I1-D11

Tool Improvements and Extensions 2

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Cottbus/I1-D11/D/PU/b1
Responsible editors:	Sergey Lukichev
Reviewers:	Kaarel Kaljurand, Tobias Kuhn
Contributing participants:	Cottbus
Contributing workpackages:	I1, I2
Contractual date of deliverable:	March 23, 2007
Actual submission date:	March 28, 2007

Abstract
In this report we describe a preliminary plan for Attempto to R2ML transformation as a cooperation result of working groups I1 and I2. The document introduces mapping principles and defines a case suite for the ACE to R2ML transformation tool under development. In addition, we describe an improvement to the verbalization of R2ML operations in the template-based R2ML verbalizer and give some comparison examples of Attempto and I1 OWL verbalizers.

Keyword List
Rules, R2ML, Rule Markup Language, Semantic Web, Rule Interchange, Attempto, ACE, Generating R2ML, Verbalization

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Tool Improvements and Extensions 2

Sergey Lukichev[1], **Gerd Wagner**[1], **Norbert E. Fuchs**[2]

[1] Institute of Informatics, Brandenburg University of Technology at Cottbus, Email: {G.Wagner, Lukichev}@tu-cottbus.de
[2] Department of Informatics, University of Zurich, Email: fuchs@ifi.unizh.ch

March 28, 2007

Abstract
In this report we describe a preliminary plan for Attempto to R2ML transformation as a cooperation result of working groups I1 and I2. The document introduces mapping principles and defines a case suite for the ACE to R2ML transformation tool under development. In addition, we describe an improvement to the verbalization of R2ML operations in the template-based R2ML verbalizer and give some comparison examples of Attempto and I1 OWL verbalizers.

Keyword List
Rules, R2ML, Rule Markup Language, Semantic Web, Rule Interchange, Attempto, ACE, Generating R2ML, Verbalization

Contents

Chapter 1

Mapping from Attempto Controlled English (ACE) to R2ML

The main focus of the Working Group I1 activities is towards rule modeling, visualization, verbalization and markup. I1 has provided a number of methodologies for UML-based rule modeling (URML) and rule markup (R2ML). The tool Strelka has been implemented in order to allow visual rule modeling by software engineers. Rule models can be serialized into R2ML for the purpose of further deployment into rule executional platforms (Deliverable D9: Case Study [WLF07]) or for the purpose of validation and verbalization. The main scenario, targeted towards software engineering community is to use Strelka for rule modeling, then rules are serialized into R2ML and then to any executional platform via R2ML translators (URML (Strelka)− >R2ML− >Rule platforms (JBoss Rules, Jena, etc)).

During last years R2ML, URML and supplementary tools, based on these languages, have advanced. I1 in cooperation with I2 is considering a solution for rules capturing from natural language. This solution targets the business rules community, where people prefer to express rules in natural language (English). The solution is implemented as ACE to R2ML translator which is based on the mapping described in this report. The usage scenario then is to use ACE for expressing rules and translate them into R2ML with further translation into executional platforms.

In this chapter we briefly introduce Attempto (Section 1.1) and then in Section 1.2 give a preliminary ideas of an Attempto to R2ML mapping. Section 1.4 contains a number of important mapping examples, which define a test suite for the first version transformation tool.

1.1 Attempto Controlled English

Attempto Controlled English (ACE) [FKS06] is a controlled natural language, i.e. a precisely defined, tractable subset of full English that can be automatically and unambiguously translated to Discourse Representation Structures (DRS) - a linguistic notation using a variant of the language of first–order logic. ACE seems completely natural, but is actually a formal language, concretely it is a first–order logic language with the syntax of a subset of English. Thus ACE is

human and machine understandable. While the meaning of a sentence in unrestricted natural language can vary depending on its - possibly only vaguely defined context, the meaning of an ACE sentence is completely and uniquely defined. As a formal language ACE has to be learned, which - as experience shows - takes about two days.

ACE was originally developed to specify software programs, but has since been used as a general knowledge representation language. For instance, ACE was used to specify an automated teller machine, Kemmerer's library data base, Schubert's Steamroller, data base integrity constraints, and Kowalski's subway regulations. Furthermore, ACE served as natural language interface for the model generator EP Tableaux, for a FLUX agent, and recently for MIT's Process Handbook. The current focus of application is the Semantic Web. Within the EU Network of Excellence REWERSE the Attempto group developed a bidirectional translation of ACE to and from OWL DL, respectively OWL 1.1. Furthermore, there is the system AceRules that executes rules expressed in ACE as prioritized courteous logic programs, or a programs with stable semantics optionally combining logical (strong) negation with negation as failure.

An AceWiki allows users to generate protein ontologies in ACE. In a separate project, ACE was translated into RuleML. In cooperation with the Yale University the Attempto group will use ACE for of clinical practice guidelines.

1.2 ACE to R2ML Mapping Principles

In this section we describe the main principles of the ACE to R2ML transformation, which allows writing R2ML rules in Attempto Controlled English. The mapping algorithm translates Attempto Discourse Representation Structure (DRS) into R2ML. Since R2ML uses different types of atoms and terms, the algorithm should resolve these types according to the particular structure of ACE conditions. This resolution requires more than one pass through the input XML representation of the DRS and is quite complex to be described at this moment using a pseudo code or other abstract syntax. Therefore, in this deliverable we describe the algorithm in general with a statement that it works only on the set of example, which are listed in 1.4. The first-version implementation of the transformation does not support plurals, can/must and questions. Unsupported ACE structures are skipped by the translator.

1.2.1 Attempto DRS

Attempto DRS is mapped to R2ML as follows:

1. The DRS, derived from the Attempto text is mapped to R2ML derivation rule with an empty conditions part. The content of the DRS is mapped to a conjunction of R2ML atoms as it is described below;

2. If the DRS contains implications, then implications are mapped to separate R2ML derivation rules. The first DRS of each implication is mapped to an R2ML condition part of a rule, the second DRS of each implication is mapped to an R2ML conclusion part of a rule;

1.2.2 ACE implication

Let $I(DRS_1, DRS_2)$ be an ACE implication with two subsequent DRS structures. Then we define a transformation function $M(I)$ as follows:

$$M(I) = DerivRule(cond(M(DRS_1)), concl(M(DRS_2)))$$

$DerivRule(cond(M(DRS_1)), concl(M(DRS_2)))$ means an R2ML derivation rule with conditions and a conclusion.

1.2.3 ACE predicate

Let $P(p, r, v, s, \{o\}, \{io\})$ be an ACE predicate p where r is a discourse referent, v is a verb, s is a predicate subject, o is an optional predicate object and io is an optional indirect object. A $GenAt(p, ObjVar(r, s))$ is an R2ML generic atom with the predicate p and the R2ML object variable r of the class s. A $RefPropAt(p, subj(ObjVar(s)), obj(ObjVar(o))$ is an R2ML reference property atom with the predicate p, the R2ML object variable s as a subject and an R2ML object variable o as an object. A $AssAt(p, objArgs(ObjVar(s), ObjVar(o), ObjVar(io))$ is an R2ML association atom with the predicate p and R2ML object variables s, o, io as R2ML object arguments of the association atom. A $EqAt(s, o)$ is an R2ML equality atom between two objects s and o. A $DatatPredAt("swrlb : equal", dataArgs(DataVar(s), TypedLit(o)))$ is an R2ML datatype predicate atom with "swrlb:equal" as a predicate and data terms s and o as data arguments of the datatype predicate atom. For more information about different kinds of R2ML atoms and terms see Deliverable I1-D8 ([WGL06]). We define $M(P)$ as follows:

1. If s is a referent to a DRS object condition, o and io are not specified, then

$$M(P) = GenAt(p, ObjVar(r, s))$$

2. If s is a DRS named entity, o and io are not specified, then

$$M(P) = GenAt(p, ObjName(s))$$

3. If s and o are referents to DRS object conditions and io is not specified, then

$$M(P) = RefPropAt(p, subj(ObjVar(s)), obj(ObjVar(o))$$

4. If s, o and io are referents to DRS object conditions, then

$$M(P) = AssAt(p, objArgs(ObjVar(s), ObjVar(o), ObjVar(io))$$

5. If $v =$ "be", s and o are referents to DRS object conditions, then

$$M(P) = EqAt(s, o)$$

6. If $v =$ "be", s is a referent to a DRS object condition and o is a referent to a string or integer condition, then

$$M(P) = DatatPredAt("swrlb : equal", dataArgs(DataVar(s), TypedLit(o)))$$

1.2.4 ACE of-relation

Let $R(o_1, o_2, n)$ be an ACE of-relation, where o_1 and o_2 are discourse referents of the corresponding relation objects and n is a noun. Then $M(R)$ is defined as follows:

1. If o_1 is an referent to a string or integer condition and o_2 is referent to a DRS object condition, then

$$M(R) = AssAt(n, subj(ObjVar(o_2)), dataVal(DataVar(o_1)))$$

2. If o_1 and o_2 are referents to a DRS object condition, then

$$M(R) = RefPropAt(n, subj(ObjVar(o_2)), obj(ObjVar(o_1)))$$

1.2.5 ACE property

Let $P(r, adj, obj)$ be an ACE property with discourse referent r, adj as an adjective and obj as a property object. Then $M(P)$ is defined as follows:

1. If obj is not specified, then

$$M(P) = ObjClassAt(class(adj), ObjVar(r))$$

2. Otherwise

$$M(P) = RefPropAt(adj, subj(ObjVar(r)), obj(obj))$$

1.2.6 ACE noun

Let $N(r, n)$ be an ACE noun, where r is a discourse referent and n is a noun. Then $M(N)$ is defined as follows:

$$M(N) = ObjClassAt(class(n), ObjVar(r))$$

1.2.7 Modifier-condition, derived from ACE adverbs

Modifier-condition, derived from ACE adverbs can be considered as a property of a predicate. For instance, "A customer enters a card quickly", "quickly" is a modifier and a property of the predicate "enter". R2ML does not support properties for predicates. Therefore, in R2ML, while processing a predicate, a distinct R2ML atom should be created with a predicate, compounded of an ACE predicate verb and an ACE modifier adverb. In the example above, R2ML predicate will be "enter_quickly". If $P(r, v)$ is an ACE predicate with the referent r, verb v and $Mod(r, adv_1)...Mod(r, adv_n)$ is a list of modifiers for this predicate (with the same referent r) with $adv_1...adv_n$ as adverbs, then it is mapped to a list of R2ML atoms (their types are defined as described in Section 1.2.3) with predicates $r\_adv_1...r\_adv_n$.

If a verb is modified by a prepositional phrase, for instance "John waits in the park", then corresponding R2ML predicate will be "wait_in".

1.3 Algorithm Description

In this section we describe an algorithm for mapping from ACE DRS to R2ML. If the DRS obtained from the Attempto text contains no implications then we generate an R2ML derivation rule with an empty condition part and a conclusion part is deriver from DRS conditions. The content of a single DRS is mapped as follows:

1. Mark ACE predicates and map them to corresponding R2ML atoms as described in Section 1.2.3;
2. Mark ACE condition objects, which are subjects and objects of ACE predicates and map them to corresponding R2ML atoms or terms;
3. Pass through DRS conditions once again and map unmarked ACE objects to corresponding R2ML atoms.

1.4 Mapping examples

Below is a set of examples, which show how different ACE sentences are mapped to R2ML. We present examples in XML syntax. The output R2ML is derived by applying the algorithm and mapping principles from the previous sections. These examples define a test suite for the future implementation of the ACE to R2ML translator. The first release of the translator will work correctly at least on this test suite.

1.4.1 The customer waits.

DRS-XML:

```
<DRS domain="A B">
  <predicate ref="A" type="unspecified" verb="wait" subj="B" sentid="1"/>
  <object ref="B" struct="atomic" noun="customer" type="person"
          measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
</DRS>
```

This ACE sentence is mapped to the unary predicate as expressed by R2ML Generic Atom with the predicate "wait" and an object variable "B" of the type "customer".

```
<r2ml:GenericAtom r2ml:predicateID="wait">
 <r2ml:arguments>
  <r2ml:ObjectVariable r2ml:name="B" r2ml:classID="customer"/>
 <r2ml:arguments>
</r2ml:GenericAtom>
```

1.4.2 John enters a card.

DRS-XML:

```
<DRS domain="A B C">
  <named ref="A" name="John" sentid="1"/>
```

```
  <object ref="A" struct="atomic" noun="named_entity" type="person" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="B" struct="atomic" noun="card" type="object" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <predicate ref="C" type="unspecified" verb="enter" subj="A" obj="B" sentid="1"/>
</DRS>
```

The predicate "C" ("enter") is mapped to the R2ML reference property atom with "John" as a subject and "B" as an object since its subject and object referents are of object types. The object with the "A" referent ("John") is mapped to the R2ML object classification atom with an object name "John" as an argument. The object with the "B" referent ("card") is mapped to the R2ML object classification atom with an object variable "B". This DRS structure corresponds to the following conjunction of three R2ML atoms:

```
<r2ml:ObjectClassificationAtom r2ml:classID="person">
  <r2ml:ObjectName r2ml:objectID="John"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="card">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="enter">
 <r2ml:subject>
   <r2ml:ObjectName r2ml:objectID="John"/>
 </r2ml:subject>
 <r2ml:object>
   <r2ml:ObjectVariable r2ml:name="B"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
```

1.4.3 A clerk gives a password to a customer.

DRS-XML:

```
<DRS domain="A B C D">
  <object ref="A" struct="atomic" noun="clerk" type="person" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="B" struct="atomic" noun="password" type="object" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="C" struct="atomic" noun="customer" type="person" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <predicate ref="D" type="unspecified" verb="give" subj="A" obj="B"
      indobj="C" sentid="1"/>
</DRS>
```

The ACE sentence is mapped to a ternary association. The predicate "D" is mapped to the R2ML association atom with three arguments since its terms have object types. The first three objects are mapped to the R2ML object classification atoms, since they define the object variables "A", "B", "C" of object types "clerk", "password", "customer". This DRS structure corresponds to the following conjunction of four R2ML atoms:

```
<r2ml:ObjectClassificationAtom r2ml:classID="clerk">
   <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="password">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="customer">
  <r2ml:ObjectVariable r2ml:name="C"/>
</r2ml:ObjectClassificationAtom>
<r2ml:AssociationAtom r2ml:referencePropertyID="enter">
 <r2ml:objectArguments>
   <r2ml:ObjectVariable r2ml:name="A"/>
   <r2ml:ObjectVariable r2ml:name="B"/>
   <r2ml:ObjectVariable r2ml:name="C"/>
 <r2ml:objectArguments>
</r2ml:AssociationAtom>
```

1.4.4 A card is valid.

DRS-XML:

```
<DRS domain="A B C">
  <object ref="A" struct="atomic" noun="card" type="object" measure="cardinality"
          unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <property ref="B" adj="valid" sentid="1"/>
  <predicate ref="C" type="state" verb="be" subj="A" obj="B" sentid="1"/>
</DRS>
```

The predicate "C" is mapped to the R2ML equality atom with "A" and "B" since these referents are interpreted as of object type. The DRS object "A" is mapped to the R2ML object classification atom (unary predicate). The DRS property "B" is mapped to the R2ML object classification atom with class "valid" and object variable "B". This DRS structure corresponds to the following conjunction of three R2ML atoms:

```
<r2ml:ObjectClassificationAtom r2ml:classID="card">
   <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="valid">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:EqualityAtom>
   <r2ml:ObjectVariable r2ml:name="A"/>
   <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:EqualityAtom>
```

The mapping above is one to one, i.e. every DRS referent is mapped to an R2ML atom. Such mapping, therefore, is inversive. If the inverse transformation is not under consideration, then DRS to R2ML can be modified in order to remove transitive closure from the output R2ML by defining new predicate "is_valid" and having only two object classification atoms (card(A) and is_valid(A)):

```
<r2ml:ObjectClassificationAtom r2ml:classID="card">
  <r2ml:ObjectName r2ml:objectID="A"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="is_valid">
  <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
```

The issue of transformation reversibility should be additionally investigated. The general discussion here is since R2ML is a first-order language, every R2ML rule can be mapped to ACE. When transforming to R2ML from ACE, linguistic meaning of some ACE constructs, like it is mentioned in this section, can be dropped. For instance, adverbs as a property of a predicate becomes a modified predicate in R2ML (See 1.2.7). Therefore, the result ACE sentence, obtained from the R2ML rule will not have adverbs, but only modified predicates.

1.4.5 A card is valid and correct.

DRS-XML:

```
<DRS domain="A B C">
  <object ref="A" struct="atomic" noun="card" type="object" measure="cardinality"
          unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <property ref="B" adj="valid" sentid="1"/>
  <property ref="B" adj="correct" sentid="1"/>
  <predicate ref="C" type="state" verb="be" subj="A" obj="B" sentid="1"/>
</DRS>
```

Similar to the 1.4.4, this DRS structure corresponds to the three R2ML object classification atoms (card(A), valid(B), correct(B)) and one equality atom (is(A, B)):

```
<r2ml:ObjectClassificationAtom r2ml:classID="card">
   <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="valid">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="correct">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:EqualityAtom>
   <r2ml:ObjectVariable r2ml:name="A"/>
   <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:EqualityAtom>
```

Similar to the 1.4.4, we may remove R2ML equality atom by introducing two new predicates "is_valid" and "is_correct". Then the R2ML for this sentence is as follows:

```
<r2ml:ObjectClassificationAtom r2ml:classID="card">
   <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
```

```
<r2ml:ObjectClassificationAtom r2ml:classID="is_valid">
  <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="is_correct">
  <r2ml:ObjectVariable r2ml:name="A"/>
</r2ml:ObjectClassificationAtom>
```

1.4.6 John is a rich customer.

DRS-XML:

```
<DRS domain="A B C">
  <named ref="A" name="John" sentid="1"/>
  <object ref="A" struct="atomic" noun="named_entity" type="person"
          measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="B" struct="atomic" noun="customer" type="person" measure="cardinality"
          unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <property ref="B" adj="rich" sentid="1"/>
  <predicate ref="C" type="state" verb="be" subj="A" obj="B" sentid="1"/>
</DRS>
```

The discourse referent "A" is mapped to the R2ML object classification atom with object name "John". The type of the object name (individual) "John" is "named_entity". Discourse referent "B" with its property "rich" is mapped to two R2ML object classification atoms (customer(B), rich(B)).

```
<r2ml:ObjectClassificationAtom r2ml:classID="named_entity">
  <r2ml:ObjectName r2ml:objectID="John"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="customer">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
 <r2ml:ObjectClassificationAtom r2ml:classID="rich">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:EqualityAtom>
   <r2ml:ObjectName r2ml:objectID="John"/>
   <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:EqualityAtom>
```

1.4.7 A customer is richer than John.

DRS-XML:

```
<DRS domain="A B C D">
  <named ref="A" name="John" sentid="1"/>
  <object ref="A" struct="atomic" noun="named_entity" type="person"
          measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="B" struct="atomic" noun="customer" type="person"
```

```
          measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <property ref="C" adj="richer_than" obj="A" sentid="1"/>
  <predicate ref="D" type="state" verb="be" subj="B" obj="C" sentid="1"/>
</DRS>
```

The discourse referent "A" ("John") is mapped to the R2ML object classification atom. The property referent "C" is mapped to the R2ML reference property atom (richer_than("John")) since it has an object referent "A" (the formula from 1.2.5 is applied). The predicate referent "D" is mapped to the R2ML equality atom (is(B, C)). R2ML:

```
<r2ml:ObjectClassificationAtom r2ml:classID="named_entity">
  <r2ml:ObjectName r2ml:objectID="John"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="customer">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="richer_than">
 <r2ml:subject>
   <r2ml:ObjectVariable r2ml:name="C"/>
 </r2ml:subject>
 <r2ml:object>
   <r2ml:ObjectName r2ml:objectID="John"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
<r2ml:EqualityAtom>
  <r2ml:ObjectVariable r2ml:name="B"/>
  <r2ml:ObjectVariable r2ml:name="C"/>
</r2ml:EqualityAtom>
```

1.4.8 A customer enters a card quickly.

DRS-XML:

```
<DRS domain="A B C">
  <object ref="A" struct="atomic" noun="customer" type="person"
          measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <object ref="B" struct="atomic" noun="card" type="object" measure="cardinality"
          unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <predicate ref="C" type="unspecified" verb="enter" subj="A" obj="B"
             sentid="1"/>
  <modifier ref="C" type="manner" adverb="quickly" sentid="1"/>
</DRS>
```

The predicate discourse referent "C" is mapped to the R2ML reference property atom. The modifier "quickly" is mapped as described in 1.2.7. R2ML:

```
<r2ml:ObjectClassificationAtom r2ml:classID="customer">
  <r2ml:ObjectVariable r2ml:classID="A"/>
</r2ml:ObjectClassificationAtom>
```

```
<r2ml:ObjectClassificationAtom r2ml:classID="card">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="enters_quickly">
 <r2ml:subject>
   <r2ml:ObjectVariable r2ml:name="A"/>
 </r2ml:subject>
 <r2ml:object>
   <r2ml:ObjectVariable r2ml:name="B"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
```

1.4.9 The rich and old customer waits.

DRS-XML:

```
<DRS domain="A B">
  <predicate ref="A" type="unspecified" verb="wait" subj="B" sentid="1"/>
  <property ref="B" adj="old" sentid="1"/>
  <property ref="B" adj="rich" sentid="1"/>
  <object ref="B" struct="atomic" noun="customer" type="person"
      measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
</DRS>
```

The property and object referent "B" is mapped to three R2ML object classification atoms. The predicate discourse referent "A" (waits) is mapped to the R2ML generic atom. R2ML:

```
<r2ml:ObjectClassificationAtom r2ml:classID="customer">
  <r2ml:ObjectVariable r2ml:classID="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="rich">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:ObjectClassificationAtom r2ml:classID="old">
  <r2ml:ObjectVariable r2ml:name="B"/>
</r2ml:ObjectClassificationAtom>
<r2ml:GenericAtom r2ml:predicateID="waits">
 <r2ml:arguments>
   <r2ml:ObjectVariable r2ml:name="B"/>
 </r2ml:arguments>
</r2ml:GenericAtom>
```

1.4.10 If there is a code then John enters it. (John enters every code.)

DRS-XML:

```
<DRS domain="A">
  <named ref="A" name="John" sentid="1"/>
  <object ref="A" struct="atomic" noun="named_entity" type="person"
```

```
      measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <Implication>
    <DRS domain="B">
      <object ref="B" struct="atomic" noun="code" type="object" measure="cardinality"
      unit="count_unit" numrel="eq" num="1" sentid="1"/>
    </DRS>
    <DRS domain="C">
      <predicate ref="C" type="unspecified" verb="enter" subj="A" obj="B" sentid="1"/>
    </DRS>
  </Implication>
</DRS>
```

Since this DRS contains an implication, it is mapped to the R2ML derivation rule with conditions and a conclusion. The conditions part contains the object classification atom (code(B)). The conclusion part contains the reference property atom (enter("John", B)). The variable "B" is implicitly universally quantified. If there are more conditions in the DRS domain, which corresponds to the R2ML conclusion part, they are mapped to conjunction or disjunction of R2ML atoms. R2ML:

```
<r2ml:conditions>
 <r2ml:ObjectClassificationAtom r2ml:classID="code">
  <r2ml:ObjectName r2ml:name="B"/>
 </r2ml:ObjectClassificationAtom>
</r2ml:conditions>
<r2ml:conclusion>
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="enter">
 <r2ml:subject>
   <r2ml:ObjectName r2ml:objectID="John"/>
 </r2ml:subject>
 <r2ml:object>
   <r2ml:ObjectVariable r2ml:name="B"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
</r2ml:conclusion>
```

1.4.11 John enters no code.

DRS-XML:

```
<DRS domain="A">
  <named ref="A" name="John" sentid="1"/>
  <object ref="A" struct="atomic" noun="named_entity" type="person"
      measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
  <Implication>
    <DRS domain="B">
      <object ref="B" struct="atomic" noun="code" type="object"
        measure="cardinality" unit="count_unit" numrel="eq" num="1" sentid="1"/>
    </DRS>
```

```
    <DRS domain="">
      <Negation>
    <DRS domain="C">
      <predicate ref="C" type="unspecified" verb="enter" subj="A" obj="B" sentid="1"/>
    </DRS>
      </Negation>
    </DRS>
  </Implication>
</DRS>
```

The mapping is similar to 1.4.10, except we have attribute "isNegated" in order to negate the reference property atom in the conclusion part. R2ML:

```
<r2ml:conditions>
 <r2ml:ObjectClassificationAtom r2ml:classID="named_entity">
  <r2ml:ObjectName r2ml:name="John"/>
 </r2ml:ObjectClassificationAtom>
 <r2ml:ObjectClassificationAtom r2ml:classID="code">
  <r2ml:ObjectName r2ml:name="B"/>
 </r2ml:ObjectClassificationAtom>
</r2ml:conditions>
<r2ml:conclusion>
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="enter"
                            r2ml:isNegated="true">
 <r2ml:subject>
   <r2ml:ObjectName r2ml:objectID="John"/>
 </r2ml:subject>
 <r2ml:object>
   <r2ml:ObjectVariable r2ml:name="B"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
</r2ml:conclusion>
```

Chapter 2

Improvements to the R2ML verbalizer

The ability to represent rules in the form of a Controlled English text is an important issue because it makes Semantic Web content more accessible and may help domain experts to understand and validate formally represented rules without being familiar with the R2ML syntax.

2.1 Verbalizing operations

In this chapter we solve an issue, which was left open in the previous I1 deliverable on verbalization [LW06]. Let's consider the following operation in the action part of the production rule:

> A sender send a message to a receiver

It may correspond to the following R2ML code:

```
1  <r2ml:InvokeActionExpression r2ml:operationID="sendMessage">
2   <r2ml:arguments>
3    <r2ml:ObjectVariable r2ml:name="sender" r2ml:classID="Person"/>
4    <r2ml:ObjectVariable r2ml:name="receiver" r2ml:classID="Person"/>
5   </r2ml:arguments>
6  </r2ml:InvokeActionExpression>
```

The verbalizer output for this action expression is:

> invoke sendMessage with Person sender as argument 1, Person receiver as argument 2

This output is acceptable in order to understand how operation "sendMessage" is defined and with which parameters, but it is quite technical and can be further improved as:

> Person sender sends message to Person receiver

In order to achieve such an output we introduce a schema for annotation properties in the R2ML vocabulary. R2ML supports internal vocabularies, which may optionally be defined for a rule base. Such vocabularies define classes, properties and class operations. An excerpt from R2ML vocabulary for such an operation is as follows:

```
1  <r2mlv:Operation r2mlv:ID="sendMessage">
2   <r2mlv:argumentsType>
3    <r2mlv:Class r2mlv:ID="Person" r2mlv:np="sender"/>
4    <r2mlv:Class r2mlv:ID="Person" r2mlv:np="receiver"/>
5   </r2mlv:argumentsType>
6   <r2mlv:ap>
7    <ver:argument ver:position="1"/>
8    <ver:name>send message to</ver:name>
9    <ver:argument ver:position="2"/>
10  </r2mlv:ap>
11 </r2mlv:Operation>
```

Lines 6-10 contain an annotation property, which defines an English sentence structure for this operation. Line 7 defines a place for an argument 1 from the operation signature. The element <ver:argument ver:position="1"/> is replaced by the vrebalization of the object variable in line 3 from the rule action above. Then comes the text from the element in line 8: `<ver:name>send message to</ver:name>` and concludes the verbalization of the object variable in line 4 from the rule definition above. So, such annotation properties in XML define the structure of the output sentences for any user-defined operation, function or predicate, used in the rule atoms.

These annotations can be created by the user in two ways:

1. Added directly to the R2ML XML file.
2. Generated by an R2ML translator from some rule language. For instance, when translating from JBoss Rules, such annotations can be generated from Java Beans annotations, defined by the beans developer.

2.2 Further Work

The state of the art of I1 rule methodologies is that it is possible to develop real working rule applications. This is proven in the I1 deliverable D9: Case Study. This case study gives a lot of test data for the verbalization work. In our plan is to continue verbalizer evaluation on this data and to make further improvements. One of such improvements is described in this chapter. Since Strelka has been improved to support reaction rules and since R2ML has an advanced events/action model the verbalization of reaction rules can be improved as well.

Chapter 3

Sample outputs of Attempto OWL verbalizer and I1 OWL verbalizer

In this chapter we give examples of the output of two OWL verbalizers developed by Working Groups I1 and I2. The I2 OWL verbalizer translates OWL axioms into ACE sentences. The I1 verbalizer is based on templates, defined in XML for each OWL axiom. The I1 verbalizer uses two types of templates: for OWL ontologies with defined verbalization annotation properties and without. When OWL ontology is annotated (See I1 deliverable [WGL06]) the output is more natural. When the ontology is not annotated, the output sounds as technical English, but is still acceptable. We consider several examples of OWL axioms and give three types of output: Attempto OWL verbalizer, I1 OWL verbalizer on annotated ontology and I1 OWL verbalizer on non-annotated ontology.

3.1 Simple subClassOf example

Let's consider the following OWL axiom with a subClassOf, which states that every Lateharvest is a Wine:

```
<owl:Class rdf:ID="LateHarvest">
  <rdfs:subClassOf rdf:resource="#Wine" />
</owl:Class>
```

The output of Attempto verbalizer for this class is as follows:

> Every LateHarvest is a Wine.

The output of I1 verbalizer on non-annotated ontology is as follows:

> A LateHarvest is a Wine.

We may annotate the class as follows:

```
<owl:Class rdf:ID="LateHarvest">
  <i1:vterm>late harvest</i1:vterm>
  <rdfs:subClassOf rdf:resource="#Wine" />
</owl:Class>
```

If we also assume, that class Wine is annotated in a similar way, then output of I1 verbalizer is as follows:

> A late harvest is a wine.

The use of "Every" in Attempto verbalizer denotes a universal quantification. The I1 verbalizer uses "A" for the same purpose. The Controlled English vocabulary, used by the I1 verbalizer is defined in [WGL06], Chapter 2, Section 2.3.

3.2 intersectionOf example

Let's consider the following OWL class axiom with intersectionOf, which is used to define the class ItalianWine (Italian wine is a wine and it is located in the Italian region):

```
<owl:Class rdf:ID="ItalianWine">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Wine" />
  <owl:Restriction>
   <owl:onProperty rdf:resource="#locatedIn" />
   <owl:hasValue rdf:resource="#ItalianRegion" />
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

The output of Attempto verbalizer for this class is as follows:

> Every ItalianWine is a Wine and locatedIns ItalianRegion.

The output of I1 verbalizer on non-annotated ontology is as follows:

> A ItalianWine is a Wine and has ItalianRegion by property locatedIn.

The output of I1 verbalizer on annotated ontology is as follows:

> A Italian wine is a wine and is located in Italian region.

3.3 allValuesFrom example

Let's consider the following OWL class axiom with allValuesFrom restriction, which defines a restriction on property hasBody of the class CabernetSauvignon:

```
<owl:Class rdf:ID="CabernetSauvignon">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasBody" />
```

```
   <owl:allValuesFrom>
    <owl:Class>
     <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:about="#Medium" />
      <owl:Thing rdf:about="#Full" />
     </owl:oneOf>
    </owl:Class>
   </owl:allValuesFrom>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The output of Attempto verbalizer for this class is as follows:

> No CabernetSauvignon hasBodys something that is not something that is Medium or that is Full.

The output of I1 verbalizer on non-annotated ontology is as follows:

> A CabernetSauvignon has something, which is either Medium or Full by property hasBody.

The output of I1 verbalizer on annotated ontology is as follows:

> A body of a cabernet sauvingon is something, which is either Medium or Full.

3.4 someValuesFrom example

Let's consider the following OWL class axiom with someValuesFrom restriction, which defines a restriction on property locatedIn of the class Wine:

```
<owl:Class rdf:ID="Wine">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#locatedIn"/>
   <owl:someValuesFrom rdf:resource="&vin;Region"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The output of Attempto verbalizer for this class is as follows:

> Every Wine locatedIns a Region.

The output of I1 verbalizer on non-annotated ontology is as follows:

> A Wine has at least one Region by property locatedIn.

The output of I1 verbalizer on annotated ontology is as follows:

> A wine has at least one region as a location region.

3.5 Verbalizing Class Descriptions

Let's verbalize a but more complex OWL axiom, which defines a WhiteNonSweetWine, using intersectionOf class description and allValuesFrom restriction:

```
<owl:Class rdf:ID="WhiteNonSweetWine">
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#WhiteWine" />
   <owl:Restriction>
    <owl:onProperty rdf:resource="#hasSugar" />
     <owl:allValuesFrom>
      <owl:Class>
       <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Dry" />
        <owl:Thing rdf:about="#OffDry" />
      </owl:oneOf>
     </owl:Class>
   </owl:allValuesFrom>
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

The output of Attempto verbalizer for this class is as follows:

> Every WhiteNonSweetWine is a WhiteWine and does not hasSugar something that is not something that is Dry or that is OffDry.

The output of I1 verbalizer on non-annotated ontology is as follows:

> A WhiteNonSweetWine is a WhiteWine and a WhiteNonSweetWine has Dry or OffDry by property hasSugar.

The output of I1 verbalizer on annotated ontology is as follows:

> A white non sweet wine is a white wine and and sugar content of white non sweet wine is Dry or OffDry.

Bibliography

[FKS06] N. E. Fuchs, K. Kaljurand, and G. Schneider. Attempto controlled english meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In *Proc. of FLAIRS'2006*, 2006.

[LW06] S. Lukichev and G. Wagner. Verbalization of the rewerse i1 rule markup language. Technical report, REWERSE IST 506779, September 2006.

[WGL06] Gerd Wagner, Adrian Giurca, and Sergey Lukichev. Language improvements and extensions. Deliverable i1-d8, REWERSE IST 506779, March 2006.

[WLF07] Gerd Wagner, Sergey Lukichev, and Norbert Fuchs. Case study 1. Deliverable i1-d9, REWERSE IST 506779, March 2007.