Defining a subset of OCL for expressing SWRL rules

Sergey Lukichev [1]

Abstract. OCL is a rich-syntax language for expressing integrity rules and many business rules can be expressed using OCL. On the other hand, UML/OCL is a mainstream modeling technology and adopting it for expressing rules in the upcoming Semantic Web is a good practical issue: rules can be initially expressed in UML/OCL and then mapped into OWL/SWRL. Since UML/OCL is more expressive than OWL/SWRL it is good to define a subset of OCL, which can easily be mapped into the SWRL. In this paper we define this subset of OCL, called OCL-Lite, by building a mapping from SWRL into OCL. In addition, we briefly sketch the correctness problem of this mapping in terms of language interpretations.

1 Introduction

There are two issues we have to motivate: *i*). Why the subset of OCL, which can be mapped into SWRL is needed? *ii*). Why we build the mapping from SWRL to OCL in order to define this subset?

UML/OCL [1] is widely used among software engineers, while the area of the Semantic Web lacks for modeling methodologies and tools, comparable in popularity and maturity with UML/OCL. Therefore, it is natural to reuse existing UML/OCL methodologies and tools for modeling Semantic Web rules, which are expressed in the Semantic Web Rule Language (SWRL) [3]. The first arising problem with mapping OCL constraints into SWRL rules is a rich OCL syntax. It allows variety of expressions and not all of them can be mapped into SWRL. There are a number of works on sharing rules between OWL/SWRL and UML/OCL [7], [5], [6]. However, these works are mainly focused on the technical side of the mapping, in particular on the implementation of the metamodel transformations using Query View Transformations (QWT). The main question, which remains open in these works is what is exact subset of OCL, which can be mapped into SWRL? In this work we bridge this gap by defining a subset of OCL, called OCL-Lite, which can be mapped into SWRL.

In order to define OCL-Lite, we build a mapping from SWRL to OCL. The expressivity of OCL is higher than the first order logic since closures over object relations can be expressed in OCL but not in FOL [4]. Moreover, the syntax of SWRL is simpler than OCL, therefore we build the mapping from the less expressive language to the more expressive one, which is a natural method in determining the mappable subset of OCL.

The paper is structured as follows: section 2 contains the mapping from SWRL to OCL, in section 3 the syntax of OCL-Lite is specified and section 4 defines the problem of the semantical correctness of SWRL to OCL mapping and lists open issues for the further research.

[1] Institute of Informatics, Brandenburg University of Technology at Cottbus, Email: Lukichev@tu-cottbus.de

2 Translating SWRL rules into OCL invariants

First, we remind the SWRL abstract syntax, which is originally defined in [3]. By `I = Imp(Antec(A), Conseq(C))` we denote a SWRL rule where `A` and `C` are conjunctions of SWRL atoms. By `classAtom(CD, t)` we denote a SWRL class atom, where `CD` is a class description, as defined in [2] (Section 2.3.2.3. OWL DL Restrictions) and `t` is an object variable or an individual. By `individualPropertyAtom(p, o1, o2)` we denote a SWRL individual property atom, where `p` is a reference to a property, and `o1` and `o2` are object variables. By `A = datavaluedPropertyAtom(p, obj, val)` we denote a SWRL datatype property atom, where `p` is a reference to an datatype property, `obj` is an object variable and `val` is a data value or a data variable.

According to the semantics of SWRL [3], SWRL rules are integrity constraints in the form of logical implications.

Let `I = Imp(Antec(A), Conseq(C))` be a SWRL rule, then the mapping T transforms it into an OCL invariant as follows:

```
T(I) := context class(X) inv:
T(A) implies T(C)
```

The context class `X` of the resulting OCL invariant is class of a universally quantified variable in the rule. This variable then is renamed to **self**. If the rule has more than one universally quantified variable, then `X.allInstances()->forAll(y|e(y))` expression is used recursively, where `X` is a class of the universally quantified variable and `e(y)` is the OCL boolean expression, resulted from the SWRL rule.

In fact, using the SWRL syntax, it is possible to express a rule, which cannot be mapped into a syntactically correct OCL invariant. In particular, this is because SWRL supports data variables, which are not explicitly supported in OCL. For instance, if we have a conjunction of two SWRL atoms: `individualPropertyAtom(age, o, x)` and `builtinAtom(>, x, 18)`, where `x` is a data variable, then this corresponds to the OCL expression `o.age>18`, i.e. we have one OCL expression from the conjunction of two SWRL atoms. If the built-in atom is dropped, then the SWRL rule still remains syntactically correct, while the resulting OCL expression will have unknown symbol `x`. We do not define such possible mappings in this paper and focus on the direct mappings of SWRL atoms.

We do not describe how URIs are mapped into names of UML classifiers since this task is about the mapping from OWL to UML while we define the mapping of SWRL rules into OCL invariants.

The body and the head of a SWRL rule is a conjunction of SWRL atoms, therefore if `A = A1, ..., An`, then

```
T(A) := T(A1) and ... and T(An)
```

Let us consider the mapping of SWRL atoms.

- If `A = classAtom(id, t)` is a SWRL class atom, where `id` is an URI reference of the class and `t` is an object variable, then

```
T(A) := t.oclIsKindOf(id)
```

- If `A = individualPropertyAtom(p, o1, o2)` where `p` is a reference to an individual property, `o1` and `o2` are object variables, then

```
T(A) := o1.p=o2
```

- If `A = datavaluedPropertyAtom(p, obj, val)` where `p` is a reference to an individual property, `obj` is an object variable and `val` is a data value or a data variable, then

```
T(A) := obj.p=val
```

- If `A = builtinAtom(builtin, {obj})` where `builtin` is a SWRL builtin and `{obj}` is a list of datatype terms, then the resulting OCL expression is of type Boolean and depends on the built-in, for instance, if `builtin` is '+' then `A = builtinAtom('+', a,b,c)` and `T(A) := a=b+c`.

If `A = classAtom(CD, t)` is a SWRL class atom with an OWL description `CD`, then we define the mapping function case wise for each possible OWL class description `CD` (the syntax of OWL class descriptions is defined in [2]):

- If `CD = UnionOf(CD1, CD2)` where `CD1` and `CD2` are OWL class descriptions, then

```
T(classAtom(UnionOf(CD1, CD2), t)) :=
T(classAtom(CD1, t)) or T(classAtom(CD2,
t))
```

  For instance, if `CD = UnionOf(Winery, Brewery)` then

```
T(classAtom(CD, t)) := t.oclIsKindOf(Winery)
or t.oclIsKindOf(Brewery)
```

- If `CD = IntersectionOf(CD1, CD2)` where `CD1` and `CD2` are OWL class descriptions, then

```
T(classAtom(CD, t)) := T(classAtom(CD1, t))
and T(classAtom(CD2, t))
```

- If `CD = ComplementOf(CD1)` where `CD1` is an OWL class descriptions, then

```
T(classAtom(CD, t)) := not T(classAtom(CD1,
t))
```

- If `CD = Restriction(oProp, allValuesFrom(CD1))` where `oProp` is a property and `CD1` is a class description, then

```
T(classAtom(CD, t)) :=
t.oProp->forAll(x|T(classAtom(CD1, x)))
```

  The variable `x` is unified with instances of the property collection `oProp` . For instance, if

```
CD = Restriction(hasMaker,
                 allValuesFrom(Winery))
```

  then

```
T(classAtom(CD, t)) :=
 t.hasMaker->forAll(x|
            T(classAtom(Winery, x))) :=
 t.hasMaker->forAll(x|
            x.oclIsKindOf(Winery))
```

- If `CD` is

```
Restriction(oProp,
            someValuesFrom(CD1))
```

  where `oProp` is a property and `CD1` is a class description, then

```
T(classAtom(CD, t)) :=
  t.oProp->exists(x|
            T(classAtom(CD1, x)))
```

- If `CD` is

```
Restriction(dProp, value(d))
```

  where `dProp` is a datatype property and `d` is data value, then

```
T(classAtom(CD, t)) := t.dProp=d
```

- If `CD` is

```
Restriction(objProp, mincardinality(n))
```

  where `objProp` is a property and `n` is minimal cardinality, then

```
T(classAtom(CD, t)) := t.objProp->size()>=n
```

  The mapping of restrictions with min- and max- cardinalities is similar.

- If `CD = OneOf({obj1,...,objn})` where `obj1,...,objn` are object URIs, then

```
T(classAtom(CD, t)) := t=obj1 or ... or
t=objn
```

  An example of the class atom with `oneOf` description

```
CA := classAtom(oneOf(White, Red), t)
```

  Then

```
T(CA) := t=White or t=Red
```

  An OWL enumerated class, which is specified by the direct enumeration of its instances, can be represented in UML by means of a UML enumeration class. Some additional mapping from OWL enumerated classes into UML enumeration classes may be needed in order to produce a syntactically correct OCL expression. However, this is the problem of UML to OWL mapping, which is beyond our rules mapping problem.

In order to show how the mapping works, let us consider a rule "*If the maker of a liquid is a winery, then the liquid is either a wine or a cognac*". The rule in SWRL is:

```
R=Implies(Antec(classAtom(Liquid, t)
  classAtom(Restriction(hasMaker,
                allValuesFrom(Winery)), t))
          Conseq(classAtom(
                UnionOf(Wine, Cognac), t)))
```

The mapping of `R` into OCL:

```
T(R):= context Liquid inv:
    T(classAtom(Restriction(hasMaker,
       allValuesFrom(Winery)), t))
    implies
    T(classAtom(UnionOf(Wine, Cognac), t)))
```

Which is finally:

```
context Liquid inv:
  self.hasMaker->forAll(x|
        x.oclIsKindOf(Winery))
implies
  self.oclIsKindOf(Wine) or
  self.oclIsKindOf(Congnac)
```

3 The syntax of OCL-Lite

The syntax of OCL-Lite expressions is defined recursively so that more complex expressions are built from simple ones. The definition below is based on the result of the mapping, defined in the previous section. In the syntax definition below we refer to OCL expressions, which are based on navigation operations, as *navigation expressions*. For instance, `self.father.brothers` is a navigation expression, which is not a new concept on top of OCL, but just a denotation of the existing OCL expression, also used in the OCL specification [1]. A navigation expression may result into a collection of instances. We introduce this denotation in order to simplify the OCL-Lite syntax by eliminating various OCL operations, which are not supported in SWRL.

1. **self** is a special variable, which type is given by the invariant context;
2. For every OCL model type t there is an unlimited number of variables v_t^i, which are OCL expressions of type t;

3. If f is an OCL operation symbol with argument types $t_1, ..., t_n$ and result type t_r, $e_1, ..., e_n$ are OCL expressions and type of e_i is t_i for all $1 \leq i \leq n$ then $f(e_1, ..., e_n)$ is OCL expression of type t_r. The set of operation symbols includes:
   - some of predefined data operations: $+, -, *$;
   - attribute operations, for instance, `self.age`, `e.salary`;
   - side-effect free operations defined by a class, for instance, `b.rentalDiscount()`;
   - navigation by role names, for instance, `self.pickupBranch`.
4. If e_1 and e_2 are OCL expressions of an appropriate type (which provides an order relation, for instance, integer) then $e_1 > e_2$, $e_1 < e_2$, $e_1 >= e_2$, and $e_1 <= e_2$ are Boolean OCL expressions. Note, that symbols $>$, $<$, $>=$, and $<=$ are OCL operations in the OCL metamodel, however here we define OCL expressions with these symbols separately since logically they are predicate symbols, while expressions, defined in the previous item are different types of terms;
5. If e is an object variable, then $e.oclIsKindOf(C)$, where C is a class name, is a Boolean expression;
6. If e_1, e_2 are Boolean expressions then $e_1\ and\ e_2$, $e_1\ or\ e_2$, $not\ e_1$, $e_1\ implies\ e_2$ are Boolean expressions;
7. If e_1, e_2 are OCL expressions of the same type, then $e_1 = e_2$, $e_1 <> e_2$ are Boolean expressions.
8. If $e(x)$ is a navigation expression, where variable x is bound by the quantifier, for instance, `self.employee`, then $e(x) \rightarrow size()\ op\ n$ is Boolean expression, where *op* is either $>$, $<$, $=$, $>=$, $<=$, or $<>$. We do not define an expression of the form $e(x).size()$, where e is of any OCL collection type, for instance, resulted from some operation on collections like *includesAll()* since they are not defined in SWRL. The defined expression of the form $e(x) \rightarrow size()\ op\ n$ is obtained via the mapping from the SWRL class atom with cardinality restriction.
9. If e is a navigation expression, for instance, `self.employee`, then $e \rightarrow forAll(x|e_1(x))$ and $e \rightarrow exists(x|e_1(x))$ are Boolean expressions, where $e_1(x)$ is a Boolean OCL expression from variable x, which is bound by universal quantifier or existential quantifier.

4 Conclusion and future work

In this paper we described the syntactic mapping from SWRL to OCL and on the result of this mapping defined the syntax of OCL-Lite, a simplified subset of the OCL syntax. OCL-Lite can be used for expressing rules, which later can be mapped into the SWRL. Some practical experiments with the implementation of this mapping are available at the REWERSE I1 website[2].

The main open issue in this work is the semantic correctness of the defined mapping. Having such mapping it is important to make sure that the set of models of a SWRL rule f coincides with the set of models of the OCL invariant $t(f)$. We describe the problem formally.

Let $\mathcal{I}$ be an interpretation of SWRL rules and $\models$ is a satisfaction relation between interpretations and SWRL rules and there are the following mappings:

- $t : S \rightarrow O$, which maps SWRL rules into OCL invariants. It is a syntactic transformation, which we have defined in this paper.
- $T : S_I \rightarrow O_I$, which maps SWRL interpretations into OCL interpretations.

The task is to prove that for any SWRL interpretation $\mathcal{J}$ and SWRL rule G if

$$\mathcal{J} \models G$$

then

$$T(\mathcal{J}) \models t(G)$$

The solution to this task is currently in our agenda and results will be published soon.

[2] REWERSE I1 Rule Translators: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15

REFERENCES

[1] Object Constraint Language (OCL), v2.0. OMG Final Adopted Specification.

[2] OWL Web Ontology Language Semantic and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/2004/OWL.

[3] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/.

[4] Luis Mandel and Maria Victoria Cengarle. On the expressive power of ocl. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 854–874, London, UK, 1999. Springer-Verlag.

[5] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. On interchanging between owl/swrl and uml/ocl. In *Proceedings of the OCLApps Workshop*, pages 81–95, Genova, Italy, 2 October 2006.

[6] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. Sharing owl/swrl and uml/ocl rules. In *Proceedings of ACM/IEEE 9th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2006)*, Genova, Italy, 1-6 October 2006.

[7] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. Towards sharing rules between owl/swrl and uml/ocl. In *Electronic Communications of European Association of Software Science and Technology*, 2007.