ToXgene: An extensible template-based data generator for XML*

Denilson Barbosa[1] Alberto O. Mendelzon[1] John Keenleyside[2] Kelly Lyons[2]

[1] Department of Computer Science
University of Toronto
{dmb,mendel}@db.toronto.edu

[2] IBM Toronto Lab
{keenley,klyons}@ca.ibm.com

Abstract. Synthetic collections of XML documents are useful in many applications, such as benchmarking (e.g., Xmark), and algorithm testing and evaluation. We present ToXgene, a template-based generator for large, consistent collections of synthetic XML documents. Templates are annotated XML Schema specifications describing both the structure and the content of the data to be generated. Our tool was designed to be declarative, and general enough to generate complex XML content and to capture most common requirements, such as those embodied in current benchmarks. In the paper, we give an overview of the ToXgene template specification language and the extensibility of our tool; we also report preliminary experiments with ToXgene carried out at the IBM Toronto Lab, which show that our tool can closely reproduce the data sets of the TPC-H and Xmark benchmarks.

1 Introduction

Synthetic collections of XML documents have many applications in benchmarking and testing algorithms, tools and systems. Moreover, different applications require documents with different complexities, sizes, etc. For example, a benchmark for data-intensive applications might be a large and relatively homogeneous document, with many references among elements, while a test suite for a parser might be an heterogeneous collection with thousand of documents. Given the complexity of writing and/or customizing hard-coded data generators for specific scenarios, we believe a declarative tool for generating synthetic XML documents will prove useful.

We present ToXgene, a template-based tool for generating large, consistent synthetic collections of complex XML documents. ToXgene's template specification language is a subset of the XML Schema [5] notation augmented with annotations for specifying other properties of the intended data, such as probability of occurrences of elements, the vocabulary of `CDATA` content, etc. This work is part of the ToX project [2], recently started at the University of Toronto.

1.1 Related work

A general purpose generator for synthetic XML documents is presented in [1]. Our work differs from that in the following ways. First, our data generation process is centered on a conceptual description of the desired data (a template). ToXgene gives the user total control over the data to be generated, unlike the method in [1], where both the structure and the content of the documents are randomly generated. We note that our tool allows some controlled randomness in its output and can generate documents with irregular structure, as shown in Section 2.4. Second, our tool generates more complex XML content, including elements with mixed content; attributes; non-gibberish text; and different numerical and date values. Finally, ToXgene supports different probability distributions.

Another template-based XML data generator is the IBM XML Generator [7]; its inputs are annotated Document Type Definitions (DTDs) specifying the structure and the characteristics of the data. There are many limitations to that tool, however. For instance, it allows one to *limit* the maximum depth of the document tree, or the number of `ID` and `IDREF` attributes in the documents, but it does not allow one to *specify* actual values for these properties. Moreover, unlike ToXgene, that tool does not support different probabilities of occurrence on a *per element* basis, nor the generation of `CDATA` content of different datatypes (e.g., strings, dates, etc.). On the other hand, the IBM generator deals with XML constructs that are not addressed in ToXgene, such as `ENTITY` declarations and processing instructions [4].

Furthermore, ToXgene differs from both approaches above in the following ways. First, ToXgene supports element sharing, i.e., the values of some elements (or attributes) can be shared among different elements (or attributes) in the same (or in different) XML documents. This allows the generation of collections of corelated documents (i.e., documents that can be *joined* by value). Second, ToXgene can produce data conforming to user-specified integrity constraints, which allows the generation of consistent references within and across documents. Third, ToXgene allows the use of existing data when generating documents; therefore, one can grow an existing collection of documents while maintaining its consistency, or mix real and

*This work is supported by the National Science and Engineering Research Council of Canada, Bell University Laboratories, and the IBM Centre for Advanced Studies.

```
<tox-distribution name="c1"
  type="exponential" minInclusive="5"
  maxInclusive="100" mean="35">
```

(a) Specification of an exponential probability distribution.

```
<simpleType name="isbn_type">
  <restriction base="string">
    <pattern value="[0-9]{10}"/>
  </restriction>
</simpleType>

<simpleType name="my_float">
  <restriction base="float">
    <tox-number tox-distribution="c1"/>
  </restriction>
</simpleType>
```

(b) Specification of a `simpleType`.

```
<complexType name="my_book">
  <attribute name="isbn" type="isbn_type"/>
  <element name="title" type="string">
    <tox-string type="text" maxLength="30"/>
  </element>
  <element name="author" type="my_author"
    minOccurs="1" maxOccurs="5"/>
  <element name="price">
    <complexType mixed="true">
      <attribute name="currency">
        <simpleType>
          <restriction base="string">
            <tox-expr value="'CDN'"/>
          </restriction>
        </simpleType>
      </attribute>
      <tox-number minInclusive="5"
      maxInclusive="100"/>
    </complexType>
  </element>
</complexType>
```

(c) A `complexType` specification.

Figure 1: Notation for specifying types, genes and probability distributions in ToXgene.

synthetic data in the generation process (e.g., use real country names as in Xmark). Finally, our tool can be easily extended to allow the generation of domain-specific content, whenever the built-in tools are insufficient.

Annotated XML Schema specifications have also been used for defining relational storage methods for XML documents [3].

The remainder of the paper is organized as follows. Next section gives an overview of the template specification language and illustrates the kinds of documents ToXgene can produce. Section 3 discusses the extensibility mechanisms of ToXgene. Section 4 reports on experiments carried out with ToXgene. Finally, Section 5 concludes the paper.

2 ToXgene template specification language

Among the various languages for specifying XML content, we chose XML Schema as the basis for our template language for two reasons: it is a W3C standard, thus it is expected to become familiar to XML practitioners; and it allows a more detailed description of XML content than DTDs. In particular, it allows the specification of types, for describing literals (i.e., `CDATA` content), elements and attributes. However, we note that having an XML Schema specification alone is not enough for generating useful synthetic data; at the very least, we need to annotate the schema for specifying which type should be used as the type for the root element of the documents to be generated.

We also define annotations for other purposes, such as specifying probability distributions of occurrences of elements and attributes; defining element sharing; defining integrity constraints over XML elements; and providing some controlled randomness in the structure of the data to be generated. More details on the ToXgene template language can be found in [9].

Types and genes. A *type* is a specification of some valid XML content, and can be either a `simpleType` or a `complexType` [5]; *instances* of a `simpleType` are `CDATA` literals, while instances of a `complexType` are either XML elements, `CDATA` literals, or a mix of both.

A *gene* is a specification of either an element (an *element* gene) or an attribute (an *attribute* gene), and contains a name and a type; an *instance* of an element gene with name n and type t is an element whose opening and closing tags are labeled with n and whose content is an instance of t; similarly, an *instance* of an attribute gene with name n and type t is an attribute whose name is n and whose content is an instance of t. An attribute gene can only be declared within a `complexType`, and its type must be a `simpleType`.

2.1 Specifying probability distributions, types and genes

Figure 1 contains the examples we use to illustrate the discussion in this section. All ToXgene-specific annotations are prefixed by the keyword `tox` and presented in a different font.

Specifying probability distributions. A probability distribution is declared using the **tox-distribution** annotation, as shown in Figure 1(a), and referenced via its name, as shown in Figure 1(b). A single template might specify multiple probability distributions, which can be used to determine, for example, the number of occurrence of elements and attributes, the length of string literals, and instances of numerical types, as in Figure 1(b). ToXgene currently supports the uniform, normal, exponential and log-normal distributions, as well as arbitrary discrete distributions, where the user provides all possible outcomes together with their respective probabilities of occurrence. In order to allow the static checking of the consistency of the templates,

```
<tox-list name="book_list" unique="[book/isbn]">
  <element name="b_rec" minOccurs="200">
    <element name="isbn" type="isbn_type"/>
    <element name="author_id" type="integer"
      maxOccurs="5">
      <tox-sample path="[author_list/author]"
        duplicates="no">
        <tox-expr value="[id/!]"/>
      </tox-sample>
    </element>
  </element>
</tox-list>
```

(a) Declaration of a list with 200 book records. Each record has a unique ISBN value and up to 5 `author_id` values, sampled (without repetition) from a list of authors, declared elsewhere.

```
<element name="book" minOccurs="200">
  <complexType>
    <tox-scan path="[book_list/b_rec]" name="a">
      <attribute name="isbn" type="isbn_type">
        <tox-expr value="[isbn/!]"/>
      </attribute>
      <element name="title" type="string"/>
      <attribute name="authors" type="IDREFS"
        tox-maxOccurs="unbounded">
        <tox-scan path="[$a/author_id]">
          <tox-expr value="'author'#[!]"/>
        </tox-scan>
      </attribute>
    </tox-scan>
  </complexType>
</element>
```

(b) Specifying queries over lists; the symbol # in the expression corresponds to the string concatenation operation.

Figure 2: Defining and querying lists.

we require each probability distribution to have a maximum and a minimum value; all values outside this interval have null probability of occurrence.

Specifying `simpleType`s. A `simpleType` is a specialization of a *base* type, e.g., string, integer, etc.; the *domain* of a `simpleType` is a subset of the domain of the base type it is built upon; and an instance of a `simpleType` is an element chosen from its domain. For example, the `isbn_type`, specified in Figure 1(b), is a specialization of the string base type; its domain is the set of strings that conform to the given pattern; and `1234567890` is one of its instances. `simpleType`s are the basic content specification tools for defining genes.

The **tox-number**, **tox-string** and **tox-date** annotations are used to refine a `SimpleType` definition (e.g., to specify that instances of `my_float` in Figure 1(b) obey the probability distribution **c1**, and that the content of instances of the `title` element in Figure 1(c) are non-gibberish strings[1], no more than 30 characters long), or to specify the generation of elements with mixed content (e.g., the `price` element in Figure 1(c)). Constants can be specified using the **tox-expr** annotation.

Specifying `complexType`s and genes. A `complexType` specification (see Figure 1(c)) contains definitions of element genes, attribute genes, or annotations defining `CDATA` literals, required for defining elements of mixed content. Both named and anonymous types are allowed in ToXgene, as shown in the figure. Our tool supports all element content models defined in [4]: character data, as in the `isbn` element; elements, as in the `book` element; and mixed, as in the `price` element.

2.2 Specifying element sharing and integrity constraints in ToXgene

ToXgene allows *element sharing* both within and across documents; i.e., different elements (or attributes), in the same or in different documents, can have the same `CDATA` content. This allows the generation of collections of *correlated* documents (i.e., documents that can be *joined* by value). In our tool, element sharing is achieved by generating all shared content prior to generating any documents. The shared content is kept in what we call **tox-list**s; such lists are queried at document generation time. Integrity constraints over the contents of a list, such as uniqueness of certain values, can be specified, thus ensuring the consistency of the collections.

Specifying lists. Lists are declared by **tox-list** annotations. Each list has a unique name, an element gene that defines its contents, and, optionally, some integrity constraints. Figure 2(a) shows the specification of a list of books, each containing an ISBN and up to 5 references to author elements, stored in another list. The number of elements in a list is determined by the gene defining it (exactly 200 in the example in the figure). The **unique** constraint in the list in Figure 2(a) ensures that no duplicate ISBN values are stored in the list; more complex integrity constraints can be specified using *where clauses* (see [9] for details).

Querying lists. ToXgene has a language for specifying expressions which are declared using **tox-expr** annotations, and are evaluated at content generation time. This language allows arithmetic and string operations; operands can be the results of queries, instances of `simpleType`s generated "on-the-fly", or constants. All expressions are typed; type checking and casting mechanisms are implemented. Expressions are useful for defining conditional instantiation of genes (see Section 2.4), and for specifying where clauses for lists and selection conditions on cursors (see below). Due to space limitations, we describe expressions defining queries only; for details, see [9].

The contents of a list are retrieved using *cursors* and *queries*, both of which are specified using *path expressions*. A path

[1]Our tool supports two built-in string types: **gibberish**, and **text**, as defined in [10]. However, ToXgene can produce strings using a different vocabulary, as discussed in Section 4.

```
<tox-document name="books" copies="1"
  <complexType>
    <element name="books">
      <complexType>
        <element name="book" type="book"
          minOccurs="200" maxOccurs="200">
          <tox-scan path="[book_list/b_rec]">
            ...
      </complexType>
    </element>
  <complexType>
</tox-document>
```

(a) Single file, multiple `book` elements.

```
<tox-document name="book" copies="200"
  <complexType>
    <element name="book" type="book">
      <tox-scan path="[book_list/b_rec]">
      ...
    </element>
  <complexType>
</tox-document>
```

(b) Collection of files, each having a single `book` element.

Figure 3: Declaring single documents and collections of documents in ToXgene.

expression is a sequence of *names* separated by '`/`'; a name can be one of: a list name, an element name, a cursor name, or `!`, which is shorthand for `CDATA`. All path expressions are enclosed by square brackets. The elements over which a cursor iterates are specified by a path expression rooted at a list or at another cursor, and, optionally, a selection condition. ToXgene provides cursors for sequential scans (using the **tox-scan** annotation), and sampling with or without repetition (using the **tox-sample** annotation). Different cursors, possibly using different access patterns, might iterate over the same content (thus allowing element sharing).

A query is an expression that extracts the actual `CDATA` content of the elements in a cursor, and is always evaluated against the current element of the cursor it refers to. By default, a query refers to its closest ancestor cursor declaration; an explicit reference to a different cursor can be made by starting the query's path expressions with that cursor's name (see Figure 2(b)).

Cursors are always declared within genes, and are updated whenever these genes are instantiated. Thus, the queries defining the contents of each of the 200 instances of the `book` element gene in Figure 2(b) will be evaluated against different `b_rec` elements. Therefore, the output of the template in Figure 2(b) will be 200 `book` elements, each with a distinct value for its `isbn` attribute[2]. Cursors can be nested (see Figure 2(b)); the name of the parent cursor (`$a` in the example) is used as the starting node of the path expression defining the nested cursor.

Consider the `authors` attribute gene specified in Figure 2(b); its content is obtained by querying the `author_id` values of the current `b_rec` element in the outer cursor. Recall that up to 5 author id's are generated per book. The **tox-maxOccurs="unbounded"** annotation in the gene specification determines that ToXgene should generate as many instances of the gene as the number of elements in the cursor defining its content. Therefore, the resulting attribute will be a comma separated list of author id's.

The ToXgene query language also defines aggregate and string manipulation functions [9].

2.3 Specifying XML documents

Documents are specified by **tox-document** annotations, as shown in Figure 3(a). Similarly to a list, a document specification contains a name and a gene, whose instance will be the *root* element of the document. Collections of documents are specified by declaring the **copies** attribute in a document declaration (see Figure 3(b)).

ToXgene's output for the specification in Figure 3(a) is a single XML document, called `books.xml`, whose root is an element called `books`, and whose children are 200 `book` elements copied from a list. ToXgene's output for the specification in Figure 3(b) is a collection with 200 XML documents, called `book0.xml`, `book1.xml`, etc.; each document has an element called `book` as root, whose contents are copied from the same list. We note that the ith book in `books.xml` corresponds to the `book`i`.xml` document in the collection.

2.4 Specifying irregular structures and recursive elements

Templates can specify control statements that determine, at content generation time, which genes are instantiated. Each control statement contains one or more *blocks* of genes; at processing time, *at most* one such block is chosen, and only the genes in that block are instantiated. ToXgene provides the customary IF-THEN-ELSE statement and also a "lottery" statement, in which a block is randomly chosen according to a probability distribution. Control statements can be nested arbitrarily. Also, the blocks in a control statement might contain completely different genes; note that this provides a way of specifying some controlled randomness in the structure of the XML documents produced by ToXgene.

Recursion in element genes. ToXgene also supports the generation of recursive XML content, determined by three parameters: the total number of (recursive) elements to be generated, the number of children per element, and the number of levels in the recursion. Any of the supported probability distributions can be used for specifying these parameters.

[2]Note that ISBN values are unique in `book_list`.

Query	sizes		ratio	
	ToXgene	dbgen	size	time
q14	1	1	1.0000	1.0366
q3	10	10	1.0000	1.0972
q2	54	44	1.2273	1.2665
q16	2789	2762	1.0098	1.2853

(a) TPC-H results.

Query	sizes		ratio	
	ToXgene	xmlgen	size	time
q1	12	12	1.0000	1.0097
q11	7661	7661	1.0000	1.0086
q12	1358	1436	0.9457	0.9888
q17	25595	25535	1.0023	1.0100

(b) Xmark results.

Table 1: Comparison between ToXgene and other data generators. The metric used for part (a) is number of tuples in the result set; the metric for part (b) is the length (in lines) of Kweelt's output. All ratios are computed by dividing the measurement with ToXgene's data by the corresponding measurement with the data from other data generator.

3 Extending ToXgene

ToXgene is a general purpose tool, and as such provides built-in tools for generating XML content conforming to the most common datatypes. Although preliminary experimentation with our tool shows it can easily reproduce the synthetic data used in complex benchmarks, using a combination of queries and string operations, ToXgene was designed to support user-defined `CDATA` generators (e.g., a generator for pseudo-random DNA sequences).

ToXgene was implemented in Java 2, and its architecture was designed in a way that Java classes implementing `CDATA` generators could be added with little effort: all that is required is registering the new code in the Gene Factory module; of course, this new code has to implement the interface defined in our code, which consists of a single method. In principle, the code that produces instances of `simpleType`s is not supposed to produce XML elements (i.e., strings containing element tags). However, there is nothing in our code that enforces this behavior. Thus, if one needed XML elements with random tag names, it would suffice to encapsulate a data generator such as [1] as a new `simpleType` in our tool. ToXgene can also be coupled with external tools, since it has the ability of storing and reading lists from files.

4 Experimental results

Our goal with ToXgene was the generation of "useful" synthetic XML documents. To test our tool, we tried to reproduce the data produced by the generators of known benchmarks; we chose the generators of TPC-H [10] (`dbgen`) and Xmark [8] (`xmlgen`). We decided to use TPC-H because it is a widely used relational benchmark and because it defines many non-trivial integrity constraints over its database. Xmark was chosen because it was designed specifically for XML, and specifies a complex document, with different levels of nesting, and thousands of references among its elements.

Experimental setting. For the TPC-H data set, we generated data for a 100Mb relational database (i.e., we used a scaling factor of 0.1), corresponding to 400Mb of XML data. These documents were loaded into a relational database in DB2 using IBM's XML Extender [6]; the schema of this database was identical to the one populated by `dbgen`. No integrity constraint was violated by our data. For Xmark, we generated a 100Mb document (i.e., we used a scaling factor of 1). Our document was validated against the DTD provided by the benchmark authors without problems.

For each benchmarking data set, we ran both queries from the benchmark workload and some *ad-hoc* queries to test specific details of the data generated. We used Kweelt[3] to run these queries on the Xmark data sets.

All experiments were run on a 4-way 500 Mhz Pentium III machine with 3Gb of RAM and 18Gb of disk storage. The data generation takes about 1 hour for TPC-H, and about 25 minutes for Xmark. The memory required for storing the lists are 750Mb and 250Mb of RAM, respectively.

Experimental results. Table 1 shows the results of some of the queries selected from the workloads of the benchmarks. We show the size of the result sets of the queries, and the relative times to execute these queries on each data set. We show queries of different result set sizes, both with fixed (e.g., "find the best n customers") or variable result set sizes. As one can see, the results are comparable.

The *ad-hoc* queries were designed to compare the distributions of values between the data sets. Table 2 shows the result of some of these queries. All data values in TPC-H are generated according to uniform distributions; this is not the case for Xmark: for instance, the average prices of items in closed auctions obey an exponential distribution, while the number of items that can be bought with a credit card obey a uniform distribution. Again, the results are comparable.

Generating Xmark text. Most of the textual content in Xmark is generated by sampling from the 17000 most common words in Shakespeare's plays, according to their frequency of occurrence. We obtain similar content by loading the list of words used by `xmlgen` into a list, which is sampled accordingly. Sentences are formed by concatenating these words. We generate the final text by copying these sentences into `emph`, `bold` and `keyword` elements in a recursive fashion, so that we can have `bold` sentences nested within `emph` sentences, etc.

[3] Available at `http://http://kweelt.sourceforge.net/`.

Feature	Target	ToXgene	dbgen
average balance on customer accounts	4500	4510.06	4470.50
average size of parts	25	25.00	25.00
average supply cost of parts	5000	499.74	499.69
average tax per item in each order	0.04	0.04	0.04

(a) TPC-H results.

Feature	Target	ToXgene	xmlgen
average price in closed auctions	100.00	100.58	96.83
average "happiness" of customers with closed auctions	5.50	5.48	5.43
percentage of items in Europe that have "United States" as location	0.75	0.7553	0.7447
percentage of items in Asia that can be purchased with credit card	0.50	0.5020	0.4945

(b) Xmark results.

Table 2: Comparison between ToXgene and the other data generators, using the *ad-hoc* queries. The target value on the tables is the expected value for the corresponding feature, given the probability distribution specified by the benchmark.

5 Conclusions and future work

In this paper we introduced ToXgene: an extensible template-based generator for consistent collections of correlated synthetic XML documents. We presented an overview of our template specification language; we discussed some of the novel features of our tool, such as element sharing, and described how ToXgene can be extended or coupled with other tools. Finally, we reported on preliminary experiments we conducted with our tool.

Development of ToXgene is continuing. We intend to provide a more general mechanism to allow the generation of text according to different vocabularies, grammars, and character encoding schemes. This would be of capital importance for generating testing data for text-intensive applications. We are also working on improving ToXgene's performance; in particular, we are interested in exploiting possible parallelism in the data generation.

As future work, we identify the extraction of ToXgene templates from existing documents as an interesting problem. The obvious application of this research would be generating synthetic data closely reproducing the characteristics of real documents. This would be especially important when inspecting the data is unfeasible or impossible (e.g., for security reasons). Also, templates capture considerable information about the data they describe, and thus are valuable metadata in themselves.

References

[1] A. Aboulnaga, J. F. Naughton, and C. Zhang. Generating synthetic complex-structured XML data. In *Proceedings of the Fourth International Workshop on the Web and Databases*, pages 79–84, Santa Barbara, CA, USA, May 24-25 2001.

[2] D. Barbosa, A. Barta, A. Mendelzon, G. Mihaila, F. Rizzolo, and P. Rodriguez-Gianolli. ToX - the Toronto XML engine. In *Proceedings of the International Workshop on Information Integration on the Web*, pages 66–73, Rio de Janeiro, Brazil, April 9-11 2001.

[3] P. Bohannon, J. Freire, P. Roy, and J. Siméon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition) - W3C recommendation. Available at `http://www.w3.org/TR/2000/REC-xml-20001006`, October 6 2000.

[5] D. C. Fallside. XML Schema part 0: Primer - W3C candidate recommendation. Available from `http://www.w3.org/TR/xmlschema-0/`, October 24 2000.

[6] IBM DB2 Universal Database XML Extender - administration and programming. Available at `http://www-4.ibm.com/software/data/db2/extenders/xmlext`, 1999.

[7] IBM XML Generator. Available from `http://www.alphaworks.ibm.com/tech/xmlgenerator`.

[8] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

[9] To*Xgene* - the ToX XML data generator. `http://www.cs.toronto.edu/tox/toxgene`, 2001.

[10] Transaction Processing Performance Council. *TPC Benchmark H - Decision Support*, 1999. Revision 1.3.0.