W3C Editor's Draft

RIF Primer

W3C Working Group Note 28 October 2012

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-primer-20121028/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-primer/
Editors:
Leora Morgenstern, SAIC
Chris Welty, IBM Research
Harold Boley, National Research Council Canada
Gary Hallmark, Oracle

This document is also available in these non-normative formats: PDF version.

Abstract

This document is a primer on the Rule Interchange Format (RIF). The primer provides a practical introduction to specifying declarative rules and production rules in RIF, in particular for the RIF BLD and PRD dialects. Examples of RIF specifications are developed in a stepwise manner.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is being published as one of a set of 12 documents:

1. RIF Overview
2. RIF Core Dialect
3. RIF Basic Logic Dialect
4. RIF Production Rule Dialect
5. RIF Framework for Logic Dialects
6. RIF Datatypes and Built-Ins 1.0

W3C Editor's Draft

7. RIF RDF and OWL Compatibility
8. OWL 2 RL in RIF
9. RIF Combination with XML data
10. RIF In RDF
11. RIF Test Cases
12. RIF Primer (this document)

Summary of Changes

@@tbd

Please Send Comments

Please send any comments to public-rif-comments@w3.org (public archive). Although work on this document by the Rule Interchange Format (RIF) Working Group is complete, comments may be addressed in the errata or in future revisions. Open discussion among developers is welcome at public-rif-dev@w3.org (public archive).

No Endorsement

Publication as a Working Group Note does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. The group does not expect this document to become a W3C Recommendation. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1 Introduction

1.1 What is a rule?

This document is meant to introduce computer professionals to basic techniques for writing declarative rules and production rules in the W3C Rule Interchange Format.

The dictionary defines a rule as a prescribed guide for conduct or a regulation governing behavior such as "Keep off the grass" or "A driver receiving a traffic ticket must appear in traffic court on the assigned date." In the computer science and logic programming communities, though, there are two different, but closely related ways to understand rules. One is closely related to the idea of an instruction in a computer program: If a certain condition holds, then some action is carried out. Such rules are often referred to as *production rules*. An example of a production rule is "If a customer has flown more than 100,000 miles, then upgrade him to Gold Member status."

Alternately, one can think of a rule as stating a fact about the world. These rules, often referred to as *declarative rules*, are understood to be sentences of the form "If P, then Q." An example of a declarative rule is "If a person is currently president of the United States of America, then his or her current residence is the White House."

Declarative rules do not specify an action that is to be carried out. For example, the rule above makes a statement about the relation between the U.S. president and the White House, but doesn't specify any action (say, to move the president into the White House). In other words, a declarative rule describes how the world is, rather than prescribing how things ought to be.

Production rules are frequently used for business applications. They typically incorporate an explicit notion of control to specify which rules are applied first. Declarative rules, on the other hand, are useful for writing down large amounts of information about a particular domain, or area of knowledge, independent of knowing, 'a priori', how these rules will eventually be used.

For example, one could write down rules in the medical domain:

- *If John's rapid-strep test is positive, then he is infected with strep.*
- *If John is infected with strep, then he needs to take antibiotics.*

Rules (whether declarative or production rules) can also be used to reason with other information that we have. In the example above, we can infer another rule:

- *If John's rapid-strep test is positive, then he needs to take antibiotics.*

It is particularly useful to reason with rules and facts, pieces of concrete information that do not make use of the if-then construct. Examples of facts are:

- *Elizabeth II became Queen of England in 1952.*
- *The Bronx is a borough of New York City.*
- *John's rapid-strep test is positive.*

To continue the example above, the fact

- *John's rapid-strep test is positive*

can be combined with the rules above to yield the conclusion:

- *John needs to take antibiotics.*

Rules and facts must be written in some sort of formal language in order for computer programs to reason with and draw conclusions from them. Such a computer program is often called a *rule engine*.

1.2 What is RIF?

There are many rule languages including SILK [GR09], OntoBroker [DEFS99], Eye [EULERSHARP10], VampirePrime [RZ09], N3-Logic [BCKSH08], and SWRL [SWRL-Ref] (declarative rule languages); and Jess [FH03], Drools [BA09], IBM ILog [ILOG10], and Oracle Business Rules [ORACLE10] (production rule languages). Many languages incorporate features of both declarative and production rule language. For example, Prolog [CM03], which is generally considered to be a declarative rule language, provides a cut operator for controlling the application of rules. Moreover, all production rule languages have a core subset that is declarative.

The abundance of rule sets in different languages can create difficulties if one wants to integrate rule sets, or import information from one rule set to another. How can a rule engine work with rule sets of different languages?

The W3C Rule Interchange Format (RIF) [RIF-Overview] is a standard that was developed to facilitate ruleset integration and synthesis. It comprises a set of interconnected dialects representing rule languages with various features. This document focus on the most basic RIF language, RIF-Core [RIF-Core], augmented by a set of datatypes and built-in functions and

predicates that can be used when writing rules [RIF-DTB]. All RIF dialects are an extension of RIF-Core plus DTB; we focus on two dialects, BLD and PRD.

1.3 What this document does and doesn't cover

This document focuses on the Basic Logic Dialect [RIF-BLD], and the Production Rule dialect [PRD], both of which are extensions of RIF-Core plus DTB. [RIF-FLD] is not discussed in this document.

This document does not include a detailed discussion of the semantics of any RIF dialect In particular, it does not discuss the model-theoretic semantics of BLD or the operational semantics of PRD. However, the document discusses the notions of assumption, consequence, and pattern matching. An intuitive understanding of these notions should enable a computer professional to understand the Primer and to write rules in RIF. Reading the BLD document [BLD] and the PRD document [PRD] is recommended for those who wish to learn more about, respectively, the model-theoretic and operational semantics used in RIF.

While this Primer is targeted at getting computer professionals to quickly learn how to write rules in RIF, it does not provide a complete specification of syntax. There are some details of RIF syntax, specifically those that are not necessary for writing most rules in RIF, which are not covered in this document. All syntactic details of RIF's logic dialects are covered in BLD, PRD, DTB, and FLD.

1.4 A note on syntax

The standard syntax for RIF is a verbose XML syntax, designed so that programs can easily generate and parse it. For human readers and writers, we generally use terser syntaxes that have a simple 1-1 correspondence with the XML. Different dialects have introduced different compact syntaxes: for example, BLD uses a compact syntax called Presentation Syntax, while PRD uses a compact syntax called Abstract Syntax.

One of the major stylistic differences between Presentation Syntax and Abstract Syntax is the way in which implications are written. In Presentation Syntax, an implication *If A then B* is written as

```
B :- A
```

while in Abstract Syntax, this implication is written as

```
If A Then B
```

We will use this syntax style, referred to as Mixed Presentation Syntax (MPS) whether presenting examples of BLD or PRD.

2 A Simple Example in RIF

2.1 Example Background

The RIF examples in this document concern the integration of data about films and plays across the Semantic Web. Suppose, for example, that one wants to combine data about films from IMDb, the Internet Movie Data Base (at http://imdb.com) with DBpedia (at http://dbpedia.org). Both resources contain facts about actors being in the cast of films, but DBpedia expresses these facts as a binary relation (aka predicate or RDF property).

In DBpedia, for example, one can express the fact that an actor is in the cast of a film:

- `starring(?Film ?Actor)`

where we use '?'-prefixed variables as placeholders. The names of the variables used in this example are meaningful to human readers, but not to a machine. These variable names are intended to convey to readers that the first argument of the DBpedia `starring` relation is a film, and the second an actor who stars in the film. (Variables are discussed in more detail in Section 2.2.2.)

In IMDb, however, one does not have an analogous relation. Rather, one can state facts of the following form about actors playing roles:

- `playsRole(?Actor ?Role)`

and one can state facts of the following form about roles (characters) being in films:

- `roleInFilm(?Role ?Film)`

Thus, for example, in DBpedia, one represents the information that Vivien Leigh was in the cast of *A Streetcar Named Desire*, as a fact

- `starring(Streetcar VivienLeigh)`

In IMDb, however, one represents two pieces of factual information, that Vivien Leigh played the role of Blanche DuBois:

- `playsRole(VivienLeigh BlancheDubois)`

and that Blanche DuBois was a character in *A Streetcar Named Desire*:

- `roleInFilm(BlancheDubois Streetcar)`

2.2 Developing the Example

The challenge in combining this data should be obvious: not only do the two data sources (IMDb and DBpedia) use different *vocabulary* (the relation names *starring, playsRole, roleInFilm*), but the *structure* is different. To combine this data, we essentially want to say something like the following:

If there are two facts in the IMDb database, saying that an actor plays a role/character, and that the character is in a film, then there is a single fact in the DBpedia database, saying that the actor is in the film.

This will be referred to as the *Basic Combination Rule*. We develop the Basic Combination Rule as a RIF rule in a stepwise manner in sections 2.2.1 through 2.3.

In this document, we incrementally introduce elements of RIF syntax and semantics, eventually building up to an example of valid RIF syntax. Often --- as is the case in the text boxes below, in sections 2.2.1 through 2.2.6 --- a single element or set of elements, that by itself is not a valid RIF rule set, is introduced. The red-tinted background in which these preliminary examples are displayed indicates that they are not fully valid RIF.

2.2.1 Atomic Formulas: Basic Facts

To prepare the Basic Combination Rule example, we first show how to write atomic formulas (atoms) in RIF.

Atoms can be formed from predicates applied to zero, one, two, or more arguments. Atoms can be used to state facts. For example:

- the nullary (0-argument) predicate application `itRains()` could stand for the proposition *it rains*.
- the unary (1-argument) predicate application `rapidStrepPos(John)` could stand for the proposition *John's rapid-strep test is positive*.
- the binary (2-argument) predicate application `playsRole(VivienLeigh BlancheDubois)` could stand for the proposition *Vivien Leigh played the role of Blanche DuBois*.

2.2.2 Constants and Variables

Like the atoms that make up molecules in the material world, logical atoms are not truly indivisible entities: they are, rather, composed of parts. The basic parts of an atom are *predicates* and *constants* (which are closely related, as discussed below), as well as *variables* and some other syntax, such as parentheses.

A constant is a term, or a symbol, used to refer to some specific (real or imagined) individual in the world (e.g. the constant `BlancheDubois` may refer to the character Blanche Dubois), or to some specific set of individuals (e.g. the constant `Actors` may refer to the set of all actors) or to some specific set of related individuals (e.g. the constant `starring` can refer to the set of all pairs of <x,y> where x stands for an actor, y stands for a film, and the starring relation holds between the individuals in each pair; the constant `producedFilm` can refer to the set of all triples <x,y,x> where x stands for a film producer, y stands for the film, and z stands for the year in which the film was produced). A set of related individuals, more commonly known as a set of ordered tuples, is known as a *predicate*.

Constant symbols mean nothing to machines; it is up to the humans who write and implement rules to interpret them in ways that make sense. As discussed below, one can write rules to help ensure the machine doesn't use the symbols in ways that violate their intended meaning.

A variable is a symbol, prefixed by a ?, that does not refer to anything specific in itself, but rather serves as a placeholder for writing general rules that range over larger sets of individuals. Variables in rules are similar, in this sense, to variables in computer programs. For example, the variable ?Actor could stand for any actor or actress.

As is the case with constants, the names of variables mean nothing to the machine, except to distinguish one variable from another. One may name the variable ?Actor in order to convey to human readers that the variable is intended to be a placeholder for Actors, but this does not yet convey that information to the machine.

2.2.3 Conjunctions and Implications

The general structure of the Basic Combination Rule is:

- *If firstatom and secondatom, then thirdatom*.

The rule contains a *conjunction*, a formula of the form *A and B* or, generally, *A1 and A2 and .... and An*.

In RIF, a conjunction is rewritten in prefix notation, e.g.

- the binary *A and B* is written as `And(A B)`.

Generally,

- the n-ary *A1 and A2 and ... and An* is written as `And(A1 A2 .... An)`.

The Basic Combination Rule also contains an *implication*, a statement of the form *if A then B*.

This implication is written almost unchanged in our notation, Mixed Presentation Syntax, as

- `If A Then B`.

Thus, one would write an implication of the form *If firstatom and secondatom, Then thirdatom* as

```
If And(firstatom secondatom) Then thirdatom
```

or in a pretty-print format with indentation indicating levels of a formula:

```
If   And(firstatom secondatom)
Then thirdatom
```

Note that in contrast, in RIF Presentation Syntax (PS) --- used, e.g., in [RIF-BLD] --- this is written in infix notation as `B :- A`, where the antecedent A and consequent B are reversed. The implication expresses the exact same meaning.

2.2.4 Quantifiers

The syntax that we have introduced so far allows us to write a rule that says:

*If IMDb contains the facts that **Vivien Leigh** played the role of **Blanche Dubois**, and that **Blanche Dubois** is a character role in **A Streetcar Named Desire**, then conclude the DBpedia fact that **Vivien Leigh** acted in **A Streetcar Named Desire** .*

This rule could be represented as

```
If   And(playsRole(VivienLeigh BlancheDubois)
         roleInFilm(BlancheDubois Streetcar))
Then starring(Streetcar VivienLeigh)
```

(Note how indentation is used to facilitate reading. **Bold-facing** is only used in this example to emphasize the correspondence between individual constants in the English and RIF versions of the rule.)

But this doesn't, of course, represent the Basic Combination Rule. The Basic Combination Rule says something about *all* actors, *all* roles, and *all* films.

To express this, we need to use variables with *quantifiers*. There are two sorts of quantifiers, *Forall*, known as the *universal quantifier*, and *Exists*, known as the *existential quantifier*. The universal and existential quantifiers can be used to form facts and rules.

For example, to say that all people like the film Casablanca, one can say:

```
Forall ?Person (likesFilm (?Person Casablanca) )
```

Note that while this is technically a rule, it is not a valid RIF rule yet.

To say that at least one person likes the film Casablanca, one can say (this, too, is not a valid RIF rule):

```
Exists ?Person (likesFilm (?Person Casablanca) )
```

Now it is possible to write a version of the Basic Combination Rule:

```
 Forall ?Actor ?Film ?Role (
  If And(playsRole(?Actor ?Role) roleInfilm(?Role ?Film))
  Then starring(?Film ?Actor)
)
```

There are still a few more steps needed to make this a valid RIF rule, as will be discussed in the next section.

Note also that in Presentation Syntax, this would read as:

```
Forall ?Actor ?Film ?Role (
  starring(?Film ?Actor) :-
    And(playsRole(?Actor ?Role) roleInFilm(?Role ?Film))
)
```

2.2.5 IRIs (Internationalized Resource Identifiers)

The formula shown in BCR-v0.1 is still not a correct formula in RIF. Individual constants like *VivienLeigh* and predicate constants like *playsRole* cannot be just used 'as is' but need to be disambiguated. This disambiguation addresses the issue that the constants used in this rule come from more than one database and may have different meanings --- that is, refer to different entities --- in each.

In RIF, disambiguation is effected using IRIs, Internationalized Resource Identifiers. (IRIs are a generalization of the concept of URIs; the primary distinction between the two is that IRIs allow characters from more alphabets than do URIs.) As with URIs, an IRI is a web address that typically includes some information about where the constant comes from (e.g. http://www.imdb.com/constants or http://dbpedia.org/resource).

Since using the full IRI notation can be cumbersome for every constant, RIF Mixed Presentation Syntax and other compact syntaxes of RIF allow an abbreviated form through namespace declarations. The specification of abbreviated form is explained in full in [RIF-BLD]. The general form of a prefix declaration can be quite complex. The basic idea is taht one can declare a namespace *ns* stands for the concept or entity described by the IRI *thisIRI* by writing the prefix declaration *Prefix(ns<ThisIRI>)*. Then the constant *name* can be disambiguated in rules using the string *ns:name*.

Consider the following example, in which, as is standard, http://example.com is an IRI reserved to demonstrate examples. Assume we are given the following declarations:

```
Prefix(imdbrel <http://example.com/imdbrelations#>)
Prefix(dbpedia <http://dbpedia.org/ontology/>)
```

The constant `imdbrel:playsRole` would be interpreted by RIF as the entity referred to by `http://example.com/imdbrelations#playsRole`, and the constant `dbpedia:starring` would be interpreted as the entity referred to by `http://dbpedia.org/ontology/starring`.

Using these prefixes, we can write our BCR with URIs for constants as follows:

```
Forall ?Actor ?Film ?Role (
  If   And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
  Then dbpedia:starring(?Film ?Actor)
)
```

(Note that in Presentation Syntax, this would read as:)

<div class="notvalid"

```
Forall ?Actor ?Film ?Role (
  dbpedia:starring(?Film ?Actor) :-
    And(imdbrel:playsRole(?Actor ?Role) imdbrel:roleInFilm(?Role ?Film))
)
```