Design and Implementation of Highly Modular Schemas for XML: Customization of RuleML in Relax NG

Tara Athan[1], Harold Boley[2]

[1] Athan Services, Ukiah, CA, USA
`taraathan AT gmail.com`
[2] Institute for Information Technology, National Research Council Canada
Fredericton, NB, Canada
`harold.boley AT nrc.gc.ca`

Abstract. We present a re-conceptualization and re-engineering of the non-SWSL portion of the Derivation Rules subfamily of RuleML in the Relax NG Compact (RNC) schema syntax. The benefits arising from RNC schemas include decreased positional sensitivity and greater flexibility in modularization (from fine-grained modular to monolithic), as well as unification of human-readable ("Content Models") and machine-readable (XSD/XML) versions. We introduce a Relax NG schema design pattern, enforced by RNC meta-schemas, that guarantees monotonicity (grammatical extension implies syntactic containment) when any of a large number of small expansion modules are merged. The original fifteen Derivation RuleML sublanguages are thus embedded in a syntactic lattice with hundreds of thousands of languages with semantics inherited from the top language. The original RuleML sublanguages are available through links, and customized languages are available through a GUI web-app. The GUI serves as the front end to a PHP-specified parameterized schema that takes a selection of customization options and returns a schema driver file. These options are encoded to facilitate determination of syntactic containment between any pair of languages. As in earlier (Derivation) RuleML language hierarchies, (logical) expressivity forms the backbone of the language lattice. The (parameterized) RNC schema serves as a pivot format from which XSD schemas, statistically-random XML test instances, monolithic simplified RNC content models, and HTML documentation are automatically generated. The RNC-based re-engineering of Derivation RuleML has already led to the discovery and patching of errata in RuleML versions 0.91 and 1.0, as well as to suggested enhancements of version 1.0 and a newly conceived version 1.1. The specifications of the RNC-based RuleML schemas are maintained at `http://wiki.ruleml.org/index.php/Relax_NG`.

1 Introduction

RuleML is a family of languages for Web rule interchange that was originally specified in Document Type Definitions (DTDs) [W3C98], then switched to XML

Schema Definition Language (XSD) schemas [TBMM04]. Here we present a re-engineering of the non-SWSL portion of the Derivation Rules subfamily of RuleML in the Relax NG Compact (RNC) schema syntax [ISO08] on the basis of lattice [Nat] and hedge automaton theory (cf. [Mur98]). This novel, RNC schema formalism has already supported the re-conceptualization and transition from RuleML version 0.91 to 1.0, and gave insights for its evolution to version 1.1 and beyond.

Goals A re-engineering of the RuleML schemas was undertaken to achieve the following:

– Maximize Alignment with Semantics: to the extent possible, semantic constraints should be incorporated into the schema.
– Maximize Customizability: A fine-grained, highly cohesive, and loosely-coupled modular schema design will allow a user to custom-build a RuleML sublanguage by assembling a selection of modules.
– Maximize Automation: The assembly of custom schemas and the production cycle of schema releases should be automated as much as possible.
– Maximize Reliability: The new schemas should be exhaustively tested against the existing hand-written XSD schemas and instances, e.g. via automatically-generated testing instances as well as hand-written exemplary instances for 'near-miss' (invalid) and 'corner' (valid) cases.
– Maximize Extensibility: The schemas should enable extension by users, as well as RuleML developers.

An Example of Customizable Schema Definition: Equations In the original RuleML 0.91 family of languages, equations are available from Horn logic languages up, which also include, e.g., negations, disjunction, and quantification. However, equations are also desirable from Hornlog down, e.g. between individual constants in Datalog, even when Datalog is further specialized to only binary relations or to only facts. Hence, in RuleML 1.0, equations should be freely combinable with the other RuleML sublanguages. Similarly, languages with only binary relations in RuleML 0.91 are just allowed for Datalog, but in RuleML 1.0 should be also allowed up the family tree. We thus propose a method to permit free combinations of fine-grained modular features for customizable schema definition.

Of course, it is always possible to author or validate with a more permissive schema, i.e., a schema defining a language that syntactically contains the language of interest. However, a minimal schema improves the efficiency of validation, enhances authoring in a content-completion environment, and improves reliability when a minimal feature set is mandated by specification.

In the previous modularization approach, a significant redefinition of the XSD schema would be required to add equations to a smaller sublanguage, such as `bindatagroundfact`. With the re-engineered Relax NG schema, we may accomplish this task with the following steps:

1. Open the GUI[1] and select only the language features desired. For the smallest language with equations, we select the first options in the radio button sets (Expressivity - Atomic Formulas, Default Attributes - Required to be Absent, and Term Sequences - None) and deselect all checkboxes except Equations.
2. Click the Refresh Schema button to see the corresponding schema driver file and the URL that may be used to perform validation. This long URL has base `http://ruleml.org/1.0/relaxng/schema_rnc.php`, which points to the PHP-specified parameterized schema, and a query string `?backbone=x0&default=x5&...` that encodes the selected language options. Notice the schema driver file contains only nine modules, out of over fifty available.
3. Associate the schema driver file with an xml file using the `xml-model` processing instruction [GK10], where the value of `href` is the URL obtained in step 2 with all ampersands escaped as `&amp;`.
4. Edit the xml file with an xml-model processor, such as oXygen[2], to create equations such as[3]:

```
<?xml-model href=
"http://ruleml.org/1.0/relaxng/schema_rnc.php?backbone=x0&amp;default=x5&amp;termseq=x0&amp;
lng=x1&amp;propo=x0&amp;implies=x0&amp;terms=x10&amp;quant=x0&amp;expr=x0&amp;serial=x0"
type="application/relax-ng-compact-syntax"?>
<RuleML xmlns="..."><Assert><formula>
      <Equal>
        <left><Ind>Lady Gaga</Ind></left>
        <right><Ind>Stefani Joanne Angelina Germanotta</Ind></right>
      </Equal>
</formula></Assert></RuleML>
```

The Original Fifteen Languages as a Lattice A partially-ordered set (poset) in which every pair of elements has both a greatest lower bound (glb, infimum) and a least upper bound (lub, supremum) in the set is called a lattice. The fifteen languages in the non-SWSL[4] portion of the Derivation RuleML language subfamily satisfies the lattice conditions with respect to the partial ordering imposed by syntactic containment, as shown in Figure 1 and may be embedded in the larger lattice described in Section 2.1. The binary numbers below each named language demonstrate how a code can be used to identify unnamed languages uniquely as well as facilitate the determination of order by bit-wise comparison. These codes were generated from the lattice diagram, starting from the bottom and proceeding through the diagram upward and left-to-right, as shown below. Given a language whose code has not yet been determined:

1. determine the conjunction (i.e. bit-wise maximum) of all of its sublanguages;
2. if the conjunction is not equal to any other code assigned so far, it may be selected as the code, but if the language contains features that are not in any of its sublanguages, one may choose to proceed to step 3;

[1] GUI: `http://ruleml.org/1.0/gui/`.
[2] oXygen: `http://www.oxygenxml.com/`.
[3] The RuleML 1.0 namespace is still open; it will appear at `http://ruleml.org/1.0/`.
[4] An extension of Hornlog RuleML was developed to serialize SWSL (Semantic Web Services Language) in XML, whose syntax goes significantly beyond the other languages (`http://www.w3.org/Submission/SWSF-SWSL/#sec-markup`), and so cannot be accommodated in the lattice shown in Figure 1.

3. otherwise add a 1 at the least-significant 'unused' (i.e. so far not used anywhere else in the lattice) bit to the conjunction from step 1.

The choices made in step 2 of this non-deterministic procedure when applied to generating the 'original fifteen' are seen in Figure 1 (note caption for caveat).

Overview of the Relax NG Language The Relax NG language was chosen for this re-engineering effort because of its decreased positional sensitivity and its greater flexibility in modularization (from fine-grained modular to monolithic), as well as unification of human-readable ("Content Models") and machine-readable (XSD/XML) versions. These benefits are achieved through unique features of the Relax NG schema language [ISO08], including the `notAllowed` reserved word to create abstract patterns, definitions with `combine` attributes (`|=`, `&=` in the compact syntax) to merge definitions that are decomposed across modules, and the interleave operator `&` (a generalization of the `xsd:all` group) to create order-insensitive content models. Because Relax NG is theoretically grounded in *hedge automaton* theory, modularization is always possible since regular hedge languages are closed under the operations of intersection, union and complement [Mur98].

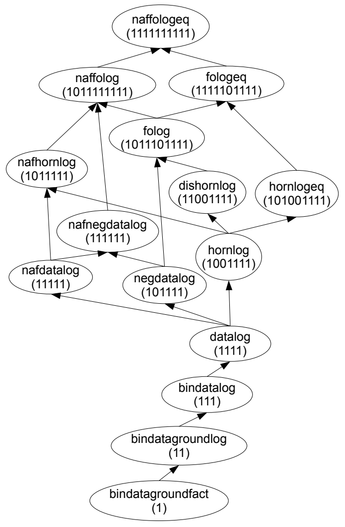

Fig. 1. Hasse diagram of the 'original fifteen' language lattice with arrows, and illustrative binary codes, indicating syntactic containment. The code assignment was generated by the procedure described in Section 1, but is not unique for this poset, as it depends on the way the Hasse diagram has been drawn (as a 2-D projection of a unique Directed-Acyclic Graph), as well as choices made in the implementation of the non-deterministic procedure. When the partial order of post-schema validation infoset (PSVI) containment (see Section A) is considered, the 'original fifteen' violate the lattice conditions due to the use of default attributes. Therefore, the implemented schemas use a different coding that reflects grammatical, syntactic, and PSVI containment, described in Section 2.1.

2 Design of the RuleML Relax NG Schema

The design consists of several components with different levels of abstraction. For the beginning user, URL redirects[5] provide default access to serializations[6] of the original fifteen RuleML sublanguages. For the advanced user, the GUI web-app

[5] E.g., the URL for Datalog in relaxed-form RNC: `http://ruleml.org/0.91/relaxng/datalog_relaxed.rnc`.

[6] See the 'Serialization' subsection of Section 2.1.

allows selection among many syntactic options and computes the URL of the dynamically-generated driver file for the customized language. A PHP-specified parameterized schema[7] implements the mapping from the syntactic options to the corresponding subsets of modules.

2.1 GUI Web-App and Language Options Encoding

The GUI web-app consists of an XHTML form that accepts a user's input of language options through radio buttons and check boxes. A URL that points to a PHP script, described in the next section, with a query string of the language options encoded compactly, is generated by the form and may be used directly for validation of instance documents.

The language options are organized into facets of semantically-related dimensions. Each dimension is Boolean, and the dimensions are freely combinable, although some are 'dormant' (produce no syntactic or semantic change) unless an 'activating option' is also selected. For example, the slot cardinality attribute, `card`, is dormant unless slotted arguments are included, because this attribute is only allowed on the `slot` element. In the GUI, each dormant option is disabled unless at least one of its activating options is selected. For each group of options (e.g. backbone, default, ...), the Boolean values are treated as bits of a hexadecimal number. The full selection of options is assembled as a hexadecimal-valued query string[8] e.g.

```
backbone=x3f&termseq=x7&default=x3&serial=xf&propo=x3f}
          &implies=x7&terms=xf3f&quant=x7&expr=xf&lng=x1
```

to form a unique syntactic code for each language. Bit-wise dominance between two codes is equivalent to syntactic containment of the corresponding languages. The option facets are described in the following subsections, with the facet parameter name(s) given parenthetically in the title of each subsection.

Backbone (backbone) The logical connectives of propositional logic and the variables and quantifiers of predicate logic are implemented in independent modules so that a great variety of expressivities may be constructed by 'mixing-in' various schema modules. However, only certain combinations of these modules are accessible from the GUI, corresponding to an unbranched hierarchy from ground atomic formulas to full first-order logic, which we call the "backbone" of the language lattice (see Figure 2).

Positional Arguments (termseq) In `Atom`ic formulas and in `Expr`essions, the sequence of positional arguments (as opposed to the bag of `slot`ted arguments) may be necessarily empty (None), limited to empty or length two (Binary), or allowed to be of arbitrary finite length (Polyadic) (see Figure 3).

[7] PHP: `http://ruleml.org/0.91/relaxng/schema_rnc.php`.

[8] When the query string is used in an `href` attribute, `&` should be escaped as `&amp;`.

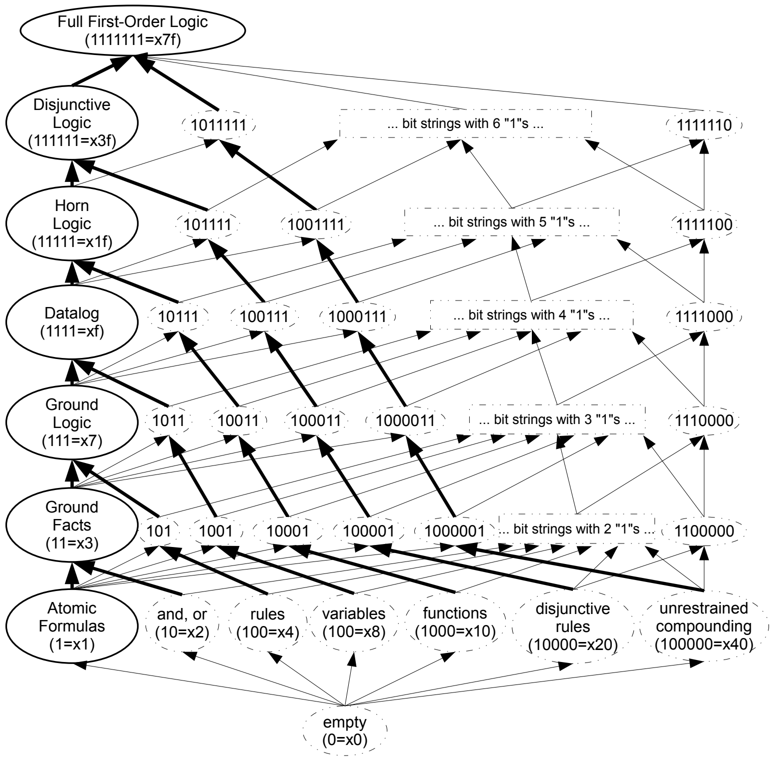

Fig. 2. Hasse diagram of the backbone sublattice with binary and hexadecimal codes. The options within solid ovals are available from the GUI, the others can be accessed through the parameterized schema, as described in Section 2.2.

Attributes with Default Values (default) In the RuleML XSD schemas, certain attributes are defined with default values. In some situations it may be advantageous to eliminate the default values so that the language is more compact; this is the first option, "Required to be Absent". The second option, "Required to be Present", allows the Relax NG schema to emulate the post-schema validation infoset (PSVI) of instances validated against XSD schemas, by requiring attributes having default values to be present. This constraint is necessary for PSVI emulation because Relax NG validation does not allow modification of the info-set, in contrast to XSD validation, which inserts attributes having default values when they are absent in the instance document. The third alternative, "Optional", allows such attributes to be absent or present, and thus is the *join* (w.r.t. lattices, the least upper bound) of the former two languages.

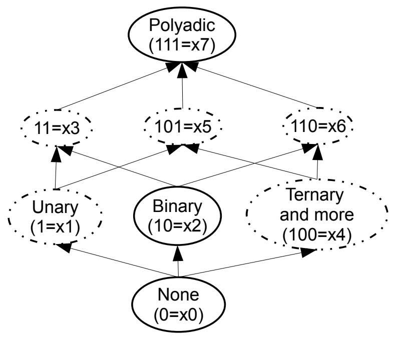

Fig. 3. Hasse diagram for the term sequence facet. Options available from the GUI include polyadic term sequences, binary term sequences, and the absence of term sequences ("None"), the latter corresponding to propositional and frame-like languages. Additional options such as unary term sequences are not yet available, but are accommodated in the parameter encoding for this facet for future implementation.

Serialization (serial) Only three serialization forms are implemented in the URL redirects: the "relaxed-", "normal-" and "mixed- form" serializations. Advanced users have additional options available through the GUI web-app. The *normal-form* serialization, corresponding to all of the serialization options unchecked, realizes canonical ordering of child elements and required 'striping'[9] as well as the "Required to be Present" treatment of attributes with default values described above in Section 2.1. The *relaxed-form* serialization, corresponding to all of the serialization options checked, is maximally insensitive to the order of child elements while still retaining unambiguous semantics, and has optional striping, as well as the "Optional" treatment of attributes with default values.[10] A *mixed-form* schema is also implemented to reproduce the syntax of the original XSD schemas for testing purposes, but is not available from the GUI.

Mix-Ins (propo, implies, terms, quant, expr) Additional syntactic options include equivalence, meta-logic, negations, semantic variants of implications, expressions and equations, slots, rest variables, object identifiers, resource identifiers (IRIs), degree of uncertainty, explicit typing, reification, and skolem constants.

Alternate Names (lng) The default abbreviated English element and attribute names may be replaced by long English names. The modules implementing that replacement serve as a template for other alternate name sets, enabling internationalization. Expansion modules are used to add the alternate names, and contraction modules are required to remove the default names. The language lattice of the alternate name set is isomorphic to that of the default name set.

[9] *Striping*: the alternation of 'Node' (upper-cased 'Type-tag') elements with 'edge' (lower-cased 'role-tag') elements.

[10] Additional options for disabling explicit datatyping and the schema location attribute are necessary for the normal form as a workaround to a bug in the translator from Relax NG to XSD schemas.

2.2 Parameterized Schema

The parameterized schema is implemented as a PHP script which accepts language options encoded in a query string[11] and generates the driver file. This file assembles the grammar by inclusion of modules, and contains only namespace declarations, start and include statements, and comments. The PHP script performs a monotonic transformation of the parameters passed in the query string into Boolean variables, each indicating the inclusion of one or more modules. In a few cases, a pair of modules is replaced by one syntactically equivalent module so that simpler grammar patterns may be employed.

The start pattern of the driver file determines the elements allowed as a document root. In general, the specification of document root in Relax NG does not translate to XSD schemas, where any global element may be the document root. In the RuleML driver files, the start pattern is constructed as a choice among all globally defined elements, in order to maintain equivalence to XSD schemas, and to allow RuleML fragments to validate.

2.3 Design Patterns for Modules

The modularization of the parameterized schema is constrained by the requirement of realizing the original fifteen languages. However, there is still considerable freedom in the design. Certain design decisions influence the nature of the "enriched" lattice by altering the coverage of the new languages that are created. Our design pattern has a number of aspects in common with the XHTML 2.0 Relax NG schema [IG09], but is significantly more constrained in order to maximize both decoupling and monotonicity of the modular system.

Module Decoupling. Like XHTML2.0 and the existing RuleML XSD schemas, the Relax NG schema uses a flat schema design pattern, which declares all the elements globally using named patterns, enhancing extensibility. There are many ways to decouple such patterns in Relax NG, including using abstract patterns, unreachable patterns and 'linking modules'. For example, in RuleML a `Negation` formula (strong negation) is allowed to occur within a `Naf` formula (weak negation, `N`egation `a`s `f`ailure), provided both kinds of negation are included in the language. The RNC code that activates this coupling is

```
NafFormula.choice |= Negation-node.choice
```

We call definitions of one pattern as a formula of other named patterns *linking* definitions, to distinguish them from definitions that explicitly define elements or attributes. We can place this linking definition in the `Negation As Failure` module `naf_module.rnc` or the `Negation` module `neg_module.rnc`, or potentially another module.

Unreachable Patterns: The linking definition may be placed in the `Negation`

[11] The query string may be manually edited to obtain some options not directly available from the GUI.

module. If a language includes strong but not weak negation, the `NafFormula.choice` pattern is valid but unreachable. This approach is efficient in lines of code, but can be hard to read in modular form, since the definition of some patterns, in this case `NafFormula.choice`, is fragmented across modules.

Abstract Patterns: If the linking definition is not placed in the `Negation As Failure` module, the module where it occurs will be invalid on its own unless we add an additional definition to make the `Negation-node.choice` pattern abstract. In Relax NG Compact (RNC) syntax, an abstract pattern is created with the `notAllowed` reserved word as follows:

```
Negation-node.choice |= notAllowed
```

If we place the linking definition into the `Negation` module, then the abstract pattern is overridden whenever this module is included, and the link is activated if both negation modules are included in the language. Having a large number of `notAllowed` definitions causes the code to look cluttered and to be more difficult to maintain, so these definitions are collected into a single 'initialization' module, which is included in every schema driver file. Similarly, patterns combined with the interleave attribute are initialized `empty`.

Linking Modules: If the linking definition is placed in a third 'linking' module, the greatest flexibility would be attained, allowing the decision to couple the two kinds of negation to be made independent of their inclusion in the language. *Linking* modules contain linking definitions, but no new element or attribute definitions. The patterns on both sides of the link must be defined in the initialization module to ensure that the modules may be combined freely to form a valid grammar.

In RuleML/RNC, we use the Linking Module design whenever feasible, as this provides maximum modularity. In particular, this design pattern is used to implement the transition from the Datalog / Horn logic languages to the full first-order logic languages without resorting to redefinition, by placing linking definitions for unrestricted formula compounding into a *folog* expansion module.

Monotonicity from Segregated Names. In Relax NG schemas, pattern names are the non-terminal symbols used to write production rules. One of the features of our schema design pattern is segregation of pattern names according to the allowed value of the `combine` attribute of their definitions. The segregated naming design pattern has been specified in a set of meta-schemas[12] in the RNC language, that can be used to validate base grammars, and expansion and contraction modules after translation into the XML-based Relax NG syntax. To illustrate the constraints on these categories, we draw examples from several RuleML modules.

An extension point and several abstract patterns for equality are initialized in `init_expansion_module.rnc` as follows:

[12] `http://www.ruleml.org/1.0/designPattern`.

```
Equal-node.choice |= notAllowed    # for alternate names of equality element
Equal-datt.choice |= notAllowed    # for required attributes of equality element
reEqual.attlist &= empty           # for optional attributes of equality element
Equal.header &= empty              # for modifying children of equality element
Equal.main |= notAllowed           # for main content of equality element
```

In `equal_expansion_module.rnc`, the above patterns are assembled as follows:

```
Equal-node.choice |= Equal.Node.def
Equal.Node.def =
    element Equal { (Equal-datt.choice & reEqual.attlist), Equal.header, Equal.main }
Equal.header &= SimpleFormula.header?
Equal.main |= leftSide-edge.choice, rightSide-edge.choice
```

Additional definitions provide the patterns for the left- and right-hand sides. In `long_name_expansion_module.rnc` we have

```
Equation-node.choice |= Equation.Node.def
Equation.Node.def =
    element Equation { (Equal-datt.choice & reEqual.attlist), Equal.header, Equal.main }
```

In `short_name_contraction_module.rnc` we have

```
Equal.Node.def &= notAllowed
```

These schema snippets illustrate the full range of definitions permitted in the Relax NG schema design pattern. We utilize three categories of pattern names.

Choice Combine: In base grammars and expansion modules, patterns with names from the choice category must be defined with the choice combine operator |=. In the example above, `Equal.choice` and `Equal.main` are names in the choice category. In practice, choice patterns are defined as `notAllowed` in the initialization expansion module, and then overridden in expansion modules, as shown above. Choice combine definitions are not allowed in contraction modules.

No Combine: In base grammars and expansion modules, patterns with names from the no-combine category must be defined, with =. In base grammars and contraction modules, it is permitted to have definitions having names from this category with the combine attribute `interleave`, whose pattern is the notAllowed reserved word. We use this construction in the alternate names modules, as shown above, to remove abbreviated element names when they are replaced with long or internationalized names. Because neither of the definitions

```
Equal.Node.def &= empty
Equal.Node.def |= notAllowed
```

would be permitted in the intialization expansion module, the names in the no-combine category are never initialized. This introduces limitations on how abstract components may be defined. To define abstract elements and attributes, we introduce a more abstract choice pattern, such as `Equal-node.choice`, as shown above. Such choice patterns are extension points that hold alternate name elements or alternate constructions that serve the same role in the grammar, and unify elements that have similar semantics.

Interleave Combine: In base grammars and expansion modules, patterns with names from the interleave combine category must be defined with the interleave combine operator `&=`. Names from the interleave combine category may not be defined in contraction modules. The interleave combine is used to initialize interleave patterns, such as lists of optional attributes, as empty. Other uses are to add attributes to an attribute list, and, in the order-insensitive syntaxes, to add children to the interleave header patterns, as shown above for the `Equal` element. An additional constraint is required to attain monotonicity. In an expansion module, the right-hand side of a definition with a combine attribute of interleave must be optional (`?`), zero-or-more (`*`), or empty, as shown above for the `reEqual.attlist` pattern.

2.4 Transformation

The RNC parameterized schema serves as a pivot format from which XSD schemas, statistically-random XML test instances, monolithic simplified RNC content models, and HTML documentation are automatically generated.

Auto-generated Normal Form XSD. The Jing/Trang software is used to transform the parameterized schema and included modules into monolithic normal-form or mixed-form XSD schemas as follows:

– Jing[13] with switch `s` is used for simplification of modular RNC schemas into monolithic Relax NG XML syntax (RNG);
– Trang[13] is used for transformation of RNG schemas into XSD.

The XSDs corresponding to the original fifteen RuleML sublanguages are made available for remote validation[14] or download in a zip archive that also includes the PHP script for the parameterized schema and Windows batch scripts for transformation and validation.[15]

Instance Generation. The oXygen software package is used to generate instances from XSD schemas. Instances generated from the original XSD schemas are used to exhaustively test that the RNC relaxed-form languages syntactically contain the corresponding original RuleML language, while instances of the normal- and mixed-form XSD schemas auto-generated from RNC are similarly employed for testing for syntactic containment or equivalence, respectively, relative to the original languages.

[13] `http://code.google.com/p/jing-trang/`.
[14] Horn logic in normal-form XSD: `http://www.ruleml.org/0.91/xsng/hornlog_normal.xsd`.
[15] Normal-form Zip Archive: `http://ruleml.org/0.91/relaxng/ruleml0-91_normal_rnc.zip`.

Simplified RNC as Content Model. The `jing -s` transformation described in Section 2.4 is also the first step in generating the simplified, monolithic RNC schemas that serve as auto-generated content-model documentation, replacing error-prone hand-generated documentation, the second step being transformation by `trang` back into RNC. The `jing -s` simplification is a by-product of the validation process, and so does not provide the ideal documentation, as meaningful pattern names, e.g. `formula-Query.Node.def`, are replaced by simplified names, e.g. `formula_3`, that somewhat obfuscate meaning in the translation. Nevertheless, this is an easy and highly reliable method for condensing the modular grammar into a monolithic, and more human-readable, form.

HTML Documentation of Syntax and Semantics. Absent an application to generate documentation directly from Relax NG schemas, chaining `Trang` transformation into XSD with oXygen documentation tools for XSD schemas provides this capability to some extent. Relax NG annotations, which are preserved under Trang transformation, provide the semantics of components. The documentation need only by prepared for the top language[16], as sublanguages inherit their semantics from the top language.

3 Implementation of the RuleML Schema Design

The RNC-based re-engineering of Derivation RuleML has already led to the discovery and patching of errata in RuleML versions 0.91 and 1.0, as well as to suggested enhancements of version 1.0 and a newly conceived version 1.1.

3.1 Implementation in RuleML 0.91

The RNC implementation for RuleML 0.91 reproduces the previously released RuleML 0.91 sublanguages, with the exception of the following patches, which fix errata[17] discovered during the Relax NG re-engineering:

- Accidental omission of type declarations in the content model of `Rulebase`, unexpectedly allowing arbitrary content in some elements.
- Relaxation of order sensitivity resulted in an overly general content model for atomic formulas, expressions, and some types of generalized lists, allowing semantically-incorrect multiple occurrences of rest variables.

In addition to the fifteen original RuleML 0.91 sublanguages, the language lattice generated by the parameterized RNC schema permits many other languages. A few notable features of the thus enriched language lattice are listed here:

- Equations are made available at all levels of expressivity.

[16] `http://www.ruleml.org/0.91/relaxng/naffologeq_relaxed.rnc`.
[17] `http://wiki.ruleml.org/index.php/XSD-Errata0.91#RuleML_0.91_XSD_Errata`.

- A short URL (`http://ruleml.org/0.91/rnc`) for the top RuleML language redirects to the parameterized schema of the most inclusive language (except for alternate element names).
- Propositional languages are introduced by allowing an option that requires positional argument sequences to be empty. To realize this, the pattern for the positional arguments in atomic formulas is initialized as empty, and only extended in optional modules for binary and polyadic term sequences.
- Expansions of the propositional languages with slots and/or object identifiers provide a pure frame-like and/or object-oriented syntax.
- The option to restrict positional argument sequences to zero or two members (as in the RuleML `bin` languages) is made available at all backbone levels. At present, the restriction is applied simultaneously to atomic formulas, expressions and plexes (generalized lists); this may be relaxed later.
- More alternatives are available for stripe-skipping and child order in `Implies` and `Entails`, including skipping the stripe of only one child (`body` or `head` in 0.91, `if` or `then` in 1.0) and simultaneously relaxing the order constraints on these children. Canonical ordering (`body` before `head`) is only imposed when both stripes are skipped. The GUI allows stripe-skipping and order-insensitivity to be selected independently.

3.2 Implementation in RuleML 1.0

The Relax NG schemas for Derivation RuleML 1.0 are a relatively small upgrade from the 0.91 versions. We adopt several name changes already incorporated into the RuleML 1.0 XSD schemas [BPS10]. Beyond Derivation RuleML, we consider Relax NG versions of all of Deliberation RuleML, including higher order logic and modal logic, as well as of Reaction RuleML, including actions and events.

3.3 Preview of Proposed RuleML 1.1

A primary goal of the proposed RuleML 1.1 revision is alignment with semantics, including removal of semantically-invalid constructs that were previously included because of limitations of XSD. Such constructs can be identified through a more formal specification of the semantics, as in PSOA RuleML [Bol11] and the planned Common Logic (CL) RuleML, and a mapping of syntactic sugar to the corresponding traditional first-order logic statements.

- The entire Fuzzy RuleML specification [DPSS06] will be implemented, where all formulas, not only `Atom`s, may have a `degree` (of uncertainty) child.
- Following the recommendations from [Vli03], for all terminal elements except `Data` and also for all attributes with arbitrary values, the `xs:string` datatype will be replaced by `xs:token`, which has the same lexical space as `xs:string`, but its value space consists of lists of tokens separated by single spaces. This is appropriate because RuleML is for the most part a 'data-oriented' application where white space is not significant, other than as a token separator. This will allow users, e.g., to more easily extend the schema to a restricted vocabulary without concern for white-space multiplicity.

- Within `Entails`, elements `if` and `then` will be allowed formulas as children, in addition to formulas wrapped in a `Rulebase` element, increasing module independence.
- The performatives module will be decomposed to separate the definition of the `Query` element from the definitions of `Assert` and `Retract`, allowing the creation of a knowledge-base language (`Assert` and `Retract` performatives only) and a query language. These languages are already being used in OO jDREW [BBH$^+$05].
- The content model of `Reify` will be restricted to Node elements, to remove the meaningless reification of edge elements.
- The `Data` element will be split into two (namely, `Data` and `Structure`), one having simple content (cf. XML Schema Part 2, Datatypes [BM04]) and the other complex content (cf. XML Schema Part 1 and Relax NG). This will give `Data` back its original 'leaf-level', `Ind`ividual-like meaning and reserve the new `Structure` for 'tree-level', `Expr`ession-like content. This is necessary to allow the restriction of `Reify` described above while maintaining auto-translation from RNC normal-form to XSD, and is also desirable to avoid a definition for `Data` that mixes simple and complex types.
- Context-sensitive constraints, such as "`Fun`ctions within `Equal`s within an `Equivalent` must be either interpreted (`per value`) on both sides or uninterpreted (`per copy`) on both sides", will be realized in RNC schemas, but are not translatable to XSD (without Schematron), as they require non-deteriministic patterns.

4 Conclusions

Through the re-conceptualization and re-engineering of the RuleML schemas, considerable progress has been made towards the goals stated in Section 1:

To increase alignment with the semantics, names were assigned to recurring grammar patterns, e. g. formulas allowed in conclusions, enabling pattern reuse.

In order to increase customizability, a schema design pattern was developed which allowed us to build a system with over fifty freely combinable modules, leading to more than $2^{50} > 10^{15}$ grammars generating an estimated 300,000 different (and meaningful) languages. Further, we used the partial-order relations of containment (of grammars and languages) to organize the resulting grammars and their generated languages into lattices, related by order-preserving mappings, and labeled by codes that facilitate the determination of containment between any pair of grammars or languages.

To increase automation and reliability, we developed a GUI, a PHP-specified parameterized schema, scripts for transformation and validation, and meta-schemas to enforce the schema design.

To increase extensibility, numerous extensions points have been introduced, as named patterns. The use of such extension points has been illustrated by modules that implement an alternate element name set. Further development of the RuleML languages will take advantage of this extensibility to introduce new features in versions 1.1 and beyond.

References

[BBH$^+$05] Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer. The OO jDREW Reference Implementation of RuleML. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *Rules and Rule Markup Languages for the Semantic Web, First International Conference (RuleML 2005), Galway, Ireland, November 10-12, 2005, Proceedings*, volume 3791 of *Lecture Notes in Computer Science*, pages 218–223. Springer, 2005.

[BM04] P.V. Biron and A. Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, W3C, October 2004. `http://www.w3.org/TR/xmlschema-2/`.

[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), Barcelona, Spain, July 2011*, Lecture Notes in Computer Science. Springer, 2011.

[BPS10] Harold Boley, Adrian Paschke, and Omair Shafiq. RuleML 1.0: The Overarching Specification of Web Rules. In *Proc. 4th International Web Rule Symposium: Research Based and Industry Focused (RuleML-2010), Washington, DC, USA, October 2010*, Lecture Notes in Computer Science. Springer, 2010.

[DPSS06] Carlos Viegas Damasio, Jeff Z. Pan, Giorgos Stoilos, and Umberto Straccia. An approach to representing uncertainty rules in ruleml. In *Proc. of the 2nd International Conference of Rules and Rule Markup Languages for the Semantic Web (RuleML-2006)*. 2006.

[GK10] Paul Grosso and Jirka Kosek. Associating schemas with xml documents 1.0 (first edition). `http://www.w3.org/TR/xml-model`, 2010.

[IG09] Masayasu Ishikawa and Markus Gylling. XHTML 2.0 RELAX NG Definition. `http://www.w3.org/TR/xhtml2/xhtml20\_relax.html\#a\_xhtml20\_relaxng`, 2009.

[ISO08] ISO. ISO/IEC 19757-2: Document Schema Definition Language (DSDL) Part 2: Regular-grammar-based validation - RELAX NG. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip`, 2008.

[Mur98] Makoto Murata. Hedge automata: a formal model for xml schemata. `http://www.horobi.com/Projects/RELAX/Archive/hedge\_nice.html`, 1998?

[Mur11] Makoto Murata. Re: Theory question: sub-grammars and sub-languages. `http://tech.groups.yahoo.com/group/rng-users/message/1345`, 2011.

[Nat] J. B. Nation. Notes on lattice theory. `http://www.math.hawaii.edu/~jb/lat1-6.pdf`.

[TBMM04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. World Wide Web Consortium, `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`, 2004.

[Vli03] Eric van der Vlist. *RELAX NG*. O'Reilly, 2003.

[W3C98] W3C. Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1. World Wide Web Consortium, `http://www.w3.org/XML/1998/06/xmlspec-report.htm`, 1998.

A Language Lattices

Informal Definitions of Containment. There is a variety of levels at which we may define a partial ordering on a family of XML markup languages and their grammars (schemas). We list here informal definitions of three of the containment-based partial orderings that are relevant to the RuleML language lattices. Formal definitions of these and other orderings and their mathematical consequences are provided on the RuleML Language Lattice Wiki Page[18].

- PSVI Containment: A language L_1 is a PSVI sublanguage of another language L_2 if every valid document in L_1 can be mapped to a valid document in L_2 with the same post-schema-validation infoset.
- Syntactic Containment: A language L_1 is a syntactic sublanguage of another language L_2 if every grammatically-valid document of L_1 is also a grammatically-valid document of L_2.
- Grammar Containment: A language L_1 is a grammatical sublanguage of another language L_2 if the grammar of L_2 is an extension of the grammar of L_1 created by adding new production rules and/or new terminal symbols.

Monotonicity from Schema Design Pattern. In general, these partial-order relations are not equivalent. We introduced above a schema design pattern that guarantees syntactic containment given grammar containment.

We consider the operation of merging two grammars as modules that are both included, without overrides, by a driver file. According to [Mur11], if the Relax NG syntax did not include the interleave combine attribute, the merger operation would be monotonic; that is, any valid instance of one of the modules would also be a valid instance of the merged grammar. Such monotonicity is very powerful, but at a high price – the fine-grained modularization we seek would be impossible without the interleave combine.

Our objective can be met with a compromise – we aim for a weaker monotonicity and allow a restricted usage of the interleave combine. Consider the operation of merging two grammars, one being the base grammar and the other we call an expansion module. If any valid instance of the base grammar is also a valid instance of the merged grammar, then we have a one-sided monotonicity that is sufficient to establish the correspondence between the subset of included modules and the lattice of languages generated by syntactic containment partial order. This monotonicity also provides modular extensibility with backward compatibility, i.e., a grammar may be extended by including an expansion module without invalidating previously valid instance documents.

The segregated names schema design pattern described in Section 2.3 provides the desired monotonicity property. The use of an interleave combine with an optional child in an expansion module can be shown to preserve monotonicity by transforming the base grammar and expansion module pair to a pair of modules without interleave combine whose merger is equivalent to the merger of the first pair. For example, the two interleave combine definitions

```
x.interleave &= a
x.interleave &= y?
```

are equivalent to the following choice combine definitions:

```
x.choice |= a
x.choice |= a & y
```

[18] http://wiki.ruleml.org/index.php/Language_Lattice.