Validation of Character Repertoires for XML Documents

Erik Wilde
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology, Zürich

(Paper URI: http://dret.net/netdret/publications#wil03h)

Abstract

XML is based on Unicode, and therefore XML documents may use the full Unicode character repertoire. However, XML-based applications often use XML interfaces to legacy software which in many cases is not capable of dealing with the full Unicode character repertoire. We therefore propose a schema language for XML which is capable of limiting the character repertoire of XML documents.

This schema language, called *Character Repertoire Validation for XML (CRVX)*, has features to permit or disallow character repertoire subsets from certain parts of an XML document, for example only for element and attribute names. CRVX uses information from the *Unicode Character Database (UCD)* to make character repertoire specification as easy as possible.

CRVX is not intended to be the only schema language in an XML application scenario, but it provides useful additional schema-based validation to protect applications from unsupported characters. XML applications typically combine different schema languages before processing XML documents, and CRVX is intended to complement other schema languages such as grammar-based languages (DTD, XML Schema) or rule-based languages (Schematron).

CRVX can be implemented in various ways. One simple solution is to use XSLT to transform an CRVX schema into an XSLT program, which is then used to validate XML documents. We briefly describe such an implementation. Other (and more efficient) implementations could be based on DOM or SAX parsers.

Contents

1 Introduction

The *Extensible Markup Language (XML)* [4] has become the single most successful format for data exchange in business-to-business application scenarios. XML is defined on the character level and uses Unicode [15] as its foundation. In many application scenarios, only certain classes of XML documents are considered useful. so called *document classes*. In the classical XML world, these classes are defined by a *Document Type Definition (DTD)*, which is a definition of a grammar for XML documents. Several weaknesses of the DTD mechanism (most notable, the lack of built-in datatypes and the lack of support for a type system of elements and attributes) led to a more complex, more powerful schema language for XML, *XML Schema* [2, 14].

While XML Schema is considerably more powerful than DTDs, it still misses some features that are required by many applications, for example co-constraints, or type information for mixed content. Generally speaking, the development of XML Schema has proven that it is possible to define a powerful schema language, but that there will always be application requirements that are not met by the language. Consequently, as an alternative, the approach of validation as a modular process has gained some momentum, most prominently demonstrated by the ISO's development of the *Document Schema Definition Languages (DSDL)* [10] framework. DSDL's idea is to provide a framework for the combination of different XML schema languages, so that individual applications may choose the schema language(s) that best fit the application's requirements. Details about DSDL can be found on the DSDL Web page `http://www.dsdl.org` or in publications by van der Vlist [16, 17].

In this paper, we present a rather small, but important addition to the existing set of XML schema languages. We describe a schema language for the validation of XML documents with respect to the character repertoire they are using. The *Character Repertoire Validation for XML* [19] schema language enables application designers to shield applications from unwanted or unsupported characters in XML documents. In addition to the mechanisms of XML Schema (i.e., regular expression facets for simple types), CRVX supports character repertoire restrictions for element and attribute names, for the content of comments and processing instructions, and for mixed content. Furthermore, CRVX can be implemented much more efficiently that XML Schema, because it is a much lighter schema language.

2 XML and Unicode

XML is defined on the character level, and relies on Unicode as the underlying standard for defining the character repertoire and encodings. With the exception of a small number of characters [8], XML documents may use the full character repertoire of Unicode. *XML 1.0* was based on Unicode 2.0, *XML 1.0 Second Edition*, the current recommendation, is based on Unicode 3.0, and the ongoing work on *XML 1.1* [6] reflects the continued evolution of the Unicode standard, which at the time of writing (June 2003) has reached release 4.0.0. XML's usage of Unicode makes the handling of character data in XML very flexible, so that the only characters that really are fixed in an XML document are the markup characters, as shown in Figure 1.

For making the Internet truly worldwide and accessible for all kinds of users, this internationalization of XML is a very important feature. However, in many application scenarios, the software dealing with XML data is not built from scratch, but existing software (often referred to as "legacy systems") augmented with XML interfaces. This means that even though the XML interchange format used between applications may be capable of representing Unicode data, the actual applications may still be limited to more restricted character repertoires, such as ASCII [1] or some flavors of ISO-8859 [11].

```
<文書 改訂日付="１９９９年３月１日">
 <題>サンプル</題>
 <段落>これはサンプル文書です。</段落>
 <!-- コメント -->
 <段落>&会社名;</段落>
 <図面 図面実体名="サンプル"/>
</文書>
```

Figure 1: XML Document with Japanese Names, Content, and Comment

One interesting question in the context of XML systems is how to shield these applications from characters that they cannot deal with. Basically, there are three approaches to this question:

- *Writing code:* After XML parsing, it would be necessary to write code for checking for unwanted characters, before passing on the XML data to the actual application logic. This approach is very efficient, but also very unflexible, because every time the XML changes, the code must be checked and possibly modified. Depending on the complexity of the required checking, this may result in considerable effort and thus costs.
- *Using an general purpose schema language:* There is some support for character repertoire checking in XML Schema, so this may be sufficient in some scenarios. However, there are two possible disadvantages, one is the restricted support for character repertoire checking in in XML Schema (e.g., mixed content is untyped and thus cannot be checked), the other one is the expense of XML Schema validation. XML Schema software typically is complex, and maybe too heavy-weight in some scenarios where character repertoire checking is required, but none of the other features of XML Schema are used.
- *Using a specialized schema language:* The third approach is to use a small and specialized declarative mechanism, a schema language specifically designed to support character repertoire validation. Since this schema language is small, it may be implemented efficiently and can be used very easily without the overhead of heaving to learn a more complex schema language.

In this paper, we describe a schema language that implements the third approach. In the following chapter, we describe the schema language itself, while Chapter 4 discusses implementation issues.

3 The CRVX Schema Language

The *Character Repertoire Validation for XML (CRVX)* schema language has the sole purpose of restricting the set of valid documents based on the characters occurring in a document. It can be used as a stand-

alone schema language, or in combination with other schema languages, such as DTDs or XML Schema. In the following sections, CRVX is described according to three different dimensions of the language; the first one is the information model of XML being used (Section 3.1), the second one is the mechanism for restricting character repertoires (Section 3.2), and the third one is the possibility to specify context-sensitive restrictions (Section 3.3).

3.1 XML Information Models

XML's information model is the subject of very interesting discussions, and the set of XML information models is constantly growing, with diverse members such as XML 1.0 itself, the *XML Information Set* [7], various versions of the *Document Object Model (DOM)*, and the XPath 1.0 [5] and 2.0 [9] information models. Some of these models have rather subtle differences, others have differences that may be very important in some applications. One example of this is the lack of support for CDATA sections in the XPath information models.

To avoid this diversity of different perspectives on XML, CRVX remains close to the XML 1.0 model, but completely ignores the physical structures of an XML document. CRVX operates on XML's logical structures, and consequently does not have any means of specifically addressing the entity structure of an XML document.

However, there is one additional aspect outside of XML 1.0 that is supported by CRVX, and this is the issue of *XML Namespaces* [3]. Namespaces are very popular with XML applications, and the usage of Namespaces implies some additional constraints for XML document. Since Namespaces introduce a new perspective on XML document, most notably by structuring names into prefixes and local names, and by introducing Namespace declarations as a special kind of attribute, they are explicitly supported by CRVX, making it possible to interpret a document either as *pure XML* (it must be well-formed) or as *Namespace XML* (it must be namespace-well-formed).

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  ...
</crvx>
```

Figure 2: Selecting the XML Information Model in CRVX

Figure 2 shows how the information model to be used for validation is selected in CRVX. It also shows the general skeleton of an CRVX schema, which is an XML document. The three attributes of the document element specify the XML information model to be used for the CRVX schema, the version of CRVX, and declare the CRVX namespace (in this case using a default namespace declaration).

3.2 Restrictions

The central purpose of CRVX is to specify character repertoire restrictions, which are then used to validate XML documents. Restrictions in CRVX have three properties, the character repertoire, the minimum and/or maximum length of character sequences, the structural parts of an XML documents the restrictions should apply to, and the context in which these restrictions should be validated:

- *Character Repertoire:* The character repertoire of an CRVX restrictions uses mechanisms from *XML Schema Datatypes* [2] to specify character classes. These character classes can be of two forms, which

are (1) a character class expression (for example '`[b-y]`' for the character set from b to y)[1], and (2) a category escape (for example '`\p{Ll}`' for all lowercase letters). Category escapes may also be used inside character class expressions, but since category escapes will probably be used very often for specifying CRVX constraints, they have been made a top-level construct in CRVX.

The category escapes use values from the *Unicode Character Database (UCD)*, in particular values from the general category and blocks. *General Categories* are used for classifying characters, such as letters, number, symbols, or punctuation characters. *Blocks* are arbitrary names for ranges of code points and are often used for grouping characters according to their source, for example a certain language (or family of languages) or application area. Figure 3 shows how blocks can be used in CRVX.

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  <restrict charrep="\p{IsBasicLatin} \p{IsLatin-1Supplement}"/>
</crvx>
```

Figure 3: Restricting an XML Document to a Character Repertoire

In this example, it can also be seen that character repertoire can be combined by using XML Schema's list type, thus separating different character repertoire restrictions into tokens separated by whitespace. However, since CRVX also supports character class expressions, the example from Figure 3 could also be specified as `charrep="[\p{IsBasicLatin}\p{IsLatin-1Supplement}]"`. If the `charrep` attribute specifies a list (i.e., contains multiple token separated by whitespace), then these are combined using a logical "or", so that the resulting character repertoire effectively is the union of the character repertoires specified by the list tokens.

- *Lengths:* Since it may be another validation requirement related to character repertoire to limit the length of certain structures of an XML document, it is possible to restrict the minimum and maximum lengths. In most cases, this feature will be used in combination with limiting the restriction to certain structures, which is why the example shown in Figure 4 combines these two features.

- *Structure:* In many cases, restrictions do not apply to any characters appearing in an XML document, but only to certain structural parts, such as element or attributes names, attribute values, or element content. As described in Section 3.1, CRVX supports two XML information models (i.e., two different perspectives of XML), which are *pure XML* and *Namespace XML*. The structural parts of an XML document that are accessible in both models are *element content*, *CDATA sections*, *attribute values*, *processing instruction targets*, *processing instruction contents*, and *comments*. For pure XML, the additional structures are *element names* and *attribute names*. For Namespace XML, the additional structures are *element local names*, *attribute local names*, *namespace names*, and *namespace prefixes*. Figure 4 shows how to use structures for a restriction.

  In this example, the actual character repertoire remains unrestricted, but the maximum length of element and attribute local names is restricted to eight characters. This example also demonstrates an advantage over XML Schema, which can apply simple type restrictions only to typed parts of an XML document (i.e., attribute values and non-mixed element content), but not to other structural parts.

[1] Character class expressions may contain single character escapes (for example '`\-`' for a hyphen), and multi-character escapes (for example '`\i`' for the initial name characters).

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  <restrict structure="elementLocalName attributeLocalName" maxlength="8"/>
</crvx>
```

Figure 4: Using Restriction Structures and Lengths

- *Context:* In some cases, it may be desirable to limit the restrictions to certain parts of an XML documents, for example to check only the character repertoire of text that appears as descendant of some given element. CRVX supports this kind of application as *context*, but since there are several ways of using contexts, they are described separately in Section 3.3.

It is thus possible to use different orthogonal dimensions for defining restrictions. Since users in many cases want to combine these dimensions differently, restrictions can be combined to yield more complex CRVX schemas, as shown in Figure 5.

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  <restrict structure="elementLocalName attributeLocalName PITarget"
            charrep="\p{IsBasicLatin}"/>
  <restrict structure="elementContent"
            charrep="\p{IsBasicLatin} \p{IsLatin-1Supplement}"/>
  <restrict structure="PITarget" minlength="3" maxlength="3"/>
</crvx>
```

Figure 5: Using multiple Restrictions

It should be noted that this CRVX schema defined two restrictions for processing instruction targets, one (together with element and attribute names) for restricting targets to basic latin (i.e., ASCII) characters, and the other one for requiring that they must have three characters. However, even by using multiple restrictions, they still always apply to the entire document scope, and the following section describes how this can be changed by using CRVX contexts.

3.3 Contexts

A context in CRVX is defined by an XSLT pattern. Consequently, contexts may only be used with Namespace XML, since the whole set of XSLT/XPath recommendation is based on an information model that requires namespace-well-formed XML. The idea of using contexts has also been used by Schematron, which uses the same XSLT pattern approach as CRVX. Contexts are defined using a special CRVX element, as shown in Figure 6.

This example shows various features of context definitions. Context definitions may contain restrictions, in which case the restriction only applies to the context selected by the context's `path` attribute. Contexts may also be nested, in which case the nested context is evaluated in the context in which it is specified. To make contexts reusable, they may also carry a `name` attribute, which can then be referenced from restrictions or contexts with a `within` attribute, thus allowing context hierarchies to be directed acyclic graphs instead of trees.

Since contexts are based on Namespace XML and use XSLT patterns which include element and/or attribute names, they may also use namespaces. Namespaces are declared using a special element, which is described in the following section.

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  <context path="figure/caption">
    <restrict charrep="\p{IsBasicLatin} \p{IsLatin-1Supplement}"/>
    <context path="link">
      <restrict structure="elementContent" maxlength="10"/>
    </context>
  </context>
</crvx>
```

Figure 6: Using Restrictions within Contexts and nested Contexts

3.4 Namespace Declarations

Namespace declarations are specified using a special CRVX element. They may only appear as direct children of the `crvx` element. All namespace prefixes used in `path` attributes of `context` elements must be declared using a `namespace` element. Figure 7 shows how a namespace is declared and used in CRVX.

```
<crvx structures="namespaceXML" version="1.0" xmlns="http://dret.net/xmlns/crvx10">
  <namespace prefix="html" name="http://www.w3.org/1999/xhtml"/>
  <context path="html:html/html:head/html:title">
    <restrict charrep="\p{IsBasicLatin}"/>
  </context>
</crvx>
```

Figure 7: Using Namespaces in CRVX

Unprefixed names in `path` attributes of `context` elements may also be declared to belong to a namespace. This can be done in a similar way to XSLT 2.0 by using the `default-path-namespace` attribute of the `crvx` element.

4 CRVX Implementation

CRVX is a rather simple schema language, and its main advantage over performing character repertoire validation within program code is that it is declarative. This means that an CRVX author only specifies what should be checked for, but not how it should be done. The actual validation is the task of an CRVX implementation. Similarly to Schematron, CRVX implementation can be done rather easily by transforming a CRVX schema into an XSLT stylesheet. This approach is described in Section 4.1. The XSLT-based approach however has some disadvantages, such as performance issues and the inability to deal with non-namespace-compliant XML. Thus, alternative implementations could be based on existing XML parsers, and this second approach is described in Section 4.2

4.1 Based on XSLT

CRVX uses XML as its notation and thus can be processed using XSLT. It is thus possible to write an implementation of CRVX that uses XSLT as a tools to transform an CRVX schema into an XSLT stylesheet, and then uses this stylesheet to perform CRVX validation. Such an implementation is described in the CRVX specification [19]. The advantage of this approach is that it does not require any special

software, the only thing that is required is an XSLT processor. The disadvantages are that a XSLT 2.0 [12] is required, because only XSLT 2.0 supports the regular expression matching that is required to perform CRVX validation.[2] Other disadvantages are that due to the information model of XSLT 2.0, there

- is no way to process non-namespace-compliant XML documents with an CRVX implementation based on XSLT, and
- it is impossible to support the structural part of CDATA sections, because XSLT's information model does not support CDATA sections.

Given these limitations, an XSLT-based implementation of CRVX is rather simple. Contexts can be implemented using template rules, nested contexts can be implemented using looping. The restrictions themselves can be implemented using regular expression matching, with lengths being mapped to XML Schema regular expression quantifiers. The different structural parts can be accessed using *XQuery 1.0 and XPath 2.0 Functions and Operators* [13], which provides functions such as `local-name()` for evaluating the local name of an element or attribute.

4.2 Based on XML Processor

While an XSLT-based implementation is sufficient for applications that can live with the rather poor performance of an XSLT stylesheet and the limitations that are caused my the underlying XSLT, more efficient and feature-complete CRVX implementations will need to use another foundation. Most likely, an XML processor will be used. Popular interfaces for writing software based on XML processors are *Document Object Model (DOM)* and *Simple API for XML (SAX)*, and in both cases the CRVX implementation would use the functions offered by the respective API to access the XML document. The *DOM3 XPath Module* [18] would make it rather easy to implement contexts, while SAX does not support this kind of functionality and thus would require more programming effort to implement contexts.

5 Future Work

The schema language described in this paper is the first publicly available version. We assume that future versions will offer different features. For example, it is unclear whether lengths and nested (as well as referenced) contexts are sufficiently popular with CRVX users to justify their inclusion in a future version. Apart form these issues, there are also some areas where CRVX could be extended:

- *Character Normalization:* The current work on XML 1.1 [6] demonstrates that character normalization is an important issue, that may be of concern to some applications. Since XML 1.0 does not and XML 1.1 will in all likelihood not prescribe any form of character normalization, this could be a potential candidate for a new feature in CRVX, extending the `restrict` element by a new attribute to specify required character normalization forms.
- *Character Representation:* Characters in XML can be represented in a variety of ways, most notably using the character itself or a character reference of the form `&#252;`. In some applications, it would be interesting to make restrictions to these representations, for example requiring that characters may not be specified using character references, or that character references must always use the

[2] It should be noted that XSLT 2.0 is in working draft status and thus may still change. Also, there are only few implementations, which also may change.

decimal form (hexadecimal character references appear as `&#xFC;`). Well-behaving XML software is required to handle all character representations equally, but in some scenarios this checking may be appropriate, for example before an XML document is accepted for storage in an archive.

Apart from these questions about CRVX evolution, we hope that CRVX will help to support the philosophy of regarding XML validation as a modular process rather than a monolithic task that involves only a single schema language. While work on DSDL is progressing, there are still many open questions, most notably the framework that will enable users to specify validation tasks in an easy and portable way.

6 Conclusions

In this paper, we present a small schema language for validating the character repertoire of XML documents. While this is only a small part of the validation that is required in many application scenarios, it has been designed deliberately to focus on a small task. It is meant as a light-weight alternative to very complex and heavy schema languages such as XML Schema, which try to solve a large number of problems in parallel, and result in a very complicated language where most users only need 10% of the language's potential. CRVX is easy to learn and easy to apply by users looking for a way to restrict the character repertoire of their XML documents.

References

[1] American National Standards Institute. Coded Character Set — 7-Bit American National Standard Code for Information Interchange. ANSI X3.4, 1992.

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

[3] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.1. World Wide Web Consortium, Candidate Recommendation CR-xml-names11-20021218, November 2002.

[4] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-20001006, October 2000.

[5] James Clark and Steven J. DeRose. XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[6] John Cowan. XML 1.1. World Wide Web Consortium, Candidate Recommendation CR-xml11-20021015, October 2002.

[7] John Cowan and Richard Tobin. XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[8] Martin J. Dürst and Asmus Freytag. Unicode in XML and other Markup Languages. World Wide Web Consortium, Note NOTE-unicode-xml-20030613, June 2003.

[9] Mary F. Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh. XQuery 1.0 and XPath 2.0 Data Model. World Wide Web Consortium, Working Draft WD-xpath-datamodel-20030502, May 2003.

[10] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL). to be published as ISO/IEC 19757.

[11] International Organization for Standardization. Information technology — 8-bit single-byte coded graphic character sets. ISO/IEC 8859, 1999.

[12] Michael Kay. XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20030502, May 2003.

[13] Ashok Malhotra, Jim Melton, and Norman Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators. World Wide Web Consortium, Working Draft WD-xpath-functions-20030502, May 2003.

[14] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. World Wide Web Consortium, Recommendation REC-xmlschema-1-20010502, May 2001.

[15] Unicode Consortium. *The Unicode Standard: Version 3.0.* Addison Wesley, Reading, Massachusetts, 2000.

[16] Eric van der Vlist. XML Schema Languages. In *Proceedings of XML Europe 2002*, Barcelona, Spain, May 2002.

[17] Eric van der Vlist. XML Schema Languages. In *Proceedings of XML 2002*, Baltimore, Maryland, December 2002.

[18] Ray Whitmer. Document Object Model (DOM) Level 3 XPath Specification. World Wide Web Consortium, Candidate Recommendation CR-DOM-Level-3-XPath-20030331, March 2003.

[19] Erik Wilde. Character Repertoire Validation for XML (CRVX) Version 1.0. Technical Report TIK-Report No. 172, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland, June 2003.