding the property argument “referredTo=name”. This means that each time when a variable is encountered, its referredVariable property will be set to the Variable (type from the OCL metamodel) having the corresponding name. This Variable will be looked up in the symbol table. The target property of referredTo (e.g., name in this case) must be of type String.

5.2.2.3. Definition of Classes Connected to Operators
If we have elements of a language’s concrete syntax that could be connected with some operators (e.g., point, star, and bracket), then those elements should be considered separately in a bridge to the language’s abstract syntax. This is due to the fact that such operators play an important role during the transformation process of elements connected with them (i.e., such an operator must be taken into account during the transformation process). In TCS, we use a specific construct called operatorTemplate for describing transformation rules for elements of concrete and abstract syntax, which can have operators connected to them. The TCS construct operatorTemplate enables defining operators that precede a definition of a metamodel element. We show an example of using this operator in Fig. 13.

operatorTemplate PropertyCallExp(operators = opPoint, source = 'source')
: name[as = identifierOrKeyword];
The above template is related to the PropertyCallExp metamodel element. PropertyCallExp can have the name property and operator point (named opPoint) that precede the name property (e.g., “nameProperty”). Also, this kind of transformation rules has a source expression which is stored into the source element of the PropertyCallExp.

6. Case Study: Implementation of Bridging for Rule Languages

This section illustrates a possible application of the method proposed in the previous section. We used the proposed method in the transformations of our case study to bridge between the rule languages abstract and concrete syntax. We first show how we bridge between the XML Schema concrete syntax and the MOF-based abstract syntax. This transformation principle applies to both R2ML and SWRL languages. Next, we describe how we bridge between an EBNF-based concrete syntax and its abstract syntax (MOF-based metamodel). This is the case of the OCL languages.

6.1. Transformations between XML Schema and Metamodel

To develop transformations between an XML-based textual concrete syntax and the abstract syntax (i.e., metamodel), we should put both syntaxes into the same technical space. The overall organization of the transformation process is shown in Fig. 14 for the case of R2ML (N.B. similar applies to SWRL). It is obvious that the transformation between the XML schema and the metamodel consists of two transformation chains, namely: 1. from the metamodel to the XML schema (i.e., from the XML TS to the MDE TS); and 2. from the XML schema to the metamodel.

6.1.1. Transforming XML Schema into MOF-based Metamodel

This transformation chain consists of the two transformations as follows.

Transformation 1. Transforming the XML-based concrete syntax into the format of a metamodeling language. This means that we have to have an XML representation into the form compliant to MOF in order to be able to apply a model transformation language on the XML-based textual concrete syntax. In Fig. 14, this transformation is denoted as step 1 (i.e., XML injection in Fig. 14).

Transformation 2. Transforming the XML-based concrete syntax into the metamodel-compliant syntax. This step is the core one for bridging between the concrete and abstract syntax of R2ML. It transforms XML models (from the previous transformation) into the model conformant to the metamodel.
models of languages under study such as R2ML. In Fig. 14, this transformation is denoted as step 3.

To illustrate the above two steps, Table 3 gives an excerpt of mappings between the R2ML XML Schema, XML metamodel (transformation 1), and R2ML metamodel (transformation 2).

<table>
<thead>
<tr>
<th>R2ML schema</th>
<th>XML metamodel</th>
<th>R2ML metamodel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleBase</td>
<td>Root</td>
<td>RuleBase</td>
<td>Captures a collection of rules.</td>
</tr>
<tr>
<td></td>
<td>name = 'r2ml:RuleBase'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntegrityRuleSet</td>
<td>Element</td>
<td>IntegrityRuleSet</td>
<td>Captures a set of integrity rules.</td>
</tr>
<tr>
<td></td>
<td>name = 'r2ml:IntegrityRuleSet'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AlethicIntegrityRule</td>
<td>Element</td>
<td>AlethicIntegrityRule</td>
<td>Represents an alethic integrity rule.</td>
</tr>
<tr>
<td></td>
<td>name = 'r2ml:AlethicIntegrityRule'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ObjectVariable</td>
<td>Element</td>
<td>basContVoc.ObjectVariable</td>
<td>Represents an object variable.</td>
</tr>
<tr>
<td></td>
<td>name = 'r2ml:ObjectVariable'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In our language of choice (ATL), transformation 1 is fully automated by using the ATL XML Injector. The XML Injector automatically transforms XML documents into the models conforming to the MOF-based XML metamodel. This XML metamodel defines XML elements such as XML Node, Element, and Attribute. Transformation 2 is defined as a sequence of rules in the ATL language. Summary of principles for defining rules is given in Section 5. Here, we only mention the usage of the ATL language to handle important issue in mapping from a concrete syntax to the abstract syntax – many-to-one transformations. This is typical for a situation when several input elements represent the same entity, while in the target model we have to have only one unique definition of that element and other should only refer to that one. For example, the ObjectVariable XML element in the input rules might be used several times (as this is a tree) and the only way to know that all of them refer to the same variable is by the value of its name attribute (e.g., customer). However, in the R2ML abstract syntax (metamodel), an ObjectVariable should only be defined once (as it is a graph), while all other parts of the model can only refer to that definition. Fig. 3a shows an R2ML IntegrityRule. The figure highlights three appearances of the $r1$ variable in the concrete syntax. Fig. 15 gives the same rule in a form of a UML object diagram representing instances of the R2ML MOF-based metamodel with only one node representing the $r1$ variable. This many-to-one mapping is implemented by using unique lazy rule (see Table 1), as it is illustrated in Fig. 16 for the problem of ObjectVariables in R2ML.

![Fig. 15. An object diagram representing the integrity rule shown in Fig. 3](image-url)
Fig. 16. A transformation excerpt from the R2ML XML syntax to the R2ML metamodel: A unique lazy rule

6.1.2. Transforming MOF-based Metamodel into XML Schema
Along with the transformation of the XML schema to the metamodel, we have also defined a transformation in the opposite direction.

**Transformation 3.** Transforming the metamodel-compliant syntax into the XML-based concrete syntax. This is a reverse transformation of transformation 2. In Fig. 14, this is step 5.

**Transformation 4.** Transforming the metamodel-based representation of XML into the XML-based textual concrete syntax. This is a reverse transformation of transformation 1. In Fig. 14, this is step 7.

Transformation 3 is developed by using similar principles described for transformation 2. The only difference is the prevalent use of lazy rules instead of unique lazy rules. This is natural, as in transformation 3, we want to translate from the abstract syntax to the concrete syntax, and thus we need to handle one-to-many transformations. An example is `ObjectVariable`, which now needs to be transformed from one node shown in Fig. 15 into three nodes shown in Fig. 3. Transformation 4 is fully automated and it is based on the use of the ATL XML Extractor.

6.2. Transformations between Textual Concrete Syntax and Metamodel
This transformation includes bridging between the (EBNF-based) concrete syntax and the abstract syntax (i.e., metamodel). An example of this transformation is between the OCL abstract and concrete syntax. In Fig. 17, we show a solution that assumes using the TCS language along with EBNF Extractor and Injector.

Fig. 17. The transformation scenario between OCL concrete and abstract syntax (metamodel)

Fig. 18 shows an excerpt of the mappings between the OCL metamodel (in the KM3 format⁶, Fig. 18a) and its textual concrete syntax defined in TCS (Fig. 18b), which are defined by following the method proposed in Section 5.

---

⁶ KM3 is a domain specific language (DSL) for defining metamodels as well as MOF or Ecore. Its syntax is very similar to the one of Java [56], while its semantics is very similar to those of MOF and Ecore.
Briefly, the above example defines the following transformation rules:

- **Class OclModule** is represented with `ownedElements` (of the type `OclModuleElement`) and is a construct which is first created (denoted with `main`).
- **Class Invariant** is represented with: the context definition (i.e., the name of the appropriate UML or MOF class), the keyword "inv", the name of this constraint if it is defined, the symbol ":", and the specification (body) of this constraint. The elements `<newline>` and `<tab>` are used in serialization of the OCL model to the OCL code, and they represent a new line feed, and text bias, respectively.
- **Class OclContextDefinition** is represented with the keyword "context" and `contextElement` element.

### 7 Empirical Analysis

To the best of our knowledge, in the MDE domain, there is no widely adopted evolution approach, which we could leverage to evaluate our proposed method. This led us to the decision to come up with our own evaluation approach. We decided to conduct an empirical analysis in which we measured the amount of efforts needed to develop transformations in the concrete applications of the proposed engineering method. In particular, we measured the number of transformation rules developed per each class of the analyzed metamodels along with the overall number of the developed transformation rules for each of the transformations.

#### 7.1 Data Collection Protocol

For each metamodel used in our experiments, we collected the following data for each class of those metamodels: number of subclasses; number of superclasses; number of attributes; number of associations (also aggregations and compositions); and if a class is abstract or not. To collect data about the developed transformations, we again associated different types of rules to their corresponding class in the metamodel of the analyzed language. In particular, we collected the following data for each class:

- Number of matched rules used for processing a certain class. We collected that data by analyzing the input and output patterns of each matched rule, depending whether we analyzed the transformation from the abstract syntax or to the abstract syntax, respectively. As different developers may split the transformation of one element of the transformed metamodel into more than one matched rule, we only used binary data (i.e., 1 and 0) representing if a certain element of the metamodel has an associated matched rule or not.
- Number of lazy and unique lazy rules used for processing a certain class. The same procedure was used as for the matched rules given that these two types of rules are inherited (subclassied in the ATL metamodel) from matched rules.
- Number of helpers directly used for processing a certain class if a class was defined as the context of a certain helper, or if a helper is directly responsible for transformation of a certain metamodel element. Other helpers are not counted.
7.2 Statistical Analysis

For each of the analyzed rule languages, we used descriptive statistics in terms of mean and standard deviation per class of the analyzed metamodels.

7.2.1 Transformations for R2ML

Given that in our study we only focused on the integrity and derivation rules of R2ML, we only analyzed the part of the R2ML metamodel related to that part. The part of the R2ML metamodel that we used contains 95 classes out of which 29 are abstract. The XML schema definition of the R2ML XML-based concrete syntax contains: 210 elements, 102 complexTypes, 3 simpleTypes, 88 attributes, and 16 enumerations. During the experiments, we created a certain number of different ATL rules and helpers for transformations. Table 4 summarizes the collected data for the R2ML metamodel and its two transformations earlier presented in the paper. The statistics about the metamodel are calculated per class of the metamodel (e.g., number of subclasses of a class). The statistics about the transformation are also calculated per class (e.g., number of used matched rules for a class).

Table 4. Descriptive statistics summary for R2ML.

<table>
<thead>
<tr>
<th>Metamodel</th>
<th>Mean</th>
<th>St. dev.</th>
<th>Transformation</th>
<th>Rule</th>
<th>Mean</th>
<th>St. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superclasses</td>
<td>0.91</td>
<td>0.46</td>
<td>XML to R2ML</td>
<td>Matched</td>
<td>0.53</td>
<td>0.50</td>
</tr>
<tr>
<td>Subclasses</td>
<td>0.97</td>
<td>1.67</td>
<td>R2ML to XML</td>
<td>Unique lazy</td>
<td>0.11</td>
<td>0.31</td>
</tr>
<tr>
<td>Attributes</td>
<td>0.33</td>
<td>0.64</td>
<td>R2ML to XML</td>
<td>Helpers</td>
<td>0.19</td>
<td>0.66</td>
</tr>
<tr>
<td>Associations</td>
<td>0.74</td>
<td>1.08</td>
<td>XML to R2ML</td>
<td>Matched</td>
<td>0.51</td>
<td>0.50</td>
</tr>
<tr>
<td>Aggregations</td>
<td>0.00</td>
<td>0.00</td>
<td>R2ML to XML</td>
<td>Lazy</td>
<td>0.07</td>
<td>0.26</td>
</tr>
<tr>
<td>Compositions</td>
<td>0.20</td>
<td>0.63</td>
<td>R2ML to XML</td>
<td>Helpers</td>
<td>0.02</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Table 5 shows the number of different ATL rules for the transformations that bridge between the R2ML XML Schema and R2ML metamodel.

Table 5. Number of different ATL rules in XML2R2ML and R2ML2XML transformations

<table>
<thead>
<tr>
<th>Transformation/rules</th>
<th>Matched rules</th>
<th>Lazy rules</th>
<th>Unique lazy rules</th>
<th>Helpers</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2ML XML schema to R2ML metamodel</td>
<td>50</td>
<td>0</td>
<td>9</td>
<td>28</td>
</tr>
<tr>
<td>R2ML metamodel to R2ML XML schema</td>
<td>49</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

The number of matched rules is slightly different between the two transformations. The reason for this is the LiteralConjunction class, which is only contained in DerivationRules. Thus, the matched rule for DerivationRules processes LiteralConjunction and consequently reduces the number of transformations. As described in section 6.1, for every element that must be defined as unique in the MDE TS (i.e., the R2ML metamodel), we used unique lazy rules for its creation. In the opposite direction, we have only used lazy rules, since we wanted to create more than one output elements with same contents from one input element. The number of helpers is much higher in the XML2R2ML transformation. This is due to the need to walk through the input XML model and find occurrences of the same ObjectVariable (Section 6.1).

7.2.2 Transformations for SWRL

The RDM metamodel that we used contains 56 classes out of which 8 are abstract. The XML schema of the SWRL OWL/XML concrete syntax contains: 43 elements, 16 complexTypes, 2 simpleTypes, and 4 attributes. Table 6 summarizes the collected data for the SWRL metamodel and its two transformations with the SWRL OWL/XML concrete syntax. The statistics about the metamodel are calculated per class of the metamodel (e.g., number of subclasses of a class). The statistics about the transformation are also calculated per class (e.g., number of used matched rules for a class).
For transformations between the SWRL OWL/XML Schema and RDM metamodel, a summary of the used types of transformation rules is shown in Table 7. As with the transformation of the R2ML metamodel into the R2ML XML Schema, the number of the matched rules is the same for both transformations. As already described for R2ML, for every element which must be defined as unique in the MDE TS (i.e., RDM metamodel), we used unique lazy rules for their creation. In the opposite direction, we used only lazy rules, because we had to create more than one output element with the same contents from the same input element (see Section 6.2). The number of helper operations is larger in the XML2RDM transformation, because we needed to search through the input XML model to find instances of the same IndividualVariable elements.

Table 7. Number of different ATL rules in the XML2RDM and RDM2XML transformations

<table>
<thead>
<tr>
<th>Transformation/rules</th>
<th>Matched rules</th>
<th>Lazy rules</th>
<th>Unique lazy rules</th>
<th>Helpers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWRL OWL/XML Schema to RDM metamodel</td>
<td>36</td>
<td>0</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>RDM metamodel to SWRL OWL/XML Schema</td>
<td>36</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

7.2.3 Transformations for OCL
The OCL metamodel contains 71 class out of which 18 are abstract. The definition of the OCL textual concrete syntax contains 80 production rules and 42 non-terminals. Table 8 summarizes the collected data for the OCL metamodel. The information about the metamodel depicts the mean and standard deviation of the number of associated elements per each class of the metamodel (e.g., number of superclasses of a class). The rest of the table reports on the statistics describing the number of templates developed in our transformation per each class of the metamodel.

Table 8. Descriptive statistics summary for OCL

<table>
<thead>
<tr>
<th>Metamodel</th>
<th>Mean</th>
<th>St. dev.</th>
<th>Metamodel</th>
<th>Mean</th>
<th>St. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superclasses</td>
<td>1.01</td>
<td>0.36</td>
<td>Compositions</td>
<td>0.51</td>
<td>0.81</td>
</tr>
<tr>
<td>Subclasses</td>
<td>0.87</td>
<td>1.56</td>
<td>Rule</td>
<td>Mean</td>
<td>0.72</td>
</tr>
<tr>
<td>Attributes</td>
<td>0.25</td>
<td>0.82</td>
<td>Template</td>
<td>0.08</td>
<td>0.28</td>
</tr>
<tr>
<td>Associations</td>
<td>0.13</td>
<td>0.34</td>
<td>OperatorTemplate</td>
<td>0.08</td>
<td>0.28</td>
</tr>
</tbody>
</table>

For the OCL metamodel and OCL concrete syntax bidirectional transformation, we defined a number of transformation rules (called templates in TCS), and a summary is shown in Table 9.

Table 9. TCS transformation between the OCL metamodel and the OCL concrete syntax

<table>
<thead>
<tr>
<th>Transformation/rules</th>
<th>primitiveTemplate</th>
<th>template</th>
<th>operatorTemplate</th>
<th>enumerationTemplate</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCL metamodel to/from OCL concrete syntax</td>
<td>6</td>
<td>51</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

In this case from Table 9, we can see that we have 51 templates, that is, one simple TCS template for every non-abstract metamodel class whose corresponding concrete syntax element does not have operator associated with it. We have 6 primitiveTemplates for every primitive type (such as integer, boolean), one enumerationTemplate which corresponds to the OCL metamodels CollectionKind enumeration, and we also have six operatorTemplates used to describe those OCL abstract syntax element that have operators associated with them.
7.3 Implications in real-world development

The approach to bridging the gap between abstract and textual concrete syntaxes of software languages in the MDE context has different implications in real-world development. The implications on modeling and programming include situations when two different syntax representations are used to construct the same model or when model processing is done in one syntax representation, but input is in another syntax representation. An example of the usage of our approach is for example in the case of UML activity diagrams (models), where the surface language is defined to represent some model parts in a textual syntax [13], but those textual parts need to be included in a model as model parts. In such cases, we need to bridge textual and model-based syntax of the language. Also, we have similar example on the ATL itself, because ATL transformations which are defined in a textual syntax, are transformed into ATL metamodel instances in order to be checked for model problems [56].

As demonstrated in the paper, existing tools and languages (e.g., ATL and TCS) has already enabled for using MDE principles. In addition, existing languages and tools allow for bridging between abstract and concrete syntaxes in the MDE context. This is already confirmed by their use in many academic and industrial projects. However, the contribution of this paper addresses the problem which has attracted less attention by the research community, that is, what methodological steps software language engineers need to follow while using MDE tooling. We hope that our proposed methodology can be used as guidance for software language engineers in solving practical problems when there is a need to bridge between abstract and concrete syntaxes (e.g., code generation). From the reported empirical data, practitioners can also learn about the amount of efforts excepted for the development of transformations in the tasks related to the problem under study of this paper. This can further be useful to evaluate how best human resources and time can be managed in solving such problems. Our proposal can also be used by practitioners to understand better which technical knowledge and skills are needed to solve similar problems that they experience in their daily practice. Of course, we need to acknowledge the fact that further research is needed on the development of the different types of measures for more accurate estimation of all these aspects.

8 Related Work

In this section, we report on several related works in this area. We first start from the discussion related to the OMG’s official specification for concrete syntax – XMI, followed by the discussion on the alternative ways to bridge between concrete and abstract syntax in other different technical spaces including, grammar, XML, and RDF-based approaches. Finally, we discuss about some specific experiences in transforming rule languages and about relations to the model weaving approaches.

XML Interchange Metadata (XMI). XMI is one of the most related works to our approach. It is the OMG’s specification providing a set of rules for mapping between MOF-based models, metamodels, and metametamodels and XML. Although XMI allows for sharing MOF-based artifacts between various applications, it is rather a verbose solution that produces many XML elements and attributes and without ways for developers to define and use their own XML-based concrete syntax suitable for specific domains such as Web rule languages. In this paper, we have investigated how model transformation languages can be used to bridge this gap between concrete and abstract syntax of languages whose abstract syntax is defined by metamodels. A special advantage of this approach is that once we have mappings between an arbitrary concrete syntax of a language (e.g., R2ML, SWRL and OCL) and their corresponding abstract syntax, we also have mappings between that new arbitrary concrete syntax and the XMI-based concrete syntax of the languages (e.g., R2ML, SWRL and OCL).

Grammar-based Approaches. Bridging between the abstract and concrete syntax of languages would also be possible to be done in the EBMN technical space. Some possible option to undertake such research would be through the use of well-known languages such as TXL [48], Stratego/XT [49], and ASF+SDF [52]. There have already been attempts to these languages for model-to-model transformations such as transformations between Higher-Order Graphs and UML Sequence Diagrams by using [51] and between Process Algebra Models and UML State Machines [53]. It is important to no-
tice that both of these approaches focused on transformations between two different languages, and not on the problem of bridging between concrete and abstract syntax of the same languages as it is done in our approach. However, some lessons learned from [51] [53] provide us with a very solid observation framework for drawing conclusion about the suitability of grammar-based transformation approaches to the problem under study of our paper.

Let us explore the case presented in [51], where the authors first created the metamodels of both languages under study by using Eclipse’ metamodeling language Ecore (equivalent and very similar to MOF). In fact, they reused the part of the UML’s metamodel for sequence diagrams and created a metamodel for Higher-Order Graphs. Then, they created corresponding TXL grammars for both of the languages under study. These grammars were in fact formalization of the XMI concrete syntax. Once they defined those TXL grammars, they developed TXL transformations for mapping between two languages. This approach nicely illustrated how model-to-model transformations can be done in the EBNF technical space. However, looking now from the perspective of bridging between concrete and abstract syntax, the authors fully relied on the use of XMI, as the primary representation of languages. Thus, they did not leverage the core features of metamodeling with the already indicated limitation of XMI to fully assume that everyone will use only XMI-based concrete syntax of languages.

Besides an important tooling problem that almost every MOF/Ecore-based tool has to rely on their own interpretation of the XMI standard (i.e., it hardly happens that two tools can fully exchange models in XMI) [53], there is also a problem of other possible concrete syntax that might be used for languages at hand, as we have shown in the examples of the three explored languages. Thus, if there is a need for transformation between two different languages, the main problem is what pathway to use. This solution would imply to go via XMI that is each concrete syntax of the language to be transformed to the XMI-based XMI concrete syntax of the language under study. However, the main issue of such an approach is related to the core nature of metamodeling itself. Namely, metamodels as abstract syntax of languages are not only class models of languages, but they also include the use of constraints expressed in OCL. As authors reported in [51], they only used Ecore class related elements, but not constraints. Thus, during the transformation process, it is not possible to enforce such constraints unlike the approaches based on ATL tools used in this paper. Of course, the experience in the use of TXL also calls for the future investigation in bridging for XML- and text-based concrete syntax.

Engelen & van den Brand [13] proposed a method for integration of textual and modeling languages for UML Activity diagrams, by defining a surface language. The surface language allows one to define textually behavior of UML Activity diagrams. The proposed method includes grammar-based (text-to-text) and a model-based (model-to-text, text-to-model and model-to-model) approaches. The grammar-based approach uses ASF+SDF [43] for description of both the source and target language. That is, those are the UML activity models and the surface language for the UML activity models. A disadvantage of approach is that it deals with XMI representation of models, which lowers the level of abstraction in proposed transformations. In the model-based approach, parts of the surface language of UML activity diagrams are first transformed from XMI to textual models and then to models that can be used by model transformation tools, to be included in standard UML models. This also brings a new issue of creating and using model-to-text and text-to-model transformations, where in our approach we do not deal with XMI representations. Additionally, to work with text-to-model and model-to-text transformations, Engelen & van den Brand used multiple languages, such as Xpand and Xtext, while we created transformations in both directions by using only one language (TCS).

XML-based Approaches. Although the use of XML is very suitable, the previous analysis of the use of XSLT for sharing knowledge indicates that XSLT is hard to maintain where modifications of input and output formats can completely invalidate previous versions of XSLTs [24]. For example, some recent experiences in transforming rule languages (SWRLp) report on constraints of XSLT (e.g., to transform unique symbols) that can only be overcome by XSLT extensions implemented in other lan-
guages such as Java and Jess [28]. This is especially amplified when transforming highly verbose XML formats such as XMI [11]. Another important drawback of the XSLT approach is that XSLT does not have any language to check validity of the XML documents regarding XML Schema to which it conforms to during the execution of the transformation and it does not have a constraint language such as OCL in MDE TS. In addition, XSLT does not have strong support for data types that causes problems when debugging transformations. In the case of model transformation languages that we have analyzed, we demonstrated the benefits such as a good support for different technical spaces through XML and EBNF injection and extraction and advanced features for creating and using a richer set of transformation rules (matched, called, lazy, and unique).

**RDF-based Approaches.** For the RDF-based approach, we considered TRIPLE [6] as a proposal for an RDF transformation language. TRIPLE is a query, inference and transformation language based on Horn-logic and F-Logic [57] that focuses on RDF inference and transformation tasks. But, it is reported that as Horn rules are not powerful enough to represent the complete semantics of languages such as OWL, on which SWRL is based on [39]. In addition, TRIPLE read all the relevant (RDF) annotations before reasoning about them, so in the case of large data sources, it is obviously had to retrieve the entire content of such sources before starting reasoning (i.e., transforming) [1]. However, TRIPLE itself is not enough, as we have to implement a transformation between an RDF (graph) representation and MOF-based (another type of graph) representation. In fact, to apply this solution, we would need to have a similar tooling, as we explained for ATL (XML Injectors/Extractors).

The problem under study has already been addressed by several researchers, where they have proposed different solutions for bridging MOF-based abstract syntax (e.g., metamodels) and textual [23] and graphical [16] concrete syntax of languages. Given that all current solution are looking at this problem at a more general level, i.e., mappings between MOF-based metamodels and EBNF grammars, our experience shows that it would be useful to have their specialization of those approaches to directly deal with some common concrete syntax languages. In particular, we refer to XML, which is a special type of EBNF grammars. Recognizing this problem, the Semantic Web community is developing a special language for transformations between XML schemas and Semantic Web ontologies (i.e., so-called lifting and lowering [50]). Their current proposal is a specialized language, so-called XSPARQL, for transforming between RDF-based models and XML.

**Model Weaving.** One another relevant initiative in this area is related to model weaving [14]. The use of matching transformations and model weaving enabled to semi-automate the production of model transformations. Matching transformations are a special kind of transformations that implement heuristics and algorithms (such as element similarity and best links [14]) to create weaving models, while weaving models are models that capture different kinds of relationships between models in a weaving metamodel. The weaving model is translated into a model transformation language (such as ATL). This approach could enable, to some extent (we can say that variables cannot be supported in this way), bridging XML/RDF based concrete syntax with their abstract syntax (i.e., metamodels) by defining weaving models between the XML metamodel and the rule language’s corresponding metamodel elements.

9 Conclusions

In this paper, we examined current technologies that can be used for bridging between concrete and abstract syntax in the MDE context. We investigated the current state of transformation languages as a solution to this problem. Here we summarize our main contributions and lessons learned:

- First, the use of model transformation languages enforces the use of valid source and target models. This means that the transformation cannot be executed properly if either of models is not fully conformant to its metamodel where metamodels also include constraint definitions.
- Second, the use of languages such as ATL is more appropriate than XSLT (Sect. 3) when transforming between the XML and MDE technical spaces, since ATL supports advanced features (e.g., lazy

---

7 The weaving metamodel is a metamodel capable of representing correspondences and links between model elements [14].
and unique lazy rules) for transforming between languages based on metamodels (i.e., graphs) and XML-based concrete syntax (trees).

- Third, given an open need, we proposed a method along with a set of principles how to develop transformations for bridging between concrete syntax and abstract syntax along with the specific recommendations for specific transformation languages as reported in Section 5.

- Fourth, while an XML-based concrete syntax seems to be easier to create than regular text-based one, the current tooling support seems to be more advance for the latter case. As reported in Section 6, languages such as TCS provide bidirectional transformations for regular text-based concrete syntax, while the use of XML-based concrete syntax requires the development of uni-directional transformation chains (typically two transformations in either direction). This clearly indicates a need for the development of specialized languages for XML-based concrete syntax, especially given a rising popularity of XML.

- Finally, we reported on empirical analysis of the proposed approach based the data collected from our experiments in transforming rule languages (Section 7). Our study showed the amount of efforts needed to bridge the gap between the studied concrete and abstract syntaxes. That is, collected data demonstrated the number of transformation rules needed to be developed per class of the studied metamodels along with the overall number of the developed rules. Given that in MDE the problem of bridging between concrete and abstract syntax recently was identified, to the best of our knowledge, there have been no widely-adopted methods for evaluation and comparison with the other related work in the domain of MDE.

The use of statistical methods in analyzing model transformations seems very promising direction of the future research. For MDE to become a mature software engineering discipline, further investigation on this area is needed. Given that empirical research in MDE in its early stages [55], the conducted study could be used as a starting point in evaluations of similar solutions in the area. In the future, we plan to extend our empirical study by using more advanced statistical methods such correlations and regressions. Our goal is to analyze mutual relations between the definitions of the syntax definitions and constructs of transformation languages. We plan to make use of a rich set of transformations publicly available in the ATL repository [56]. Furthermore, we also need to investigate how we can compare results between different approaches that are based on different methods and transformation languages (i.e., those that do not follow our recommendations and technical framework). Finally, we plan to further investigate the relations between metamodeling and proven compiler construction methods for defining abstract syntax of software languages.

10 Acknowledgements

The research of Athabasca University has in part been support by Natural and Engineering Sciences Research Council of Canada.

11 References


9th International Conference the Model Driven Engineering Languages and Systems, LNCS 4199, Genoa, Italy, 2006, pp. 661-676.


38. REWERSE I1 Rule Markup Language (R2ML), http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6, 2009.


27

StrategoXT-0.9,” In C. Lengauer et al., editors, Domain-Specific Program Generation, LNCS 3016, Springer-Verlag, 2004, pp. 216-238.


Appendix 1. Glossary


MDE (Model Driven Engineering). A software engineering discipline where models are first class concepts for developing software systems and languages.

MOF (Meta-Object Facility). The OMG’s standard for a metamodeling language.

OCL (Object Constraint Language). The OMG’s standard for a constraint language to be used in software models.

ODM (Ontology Definition Metamodel). A metamodel of the Web Ontology Language (OWL). Also, the OMG’s standard for describing a family of ontology languages along with their accompanying transformations.

OWL (Web Ontology Language). A language officially recommended by the W3C consortium for representing ontologies.

QVT (Query View Transformation). The OMG’s standard language for model transformations.


RDM (Rule Definition Metamodel). A metamodel of the SWRL language.

RIF (Rule Interchange Format). The W3C’s initiative to define an intermediary language between various rule languages.

SWRL (Semantic Web Rule Language). A Web rule language defined on top of the OWL ontology language.

TS (Technical Space). A working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities.

TCS (Textual Concrete Syntax). A domain specific language for defining textual concrete syntax in MDE. It can be used for parsing text-to-model and serialization model-to-text.