RSS XML Schema Grammar Induction

18879: Stochastic Optimization Report

Vadim Zaliva
Department of Electrical and Computer Engineering
Carnegie Mellon University
NASA Research Park
Moffett Field, CA 94035
vzaliva@cmu.edu

Abstract

In this paper, we will describe our attempt of applying Evolutionary Algorithms (EA) to the XML Schema grammar induction problem. Our goal is to derive an XML Schema which formally describes the grammar of a given corpus of RSS XML documents. This process is known as a *grammar induction*. The goal of the grammar induction process is to learn a formal grammar for a constructed or a natural human language based on a corpus of text. A formal grammar is usually defined via a hierarchy of production rules which could be organized into a tree. EA could be used to evolve this tree to find the best matching grammar. The fitness function reflects how well samples from the training corpus match candidate grammar. Production metrics, like the number of non-terminals, could also be a part of the fitness function to encourage discovery of specific rather than generic grammars.

1 Introduction

XML[3] is very popular language which is widely used to encode a variety of data. When used to encode a particular kind of document, the XML syntax is usually constrained using XML grammar. There are several ways to define such grammar. The DTD is a part of the original XML specification, but is not widely used nowadays due to it's shortcomings. XML Schema, published by W3C, is an alternative to DTD. Another widely used alternative schema language RELAXNG[8] was defined by the Organization for the Advancement of Structured Information Standards (OASIS). In our project, we will use RELAXNG, mainly because of its good tool chain support and our familiarity with it.

In practice, XML-based domain specific languages are often created ad-hoc foregoing the step of formal schema definition. In many cases, the original schema becomes obsolete because of the language evolution. Lack of up to date schema makes it difficult to ensure document validity. This was the motivation for this project. Our hypothesis was that by examining a sufficiently large corpus of homogeneous XML documents, we should be able to automatically derive and describe RELAXNG schema. Given the expressiveness of the schema language, this turns out to be a non-convex optimization problem with a very large search space, which makes it a good candidate for applying EA algorithms.

2 Background

Evolutionary algorithms have been used in conjunction with formal grammars before. In a survey[5] McKay, et. al traces the relationship of EA and formal grammars to the early 1990s. They broadly

define this area as Grammar Guided Genetic Programming (GGGP). The most popular part of it is Genetic Programming (GP).

Grammar induction using EA could be considered as a special case of GP. In this case the grammar itself is a program in a declarative grammar language. The main difference is that it is not executed in the usual way as an imperative program is executed to produce a result in GP, but instead it is used to validate a document.

According to [1], "The use of GA for grammatical inference is still a small field." For example, the EA grammar induction has been applied to both natural[7] as well as artificial toy[6] languages. To our knowledge, it has not been applied so far to SGML (or XML) based languages.

3 Algorithm and Analysis

3.1 Data collection

The particular XML format we have chosen for our experiments is the Rich Site Syndication (RSS) format. This format is widely used by web sites to publish regularly updated "feeds" of news items for the purpose of syndication. There are two major types of this format, RSS 1.0 and RSS 2.0. The key difference is that RSS 1.0 is RDF-based, which adds another layer of complexity: RSS 1.0 is expressed in RDF, which could be described by RDF schema. RDF in turn is an XML-based language, which could be described by XML schema[1]. To avoid one extra level of syntax (RDF), we decided to start with the simpler and more popular RSS 2.0 format.

We started by collecting a dataset consisting of 48 RSS documents. The documents were fetched using a simple Python script, which was given as input a list of URLs. The script performed a "sanity check" ensuring that each fetched document was in XML format and the top-level element specifed that it was indeed RSS 2.0.

For each document, we generated a "naïve" RNG schema. Such schema describes the exact structure of the particular RSS document, including exact node names and their order and quantities. It was guaranteed to validate the original document but not guaranteed to validate any other RSS documents unless they had the exact same structure. Each *individual* in the original EA population was such a naïve RNG schema. The population size was 48 individuals – one per sample RSS document. The goal of the EA was to evolve a schema which validates as many documents as possible and has minimal size.

3.2 Representation

RNG schema is an XML document itself and could be represented as a tree. We used XML trees of RNG schema as individual representations. In memory, they were Python data structures, produced by an *lxml* parsing module. When they had to be printed on screen or written to a file, they were serialized as RNG schema documents using XML syntax[2].

Simplified example of RSS document:

```
<rss version="2.0">
    <channel>
        <title>W3Schools Home Page</title>
        <link>http://www.w3schools.com</link>
        <description>Free web building tutorials</description>
        <item>
            <title>RSS Tutorial</title>
            <link>http://www.w3schools.com/rss</link>
            <description>Item 1 description</description>
        </item>
        <item>
            <title>XML Tutorial</title>
```

[1] To go even further, XML is a profile (subset) of SGML

[2] RNG also supports "simple" syntax, which we did not use in this project

```
            <link>http://www.w3schools.com/xml</link>
            <description>Item 2 description</description>
        </item>
    </channel>
</rss>
```

A simplified RNG RSS schema fragment is shown below. For comparison, we have also developed a full RSS RNG schema, which is included in Appendix A.

```
<element xmlns="http://relaxng.org/ns/structure/1.0" name="rss">
    <element name="channel">
        <interleave>
            <element name="title"><text/></element>
            <element name="link"><text/></element>
            <element name="description"><text/></element>
        </interleave>
        <zeroOrMore>
            <element name="item">
                <interleave>
                    <element name="title"><text/></element>
                    <element name="description"><text/></element>
                    <optional>
                        <element name="author"><text/></element>
                    </optional>
                </interleave>
            </element>
        </zeroOrMore>
    </element>
</element>
```

In the example above, *element* nodes are used to match nodes bearing name given by *name* attribute in the XML document. We will call RNG constructs *zeroOrMore*, *optional*, and *oneOrMore* qualifiers. They specify how many instances of an enclosed node were allowed to occur. Without a qualifier, exactly one node is expected. Finally *interleave* specifies that the enclosed elements could occur in any order. Without it, a sequence of elements is matched in the exact order they are specified in RNG schema. This is a simplified explanation of basic RNG constructs and interested readers are referred to [8] for more details.

3.3 Algorithm Outline

Below is a brief outline of the algorithms. Each part will be described in more detail in corresponding later sections.

Both crossover and mutation were used with respective probabilities 0.1 and 0.9. Several mutation operators were used with different probabilities. Some of them had fixed probabilities, while others had probabilities which depended on population statistics. Probabilities of mutation operators are shown in the Table 1 where v denotes a fraction of the total documents validated by the best individual:

$$v = \frac{\sum_{d \in D} \mathrm{validate}(d, \mathrm{best\_individual})}{|D|}, \tag{1}$$

where $best\_individual$ is the best-fit individual in the current population, D is a corpus of test RSS documents, and $\mathrm{validate(d, s)}$ is a validation predicate which returns one if the document d successfully validates using given schema s and zero otherwise.

Our motivation for selecting these particular expressions for operator probabilities will be given later in this report after definition of the respective operators.

Operator	Probability
Adding Qualifier	$0.4 * (1 - v)$
Removing Qualifier	$0.2 * v$
Adding *interleave*	0.15
Removing *interleave*	0.05
Removing Node	$0.2 * v$
Copying Node	0.1
Merging Trees	0.3

Table 1: Mutation operator probabilities

For parent selection, the *Stochastic Universal Sampling* (SUS) algorithm was used. To avoid the loss of genetic material, a custom survivor selection algorithm was used.

3.4 Fitness Criteria

Our algorithm has two optimization objectives: maximize the number of validated documents (the main objective) and minimize the schema size (secondary objective). The first objective is the most important. The algorithm should never sacrifice test set coverage for schema size. The objectives were weighted accordingly using the following fitness function:

$$f(i) = \sum_{d \in D} \mathrm{validate}(d, i) + \left(1 - \frac{\mathrm{size}(i)}{\max_{x \in P} \mathrm{size}(x)}\right), \tag{2}$$

where i is an individual being evaluated, D is a corpus of test RSS documents, P is the current population, and $\mathrm{validate(d, s)}$ is a validation predicate which returns one if the document d successfully validates using given schema s and zero otherwise.

3.5 Survivor Selection

We used a custom variant of the fitness-based survivor selection algorithm with elitism. The initial selection pool was the results of the mutation and crossover operators application combined with the previous generation ($\mu + \lambda$). The new generation was chosen from this pool using the following rules:

Elitism Top 20% of population, the best fit individuals are always preserved.

Fitness A fixed number (currently 48) of individuals is randomly selected with probabilities proportional to their relative fitness using the *roulette wheel* algorithm.

Diversity In order to preserve all genes, the following additional step is performed. From the individuals already selected to be included in the new generation during the two previous steps, a set B of individual genes is built. Additionally, a set A of **all** the genes present in the candidate pool is composed. A relative complement of A with respect to B gives us a set of a potentially lost genes $M = A \setminus B$. To avoid their irrecoverable loss, additional individuals containing these genes are added to the new generation. If more than one individual containing a particular gene is available, only one is selected using fitness-proportional *roulette wheel* selection.

3.6 Crossover

We implemented uniform crossover as follows. Two individuals were selected from the population and part of them were swapped. Initially, we swapped random parts of their trees, but it quickly become apparent that this was very inefficient as RSS schema has a regular structure and it does not make sense, for example, to swap *channel* and *item* nodes. To address this, we modified the crossover to swap nodes which were located at the same depths in their respective XML trees.

The second refinement of this operator was to correlate the probability of selecting a node in an XML document with the length of path from it to the root of the tree (it's depth). The rationale was

to control the average amount of genetic information exchanged. Swapping big sub-trees causes transfer of all nodes below them. Making nodes located closer to the top of the tree swap less frequently than nodes located deeper in the tree ensured more even information exchange. We call this XML node selection method *depth-proportional selection* since we use it in other operators.

We found that the effect of the crossover operator on algorithm performance was fairly low but not negligible. Thus, it was assigned a low but non-zero probability.

3.7 Adding or Removing a Qualifier

This operator adds or removes a random qualifier to a randomly selected schema *element*. Initially we used all three qualifiers but later decided to simplify and use only *zeroOrMore* as the two others, (*optional* and *oneOrMore*), constitute *syntactic sugar* and were not strictly necessary to build a working schema.

The qualifiers are exclusive, so we never add a qualifier on top of another qualifier. Elements to which qualifiers are added or from which they are removed are selected at random.

Adding a qualifier generalizes schema, while removing it makes it more specific. In the early stages of EA execution, we want to generalize schema as much as possible to validate the greatest number of the documents. Once it is "saturated" with the qualifiers and can not be generalized anymore, parsimony pressure is used to make it more specific and to shrink it, without losing the ability to validate all of the already validated documents. This logic was implemented by assigning probabilities of adding and removing the qualifier depending on how many documents were successfully validated by the current population.

3.8 Adding or Removing *interleave*

Adding *interleave* to a group of nodes which do not have one yet relaxes ordering restriction. Removing it, on the other hand, enforces it. Most XML documents, including RSS 2.0, do not rely on element order. In view of this, the probability of removing *interleave* could be lowered.

Unlike a qualifier, *interleave* is added as a child of a randomly selected element, and all existing children of the selected element are absorbed into it.

3.9 Removing Node

This operator randomly selects any *element* node (along with its qualifiers) and removes it from the document. In some cases, this could produce an invalid RNG schema, so the operator checks whenever the resulting schema is valid. If the schema is not valid, the operator leaves the individual unchanged. Initially, we implemented a retry policy to handle invalid schema situations, but this caused significant performance degradation in cases where the schema contained very few elements which could be safely removed.

The rationale of this operator is to remove redundant schema elements. When combining two schemata, some elements could become redundant, for example, resulting in $\langle title \rangle$ followed by $\langle zeroOrMore \rangle \langle title \rangle$. While this schema will correctly validate RSS documents (where the *title* must occur at most once), the first one is redundant and could be removed to decrease the schema size.

Another scenario where this operator is useful is the removal of out of place elements. An element placed at wrong place could prevent schema from validating documents, and this operator could correct this.

3.10 Copying Node

This operator is similar to crossover, but it is a mutation and as such, modifies only one individual. The first individual (the target) is selected using regular fitness-proportional parent selection. The second one (the source) is selected at random, without taking into account its fitness. The rationale for that is that useful genetic material can linger in otherwise unfit nodes. We initially implemented fitness-based selection for both nodes, but modified it to the above-described method later, which

lead to improved algorithm performance. The source and target were selected with replacement, so they could be the same, in which case some nodes could be copied inside an individual.

As in crossover, the sub-trees are selected using *depth-proportional selection*, and the elements are copied at the same level.

This operator is the main way of copying genetic material between the individuals.

3.11 Merging Trees

This operator is similar to *Copy Node*, but instead of copying a single element, it merges the content of two nodes with the same name and located at the same level from two individuals. Only the main document is updated with the merged content.

Strictly speaking, this operator is just an optimization of *Copy Node*. The same result could be achieved by copying elements one by one using *Copy Node*, but we observed that often it was more efficient to copy already partially evolved node content in one step.

3.12 Implementation

The algorithm was implemented in Python programming language running under MacOS and Linux. The implementation was done from scratch, without using any existing EA libraries. XML parsing and schema validation was performed using Python's *lxml* module.

One of the challenges was the performance. Schema validation was taking most of the CPU time. From our previous experience in implementing DTD parser, we could speculate that the RNG validator probably works in a similar way by compiling the schema to an NFA which is then is converted to a DFA and the latter is applied to the token stream from an XML document. Some RNG constructs could lead to DFA state explosion and even a simple schema could produce a huge DFA. This could lead to slow schema validation.

Even though the *lxml* RNG validator was implemented in C, it was still fairly slow. Python's GIL limitations[2] prevented us from using multiple CPU cores via multithreading as we would normally do in other languages like Java. To make use of multiple CPU cores from Python, we used a *multiprocessing* module to fork multiple processes and to distribute the application of mutation operators between them.

4 Experiments

For our experiments, we used an 8-core machine running XEN hypervisor and two 4-core VMs. This allowed us to run two experiments simultaneously using different parameters (e.g. operators' probabilities). On 4 CPU cores, each generation took between 2 and 15 seconds which limited the number of the experiments we could perform.

By trial and error, we arrived at some insights, which helped us to improve the performance of the algorithm. The insights below are representative but not exhaustive.

- Some operators could produce invalid schema.
- Retrying operators which succeed with very low probability could impact performance.
- Swapping elements taken from different depths is not a good idea.
- Copying elements at different depths transfers different amounts of genetic material.
- In a small population, it is easy to lose some genetic material which could no longer be recovered. Special caution should be taken to avoid losing it.
- Schema could become "saturated" with qualifiers and attempts to add new ones will fail.
- We found it easier to guide the algorithm to optimize the main objective first and then apply parsimony pressure (the secondary objective).

5 Results, Conclusions, and Future Directions

5.1 Results

Figure 1 shows a plot of fitness for a typical run of the algorithm.

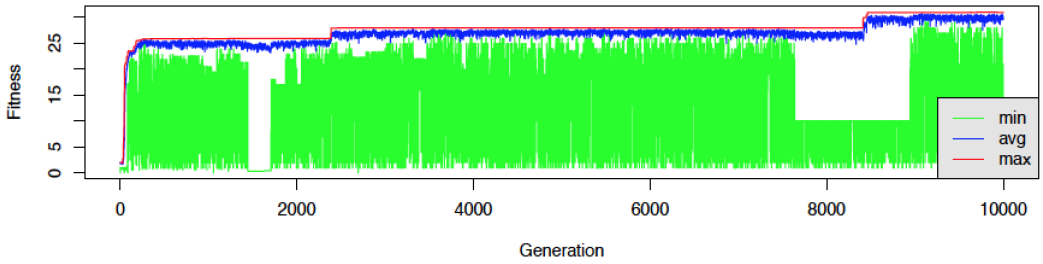

Figure 1: Best run to date

Due to project time constraints, we were unable to average several runs. The algorithm was adjusted until the project deadline and each time it had to be re-run from the beginning, which took a significant amount of time. Our last run, was able to validate 30 out of 48 documents. It was shown to perform a meaningful search with results improving with the number of generations.

The best resulting schema produced by the algorithm is enclosed in Appendix B. Upon examination, it seems to be a good approximation of RSS schema. It has the right structure and right elements in the right places. It even found some more subtle constraints, such as the fact that element *title* in channel is not optional, while element *copyright* is optional. The main reason for not validating all the documents was that it was still missing some less frequently used elements which were in the population but it needed more generations for them to get consolidated into the best individual. The size of the generated schema was comparable to a manually written one.

We can conclude that the approach of deriving a schema from a corpus of XML documents is proven to be viable, and the project was successful in achieving the stated goals.

5.2 Future Directions

Because of the time and resource constraints of this project, there were a few things we did not have the chance to try or to complete. First of all, we would have liked to run the final version of the algorithm until it converged to the shortest possible schema which would validate all the training documents.

As the next step, we could enable more qualifiers (*optional* and *oneOrMore*), which could potentially improve expressiveness and the size of the schema.

Presently, we chose not to validate element attributes. They are described and validated in a very similar manner to elements, and our approach could be trivially extended to attributes as well. We chose not to do this yet because adding attributes would significantly increase the search space and consequently the search time. Provided with more time and computational power, it would be interesting to enable attribute validation.

From linguistics and cognitive sciences, we know from famous Gold's theorem[4] that a language could not be learned using only samples of correct utterances. We can consider extending our approach by using both *positive* and *negative* test samples. The fitness function would then need to maximize the number of positive samples validated and at the same time minimize the number of negative ones passing the validation.

Finally, it would be interesting to see how this approach would generalizes to other types of XML documents beyond RSS.

References

[1] Representational issues for context free grammar induction using genetic algorithms. In *ICGI '94 Proc. Second Int. Colloq. Gramm. Inference Appl.* (1994), vol. 862, pp. 222–235.

[2] BEAZLEY, D. Understanding the python gil. In *PyCON Python Conference. Atlanta, Georgia* (2010).

[3] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M., MALER, E., AND YERGEAU, F. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210. http://www. w3. org/TR/1998/REC-xml-19980210* (1998).

[4] JOHNSON, K. Gold's Theorem and Cognitive Science. *Philos. Sci. 71*, October (2004), 571–592.

[5] MCKAY, R. I., HOAI, N. X., WHIGHAM, P. A., SHAN, Y., AND O'NEILL, M. Grammar-based Genetic Programming: a survey. *Genet. Program. Evolvable Mach. 11*, 3-4 (May 2010), 365–396.

[6] PANDEY, H. M. Context free grammar induction library using Genetic Algorithms. *2010 Int. Conf. Comput. Commun. Technol.* (Sept. 2010), 752–758.

[7] UNOLD, O. Context-free grammar induction using evolutionary methods. In *WSEAS AIC-ISTASC-ISCGAV* (2003).

[8] VAN DER VLIST, E. *Relax Ng*. O'Reilly Media, Inc., 2003.

6 Appendices

A Hand-coded RSS 2.0 RNS Schema

```
<!--
     RSS 2.0 RELAX NG schema
     Specification: http://cyber.law.harvard.edu/rss/rss.html

     This is a quick and dirty attempt on RSS 2.0 RNG schema.
     While it includes all elements and attributes described in the
     spec, some data types needs to be specified more precisely
     using XSD data type library.

     TODO:
     * RNG 'list' should be used for 'category'.
     * Need to limit max number of sub-items and their range in skip*
     elements
     * All date elements may need to use date type

     Author: Vadim Zaliva <lord@crocodile.org>
     -->

     <grammar xmlns="http://relaxng.org/ns/structure/1.0"
              datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

       <define name='categories'>
         <zeroOrMore><element name="category">
           <optional><attribute name="domain"/></optional>
         <text/></element></zeroOrMore>
       </define>

       <define name='extensionNSElement'>
         <zeroOrMore>
           <choice>
             <text/>
             <element>
               <anyName>
                 <except>
                   <nsName/>
                 </except>
               </anyName>
               <ref name='extensionNSElement'/>
               <zeroOrMore>
                 <attribute>
                   <anyName/>
                 </attribute>
               </zeroOrMore>
             </element>
           </choice>
         </zeroOrMore>
       </define>

       <define name='linkElement'>
         <element name="link">
```

```
          <!--
               Strictly speaking the following type should be used, but
               some URLs in our test dataset fail to validate, so we will
               relax this datatype restriction and treat it as text.

<data type="anyURI"/>
          -->
          <text/>
        </element>
      </define>

      <start>
        <element xmlns="http://relaxng.org/ns/structure/1.0" name="rss">
          <interleave>
            <attribute name="version">
              <value>2.0</value>
            </attribute>
            <zeroOrMore>
              <attribute>
                <anyName>
                  <except><nsName/></except>
                </anyName>
              </attribute>
            </zeroOrMore>
          </interleave>

          <element name="channel">
            <interleave>
              <element name="title"><text/></element>
              <ref name='linkElement'/>
              <element name="description"><text/></element>

              <optional><element name="language">
                <data type="language"/>
              </element></optional>
              <optional><element name="copyright"><text/></element></optional>
              <optional><element name="managingEditor"><text/></element></optional>
              <optional><element name="webMaster"><text/></element></optional>
              <optional><element name="pubDate"><text/></element></optional>
              <optional><element name="lastBuildDate"><text/></element></optional>
              <optional><element name="generator"><text/></element></optional>
              <optional><element name="docs">
                <data type="anyURI"/>
              </element></optional>
              <optional><element name="cloud">
                <interleave>
                  <attribute name="domain"/>
                  <attribute name="port">
                    <data type="integer"/>
                  </attribute>
                  <attribute name="path"/>
                  <attribute name="registerProcedure"/>
                  <attribute name="protocol"/>
                </interleave>
              </element></optional>
              <optional><element name="ttl">
                <data type="integer"/>
              </element></optional>
              <optional><element name="rating"><text/></element></optional>

              <optional><element name="textInput">
                <interleave>
                  <attribute name="title"/>
                  <attribute name="description"/>
                  <attribute name="name"/>
                  <ref name='linkElement'/>
                </interleave>
              </element></optional>

              <optional><element name="image">
                <interleave>
                  <element name="url">
                    <data type="anyURI"/>
                  </element>
                  <element name="title"><text/></element>
                  <ref name='linkElement'/>
                  <optional><element name="description"><text/></element></optional>
                  <optional><element name="width">
                    <data type="integer"/>
                  </element></optional>
                  <optional><element name="height">
                    <data type="integer"/>
                  </element></optional>
                </interleave>
              </element></optional>

              <optional><element name="skipHours">
                <zeroOrMore>
                  <element name="hour">
                    <data type="integer"/>
                  </element>
                </zeroOrMore>
```

```
            </element></optional>

            <optional><element name="skipDays">
              <zeroOrMore>
                <element name="day">
                  <data type="integer"/>
                </element>
              </zeroOrMore>
            </element></optional>

            <ref name='categories'/>

            <optional>
              <ref name='extensionNSElement'/>
            </optional>

          </interleave>
          <zeroOrMore>
            <element name="item">
              <interleave>
                <!-- At least one of title,group should be present -->
                <choice>
                  <group>
                    <element name="title"><text/></element>
                    <element name="description"><text/></element>
                  </group>
                  <element name="title"><text/></element>
                  <element name="description"><text/></element>
                </choice>
                <!-- Item's optional elements below -->
                <optional><element name="guid">
                  <optional><attribute name="isPermaLink"/></optional>
                  <text/>
                </element></optional>
                <optional><element name="pubDate"><text/></element></optional>
                <optional>
                  <ref name='linkElement'/>
                </optional>
                <optional><element name="author"><text/></element></optional>

                <ref name='categories'/>

                <optional><element name="comments">
                  <data type="anyURI"/>
                </element></optional>

                <optional><element name="enclosure">
                  <interleave>
                    <attribute name="url">
                      <data type="anyURI"/>
                    </attribute>
                    <attribute name="length"/>
                    <attribute name="type"/>
                  </interleave>
                <text/></element></optional>

                <optional><element name="source">
                  <attribute name="url">
                    <data type="anyURI"/>
                  </attribute>
                </element></optional>

                <optional>
                  <ref name='extensionNSElement'/>
                </optional>

              </interleave>
            </element>
          </zeroOrMore>
        </element>
      </element>
    </start>
  </grammar>
```

B Best individual produced by the algorithm

```
<element xmlns="http://relaxng.org/ns/structure/1.0" name="rss">
  <zeroOrMore>
    <attribute>
      <anyName/>
    </attribute>
  </zeroOrMore>
  <element name="channel">
    <interleave>
      <zeroOrMore>
        <attribute>
          <anyName/>
        </attribute>
```

```
</zeroOrMore>
<element name="title">
  <text/>
  <zeroOrMore>
    <attribute>
      <anyName/>
    </attribute>
  </zeroOrMore>
</element>
<zeroOrMore>
  <element name="copyright">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="link">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<element name="description">
  <text/>
  <zeroOrMore>
    <attribute>
      <anyName/>
    </attribute>
  </zeroOrMore>
</element>
<zeroOrMore>
  <element name="language">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="lastBuildDate">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="info" ns="http://rssnamespace.org/feedburner/ext/1.0">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <empty/>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="link" ns="http://www.w3.org/2005/Atom">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <empty/>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="treatAs" ns="http://www.microsoft.com/schemas/rss/core/2005">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="image">
    <interleave>
      <zeroOrMore>
        <attribute>
```

```
          <anyName/>
        </attribute>
      </zeroOrMore>
      <element name="url">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
      <element name="title">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
      <element name="link">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
    </interleave>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="item">
    <interleave>
      <zeroOrMore>
        <attribute>
          <anyName/>
        </attribute>
      </zeroOrMore>
      <element name="title">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
      <zeroOrMore>
        <element name="description">
          <text/>
          <zeroOrMore>
            <attribute>
              <anyName/>
            </attribute>
          </zeroOrMore>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="guid">
          <text/>
          <zeroOrMore>
            <attribute>
              <anyName/>
            </attribute>
          </zeroOrMore>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="pubDate">
          <text/>
          <zeroOrMore>
            <attribute>
              <anyName/>
            </attribute>
          </zeroOrMore>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="Location">
          <text/>
          <zeroOrMore>
            <attribute>
              <anyName/>
            </attribute>
          </zeroOrMore>
        </element>
      </zeroOrMore>
      <zeroOrMore>
        <element name="MuseumAddress">
          <text/>
          <zeroOrMore>
            <attribute>
              <anyName/>
```

```
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="Museum">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="OpeningDate">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="creator" ns="http://purl.org/dc/elements/1.1/">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="origLink" ns="http://rssnamespace.org/feedburner/ext/1.0">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="author">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="category">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="enclosure">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <empty/>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="date" ns="http://purl.org/dc/elements/1.1/">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="CloseText">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
```

```
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="link">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="subject" ns="http://purl.org/dc/elements/1.1/">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="comments">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="thumbnail" ns="http://search.yahoo.com/mrss/">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <empty/>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="ClosingDate">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="encoded" ns="http://purl.org/rss/1.0/modules/content/">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="link" ns="http://www.w3.org/2005/Atom">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <empty/>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="MuseumCity">
    <text/>
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
  </element>
</zeroOrMore>
<zeroOrMore>
  <element name="content" ns="http://search.yahoo.com/mrss/">
    <zeroOrMore>
      <attribute>
        <anyName/>
      </attribute>
    </zeroOrMore>
    <element name="thumbnail" ns="http://search.yahoo.com/mrss/">
      <zeroOrMore>
        <attribute>
```

```
                    <anyName/>
                  </attribute>
                </zeroOrMore>
                <empty/>
              </element>
            </element>
          </zeroOrMore>
        </interleave>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="pubDate">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="ttl">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="category">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
    </zeroOrMore>
    <zeroOrMore>
      <element name="generator">
        <text/>
        <zeroOrMore>
          <attribute>
            <anyName/>
          </attribute>
        </zeroOrMore>
      </element>
    </zeroOrMore>
  </interleave>
 </element>
</element>
```