Rule Interchange in the Semantic Web

Z. M. MA+ AND XING WANG
College of Information Science and Engineering
Northeastern University
Shenyang, 110819 P.R. China

Rule interchange has become one of the most important issues in the Semantic Web. As a recommendation of the W3C (WWW Consortium), RIF (Rule Interchange Format) is dedicated to rule interchange between different rule languages. To be a general rule interchange format, RIF should first support rule interchange with SWRL (Semantic Web Rule Language), RuleML (Rule Markup Language), R2ML (REWERSE Rule Markup Language) and F-logic (Frame Logic), before completely implementing rule interchange with other rule languages. In the paper, we propose and construct RIAXML (Rule Interchange Architecture based on XML Syntaxes) – a rule interchange structure centered on RIF, which enables bidirectional rule interchange between RIF and SWRL, RuleML, R2ML and F-logic. Furthermore, we explore the issue of information loss and present corresponding remedial measures. More importantly, we design and implement a prototype system of RIAXML, and give its experimental analysis.

Keywords: rule interchange, semantic web, RIF, rule interchange architecture, RIAXML, information loss, RIAXML 1.0

1. INTRODUCTION

The Semantic Web [1, 5, 6], as a great vision for the future Web architecture, has gained considerable attention. One of the declared goals of the Semantic Web is to reason knowledge which is pervaded on the Web. How to represent knowledge and how to reason with it has been firmly put on the agenda.

As an important representation of knowledge, rules play an important role in the Semantic Web. They possess many features including high expressiveness, terseness and requiring small memory space, and they can enhance the expressive power of the Semantic Web ontologies [9], which let rules become indispensable and widely used in the Semantic Web.

With the development of rules, several kinds of rule languages including SWRL (Semantic Web Rule Language) [8] come into being. They are used to construct different rule systems applied in Web and Semantic Web. It is unavoidable that these systems communicate with each other (i.e., interchange rules). However, because of the heterogeneity of different rule languages in the respect of syntaxes and semantics, creating a generally ac-
cepted interchange format is by no means a trivial task [3]. RuleML (Rule Markup Language) [4] proposed by the RuleML Initiative tries to encompass different kinds of rules. R2ML (REWERSE Rule Markup Language) [19] which integrates OCL (Object Constraint Language), SWRL and RuleML has strong markup ability. Especially, RIF (Rule Interchange Format) [16], a recommendation of the W3C (WWW Consortium), aims at implementing rule interchange between different rule languages and thus between different rule systems in the Semantic Web.

Milanović proposes a transformation between SWRL and R2ML in [14], and offers a transformation approach to sharing rules between SWRL and OCL in [13]. Wagner implements a translator [17], which can perform transformations between R2ML and some others languages. We note that all the above transformations are based on R2ML.

Rule interchange centered on RIF is the mainstream direction of rule interchange. However, there is no published literature which refers to a rule interchange structure or an interchange system centered on RIF. Before completely fulfilling rule interchange with any other rule language, RIF should first support rule interchange with SWRL, R2ML, RuleML and F-logic (Frame Logic, an important language containing rules) [11]. Based on the facts above, in the paper, we propose and construct RIAXML (Rule Interchange Architecture based on XML syntax), a rule interchange structure centered on RIF, which enables rule interchange between RIF and SWRL, RuleML, R2ML and F-logic. The transformations are all bidirectional and based on XML syntaxes. Moreover, because of different expressive power of the five languages, information loss in the process of rule interchange is unavoidable. We explore the issue of information loss, and present corresponding remedial measures to reduce the loss. More importantly, based on a programming language Java, we implement RIAXML 1.0 (version 1.0), a prototype system of RIAXML, which supports rule interchange between RIF and SWRL, RuleML, R2ML and F-logic. Transformations among SWRL, RuleML, R2ML, and F-logic, and transformations between RIF and rule languages based on SWRL, RuleML, R2ML and F-logic are also supported by RIAXML 1.0. To let computers “understand” RIF documents, in RIAXML 1.0, the transformations between RIF presentation syntax and RIF XML syntax are also implemented.

One of the advantages of RIAXML centered on RIF is that it can markedly reduce the number of mappings, and can lead to a remarkably high efficiency of transformations, especially when the number of rule languages is large, which may correspond to the transformations between languages based on SWRL, RuleML, R2ML and F-logic. Considering the number of available rule languages, it would be a lot of effort to develop one-to-one transformations between the rule languages. For n rule languages, n × (n − 1) transformations should be constructed. On the contrary, using RIF as an intermediary language would considerably reduce the number of transformations. For n rule languages, the number of transformations will reduce to 2 × n (n transformations to RIF, and then n transformations from RIF back to all the other rule languages). RIAXML 1.0 can be employed to interchange rules between different intelligent agents in Web Services, and it has good scalability, which let RIAXML 1.0 have an extensive prospect in application.

The remainder of the paper is organized as follows. Section 2 constructs RIAXML, explores the issue of information loss, and presents corresponding remedial measures. Section 3 designs and implements RIAXML 1.0. Section 4 concludes this paper and gives future work.
2. RULE INTERCHANGE ARCHITECTURE RIA\_XML

Based on the mappings of atoms and axioms between RIF and SWRL, RuleML, R2ML and F-logic, we will construct rule interchange architecture RIA\_XML in this section. As shown in Fig. 1, RIA\_XML is centered on RIF, and it supports rule interchange between RIF and SWRL, RuleML, R2ML and F-logic. Pairwise transformations among the five languages can be supported. RIA\_XML also supports rule interchange between RIF and languages which are based on SWRL, RuleML, R2ML and F-logic. Among others, all the transformations are bidirectional. Because of the extensibility of RIF dialects, our architecture is also extensible. With the enhancement of expressiveness of RIF, RIA\_XML will support more transformations between RIF and other rule languages.

![Fig. 1. Rule interchange architecture RIA\_XML.](image)

In the following, we first introduce RIF dialects. And then we construct four pairs of transformations, i.e., between RIF and SWRL, between RIF and RuleML, between RIF and R2ML, and between RIF and F-logic.

2.1 RIF

RIF is dedicated to developing a Web standard for exchanging rules between different rule languages and thus between different rule systems. Despite its humble name, RIF is not just a format and is not primarily about syntax [10]. It is an extensible framework for rule-based languages, called RIF dialects, which include precise and formal specifications of the syntax, semantics and XML serialization. To be a W3C’s standard of rule interchange, the RIF Working Group has published six RIF recommendations including RIF-BLD (RIF Basic Logic Dialect) [2].

RIF-BLD is the first dialect of RIF dialects. It corresponds to the languages of definite Horn rules with equality and standard first-order semantics. RIF-BLD has its presentation syntax and semantics, and it also has a normative XML schema extension. Although RIF-BLD has limited expressiveness, it inherits the feature of extensibility of RIF, so we can extend it according to practical needs, which is also an important reason that we select RIF-BLD as our central language to construct the rule interchange architecture. It should be noted that, RIF mentioned in the rest of this paper is RIF-BLD, unless stated otherwise.

2.2 Transformations between RIF and SWRL, RuleML, R2ML and F-logic

In this section, we discuss the transformations between RIF and SWRL, RuleML, R2ML and F-logic. Mappings of main elements between languages are given.
2.2.1 The transformations between RIF and SWRL

In order to share rules between SWRL and RIF, we define mappings between constructors and axioms of SWRL and RIF on the level of their abstract syntaxes. Every SWRL rule is mapped to a RIF universal rule, and there will be a rule implication in it. In Table 1, we show the mappings between SWRL and RIF atoms and axioms in detail. Here classID and DatatypeID are identifiers of classes and data ranges; \( t \) is a variable name; CD, CD\(_1\) and CD\(_2\) represent class descriptions; PD\(_1\) and PD\(_2\) denote property IDs; Unionof and Intersectionof stand for disjunction and conjunction; ClassAtom(classID, \( t \)) stands for class atoms which consist of a description and a variable; ClassAtom(Unionof(CD\(_1\), CD\(_2\), \( t \))\) stands for a disjunction of two class atoms; ClassAtom(Intersectionof(CD\(_1\), CD\(_2\), \( t \))\) stands for a conjunction of two class atoms; DatarangeAtom(DatatypeID, \( t \)) stands for datarange atoms which consist of a data range and a variable; IndividualvaluedPropertyAtom(objectID\(_1\), objectID\(_2\)) stands for properties which consist of a property name and two elements that can be individuals and variables, DatavaluedPropertyAtom(objectID\(_1\), objectID\(_2\)) stands for properties which consist of a property name and two elements objectID\(_1\) and objectID\(_2\), where objectID\(_1\) can be individuals or variables and objectID\(_2\) stands for data values; SubClassof(CD\(_1\), CD\(_2\)) means \( CD_1 \subseteq CD_2 \); EquivalentClasses(CD\(_1\), CD\(_2\)) means \( i.e. \), Conjunction(SubClassof(CD\(_1\), CD\(_2\)), SubClassof(CD\(_2\), CD\(_1\)))

<table>
<thead>
<tr>
<th>SWRL Expression</th>
<th>RIF Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassAtom(classID, ( t ))</td>
<td>Formula1(classID, ( t )) (<em>abstract class</em>)</td>
</tr>
<tr>
<td>ClassAtom(Unionof(CD(_1), CD(_2), ( t )))</td>
<td>Or(( 2\mu \text{IF}(\text{ClassAtom(CD}_1, ( t )), \text{ClassAtom(CD}_2, ( t ))))</td>
</tr>
<tr>
<td>ClassAtom(Intersectionof(CD(_1), CD(_2), ( t )))</td>
<td>And(( 2\mu \text{IF}(\text{ClassAtom(CD}_1, ( t )), \text{ClassAtom(CD}_2, ( t )))))</td>
</tr>
<tr>
<td>ClassAtom(OneOf({objID(_1), …, objID(_n)}, ( t )))</td>
<td>Or(Equality(objID(_1), ( t )), …, Equality(objID(_n), ( t )))</td>
</tr>
<tr>
<td>ClassAtom(ObjectRestriction(objPropID, allValuesFrom(CD)), ( t ))</td>
<td>UniversalRule(x, Implication(( 2\mu \text{IF}(\text{ObjectRestriction(objPropID, t, x),}) ( 2\mu \text{IF}(\text{ClassAtom(CD, t)))))</td>
</tr>
<tr>
<td>ClassAtom(ObjectRestriction(objPropID, someValuesFrom(CD)), ( t ))</td>
<td>And(( 2\mu \text{IF}(\text{ObjectRestriction(objPropID, t, x),}) ( 2\mu \text{IF}(\text{ClassAtom(CD, t)))))</td>
</tr>
<tr>
<td>DatarangeAtom(DatatypeID, ( t ))</td>
<td>Formula1(ID, ( t )) (<em>concrete class</em>)</td>
</tr>
<tr>
<td>DatarangeAtom(DataRestriction(dataPropID, allValuesFrom(DatatypeID)), ( t ))</td>
<td>UniversalRule(x, Implication(( 2\mu \text{IF}(\text{DataRestriction(dataPropID, t, x),}) ( 2\mu \text{IF}(\text{DatarangeAtom(datatypeID, t)))))</td>
</tr>
<tr>
<td>IndividualvaluedPropertyAtom(objectID(_1), objectID(_1))</td>
<td>Formula1(( 2\mu \text{IF}(\text{IndividualvaluedPropertyID, objectID}_1,) objectID(_2))) (<em>abstract property</em>)</td>
</tr>
<tr>
<td>DatavaluedPropertyAtom(objectID(_1), objectID(_2))</td>
<td>Formula1(( 2\mu \text{IF}(\text{DatavaluedPropertyID, objectID}_1,) datavalue)) (<em>concrete property</em>)</td>
</tr>
<tr>
<td>SubClassof(CD(_1), CD(_2))</td>
<td>Subclass(( 2\mu \text{IF}(\text{CD}_1, \text{CD}_2)))</td>
</tr>
<tr>
<td>EquivalentClasses(CD(_1), CD(_2), i.e., Conjunction(SubClassof(CD(_1), CD(_2)), SubClassof(CD(_2), CD(_1)))</td>
<td>Equality(( 2\mu \text{IF}(\text{CD}_1, \text{CD}_2)))</td>
</tr>
<tr>
<td>SubPropertyof(PD(_1), PD(_2))</td>
<td>Subclass(( 2\mu \text{IF}(\text{PD}_1, \text{PD}_2)))</td>
</tr>
<tr>
<td>EquivalentProperties(PD(_1), PD(_2), i.e., Conjunction(SubPropertyof(PD(_1), PD(_2)), SubPropertyof(PD(_2), PD(_1)))</td>
<td>Equality(( 2\mu \text{IF}(\text{PD}_1, \text{PD}_2)))</td>
</tr>
</tbody>
</table>
CD₁ = CD₂; x is a variable; Formula, (i = 1, 2) denotes RIF formulas which are unary or binary; χᵣᵢᶠ is a transformation function (i.e., χᵣᵢᶠ(SWRL atoms) = RIF atoms).

It should be noted that, equivalence in SWRL consists of two cases: EquivalentClasses and EquivalentProperties. Moreover, Const in RIF corresponds to five cases in SWRL, i.e., constant names of individual, data value, class, individual property and data-valued property. More importantly, atoms, expressions and frames in RIF are represented by classes and properties in SWRL. Properties in SWRL can only represent binary relation. N-ary relations in RIF (including frames) are substituted by several binary relations in SWRL, and the relations between these binary relations are conjunctive. To save place, from section 2.2.2, the mappings of the main elements between languages will be represented by their XML syntaxes, rather than mapping tables of their abstract syntaxes.

2.2.2 The transformations between RIF and RuleML

From the RIF XML Schema [2] and the Glossary of RuleML 0.91 [7], we obtain the following conclusions. Firstly, <Document>, <Group>, <Implies>, <And>, <Or>, <Exists>, <ForAll>, <Member>, <Subclass>, <Equal> and <Var> in RIF correspond to <RuleML>, <Rulebase>, <Implies>, <And>, <Or>, <Exists>, <ForAll>, <InstanceOf>, <SubclassOf>, <Equal> and <Var>, respectively. Secondly, constants in RIF correspond to a relation name, a function name, or a slot name, as appropriate. Among others, relations in RuleML can be binary and n-ary. Thirdly, <Atom>, <Expr> and <Frame> in RIF correspond to <Atom> and <Expr> in RuleML. Especially, slots in RIF are represented by several conjunctive binary relations in RuleML. A slot name corresponds to a relation name, and a frame name and a filler correspond to arguments of the relation, i.e., a constant (individual or data value) or a variable.

2.2.3 The transformations between RIF and R2ML

Through analyzing the corresponding relations between the RIF XML Schema and the R2ML XML Schema [18], we obtain the following conclusions. In the first place, <Document>, <Group>, <Implies>, <And>, <Or>, <Member>, <Equal> and <Var> in RIF correspond to <r2ml:Rulebase>, <r2ml:DerivationRuleSet>, <r2ml:DerivationRule>, <r2ml:Conjunction>, <r2ml:Disjunction>, <r2ml:ObjectClassificationAtom>, <r2ml:EqualityAtom> and <r2ml:ObjectVariable>, respectively. In the second place, constants in RIF correspond to TypedLiteral name, ObjectClassificationAtom name, ReferencePropertyAtom name, or AttributionAtom name in R2ML. Additionally, there is no difference between <Atom> and <Expr> in R2ML. Relations in R2ML can only be binary. As with transformation from RIF to RuleML, slots in RIF are represented by several conjunctive binary relations in R2ML. A slot name corresponds to a relation name, and a frame name and a filler correspond to arguments of the relation, i.e., a constant (TypedLiteral) or a variable (ObjectVariable).

2.2.4 The transformations between RIF and F-logic

F-logic is an important language containing rules. Many new rule languages are based on F-logic. Therefore, it is necessary to construct mappings between RIF and F-logic. Frame
is the most basic element in F-logic. Many RIF elements are represented by frames. 
<Document>, <Implies>, <Or> and <Var> in RIF correspond to <flg:flogic>, <rule>,
<disjunct> and <variable> in F-logic, respectively. <Atom>, <Expr>, <Frame>, <Member>
and <Subclass> in RIF are all represented by <molecule> in F-logic, where <slot>
corresponds to <methodSpec arrow="\rightarrow">. It should be noted that, in F-logic, variables in
the head of a rule are existential, while variables in the body are universal, and conjunction
in F-logic is omitted.

2.3 Information Loss and Remedial Measures in RIAXML

Because expressive power of different rule languages differs, the issue of information
loss in rule interchange is not ignorable. Since transformations are bidirectional, informa-
tion loss is also bidirectional, i.e., from RIF to SWRL, RuleML, R2ML and F-logic, and
from SWRL, RuleML, R2ML and F-logic to RIF. Although information loss is unavoi-
dable, remedial measures can minish its influences. In the following, we discuss information
loss, and present corresponding remedial measures from two inverse directions. For brev-
ity, only most important aspects are given and some points are omitted.

2.3.1 From RIF to SWRL, RuleML, R2ML and F-Logic

Some elements in RIF cannot be mapped into the other four rule languages, which is
the exact reason that information loss occurs in the process of transformations from RIF to
SWRL, RuleML, R2ML and F-logic.

From RIF to SWRL  In this transformation, information loss shows mainly in two as-
pects. One is about rule implication, and the other is about frame. SWRL rules only allow
antecedent with the form of conjunctions of atoms, and consequent with a conjunction or
an atom. When RIF rules are of the form: antecedent with the form of disjunctions of at-
om, and consequent with conjunctions of atoms, the transformation will definitely fail.
However, we can indirectly implement this transformation. Because arbitrary proposition
can be transformed into disjunction norms, and \((a \lor b \rightarrow c)\) is equivalent to \(((a \rightarrow c) \land
(b \rightarrow c))\), the antecedent in RIF can be transformed into conjunctions of atoms. Fur-
thermore, using standard Lloyd-Topor transformation [12], a rule with conjunctive conse-
quenct can be transformed into conjunctions of rules whose consequents are atomic con-
sequences. Therefore, both antecedent and consequent can be transformed into the forms
of SWRL rules. For the second aspect, there is no element in SWRL to correspond to the
frame element in RIF, so direct transformation is impossible. However, frame can be approx-
imately considered as an \(n\)-ary relation. An \(n\)-ary relation can be transformed into \(C_n^2\)
binary relations. Therefore, we use binary relations to approximately substitute frames.
In SWRL, binary relations used to substitute the same frame have a common feature, i.e.,
they possess one and the same argument, i.e., frame name.

From RIF to RuleML  Although RuleML itself has \(n\)-ary relations, these relations are
different from frames in RIF. As with SWRL, approximate substitution is employed to
transform frames into several binary relations. Frame names become individuals or vari-
ables of RuleML binary relations, so do fillers in slots. RIF puts emphasis on the safety of
rules in it, while RuleML does not, \textit{i.e.,} in RuleML, there is no such restriction that variables that occurs in consequent must occur in antecedent.

**From RIF to R2ML**  
R2ML does not have elements to correspond to frame in RIF, either. The solution is also approximate substitution. Slot names in a frame can be considered as relation names in R2ML, frame names play the role of subjects in R2ML binary relations, and fillers in slots are considered as objects in R2ML binary relations. It should be noted that, there is no element in R2ML to correspond to subclass element in RIF.

**From RIF to F-logic**  
As with R2ML, F-logic only has unary and binary relations. Therefore, an \(n\)-ary relation in RIF is approximately represented by \(C_n^2\) binary relations in F-logic. And these binary relations are conjunctive. There are no built-ins in F-logic, so F-logic has to use unary and binary relations to substitute built-ins in RIF.

### 2.3.2 From SWRL, RuleML, R2ML and F-logic to RIF

The constructors in RIF are not rich enough to represent the real world knowledge, which leads to its relatively weak expressive power. In the following, we discuss the information loss from SWRL, RuleML, R2ML and F-logic to RIF in detail.

**From SWRL to RIF**  
Because of relatively weak expressiveness of RIF, there is no construct to correspond to ClassAtom (ComplementOf \((CD_1, CD_2), t\)) in SWRL. This is caused by the origin design goal of RIF. RIF was designed to be a simple dialect with limited expressiveness that lies with the intersection of first-order and logic-programming systems. This also tells us that, if we want to extend RIA\_XML to include more constructors, we have to breakthrough the limitation of RIF, \textit{i.e.,} we should be with an eye to the whole RIF dialects. Moreover, in RIF, there are no definitions for some attributes SWRL adopts from OWL, such as symmetry, transitivity, \textit{etc.} We have given some macro definitions to handle this issue.

**From RuleML to RIF**  
As with SWRL, the negative information in RuleML cannot be represented by RIF elements, let alone negation as failure (abbreviated NAF) which can represent nonmonotonic knowledge. More importantly, since RuleML version 0.9, RuleML has the power to represent fuzzy knowledge. It uses element \(<\text{degree}>\) to represent weights of atoms in rules, and to measure possibility degree with which an individual or a variable belongs to a given class. However, RIF does not have this power. The whole RIF dialects do not have enough power to represent fuzzy knowledge, either.

**From R2ML to RIF**  
As with RuleML, negation and NAF in R2ML cannot be represented by RIF elements. For production rules and reaction rules in R2ML, RIF has to fall back on RIF-PRD to look for counterparts.

**From F-logic to RIF**  
An important feature of F-logic is that disjunction in the head of a rule is allowed, while RIF is not. Therefore, direct transformations will for sure fail. These rules should be divided into several rules, each of which has only one atom or one conjunction in the head. And the relations between these rules which have been transformed should be disjunctive.
3. PROTOTYPE SYSTEM RIAXML 1.0

Based on rule interchange architecture RIAXML and fully considering information loss, we design and implement a prototype system of RIAXML, named RIAXML 1.0. We also give its experimental analysis.

3.1 System Implementation

Based on a programming language Java and on the Eclipse platform, we implement RIAXML 1.0, a prototype of RIAXML. To let computers “understand” RIF presentation syntax, we also implement transformations between RIF presentation syntax and RIFXML syntax in RIAXML 1.0, which is based on the transformation table in [2]. This way, RIAXML 1.0 supports bidirectional transformations among RIF presentation syntax, RIF XML syntax, SWRL, RuleML, R2ML and F-logic.

RIAXML 1.0 is implemented by adopting factory patterns and DOM (Document Object Model) models (as shown in Fig. 2). Concretely, When an XML document is inputted, a syntax checker will be invoked; through a dom4j library, the XML document is transformed into a DOM tree; with a transformation factory, the source document is transformed into the target document which is a DOM tree; this DOM tree is checked by a syntax checker to decide the correctness of the DOM tree; through the same Dom4j library, the target Dom tree is transformed into its XML document. Where, the transformation library consists of five modules, i.e., ria.p2x, ria.r2s, ria.r2rm, ria.r2r2 and ria.r2f, which are employed to transform DOM trees between RIF presentation syntax and RIF XML syntax, between RIF and SWRL, between RIF and R2ML, between RIF and R2ML, and between RIF and F-logic. All the transformations in RIAXML 1.0 adopt factory patterns which include a view part and a control part. All input documents are preprocessed by DOM models. We employ depth-first approach to traverse DOM trees. At the same time, the main function invokes static factory functions to deal with every node. Through one-to-one mappings and formatting, output documents are finally produced.

The main interface of RIAXML 1.0 is shown in Fig. 3. Let’s take a transformation between RIF XML syntax and R2ML as an example to illustrate implementation processes. The operational process is as follows: choosing transformation direction “RIF XML $\rightarrow$ R2ML” (in the bottom left corner), inputting a RIF XML document, pressing button “transform” (this moment, an imbedded syntax checker will check the syntax of input document), correcting syntax errors (if any), and displaying implementation results in the output box.
In the following, we give a running example to illustrate an application of our system. Inspired from the field of mobile communication, in order to increase profit, China Mobile (a mobile communication corporation based on GSM network in China) presents a new business. It is used to assert that if a person chooses M-ZONE service (a kind of mobile service) and he/she deposits more than 500 RMB into his new SIM card, then he/she will get a new Nokia mobile phone numbered 1861C for free. The presentation syntax of the above-mentioned marketing policy is shown in Table 2 and formally described as follows.

```
Document
Prefix (sce http://www.chinamobile.com/services#)
Prefix (cpt http://www.chinamobile.com/cpt#)
Prefix (pred http://www.w3.org/2007/rif-builtin-predicate#)
```

The following two code snippets are RIF XML syntax, and R2ML forms of the above example (the prefix is omitted), respectively. For saving space, the SWRL, RuleML and F-logic codes of the example are omitted. All the transformations mentioned above are available at [23]. First the RIF XML syntax of the above example is shown as follows.

```
<Document
xmlns="http://www.w3.org/2007/rif"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:xs="http://www.w3.org/2001/XMLSchema#">
<Group>
```

Table 2. The presentation syntax of the rule example.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Antecedent</th>
<th>Consequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>The marketing Policy</td>
<td>chooseService(?X, M-ZONE) deposit(?X, ?Y) numeric-great-than(?Y, 500)</td>
<td>Nokia1861C-Owner(?X)</td>
</tr>
</tbody>
</table>
\[ \forall X \forall Y \left( \frac{\text{&cpf;Nokia1861C-Owner}(X)}{\begin{align*} &\text{&cpf;chooseService}(X, \text{&sce;M-ZONE}) \\ &\text{&cpf;deposit}(X, Y) \\ &\text{&pred;numeric-great-then}(Y, 500) \end{align*}} \right) \]
is a declaration set of all variables used in rule implications; <Implies> is rule implication which consists of <then> and <if>; <then> is the consequent of the rule, and it includes atom Nokia1861C-Owner(X); <if> is the antecedent of the rule, and it includes there atoms defined in the presentation syntax of RIF; <Const> is the definition of class names, property names, individuals or constants; <args> is used to sort atoms in classes and properties.

Second the R2ML forms of the above example are shown as follows.

```xml
<r2ml:RuleBase>
  <r2ml:DerivationRuleSet>
    <r2ml:DerivationRule>
      <r2ml:Conclusion>
        <r2ml:ObjectClassificationAtom r2ml:classID="cpt:Nokia1861C-Owner">
          <r2ml:ObjectVariable r2ml:name="X"/>
        </r2ml:ObjectClassificationAtom>
      </r2ml:Conclusion>
      <r2ml:Condition>
        <r2ml:ReferencePropertyAtom r2ml:ReferencePropertyID="cpt:chooseService">
          <r2ml:object><r2ml:ObjectVariable r2ml:name="X"/></r2ml:object>
          <r2ml:subject><r2ml:ObjectVariable r2ml:name="services:M-ZONE"/></r2ml:subject>
        </r2ml:ReferencePropertyAtom>
        <r2ml:ReferencePropertyAtom r2ml:ReferencePropertyID="cpt:deposit">
          <r2ml:object><r2ml:ObjectVariable r2ml:name="X"/></r2ml:object>
          <r2ml:subject><r2ml:ObjectVariable r2ml:name="Y"/></r2ml:subject>
        </r2ml:ReferencePropertyAtom>
        <r2ml:DataPredicateAtom r2ml:DataPredicateAtomID="swrlb:greatThan">
          <r2ml:dataArguments>
            <r2ml:ObjectVariable r2ml:name="Y"/>
            <r2ml:TypeLiteral r2ml:lexicalValue="500" r2ml:type="xs:Integer"/>
          </r2ml:dataArguments>
        </r2ml:DataPredicateAtom>
      </r2ml:Condition>
    </r2ml:DerivationRule>
  </r2ml:DerivationRuleSet>
</r2ml:RuleBase>
```

R2ML defines three kinds of rules: integrity rules, derivation rules and production rules. Here, the example is a derivation rule. <r2ml:RuleBase> is the root element, and it contains a derivation rule set <r2ml:DerivationRuleSet> which contains a derivation rule <r2ml:DerivationRule>; <r2ml:DerivationRule> consists of consequent <r2ml:Conclusion> and antecedent <r2ml:Condition>; <r2ml:Conclusion> is a single atom Nokia1861C-Owner(X) included in <r2ml:ObjectClassificationAtom>, while <r2ml:Condition> includes three atoms defined in the presentation syntax of RIF; <r2ml:ReferencePropertyAtom> markups property atoms, and <r2ml:object> and <r2ml:subject> are used to mark up objects and subjects of properties; <r2ml:DataPredicateAtom> denotes built-in predicates.
3.2 Experimental Analysis

Existing rule interchange systems are implemented based on non-RIF languages (such as R2ML), which have different transformation tasks and transformation abilities. Therefore, performance comparisons between RIAXML 1.0 and the existing interchange systems are not included here.

The performance of RIAXML 1.0 is tested (as shown in Table 3) by transforming different rule examples. In Table 3, the mean time of all the transformations in RIAXML 1.0 is given. For each transformation, we give 20 different input documents. For transformations of each document, we choose the minimum value as the transformation time. There are two kinds of transformations in Table 3. They are one-step and two-step transformations. The former is like RIFpre \( \rightarrow \) SWRL, while the latter is of the form RuleML \( \rightarrow \) SWRL (in fact, it is RuleML \( \rightarrow \) RIFXML \( \rightarrow \) SWRL). Through comparing the mean time of one-step transformations with the mean time of two-step transformations, we note that, the former is not half of the latter, which means that much time is spent in the process of import and export of documents.

### Table 3. Mean time of all transformations in RIAXML 1.0.

<table>
<thead>
<tr>
<th></th>
<th>RIFpre</th>
<th>RIFXML</th>
<th>SWRL</th>
<th>RuleML</th>
<th>R2ML</th>
<th>F-logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIFpre</td>
<td>–</td>
<td>0.156</td>
<td>0.179</td>
<td>0.170</td>
<td>0.190</td>
<td>0.169</td>
</tr>
<tr>
<td>RIFXML</td>
<td>0.047</td>
<td>–</td>
<td>0.203</td>
<td>0.203</td>
<td>0.203</td>
<td>0.187</td>
</tr>
<tr>
<td>SWRL</td>
<td>0.173</td>
<td>0.187</td>
<td>–</td>
<td>0.234</td>
<td>0.234</td>
<td>0.223</td>
</tr>
<tr>
<td>RuleML</td>
<td>0.179</td>
<td>0.187</td>
<td>0.234</td>
<td>–</td>
<td>0.218</td>
<td>0.212</td>
</tr>
<tr>
<td>R2ML</td>
<td>0.198</td>
<td>0.187</td>
<td>0.234</td>
<td>0.218</td>
<td>–</td>
<td>0.218</td>
</tr>
<tr>
<td>F-logic</td>
<td>0.160</td>
<td>0.187</td>
<td>0.210</td>
<td>0.203</td>
<td>0.218</td>
<td>–</td>
</tr>
</tbody>
</table>

We have performed 600 transformations. It is shown that RIAXML 1.0 fully supports transformations between RIF presentation syntax and RIF XML syntax, and transformations between RIF and SWRL, R2ML, RuleML and F-logic, and thus between systems based on RIF and systems based on SWRL, R2ML, RuleML and F-logic. It is also shown that the use of RIF as the intermediary language in RIAXML 1.0 can considerably reduce the number of transformations and can hereby enhance transformation efficiency.

4. CONCLUSIONS

Rule interchange centered on RIF is the de facto mainstream direction of rule interchange. This paper has proposed a rule interchange architecture centered on RIF, which supports two-way transformations between RIF and SWRL, RuleML, R2ML and F-logic. The issue of information loss caused by different expressiveness of rule languages has been deeply discussed, and the corresponding remedial measures are given. A prototype system of rule interchange has been implemented.

In the near future, we plan to extend RIAXML 1.0 in the following three directions. Firstly, we will improve its robustness and performance (especially reducing execution time). Secondly, we will extend RIAXML 1.0 to support production rules and nonmonotonic
rules. Because there is much imprecise and uncertain knowledge in the Semantic Web, which can only be represented and reasoned by fuzzy rule languages, interchanging rule which can express fuzzy knowledge has become an important trend. So the third part of our future work is to fuzzify RIAXML 1.0 to enable rule interchange between fuzzy rule languages including f-SW-if-then-unless-RL [22], f-SWRL [15], Vague-SWRL [20] and f-NSWRL [21].

REFERENCES

23. RIAXML 1.0, ftp://idke.neu.edu.cn/ria.jsp.

Z. M. Ma (馬宗民) received his Ph.D. degree from City University of Hong Kong in 2001. He has been a Full Professor of College of Information Science and Engineering, Northeastern University, Shenyang, China, since 2004. His research interests mainly include intelligent database systems, semantic web, image retrieval, and artificial intelligence.

Xing Wang (王星) received his Ph.D. degree from Northeastern University, China. He is a Lecturer of School of Electronic and Information Engineering, Liaoning Technical University, Huludao, China. His current research interests are semantic web and artificial intelligence.