Protecting Legacy Applications from Unicode

Erik Wilde
Swiss Federal Institute of Technology (ETH)
Zürich, Switzerland
icete2004@dret.net

Key words: XML, CRVX, XML validation, Unicode, DSDL

Abstract: While XML-based Web Service architectures are successfully turning the Web into an infrastructure for cooperating applications, not all problems with respect to interoperability problems have yet been solved. XML-based data exchange has the ability to carry the full Unicode character repertoire, which is approaching 100’000 characters. Many legacy application are being Web-Service-enabled rather than being re-built from scratch, and therefore still have the same limitations. A frequently seen limitation is the inability to handle the full Unicode character repertoire. We describe an architectural approach and a schema language to address this issue. The architectural approach proposes to establish validation as basic Web Service functionality, which should be built into a Web Services architecture rather than applications. Based on this vision of modular an infrastructure-based validation, we propose a schema language for character repertoire validation. Lessons learned from the first implementation and possible improvements of the schema language conclude the paper.

1 Introduction

For many applications today, the *Extensible Markup Language (XML)* (Bray et al., 2000) is the way to exchange data, either through proprietary mechanisms, or using some XML-bases standards such as *Web Services*. In either case, the actual data is exchanged using *Unicode* (Unicode Consortium, 2000), because XML is based on Unicode. XML allows different character encodings, but the only character encodings that must be supported by all XML implementations are UTF-8 and UTF-16. These two encodings are capable of encoding the full Unicode character repertoire, so that XML documents may contain any Unicode character.

In many cases, the applications implementing XML-based interfaces are not fully Unicode-compliant and require some sort of protection from receiving documents using the full Unicode character repertoire. In these cases, it is necessary to only forward documents using supported character ranges to the application, while XML documents containing unsupported characters should be rejected or at least trigger some kind of exception in the workflow.

In principle, filtering out unwanted documents in an XML-based environment is the job of *schema languages*. While XML’s built-in *Document Type Definition (DTD)* schema language does not support datatypes, the more recent *XML Schema* (Fallside, 2001) supports the concept of datatypes, including regular expressions and unicode character classes. Thus, Web Services based on XML Schema may define some restrictions of character repertoires. However, there are two major problems with this approach:

- *Monolithic Design:* XML Schema is a heavy schema language integrating many different concepts such as a type system for element and attribute types, identity constraints for specialized co-constraints, a library of simple datatypes, and defaulting mechanisms for elements and attributes. Application developers requiring only character repertoire capabilities may be overwhelmed by XML Schema’s complexity.
- *Incomplete Support:* XML Schema provides datatype support for attribute values and element content (excluding mixed content). This means that character repertoire restrictions cannot be defined for mixed content, element and attribute names, comments, and processing instructions.

In this paper, we present an approach overcoming these two limitations. We present a specialized schema language which is specifically designed for character repertoire validation. This makes it easier to use this schema language, in particular in scenarios where complex schema support is not yet required. The approach of *modular validation* presented in Section 2.1 is based on this perspective of XML validation as a sequence of processing steps, possibly performed at different nodes of an XML-based workflow. In Section 2.2 we give a more general description of our perspective on XML processing and how the Web Service world evolves towards a network-like infrastructure.

In Section 3 we describe the schema language for character repertoire validation in greater detail. The schema language is purely declarative and can be implemented on top of different XML technologies.

2 XML Validation and Web Services

According to (Booth et al., 2004), "a Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

For our purposes, the most important observations are that Web Services are based on the exchange of XML messages, and that the architecture is in no way restricted to a certain topology. Specifically, the well-known architectural concepts of computer networks, using concepts such as bridges, routers, gateways, firewalls, and proxies, can be employed to describe Web Service architectures. In Section 2.1, we discuss how this networked exchange of XML-based messages can be used to process XML in a modular and distributed way. Section 2.2 goes one step further and explains how this distributed processing of XML-based messages can be used to establish the concept of Web Services networks.

2.1 Modular Validation

For many XML users today, validation is a one-step process. Using a schema (in most cases a DTD or an XML Schema), an XML document is validated and is either successfully validated or classified as invalid. While this view of validation often is true, it is possible to look at validation in a modular way. In this scenario, validation is a modular process involving different schemas, which are checking different facets of an XML document.

For example, in the Web Service architecture of a company it might make sense to first check all incoming document against a schema that restricts the allowed character repertoire, because it is known that characters outside this repertoire are not supported within the company's workflow and thus should be rejected. If a Web Service request is valid with respect to the character repertoire, it is validated against the XML Schema for SOAP messages in general, and then against the XML Schema for the specific SOAP message type. Finally, before the SOAP message is passed to the application, it is validated against a message-specific schema, for example checking that some identification number inside the SOAP message corresponds to an existing item in the company's database.

The scenario discussed in the previous paragraph enables us to make some interesting observations and, in particular, generalizations regarding XML validation:

- *Distributed Validation:* The validation steps described above could be performed at different points within the company, for example the character repertoire and XML Schema validation could be performed on a central SOAP intermediary, while the application-specific validation takes place on the system implementing the SOAP service.
- *Increasing Specificity:* Typically, multiple validation steps increase the data quality by filtering out unwanted documents. While character repertoire validation is rather generic, the lookup in a database for some identifier is very specific and can only be performed by an application-specific validation step.
- *Application-Specific Validation:* As mentioned already, validation does not always have to be generic. Application-specific validation provides the benefits of describing assertions declaratively, making it easier to maintain and modify the application-specific schema. Application-specific validation also makes it easier for developers to encapsulate assertions and separate them from the code processing SOAP calls. (Nentwich et al., 2003) discuss this in depth from a software engineering point of view.

Based on these observations, it can be concluded that a more modular way of dealing with validation can be advantageous compared to the monolithic approach advertised by the more traditional approach to validating XML. The *Document Schema Definition Languages (DSDL)* currently under development by ISO is one example of a modular approach towards XML validation. It is based on a collection of schema languages, and a framework for specifying how to validate documents against possibly multiple schemas. However, DSDL will probably not provide built-in support for distributed validation.

It would be possible to look at validation as a Web Service itself, but this would make validation a rather expensive task. Instead, validation should be viewed as some kind of processing that is applied to SOAP messages in transit. This perspective brings us to the view of Web Services as something very similar to computer networks, an analogy we explore in the following section.

2.2 Web Services Networks

For computer networks, a number of standard networking devices are well-known and established, and composing networks out of bridges, routers, gateways, firewalls, and proxies is common knowledge. For Web Services, such a standard set of technologies and devices has not yet evolved, and we argue that many of the metaphors known from computer networking can be reused in the field of Web Services. (Wilde and Steiner, 2004; Jeckle and Wilde, 2004) give a more detailed discussion of the similarities of traditional (i.e., network-oriented) and Web Service protocol stacks, and describe how some of these similarities can be exploited to use existing experience and design patterns to build Web Service architectures.

As a concrete example, (Riggs, 2003) describes the concept of *Data Quality Firewalls*, which are devices that should be placed strategically to prevent or at least detect the inevitable loss of data quality in a workflow where many different peers are collaborating. From the XML and Web Services point of view, these firewalls could be implemented as devices capable of performing XML validation. They could then be configured to filter out SOAP messages which are violating the data quality standards (which are defined by schemas). The analogy between regular (i.e., computer network level) and Web Services firewalls is evident, both devices have built-in knowledge of the data to be processed, and both devices can be configured to use this knowledge for classifying passing data.

In the current version of SOAP (Gudgin et al., 2003), the notion of *SOAP Intermediaries* has already been established. This is a big step forward towards the vision of Web Services Networks. However, there still remain many gaps to be filled. An interesting approach is described by (Melzer and Jeckle, 2003). They have implemented a gateway that signs SOAP messages (encryption support is planned to follow). This could be compared to the setup of a *Virtual Private Network (VPN)*, where the intranet is a trusted network, but network traffic leaving the intranet is encrypted.

The concept of modular validation could be built into different devices in a Web Services Networks. SOAP gateways could be configured to perform validation (thus becoming “validating intermediaries”), and validation could also be integrated into development frameworks, providing a clear separation between the validation part of a Web Services (the assertions to be checked before the actual service invocation), and the service implementation. Modular validation is one facet of looking at Web Service from the perspective of modeling them after networking protocol stacks, and the interesting similarity between “validating intermediaries” and traditional firewalls is that both components should be configured rather than programmed, so that re-configuration can be done rather quickly.

3 CRVX

In the preceding section, we describe why and how modular validation can be useful. As one step in a validation pipeline, checking for character repertoires is a useful functionality (it is listed as one of the areas of the DSDL framework), and the *Character Repertoire Validation for XML (CRVX)* (Wilde, 2003a; Wilde, 2003b) language described here can be used to perform this kind of validation.

3.1 XML Information Models

The set of XML information models is constantly growing, with diverse members such as XML 1.0 itself, the *XML Information Set* (Cowan and Tobin, 2001), various versions of the *Document Object Model (DOM)*, and the XPath 1.0 (Clark and DeRose, 1999) and 2.0 (Fernández et al., 2003) information models. Some of these models have rather subtle differences, others have differences that may be very important for some applications. One example for 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 works on top of 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* (Bray et al., 2004a). Namespaces are very popular with XML applications, and the usage of Namespaces implies some additional constraints for XML documents. 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 ex-

Listing 1:

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

Listing 2:

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

Listing 3:

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

Listing 4:

```
<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>
```

Listing 5:

```
<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>
```

Listing 6:

```
<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 1: CRVX Examples

plicitly 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).

Listing 1 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 purpose of CRVX is to specify character repertoire restrictions, which are then used to validate XML documents. Restrictions in CRVX have four 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* (Biron and Malhotra, 2001) to specify character classes. These character classes can have 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. Listing 2 shows how blocks can be used in CRVX.

  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. 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:* It may be a validation requirement closely related to character repertoires to limit the length of certain structures of an XML document. Consequently, 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 Listing 3 combines these two features.

- *Structure:* In many cases, restrictions do not apply to all 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, 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*. Listing 3 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 character-only element content), but not to other structural parts.

- *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 Listing 4.

It should be noted that this CRVX schema defines 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. Contexts are defined using a special CRVX element, as shown in Listing 5.

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 be nested, in which case the nested context is evaluated in the context in which it is speci-

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

fied. 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.

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 special `namespace` element (i.e., it is not sufficient or required that they are declared as namespaces in the CRVX schema document). Listing 6 shows how a namespace is declared and used in CRVX.

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, a CRVX implementation can be programmed rather easily by transforming a CRVX schema into an XSLT stylesheet. This approach is described in Section 4.1. However, the XSLT-based approach 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 (Wilde, 2003a). 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 (Kay, 2003) processor 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 and 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* (Malhotra et al., 2003), 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 by 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 the *Document Object Model (DOM)* and the *Simple API for XML (SAX)*. The *DOM3 XPath Module* (Whitmer, 2003) 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 Further Work

CRVX has been designed as a small schema language and as a starting point. It is useful for basic validation tasks for character repertoires, but is also limited in a number of ways. The following points have been identified as possible extensions of CRVX:

- *Treatment of Nested Restrictions:* Currently, nested restrictions are treated as a logical union, so that the restrictions of a nested part of an XML document

[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.

are the union of all restrictions applying nodes further up the XML document tree structure. This implicit way of joining nested restrictions could be made explicit and variable, for example allowing overwriting of restrictions.

- *Character Reference Restrictions:* Characters in XML may appear literally or as character reference. Most XML information models do not distinguish between these two forms and pass the character to the application. However, it may be required to disallow the use of character references or, on the contrary, force characters from certain character ranges to appear as character references only.
- *XPath 2.0:* The XPath expression supports in CRVX currently are based on XPath 1.0. XPath 2.0 defines a more powerful language for addressing parts of an XML document (which may include type information, if the type information for a document can be inferred from some schema).
- *Character Normalization:* Unicode defines canonical and compatibility equivalences between characters or sequences of characters. For processing purposes, it can be useful to normalize a document. XML (including the emerging XML 1.1 (Bray et al., 2004b)) does not require character normalization, so that normalization checks are necessary for all documents which are not certified as being in normalized form (Dürst et al., 2003).

While these improvements are targeting CRVX, the surrounding infrastructure also need to evolve, before visions such as distributed and transparent validation in a Web Services network can be implemented in an easy way. In particular, the ideas of validation and SOAP intermediaries need to be merged to yield validating intermediaries, ideally configurable in a fully declarative way.

6 Conclusions

Web Services and the deployment of existing services through Web Service interfaces still are young disciplines. The schema language and the architectural approach to Web Service networks presented in this papers are contributions which hopefully help to increase the success of Web Service technologies. The major hurdle for Web Service deployment today is security, and if the Web Service standards evolve to a point where using an encrypting SOAP intermediary is as easy and natural as setting up a VPN, then Web Services will become a commodity as computer networks are today.

7 Acknowledgements

I would like to thank Diederik Gerth van Wijk and Martin Bryan for their comments regarding CRVX and possible improvements.

REFERENCES

Biron, P. V. and Malhotra, A. (2001). XML Schema Part 2: Datatypes. World Wide Web Consortium, Recommendation REC-xmlschema-2-20010502.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web Services Architecture. World Wide Web Consortium, Note NOTE-ws-arch-20040211.

Bray, T., Hollander, D., Layman, A., and Tobin, R. (2004a). Namespaces in XML 1.1. World Wide Web Consortium, Recommendation REC-xml-names11-20040204.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., and Maler, E. (2000). Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-20001006.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., and Cowan, J. (2004b). XML 1.1. World Wide Web Consortium, Recommendation REC-xml11-20040204.

Clark, J. and DeRose, S. J. (1999). XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116.

Cowan, J. and Tobin, R. (2001). XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024.

Dürst, M. J., Yergeau, F., Ishida, R., Wolf, M., and Texin, T. (2003). Character Model for the World Wide Web 1.0. World Wide Web Consortium, Working Draft WD-charmod-20030822.

Fallside, D. C. (2001). XML Schema Part 0: Primer. World Wide Web Consortium, Recommendation REC-xmlschema-0-20010502.

Fernández, M. F., Malhotra, A., Marsh, J., Nagy, M., and Walsh, N. (2003). XQuery 1.0 and XPath 2.0 Data Model. World Wide Web Consortium, Working Draft WD-xpath-datamodel-20031112.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Frystyk Nielsen, H. (2003). SOAP Version 1.2 Part 1: Messaging Framework. World Wide Web Consortium, Recommendation REC-soap12-part1-20030624.

Jeckle, M. and Wilde, E. (2004). Identical Principles, Higher Layers — Modeling Web Services as Protocol Stack. In *Proceedings of XML Europe 2004*, Amsterdam, Netherlands.

Kay, M. (2003). XSL Transformations (XSLT) Version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20031112.

Malhotra, A., Melton, J., and Walsh, N. (2003). XQuery 1.0 and XPath 2.0 Functions and Operators. World Wide Web Consortium, Working Draft WD-xpath-functions-20031112.

Melzer, I. and Jeckle, M. (2003). A Signing Proxy for Web Services Security. In Tolksdorf, R. and Eckstein, R., editors, *Berliner XML Tage 2003*, pages 292–304, Berlin, Germany.

Nentwich, C., Emmerich, W., Finkelstein, A., and Ellmer, E. (2003). Flexible Consistency Checking. *ACM Transactions on Software Engineering and Methodology*, 12(1):28–63.

Riggs, S. (2003). Data Quality and XML Validation. In *Proceedings of XML Europe 2003*, London, UK.

Unicode Consortium (2000). *The Unicode Standard: Version 3.0*. Addison Wesley, Reading, Massachusetts.

Whitmer, R. (2003). Document Object Model (DOM) Level 3 XPath Specification. World Wide Web Consortium, Candidate Recommendation CR-DOM-Level-3-XPath-20030331.

Wilde, E. (2003a). 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.

Wilde, E. (2003b). Validation of Character Repertoires for XML Documents. In *Proceedings of the Twenty-fourth Internationalization and Unicode Conference*, Atlanta, Georgia.

Wilde, E. and Steiner, A. (2004). Networking Metaphors for E-Commerce. Technical Report TIK-Report No. 190, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology, Zürich, Switzerland.