Specifying PSOA RuleML/XML 1.03: MYNG-modularized Schemas for the RNC & XSD Validation of XSLT-normalized Data and Knowledge*

Tara Athan[1], Harold Boley[2], Rima Chaudhari[3]

[1] Athan Services, West Lafayette, Indiana, USA
taraathan[AT]gmail[DT]com
[2] Faculty of Computer Science, University of New Brunswick
Fredericton, NB, Canada
harold[DT]boley[AT]unb[DT]ca
[3] CB Unique Solutions, West Lafayette, Indiana, USA
rimachaudhari5681[AT]gmail[DT]com

Abstract. Positional-Slotted Object-Applicative (PSOA) RuleML has been used for graph-relational data and knowledge representation in an executable presentation syntax. In support of interoperation, the XML specifications of Deliberation RuleML Versions 1.02 and 1.03 are extended for Deliberation PSOA RuleML V. 1.03 (PSOA RuleML/XML 1.03). To specify PSOA RuleML's new kinds of atoms (e.g., allowing dependent slots and explicit tuples) and functional expressions, RuleML's MYNG-defined schema system is extended and modified. The PSOA-extended modular Relax NG schemas, in Compact syntax (RNC), are transformed to monolithic XSD schemas by application of the Trang Schema Converter, followed by an XSLT postprocessor previously developed for Deliberation RuleML. Validation of a PSOA rulebase instance can thus be performed w.r.t. RNC, XSD, or dual RNC+XSD schemas. The earlier XSLT-based Deliberation RuleML normalizer is likewise extended for PSOA RuleML. As for all of RuleML, a "normalidation" workflow, where rulebases undergo "first normalization, second validation", prevents false negatives on direct validation against the XSD or RNC+XSD schemas.

1 Introduction

Advanced AI systems need to be based on both data (e.g., variableless facts) and knowledge (e.g., rules), where knowledge can be engineered by humans and/or learned by machines from data. The data and knowledge representation language Positional-Slotted Object-Applicative (PSOA) RuleML [Bol11,BZ19,Bol18] as

* Copyright 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2438/paper7.pdf

realized by the execution system PSOATransRun [ZB16,Zou18] has been employed in various graph-relational use cases, including ATC KB [DMA+19].[1] For this, PSOA RuleML's (plain-text) presentation syntax has mostly been utilized. However, data and knowledge validation, transformation, interchange, and reuse benefit from serialization (XML) syntaxes as provided by RuleML[2]. A translator from versions of PSOA RuleML/XML to PSOA presentation syntax has established a link in one direction [ARBB12][3].

The specification of RuleML itself has usually employed both presentation and serialization syntaxes. Of these, RuleML/XML is normatively specified with Relax NG schemas [AB11] while informative XSD schemas are (Trang+XSLT)-generated for major languages, called "anchor languages", including `hornlog`.

PSOA RuleML Version 1.0[4] was specified with model-theoretic semantics and transformational realizations over an EBNF and – in PSOATransRun – ANTLR-defined presentation syntax, also initializing the XML syntax. Moreover, the specification of RuleML starting with Version 1.02[5] permits the definition of semantic styles, including predefined styles[6], one of which being for Horn-PSOA Tarski[7].

In general, a *RuleML logic* is considered as a (syntactic) language paired with a (semantic) style [Bol16][8]. The PSOA RuleML page thus specifies syntaxes[9] and semantics[10]. In particular, the *PSOA RuleML logic* of interest here is the `hornlogPSOA` language of the PSOA RuleML/XML Version 1.03 family paired with the semantic style Horn-PSOA Tarski.

PSOA RuleML/XML Version 1.03 is specified on the RuleML website[11] – carrying through the sketch in [Bol15], Section 6 – and described in the current paper. PSOA RuleML/XML 1.03 extends the XML specifications of Deliberation RuleML Versions 1.02 (released) and 1.03 (ongoing). PSOA RuleML's extended "psoa atoms" (e.g., allowing dependent slots and explicit tuples) lead to extensions of Deliberation PSOA RuleML/XML Version 1.03 (relative to Deliberation RuleML/XML Version 1.03) focused on new modules that define the new PSOA features (dependent slots and explicit tuples) and modifications of the modules that define serialization-sensitive named patterns[12] for atoms and functional expressions. From there, because of the modular structure of the Relax NG schemas, the PSOA-schema extensions are applied to the rest of the syntac-

[1] `http://wiki.ruleml.org/index.php/PSOA_RuleML`
[2] `http://wiki.ruleml.org/index.php/Introducing_RuleML`
[3] `http://wiki.ruleml.org/index.php/PSOA_RuleML_API`
[4] `http://ruleml.org/talks/PSOAPerspectivalKnowledge-talk.pdf`
[5] `http://deliberation.ruleml.org/1.02`
[6] `http://wiki.ruleml.org/index.php/Predefined_Semantic_Styles_of_RuleML_1.03`
[7] `http://ruleml.org/1.03/profiles/HornPSOA-Tarski`
[8] `http://ruleml.org/talks/RuleMLKnowleropHub-talk.pdf`
[9] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Syntaxes`
[10] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Semantics`
[11] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Modular_Syntax`
[12] `https://relaxng.org/compact-tutorial-20030326.html#id2814516`

tic structures, including to queries and rules, whenever the new PSOA modules are included in the governing driver schema. Existing RuleML languages, whose drivers do not include the PSOA modules, are not affected by these extensions.

This paper describes the current prerelease of PSOA RuleML/XML 1.03[13]as a major addition to RuleML's open-source repository:

- PSOA-enabling changes are made to the existing named patterns in the Relax NG schema modules that define the content models of atoms (Node: Atom) and functional expressions (Nodes: Expr and Plex).
- Four completely new modules are introduced in order to define new elements: the element `slotdep` (to complement existing slot), edge elements `tup` and `tupdep`, as well as a Node element, `Tuple`.
- New anchor languages for PSOA's existing collection[14], as previewed in the left branch of Fig. 1 (which will be explained in Section 5.4):
  - datalogPSOA
  - hornlogPSOA
  - naffologeqPSOA

Next, following this introduction, the paper will give PSOA/XML examples (Section 2). Then, complete instance documents will be exposed in the sequence of "first validation, second normalization"[15] (Section 3). After the preceding two sections' presentation of the "what", the subsequent four sections will proceed to the "how". To provide context, the existing RuleML V. 1.03 schema methodology will be reviewed (Section 4). Then, PSOA changes to the schema system will be explained (Section 5). Continuing the expository sequence, PSOA changes to the normalizer will be discussed (Section 6). Shifting perspective to the level of supporting infrastructure, configuration and test scripts will be discussed (Section 7). The paper will conclude with hints for future work (Section 8). Finally, online validation (Appendix A) and normalization (Appendix B) will be demonstrated.

2 PSOA RuleML Examples

Examples will be given in a "relaxed" XML serialization (containing and merging both "compact" and "normalized" serializations as well as adding positional freedom), where aligned XML comments will show the corresponding presentation syntax.

2.1 Examples with Dependent Slots

Dependent slots as discussed in [BZ19] are employed for making slot names unambiguous by interpreting them under the perspective of a predicate. A use case for dependent slots and other PSOA RuleML features is explained in [DMA+19].

[13] `https://github.com/RuleML/deliberation-ruleml/tree/1.03-psoa`

[14] `http://deliberation.ruleml.org/1.03-psoa/relaxng/#anchor`

[15] This sequence is inverse to the "normalidation" workflow of "first normalization, second validation", which will be explained in Section 4.3.

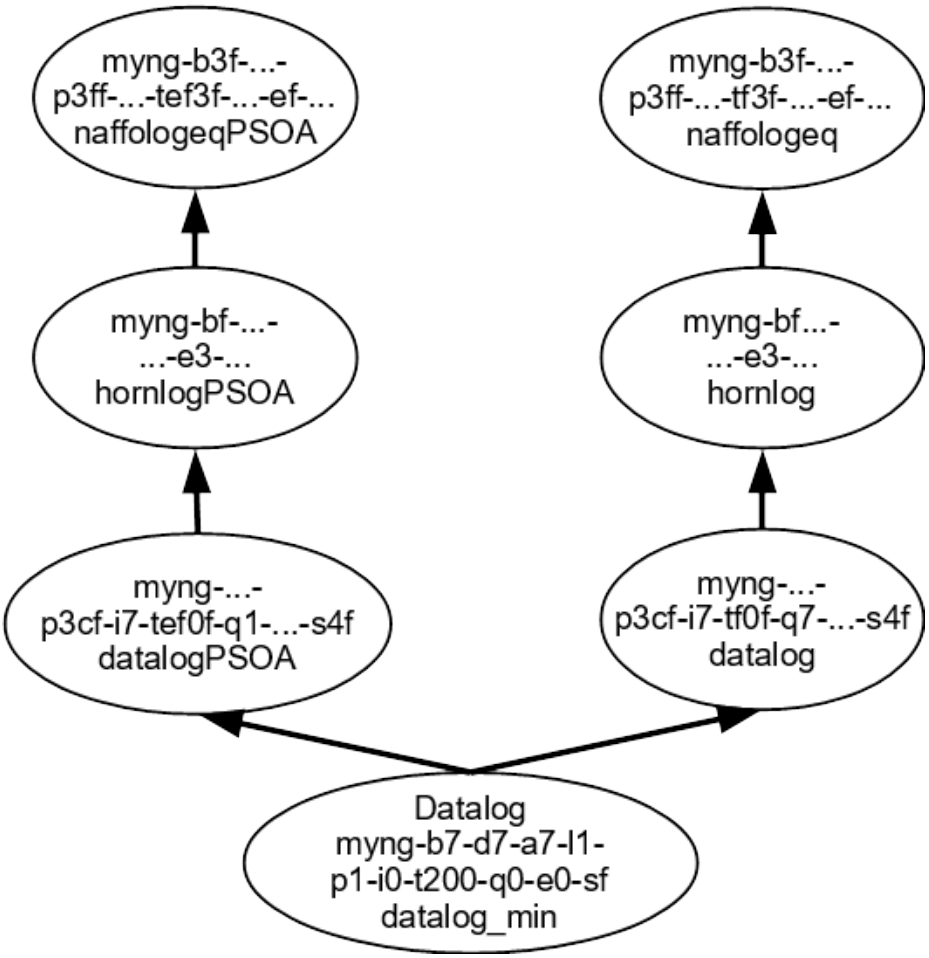

Fig. 1. Hasse diagram for a subset of the RuleML language semilattice with infimum corresponding to the Datalog language (anchor `datalog_min`) and suprema corresponding to the PSOA language (anchor `naffologeqPSOA`) and the non-PSOA language (anchor `naffologeq`). All of the vertices of the depicted semilattice are identified by their myng-codes. The myng-code for Datalog is shown in its entirety, while for other vertices only the components that differ from their nearest sublanguage are shown.

Dependent-Slotted Fact (Reduced to its Atom)

Example S1. Purchase record with dependent slots. An OID transaction200 is typed by the predicate purchase, which is applied to three purchase-dependent slots. A slot name like `item` is thus disambiguated to be the item-of-the-purchase. Because this atom is oidful, slotted, and dependent, according to the metamodel of [BZ19] it is an example of a *pairpoint*.

```
<Atom>
  <oid><Ind>transaction200</Ind></oid>
                                                <!-- transaction200#  -->
  <op><Rel>purchase</Rel></op>
                                                <!--   purchase(      -->
  <slotdep><Ind>buyer</Ind><Ind>John</Ind></slotdep>
                                                <!--     buyer+>John  -->
  <slotdep><Ind>seller</Ind><Ind>Mary</Ind></slotdep>
                                                <!--     seller+>Mary -->
  <slotdep><Ind>item</Ind><Ind>Fido</Ind></slotdep>
                                                <!--     item+>Fido)  -->
</Atom>
```

Dependent-Slotted Rule

Example S2. Liability rule based on purchase record. An independent-slotted version of this was introduced elsewhere[16]. The slot name `item` changes its dependence from the predicate `purchase` (item-of-the-purchase) in the condition to the predicate `liability` (item-of-the-liability) in the conclusion. The slot filler-variable `b` moves from slot name `buyer` in the condition to slot name `bearer` in the conclusion. The rule is applicable to the dependent-slotted fact of Example S1, deducing a liability record (conclusion: `<then>` part) from that purchase record (condition: `<if>` part), where a fresh liability OID, liabilityID(transaction200), is generated as a functional expression from the purchase OID, transaction200.

```
<Forall>
  <Var>b</Var>
  <Var>s</Var>
  <Var>i</Var>
  <Var>t</Var>
  <formula>
    <Implies>
      <if>      <!-- ?t#purchase(buyer+>?b seller+>?s item+>?i) -->
        <Atom>
          <oid><Var>t</Var></oid>
          <op><Rel>purchase</Rel></op>
          <slotdep><Ind>buyer</Ind><Var>b</Var></slotdep>
          <slotdep><Ind>seller</Ind><Var>s</Var></slotdep>
          <slotdep><Ind>item</Ind><Var>i</Var></slotdep>
        </Atom>
      </if>
```

[16] http://wiki.ruleml.org/index.php/PSOA_RuleML#Presentation_Preview

```
      <then>    <!-- liabilityID(?t)#liability(bearer+>?b item+>?i) -->
        <Atom>
          <oid>
            <Expr>
              <op><Fun>liabilityID</Fun></op>
              <arg index="1"><Var>t</Var></arg>
            </Expr>
          </oid>
          <op><Rel>liability</Rel></op>
          <slotdep><Ind>bearer</Ind><Var>b</Var></slotdep>
          <slotdep><Ind>item</Ind><Var>i</Var></slotdep>
        </Atom>
      </then>
    </Implies>
  </formula>
</Forall>
```

2.2 Examples with Explicit Dependent Tuples

While single (predicate-)dependent tuples are widespread, multiple dependent tuples as discussed in [Bol11,BZ19] are novel. While a single tuple can be implicit in its positional argument sequence, multiple tuples must be explicit to separate their argument sequences.

Dependent-Tupled Facts (Reduced to their Atoms)

Example T1. Purchase tables with A. single and B. multiple rows.

A. Single tuple represents single row. Because this atom is oidless, single-tupled, and `de`pendent, according to the metamodel of [BZ19] it is an example of a *relationship* and a special case of what is referred to by systematic name `de1`.

```
<!-- Implicit tuple (already in Deliberation RuleML) -->
<Atom>
  <Rel>purchase</Rel>  <!-- purchase(John Mary Fido) -->
  <Ind>John</Ind>
  <Ind>Mary</Ind>
  <Ind>Fido</Ind>
</Atom>
```

```
<!-- Explicit tuple (added in Deliberation PSOA RuleML) -->
<Atom>
  <Rel>purchase</Rel>  <!-- purchase(+[John Mary Fido]) -->
  <tupdep>
    <Tuple>
      <Ind>John</Ind>
      <Ind>Mary</Ind>
      <Ind>Fido</Ind>
    </Tuple>
  </tupdep>
</Atom>
```

B. Multiple tuples represent multiple rows. Because this atom is oidless, multiple-tupled, and dependent, according to the metamodel of [BZ19] it is a general case of what is referred to by systematic name `de1`.

```
<!-- Explicit tuples (added in Deliberation PSOA RuleML) -->
<Atom>
  <Rel>purchase</Rel>  <!-- purchase(                    -->
  <tupdep>             <!--          +[John Mary Fido]   -->
    <Tuple>
      <Ind>John</Ind>
      <Ind>Mary</Ind>
      <Ind>Fido</Ind>
    </Tuple>
  </tupdep>
  <tupdep>             <!--          +[Pedro Amy Tabby]  -->
    <Tuple>
      <Ind>Pedro</Ind>
      <Ind>Amy</Ind>
      <Ind>Tabby</Ind>
    </Tuple>
  </tupdep>
  <tupdep>             <!--          +[Mary John Coco]   -->
    <Tuple>
      <Ind>Mary</Ind>
      <Ind>John</Ind>
      <Ind>Coco</Ind>
    </Tuple>
  </tupdep>
  <tupdep>             <!--          +[Amy Pedro Tabby]) -->
    <Tuple>
      <Ind>Amy</Ind>
      <Ind>Pedro</Ind>
      <Ind>Tabby</Ind>
    </Tuple>
  </tupdep>
</Atom>
```

Dependent-Tupled Rules

Example T2. Liability rule based on single purchase tuple at a time. This was introduced elsewhere[17]. The rule is applicable to the dependent-tupled facts of Example T1, deducing a liability tuple for each of A's and B's purchase tuples.

Single row:

```
<Forall>
  <Var>b</Var>
  <Var>s</Var>
  <Var>i</Var>
```

[17] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Serialization_Preview`

```
  <formula>
    <Implies>
      <if>
        <Atom>
          <Rel>purchase</Rel>        <!-- purchase(+[?b ?s ?i]) -->
          <tupdep>
            <Tuple>
              <Var>b</Var>
              <Var>s</Var>
              <Var>i</Var>
            </Tuple>
          </tupdep>
        </Atom>
      </if>
      <then>
        <Atom>
          <Rel>liability</Rel>       <!-- liability(+[?b ?i]) -->
          <tupdep>
            <Tuple>
              <Var>b</Var>
              <Var>i</Var>
            </Tuple>
          </tupdep>
        </Atom>
      </then>
    </Implies>
  </formula>
</Forall>
```

Example T3. Cyclic-purchase rule based on one or more purchase tuples. The condition (first child of `Implies`)[18] detects a tuple, anywhere amongst the tuples, for which there also is, again anywhere, an 'inverse' tuple (where the buyer x and seller y are swapped to y and x while the item i is kept the same). A special case of this cyclicity condition is x = y (someone purchasing from themselves). The conclusion (second child of `Implies`)[18] alerts about such a cyclic purchase by showing the tuple of the involved buyer, seller and item, hence provides more information than the 'falsity' conclusion of a corresponding integrity rule would. The rule is applicable, in two possible ways, to the dependent-tupled fact of Example T1.B, deducing two alert tuples: `+[Pedro Amy Tabby]` and `+[Amy Pedro Tabby]`.

Multiple rows (in condition atom):

```
<Forall xmlns="http://ruleml.org/spec">
  <Var>x</Var>
  <Var>y</Var>
  <Var>i</Var>
```

[18] The relaxed serialization allows the `if` and `then` tags to be skipped.

```
  <Implies>
    <Atom>
      <Rel>purchase</Rel>        <!-- purchase(             -->
      <tupdep>                   <!--          +[?x ?y ?i]  -->
        <Tuple>
          <Var>x</Var>
          <Var>y</Var>
          <Var>i</Var>
        </Tuple>
      </tupdep>
      <tupdep>                   <!--          +[?y ?x ?i]) -->
        <Tuple>
          <Var>y</Var>
          <Var>x</Var>
          <Var>i</Var>
        </Tuple>
      </tupdep>
    </Atom>
    <Atom>
      <Rel>cyclic-purchase</Rel> <!-- cyclic-purchase(             -->
      <tupdep>                   <!--                 +[?x ?y ?i]) -->
        <Tuple>
          <Var>x</Var>
          <Var>y</Var>
          <Var>i</Var>
        </Tuple>
      </tupdep>
    </Atom>
  </Implies>
</Forall>
```

3 Validating and Normalizing Instances

Based on examples of the previous section, the current section discusses characteristic XML processing of instance documents, which for RuleML are rulebases (including facts): their validation w.r.t. schemas and their XML-to-XML transformation (here, normalization).

3.1 Instance Validation

Deliberation RuleML's RNC- & XSD-based validation checks whether instances conform to its RNC- & XSD-schema definition.[19] This has been extended for PSOA RuleML's additional features of dependent slots and explicit tuples, as will be explained in Section 5.

For example, the dependent-slotted fact in Section 2.1 can be completed to a PSOA RuleML instance document with `<RuleML>` and `<Assert>` wrappers:

[19] `http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.03#Appendix_3:_Validating_a_RuleML_Instance_Against_a_Relax_NG_Schema`

```
<RuleML xmlns="http://ruleml.org/spec">
  <Assert>
    <Atom>
      <oid><Ind>transaction200</Ind></oid>
                                                <!-- transaction200#  -->
      <op><Rel>purchase</Rel></op>
                                                <!--   purchase(      -->
      <slotdep><Ind>buyer</Ind><Ind>John</Ind></slotdep>
                                                <!--     buyer+>John  -->
      . . .
    </Atom>
  </Assert>
</RuleML>
```

This instance is valid w.r.t. the RNC & XSD schemas. An instance with, e.g., one instead of two `<slotdep>` children such as `<slotdep><Ind>buyer</Ind> </slotdep>`, is invalid. Readers are encouraged to try these and other instances online (cf. Appendix A).

3.2 Instance Normalization

Deliberation RuleML's XSLT-based normalizer (see Section 4.3) fills in skipped edges and sorts elements into a canonical order.[20] Again, this has been extended for PSOA RuleML's additional features, as will be explained in Section 6.

Much of the PSOA RuleML normalizer's functionality can be seen for Example T3 of Section 2.2, which becomes transformed thus:[21]

```
<Forall xmlns="http://ruleml.org/spec">
  <ruleml:declare xmlns:ruleml="http://ruleml.org/spec">
    <Var>x</Var>
  </ruleml:declare>
  <ruleml:declare xmlns:ruleml="http://ruleml.org/spec">
    <Var>y</Var>
  </ruleml:declare>
  <ruleml:declare xmlns:ruleml="http://ruleml.org/spec">
    <Var>i</Var>
  </ruleml:declare>
  <ruleml:formula xmlns:ruleml="http://ruleml.org/spec">
    <Implies>
      <ruleml:if>
        <Atom>
          <!-- purchase(             -->
          <ruleml:op><Rel>purchase</Rel></ruleml:op>
```

[20] `http://wiki.ruleml.org/index.php/Specification_of_Deliberation_RuleML_1.03#XSLT-Based_Normalizer`

[21] We reproduce the original output of our current XSLT normalizer, which preserves input tags, e.g. `<tupdep>`, and creates new tags, e.g. `<ruleml:if>`, with the prefix `ruleml`. The normalizer adds a namespace declaration `xmlns:ruleml="http://ruleml.org/spec"` to new tags when needed.

```
          <tupdep>
            <!--          +[?x ?y ?i]  -->
            <Tuple>
              <ruleml:arg index="1"><Var>x</Var></ruleml:arg>
              <ruleml:arg index="2"><Var>y</Var></ruleml:arg>
              <ruleml:arg index="3"><Var>i</Var></ruleml:arg>
            </Tuple>
          </tupdep>
          <tupdep>
            <!--          +[?y ?x ?i]) -->
            <Tuple>
              <ruleml:arg index="1"><Var>y</Var></ruleml:arg>
              <ruleml:arg index="2"><Var>x</Var></ruleml:arg>
              <ruleml:arg index="3"><Var>i</Var></ruleml:arg>
            </Tuple>
          </tupdep>
        </Atom>
      </ruleml:if>
      <ruleml:then>
        <Atom>
          <!-- cyclic-purchase(             -->
          <ruleml:op><Rel>cyclic-purchase</Rel></ruleml:op>
          <tupdep>
            <!--                 +[?x ?y ?i]) -->
            <Tuple>
              <ruleml:arg index="1"><Var>x</Var></ruleml:arg>
              <ruleml:arg index="2"><Var>y</Var></ruleml:arg>
              <ruleml:arg index="3"><Var>i</Var></ruleml:arg>
            </Tuple>
          </tupdep>
        </Atom>
      </ruleml:then>
    </Implies>
  </ruleml:formula>
</Forall>
```

An equivalent normal form is obtained from the various non-normal forms of Example T3, e.g. with only some of the edges skipped. Readers are again encouraged to try these and other instances online (cf. Appendix B).

4 Schema Methodology

In this section, we review the RuleML schema methodology that was developed, primarily, for RuleML V. 1.0 [AB11][22]. In the subsequent Versions 1.01 [AB14], 1.02 [ABP15] and 1.03[23] of Deliberation RuleML, the schema modularization system has been extended and enhanced, while upholding the original design

[22] http://deliberation.ruleml.org/1.0/
[23] http://deliberation.ruleml.org/1.03/

principles. The following subsections described the aspects of RuleML's schema system relevant to the new PSOA/RuleML XML extension, including modularization design principles, and serialization, normalization, initialization, pivot-schema, customization, and test-automation methodologies.

4.1 Modularization Design Principles

Beginning with Version 1.0, Deliberation RuleML introduced a modularization approach, based on a restriction of Relax NG [AB11], whereby a RuleML syntax is defined by a set of RNC module inclusions in a driver schema. The restricted Relax NG is **monotonic**, meaning when two drivers are combined, i.e. by forming the union of the module inclusions in a larger driver, the syntax defined by the larger driver contains both of the smaller syntaxes [AB14]. Because of this monotonicity property, the more than one hundred Deliberation RuleML schema modules[24] may be freely (orthogonally) combined to define a fine-grained poset lattice of RuleML syntaxes, with a partial order based on syntactic containment.

As may be seen in the schema snippets in Section 5, the RuleML RNC schema employs element definition conventions consisting of several layers of named patterns, in order to optimize schema extensibility.

In summary, the main design principles of the RuleML schema modularization system are:

- Monotonicity
- Orthogonality
- Extensibility

4.2 Serialization Methodology

There are four serializations of Deliberation RuleML – normalized, relaxed, mixed, and compact.

The "normalized" serialization requires all content models to have a canonical ordering of their child elements. For example, the canonical ordering for atoms requires positional arguments to precede slots, should both occur in the same atom. The canonical ordering of the contents of functional expressions is maintained to be identical to that of atoms.

The "relaxed" serialization allows significantly more positional freedom in content models. It also allows certain edges to be skipped, provided they can be reconstructed unambiguously.

In general, the RNC patterns for the relaxed serialization content models are similar to that of the corresponding normalized serialization patterns after substitution of the interleave symbol '`&`' for commas in the definition of sequences. RuleML takes advantage of this convenient RNC mechanism for implementing positional freedom in content models to define the relaxed serialization. However, the interleave pattern cannot be accurately translated into XSD, due to the

[24] `http://deliberation.ruleml.org/1.03/relaxng/modules/`

XSD requirement of deterministic models[25]. The "mixed" serialization (of some language) is constructed to be the greatest subset of the relaxed serialization (of that language) with the most positional freedom that can still be defined within XSD (see Section 4.5).

The "compact" serialization is similar to the normalized one in that it requires the same canonical ordering of elements. It differs from the normalized one in that it requires all edges skippable in the relaxed serialization to be skipped.

Elements whose content models have some positional freedom among their children in the relaxed and mixed serializations, including atomic formulas (Atom) and functional expressions (Expr and Plex), have their content models for the four serializations specified in three RNC serialization modules[26] [27] [28], resp. The main exception to the RuleML schema design principle of orthogonality occurs in regard to serialization: a (valid) driver schema may have only one serialization module inclusion.

4.3 Normalization Methodology

The goals of the RuleML normalizer include the following:

- Reconstruct all skipped edge tags to produce a fully striped form (since edge tags correspond to Resource Description Framework (RDF)[29] properties, this simplifies interoperation between RuleML/XML and, e.g., 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)

We say that using this normalizer followed by schema-based RuleML validation performs "normalidation" on RuleML instances. That is, not only are missing edge elements inserted and sibling elements sorted in the XSLT output, but this transformed instance is subsequently validated syntactically to ensure that Node and edge elements only appear in correct positions, and with permitted content. Normalidation is required when using the XSD schemas for validation of general RuleML instances, because the positional freedom allowed by the relaxed serialization of the Relax NG schemas is beyond the expressivity of XSD. This is to prevent valid instances being deemed as invalid ("false negatives").

4.4 Initialization Methodology

Another exception to the orthogonality principle is required in order to support the development and implementation of the (otherwise) orthogonal system: the

[25] `http://books.xmlschemata.org/relaxng/relax-CHP-16-SECT-2.html`

[26] Normalized and compact: `http://deliberation.ruleml.org/1.03/relaxng/modules/ordered_groups_expansion_module.rnc`

[27] Relaxed: `http://deliberation.ruleml.org/1.03/relaxng/modules/unordered_groups_expansion_module.rnc`

[28] Mixed: `http://deliberation.ruleml.org/1.03/relaxng/modules/unordered_deterministic_groups_expansion_module.rnc`

inclusion of the initialization module[29], containing vacuous (either empty or not allowed) definitions of many named patterns, is required in all (valid) drivers so that named patterns that are used in an included module but defined in an excluded module are never left undefined in the inclusion closure of the driver. In particular, singleton inclusion sets (a driver containing only one module inclusion) are not valid, in general, within the RuleML modularization system. Such drivers become valid upon inclusion of the initialization module, enabling fine-grained validation during development as well as ensuring orthogonality in implementation.

4.5 Pivot-schema Methodology

The RNC schemas act as normative "pivot schemas"[30] that define the XSD schemas for Deliberation RuleML languages by translation using Trang[31] followed by XSLT post-processing [AB11]. The XSD schema for the normalized serialization of a Deliberation RuleML language is obtained in a straightforward manner by this translation mechanism. In order to define a 'more relaxed' XSD schema having as much positional freedom in content models as possible, while still remaining deterministic, the 'mixed' serialization is used. The Relax NG for the mixed serialization implements this trade-off through involved regular expressions.

4.6 Customization Methodology

In order to manage the large number of RuleML syntaxes generated by the above modularization approach, the Modular sYNtax confiGurator (MYNG) application [Ath11] was developed to provide a unified parameterized RNC schema accessible either directly, using a REST interface implemented in PHP, or through a GUI[32] that exposes the REST interface. The parameters which specify directly the features of the syntax, and indirectly the module inclusion set needed to define those features, are efficiently captured in a string called the "myng-code". The myng-code is partitioned into sets of features called "option facets". Each feature of a facet is assigned a unique bit position, from zero up, where a digit of 1 in that position indicates the feature is selected and 0 otherwise. Although myng-codes are interpreted according to their binary representation, they are expressed and transmitted in hexadecimal for brevity.

RNC and XSD schemas for RuleML anchor languages are obtained from human-readable URLs via URL redirects to the MYNG PHP engine with the specifying myng-code parameters expressed in the query string. The redirects are implemented through a `.htaccess` file that is auto-generated by a configuration bash script.

[29] `http://deliberation.ruleml.org/1.03/relaxng/modules/init_expansion_module.rnc`
[30] `http://books.xmlschemata.org/relaxng/relax-CHP-1-SECT-6.html`
[31] `https://relaxng.org/jclark/trang.html`
[32] `http://deliberation.ruleml.org/1.03/myng`

4.7 Test-automation Methodology

RuleML's automated testing procedures include Jing validation of RNC modules and driver schemas, as well as JAXB[33] validation of XSD schemas. Cross-testing of the schemas is carried out by auto-generating XML instances from the XSD schemas and validating them against the RNC schemas.

5 PSOA Enhancements of the Schema System

The modifications necessary to implement the new PSOA/XML syntax, described below, are guided by the structure and principles of the RuleML schema system as reviewed in Section 4. Validation of PSOA RuleML instances against PSOA schemas is illustrated in Appendix A.

5.1 Dependent Slots

The PSOA feature of dependent slots in atoms and functional expressions is implemented, in part, by means of the introduction into the modular RNC schema system of one new module[34].

The new module defining the dependent slot element is essentially identical to the existing module defining the independent slot element[35] after substitution of `slotdep` for most occurrences of `slot`. An RNC snippet of the module is shown below.

```
slotdep-edge.choice |= slotdep.edge.def
slotdep.edge.def =

  ## A user-defined dependent slot (dependent property)...
  element slotdep { slotdep.type.def }
slotdep.type.def = (slotdep-datt.choice & reslotdep.attlist),
                                                        slotdep.content
...
## The slotdep content model consists of a key (first position) and
                                              a filler (second position).
slotdep.content |= slotdepKeyTerm.choice, slotdepFillerTerm.choice

## The key (first position) in a slotdep contains an interpreted term or
                                    data, which may be simple or compound.
slotdepKeyTerm.choice |= SimpleKeyTerm.choice | CompoundTerm.choice
.  .  .
## The filler (second position) in a slotdep contains any single term.
slotdepFillerTerm.choice |= AnyTerm.choice
```

[33] https://github.com/eclipse-ee4j/jaxb-ri

[34] http://deliberation.ruleml.org/1.03-psoa/relaxng/modules/slotdep_expansion_module.rnc

[35] http://deliberation.ruleml.org/1.03-psoa/relaxng/modules/slot_expansion_module.rnc

```
.  .  .
edge.choice |= slotdep.edge.def
...
```

5.2 Explicit Tuples

The PSOA feature of explicit tuples in atoms and functional expressions is partially implemented in the modular RNC schema system by means of the introduction of three new modules[36] [37] [38] . RNC snippets from these modules are shown below.

From `tuple_expansion_module.rnc`

```
Tuple-node.choice |= Tuple.Node.def
## The children of a generalized list (tuple) are divided into two
                                                                  sections,
## a header section for modifiers,
                               and a main section for the list arguments.
Tuple.Node.def =

  ## A collection of (ordered) arguments (optionally enclosed by <arg>)
  and/or (unordered) user-defined slots (<slot>),
  ## identical to an uninterpreted expression (<Expr in="no">)
             except not having a user-specified function name (equivalent
  ## to having a system-specified constructor 'Tuple').
               Rest variables (<repo> and <resl>) are also permitted. ...
  ## within repo
  element Tuple { Tuple.type.def }
Tuple.type.def = Tuple.attlist, Tuple.header, Tuple.main

Tuple.attlist &= commonNode.attlist?

## Generalized lists accept the header pattern common to Nodes.
Tuple.header &= Node.header?
# For the declaration of the Node header,
                                  see the modules meta_expansion_module).

## A generalized list within a positional rest variable
                                  contains a positional argument sequence
Tuple.main |= positionalArgumentsForAtoms.sequence
```

From `tupdep_expansion_module.rnc`

---

[36] `http://deliberation.ruleml.org/1.03-psoa/relaxng/modules/tuple_expansion_module.rnc`

[37] `http://deliberation.ruleml.org/1.03-psoa/relaxng/modules/tupdep_expansion_module.rnc`

[38] `http://deliberation.ruleml.org/1.03-psoa/relaxng/modules/tup_expansion_module.rnc`

```
## In atomic formulas, zero or more tupdep are allowed.
tupdepTerms.nonemptysequence.choice |= tupdep-edge.choice+
tupdepTerms.sequence |= tupdepTerms.nonemptysequence.choice?
tupdepTermsForAtoms.sequence |= tupdepTerms.sequence

## In expressions, zero or more tupdep are allowed.
tupdepTermsForExpressions.sequence |= tupdepTerms.sequence

## an extension point for specializations of the tupdep tag name.
tupdep-edge.choice |= tupdep.edge.def
tupdep.edge.def =

    ## A user-defined tupdep (property)...
    element tupdep { tupdep.type.def }
tupdep.type.def = tupdep.attlist? & tupdep.content

## The tupdep content model consists of a Tuple.
tupdep.content |= Tuple-node.choice
.  .  .
tupdep.attlist &= commonInit.attlist?
```

The `tup_expansion_module.rnc` module is similar to that of the already discussed `tupdep_expansion_module.rnc`.

5.3 Integration of New Modules with Serializations

For the canoncial ordering of the normalized serialization, an XML realization of the **left-tuple, left-dependent canonical ordering** is used that was given in [BZ19] for the PSOA presentation syntax. One reason for preferring this ordering is that the basic atoms of *relationships*, which are single-tuple, should be captured such that their arguments do not need to be moved when more descriptors are added.

The RNC snippet specifying this canonical ordering for PSOA atomic formulas is as follows:

```
## The main section of an atomic formula contains an operator edge and
                                          a collection of argument edges.
## Prefix operator notation for atomic formulas is required in
                                            the normalized serialization.
Atom.main |= operatorForAtoms-edge.choice, argumentsForAtoms.sequence

## Slotted arguments follow positional arguments in atomic formulas
## in the normalized serialization.
argumentsForAtoms.sequence |=
    (positionalArgumentsForAtoms.sequence | tuplesForAtoms.sequence),
                                                    slotsForAtoms.sequence

## Positional rest arguments follow the ordinary positional arguments
## in atomic formulas in the normalized serialization.
```

```
positionalArgumentsForAtoms.sequence |=
    termsForAtoms.sequence, restOfPositionalArguments-edge.choice?
...
```

In support of the orthogonality principle, the new named patterns introduced for a sequence of dependent slots are initialized (see Section 4.4) as zero-length sequences to accommodate non-PSOA languages having no dependent slots. The RNC snippet from the initialization module that accomplishes this is as follows:

```
slotdepTermsForAtoms.sequence |= empty
slotdepTermsForExpressions.sequence |= empty
```

The RNC for the relaxed serialization is similar to that of the normalized serialization shown above after substitution of the interleave symbol '`&`' for commas in the definition of sequences, as described in Section 4.2.

The XSDs for PSOA RuleML anchor languages are auto-generated as described in Section 4.5.

5.4 Integration of New Modules with the MYNG Engine

The myng-code (specifying Deliberation RuleML syntaxes, see Section 4.6) is extended to accommodate the PSOA/XML features. Three unused bits (binary positions 13 through 15) of the `terms` option facet are utilized for the features of dependent slots, dependent tuples, and independent tuples, respectively.

The inclusion of the `slotdep_expansion_module.rnc` module in a myng-code-specified driver schema occurs whenever its myng-code (expanded from hexadecimal to binary) has 1 as the binary digit in position 13 (2^{13}) of the `terms` option facet. Similarly, the `tupdep_expansion_module.rnc` and `tup_expansion_module.rnc` modules are associated with positions 14 and 15 of this facet. The inclusion of the `tuple_expansion_module.rnc` module is required if either `tupdep_expansion_module.rnc` or `tup_expansion_module.rnc` are included, since both edges are required to contain a `Tuple` element – thus, it is not an orthogonal feature.

Consequently, myng-codes for RuleML languages having full PSOA capability have the value 'e' or 'f' in hexadecimal position three (16^3) of the `terms` option facet, e.g. `bf-d7-a7-l1-p3cf-i7-tef0f-q1-e3-s4c` for `hornlogPSOA_normal`.

A portion of the PSOA-extended RuleML anchor language semilattice depicting the new PSOA anchor languages and their nearest non-PSOA counterparts is shown in Fig. 1. The first row of vertices above the infimum corresponds to additional features of RuleML's `datalog` and `datalogPSOA` languages that are not part of textbook Datalog, differing from each other only with respect to the PSOA features (`tef0f` vs. `tf0f`) and rest variables (`q1` vs. `q7`). The vertices in the second row above the infimum correspond to the additional feature characteristic of Horn logic, i.e. functional expressions (`bf` for `Expr` and `e3` for `Plex`). The third row of vertices above the infimum corresponds to additional features characteristic of First-Order Logic (`b3f`), as well as weak negation (`p3ff`), equations (`tef3f`), and set-valued and interpreted expressions (`ef`).

The following PHP snippet shows the modifications to the MYNG engine which implement the extension of the myng-code for PSOA features.

```
$terms_slotdep = 13;
$terms_tupdep = 14;
$terms_tup = 15;
.  .  .
$needTupDep = extractBit($bterms, $terms_tupdep);
$needTup = extractBit($bterms, $terms_tup);
$needTuple = max($needTupDep, $needTup);
$needSlotDep = extractBit($bterms, $terms_slotdep);
.  .  .
    // Include dependent slots
    if ($needSlotDep){
      echo "#\n# DEPENDENT SLOTS INCLUDED\n";
      echo "#\n".'include "' . $modulesLocation .
          'slotdep_expansion_module.rnc"'."$end\n";
.  .  .
 // Include dependent tuple edges if needed
    if ($needTupDep){
      echo "#\n# DEPENDENT TUPLE EDGES INCLUDED\n";
      echo "#\n".'include "' . $modulesLocation .
          'tupdep_expansion_module.rnc"'."$end\n";
    }
    // Include independent tuple edges if needed
    if ($needTup){
      echo "#\n# INDEPENDENT TUPLE EDGES INCLUDED\n";
      echo "#\n".'include "' . $modulesLocation .
          'tup_expansion_module.rnc"'."$end\n";
    }
    // Include tuple nodes if needed
    if ($needTuple){
      echo "#\n# TUPLE NODES INCLUDED\n";
      echo "#\n".'include "' . $modulesLocation .
          'tuple_expansion_module.rnc"'."$end\n";
    }
```

6 PSOA Enhancements of the Normalizer

The normalizer methodology reviewed in Section 4.3 is enhanced for PSOA RuleML. Transformation employing this normalizer is illustrated in Appendix B. The modified normalizer[39] inserts skipped stripes within tuples as follows.

```
<xsl:template
  match="ruleml:Tuple/*[ruleml:isNode(.)]"
  mode="phase-1">
    <xsl:call-template name="wrap">
```

[39] http://deliberation.ruleml.org/1.03-psoa/xslt/normalizer/normalizer.xslt

```
      <xsl:with-param name="tag">arg</xsl:with-param>
    </xsl:call-template>
</xsl:template>
```

The normalizer is further modified to canonically order slots and tuples in atoms as follows.

```
<xsl:template match="ruleml:Atom" mode="phase-2">
  <xsl:copy>
  ...
  <xsl:apply-templates select="ruleml:oid" mode="phase-2"/>
  ...
  <xsl:apply-templates select="ruleml:op" mode="phase-2"/>
  ...
  <xsl:apply-templates select="ruleml:tupdep" mode="phase-2"/>
  <xsl:apply-templates select="ruleml:tup" mode="phase-2"/>
  ...
  <xsl:apply-templates select="ruleml:slotdep" mode="phase-2"/>
  <xsl:apply-templates select="ruleml:slot" mode="phase-2"/>
  ...
  </xsl:copy>
</xsl:template>
```

7 Scripts for Configuration and Testing

Updates and improvements have been made to RuleML's bash scripts[40] for (anchor-language) configuration and (XSD-schema) testing.

7.1 Updates to Configuration Scripts

The modified configuration text files used by the configuration scripts (in bash) implement the URL redirects for the new PSOA anchor languages. In the configuration file snippet below, the second and third columns contain the facet portion of the myng-code for the normalized and relaxed serializations, respectively, for the anchor language stem in the first column.

```
datalogPSOA      b07-d7-a7-l1-p3cf-i07-tef0f-q1-e0-s4c
                              b07-d7-a7-l1-p3cf-i07-tef0f-q1-e0-s4f
hornlogPSOA      b0f-d7-a7-l1-p3cf-i07-tef0f-q1-e3-s4c
                              b0f-d7-a7-l1-p3cf-i07-tef0f-q1-e3-s4f
naffologeqPSOA   b3f-d7-a7-l1-p3ff-i7f-tef3f-q1-ef-s4c
                              b3f-d7-a7-l1-p3ff-i7f-tef3f-q1-ef-s4f
```

For example, execution of the configuration bash script will add a redirect to the `.htaccess` file that redirects a request for `http://deliberation.ruleml.org/1.03-psoa/relaxng/hornlogPSOA_normal.rnc` to `http://deliberation.ruleml.org/1.03-psoa/relaxng/myng-b0f-d7-a7-l1-p3cf-i07-tef0f-q1-e3-s4c.rnc` Similar redirects have been implemented for XSD schemas and simplified RNC schemas.

[40] `https://github.com/RuleML/deliberation-ruleml/blob/1.03-psoa/bash`

7.2 Improvements to the Suite of Test Scripts

Among other tests, RuleML testing procedures have always performed JAXB validation of XSD schemas. However, upon detailed examination of the output from JAXB validation of RuleML XSD schemas during the development of the PSOA extension, it has been determined that only the schemas for the normalized serialization produce effective Java classes, even though the JAXB validation of the mixed serialization XSD schemas do not generate validation errors. Therefore, the test scripts have been modified to perform JAXB validation only on the XSD schemas for the normalized serialization. The mixed serialization XSD schemas are instead validated manually using Xerces[41].

In the PSOA relaxed serializations, it is allowed to reverse the canonical order of `slotdep` with `slot` or `tupdep` with `tup`. However, this aspect of positional freedom is not allowed in the mixed serialization (the most permissive serialization used in the XSD schemas, see Section 4.2). Therefore, the extended test suite applies the XSLT normalizer to the RNC test suites, which include such instances in the relaxed serialization.

8 Conclusions and Future Work

In conclusion, this paper connects PSOA RuleML work based on its presentation syntax with Deliberation RuleML work based on its serialization syntax. The paper describes the prerelease of PSOA RuleML/XML 1.03 for the representation and processing of data (facts) and knowledge (rules). Deliberation RuleML's syntactic definition via a MYNG-defined schema system, as developed in V. 1.0 and enhanced in subsequent versions through V. 1.03, is further extended and modified to accommodate PSOA RuleML. The focus is on the validation as well as normalization of rulebases utilizing the new PSOA features of dependent slots and explicit tuples.

Future work[42] encompasses the completion of the PSOA RuleML/XML 1.03 release[43].

This includes syntactic completion, e.g. by incorporating XML elements for `Subclass` (‘`##`’), which was already used in other RuleML languages, e.g. for SWSL, and `Equal` (‘`=`’), which was already used in several Deliberation RuleML languages, e.g. in `hornlogeq`.

Conversely, while OIDs, e.g. for functional expressions, already are optional only, they could be entirely prohibited for expressions in a `hornlogPSOA`-restricting anchor language that defines a more precise, `Expr`-oidless and `Plex`less syntax for the semantic style Horn-PSOA Tarski realized by PSOATransRun.

[41] `https://xerces.apache.org`

[42] Also see the PSOATransRun work overlap: `http://wiki.ruleml.org/index.php/PSOATransRun_Development_Agenda`

[43] See here for keeping track of the release: `http://wiki.ruleml.org/index.php/Timeline_of_Deliberation_RuleML_1.03-psoa`

Similarly, while the "extended PSOA" introduced in [Bol15], Section 6, allowing a "positional rest" within a tuple, has been adopted from Deliberation RuleML's schemas as an option by PSOA RuleML V. 1.03, it could be entirely prohibited in "pure PSOA" (anchor) languages.

Moreover, in a merged Deliberation PSOA RuleML, a PSOA-derived semantic style for built-in "slotribution" [Bol11] could be alternated with a Deliberation-derived semantic style for an explicit "rest of slots" [Bol15].

Completion of the release also includes technical completion, e.g.:

- Update the MYNG GUI (in Javascript) to accommodate PSOA features that are already implemented in the MYNG PHP engine.
- Extend for PSOA the V. 1.03 compact serialization and associated transformer (compactifier).
- Automate the PSOA validation of mixed-serialization XSD schemas.
- Implement PSOA features as syntactic options within the RuleML Holog syntax. This is particularly important for restoring a single supremum to the Deliberation RuleML syntactic structure in a release merging Deliberation RuleML V. 1.03 and PSOA RuleML V. 1.03.

As the abstract inverse of the translator from PSOA RuleML/XML to PSOA presentation syntax [ARBB12], a translator from PSOA presentation syntax to PSOA RuleML/XML should be written. This could be layered on PSOATransRun's ANTLR-based parser for PSOA presentation syntax into Abstract Syntax Trees (ASTs) [ZPPBR12], adding an AST tree walker that generates PSOA RuleML/XML.

The results of the current work on PSOA-extended Deliberation RuleML should be transferred to Reaction RuleML and Consumer RuleML [ABP15], which could then target a further developed PSOA Prova [GBP18] engine.

Support for (PSOA) RuleML/XML editing, validation, and transformation could be enhanced through the development of plugins or frameworks in professional tools, e.g. the Oxygen XML Editor suite[44], which is aware of Relax NG as well as XSD.

Strengthening an early connection, it would be interesting to compare PSOA RuleML logics with N3Logic [BLCK$^+$08], building, e.g., on the PhD work of Gen Zou [Zou18], Chapter 6, and of Dörthe Arndt [ASDRV19] as well as work by the W3C Notation 3 (N3) Community Group[45]. This can lead to the definition of a new RuleML logic customized for a revised N3.

[44] `https://www.oxygenxml.com`
[45] `https://www.w3.org/community/n3-dev/`

References

AB11. Tara Athan and Harold Boley. Design and Implementation of Highly Modular Schemas for XML: Customization of RuleML in Relax NG. In Frank Olken, Monica Palmirani, and Davide Sottara, editors, *Rule-Based Modeling and Computing on the Semantic Web, Proc. 5th International Symposium (RuleML-2011 America), Ft. Lauderdale, FL, Florida, USA*, volume 7018 of *Lecture Notes in Computer Science*, pages 17–32. Springer, November 2011.

AB14. Tara Athan and Harold Boley. The MYNG 1.01 Suite for Deliberation RuleML 1.01: Taming the Language Lattice. In Theodore Patkos, Adam Wyner, and Adrian Giurca, editors, *Proceedings of the RuleML 2014 Challenge, at the 8th International Web Rule Symposium*, volume 1211. CEUR, August 2014.

ABP15. Tara Athan, Harold Boley, and Adrian Paschke. RuleML 1.02: Deliberation, Reaction, and Consumer Families. In *Proceedings of the RuleML2015 Challenge, at the 9th International Symposium on Rules*. CEUR, August 2015.

ARBB12. Mohammad Sadnan Al Manir, Alexandre Riazanov, Harold Boley, and Christopher J. O. Baker. PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes. In Antonis Bikakis and Adrian Giurca, editors, *Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France*, volume 7438 of *Lecture Notes in Computer Science*, pages 280–288. Springer, August 2012.

ASDRV19. Dörthe Arndt, Tom Schrijvers, Jos De Roo, and Ruben Verborgh. Implicit Quantification Made Explicit: How to Interpret Blank Nodes and Universal Variables in Notation3 Logic. *Journal of Web Semantics*, 2019.

Ath11. Tara Athan. Myng: Validation with ruleml 1.0 parameterized relax ng schemas. In *Proceedings of the 5th International RuleML2011@BRF Challenge, co-located with the 5th International Rule Symposium*. CEUR-WS.org, 2011.

BLCK+08. Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3Logic: A Logical Framework for the World Wide Web. *Theory and Practice of Logic Programming (TPLP)*, 8(3), May 2008.

Bol11. Harold Boley. A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In *Proc. 5th International Symposium on Rules: Research Based and Industry Focused (RuleML-2011 Europe), Barcelona, Spain*, Lecture Notes in Computer Science, pages 194–211. Springer, July 2011.

Bol15. Harold Boley. PSOA RuleML: Integrated Object-Relational Data and Rules. In Wolfgang Faber and Adrian Paschke, editors, *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.

Bol16. Harold Boley. The RuleML Knowledge-Interoperation Hub. In José Júlio Alferes, Leopoldo E. Bertossi, Guido Governatori, Paul Fodor, and Dumitru Roman, editors, *Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings*, volume 9718 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 2016.

Bol18. Harold Boley. PSOA RuleML Explained with Blockchain Examples: Grailog Visualizations, Herbrand Models, and PSOATransRun Queries. RuleML Technical Report http://wiki.ruleml.org/index.php/PSOA_RuleML_Explained_with_Blockchain_Examples:_Grailog_Visualizations,_Herbrand_Models,_and_PSOATransRun_Queries. Keynote of Workshop on Rules: Logic and Applications, 19-20 December 2018, National Technical University of Athens (NTUA), http://fsvg.math.ntua.gr/rulesworkshop.html, December 2018.

BZ19. Harold Boley and Gen Zou. Perspectival Knowledge in PSOA RuleML: Representation, Model Theory, and Translation. *CoRR*, abs/1712.02869, v3, 2019.

DMA+19. Marjolein Deryck, Theodoros Mitsikas, Sofia Almpani, Petros Stefaneas, Panayiotis Frangos Iakovos Ouranos, Harold Boley, and Joost Vennekens. Aligning, Interoperating, and Co-executing Air Traffic Control Rules Across PSOA RuleML and IDP. In *Rules and Reasoning - Third International Joint Conference, RuleML+RR 2019, Bolzano, Italy, September 16-19, 2019, Proceedings*, Lecture Notes in Computer Science. Springer, 2019.

GBP18. Lukas Grätz, Harold Boley, and Adrian Paschke. PSOA Prova: PSOA Translation of Pure Production Rules to the Prova Engine. In Wolfgang Faber, Paul Fodor, Giovanni De Gasperis, Adrian Giurca, and Kia Teymourian, editors, *Proc. Doctoral Consortium and Challenge, 2nd International Joint Conference on Rules and Reasoning (RuleML+RR), Luxembourg, Sept. 20-26*, volume 2204 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

ZB16. Gen Zou and Harold Boley. Minimal Objectification and Maximal Unnesting in PSOA RuleML. In José Júlio Alferes, Leopoldo E. Bertossi, Guido Governatori, Paul Fodor, and Dumitru Roman, editors, *Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings*, volume 9718 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2016.

Zou18. Gen Zou. *Translators for Interoperating and Porting Object-Relational Knowledge*. PhD thesis, Faculty of Computer Science, University of New Brunswick, April 2018.

ZPPBR12. Gen Zou, Reuben Peter-Paul, Harold Boley, and Alexandre Riazanov. PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners. In Antonis Bikakis and Adrian Giurca, editors, *Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France*, volume 7438 of *Lecture Notes in Computer Science*, pages 264–279. Springer, August 2012.

A Validating a PSOA Instance against an RNC Schema

The tool Validator.nu[46] can be used to validate a PSOA RuleML/XML instance against a Relax NG (RNC) schema, as explained generally for RuleML 1.03[47]. E.g.:

`Text Field` - Paste: instance from Section 3.1 (elided content from Section 2.1)

`Schemas` - Paste: `http://deliberation.ruleml.org/1.03-psoa/relaxng/datalogPSOA_relaxed.rnc`

B Normalizing a PSOA Instance via an XSLT Stylesheet

The tool Online XSLT 2.0 Service[48] can be used to normalize a PSOA RuleML/XML instance via an XSLT stylesheet, as explained generally for RuleML 1.03[49]. E.g.:

`URI for xsl resource` - Paste: `http://deliberation.ruleml.org/1.03-psoa/xslt/normalizer/normalizer.xslt`

`URI for xml resource` - Paste: `http://deliberation.ruleml.org/1.03-psoa/exa/DatalogPSOA/cyclic-purchasePSOA.ruleml`

Clicking the `transform` button generates the result[50], which contains the normalized serialization of a complete rulebase (expanded from Section 3.2) including the fact Example T1.B and the rule Example T3 in Section 2.2.

---

[46] `https://validator.nu`

[47] `http://wiki.ruleml.org/index.php/Validating_with_Relax_NG_for_RuleML_1.03`

[48] `https://www.w3.org/2005/08/online_xslt/`

[49] `http://wiki.ruleml.org/index.php/XSLT_Processing_for_Deliberation_RuleML_1.03`

[50] `http://services.w3.org/xslt?xslfile=http%3A%2F%2Fdeliberation.ruleml.org%2F1.03-psoa%2Fxslt%2Fnormalizer%2Fnormalizer.xslt&xmlfile=http%3A%2F%2Fdeliberation.ruleml.org%2F1.03-psoa%2Fexa%2FDatalogPSOA%2Fcyclic-purchasePSOA.ruleml&content-type=&debug=on&submit=transform`