Content Models for RuleML

Tshering Dema, David Hirtle, Omair Shafiq, Derek Smith

2011-09-25, version 1.0

Introduction

This document is a collection of content models, i. e. the content permitted within a particular element, for all RuleML elements and attributes as of version 1.0, organized alphabetically by module name. Each module is a grouping of related (XML) elements and/or attributes (prefixed with “@”). Under each element name, the attributes allowed are listed in the first line, with suffix ? to indicate optional attributes. We are able to do this because the content models of all attributes are independent of context. Following the attribute list, the element and text portion of the content model is given in compact Relax NG syntax, except for a few cases where the content model is cumbersome to express in Relax NG (Reify, Data). In those cases, we simply reproduce the XSD code. In attributes, default values, if any, are first in the list of possible values, and are shown in bold for emphasis. See http://www.ruleml.org/1.0/xsd/modules for the actual XML schemas of the modules and the RuleML glossary for the meaning of each tag.

Since RuleML is a family of sublanguages, it is important to note that the content model of a given node often varies according to the current sublanguage. In such cases, all variations of the content model are provided along with the corresponding sublanguage(s). The modularization of RuleML, including all sublanguages, is explained at http://www.ruleml.org/modularization.

Content models may also vary depending on context, i.e. surrounding elements (especially parent elements). In these cases, the content models are listed under a heading such as “within x…” where x indicates the context.

For clarification on any RuleML-related topic, including this document, the RuleML-all mailing list may be quite helpful. The RuleML tutorial serves as an introduction.

Index

atom

Atom

attributes: @closure?

in bindatagroundlog, bindatagroundfact and bindatalog:

```
( oid?, degree?, (op | Rel), slot*,((arg | arg_content), (arg | arg_content), slot*)? )
```

in datalog, nafdatalog, nafnegdatalog, and negdatalog:

```
( oid?, degree?, (op | Rel), slot*, ((arg | arg_content)+, slot*)? )
```

in hornlog & up (except framehohornlogeq):

```
( oid?, degree?, (op | Rel), slot*, resl?, ((((arg | arg_content)+, repo?) | repo), slot*,
      resl?)? )
```

in framehohornlogeq:

```
( oid, ( op | op_content )?, slot* )
```

within Reify in SWSL languages (SHOULD NOT BE PRESENT):

```
oid?, degree?, (op | Rel), slot*, resl?, ((((arg | Var | Skolem | Reify | Const | Uniterm)+,
      repo?) | repo), slot*, resl?)?
```

degree

attributes: none

in all sublanguages:

```
(Data)
```

op (context sensitive; see also the holog and expr modules)

attributes: none

within Atom...

in all sublanguages (except SWSL languages):

```
( Rel )
```

Rel

attributes: @iri?

in all sublanguages:

```
( text )
```

connective

if (context sensitive)

attributes: none

within Implies...

in all sublanguages:

```
( if_implies_content )
```

where if_implies_content =

in datalog & down and hornlog, dishornlog:

```
( Atom | And | Or )
```

in negdatalog:

```
( Atom | And | Or | Neg )
```

in nafdatalog & nafhornlog:

```
( Atom | And | Or | Naf )
```

in nafnegdatalog:

```
( Atom | And | Or | Neg | Naf )
```

in hornlogeq:

```
( Atom | And | Or | Equal )
```

in hohornlog:

```
( And | Or | Naf | Uniterm | Neg )
```

in hohornlogeq:

```
( And | Or | Naf | Uniterm | Neg | Equal )
```

in framehohornlogeq:

```
( And | Or | Naf   | Uniterm   | Neg   | Equal   | Atom   | InstanceOf   | SubclassOf   |
      Signature )
```

in folog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffolog:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists )
```

in fologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

in naffologeq:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists | Equal )
```

within Entails...

in all sublanguages:

```
( if_entails_content )
```

where if_entails_content =

in all sublanguages:

```
( Rulebase )
```

Implies

attributes: @closure?, @direction?, @material? (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, ((then, if) | (if, then) | (if_implies_content, then_implies_content)) )
```

Entails

attributes: none

in all sublanguages:

```
( oid?, (if | Rulebase), (then | Rulebase) )
```

Equivalent

attributes: @closure? (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, ((torso, torso) | ( torso_content, torso_content )) )
```

torso

attributes: none

in all sublanguages:

```
( torso_content )
```

where torso_content =

in datalog & down and up to (and including) dishornlog:

```
( Atom )
```

BUT in dishornlog SHOULD BE:

```
( Atom | Or )
```

in hornlogeq:

```
( Atom | Equal )
```

in hohornlog:

```
( Uniterm )
```

in hohornlogeq:

```
( Uniterm | Equal )
```

in framehohornlogeq:

```
( Atom | Uniterm | InstanceOf | SubclassOf | Signature | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in fologeq & naffologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

Rulebase

attributes: @mapDirection?, @mapClosure? and @mapMaterial?

in all sublanguages:

```
( oid?, (formula | formula_rulebase_content)* )
```

And

attributes {within Query only: @closure?} (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_andor_content)* )
```

Or

attributes {within Query only: @closure?} (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_andor_content)* )
```

formula (context sensitive; see also performative and quantifier modules)

attributes: none

within Rulebase...

```
( formula_rulebase_content )
```

where formula_rulebase_content =

in bindatagroundfact:

```
( Atom )
```

in bindatagroundlog:

```
( Atom | Implies | Equivalent )
```

in bindatalog, datalog, nafdatalog, hornlog, nafhornlog dishornlog:

```
( Atom | Implies_2 | Equivalent_2 | Forall_2 )
```

in negdatalog, nafnegdatalog:

```
( Atom | Implies_2 | Equivalent_2 | Forall_2 | Neg )
```

in hornlogeq:

```
( Atom | Implies_2 | Equivalent_2 | Forall_2 | Equal )
```

in hohornlog:

```
( Implies | Equivalent | Forall | Uniterm | Neg )
```

in hohornlogeq:

```
( Implies | Equivalent | Forall | Uniterm | Neg | Equal )
```

in framehohornlogeq:

```
( Implies | Equivalent | Forall | Uniterm | Neg | Equal | Atom | InstanceOf | SubclassOf |
    Signature )
```

in folog, naffolog, naffologeq:

```
( Atom | Implies_2 | Equivalent_2 | Forall_2 | And | Or | Neg | Exists )
```

in fologeq:

```
( Atom | Implies_2 | Equivalent_2 | Forall_2 | And | Or | Neg | Exists | Equal )
```

within And/Or...

```
( formula_andor_content )
```

where formula_andor_content =

in datalog & down, hornlog and dishornlog:

```
( Atom | And | Or )
```

in negdatalog:

```
( Atom | And | Or | Neg )
```

in nafdatalog and nafhornlog:

```
( Atom | And | Or | Naf )
```

in nafnegdatalog:

```
( Atom | And | Or | Naf | Neg )
```

in hornlogeq:

```
( Atom | And | Or | Equal )
```

in hohornlog:

```
( And | Or | Naf | Uniterm | Neg )
```

in hohornlogeq:

```
( And | Or | Naf | Uniterm | Neg | Equal )
```

in framehohornlogeq:

```
( And | Or | Naf   | Uniterm   | Neg   | Equal   | Atom   | InstanceOf   | SubclassOf   |
      Signature )
```

in folog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffolog:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists )
```

in fologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

in naffologeq:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists | Equal )
```

@mapMaterial

(**yes** | no)

@material

(**yes** | no)

@mapDirection

(**bidirectional** | forward | backward)

@direction

(**bidirectional** | forward | backward)

@mapClosure

(universal | existential)

@closure

(universal | existential)

desc

oid

attributes: none

in bindatagroundfact and bindatagroundlog :

```
( arg_content | Var )
```

in all other sublanguages except framehohornlogeq:

```
( arg_content )
```

framehohornlogeq:

```
( op_content )
```

equality

Equal

attributes: none

in hornlogeq, fologeq, naffologeq, hohornlogeq, framehohornlogeq

```
( oid?, degree?, ((left, right) | (arg_content, arg_content)) )
```

left

attributes: none

in hornlogeq, fologeq, naffologeq , hohornlogeq, framehohornlogeq:

```
( arg_content )
```

right

same as left

@oriented

(**no** | yes)

@val

(**0..** | 1)

expr

Expr

attributes: @type?

in hornlog & up (except hohornlog, etc):

```
( oid?, (op | Fun), slot*, resl?, ((((arg | arg_content)+, repo?) | repo), slot*, resl?)? )
```

op (context sensitive; see also the atom and holog modules)

within Expr:

attributes: none

in hornlog & up (except SWSL languages):

```
( Fun )
```

Fun

attributes: @iri?

in hornlog & up (except SWSL languages):

```
( text )
```

Plex (context sensitive)

attributes: none

within Atom, Plex, slot...

in hornlog & up :

```
( oid?, slot*, (((arg | arg_content)+, repo?, slot*, resl?)? | (repo, slot*, resl?) | resl) )
```

within repo...

in hornlog & up:

```
( ( arg | arg_content | repo )* )
```

within resl...

in hornlog & up including hohornlog, etc. :

```
( (slot | resl )* )
```

@per

```
( copy | open | value | effect | model )
```

frame

Set

attributes: none

in framehohornlogeq:

```
( arg_content* )
```

InstanceOf

attributes: none

in framehohornlogeq:

```
( arg_content, arg_content )
```

SubclassOf

attributes: none

in framehohornlogeq:

```
( arg_content, arg_content )
```

Signature

attributes: none

in framehohornlogeq:

```
( oid, (op | op_content)?, slot* )
```

Get

attributes: none

in framehohornlogeq:

```
( oid, SlotProd )
```

SlotProd

attributes: none

in framehohornlogeq:

```
( ( arg_content)+ )
```

holog

Uniterm

attributes: none

in hohornlog, hohornlogeq & framehohornlogeq:

```
( oid?, (op | op_content), slot*, resl?, (((arg | arg_content)+, repo?) | repo), slot*, resl?)?)
```

op (context sensitive; see also the atom and expr modules)

attributes: none

within Uniterm and Signature...

in hohornlog & up:

```
( op_content )
```

where op_content =

```
( Const | Skolem | Var | Reify | Uniterm )
```

Const

attributes: @iri?, @type?

in hohornlog & up:

```
( text )
```

@minCard

in hohornlog & up:

```
( xsd:nonNegativeInteger )
```

@maxCard

in hohornlog & up:

```
( xsd:nonNegativeInteger )
```

iri

@iri

in all sublanguages:

```
( xsd:anyURI )
```

naf

Naf

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, ( weak | weak_content) )
```

weak

attributes: none

in all sublanguages:

```
( weak_content )
```

where weak_content =

in nafdatalog:

```
( Atom )
```

in nafnegdatalog:

```
( Atom | Neg )
```

in hohornlog

```
( Uniterm )
```

in naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffologeq:

```
( Atom | And| Or| Neg| Implies| Equivalent| Forall| Exists| Equal )
```

neg

Neg

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (strong | strong_content) )
```

strong

attributes: none

in all sublanguages:

```
( strong_content )
```

where strong_content =

in negdatalog and nafnegdatalog:

```
( Atom )
```

in hohornlog:

```
( Uniterm )
```

in hohornlogeq & up:

```
( Uniterm | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in fologeq and naffologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

performative

RuleML

attributes: none

in all sublanguages:

```
( oid?, (act | Assert | Query | Retract)* )
```

act

attributes: none

in all sublanguages:

```
( Assert | Query | Retract )
```

Assert

attributes: @mapDirection?, @mapClosure? and @mapMaterial?

in all sublanguages:

```
( oid?, (formula | formula_assert_retract)* )
```

Retract

same as Assert

Query

attributes: @closure? (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_query)* )
```

formula (context sensitive: see also connective and quantifier modules)

attributes: none

within Assert and Retract…

```
( formula_assert_retract )
```

where formula_assert_retract =

in bindatagroundfact:

```
( Rulebase | Atom | Entails )
```

in bindatagroundlog:

```
( Rulebase | Atom | Implies | Equivalent | Entails )
```

in bindatalog, datalog, hornlog & dishornlog, nafdatalog, nafhornlog:

```
( Rulebase | Atom | Implies | Equivalent | Entails | Forall )
```

in negdatalog and nafnegdatalog:

```
( Rulebase | Atom | Implies | Equivalent | Entails | Forall | Neg )
```

in hornlogeq:

```
( Rulebase | Atom | Implies | Equivalent | Entails | Forall | Equal )
```

in hohornlog:

```
( Rulebase | Implies | Equivalent | Entails | Forall | Uniterm | Neg )
```

in hohornlogeq:

```
( Rulebase | Implies   | Equivalent   | Entails   | Forall   | Uniterm | Neg | Equal )
```

in framehohornlogeq:

```
( Rulebase | Implies   | Equivalent   | Entails   | Forall   | Uniterm   | Neg   | Equal   |Atom
      | InstanceOf | SubclassOf | Signature )
```

in folog and naffolog:

```
( Rulebase | Atom | Implies   | Equivalent   | Entails | Forall   | And   | Or   | Neg   | Exists
      )
```

in fologeq and naffologeq:

```
( Rulebase | Atom   | Implies   | Equivalent   | Entails | Forall   | And   | Or   | Neg   |
      Exists | Equal )
```

within Query...

```
( formula_query )
```

where formula_query =

in bindatagroundfact and bindatagroundlog:

```
( Rulebase | And | Or | Atom | Entails_)
```

in bindatalog, datalog, hornlog, dishornlog:

```
( Rulebase | Atom | And | Or | Entails | Exists )
```

in nafdatalog, nafhornlog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Naf )
```

in negdatalog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg )
```

in nafnegdatalog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Naf )
```

in framehohornlogeq:

```
( Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg | Equal | Atom | InstanceOf |
      SubclassOf | Signature )
```

in hohornlog:

```
( Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg )
```

in hohornlogeq:

```
( Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg | Equal )
```

in folog:

```
( Rulebase | Atom | And | Or  | Entails   | Exists   | Neg   | Implies | Equivalent | Forall )
```

in fologeq:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Equal )
```

in naffolog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Naf )
```

in naffologeq:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Naf |
    Equal )
```

quantifier

Forall

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (declare | Var)+, (formula | formula_forall) )
```

Exists

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (declare | Var)+, (formula | formula_exists) )
```

declare

attributes: none

in all sublanguages:

```
( Var )
```

formula (context sensitive; see also the connective and performative modules)

attributes: none

in bindatalog and up:

within Forall...

```
( formula_forall )
```

where formula_forall =

in bindatalog, datalog & up to (including) hornlog and dishornlog:

```
( Atom | Implies | Equivalent | Forall )
```

in hornlogeq:

```
( Atom | Implies | Equivalent | Forall | Equal )
```

in hohornlog:

```
( Uniterm | Implies | Equivalent | Forall )
```

in hohornlogeq:

```
( Uniterm | Implies | Equivalent | Forall | Equal )
```

in framehohornlogeq:

```
( Atom | Uniterm | InstanceOf | SubclassOf | Signature | Implies | Equivalent | Forall | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in fologeq and naffologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

within Exists...

```
( formula_exists )
```

where formula_exists =

in bindatalog, datalog & up to (including) hornlog and dishornlog:

```
( Atom | And | Or | Exists )
```

in hornlogeq:

```
( Atom | And | Or | Exists | Equal )
```

in hohornlog:

```
( Uniterm | And | Or | Exists )
```

in hohornlogeq:

```
( Uniterm | And | Or | Exists | Equal )
```

in framehohornlogeq:

```
( Atom | Uniterm | InstanceOf | SubclassOf | Signature | And | Or | Exists | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in fologeq and naffologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

<u>rest</u>

repo

attributes: none

in hornlog & up:

```
    ( Var | Plex )
```

resl

attributes: none

in hornlog & up:

```
    ( Var | Plex )
```

slot

slot (context sensitive)

attributes: @card?, @weight?

within Atom, Expr, Plex, Uniterm

in bindatagroundlog, & up (except for the SWSL languages):

```
( (Ind | Data), arg_content )
```

in hohornlog & hohornlogeq:

```
( ( Const | Uniterm ), arg_content )
```

in framehohornlogeq:

```
( ( Const | Uniterm | Get ), arg_content )
```

within Signature, Atom-frame...

attributes: (+ @minCard? and @maxCard?)

in framehohornlogeq:

```
( ( Const | Uniterm | Get ), arg_content? )
```

@card

in all sublanguages:

```
( xsd:nonNegativeInteger )
```

@weight

in all sublanguages:

```
( xsd:decimal { minInclusive = "0" maxInclusive = "1" } )
```

term

arg

attributes: @index

in all sublanguages:

```
( arg_content )
```

where arg_content =

in bindatalog, datalog & up to hornlog:

```
( Ind | Data | Skolem | Var | Reify )
```

in bindatagroundlog and bindatagroundfact:

```
( Ind | Data | Skolem | Reify )
```

in hornlog & up (except hohornlog, etc):

```
( Ind | Data | Skolem | Var | Reify | Expr | Plex )
```

in hohornlog & hohornlogeq:

```
( Const | Skolem | Var | Reify | Uniterm )
```

in framehohornlogeq:

```
( Const | Skolem | Var | Reify | Uniterm | Get | Set )
```

Ind

attributes: @iri?, @type?

in all sublanguages:

```
( text )
```

Data

attributes: @xsi:type?

in all sublanguages:

```
( xsd:anyType )
```

Var

attributes: @type?

in all sublanguages:

```
( text )
```

Skolem

attributes: @type?

in all sublanguages:

```
( text )
```

Reify

attributes: none

in all sublanguages:

```
( <xs:any processContents="strict" namespace="##targetNamespace"/>? )
```

@type

in all sublanguages:

```
( text )
```

@index

in all sublanguages:

```
( xsd:positiveInteger )
```