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
[2] Faculty of Computer Science, University of New Brunswick Fredericton, NB, Canada
[3] CB Unique Solutions, West Lafayette, Indiana, USA

13th International Rule Challenge, RuleML+RR 2019, September 16-19, 2019

Outline

1. Introduction
2. PSOA RuleML Examples
3. Presentation and Serialization Syntax
4. Schema Methodology
5. PSOA's Schema Enhancements
6. Conclusions and Future Work
7. Hands-on Training

Outline

1. Introduction
2. PSOA RuleML Examples
3. Presentation and Serialization Syntax
4. Schema Methodology
5. PSOA's Schema Enhancements
6. Conclusions and Future Work
7. Hands-on Training

Background

- The data and knowledge representation language Positional-Slotted Object-Applicative (PSOA) RuleML has been employed in graph-relational use cases, including ATC KB (Deryck, Mitsikas et al. 2019)
- 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

PSOA RuleML/XML 1.03

- Extends the XML specifications of Deliberation RuleML 1.02 (released) and 1.03 (ongoing)
- PSOA RuleML's "psoa atoms" allow dependent slots and explicit tuples
- Hence extensions of Deliberation PSOA RuleML/XML 1.03 with new modules and modifications of the modules that define named patterns for atoms & functional expressions
- Because of modular structure of Relax NG schemas, extensions are then applied to rest of syntactic structures, including to queries and rules, whenever the new PSOA modules are included in the governing driver schema

Prerelease of PSOA RuleML/XML 1.03

- PSOA changes to named patterns in Relax NG schema modules for content models of atoms (Node: Atom) and functional expressions (Nodes: Expr and Plex)
- New modules for:
  element `slotdep` (to complement existing `slot`),
  edge elements `tup` and `tupdep`, as well as
  Node element, `Tuple`
- New anchor languages, as in the left branch of the Figure:
  - datalogPSOA
  - hornlogPSOA
  - naffologeqPSOA

PSOA and Non-PSOA Languages Aligned

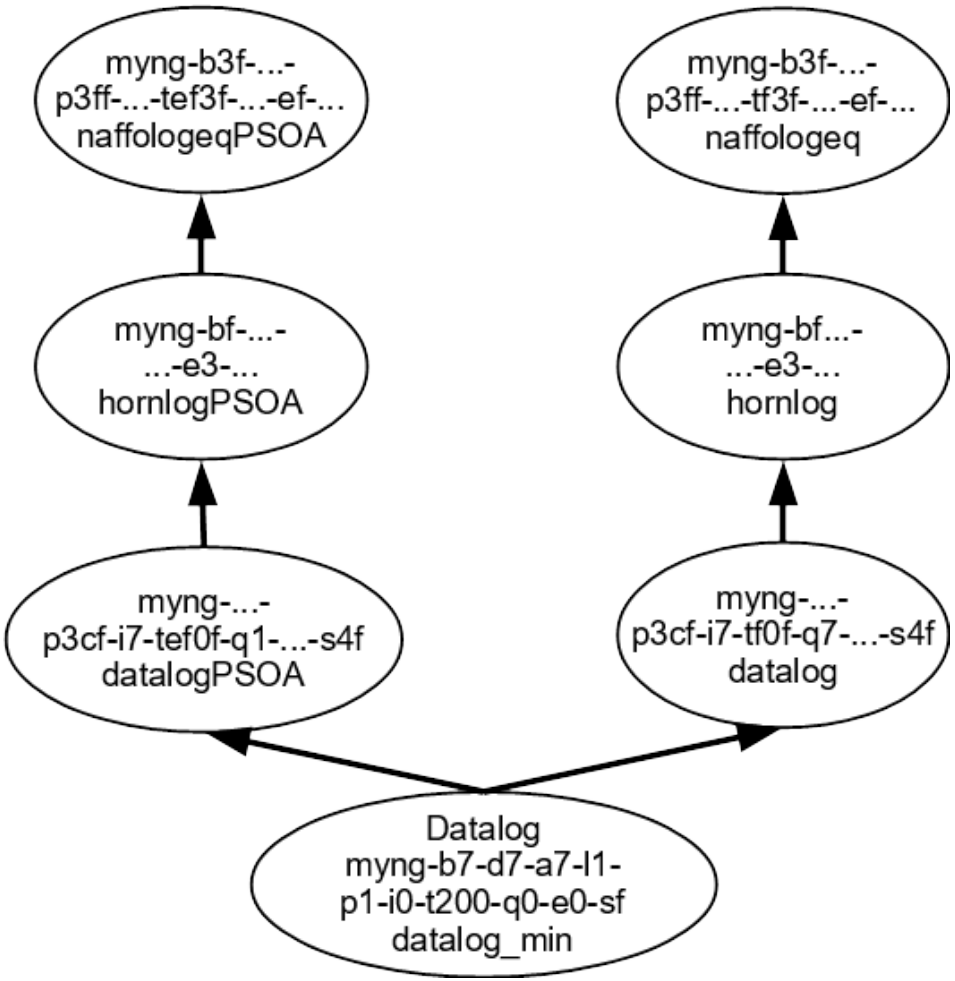

Figure: Subset of RuleML language semilattice with infimum for Datalog language (anchor `datalog_min`) and suprema for PSOA (anchor `naffologeqPSOA`) and non-PSOA (anchor `naffologeq`) languages.

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

Example S1. Purchase Record with Dependent Slots

An Object IDentifier (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 (in contrast to Example S2's item-of-the-liability).

```
<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>
```

Example S2. Liability Rule for Purchase Facts

The following rule is applicable to
the dependent-slotted fact of Example S1,

augmenting this purchase record (condition: `<if>` part)
with a liability record (conclusion: `<then>` part),

where a fresh liability OID, liabilityID(transaction200),
is generated as a functional expression
from the purchase OID, transaction200:

Example S2. Liability Rule for Purchase Facts (Cont'd)

```
<Forall>
  <Var>b</Var> ...
  <Implies>
    <if>   <!-- ?t#purchase(buyer+>?b seller+>?s item+>?i) -->
      <Atom>
        <oid><Var>t</Var></oid>
        <Rel>purchase</Rel>
        <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>
    <then> <!-- liabilityID(?t)#liability(bearer+>?b item+>?i) -->
      <Atom>
        <oid>
          <Expr>
            <Fun>liabilityID</Fun>
            <Var>t</Var>
          </Expr>
        </oid>
        <Rel>liability</Rel>
        <slotdep><Ind>bearer</Ind><Var>b</Var></slotdep>
        <slotdep><Ind>item</Ind><Var>i</Var></slotdep>
      </Atom>
    </then>
  </Implies>
</Forall>
```

Example S2. Liability Rule for Purchase Facts (Cont'd)

Two independent `<slot>`s, `buyer->?b` and `bearer->?b`,
would still disambiguate
the different-slot-name occurrences:

The slot filler-variable `?b` moves
from the condition slot name `buyer`
to the conclusion slot name `bearer`

An independent `<slot>`, `item->?i`,
would not disambiguate
the same-slot-name occurrences under different predicates:

The slot name `item` changes its dependence
from the condition predicate `purchase` (item-of-the-purchase)
to the conclusion predicate `liability` (item-of-the-liability)

Such dependence becomes most useful for disambiguation of
OID multi-membership in different predicates

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

Presentation Syntax: Template for Psoa Terms

Four lines of subsequences for four kinds of descriptors, where superscripts indicate subterms that are part of dependent ($^+$) vs. independent ($^-$) descriptors, and **left-tuple, left-dependent normal form** is assumed:

$$
\begin{aligned}
&o\#f(+[t^+_{1,1} \ldots t^+_{1,n_1^+}] \ldots +[t^+_{m^+,1} \ldots t^+_{m^+,n^+_{m^+}}] \\
&\quad -[t^-_{1,1} \ldots t^-_{1,n_1^-}] \ldots -[t^-_{m^-,1} \ldots t^-_{m^-,n^-_{m^-}}] \\
&\quad p^+_1 +> v^+_1 \ldots p^+_{k^+} +> v^+_{k^+} \\
&\quad p^-_1 -> v^-_1 \ldots p^-_{k^-} -> v^-_{k^-})
\end{aligned}
$$

Serialization Syntax: Template for Psoa Terms

General case of psoa terms in serialization syntax can be instantiated for atoms, where (decorated) letters $\mathtt{o}$, $\mathtt{f}$, $\mathtt{t}$, $\mathtt{p}$ & $\mathtt{v}$ are understood to stand for recursively serialized OIDs, predicates, terms, properties & values, respectively:

<Atom>
<oid>$\mathtt{o}$</oid><op>$\mathtt{f}$</op>
<tupdep><Tuple>$\mathtt{t}^{+}_{1,1}$ ... $\mathtt{t}^{+}_{1,n^{+}_{1}}$</Tuple></tupdep> ...
<tupdep><Tuple>$\mathtt{t}^{+}_{m^{+},1}$ ... $\mathtt{t}^{+}_{m^{+},n^{+}_{m^{+}}}$</Tuple></tupdep>
<tup><Tuple>$\mathtt{t}^{-}_{1,1}$ ... $\mathtt{t}^{-}_{1,n^{-}_{1}}$</Tuple></tup> ...
<tup><Tuple>$\mathtt{t}^{-}_{m^{-},1}$ ... $\mathtt{t}^{-}_{m^{-},n^{-}_{m^{-}}}$</Tuple></tup>
<slotdep>$\mathtt{p}^{+}_{1}$ $\mathtt{v}^{+}_{1}$</slotdep> ... <slotdep>$\mathtt{p}^{+}_{k^{+}}$ $\mathtt{v}^{+}_{k^{+}}$</slotdep>
<slot>$\mathtt{p}^{-}_{1}$ $\mathtt{v}^{-}_{1}$</slot> ... <slot>$\mathtt{p}^{-}_{k^{-}}$ $\mathtt{v}^{-}_{k^{-}}$</slot>
</Atom>

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

Modularization Design Principles

Monotonicity RuleML's restricted Relax NG is **monotonic**: 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

Orthogonality Because of this monotonicity property, Deliberation RuleML schema modules may be freely (**orthogonally**) combined to define a fine-grained poset lattice of RuleML syntaxes, with a partial order based on syntactic containment

Extensibility RuleML RNC schemas employ element definition conventions consisting of several layers of named patterns, in order to optimize schema **extensibility**

Serialization Methodology

Normalized All content models have a canonical ordering of their child elements

Relaxed Allows significantly more positional freedom in content models. It also allows certain edges to be skipped, provided they can be reconstructed unambiguously

Mixed Constructed to be the greatest subset of the relaxed serialization with the most positional freedom that can still be defined within XSD

Compact Similar to the normalized one in that it requires the same canonical ordering of elements. Differs from the normalized one in that it requires all edges skippable in the relaxed serialization to be skipped

Normalization Methodology

Goals of the RuleML normalizer:

- Reconstruct all skipped edge tags to produce a fully striped form
- Perform canonical ordering of sibling elements

Using this normalizer followed by schema-based RuleML validation performs “normalidation” on RuleML instances

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”)

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

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:

`slotdep_expansion_module.rnc`

The new module defining the dependent slot element is essentially identical to the existing module defining the independent slot element after substitution of `slotdep` for most occurrences of `slot`

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:

`tup_expansion_module.rnc`
`tupdep_expansion_module.rnc`
`tuple_expansion_module.rnc`

These modules are most similar to existing modules for `arg` and `Plex`

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 for the PSOA presentation syntax

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

The XSDs for PSOA RuleML anchor languages are auto-generated

Integration of New Modules with the MYNG Engine

The myng-code is extended to accommodate the PSOA/XML features

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 the Figure

PSOA and Non-PSOA Languages Aligned (Revisited)

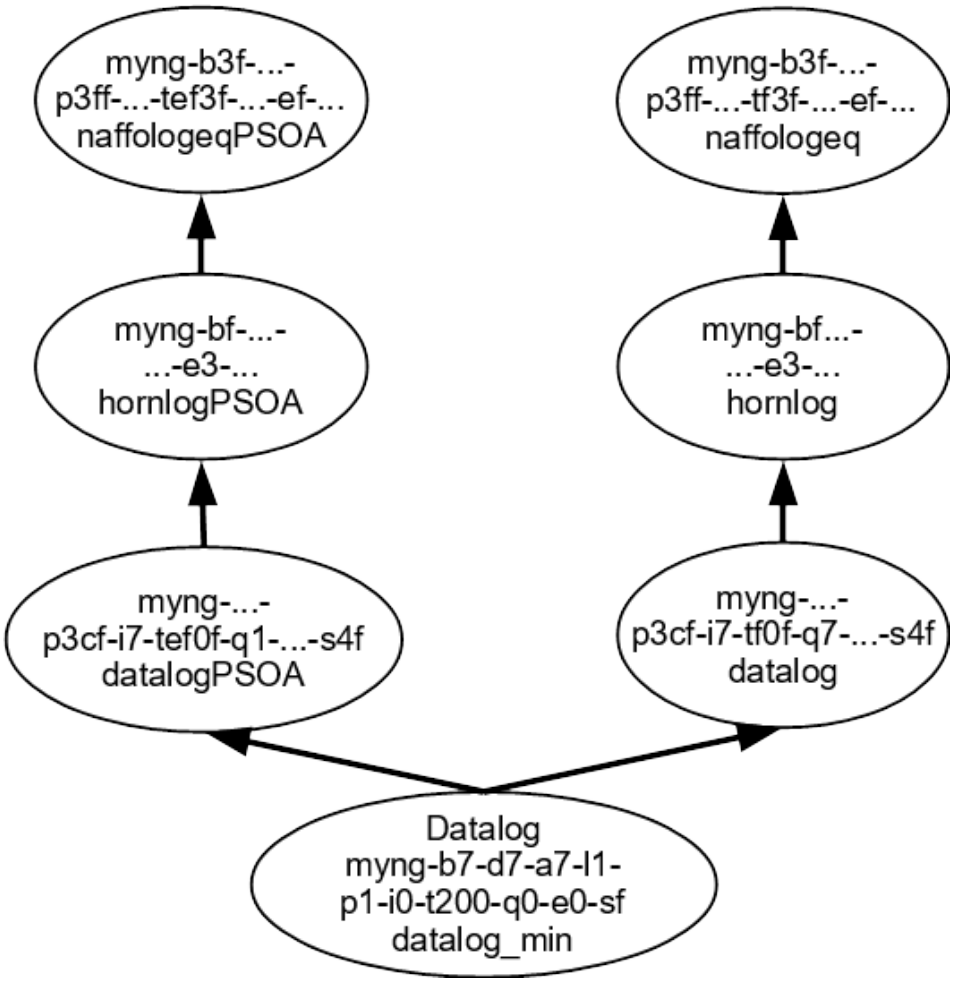

Figure: All vertices of depicted semilattice are identified by myng-codes. The myng-code for Datalog is shown in its entirety, while for other vertices only the components differing from their nearest sublanguage are shown.

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

Conclusions

- Connect PSOA RuleML work based on its presentation syntax with Deliberation RuleML work based on its serialization syntax
- PSOA-extend and -modify Deliberation RuleML's syntactic definition via a MYNG-defined schema system
- Focus on validation as well as normalization of rulebases utilizing the new PSOA features of dependent slots and explicit tuples

Future Work

Completion of PSOA RuleML/XML 1.03 release

- Syntactic completion, e.g.
  - By incorporating XML elements for `Subclass` (‘##’)
- Technical completion, e.g.:
  - Update the MYNG GUI
  - Extend for PSOA the V. 1.03 compact serialization and associated transformer (compactifier)
  - Implement PSOA features in RuleML Holog syntax

Outline

1 Introduction
2 PSOA RuleML Examples
3 Presentation and Serialization Syntax
4 Schema Methodology
5 PSOA's Schema Enhancements
6 Conclusions and Future Work
7 Hands-on Training

Validating a PSOA Instance against an RNC Schema

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

`Text Field` - Paste: instance with Slide 9 content (cf. paper)

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

Normalizing a PSOA Instance via an XSLT Stylesheet

The tool Online XSLT 2.0 Service can be used to normalize a PSOA RuleML/XML instance via an XSLT stylesheet, as explained generally for RuleML 1.03. 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, which contains the normalized serialization of a complete rulebase (cf. paper)