Character Repertoire Validation for XML Documents

Erik Wilde, ETH Zürich (Swiss Federal Institute of Technology)

Abstract

XML documents may contain a large diversity of characters. The *Character Repertoire Validation for XML (CRVX)* language is a simple schema language for specifying character repertoire constraints. These constraints can be specific for syntax- and/or context-based parts of an XML document. The constraints are based on the character classes introduced by XML Schema's regular expressions.

1 Introduction

One component among the different validation tasks defined by the *Document Schema Definition Languages (DSDL)* framework is the character repertoire validation. The *Character Repertoire Validation for XML (CRVX)* language presented here is one possible way to express character repertoire constraints for XML documents, and may be used in a validation pipeline to protect subsequent components from characters that they are not capable of dealing with.

2 Characters in XML

XML uses the ISO 10646 [3] character set (also known as Unicode), and consequently XML documents may contain a large diversity of characters, even for markup such as element and attribute names. Not all applications are capable of dealing with this diversity. In reality, many XML applications still operate in environments which are not Unicode-compliant, and to ensure error-free processing, it is necessary to reject documents that do not comply with the character repertoire supported by the processing pipeline.

3 CRVX Schemas

A CRVX schema is a simple document describing the character repertoire constraints that an XML document must follow. CRVX schemas are XML documents based on an XML Schema and some additional constraints (which cannot be expressed in an XML Schema). The main idea of CRVX is to make character repertoire validation syntax- and context-specific, so that character repertoire constraints can be based on syntax and/or context definitions.

Because CRVX uses XPath to specify contexts (as described in Section 3.3), CRVX contains a mechanism to declare namespaces. This makes it possible to use prefixes in qualified names, which are then interpreted according to the namespace mechanism.

3.1 Character Repertoires

The definition of character repertoires is based on XML Schema Datatypes [1], which defines that "a *character class* is an atom that identifies a set of characters". We use this subset of the XML Schema regular expression syntax to specify a character repertoire, which we define to be a sequence of at least one character class. XML Schema character classes are either a *character class escape* (identifying a predefined character class[1]), or a *character class expression* (a group of explicitly allowed or disallowed characters).

3.2 Syntax-based Constraints

Character repertoire constraints are based on the syntactic structure of an XML document. Each constraint is defined by a `restrict` element, which has a `charrep` attribute for the actual character repertoire constraint, and optionally the syntactic structure(s) for which this constraint applies. The allowed syntactic structures are *element names*, *element content*, *attribute names*, *attribute values*, *processing instruction targets*, *processing instruction content*, *comments*, and *entity names*. If a `restrict` element does not have a `structure` attribute, the constraint applies to all the syntactic structures.

It is possible to define multiple constraints, which are combined using a logical 'and' (i.e., a syntactic structure is tested for all constraints which are defined for it). This way, it is possible to define some rather general restrictions for all syntactic structures of an XML document, and then define some more specific constraints only for certain syntactic parts (such as element content). However, with syntax-based constraints alone, it is only possible to define character repertoire constraints that are global for an XML document (albeit potentially specific to certain syntactic structures).

3.3 Path-based Contexts

Contexts may be used to make character repertoire validation more specific. The `context` element has a `path` attribute that specifies an XSLT pattern [2]. Optionally, the `context` element may have a `name` attribute defining the context's name. Contexts can be used in different ways:

- *Context-specific constraints:* Constraints may be made context-specific by either specifying them inside of a `context` element, or by referencing the required context(s) using a `within` attribute for the `restrict` element.
- *Nested contexts:* Contexts can be nested by either nesting `context` elements (a context may contain several contexts), or by referencing the required context(s) using a `within` attribute for the `context` element. Context nesting is accomplished by simply

[1] These character classes may refer to properties from the *Unicode Character Database (UCD)*, which classifies characters according to properties (such as 'uppercase letter') and blocks (such as 'Hebrew').

concatenating patterns (and inserting a '/'), and no attempt is made to intelligently merge patterns where the second pattern uses multiple location steps.

The option to specify nested contexts by reference rather than by inclusion has been chosen to enable applications to re-use contexts and thus eliminate redundancy in context patterns and/or constraint definitions.

4 Examples

CRVX is a very small and rather simple language, designed to accomplish a rather simple purpose, and is easily understood and learned. The following examples illustrate some simple use cases.

```
<crvx xmlns="http://dret.net/xmlns/crvx10">
 <restrict structure="ename aname pitarget"
           charrep="\p{IsBasicLatin}"/>
 <restrict structure="ename aname"
           charrep="[^0-9]"/>
</crvx>
```

In this example, syntax-based constraints are used to assert that elements, attributes, and processing instruction targets must use ASCII characters, but (with the exception of processing instruction targets) may not use numbers[2]. Both restrictions are combined by a logical 'and', thus enforcing that all names are restricted to a rather small character repertoire.

```
<crvx xmlns="http://dret.net/xmlns/crvx10">
 <restrict structure="econtent"
           charrep="\p{IsBasicLatin}
                    \p{IsLatin-1Supplement}"/>
 <context name="eng" path="chap[@lang='en']"/>
 <restrict within="eng"
           structure="econtent"
           charrep="\p{IsBasicLatin}"/>
</crvx>
```

This example shows how a context is defined that contains everything in `chap` elements with a `lang="en"` attribute. The first constraint restricts all element content to be ISO 8859-1 compliant, while the second constraint (which only applies to element content within the `eng` context) further limits the allowed character repertoire to be ASCII only.

```
<crvx xmlns="http://dret.net/xmlns/crvx10">
 <namespace prefix="test" uri="..."/>
 <context name="c1" path="test:element1"/>
 <context within="c1" name="c2"
          path="test:element2"/>
 <context within="c1" name="c3"
          path="test:element3">
  <restrict charrep="\p{IsBasicLatin}"/>
 </context>
 <restrict within="c2 c3"
           charrep="\P{IsTelugu}"/>
</crvx>
```

[2]It should be noted that the naming constraints defined by XML itself always apply.

This last example demonstrates some of CRVX's features. It uses a namespace declaration, which is then used in context patterns. All constraints are only specified for context `c1`, but only indirectly through contexts `c2` and `c3`. The first constraint is specified for context `c2` (implicitly, by being a child of the `context` element), while the second constraint is specified for contexts `c2` and `c3` (by explicitly referencing them). Here it is worth mentioning that constraints (as well as contexts themselves) may only use either explicit or implicit contexts, but not both (i.e., they may not be a child of a `context` element and have a `within` attribute).

5 Implementation

The implementation of CRVX currently is rather simple and very similar to the model of *Schematron*. A CRVX schema is processed by XSLT and transformed into XSLT. The resulting XSLT must be processed by a XSLT 2.0 [4] processor because of its built-in support for regular expressions (regular expression matching is a standard XPath 2.0 function [5]).

Unfortunately, this implementation restricts CRVX processing to namespace-conforming XML. Ideally, CRVX schemas should be able to process any well-formed XML, at least as long as they only use syntax-based constraints[3]. A native implementation of CRVX should overcome this restriction and also provide much better performance.

6 Conclusions

CRVX is a simple solution to a simple problem. However, the problem is an important one, since many applications in XML processing pipelines are not prepared to handle the full diversity of Unicode that XML supports. While our XSLT-based implementation serves as a proof of concept, it should be replaced with more efficient DOM- or SAX-based implementations in real-world applications. In the future, CRVX or something similar will be used in validation pipelines such as DSDL.

References

[1] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502, May 2001.

[2] James Clark. XSL Transformations (XSLT) Version 1.0. World Wide Web Consortium, Recommendation REC-xslt-19991116, November 1999.

[3] International Organization for Standardization. Information technology — Universal Multiple-Octet Coded Character Set (UCS). ISO/IEC 10646, 1993.

[4] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20021115, November 2002.

[5] Ashok Malhotra, Jim Melton, Jonathan Robie, and Norman Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0. World Wide Web Consortium, Working Draft WD-xquery-operators-20021115, November 2002.

[3]Contexts are based on XPath and thus only work with namespace-conforming XML.