Translating R2ML into F-Logic

A Tutorial

Adrian Giurca, Gerd Wagner
Institute of Informatics,
Brandenburg University of Technology at Cottbus
{Giurca, G.Wagner}@tu-cottbus.de

July 19, 2006

Contents

Chapter 1

Introduction

1.1 The REWERSE I1 Rule Markup Language, R2ML

The history of modeling a general rule markup language that will support not only the power of logic programming concepts, but also the widely used object oriented programming paradigm comes from 2003 (see, for example, [14]). Recently (April 11, 2006), the R2ML [11] rule markup language was launched.

The SWRL [2] rule language combines RuleML [12] with OWL [13]. However, both languages have severe limitations regarding the representation of well known concepts from software engineering like : data types, operations calls, etc.

There are at least 5 main reasons for choosing R2ML [11] for XML representation of rules:

1. General purpose rule interchange formats, such as RuleML [12] and R2ML [11], address the Platform-Independent Model, PIM[1] level. They support a PSM to PSM interchange via the PIM level. Since there will be several rule interchange formats, there is also the issue of mapping them on each other. R2ML can encode, RuleML, most of OCL and SWRL.

[1]The term platform-independent model is most frequently used in the context of the Model Driven Architecture (MDA) approach which corresponds the Object Management Group (OMG) vision of Model Driven Engineering (MDE). The Meta-Object Facility (MOF), is the OMG standard for Model Driven Engineering. The main idea is that it should be possible to use a model transformation language (MTL) to transform a Platform-Independent Model into a Platform-Specific Model (PSM).

2. In general, a rule interchange will not be loss-free. For instance, since RuleML cannot represent several linguistic distinctions made in OCL [9] and SWRL [2], an OCL to SWRL interchange is not well supported by RuleML, while R2ML allows a loss-free interchange (see [16]). R2ML provides markup for rules written in various languages like: Prolog, F-Logic [8], SQL, OCL [9], Jena [4], Jess [5], ILR [3], RuleML [12], SWRL [2].

3. R2ML distinguish between different categories of terms: object terms and data terms

4. R2ML distinguish between different categories of atoms (adopt distinctions made in UML and OWL)

This tutorial refers to the R2ML version 0.2 (see `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6` for more details).

1.2 F-Logic

F-Logic [8] is a deductive, object oriented database language which combines the declarative semantics and expressiveness of deductive database languages with the rich data modeling capabilities supported by the object oriented data model. The bases for a logic programming language which use objects comes from the early 1986, when Maier [6] presents his "logic for objects", O-Logic which was then revisited in Kifer et Wu [7] and finalized with the J. of ACM paper of Kifer et al [8].

There are many implementation of the language.

- Unnamed (U. Melbourne - M. Lawley) - early 90's; first Prolog-based implementation
- FLORID (U. Freiburg - Lausen et al.) - late 90's; the only C++ based implementation, `http://www.informatik.uni-freiburg.de/~dbis/florid/`
- FLIP (U. Freiburg - Ludaescher) - late 90's; first XSB based implementation. Inspired the $\mathcal{FLORA}$ effort
- TFL (Tech. U. Valencia - Carsi) - late 90's;
- SILRI (Karlsruhe - Decker et al.) - late 90's; Java based

- TRIPLE (Stanford - Decker et al.) - early 2000's; Java, `http://www.dfki.uni-kl.de/frodo/triple/`
- OntoBroker (Ontoprise) - 2000; commercial, `http://www.ontoprise.de`
- $\mathcal{FLORA}-2$ (Kifer et al.) - from 2000, `http://flora.sourceforge.net/`

This paper refers to the syntactic and semantic capabilities of F-Logic as implemented by Ontoprise GmbH (`http://www.ontoprise.de`), Ontobroker version V 4.3. The reader may consult also the F-Logic Tutorial of Ontoprise [1].

Chapter 2

The R2ML-to-FLogic Translator

This chapter describes the general mapping principles from R2ML (REWERSE I1 Rule Markup Language) markup language to the F-Logic language as implemented by Ontoprise GmbH (`http://www.ontoprise.de`), Ontobroker version V 4.3. The reader may consult also the F-Logic Tutorial of Ontoprise [1].

Regarding the R2ML language the reader may consult the official R2ML page at `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6`.

2.1 Using Namespaces

Since almost of names from R2ML rule bases are qualified names, they must have declared the corresponding namespaces. Moreover, because XML qualified names use colon ':' as a separator and F-Logic use '`#`' for the same purpose (colon is reserved for membership declaration), the translator will "normalize" all qualified names by substituting ':' with '`#`'.

All declared R2ML namespaces are collected and the translator creates the `<ns>` element as indicated in the F-Logic tutorial.

2.2 Mapping R2ML Derivation Rules

A R2ML derivation rule has *conditions* and a *conclusion*. In R2ML framework the conditions of a derivation rule are quantifier-free general formulas (the content of `r2ml:conditions` role element). Conclusions are restricted to quantifier-free disjunctive normal forms without NAF (the con-

tent of `r2ml:conclusion` role element). Rules have also an optional attribute r2ml:id which identifies unique a rule in a derivation rule set.

Mapping Rule 1 (DerivationRule)
Any ***r2ml:DerivationRule*** *is mapped into an F-Logic rule*
FORALL VList1 At1 AND ... AND Atn <- EXISTS VList2 Formula*, where*
VList1 *represents the list of all variables appearing in the conclusion of the rule,*
VList2 *represents the list of all variables appearing in the conditions of the rule but not in the conclusion,*
At1 AND ... AND Atn *represents the* ***r2ml:conclusion*** *enumeration of atoms using the F-Logic keyword* ***AND****, and*
Formula *represents the R2ML formula which is content of the* ***r2ml:conditions*** *element. If this element contains an enumeration, then the enumeration is translated into a conjunction using the F-Logic keyword* ***AND****.*

If r2ml:id is present the F-Logic rule is prefixed with ***RULE name*** *where* ***r2ml:id='name'****.*

Example 2.2.1 (DerivationRule)
The following derivation rule

```
<r2ml:DerivationRule r2ml:id="DR005">
 <r2ml:conditions>
  <r2ml:ReferencePropertyAtom r2ml:referenceProperty="father">
   <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="X"/>
   </r2ml:subject>
   <r2ml:object>
    <r2ml:ObjectVariable r2ml:name="Z"/>
   </r2ml:object>
  </r2ml:ReferencePropertyAtom>
  <r2ml:ReferencePropertyAtom r2ml:referenceProperty="ancestor">
   <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="Z"/>
   </r2ml:subject>
   <r2ml:object>
    <r2ml:ObjectVariable r2ml:name="Y"/>
   </r2ml:object>
  </r2ml:ReferencePropertyAtom>
  </r2ml:conditions>
  <r2ml:conclusion>
```

```
  <r2ml:ReferencePropertyAtom r2ml:referenceProperty="ancestor">
   <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="X"/>
   </r2ml:subject>
   <r2ml:object>
    <r2ml:ObjectVariable r2ml:name="Y"/>
   </r2ml:object>
  </r2ml:ReferencePropertyAtom>
 </r2ml:conclusion>
</r2ml:DerivationRule>
```

is mapped into the F-Logic rule:

```
RULE DR005
  FORALL Z,Y uncle(Z, Y) <- EXISTS X parent(X, Y)
                                      AND brother(X, Z).
```

2.3 Mapping R2ML formulas

2.3.1 Mapping r2ml:qf.Conjunction and r2ml:qf.Disjunction

Mapping Rule 2 (qf.Conjunction)
Any ***r2ml:qf.Conjunction*** *(*qf *stands from* quantifier free*) is mapped directly into a F-Logic conjunctive expression using the keyword* ***AND*** *as a separator.*

Mapping Rule 3 (qf.Disjunction)
Any ***r2ml:qf.Disjunction*** *(*qf *stands from* quantifier free*) is mapped directly into a F-Logic disjunctive expression using the keyword* ***OR*** *as a separator.*

2.3.2 Mapping Negation

As any R2ML atom has an optional attribute boolean `r2ml:isNegated` which express if the atom is or is not negated. If the attribute `r2ml:IsNegated` is missing, this is interpreted in R2ML by the default `r2ml:isNegated='false'`. If the atom is negated i.e `r2ml:isNegated='true'` it is mapped using the keyword `NOT` before the atom translation.

Mapping Rule 4 (Negation, StrongNegation, NegationAsFailure) *All R2ML negations i.e.* `r2ml:qf.Negation`, `r2ml:qf.StrongNegation`, *and* `r2ml:qf.NegationAsFailure` *collapse in the F-Logic negation denoted by the keyword* `NOT`.

If one of the R2ML negations elements apply to a negated atom (using `r2ml:isNegated='true'` *then the F-Logic translation it will be the corresponding un-negated atom.*

Example 2.3.1 (Negation)
The following R2ML Negation atom

```
<r2ml:Negation>
 <r2ml:ReferencePropertyAtom r2ml:referenceProperty="father">
  <r2ml:subject>
   <r2ml:ObjectVariable r2ml:name="X"/>
  </r2ml:subject>
  <r2ml:object>
   <r2ml:ObjectVariable r2ml:name="Z"/>
  </r2ml:object>
 </r2ml:ReferencePropertyAtom>
</r2ml:Negation>
```

maps into the F-Logic expression: `NOT X[father -> Z]`.

2.4 Mapping R2ML Atoms

2.4.1 Mapping r2ml:GenericAtom

The `r2ml:GenericAtom` is designed to cover the general first order logic concept of atom. A `r2ml:GenericAtom` has a mandatory attribute `r2ml:predicate` (`xs:Qname`) specifying the generic atom predicate name.

Mapping Rule 5 (`r2ml:GenericAtom`) *A non negated* `r2ml:GenericAtom` *i.e* `r2ml:isNegated='false'`, *is mapped into F-Logic as a standard first order logic atom i.e.* `p(t1,...,tn)` *where* `p` *is the predicate name i.e.* `r2ml:predicate value='p'` *and* `t1`,...,`tn` *are terms (the children of the* `r2ml:arguments` *role element).*

Example 2.4.1 (GenericAtom)
The following R2ML generic atom

```
<r2ml:GenericAtom r2ml:predicate="natural_number">
 <r2ml:arguments>
  <r2ml:FunctionTerm r2ml:functor="succ">
   <r2ml:arguments>
    <r2ml:Variable r2ml:name="N"/>
   </r2ml:arguments>
  </r2ml:FunctionTerm>
 </r2ml:arguments>
</r2ml:GenericAtom>
```

is mapped into F-Logic standard predicate: `natural\_number(succ(N))`.

2.4.2 Mapping r2ml:ObjectClassificationAtom

The intended meaning of `r2ml:ObjectClassificatioAtom` is to capture the instanceOf relationship between objects and classes. Any R2ML object classification atom consists into a mandatory attribute `r2ml:class` (`xs:QName`) and an object term as an argument. Since F-Logic provides the ':' operator to declare class membership, the mapping rule is straightforward:

Mapping Rule 6 (ObjectClassificationAtom)
A positive `r2ml:ObjectClassificationAtom` *is mapped into an F-Logic membership declaration i.e.* `t:C` *where* `t` *is the corresponding term (object name and/or object variable) and* `C` *is the value of* `r2ml:class` *attribute.*

Example 2.4.2 (ObjectClassificationAtom)
The object classification atom:

```
<r2ml:ObjectClassificationAtom r2ml:class="PremiumCustomer">
 <r2ml:ObjectVariable r2ml:name="customer"/>
</r2ml:ObjectClassificationAtom>
```

maps into the F-Logic membership declaration: `Customer:PremiumCustomer`

2.4.3 r2ml:DataClassificationAtom

The main purpose of the R2ML data classification atom is to classify an data terms (data variables, data literals and functional data terms) with respect to a datatype indicated by a reference. Since

F-Logic don't provides user-defined datatypes, therefore such an atom is map similarly with `r2ml:ObjectClassificationAtom`.

2.4.4 Mapping r2ml:ReferencePropertyAtom and r2ml:AttributionAtom

The intended meaning of the `r2ml:ReferencePropertyAtom` is to capture object valued properties of objects. The `r2ml:AttributionAtom` captures data valued properties of objects. Since F-Logic does not make this distinction is its concept of F-Molecule both R2ML atoms have similar mapping rules.

Mapping Rule 7 (ReferencePropertyAtom)
Any positive `r2ml:ReferencePropertyAtom` *maps into an F-Logic F-molecule i.e.* `s[p->o]` *where* `s` *is the content of the child role element* `r2ml:subject` *of the atom,* `p` *is the name of the property i.e. the* `r2ml:referenceProperty` *value and* `o` *is the content of the child role element* `r2ml:object` *of the atom.*

Example 2.4.3 (ReferencePropertyAtom)
The following r2ml:referencePropertyAtom

```
<r2ml:ReferencePropertyAtom r2ml:referenceProperty="ancestor">
 <r2ml:subject>
  <r2ml:ObjectVariable r2ml:name="X"/>
 </r2ml:subject>
 <r2ml:object>
  <r2ml:ObjectVariable r2ml:name="Y"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
```

maps into the F-Logic expression: `X[ancestor -> Y]`.

Mapping Rule 8 (AttributionAtom)
Any positive `r2ml:AttributionAtom` *mapps into an F-Logic F-molecule i.e.* `s[a->v]` *where* `s` *is the content of the child role element* `r2ml:subject` *of the atom* `a` *is the value of the attribute* `r2ml:attribute` *and* `v` *is the content of the child role element* `r2ml:value` *of the atom.*

Example 2.4.4 (AttributionAtom)
The following R2ML attribution atom

```
<r2ml:AttributionAtom r2ml:attribute="discount">
 <r2ml:subject>
  <r2ml:ObjectVariable r2ml:name="customer"/>
 </r2ml:subject>
 <r2ml:value>
```

```
  <r2ml:TypedLiteral
     r2ml:datatype="xs:decimal" r2ml:lexicalValue="7.5"/>
 </r2ml:value>
</r2ml:AttributionAtom>
```

maps into the F-Logic F-molecule: `Customer[discount -> 7.5]`.

2.4.5 Mapping r2ml:ObjectDescriptionAtom

The `r2ml:ObjectDescriptionAtom` is a convenience construct to describe a set of property-object value pairs and/or a set of attribute-value pairs which refers to the same object as 'subject'.

Mapping Rule 9 (ObjectDescriptionAtom)
A positive ***r2ml:ObjectDescriptionAtom*** *is mapped into a composite F-Logic F-molecule i.e.* ***s[p_1->v_1; ..., p_n->v_n]*** *where* ***s*** *is the content of the element* ***r2ml:subject*** *of the atom,* ***p_i*** *is the value of* ***r2ml:referenceProperty*** *or* ***r2ml:attribute*** *(F-Logic do not make distinction between data and objects) and* ***v_i*** *is the content of* ***r2ml:object*** *or* ***r2ml:value*** *elements of the atom.*

Example 2.4.5 (ObjectDescriptionAtom)
The following object description atom:

```
<r2ml:ObjectDescriptionAtom r2ml:class="Person">
<r2ml:subject>
 <r2ml:ObjectVariable r2ml:name="x"/>
</r2ml:subject>
<r2ml:DataSlot r2ml:attribute="hasAge">
 <r2ml:value>
  <r2ml:DataVariable r2ml:name="a1"/>
 </r2ml:value>
</r2ml:DataSlot>
<r2ml:ObjectSlot r2ml:referenceProperty="father">
 <r2ml:object>
  <r2ml:ObjectVariable r2ml:name="F"/>
 </r2ml:object>
</r2ml:ObjectSlot>
</r2ml:ObjectDescriptionAtom>
```

maps into the F-Logic, F-molecule: `X[hasAge -> A1; father -> F]`

2.4.6 Mapping r2ml:AssociationAtom

An R2ML association atom is constructed using an n-ary predicate as association predicate (the value of the mandatory `r2ml:associationPredicate` attribute), a collection of data terms as "data arguments" (the content of the `r2ml:dataArguments` role element) and a collection of object terms as "object arguments" (the content of the r2ml:objectArguments role element). It corresponds to a general first order atom.

Mapping Rule 10 (AssociationAtom)
The `r2ml:AssociationAtom` maps into the F-Logic first order predicate `p(t1,...,tn)` where `p` is the value of `r2ml:associationPredicate` attribute and `t1`,...,`tn` are terms appearing in `r2ml:dataArguments` and/or `r2ml:objectArguments` role elements.

Example 2.4.6 (AssociationAtom)
The association atom

```
<r2ml:AssociationAtom r2ml:associationPredicate="buy">
 <r2ml:objectArguments>
  <r2ml:ObjectVariable r2ml:name="customer"/>
  <r2ml:ObjectVariable r2ml:name="product"/>
 </r2ml:objectArguments>
</r2ml:AssociationAtom>
```

maps into the F-Logic predicate `buy(Customer, Product)`.

2.4.7 Mapping r2ml:EqualityAtom and r2ml:InequalityAtom

The `r2ml:EqualityAtom` is intended to express equality of two object terms. The `r2ml:InequalityAtom` is just a convenience construct to express the negation of the object terms equality. Since F-Logic provides the built-in predicate equals the translation is straightforward:

Mapping Rule 11 (Equality/InequalityAtom)
The `r2ml:EqualityAtom` and `r2ml:InequalityAtom` are mapped using the keyword `NOT` and the F-Logic `equal` built-in predicate in a structure like `equal(t1,t2)` or `NOT equal(t1,t2)` where `t1` and `t2` are the object terms arguments in `r2ml:EqualityAtom` or `r2ml:InequalityAtom`.

Example 2.4.7 (EqualityAtom)
The following equality atom

```
<r2ml:EqualityAtom>
  <r2ml:ObjectVariable r2ml:name="r1"/>
  <r2ml:ObjectVariable r2ml:name="r2"/>
</r2ml:EqualityAtom>
```

maps into `equal(R1,R2)`.

2.4.8 Mapping r2ml:DatatypePredicateAtom

The R2ML DatatypePredicateAtom is designed to capture user-defined built-in predicates. It refers to a user-defined datatype predicate by its mandatory `r2ml:datatypePredicate` attribute and consists of a number of data terms as data arguments (the children of `r2ml:dataArguments` role element). Since this atom refers to a user-defined datatype predicate it is difficult to give a general mapping rule. The actual XSLT transformation maps some SWRL built-in predicates into the F-Logic built-ins as they are described in the Appendix A from the Ontoprise F-Logic Tutorial.

Example 2.4.8 (DatatypePredicateAtom)
The following datatype predicate atom:

```
<r2ml:DatatypePredicateAtom
        r2ml:datatypePredicate="swrlx:greaterThan">
 <r2ml:dataArguments>
  <r2ml:DataVariable r2ml:name="a2"/>
  <r2ml:DataVariable r2ml:name="a1"/>
 </r2ml:dataArguments>
</r2ml:DatatypePredicateAtom>
```

maps into the F-logic built-in predicate: `greater(A2,A1)`.

2.5 Mapping R2ML Terms

Following RuleML, our framework defines the generic concepts of variable. However, R2ML make a clear distinction between object terms and data terms.

Typed terms are either *object terms* standing for *objects*, or *data terms* standing for *data values*. The concrete syntax of first-order non-Boolean OCL expressions can be directly mapped to our abstract concepts of *ObjectTerm* and *DataTerm*, which can be viewed as a predicate-logic-based reconstruction of the standard OCL [9] abstract syntax.

2.5.1 r2ml:Object

The `r2ml:Object` construct is used to encode individuals i.e. objects. It consists in a mandatory attribute, `r2ml:objectID` (`xs:QName`) and from an optional one `r2ml:class` which denoted the class membership of the object.

Since this attribute is a `xs:string` other name restrictions implied in the F-Logic BNF-grammar may apply.

Example 2.5.1 (Object)
The following R2ML object

```
<r2ml:ObjectName r2ml:objectID="ind"/>
```

maps into the F-Logic constant `"ind"`.

2.5.2 r2ml:TypedLiteral and r2ml:PlainLiteral

Since in F-logic there is no distinction between data literals, R2ML data literals i.e. `r2ml:TypedLiteral` and `r2ml:PlainLiteral` are mapped into F-Logic constants using the value of the `r2ml:lexicalValue` and the `r2ml:languageTag` (in the case of plain literal) attributes.

Since this attribute is a `xs:string` other restrictions implied in the F-Logic BNF-grammar may apply.

Mapping Rule 12 (Typed/PlainLiteral) *An* `r2ml:TypedLiteral` *maps into an F-Logic constant* `v` *if* `r2ml:lexicalValue='v'`.

An r2ml:PlainLiteral maps into the F-Logic constant "`v_tag`*" where* `r2ml:lexicalValue='v'` *and* `r2ml:languageTag='tag'`.

`r2ml:languageTag` *attribute models the basic unit of language specification, based on RFC 3066 (and ultimately RFC 3066bis).*

Example 2.5.2 (Typed/PlainLiteral)
The following R2ML literals

```
<r2ml:TypedLiteral
    r2ml:datatype="xs:decimal" r2ml:lexicalValue="7.5">
<r2ml:PlainLiteral
    r2ml:languageTag="en" r2ml:lexicalValue="7">
```

maps into the F-Logic constant `"7.5"` *and, correspondingly, into* `"7_en"`.

2.5.3 r2ml:Variable, r2ml:ObjectVariable and r2ml:DataVariable

All R2ML variables are mapped into F-Logic variables.

Mapping Rule 13 (Variables) *The* `r2ml:Variable`, `r2ml:ObjectVariable` *and* `r2ml:DataVariable` *are mapped into the F-Logic concept of variable using the value of the* `r2ml:name` *attribute (*`xs:NCName`*), i.e. if* `r2ml:name='variable'` *then the F-Logic variable will be* `Variable`*.*

Any variable name is transformed by capitalizing the first letter. If the original name don't start with a letter the translator will generate a correct F-Logic variable name.

Other F-Logic restrictions regarding the names of objects and variables will apply.

Example 2.5.3 (ObjectVariable)
The next R2ML object variable

```
<r2ml:ObjectVariable r2ml:name="X"/>
```

maps to the F-Logic variable `X`*.*

2.5.4 R2ML functional Terms

All R2ML functional terms are mapped into standard first order logic terms i.e. `f(t1,...,tn)` where `f` is obtained from the value of the corresponding attribute describing the functional symbol, and `t1`,...,`tn` are obtained by the corresponding term arguments content.

r2ml:FunctionTerm

The `r2ml:FunctionTerm` corresponds to the standard first order logic functional term concept.

Mapping Rule 14 (FunctionTerm)
A `r2ml:FunctionTerm` *maps into the F-Logic functional term construct* `f(t1,...,tn)` *where* `r2ml:functor='f'` *and* `t1`,...,`tn` *are terms which are children of the* `r2ml:arguments` *role element.*

Example 2.5.4 (FunctionTerm)
The following R2ML function term

```
<r2ml:FunctionTerm r2ml:functor="succ">
 <r2ml:arguments>
  <r2ml:Variable r2ml:name="N"/>
 </r2ml:arguments>
</r2ml:FunctionTerm>
```

maps into F-Logic functional term `succ(N)`.

r2ml:RoleFunctionTerm

Similar with `r2ml:FunctionTerm` but using `r2ml:referenceProperty` for obtaining the function name.

Example 2.5.5 (RoleFunctionTerm)
The following R2ML role function term

```
<r2ml:RoleFunctionTerm r2ml:referenceProperty="husband">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="m" r2ml:class="Male"/>
 </r2ml:contextArgument>
</r2ml:RoleFunctionTerm>
```

maps into F-Logic functional term `husband(M)`.

r2ml:DatatypeFunctionTerm

Similar with `r2ml:FunctionTerm` but using `r2ml:datatypeFunction` for obtaining the function name.

Example 2.5.6 (DatatypeFunctionTerm)
The following R2ML datatype function term

```
<r2ml:DatatypeFunctionTerm r2ml:datatypeFunction="ex:holidays">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="p" r2ml:class="Person"/>
 </r2ml:contextArgument>
 <r2ml:dataArguments>
  <r2ml:DataVariable r2ml:name="startDate"/>
  <r2ml:DataVariable r2ml:name="endDate"/>
 </r2ml:dataArguments>
</r2ml:DatatypeFunctionTerm>
```

maps into F-Logic functional term `holidays(P, StartDate, EndDate)`.

r2ml:ObjectOperationTerm and r2ml:DataOperationTerm

Similar with `r2ml:FunctionTerm` but using `r2ml:operation` for obtaining the function name.

Example 2.5.7 (DataOperationTerm)
The next R2ML data operation term

```
<r2ml:DataOperationTerm r2ml:operation="decr">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="r" r2ml:class="Rental"/>
 </r2ml:contextArgument>
 <r2ml:arguments>
   <r2ml:DataVariable r2ml:name="startDate"/>
  <r2ml:DataVariable r2ml:name="endDate"/>
 </r2ml:arguments>
</r2ml:DataOperationTerm>
```

maps into F-Logic functional term `decr(R, StartDate, EndDate)`.

r2ml:AttributeFunctionTerm

Similar with `r2ml:FunctionTerm` but using `r2ml:attribute` for obtaining the function name.

Example 2.5.8 (AttributeFunctionTerm)
The following R2ML attribute function term

```
<r2ml:AttributeFunctionTerm r2ml:attribute="reservationDate">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="rd" r2ml:class="Rental"/>
 </r2ml:contextArgument>
</r2ml:AttributeFunctionTerm>
```

maps into F-Logic functional term `reservationDate(Rd)`.

Bibliography

[1] Ontoprise GmbH. F-Logic Tutorial.
http://www.ontoprise.de/content/e799/e893/e938/e954/e957/F-Logic_Tutorial_eng.pdf

[2] Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B.,Dean M. (2004) *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C Member Submission 21 May 2004,
http://www.w3.org/Submission/SWRL/

[3] *The ILOG Rule Language.*
http://www.ilog.com/

[4] *Jena The Semantic Web Framework.*
http://jena.sourceforge.net/

[5] Jess, Sandia Lab.,
http://herzberg.ca.sandia.gov/jess/

[6] D. Maier. A Logic for Objects. *In: Preprints of Workshop on Foundations of Deductive Databases and Logic Programming, ed. Jack Minker, Washington DC, August 1986.*

[7] M. Kifer and J. Wu. A Logic for Object-Oriented Logic Programming (Maier's O-Logic Revisited). *In Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 1989, 379-383.*

[8] M. Kifer, G. Lausen, and J. Wu. Logical foundations of objectoriented and frame-based languages. *Journal of the ACM, 42(4):741-843, 1995.*

[9] Object Constraint Language (OCL), *v2.0,*
http://www.omg.org/docs/ptc/03-10-14.pdf

[10] Object Management Group (OMG),
http://www.omg.org

[11] R2ML - The REWERSE I1 Rule Markup Language, *http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6*

[12] The Rule Markup Initiative, *RuleML*, *http://www.ruleml.org*.

[13] *Patel-Schneider, Peter F., Horroks I. (2004)* OWL Web Ontology Language Semantic and Abstract Syntax, *http://www.w3.org/2004/OWL*

[14] *Wagner, G. (2003),* Seven Golden Rules for a Web Rule Language. *Invited contribution to the Trends & Controversies section of IEEE Intelligent Systems 18:5, Sept/Oct 2003.*

[15] *Wagner, G., Giurca, A., Lukichev, S. (2005)* R2ML: A General Approach for Marking up Rules, *Dagstuhl Seminar Proceedings 05371, In: F. Bry, F. Fages, M. Marchiori, H. Ohlbach (Eds.) Principles and Practices of Semantic Web Reasoning, 2005.*

[16] *Wagner, G., Giurca, A., Lukichev, S. (2006)* A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL, *RoW2006, Edinburgh, UK, May 22nd, 2006.*