Sharing OCL Constraints by Using Web Rules

Milan Milanović[1], Dragan Gašević[2], Adrian Giurca[3],
Gerd Wagner[3], and Vladan Devedžić[1]

[1] FON-School of Business Administration, University of Belgrade, Serbia
milan@milanovic.org, devedzic@etf.bg.ac.yu
[2] School of Computing and Information Systems, Athabasca University, Canada
dgasevic@acm.org
[3] Institute of Informatics, Brandenburg Technical University at Cottbus, Germany
Giurca@tu-cottbus.de, G.Wagner@tu-cottbus.de

Abstract. This paper presents an MDE-based approach to interchanging rules between the Object Constraint Language (OCL) and REWERSE I1 Rule Markup Language (R2ML). The R2ML tends to be a standard rule markup language by following up the W3C initiative for Rule Interchange Format (RIF). The main benefit of this approach is that the transformations between languages are completely based on the languages' abstract syntax (i.e., metamodels) and in this way we keep the focus on the language concepts rather than on technical issues caused by different concrete syntax. In the current implementation, we have supported translation of the OCL invariants into the R2ML integrity rules. While most of the OCL expression could be represented in the R2ML and other rule languages, we have also identified that collection operators could only be partially supported in other rule languages (e.g., SWRL).

1. Introduction

The Unified Modeling Language (UML) [31] presents a de-facto standard for modeling object-oriented systems. In the UML, various model elements like classes or state machines can be annotated by logical constraints defined by using the Object Constraint Language (OCL). In this way, UML models constrained by OCL expressions are more accurate and complete. The OCL is today used in a number of tools, and it is accepted as a standard by the OMG (Object Management Group); it can be also used to define constraints on MOF (Meta Object Facility)-based metamodels [20]. The OCL 2.0 specification [24] explicitly defines a concrete and an abstract syntax of the language, i.e., a MOF-based metamodel and a textual concrete syntax.

In the research community, there have been a lot of efforts to enable sharing UML models with other languages. One such effort is to share UML models and ontologies, and thus enable the reconciliation of the Semantic Web and software engineering communities [5][7][10]. However, none of the present efforts have so far considered the problem of sharing OCL constraints with other types of constraint or rule languages such as Semantic Web Rule Language [13], F-Logic, or Jess. This has a consequence that one can not translate OCL constraints defined on UML models into, for example, corresponding constraints defined over OWL ontologies.

Nevertheless, the W3C consortium started an initiative called Rule Interchange Format (RIF) [11], which tries to define a standard for sharing rules. That is, RIF should be expressive enough, so that it can represent concepts of various rule languages. Besides RIF, one should also develop a (two-way) transformation between RIF and any rule language that should be shared by using RIF. Currently, there is no official submission to RIF, but the RuleML [12] and the REWERSE I1 Rule Markup Language (R2ML) [32] are two well-known RIF proposals.

In this paper, we propose transformations between the R2ML and OCL to enable interchanging OCL rules with other rule languages via R2ML. However, we want our solution to be completely based on the abstract syntax of both languages, unlike other similar approaches proposed in the context of rule interchange [30] that mainly focus on a concrete syntax without efficient mechanisms to check whether the implemented transformations are valid w.r.t. the abstract syntax. In this paper, we propose using Model-Driven Engineering (MDE) principles and model transformations to address this issue. This means that we have to provide a two way mapping between the OCL and R2ML. The main benefit of such an approach is that we can actually map OCL constraints into all other rule languages (e.g., SWRL, Jess, F-Logic, and Prolog) that have mappings defined with R2ML. In our previous work [18], we defined technical requirements for fully implementation of this approach, while in this paper we focus on the details of the mappings between OCL and R2ML constructs. The mappings between the OCL and R2ML include those OCL constructs which are interchangeable with other rule languages, i.e., we have defined mappings of such OCL expressions that could be represented in rule languages for which we have already defined mappings.

2. Motivation

In order to motivate sharing rules expressed in the OCL and R2ML, let us consider the following UML model from Fig. 1 that represents an excerpt from the EU-Rent Vocabulary Business Context. EU-Rent is a car rental company owned by EU-Corporation and it is used as an example in the *Semantics of Business Vocabulary and Business Rules* (SBVR) standard [28]. At the UML class *Person*, there is a following OCL invariant defined: *a barred driver is a person known to EU-Rent as a driver who has at least 3 bad experiences*. This invariant is in a UML note attached to the *Person class* and shown on the UML diagram from Fig. 1.

Given the great diversity of rule concepts and existing rule languages, the R2ML metamodel consists of overlapping metamodels for the following types of rules: integrity, derivation, reaction, and production rules. This means, we first have to decide to what type of R2ML rules we should transform the above OCL constraint. Having in mind the nature of the OCL invariant above, which defines that something must be true for all instances of that type at any time, we actually should transform the above rule into an R2ML integrity rule, or more specifically an alethic integrity rule (see more details about notion of R2ML integrity rules in Sect. 4.1) [32]. In general, we can say that an OCL invariant, which is universally quantified formula over a set of objects corresponding to the context in a form of an alethic integrity rule (necessity),

can be translated to an R2ML rule. Due to the nature of the OCL invariants, it has to be translated onto the R2ML integrity rule. In Fig. 2, we show the OCL invariant from Fig. 1 in the R2ML XML-based concrete syntax. This R2ML alethic rule has a universally quantified formula as its constraint, while this universally quantified formula is an implication which is obtained from the OCL *implies* element. Mappings between the OCL invariant shown in Fig. 1 and the R2ML rule shown in Fig. 2, will be explained in detail in Sect. 4.3.

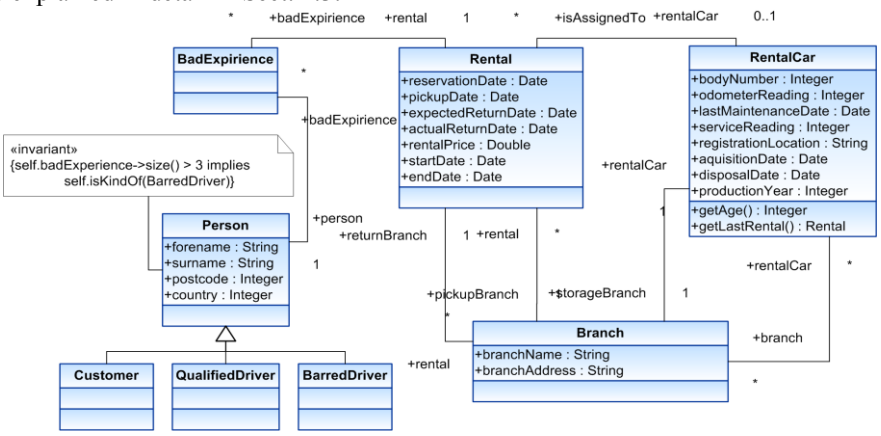

Fig. 1. OCL invariant and its corresponding UML class *Person* in the UML class diagram

Once we transform the OCL invariant into an R2ML alethic integrity rule, we can further transform it onto all other rule languages supporting integrity rules by exploiting the existing transformations for R2ML [26] (e.g., SWRL [18]). However, we should mention here that we have supported only those OCL constructs which can be translated into other rule languages (see Sect. 4.3 for details).

From the above facts, it is obvious that in both examples we use the concrete syntax of the languages (i.e., OCL and R2ML). However, a language is usually defined by its abstract syntax (i.e., metamodel), while concrete (visual or textual) syntax is employed to represent physically rules. Thus, defining and implementing mappings between languages should be done on the level of their abstract syntax, as this actually al-

```
<r2ml:AlethicIntegrityRule>
  <!--Namespace definitions are omitted to reduce the size of this example-->
  <r2ml:constraint>
    <r2ml:UniversallyQuantifiedFormula>
      <r2ml:ObjectVariable r2ml:name="person" r2ml:classID="ex:Person"/>
      <r2ml:Implication>
        <r2ml:antecedent>
          <r2ml:DatatypePredicateAtom
            r2ml:datatypePredicateID="swrlb:greaterThan">
            <r2ml:dataArguments>
              <r2ml:DatatypeFunctionTerm r2ml:datatypeFunctionID="fn:count">
                <r2ml:dataArguments>
                  <r2ml:AttributeFunctionTerm
                    r2ml:attributeID="ex:badExperience">
                    <r2ml:contextArgument>
                      <r2ml:ObjectVariable r2ml:name="person"
                                           r2ml:classID="ex:Person"/>
                    </r2ml:contextArgument>
                  </r2ml:AttributeFunctionTerm>
                </r2ml:dataArguments>
              </r2ml:DatatypeFunctionTerm>
              <r2ml:TypedLiteral r2ml:lexicalValue="3"
                                 r2ml:datatypeID="xs:integer"/>
            </r2ml:dataArguments>
          </r2ml:DatatypePredicateAtom>
        </r2ml:antecedent>
        <r2ml:consequent>
          <r2ml:ObjectClassificationAtom r2ml:classID="ex:BarredDriver">
            <r2ml:ObjectVariable r2ml:name="person" r2ml:classID="ex:Person"/>
          </r2ml:ObjectClassificationAtom>
        </r2ml:consequent>
      </r2ml:Implication>
    </r2ml:UniversallyQuantifiedFormula>
  </r2ml:constraint>
</r2ml:AlethicIntegrityRule>
```

Fig. 2. An R2ML (alethic) integrity rule equivalent to the OCL invariant from Fig. 1

lows us to focus on mappings between language constructs, rather than on the implementation details of their concrete syntax. Being driven by this approach, in the rest of the paper, we describe mappings between R2ML and OCL on the level of their abstract syntax, and yet bridge the gap between R2ML and OCL's abstract and concrete syntax by using MDE principles.

3. Model Transformations for Rules

In this section, we summarize transformation chain used to implement mappings between two languages, while the detailed discussion on the technical requirements is given in [18]. The first step (see Fig. 3) is between OCL rules (invariants) represented in the OCL concrete syntax (i.e., in the EBNF technical space) and models compliant with the OCL metamodel (in the MOF technical space) [24]. In the second step, the MOF-based OCL rules obtained (i.e., OCL models) are transformed to R2ML models compliant with the R2ML metamodel. In the third step, R2ML models are transformed into the XML models (i.e., instances of the XML metamodel) by using transformations that we created in our previous work for bridging between the R2ML abstract and concrete syntax [19]. Finally, in the fourth step, such XML models (from the MOF technical space) are serialized into the R2ML XML format (compliant with the R2ML XML Schema) by using the ATL XML Extractor tool.

Having in mind all the above transformations, we have the core of the solution that is based on abstract syntax, but we actually can transform between OCL invariants and R2ML XML-based rules.

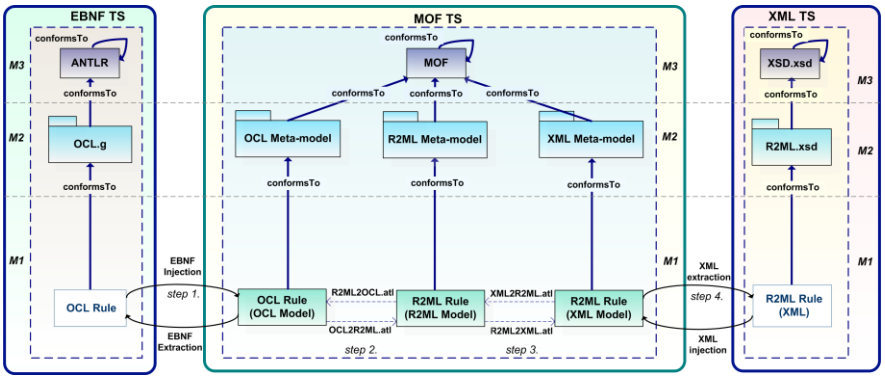

Fig. 3. The transformation scenario between OCL and R2ML

4. Mappings between R2ML and OCL

In this section, we first describe the parts of the R2ML abstract syntax relevant for representing OCL rules. We then describe the OCL abstract syntax, and finally, mappings between R2ML and OCL in detail.

4.1 R2ML Abstract Syntax

The R2ML metamodel is defined by using the MOF metamodeling language. R2ML supports four kinds of rules, namely, integrity rules, derivation rules, production rules, and reaction rules. R2ML covers almost all of the use case requirements of RIF [11]. Although OCL can represent both integrity constraints and derivation rules, we only describe R2ML integrity rules here. An *integrity rule*, also known as *(integrity) constraint*, consists of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL (see Fig. 4). The R2ML framework supports two kinds of integrity rules: the *alethic* and *deontic* ones. An alethic integrity rule can be expressed by a phrase, such as "*it is necessarily the case that*" and a deontic one can be expressed by phrases, such as "*it is obligatory that*" or "*it should be the case that*."

The corresponding *LogicalFormula* must have no free variables, that is, all the variables from this formula must be quantified. R2ML defines the general concept of *LogicalFormula* (see Fig. 5) that can be *Conjunction*, *Disjunction*, *NegationAsFailure*, *StrongNegation*, and *Implication*. The concept of a *QuantifiedFormula* is essential for R2ML integrity rules, and it subsumes existentially quantified formulas and universally quantified formulas. Fig. 5 also contains elements such as *AtLeastQuantifiedFormula*, *AtMostQuantifiedFormula*, and *AtLeastAndAtMostQuantifiedFormula* for defining cardinality constraints with R2ML rules. *Atoms* are basic constituents of formulas in R2ML, and they together with formulas correspond to the boolean OCL expressions. Atoms are compatible with all important concepts of UML and OCL. R2ML distinguishes object atoms, data atoms, and generic atoms.

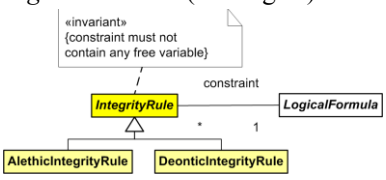

Fig. 4. The metamodel of integrity rules

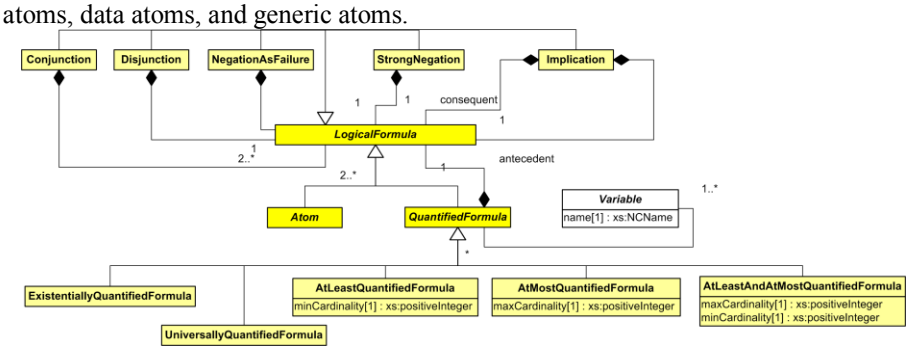

Fig. 5. The concept of a logical formula in R2ML

Terms are the basic constituents of atoms, which can be viewed as a first-order predicate-logic-based version of the OCL metamodel fragment of non-Boolean OCL expressions. Similarly to atoms, the R2ML language distinguishes between object terms, data terms and generic terms. We here describe only data terms due to the size limit, while object and generic atoms are defined in a similar way. A *DataTerm* is a *DataLiteral*, *DataVariable*, or *DataFunctionTerm* that can be *DataOperationTerm*, *AttributeFunctionTerm*, and *DatatypeFunctionTerm* (see Fig. 6).

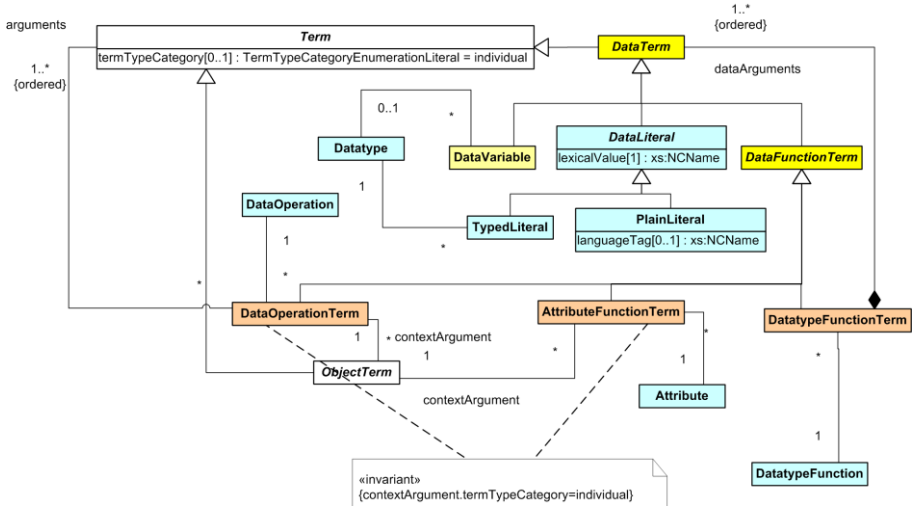

Fig. 6. R2ML Data Terms

A *DataOperationTerm* is formed with the help of a *contextArgument*, a user-defined operation, and an ordered collection of arguments. The *AttributeFunction-Term* corresponds to a datatype attribute in a UML class model. *DatatypeFunction-Term* is represented with *datatypeFunction* and *dataArguments*. *DataVariable* stands for plain data types, while *DataLiteral* can be *PlainLiteral* and *TypedLiteral* with some datatype.

4.2 OCL Abstract Syntax

The OCL metamodel (i.e., abstract syntax for OCL version 2.0) is also defined by using MOF [24]. In this abstract syntax, a number of meta-classes from the UML 2.0 metamodel are imported [31]. The OCL metamodel is divided into several packages: the *Types* package describes the concepts that define the type system of OCL. It shows the types predefined in OCL as well as the types that are deduced from the UML models; the *Expressions* package describes the structure of OCL expressions; and the *EnhancedOCL*[1] package that we have added to the standard OCL metamodel to represent invariant constructs that are not supported in the standard OCL.

The *Expressions* package defines kinds of OCL expressions. An overview of the inheritance relationships between all classes defined in the package is shown in Fig. 7. The basic structure of the package consists of the OCL metamodel classes such as *OclExpression* that is an abstract superclass for all OCL expressions; and *Feature–CallExp* that is superclass for the *OperationCallExp* and *PropertyCallExp* classes. *OperationCallExp* represents an operation defined on a *Classifier*, while *Property–CallExp* models a reference to an *Attribute* of a *Classifier* defined in a UML model.

[1] We are very grateful to Mr. Mariano Belaunde for his generous help in defining the *EnhancedOCL package*.

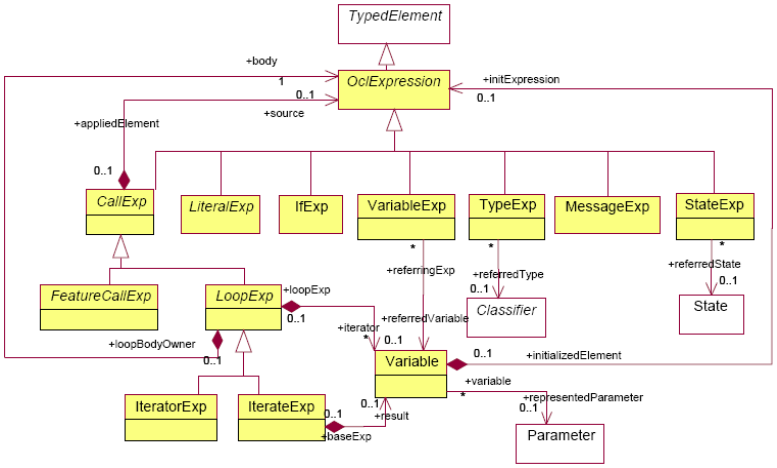

Fig. 7. The basic structure of expressions in the OCL metamodel

Since the standard specification of the OCL metamodel [24] does not contain support for OCL invariants, we introduced the *EnhancedOCL* package. This package contains the *Invariant* class, as a subclass of the *OclModuleElement* class (see Fig. 8, white classes are from the UML metamodel, white gray colored ones are from the standard OCL metamodel and dark gray are classes that we have defined).

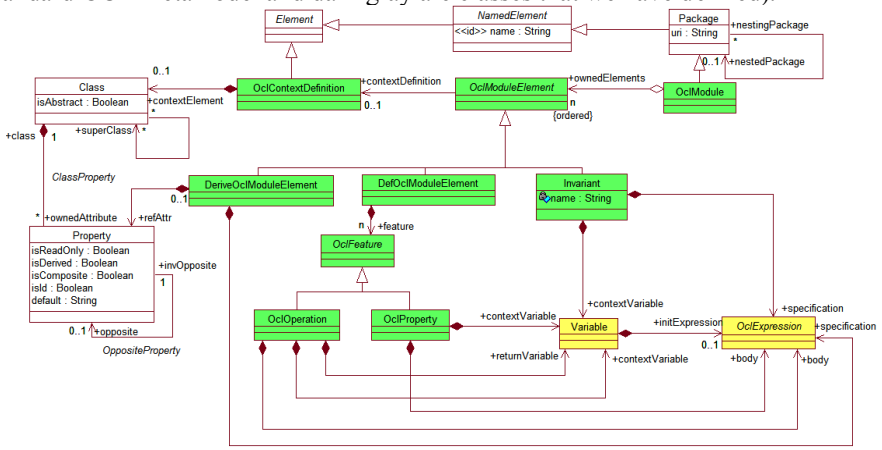

Fig. 8. Elements of the EnhancedOCL package in the OCL metamodel

OclModuleElement represents a superclass for following elements: OCL invariant elements (represented with the *Invariant* class); OCL operations and properties, i.e., “def” elements (represented with the abstract class *OclFeature*) that are represented with classes *OclOperation* and *OclProperty*, respectively; and OCL derivation rules, i.e., "derive" elements (represented with class *DeriveOclModuleElement*).

OclModuleElement contains a definition of an invariant context that is represented with the *OclContextDefinition* class. In addition, the *OclModule* class is introduced to

represent a basic class in an OCL model, and it contains other *OclModuleElement*s. We also added the *OperatorCallExp* class into the standard expression package of the OCL metamodel to represent operation call expressions that use operators, and it inherits *OperationCallExp* class. We had to do this in order to be able to develop the OCL parser that fully covers all OCL constructs. Due the limited size for this paper, we do not show these classes and their relations.

4.3 Conceptual mappings between OCL and R2ML

In order to share rules between OCL and R2ML, we have defined isomorphic mappings between certain constructs of OCL and R2ML on the level of their abstract syntax. Every OCL invariant which is in the form of an OCL *implies* is mapped to an R2ML *AlethicIntegrityRule* whose *constraint* is a *UniversallyQuantifiedFormula*, while its formula is an *Implication* mapped from an OCL *implies* element. Besides OCL *implies* expression, we have supported *all other OCL expression* that can be written in invariants. We further expand the mappings between OCL and R2ML to specify mappings between elements that are part of OCL and R2ML expressions. In Table 1, we give an excerpt of the mappings that we defined between both languages metamodels. The complete mappings between the R2ML and OCL metamodels contain 37 rules. In the rest of this subsection, we describe the mappings of the main OCL expressions, which could be used in invariants, into the R2ML. As an illustration, we refer to the EU-Rent case study shown in Fig. 1. (N.B. *every OCL invariant expression is defined in the context of a UML Class* such as *context RentalCar*).

- **OCL attribute** (e.g., *self*.age), which is represented in the OCL metamodel as *OperatorCallExp*, is mapped into an R2ML *AttributionAtom*, which consists of an object term as "subject" and a data term as "value". For example, *startDate(r1, sd)*, where *startDate* is an attribute, *r1* is an object term (subject), and *sd* is a data term (data value). Note that the *AttributionAtom*, as well as all other R2ML atoms, inherit the *Atom* class shown in Fig. 5.
- **OCL operation call**, which is represented in the OCL metamodel as *OperationCallExp*, is mapped:
  - to R2ML *DataOperationTerm*, if the operation call returns a primitive OCL datatype. The *DataOperationTerm* refers to a non-state-changing user-defined *Operation* and consists of a list of data or object arguments and an object term as a context argument. The result of an *Operation* is a data term (value). An example of a *DataOperationTerm* is *x.getAge()*, which returns the *age* of a car *x*. The *DataOperationTerm* corresponds to a Java path expression calling an operation which returns a datatype as result (e.g., int).
  - to R2ML *ObjectOperationTerm*, if the operation call returns an object. The *ObjectOperationTerm* refer to a non-state-changing operation. The *ObjectOperationTerm* may have data terms and object terms as operation arguments and is evaluated to an object. For example, the expression *x.getLastRental()*, which returns the last rental of a rental car, is an object operation term, where *getLastRental()* denotes an operation and *x* is the context argument (see Fig. 1). The *ObjectOperationTerm* corresponds to a Java path expression calling an operation which returns an object as result.

- A **UML association** is navigated in the OCL by using its opposite association end (in the OCL metamodel, this is represented with *PropertyCallExp*). If the maximal multiplicity of the association end is 1 (i.e., cases "0..1" or "1"), then the value of this expression is an object, and such an expression is mapped into an R2ML *ReferencePropertyFunctionTerm*. The R2ML *ReferencePropertyFunctionTerm* is a function, which returns the value of an association end for a given object. For example, the expression *x.pickupBranch*, where *pickupBranch* is an association end name of a Branch in the association between classes *Rental* and *Branch* (see Fig. 1) is a reference property function term, where *x* is an object term and *pickupBranch* is a reference property. However, if the multiplicity of the association end is more than one ("*"), then the navigation will result in a Set, and such an expression is mapped into an R2ML *AttributeFunctionTerm* (with "set" as the type category). The *AttributeFunctionTerm* refers to an attribute and an object term as a context argument. For example, the expression *x.reservationDate* is an attribute function term, where *x* is an object term and *reservationDate* is an attribute of class *Rental*. When the association is adorned with *{ordered}*, the navigation results in an *OrderedSet* and in this case the *AttributeFunctionTerms* type category is "orderedSet." For two other kinds of collections (*Bag* and Sequence), the *AttributeFunctionTerm* has corresponding type categories.
- OCL collections may have a large number of predefined **collections operations** on them (e.g., the *size* operation, which returns the number of elements in a collection, or the *isEmpty* operation, which returns *true* if the collection is empty or *false* otherwise). These operation calls (represented in the OCL metamodel with *OperationCallExp*) are mapped into the R2ML *DatatypePredicateAtom*s, in the case when on the left side of the equality operator is collection operation call and on the right side is any other OCL expression (e.g., *badExpirience->size()=1*). The operation calls are also mapped into the R2ML *DatatypePredicateAtoms*, in the case when a collection operation call on an association end (with multiplicity more than one) is evaluated to a boolean value (e.g., *badExpirience->isEmpty*). The *DatatypePredicateAtom* describes a relation between several data terms, using a data predicate which represents a SWRL built-in function [13] that is translated from the operation on a collection. For example, *self.badExpirience->isEmpty()* is translated into an R2ML *DatatypePredicateAtom* where „*swrlb:empty*" is data predicate and *self* is an *AttributeFunctionTerm* (with the *Person* class as an object term) with "bag" as the type category. In the case of the *notEmpty* operation, the *DataTypePredicateAtom* is negated (with the property *isNegated*, which is set to *true*). However, in case when the collection operation call is used in an expression with comparison operators to some other evaluated expression (e.g., *self.badExperience->size()>3*), the collection operation call is mapped into an R2ML *DatatypeFunctionTerm*, where the collection operation call is translated into the (XPath) datatype function. We have made this decision because comparison operators (other than equality) are mapped into *DatatypePredicateAtom*s whose arguments must be terms. For example, the OCL *size* operation is translated into the XPath *count* operation. Besides this, we have also a special case when the collection operation call returns just one element, but not the entire collection (e.g., the *first* operation), in which case the operation call is mapped into an R2ML *ObjectOperationTerm*.

- **OCL equality operation** between two association ends, which is represented in the OCL metamodel with *OperationCallExp*, is mapped into an R2ML *ReferencePropertyAtom*. The *ReferencePropertyAtom* associates object terms as "subjects" with other object terms as "objects." For example, *returnBranch(r1, rb)*, where *returnBranch* is a reference property and *r1* (subject) and *rb* (object) are object terms. In the case of the inequality operator, the property *isNegated* of the *ReferencePropertyAtom* is set to *true*. Note that translation of any negated OCL expression (denoted with "*not*" operator) into an R2ML atom is done by setting property *isNegated* of such atom to *true.*
- **OCL *oclIsKindOf(t)*** operator, which is a property that determines whether *t* is either the direct type or one of the supertypes of an called object, is mapped into an R2ML *ObjectClassificationAtom*. The *ObjectClassificationAtom* consists of a class type (as "base type") and an object term, variable, constant or function term. *ObjectClassificationAtom* accommodates the concept of an *OperationCallExp* in the OCL metamodel with a *TypeExp* argument (e.g., Rental(r1)).
- **OCL *implies* operation**, which is represented in the OCL metamodel as *OperationCallExp*, is mapped into an R2ML *Implication*. The R2ML *Implication* consists of an antecedent (body) and a consequent (head), each of which consists of a set of atoms.
- In R2ML, functions range over individuals, like in standard first-order predicate logic. Since OCL allows function terms (such as *navigation call expressions*) to range over sets (more precisely, *collections*) and because the current R2ML metamodel does not support collections, the R2ML only captures a fragment of the OCL collection expressions. This will be subject for the future work to allow set-valued functions in the R2ML metamodel and then to provide a full support for such OCL expressions.
- In the current implementation, we have partially supported following OCL **collection operations**: *select*, *reject*, *includesAll*, and *forAll*. Due to size of this paper we do not describe mapping of every operation in detail. As an example, we may say that the *select* operation (represented in the OCL metamodel as *IteratorExp*), which specifies a subset of a collection, is mapped into an R2ML *Conjunction* of an *AttributionAtom* and *ExistentiallyQuantifiedFormula*. The *AttributionAtom* represents a mapping of an association end which is a collection, and the *ExistentiallyQuantifiedFormula* is mapped from the *select*'s boolean expression (also, iterator variables are mapped into the R2ML *GenericAtom*s). Note that there a constraint here, that is, we have only supported the translation of the following *select* construct: *collection->select(v | boolean-expression-with-v)*, where *v* is called iterator variable. When the select construct is evaluated, *v* iterates over the *collection* and the *boolean-expression-with-v* is evaluated for each *v*. The *v* variable is a reference to the object from the *collection* and can be used to refer to the objects themselves from the *collection* (e.g., *self.employee->select(age>10)->notEmpty()*). In a similar way, we have mapped other collection operations.
- **OCL *tuple* type**, which is used to compose several values and consists of named parts and which is represented in the OCL metamodel with *TupleLiteralExp*, is mapped into the R2ML *ObjectDescriptionAtom*. The *ObjectDescriptionAtom* refers to a class as a base type and to zero or more classes as categories, and consists of a number of property/term pairs (i.e., R2ML attribute data term pairs and refer-

ence property object term pairs). Any instance of such atom refers to one particular object, that is referenced by an objectID, if it is not anonymous.

Using the mappings between OCL and R2ML presented in above and shown in Table 1, we now illustrate the transformation process by using the example of the OCL rule from Fig. 1 and its corresponding R2ML rule from Fig. 2. As we have already mentioned, the OCL invariants (*Invariant* elements from Fig. 8) are transformed into R2ML integrity rules (*AlethicIntegrityRule* elements from Fig. 2). An OCL *Implies* element (i.e., *OperationCallExp* class with name "implies") is transformed to an R2ML *AlethicIntegrityRule* (shown in Fig. 4) with *UniversallyQuantifiedFormula* element as its *constraint*, where *UniversallyQuantifiedFormula* has an *Implication* for its formula. The R2ML *ObjectVariable* in the R2ML *UniversallyQuantifiedFormula* element is obtained by transforming the contextual class (*Class* element) from the OCL *Invariant* element (shown in Fig. 8). As it is shown in Table 1, an OCL operator call expression (*OperatorCallExp* element) which with name "=", is transformed into an R2ML *AttributionAtom* element, where the OCL *OperatorCallExp*'s source element is transformed into the *AttributionAtom*'s *attribute* element (i.e., *Attribute* class). The R2ML *AttributionAtom*'s *dataValue* element is obtained from the OCL *OperatorCallExp*'s argument element. The OCL *OperatorCallExp* ">" is transformed into an R2ML *DatatypePredicateAtom*, and an OCL *OperationCallExp,* represents an operation call, is transformed into an R2ML *DataOperationTerm*, if such an operation returns a data value, otherwise, it is transformed to an R2ML *ObjectOperationTerm*.

Table 1. An excerpt of mappings between the R2ML metamodel elements and the OCL metamodel elements

R2ML metamodel	OCL metamodel
RuleBase	OclModule
AlethicIntegrityRule	Invariant
Conjunction	OperatorCallExp (name = 'and')
Implication	OperationCallExp (name = 'implies')
AttributionAtom	OperatorCallExp (name = '=') source = PropertyCallExp (subject)
ObjectVariable	VariableExp
ReferencePropertyFunctionTerm	PropertyCallExp referredProperty (name = 'property') source = VariableExp
ObjectOperationTerm	CollectionOperationCallExp
DatatypePredicateAtom	OperatorCallExp (name = ">") source = OperationCallExp
DataOperationTerm	OperationCallExp

5. Implementation Experience

In this section, we explain the transformation steps undertaken to transform between OCL invariants and R2ML integrity rules. Here we refer to Fig. 3 from Section 3.3 in order to position each specific transformation/step in this process of transformation. As we have already mentioned in Section 3.3, the transformation process between R2ML and OCL is split into four major steps.

Step 1. In this step, we bridge between the OCL (EBNF-based) concrete syntax and the OCL abstract syntax (i.e., OCL metamodel). Because the OCL textual concrete syntax is located in the EBNF technical space, we need to create an instance of the OCL metamodel (abstract syntax) in the MOF technical space. To do this, we first use the EBNF injector, (see Fig. 3, step 1: EBNF injection), a part of the ATL toolkit, and the OCL Lexer and Parser. We generated the OCL Parser and Lexer by using the TCS (Textual Concrete Syntax) tools which are also part of the ATL toolkit [15]. TCS represents *domain specific language* (DSL) for defining textual concrete syntax in MDE. The OCL Parser automatically transforms OCL invariants like the one given in Fig. 1 into the models conforming to the MOF-based OCL metamodel. Once we created the OCL TCS and generated OCL Parser and Lexer based on it, the EBNF injector takes for input the OCL metamodel, OCL code that we want to parse (as *.ocl* textual file), generated OCL Lexer and Parser, and it returns a MOF-based OCL model as output. Once we inject OCL invariants into a MOF-based representation (OCL Rule in Fig. 3), we can manipulate with them like with any other MOF-based model.

Step 2. This step is the core of our transformation between the OCL abstract syntax (i.e., OCL metamodel) and the R2ML abstract syntax (Fig. 3, step 2). This transformation step is fully based on the conceptual mappings between the elements of the OCL and R2ML metamodel described in Section 4.2. The transformations between the OCL metamodel and the R2ML metamodel are defined as a sequence of rules in the ATL language (see Fig. 3, OCL2R2ML.atl and R2ML2OCL.atl).

Step 3. In order to serialize the R2ML model (from the MOF technical space) that is obtained in the previous step into the R2ML XML concrete syntax (i.e., to the XML technical space), we first need to use the R2ML2XML.atl transformation (Fig. 3, step 3) to get an XML model from R2ML model. After applying this transformation to the input R2ML model XML models are stored in the model repository (R2ML rule - XML model from Fig. 3). The output XML model conforms to the XML metamodel. Such XML model is serialized into the R2ML XML format in the next step. [19] gives details about bridging the R2ML concrete and abstract syntax.

Step 4. The step is the XML extraction from the MOF technical space to the XML technical space (Step 4 in Fig. 3). We transform the XML model (shown in Fig. 3) which conforms to the MOF-based XML metamodel and is generated in step 3 to an R2ML rule represented the R2ML XML concrete syntax, which is shown in Fig. 2.

An R2ML rule in the R2ML XML concrete syntax can be transformed into some other language for which there is a translator defined with the R2ML language [27] [25]. We have also defined the opposite transformations, from the XML metamodel into the R2ML metamodel (XML2R2ML.atl in Fig. 3), and from the R2ML metamodel into the OCL metamodel (R2ML2OCL.atl in Fig. 3). As the ATL toolkit has the XML Injector tool, that can transform an R2ML rule from the R2ML XML concrete syntax into the XML metamodel (i.e., the MOF technical space), such an XML model can then be transformed into an R2ML model by using the XML2R2ML.atl transformation. As well, by using the R2ML2OCL.atl transformation, we can transform that R2ML model into the OCL model. The ATL toolkit has also the EBNF extractor tool that can extract an OCL model (i.e., from the MOF technical space) into the OCL concrete textual syntax by using the OCL TCS that we defined. In this way, we have enabled a round-trip engineering between the R2ML general rule interchange language and the OCL language).

6. Discussion

The transformations implemented between the OCL and R2ML abstract syntax comprise translation of the OCL invariants. However, we have yet not finalized the implementation of all OCL *iterator* construct variations (e.g., select, forAll, and collect), because those constructs do not exists in other rule languages, and thus R2ML. Our current transformations do not support the full transformation of all UML class related elements (e.g., associations) to R2ML. In the future, we plan to support fully the translation of all UML (core) model elements into the R2ML Vocabulary. This will enable us to recognize property types in the OCL textual concrete syntax (e.g., when a property is referenced via another property). With the currently implemented solution, we can translate an OCL invariant from Fig. 1 into R2ML rule (see Fig. 2) and then into some other language for which there is a transformation with R2ML already defined. For example, we can translate that OCL invariant into a SWRL rule [13], as we have defined transformations between SWRL and R2ML [18].

As we have shown in this paper, our transformations can translate OCL invariants into the R2ML integrity rules. However, our OCL Parser also supports OCL derivation rules (i.e., "*derive*" expressions), and we plan to extend our transformations between OCL and R2ML to enable for the translation between OCL derive rules and R2ML derivation rules. Generally, this will only require adding rules for translating head of OCL derive rules, since their body expression is the same as in invariants (and that is represented with the *OclExpression* element in the OCL metamodel). Once we support derivation rules, it will be possible to translate OCL rules into F-Logic, Jess, RuleML, which are supported by the present R2ML translators for derivation rules [27].

We have tested our transformations between OCL and R2ML on 25 OCL invariant examples which included all of the OCL expressions described in Sect. 4.3. Those OCL invariants are collected from different sources such as EU Rent case study [9], Warmer and Kleppe' book [25], and Dresden OCL Toolkit (i.e., OCL test and demonstration constraints) [8]. All these OCL invariants are also translated to SWRL [13] via R2ML. Note also that the complete source code of the transformations presented can be found in the ATL Transformations Zoo [1] [2].

7. Conclusion

In this paper, we have shown how to transform rules between R2ML and OCL by employing model transformation principles. We have mapped OCL invariants into the R2ML integrity rules, and thus we enabled sharing OCL invariants with other rule languages. In the current implementation, we support only OCLL invariants, but in the next versions of our transformation, we plan to support other kinds of the OCL constraints (i.e. *derive*, *init*, *pre-* and *post- conditions*), which we have already supported in the TCS-based parser and lexer for OCL. The mappings between R2ML and OCL do not cover OCL collection operators completely, but just a basic ones (as it has been shown in Sect. 4.3). We have made this decision since all of the OCL collection operations could not be represented in the R2ML, because the R2ML does not support collection operators as OCL does. Note that the desing of the R2ML is based

on a hypothesis that most of web rule languages (e.g., F-Logic, Jess, JenaRules, ILOG JRules, and JBoss Rules) do not have collection operators supported in OCL. The transformation implementation is done by using the ATL (between OCL and R2ML, and the R2ML MOF-based abstract syntax and R2ML XML schema) and the TCS (between the OCL MOF-based abstract syntax and the OCL textual concrete syntax).

The solution presented in this paper represents the first practical example of approaching Web and Software engineering rule and constraints standards, after the activities done in the ODM standardization [10]. To the best of our knowledge, there is no available solution of mapping between Web rule languages and OCL, and thus our solution represents an important contribution to the further reconciliation of the software engineering and Web communities. We hope that our results will stimulate collaborative research of the two communities, so that the designs of rule languages (e.g., RIF) will integrate needs and best practices of both communities. For example, in this paper, we demonstrated that the current Web rule languages (R2ML and RIF) do not have support for advance OCL collection operators, and this could be an important input of the OCL community to the RIF standardization efforts. This could the allow developers to leverage OCL when model Semantic Web applications.

A similar approach to ours is applied in the ODM specification [21] where the (model) transformations between OWL and the languages such as UML, Topic Maps, and ER models are defined on the level of their abstract syntax (i.e., metamodels). Our solution goes one step further and demonstrates how to bridge between concrete and abstract syntax of rule languages. Besides the obvious benefit of developing transformations between rule languages on the level of abstract syntax, the use of model transformations and languages such as ATL is more suitable than XSLT. Although, in principle, we could use XSLT to map between abstract syntax thanks to XMI in which all MOF-based metamodels can be stored, the available 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 [14].

We are now in the phase of the evaluation of the results of the translation between the OCL and R2ML languages, and potentials for sharing rules between OCL and other rule languages via R2ML. In this paper, we just reflected on an exchange with the SWRL language, while our subsequent analysis will fully explore this exchange and exchange of OCL constraints with other relevant rule languages such as Jess, Jena, and F-Logic. In our future publications, we are going to report on transformation implementation in more detail and evaluation results. We also plan to use this approach to provide mappings between R2ML, Web services, and policy rule-based languages (e.g., KAoS and Rei). This will enable modeling Web services and policies by using MDE principles and will be a further reconciliation of Web research efforts with MDE principles.

8. References

1. ATL Scenario OCL to R2ML, http://www.eclipse.org/m2m/atl/atlTransformations/#OCL2R2ML.
2. ATL Scenario: R2ML to OCL: http://www.eclipse.org/m2m/atl/atlTransformations/#R2ML2OCL.

3. ATL Scenario: R2ML to SWRL, http://www.eclipse.org/m2m/atl/atlTransformations/#R2ML2SWRL.
4. ATLAS Transformation Lang. (ATL). http://www.sciences.univ-nantes.fr/lina/atl, 2006.
5. Baclawski, K., et al. "Extending the Unified Modeling Language for ontology development," *Software and Systems Modeling*, Vol. 1, No. 2, 2002, pp. 142-156.
6. Bézivin, J. "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171-188, 2005.
7. Cranefield, S. "UML and the Semantic Web," *In Proc. of the Int'l Semantic Web Work. Symp.*, Stanford University, CA, USA, 2001.
8. Dresden OCL Toolkit, http://dresden-ocl.sourceforge.net.
9. EU Rent Case Study, http://www.eurobizrules.org/ebrc2005/eurentcs.
10. Gašević, D., Djurić, D., Devedžić, V., *Model Driven Architecture and Ontology Development*, Springer, Heidelberg-Berlin, 2006.
11. Ginsberg, A. "RIF Use Cases and Requirements," *W3C Working Draft*, http://www.w3.org/TR/rif-ucr/, 2006.
12. Hirtle, D., et al. "Schema Spec. of RuleML 0.91," http://www.ruleml.org/spec/, 2006.
13. Horrocks, I., et al. "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," *W3C Member Submission*, http://www.w3.org/Submission/SWRL, 2004.
14. Jovanović, J. & Gašević, D. "XML/XSLT-Based Knowledge Sharing," *Expert Systems with Applications*, vol. 29, no. 3, 2005, pp. 535-553, 2005.
15. Jouault, F., Bézivin, J., Kurtev, I., "TCS: Textual Concrete Syntax", *In Proceedings of the 2nd AMMA/ATL Workshop ATLAS group (INRIA & LINA)*, Nantes, France, 2006.
16. Kurtev, I., Bézivin, J., & Aksit, M. "Technological Spaces: an Initial Appraisal," *In Proceedings of the CoopIS, DOA'2002 Federated Confs.*, Industrial track, Irvine, USA, 2002.
17. Miller, J. & Mukerji, J., Eds. "MDA Guide Version 1.0.1," *OMG Doc. omg/03-06-01*, http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2003.
18. Milanović, M., et al. "On Interchanging between OWL/SWRL and UML/OCL," *In Proc. of the 6th WSh on OCL for (Meta-)Models in Multiple Application Domains*, pp. 81-95, 2006.
19. Milanović, M., et al. "Bridging Concrete and Abstract Syntax of Web Rule Languages", *In Proc. of the 1st Int'l Con. on Web Reasoning and Rule Systems*, Innsbruck, Austria, 2007.
20. Meta Object Facility (MOF) Core, v2.0. *OMG Document formal/06-01-01*, http://www.omg.org/cgi-bin/doc?formal/2006-01-01, 2006.
21. OMG Ontology Definition Metamodel (ODM), Sixth Revised Submission. *OMG Document ad/2006-05-01*, http://www.omg.org/docs/ad/06-05-01.pdf, 2006.
22. MOF QVT Final Adopted Specification. *OMG document 05-11-01*, http://www.omg.org/docs/ptc/05-11-01.pdf, 2005.
23. Meta Object Facility (MOF) 2.0 XMI Mapping Specification, v2.1. *OMG Document formal/2005-09-01*, http://www.omg.org/cgi-bin/doc?formal/2005-09-01, 2005.
24. OMG Object Constraint Language, OMG Specification, Version 2.0, formal/06-05-01, http://www.omg.org/docs/formal/06-05-01.pdf, 2006.
25. Warmer, J., Kleppe, A., *The Object Constraint Language: Getting Your Models Ready for MDA*, *Second Edition*, Addison Wesley, 2003.
26. REWERSE I1 Rule Markup Language (R2ML). http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6, 2006.
27. R2ML Translators. http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15, 2006.
28. OMG Semantics of Business Vocabulary and Business Rules (SBVR), Revised Submission to BEI RFP br/2003-06-03, http://www.omg.org/docs/bei/05-08-01.pdf, 2005.
29. Seidewitz, E. "What Models Mean," *IEEE Software*, vol., 20, no.5, pp. 26-32, 2003.
30. Translator from RuleML to Jess, http://www.ruleml.org/jess/, 2006.
31. OMG Unified Modeling Language 2.0, Docs. formal/05-07-04 & formal/05-07-05, 2005.
32. Wagner, G. et al. "A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL," *In Proc. of the WSh of Reasoning on the Web*, Edinburgh, UK, 2006.