The RuleML Knowledge-Interoperation Hub

Harold Boley

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
harold[DT]boley[AT]unb[DT]ca

Abstract. The RuleML knowledge-interoperation hub provides for syntactic/semantic representation and internal/external transformation of formal knowledge. The representation system permits the configuration of textbook and enriched Relax NG syntax as well as the association of syntax with semantics. The transformation tool suite includes serialized formatters (normalizers and compactifiers), polarized parsers and generators (the RuleML↔POSL tool and the RuleML→PSOA/PS generator and PSOA/PS→AST parser), as well as importers and exporters (the importer from Dexlog to Naf Datalog RuleML and the exporter from FOL RuleML languages to TPTP). An N3-PSOA-Flora knowledge-interoperation use case is introduced for illustration.

1 Introduction

RuleML focuses on structured knowledge, as used, e.g., in data, domain, and process modeling. Such knowledge is often represented with ontologies and rules, which may be combined in hybrid or homogeneous ways. Description logics, underlying various ontologies, can be homogeneously combined with the decidable Datalog$^{\pm}$ [1], a compact rule language including head existentials; corresponding rule engines are being increasingly used for efficient ontology reasoning [2]. Similarly, ontological subsumption axioms and rule-based mappings can be combined for uniform Rule-Based Data Access [3]. Moreover, Inductive (Functional-Logic) Programming is based on the rule paradigm, and employed in industrial applications [4]. When decidability of querying is not aimed for, knowledge-representation expressivity can be extended from Datalog, Datalog$^{\pm}$, and description logics to, e.g., Datalog^{+}, Horn logic, as well as FOL and higher-order rule languages.

This article presents the RuleML hub for interoperating structured knowledge formalized via a system of rule families ranging from declarative/deliberative condition-conclusion rules to stateful/reactive event-condition-action rules.

Formal rule knowledge can be collected in knowledge bases (KBs) and transformed on a network such as an intranet or the Internet, specifically the Web. Utilizing a hub and spoke model, *knowledge interoperation* benefits from a canonical knowledge-representation language allowing knowledge transformation via translators mapping through this canonical form. RuleML as an open non-profit organization has developed a series of rule specifications leading to Version 1.02[1]

[1] http://wiki.ruleml.org/index.php/Specification_of_RuleML_1.02.

of the RuleML system, whose novel Consumer RuleML family achieves an initial integration of the main Deliberation and Reaction RuleML families.

The Web-based RuleML tools for knowledge interoperation (representation and transformation) have reached a critical mass, where synergies are becoming possible such as novel chains of translators mapping through RuleML/XML. A variety of useful interoperation tools is described on, or linked from, the categorized RuleML Wiki[2] (currently consisting of 1247 pages), although they have not yet been discussed in a synthesis article, in spite of interoperation being central to RuleML.

The current presentation thus gives a top-down account of the RuleML knowledge-interoperation hub, focusing on advances in syntactic/semantic representation and internal/external transformation. The two main knowledge-interoperation components will be expanded in Section 2, on knowledge representation, and Section 3, on knowledge transformation, followed, in Section 4, by a knowledge-interoperation use case and, in Section 5, conclusions.

2 Knowledge Representation System

The RuleML knowledge-representation architecture consists of a system of families of **languages** of XML-serialized instance documents (containing KBs and queries) specified syntactically through schemas (for Deliberation RuleML, normatively in Relax NG, from which XSD is generated) and associated with semantic **profiles** through syntax-semantics-pairing **logics** as appropriate. For each pair *logic* = (*language*, *profile*), *language* is predefined but *profile* and *logic* are predefined or user-defined (where *logic* can be predefined only if *profile* is).

2.1 Configuration of Textbook and Enriched Relax NG Syntax

RuleML's modular schemas permit rule interchange with high precision. Deliberation RuleML 1.0 introduced a modularization approach based on the schema language Relax NG [5], restricted to be *monotonic*: When two modules are combined, e.g. by including them both into a larger schema, the language defined by the larger schema contains both of the languages defined by the modules. Because of this monotonicity property, the more than fifty Deliberation RuleML 1.02 schema modules may be freely combined to define a fine-grained poset lattice of languages, with a partial order based on syntactic language containment.

To select from the many resulting predefined languages of the Deliberation RuleML family, the Modular sYNtax confiGurator (MYNG) application [6] was developed for providing a unified parameterized schema accessible either directly, using a REST interface, or through a GUI that exposes the REST interface.

MYNG may be used to configure a RuleML language with a set of desired features. Relax NG schemas configured using MYNG 1.02[3] may be employed

[2] `http://wiki.ruleml.org/index.php/Special:Categories`.
[3] `http://deliberation.ruleml.org/1.02/myng/`.

outside of MYNG for schema-aware authoring, instance validation, or parser generation through XML tools such as oXygen XML and JAXB.

All MYNG-configured RuleML languages have a unique myng-code URL. Members of the subset of *anchor languages* additionally have a (composite) name. An example from Deliberation RuleML is the anchor language Datalog$^+$ (more precisely, the language defined by its Relax NG schema[4]). Starting with its Version 1.01, an "Instructive KB"[5] has been made available including examples from [1]. This KB (with embedded queries) has acted as a Datalog$^+$ RuleML paradigm also for the Rulebase Competition 2014 held at the 8th International Rule Challenge[6], whose KBs from the RCC-geospatial, investment-regulation, and car-insurance domains, along with their descriptions in the proceedings papers, are collected in one place[7].

RuleML – as a rich knowledge modeling system supporting, e.g., Web rules – comes with supplementary features for (Semantic) Web applications such as (optional) IRIs, OIDs, types, and slots in its specification of anchor languages, including Datalog($^+$), Hornlog($^+$), and FOLog. For users who just need unsupplemented languages (where such features are not even optional), RuleML has started to introduce "textBooK" (BK) language versions without any supplementary features, including DatalogBK($^+$), HornlogBK($^+$), and FOLogBK. Since myng-codes for such unsupplemented languages already exist amongst the vast variety [6] of myng-codes of the language lattice, once identified, they can be easily designated as BK anchor languages.

2.2 Logics Associating Syntactic Languages with Semantic Profiles

Rather than assuming a default semantics for syntactically defined languages, RuleML 1.02 leaves their semantics unspecified by default; this is motivated by concerns for security, scalability, refinability, and application requirements.[8] Instead, each RuleML 1.02 document permits to prominently refer to a logic.

RuleML logics are *syntax-semantics* pairs, associating a syntactic language with a semantic profile. The *syntax* is a predefined language as MYNG-configured in Section 2.1. The *semantics* is defined by a profile of descriptors including: (a) a classification distinguishing Proof(-theoretic) vs. Model(-theoretic), with the former being subclassified as Resolution vs. ASP etc., the latter as Herbrand vs. Tarski, all of which can be further qualified by fine distinctions; (b) a reference to a Web-published semantics and a mapping between its syntax and RuleML/XML syntax.

---

[4] `http://deliberation.ruleml.org/1.02/relaxng/datalogplus_min_relaxed.rnc`.

[5] `http://deliberation.ruleml.org/1.02/exa/DatalogPlus/datalogplus_min.ruleml`.

[6] `http://2014.ruleml.org/challenge`.

[7] `http://deliberation.ruleml.org/1.02/exa/RulebaseCompetition2014/`.

[8] `http://wiki.ruleml.org/index.php/Specification_of_RuleML_1.02`.

The two components constituting a logic allow for many-to-many relationships, where multiple syntaxes can have one semantics (see Section 3), and one syntax can have multiple semantics, as exemplified next.

The *predefined logics* of RuleML 1.02 include Horn-Herbrand, associating the Hornlog RuleML syntax with Herbrand semantics. For a user group requiring Tarski models, this logic can be complemented by a *user-defined logic* Horn-Tarski, associating the same syntax with Tarski semantics (in a future RuleML version, this might also become predefined). Moreover, both components of Horn-Herbrand can be refined, e.g. for negation-as-failure, leading to, e.g., NafHorn-HerbrandWF (Naf with Well-Founded semantics) or to NafHorn-HerbrandSM (Naf with Stable Model semantics).

Positional-Slotted, Object-Applicative RuleML (PSOA RuleML) [7, 8][9] uses psoa atoms which permit the application of a predicate (acting as a relation) to be [in an *oidless*/*oidful* dimension] without or with an Object IDentifier (OID) – typed by the predicate (acting as a class) – and the predicate's arguments to be [in an orthogonal dimension] *positional*, *slotted*, or *combined*.

PSOA RuleML's presentation syntax PSOA/PS [7] was complemented by an XML syntax [8] defined by an XSD schema[10], which can be translated to a Relax NG schema called HornPSOA.

PSOA RuleML has a Tarski semantics [7], which could be complemented by a Herbrand semantics. Both semantics provide a model theory for PSOA RuleML, whose object identifiers lead to (head-)existential rules (which can be Skolemized to Horn rules). While the Tarski semantics of PSOA RuleML refers to the online version of [7], its Herbrand semantics could refer to a direct definition or to a definition with the domain of a PSOA RuleML semantic structure in [7] becoming the set of all equivalence classes over the Herbrand PSOA RuleML universe, adapting the Herbrand RIF-FLD Subframework[11].

The resulting predefined logic HornPSOA-Tarski could be complemented by a user-defined logic HornPSOA-Herbrand (in a future RuleML version, this might also become predefined).

An interoperation use case for bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping) of – flat and nested – addresses is developed in the PSOA RuleML tutorial [8]. For a geospatial use case see [9].

3 Knowledge Transformation Tool Suite

Based on the RuleML knowledge representation of Section 2, we now proceed to the suite of tools for (semantics-preserving) knowledge transformation (RuleML/XML is the 'machine-oriented' RuleML serialization syntax; *RuleML/short* stands for 'human-oriented' RuleML shorthand syntaxes such as POSL and PSOA/PS; *foreign* stands for non-RuleML syntaxes such as Prolog and RIF/PS):

[9] `http://wiki.ruleml.org/index.php/PSOA_RuleML`.
[10] `http://wiki.ruleml.org/index.php/PSOA_RuleML_API`.
[11] `https://www.w3.org/TR/rif-fld/#Appendix:_A_Subframework_for_Herbrand_Semantic_Structures`.

- Internal: RuleML-to-RuleML
  - Serialized: RuleML/XML-to-RuleML/XML
    - Upgraders (e.g., to Version 1.02[12])
    - Formatters (e.g., for Version 1.02[13])
      - Normalizer (Section 3.1)
      - Compactifiers (Section 3.1)
  - Polarized (Sections 3.2, 3.2): Between-RuleML/XML-and-*RuleML/short*
    - Parsers: *RuleML/short*-to-RuleML/XML
    - Generators: RuleML/XML-to-*RuleML/short*
- External: Between-RuleML/XML-and-*foreign*[14]
  - Importers (Section 3.3): *foreign*-to-RuleML/XML
  - Exporters (Section 3.3): RuleML/XML-to-*foreign*

On the top-level, this tool-suite taxonomy distinguishes transformations that are Internal to RuleML from those that are External in the sense of mapping – in either direction – between RuleML and foreign syntaxes. The more deeply differentiated Internal branch is then divided into Serialized transformations (staying within the XML syntax of RuleML) and Polarized transformations (having both a RuleML/XML and a *RuleML/short* side). The Parsers and Generators under the Polarized sub-branch of the Internal branch as well as the Importers and Exporters of the External branch can be composed. This creates transformation chains mapping through RuleML/XML as in the following compositions, where POSL and PSOA/PS are two 'shorthand' syntaxes for a Deliberation RuleML subset,[15] while Dexlog [10] and TPTP refer to subsets of two 'foreign' syntaxes:

- Internal-Internal: POSL⟶RuleML/XML⟶PSOA/PS
- External-External: Dexlog⟶RuleML/XML⟶TPTP
- Internal-External: POSL⟶RuleML/XML⟶TPTP
- External-Internal: Dexlog⟶RuleML/XML⟶PSOA/PS

The following subsections will traverse this taxonomy, expanding on a selection of its leaf nodes.

3.1 Serialized Formatters

Normalizer RuleML has always allowed abbreviated serialization (skipped edge tags) and some freedom in the ordering of elements. XSLT stylesheets have been developed[16] for normalizing the syntax used in a given Version 1.02 instance, filling in any skipped edges and sorting elements into a canonical order.

The goals of the RuleML Normalizer include the following:

[12] `http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.02#XSLT-Based_Upgrader`.

[13] `http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.02#XSLT-Based_Formatters`.

[14] External transformations should be defined via the normative RuleML/XML, rather than via any *RuleML/short*, while *foreign* may be any normative XML or other format.

[15] On the other hand, Prova is a shorthand syntax for a Reaction RuleML subset.

[16] `http://deliberation.ruleml.org/1.02/xslt/normalizer/`.

- Reconstruct all skipped edge tags to produce a fully striped form [since edge tags correspond to (RDF) properties, this simplifies interoperation between RuleML/XML and directed labeled (RDF) graphs]
- Perform canonical ordering of sibling elements [this reduces the complexity of equality comparison across RuleML/XML serializations, for both humans and machines]

As a first example, the Existential Datalog$^+$ rule

```
<Forall>
  <Var>H</Var>
  <Implies>
    <Atom>
      <Rel>human</Rel>
      <Var>H</Var>
    </Atom>
    <Exists>
      <Var>M</Var>
      <Atom>
        <Rel>hasMother</Rel>
        <Var>H</Var>
        <Var>M</Var>
      </Atom>
    </Exists>
  </Implies>
</Forall>
```

is normalized to

```
<Forall>
  <declare><Var>H</Var></declare>
  <formula>
    <Implies>
      <if>
        <Atom>
          <op><Rel>human</Rel></op>
          <arg index="1"><Var>H</Var></arg>
        </Atom>
      </if>
      <then>
        <Exists>
          <declare><Var>M</Var></declare>
          <formula>
            <Atom>
              <op><Rel>hasMother</Rel></op>
              <arg index="1"><Var>H</Var></arg>
              <arg index="2"><Var>M</Var></arg>
            </Atom>
          </formula>
        </Exists>
      </then>
```

```
    </Implies>
  </formula>
</Forall>
```

As a second example, the compact version[17] of the Datalog$^+$ example is normalized to the expanded version[18].

Normalization is a preparatory step for many other transformations such as the compactifiers (cf. Section 3.1) and the TPTP exporters (cf. Section 3.3).

Compactifiers The compactifier XSLTs[19] specify formatting into a compact serialization, which has fewer elements (i.e. is more compact) than the normalized serialization.

Two variations of this formatter specification are provided due to some limitations of XSD schemas. Both first apply the normalizer of Section 3.1 to sort the child nodesets into the canonical order. Then the "full" compactifier specifies the removal of all skippable RuleML stripes, while the "ifthen" compactifier retains `<if>` and `<then>` edges to provide disambiguating contexts for certain elements.

For example, as two inversions of the first example in Section 3.1, the full and ifthen compactifiers transform the expanded version to their compact versions without (shown there) and with retained `<if>` and `<then>` edges, respectively.

3.2 Polarized Parsers and Generators

RuleML↔POSL Parser & Generator Tool The POsitional-SLotted (POSL) shorthand syntax of Hornlog RuleML combines the essence of Prolog's positional and F-logic's slotted syntaxes [11][20].

A pair of inverse translators has been developed for polarized internal transformation under a common GUI: A parser building RuleML-serialization syntax from POSL-shorthand syntax and a generator working in the opposite direction. Currently in Version 1.0, the tool is available online through its "Java Web Start" implementation.[21]

These translators have enabled writing KBs in the POSL shorthand while deploying them in the RuleML/XML serialization, as well as getting RuleML/XML rendered as POSL. Several Hornlog KBs[22] have been built in POSL and serialized with the RuleML←POSL parser, initializing the RuleML knowledge hub.

---

[17] `http://deliberation.ruleml.org/1.02/exa/DatalogPlus/datalogplus_min.ruleml`.

[18] `http://deliberation.ruleml.org/1.02/exa/DatalogPlus/datalogplus_min_normal.ruleml`.

[19] `http://deliberation.ruleml.org/1.02/xslt/compactifier/`.

[20] `http://ruleml.org/submission/ruleml-shortation.html`.

[21] `http://www.jdrew.org/oojdrew/demo.html`.

[22] `http://wiki.ruleml.org/index.php/Rulebases:Master`.

RuleML→PSOA/PS Generator and PSOA/PS→AST Parser The RIF-like presentation syntax of PSOA RuleML (PSOA/PS) is a shorthand that goes beyond RIF/PS by capturing PSOA RuleML's integration of, e.g., relationships and frames [7, 8].

The PSOA RuleML API[23] supports creating and manipulating abstract syntax objects (ASOs) using factory-based Java methods. These are employed to read the XML-based concrete syntax (serialization) of PSOA RuleML into ASOs, and render ASOs as PSOA/PS. This read-render composition, developed with the API, amounts to a generator of the presentation syntax from PSOA RuleML's XML syntax. The PSOA RuleML API is wrapped into an online demo Web application[24], which shows a list of PSOA RuleML/XML rulebases[25] and generates their equivalent forms in the presentation syntax.

For the inverse direction, RuleML←PSOA/PS, the PSOATransRun implementation[26] of PSOA RuleML provides a parser of PSOA/PS into ANTLR abstract syntax trees (ASTs). From there, it generates either TPTP [12] or Prolog but does not currently transform ASTs into RuleML/XML, although this should be easy, since XML trees are structurally similar to the ASTs themselves.

3.3 Importers and Exporters

Importer from Dexlog to Naf Datalog RuleML Dexter [10][27] is a browser-based, domain-independent data explorer for the everyday user.

Dexter among other things allows to create, edit and query tables locally in the browser, to define (integrity and derivation) rules in Dexlog, an extension of Datalog using negation-as-failure, sets, tuples, aggregates, and built-in arithmetic and comparison operators, as well as to import data, and export data and rules.

One Dexter export format (which becomes imported to RuleML) is Naf Datalog RuleML/XML. It was developed in a joint effort by the Stanford Logic Group and RuleML.[28] The JavaScript-implemented translator maps tables and rules from a subset of Dexlog to a subset of Naf Datalog RuleML.

For example, the Dexlog rules (without negation-as-failure etc.)

```
ancestor(X, Y) :- parent(X, Y)
ancestor(X, Y) :- parent(X, Z) & ancestor(Z, Y)
```

are translated to the following rules, valid w.r.t. Naf Datalog RuleML (which can be strengthened to validity w.r.t. RuleML's Datalog and BinDatalog and to their textbook versions – according to Section 2.1 – DatalogBK and BinDatalogBK):

[23] `http://wiki.ruleml.org/index.php/PSOA_RuleML_API`.
[24] `http://psoa-rulemlapi.rhcloud.com/psoaxml2ps/`.
[25] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Test_Cases`.
[26] `http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun`.
[27] `http://dexter.stanford.edu`.
[28] `http://wiki.ruleml.org/index.php/Dexter_and_RuleML`.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://deliberation.ruleml.org/1.01/xsd/nafdatalog.xsd"?>
<RuleML xmlns="http://ruleml.org/spec">
  <Assert>
    <Forall>
      <Var>X</Var>
      <Var>Y</Var>
      <Implies>
        <Atom>
          <Rel>parent</Rel>
          <Var>X</Var>
          <Var>Y</Var>
        </Atom>
        <Atom>
          <Rel>ancestor</Rel>
          <Var>X</Var>
          <Var>Y</Var>
        </Atom>
      </Implies>
    </Forall>
    <Forall>
      <Var>X</Var>
      <Var>Z</Var>
      <Var>Y</Var>
      <Implies>
        <And>
          <Atom>
            <Rel>parent</Rel>
            <Var>X</Var>
            <Var>Z</Var>
          </Atom>
          <Atom>
            <Rel>ancestor</Rel>
            <Var>Z</Var>
            <Var>Y</Var>
          </Atom>
        </And>
        <Atom>
          <Rel>ancestor</Rel>
          <Var>X</Var>
          <Var>Y</Var>
        </Atom>
      </Implies>
    </Forall>
  </Assert>
</RuleML>
```

An inverse translator, from a subset of Naf Datalog RuleML to a subset of Dexlog, could be built with the PSOA RuleML API of Section 3.2, where the generation of PSOA/PS is replaced with Dexlog generation.

Exporters from Datalog$^+$ / Hornlog$^+$ / FOL RuleML to TPTP "Thousands of Problems for Theorem Provers" (TPTP [12][29]) is a widely used syntax and library for Automated Theorem Proving (ATP) test/benchmark problems. RuleML2TPTP[30] is an XSLT 2.0-based translator from Deliberation RuleML/XML 1.01 to TPTP. Originally implemented for Datalog$^+$ RuleML, it was later extended to Hornlog$^+$ RuleML, and then to all of FOL RuleML (with Equality).

RuleML2TPTP first uses the RuleML Normalizer (cf. Section 3.1) to transform Deliberation RuleML/XML to its normalized version. With the fully striped normalization avoiding conditional branching between stripe-skipped and striped forms, the XSLT stylesheet[31] then performs recursive case analysis to linearize XML trees to TPTP texts. The generated TPTP can finally be validated and executed with an ATP system via the "System on TPTP" page[32] such as with Vampire and the E prover.

For instance, the first example's input in Section 3.1 will be normalized to the output shown there. This is then transformed to the following TPTP:

```
fof(example,axiom,(
  ! [H] :
    ( human(H) =>
    ? [M] : hasMother(H,M) ) )).
```

RuleML2TPTP has been used, e.g., to translate the "Instructive KB" for Datalog$^+$ in Section 2.1 to TPTP;[33] also, on an OpenRuleBench-derived RuleML version of the well-known Wine Ontology, generating a TPTP KB for the Semantic Web.[34]

Preparatory planning for an inverse translator, TPTP2RuleML[35], has started, whose implementation is intended as a joint endeavor of the RuleML and TPTP communities.

4 N3-PSOA-Flora Knowledge-Interoperation Use Case

While Section 3 discussed various interoperation test cases under the perspective of the tool-suite taxonomy, the present section proposes a use case bridging the gap between two languages of particular relevance to the rule-based Semantic Web, both also supporting the (light-weight-)ontology-based Semantic Web: N3 [13] and Flora-2/F-logic [14][36]. The N3-PSOA-Flora use case is focusing on the interoperation from N3 to Flora-2/F-logic, although the opposite direction can be easily constructed from the alignment provided. This also demonstrates

[29] http://www.cs.miami.edu/~tptp/.
[30] http://wiki.ruleml.org/index.php/TPTP_RuleML.
[31] https://github.com/RuleML/RuleML2TPTP/archive/v1.02.zip.
[32] http://www.cs.miami.edu/~tptp/cgi-bin/SystemOnTPTP.
[33] http://deliberation.ruleml.org/1.01/exa/DatalogPlus/datalogplus_min/.
[34] http://ruleml.org/usecases/wineonto#Step4:RuleML1.0-RuleML1.01Conversion.
[35] http://wiki.ruleml.org/index.php/TPTP_RuleML#TPTP2RuleML.
[36] With "Flora-2/F-logic" we refer to the current F-logic version as part of Flora-2.

the role of PSOA RuleML [7, 8] as an intermediate (canonical) format that focuses entirely on the knowledge-representation layer rather than programming-language details, but makes syntactic assumptions (e.g. quantifiers) explicit. After having introduced the central rule of this use case in (controlled) English, the rule and a fact will be given as the N3 source, as the Flora-2/F-logic target, and as three variants of the PSOA RuleML canonical form.

English: "If the relation addressRel holds between a name, a street, and a town, then there exists an object, addressObj, with a name slot and a place slot for which there exists an object, placeObj, with a street slot and a town slot."

Source: N3 fact and rule, where the default namespace (N3's ":" prefix) is RuleML's GeospatialRules [9] and `rel:arglist` is an N3 property defined in the PSOA RuleML namespace for an N3 vocabulary that emulates relations:

```
@prefix : <http://psoa.ruleml.org/GeospatialRules#>.
@prefix rel: <http://psoa.ruleml.org/n3/vocab/rel#>.

[a :addressRel;
 rel:arglist ("Computer Science" "Engineering Dr" "Stony Brook, NY 11794")].

{
 [a :addressRel;
  rel:arglist (?Name ?Street ?Town)]
}
=>
{
 [a :addressObj;
  :name ?Name;
  :place [a :placeObj;
          :street ?Street;
          :town ?Town]]
}.
```

Target: Flora-2/F-logic fact and rule, where the compiler option for experts enables the use of the embedded ISA-literal (Flora-2's ":" infix) in the rule head, as described in [14], Section 48:

```
:- compiler_options{expert=on}.

addressRel('Computer Science','Engineering Dr','Stony Brook, NY 11794').

\#(?Name,?Street,?Town):addressObj[
     name->?Name,
     place->\#(?Name,?Street,?Town):placeObj[
       street->?Street,
       town->?Town]] :-
    addressRel(?Name,?Street,?Town).
```

Canonical, presentation syntax: PSOA RuleML/PS fact and rule, where the rule, from [8], uses FOL-style explicit quantifiers (adapted from FOL RuleML/XML as well as W3C RIF/XML and RIF/PS):

```
addressRel("Computer Science" "Engineering Dr" "Stony Brook, NY 11794")

Forall ?Name ?Street ?Town (
  Exists ?O1 ?O2 ( ?O1#addressObj(name->?Name
                                  place->?O2#placeObj(street->?Street
                                                      town->?Town)) )  :-
    addressRel(?Name ?Street ?Town)
                            )
```

Canonical, compact serialization: Stripe-skipped PSOA RuleML/XML for the fact and rule:

```
<Atom>
  <Rel>addressRel</Rel>
  <Data>Computer Science</Data>
  <Data>Engineering Dr</Data>
  <Data>Stony Brook, NY 11794</Data>
</Atom>

<Forall>
  <Var>Name</Var>
  <Var>Street</Var>
  <Var>Town</Var>
  <Implies>
    <Atom>
      <Rel>addressRel</Rel>
      <Var>Name</Var>
      <Var>Street</Var>
      <Var>Town</Var>
    </Atom>
    <Exists>
      <Var>O1</Var>
      <Var>O2</Var>
      <Atom>
        <oid><Var>O1</Var></oid>
        <Rel>addressObj</Rel>
        <slot><Ind>name</Ind><Var>Name</Var></slot>
        <slot>
          <Ind>place</Ind>
          <Atom>
            <oid><Var>O2</Var></oid>
            <Rel>placeObj</Rel>
            <slot><Ind>street</Ind><Var>Street</Var></slot>
            <slot><Ind>town</Ind><Var>Town</Var></slot>
          </Atom>
        </slot>
      </Atom>
    </Exists>
  </Implies>
</Forall>
```

Canonical, normalized serialization:[37] Fully striped PSOA RuleML/XML:

```
<Atom>
  <op><Rel>addressRel</Rel></op>
  <arg><Data>Computer Science</Data></arg>
  <arg><Data>Engineering Dr</Data></arg>
  <arg><Data>Stony Brook, NY 11794</Data></arg>
</Atom>

<Forall>
  <declare><Var>Name</Var></declare>
  <declare><Var>Street</Var></declare>
  <declare><Var>Town</Var></declare>
  <formula>
    <Implies>
      <if>
        <Atom>
          <op><Rel>addressRel</Rel></op>
          <arg><Var>Name</Var></arg>
          <arg><Var>Street</Var></arg>
          <arg><Var>Town</Var></arg>
        </Atom>
      </if>
      <then>
        <Exists>
          <declare><Var>O1</Var></declare>
          <declare><Var>O2</Var></declare>
          <formula>
            <Atom>
              <oid><Var>O1</Var></oid>
              <op><Rel>addressObj</Rel></op>
              <slot><Ind>name</Ind><Var>Name</Var></slot>
              <slot>
                <Ind>place</Ind>
                <Atom>
                  <oid><Var>O2</Var></oid>
                  <op><Rel>placeObj</Rel></op>
                  <slot><Ind>street</Ind><Var>Street</Var></slot>
                  <slot><Ind>town</Ind><Var>Town</Var></slot>
                </Atom>
              </slot>
            </Atom>
          </formula>
        </Exists>
      </then>
    </Implies>
  </formula>
</Forall>
```

[37] Normalization here refers to PSOA RuleML (edge-)stripe reconstruction etc. like in RuleML 1.02, rather than to unnesting using PSOATransRun 1.1 (cf. footnote 9).

The central rule of this use case also clarifies a point – shown in parts A) and B) – that may be surprising to readers new to the rule-based Semantic Web:[38]

A) The rule can (1) be enriched by light-weight-ontological knowledge in the form of taxomic subsumptions – using PSOA's "`##`" infix – such as `addressObj##geoObj` and `placeObj##geoObj` and (2) be employed to align (transform) given facts/instances populating a *relational* address ontology such as the `addressRel` fact/instance from above with (into) derivable facts/instances for populating an *object-centered* address ontology such as the following derivable fact/instance:

```
skolem1#addressObj(name->"Computer Science"
                   place->
                         skolem2#placeObj(street->"Engineering Dr"
                                          town->"Stony Brook, NY 11794"))
```

B) But, following up on Section 3, such a rule, e.g. in the above normalized serialization variant, is itself the subject of interoperation, e.g. using XSLT for transformation. Moreover, the rule-transformation rules, e.g. XSLT templates, could again be interoperated. This could also be done based on the RuleML hub technology by encoding the RuleML rules as RuleML facts[39] and transcribing the XSLT templates into RuleML metarules translating those rule-encoding facts. Since XSLT templates can be conceived as term-rewriting rules over XML trees, this could employ Functional RuleML[40].

5 Conclusions

RuleML has become a hub for the interoperation of formal knowledge by providing a foundational representation layer topped by a transformation layer. Ongoing representation work includes Deliberation RuleML's PSOA, Higher-Order, Modal, and Defeasible subfamilies as well as the further formalization – and transition from XSD to Relax NG – of Reaction RuleML. Novel transformation chains are already emerging from unexpected translator compositions such as between subsets of Dexlog, Datalog RuleML/XML, and TPTP. Future development of the hub should give rise to further interoperation pathways for knowledge sharing and reuse. Readers are invited to consult the links and references about some of the RuleML features and tools not detailed in this article.

6 Acknowledgements

Thanks to my RuleML 1.02 Taskforce colleagues Tara Athan and Adrian Paschke, as well as to Gen Zou, Sadnan Al Manir, Adrian Giurca, Alexandre Riazanov, Michael Genesereth, Sudhir Agarwal, Marcel Ball, Meng Luan, Leah Bidlake, and many others, for their contributions leading to the RuleML hub. Thanks also to Paul Fodor and the entire Organizing Committee chairing RuleML 2016.

[38] The canonical PSOA RuleML format, presentation variant, is employed here, from which the other two formats can be obtained via their alignments.

[39] `http://ruleml.org/indoo/indoo.html#Programs-as-Data`.

[40] `http://ruleml.org/fun/`.

References

1. Calì, A., Gottlob, G., Lukasiewicz, T.: A General Datalog-Based Framework for Tractable Query Answering over Ontologies. Journal of Web Semantics **14** (July 2012) 57–83
2. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: A Highly-Scalable RDF Store. In: Int'l Semantic Web Conference (ISWC 2015), Proceedings, Part II. Volume 9367 of LNCS., Springer (2015) 3–20
3. Boley, H., Grütter, R., Zou, G., Athan, T., Etzold, S.: A Datalog+ RuleML 1.01 Architecture for Rule-Based Data Access in Ecosystem Research. In Bikakis, A., Fodor, P., Roman, D., eds.: Rules on the Web: From Theory to Applications (RuleML 2014) - Proc. 8th Int'l Symposium, Prague, Czech Republic, August 18-20. Volume 8620 of LNCS., Springer (2014) 112–126
4. Hernández-Orallo, J., Muggleton, S.H., Schmid, U., Zorn, B.: Approaches and Applications of Inductive Programming (Dagstuhl Seminar 15442). Dagstuhl Reports **5**(10) (2016) 89–111
5. Athan, T., Boley, H.: Design and Implementation of Highly Modular Schemas for XML: Customization of RuleML in Relax NG. In Olken, F., Palmirani, M., Sottara, D., eds.: Rule-Based Modeling and Computing on the Semantic Web, Proc. 5th Int'l Symposium (RuleML-2011 America), Ft. Lauderdale, FL, Florida, USA. Volume 7018 of LNCS., Springer (November 2011) 17–32
6. Athan, T., Boley, H.: The MYNG 1.01 Suite for Deliberation RuleML 1.01: Taming the Language Lattice. In Patkos, T., Wyner, A., Giurca, A., eds.: Proceedings of the RuleML 2014 Challenge, at the 8th Int'l Web Rule Symposium. Volume 1211., CEUR (August 2014)
7. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: Proc. 5th Int'l Symposium on Rules: Research Based and Industry Focused (RuleML-2011 Europe), Barcelona, Spain. LNCS, Springer (July 2011) 194–211
8. Boley, H.: PSOA RuleML: Integrated Object-Relational Data and Rules. In Faber, W., Paschke, A., eds.: Reasoning Web. Web Logic Rules (RuleML 2015) - 11th Int'l Summer School 2015, Berlin, Germany, July 31- August 4, 2015, Tutorial Lectures. Volume 9203 of LNCS., Springer (2015)
9. Zou, G.: PSOA RuleML Integration of Relational and Object-Centered Geospatial Data. In Bassiliades, N., Fodor, P., Giurca, A., Gottlob, G., Kliegr, T., Nalepa, G.J., Palmirani, M., Paschke, A., Proctor, M., Roman, D., Sadri, F., Stojanovic, N., eds.: Proceedings of the RuleML 2015 Challenge, Berlin, Germany, August 2-5, 2015. Volume 1417 of CEUR Workshop Proceedings., CEUR-WS.org (2015)
10. Agarwal, S., Mohapatra, A., Genesereth, M., Boley, H.: Rule-Based Exploration of Structured Data in the Browser. In: Proc. 9th Int'l Web Rule Symposium (RuleML 2015), Berlin, Germany. LNCS, Springer (August 2015)
11. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. J. of Emerging Technologies in Web Intelligence **4**(2) (November 2010) 343–353
12. Sutcliffe, G.: The TPTP problem library and associated infrastructure. J. Autom. Reasoning **43**(4) (2009) 337–362
13. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A Logical Framework For the World Wide Web. Theory and Practice of Logic Programming (TPLP) **8**(3) (May 2008)
14. Kifer, M., Yang, G., Wan, H., Zhao, C.: $\mathcal{E}$RGO^{Lite} (a.k.a. $\mathcal{F}$lora-2): User's Manual, v1.1 (2015) `http://flora.sourceforge.net/docs/floraManual.pdf`.