HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Laboratory of Software Technology

Henri Sivonen

An HTML5 Conformance Checker

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Helsinki, 7 May 2007

Supervisor and Instructor: Professor Jorma Tarhio

Author:	Henri Sivonen
Department:	Computer Science and Engineering
Major:	Software Systems
Minor:	Strategy and International Business
Title of the thesis:	An HTML5 Conformance Checker
Number of pages:	xiv + 108
Date:	7 May, 2007
Professorship:	T-106 Software Technology
Supervisor:	Professor Jorma Tarhio
Instructors:	Professor Jorma Tarhio

The Web Hypertext Application Technology Working Group (WHATWG) is developing HTML5 and its parallel XML version, XHTML5, as successors for HTML 4.01 and XHTML 1.0. An (X)HTML5 conformance checker is expected to take the role that DTD-based validators have had with earlier (X)HTML. Conformance checking goes beyond the capabilities of DTDs. The WHATWG does not prescribe an implementation strategy for conformance checkers and does not endorse schema languages.

Realizing that no schema language is adequate for describing the conformance requirements for (X)HTML5, a mainly RELAX NG-based implementation approach was chosen nonetheless for this project. In this project, the bulk of the (X)HTML5 language is described as a RELAX NG schema that is supported by a custom datatype library written in Java. A Schematron schema is used alongside RELAX NG for enforcing constraints for which RELAX NG is not suitable. The remaining requirements are enforced by custom code written in Java. For checking HTML5, which is a language on its own and is not an SGML or XML vocabulary, a special-purpose parser was developed so that the XML tools can work on XHTML5-like parse events.

The design of the system is discussed and found to be successful. The ease of expressing and changing the grammar is identified as the main benefit of RELAX NG. The inability to easily fine-tune error messages is identified as a drawback. Schematron is found to be more suitable than RELAX NG for expressing exclusions and referential integrity constraints. A checker for checking the integrity of HTML tables is presented as the main example of a non-schema-based checker implemented in Java.

Keywords: HTML5, conformance checking, HTML, validation, XHTML, XML, WHATWG, RELAX NG, Schematron, SAX, Web

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Henri Sivonen
Osasto:	Tietotekniikka
Pääaine:	Ohjelmistojärjestelmät
Sivuaine:	Yritysstrategia ja kansainvälinen liiketoiminta
Työn nimi:	HTML5-konformanssitarkistin
Sivumäärä:	xiv + 108
Päiväys:	7. toukokuuta 2007
Professuuri:	T-106 Ohjelmistotekniikka
Työn valvoja:	Professori Jorma Tarhio
Työn ohjaajat:	Professori Jorma Tarhio

Web Hypertext Application Technology Working Group (WHATWG) kehittää HTML5:tä ja sen rinnakkaista XML-versiota, XHTML5:tä, HTML 4.01:n and XHTML 1.0:n seuraajiksi. (X)HTML5-konformanssitarkistimen odotetaan ottavan rooli, joka DTD-pohjaisilla validaattoreilla on ollut aiemman (X)HTML:n kohdalla. Konformanssitarkistus menee DTD:iden kykyjä pidemmälle. WHATWG ei määrää toteutusstrategiaa konformanssitarkistimille eikä tue mitään skeemakieliä.

Vaikka mikään skeemakieli ei ole riittävä kuvaamaan (X)HTML5:n konformanssivaatimuksia, pääasiassa RELAX NG -pohjainen toteutustapa valittiin tähän projektiin siitä huolimatta. Tässä projektissa valtaosa (X)HTML5-kielestä kuvataan RELAX NG-skeemana, jota tukee Javalla kirjoitettu räätälöity datatyyppikirjasto. Schematron-skeemaa käytetään RELAX NG:n ohella valvomaan rajoitteita, joihin RELAX NG ei sovellu. Jäljelle jääviä rajoitteita valvotaan räätälöidyllä Javakoodilla. HTML5:n, joka on itsenäinen kieli eikä SGML- tai XML-sanasto, tarkistamiseen kehitettiin tätä tarkoitusta varten jäsennin, jotta XML-työkalut voivat kuunnella XHTML5:n kaltaisia jäsennystapahtumia.

Järjestelmän suunnitteluratkaisuja käsitellään ja ne todetaan onnistuneiksi. Kieliopin ilmaisemisen ja muuttamisen helppous tunnistetaan RELAX NG:n pääeduksi. Kykenemättömyys virheilmoitusten helppoon hienosäätöön tunnistetaan haitaksi. Schematron todetaan RELAX NG:tä soveltuvammaksi ekskluusioiden ja viite-eheysrajoitteiden ilmaisuun. Tarkistin HTML-taulukoiden eheyden tarkistamiseen esitellään pääesimerkkinä Javalla toteutetusta ei-skeemapohjaisesta tarkistimesta.

Avainsanat: HTML5, konformanssitarkistus, HTML, validointi, XHTML, XML, WHATWG, RELAX NG, Schematron, SAX, Web

Acknowledgements

This Master's thesis has been written at the Laboratory of Software Technology of Helsinki University of Technology.

I want to thank Ian Hickson for all his work on HTML5, without which this thesis would not exist.

I wish to thank Elika Etemad (fantasai) for developing the core RELAX NG schema for HTML5, for letting me build upon the schema, for reviewing and commenting on my changes to the schema, and for reviewing drafts of this thesis.

I would also like to thank the Mozilla Foundation for funding this project and Frank Hecker of the Mozilla Foundation for supporting this project.

I want to thank James Clark for developing the Jing validation engine that the software developed in this project is based on.

My gratitude also goes to members of the #turska and #whatwg IRC channels as well as the members of the WHATWG mailing list.

I would like to thank YesLogic Pty. Ltd., SyncRO Soft Ltd. and Oskar Ojala for software that I used to make this thesis publishable.

I wish to thank my instructor and supervisor professor Jorma Tarhio.

Finally, I would like to thank my family.

Helsinki, 7 May 2007
Henri Sivonen

Contents

Glossary

API	Application Programming Interface, a convention for accessing the functionality provided by a program module.	31
ASCC	Academia Sinica Computing Centre, the birthplace of Schematron.	23
ASCII	American Standard Code for Information Interchange, a 7-bit character encoding.	53
Character	An atomic semantic component of text.	7
Character encoding	A way to encode a sequence of code points as a sequence of bytes.	7
Character set	A collection of characters with a code point assigned to each character (strictly a *coded* character set).	7
Code point	An integer that identifies a character.	21
Code unit	A cluster of bits treated as a unit in a character encoding.	54
Conformance checking	The process of checking whether a document satisfies the conformance criteria given in the specification for the language the document is expressed in.	2
Conforming	Satisfying the conformance criteria given in the applicable specification.	2
CSS	Cascading Style Sheets, the style sheet language of the Web.	9
DDML	Document Definition Markup Language, a schema language for XML that was abandoned in favor of the XSD.	21
Document tree	The tree structure embodied in the syntax of an HTML or XML document which can be parsed from the syntax and represented as a concrete data structure.	33

DOM	Document Object Model, an API and the associated data model for representing HTML and XML document trees.	16
DSD	Document Structure Description, a schema language for XML.	20
DSDL	Document Schema Definition Language, a family of ISO standards that define schema languages for XML (including RELAX NG and ISO Schematron).	21
DTD	Document Type Definition, the built-in schema language of SGML and XML.	17
GML	Generalized Markup Language, IBM's predecessor to SGML.	6
HTML	HyperText Markup Language, the language most Web pages are expressed in.	5
HTML5	HyperText Markup Language 5, a new version of HTML drafted by the WHATWG.	14
HTTP	Hypertext Transfer Protocol, the primary transfer protocol of the Web.	59
IANA	Internet Assigned Numbers Authority, an organization that maintains registries of Internet media types, language tags, IRI schemes, etc.	45
IBM	International Business Machines, a corporation.	6
IE	Internet Explorer, Microsoft's Web browser.	14
IETF	Internet Engineering Task Force, a stardardization organization for Internet technologies.	6
IIIR	Integration of Internet Information Resources, an IETF working group that published an early draft specification for HTML.	6
Infoset	The information structure embodied in an XML document, a formalization of the document tree.	18
IRI	Internationalized Resource Identifier, a network resource addressing scheme that allows Unicode characters (as opposed to mere ASCII).	44
ISO	International Organization for Standardization, a standardization organization.	10
JSP	JavaServer Pages, a way of mixing markup literals and Java code.	28

Markup language	A computer language that contains machine-readable annotations (markup) to human-readable text.	1
MSV	Sun Multi-Schema Validator, a validation engine for RELAX NG and some other schema languages for XML.	28
OASIS	Organization for the Advancement of Structured Information Standards, an XML-oriented standardization organization.	21
PDA	Personal Digital Assistant, a product class of portable computing devices.	12
PSVI	Post-Schema Validation Infoset, the infoset of an XML document augmented with datatype information as the result of XSD validation.	20
RELAX	Regular Language description for XML, a schema language for XML which together with TREX was used as the basis of RELAX NG.	21
RELAX NG	A grammar-based schema language for XML.	21
REST	Representational State Transfer, an architechtural style for distributed systems embodied in the design of HTTP.	64
RFC	Request for Comments, a numbered memorandum published by the IETF.	6
SAX	Simple API for XML, a de facto standard parse event-based API through which an XML parser reports the infoset of the parsed XML document to an application as callbacks.	31
Schema	A formal definition (expressed in a schema language) that partitions the set of all possible XML documents into two disjoint sets so that each document is in either set: documents that are valid according to the schema and documents that are not valid according to the schema.	17
Schema language	A computer language for expressing a schema.	17
Schematron	An assertion-based schema language for XML.	23
SGML	Standard Generalized Markup Language, a syntax framework for defining markup languages.	5
SVG	Scalable Vector Graphics, an XML language for representing two-dimensional vector graphics.	13

TREX	Tree Regular Expressions for XML, a schema language for XML which together with RELAX was used as the basis of RELAX NG.	21
Unicode	The universal coded character set.	53
UTF-8	Unicode Transformation Format with 8-bit code units, the preferred character encoding for information interchange.	53
UTF-16	Unicode Transformation Format with 16-bit code units, the character encoding used for strings in Java.	54
UTF-32	Unicode Transformation Format with 32-bit code units, a character encoding where a single Unicode code point is always encoded as a single code unit.	54
W3C	World Wide Web Consortium, an industry consortium that publishes specifications for World Wide Web technologies.	8
Valid	Satisfying a schema.	17
Validation	The process of checking if a document is valid according to a schema.	17
WDG	Web Design Group, a group that offers materials and tools (a validator, in particular) related to HTML.	26
WHATWG	Web Hypertext Application Technology Working Group, a collaborative group of browser vendors, Web developers and other interested parties that works on the next generation of HTML.	14
WXS	W3C XML Schema, a schema language for XML.	19
XHTML	Extensible HyperText Markup Language, HTML reformulated as an XML language.	10
(X)HTML	(Extensible) HyperText Markup Language, a catch-all term for both HTML and XHTML.	12
XHTML5	XML serialization of HyperText Markup Language 5, the XML language defined in parallel with HTML5.	14
(X)HTML5	(XML serialization of) HyperText Markup Language 5, a catch-all term for both HTML5 and XHTML5	14
XML	Extensible Markup Language, a syntax framework for defining markup languages, a stand-alone simplification of SGML.	10

XSD	19
XML Schema Definition, the filename extension for and, consequently, the common way to refer to W3C XML Schema.	
XSLT	28
Extensible Stylesheet Language Transformations, a programming language designed for transforming XML documents into different XML documents.	

Chapter 1

Introduction

The Web Hypertext Application Technology Working Group (WHATWG) is developing HTML5 and its parallel XML version, XHTML5, as successors for HTML 4.01 and XHTML 1.0. HTML5 and XHTML5 are defined by the combination of WHATWG's *Web Applications 1.0* [WebApps] and *Web Forms 2.0* [WebForms2] specifications. To be successful, a new markup language not only needs support from browsers, it also needs tools that support authoring. Authoring-side tools include editors, content management systems and quality assurance tools for checking the correctness of markup. This thesis focuses on the last.

1.1 Motivation

Web authors tend to make mistakes when writing HTML. The vast majority of HTML documents on the Web are syntactically incorrect. A test of the HTML5 parsing algorithm on several billion documents spidered by Google indicated that 93% of documents had errors on the lowest levels of the syntax [Several]. (Documents in the remaining 7% may well have higher-level errors that are not found by the parsing algorithm and would require a full conformance checker to find.)

Even though most Web content is broken without hope of repair and browsers will do *something* with *any* input purporting to be HTML, it is still useful to provide a quality assurance tool for authors. Even if browsers adopt the well-defined error-recovering processing models of HTML5, authors generally do not make errors on purpose in order to elicit particular error recovery response. Silent recovery from inadvertent mistakes – even if deterministic and well defined – may still confuse an author who did not mean to invoke error recovery. The issue becomes more apparent when an author uses a style sheet or a script that assumes the document to be correct. Therefore, it is worthwhile to provide a conformance checker that helps authors find their mistakes.

1.2 Objectives

The functional objective of the project described in this thesis was developing a partial (X)HTML5 conformance checker that is comprehensive enough to demonstrate that it can be taken to completion once (X)HTML5 itself has stabilized. The research goals were 1) finding out if a hybrid implementation based both on schemata and on custom code developed in a general-purpose programming language is feasible and 2) finding out if an XML toolchain can be successfully applied to checking the non-XML serialization of HTML5. Only markup checking without executing scripts is considered due to the halting problem [Computable].

1.3 Methods

For HTML 4.01 and XHTML 1.0, validators based on Document Type Definitions (DTDs), the built-in schema language of SGML and XML, have traditionally been used as the quality assurance tools for checking correctness even though they do not check for all machine-checkable conformance requirements. For (X)HTML5, a conformance checker is expected to take the role that DTD-based validators have had with earlier (X)HTML. Conformance checking goes beyond the capabilities of DTDs.

The WHATWG does not prescribe an implementation strategy for conformance checkers and does not endorse schema languages. Not only are schema languages unendorsed but also they are seen as being clearly inadequate. Therefore, a non-schema-based implementation strategy is implied. Yet, as an initial impression, abandoning schemata altogether just because they cannot be used for checking every machine-checkable constraint seems overly drastic. Hence, I chose a hybrid approach that uses schemata for what they are good for and uses a non-schema-based implementation strategy for what schemata are not good for.

I chose RELAX NG as the primary schema language, and Schematron as a supporting schema language. Using RELAX NG for document-oriented schemata (as opposed to databinding-oriented schemata) had gained acceptance as the best practice among users of XML schema languages. Schematron had gained popularity as a language for refining RELAX NG schemata. Elika Etemad had already started a project for developing a RELAX NG schema for HTML5 [HTML5RNG]. Moreover, I had already developed a service that allows Web users to validate XML documents against arbitrary RELAX NG and Schematron 1.5 schemata [ValidatorAbout]. I had developed the service in the Java programming language due to the excellent availability of XML tools for Java. I chose Etemad's schema project and the service I had already developed as starting points for this thesis project. Since I had written my pre-existing software in Java, it followed that I would also write the new non-schema code in Java.

The parsed syntax tree for HTML5 and the parsed syntax tree for XML are very similar. Since reusable tools exist for XML, I decided to use XML tools and to map HTML5 documents to equivalent XHTML5 representations in the parser. To this end, I wrote an HTML parser (page 32) that acts as if it were an XML parser parsing XHTML.

1.4 Availability of the Software

The generic validation service that I used as the basis of the conformance checker is usable online at http://hsivonen.iki.fi/validator/.

The work I did in order to add HTML5 support to the generic validation service included:

- an HTML parser (page 32)
- significant work on a RELAX NG schema for (X)HTML5 (page 38)
- a Schematron schema complementing the RELAX NG schema (page 46)
- a RELAX NG datatype library for HTML5 datatypes (page 43)
- non-schema-based checkers for requirements that schemata cannot express (page 49)

The software I developed is Free Software / Open Source. The source code may be obtained by following links from http://hsivonen.iki.fi/validator-about/.

The product of this thesis project is usable online at http://hsivonen.iki.fi/validator/html5/.

1.5 Organization of this Thesis

This thesis has two thematic parts. The first part (the next three chapters) reviews the context of this work. HTML5 is placed in historical context, schema languages for XML are reviewed and prior work on online markup checking services is reviewed. The second part (the last five chapters) focuses on the software implemented in this project. The implementation of the software, its shortcomings, and its applicability to other contexts are discussed. Finally, the need for future work is reviewed and the conclusions given.

Chapter 2

History of HTML Leading to HTML5

This chapter reviews the history of Hypertext Markup Language (HTML) leading to HTML5.

HTML is, in principle, a semantic markup language. That is, it encodes, for example, that a particular piece of text is a heading as opposed to encoding the exact presentation. HTML has never been only about presentation and has never been only about encoding the profound semantics of text. The positioning of HTML somewhere in between these extremes has shifted in both directions with different versions.

Since one of the major changes in HTML5 is the way the specification deals with parsing and the stance the specification takes with respect to Standard Generalized Markup Language (SGML [ISO8879]), each version of HTML prior to HTML5 is summarized in terms of the key features introduced and in terms of the *stated* relationship to SGML or XML (Extensible Markup Language [XML]). SGML is a syntax framework for defining markup languages. XML is a simplification of SGML. SGML and XML define the parsing layer of markup language processing.

2.1 Early HTML

In this review, HTML versions prior to HTML 4 are considered "early", as they are no longer in active use when new documents are created.

2.1.1 Initial HTML at CERN

Tim Berners-Lee invented the Web in 1989. He released the first version of his browser in 1990 [Raggett]. The system used HTML, but the language was not formally specified at first. Tim Berners-Lee designed HTML using ideas from SGML [Weaving]. However, HTML was not layered on top of the SGML standard but rather used a similar syntax without being a true application of SGML.

The element names available in HTML were largely taken from SGMLguid, an application of SGML used at CERN. SGMLguid, in turn, was similar to Waterloo

SCRIPT GML [WaterlooGML], a GML language specified at University of Waterloo. (GML [Generalized] was IBM's predecessor to SGML.) [EarlyHistory] There are also similarities with the language given in the tutorial of the SGML standard [ISO8879].

2.1.2 The IIIR Draft

Tim Berners-Lee and Dan Connolly wrote an Internet Draft specification for HTML as part of the activity of the Integration of Internet Information Resources (IIIR) working group of the Internet Engineering Task Force (IETF). The Internet Draft was published in June 1993. [IIIR-HTML]

The draft said that HTML was defined in terms of SGML. However, the specification did not specify an HTML document as a conforming SGML document entity but instead said how to construct an SGML document from an HTML file [IIIR-HTML]. The draft also suggested that an HTML parser would not need to be a full SGML parser but a parser that only deals with the document instance after the DTD [IIIR-HTML]. W. Eliot Kimber, an SGML expert, challenged the purity of the drafted HTML approach in terms of SGML [ToBeDeleted]. The stated approach was changed in later specifications to make an HTML file directly an SGML document. However, browsers continued to use special-purpose parsers (as opposed to SGML parsers) as before. The mailing list discussions about the relationship of HTML to SGML are summarized in [Cascading].

The IIIR draft already included the `IMG` element for images. The `P` element was defined as an empty element that indicates paragraph breaks. As an interesting detail, the `XMP`, `LISTING` and `PLAINTEXT` elements for including verbatim text in HTML were considered *obsolete* as early as the IIIR draft (although they would still show up in the HTML5 parsing algorithm over a decade later [WebApps]). [IIIR-HTML]

The draft expired and did not reach the RFC status.

2.1.3 HTML+

Dave Raggett, one of the participants of the www-talk for discussing Web matters, visited Tim Berners-Lee at CERN to discuss further development face to face. Based on the discussion, Raggett drafted a new version of HTML called HTML+. [Raggett]

The draft specification for HTML+, published in late 1993, specifically stated that HTML+ was "based on the Standard Generalized Markup Language". It also had a Document Type Definition (DTD), a formal grammar expressed using the built-in schema language (page 17) of SGML. In theory, having a DTD enabled the use of SGML parsers. However, the draft implied that there would be "HTML+ parsers" which would be different from "other SGML parsers". HTML+ explicitly excluded SGML minimization features. It used the `P` element as a paragraph *container* but said that authors may think of the `P` tag as a paragraph *separator*. [HTMLplus]

HTML+ had a number of elements that never entered into actual usage, such as `BYLINE`, `ONLINE`, `PRINTED`, and `ABSTRACT`. As a curious detail, HTML+ included an element called `IMAGE`, which used the element content as the alternative text – a feature that would still be discussed over a decade later. HTML+ attempted to address the issue of mathematical formulae, but the coverage of types of formulae was not particularly comprehensive. [HTMLplus]

HTML+ defined markup for tables. The table markup is roughly what was later adopted in HTML 4. HTML+ also defined markup for forms similar to what was actually adopted in browsers. However, the field types also included types that were not adopted, such as `URL`, `DATE` and `SCRIBBLE` (for drawing). [HTMLplus]

Mainstream browsers never adopted HTML+. However, at the first World Wide Web conference it was agreed that the ideas from HTML+ should be carried forward. [Raggett]

2.1.4 HTML 2.0

Dan Connolly had advocated a cross-browser HTML specification at the first World Wide Web conference in early 1994. Subsequently, the IETF formed a working group to specify HTML. The working group – with Connolly in the lead – defined HTML 2.0 based on the then-current practice. The HTML 2.0 draft was published in July 1994. [Raggett]

The HTML 2.0 specification reached the RFC status in November 1995. The specification stated that "HTML is an application of ISO Standard 8879:1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML)." [RFC1866] However, it was too late to make browsers use SGML parsers. Instead, browsers continued to use special-purpose HTML parsers without standardized error recovery behavior. HTML 2.0 included a DTD, but the DTD was of no interest to browsers.

Unlike the elements proposed in HTML+, the elements of HTML 2.0 were (and still are) actually supported by browsers. HTML 2.0 included forms but did not include tables, which had been proposed in HTML+. Regardless, Netscape implemented tables in its browser in the HTML 2.0 era and made tables popular.

HTML 2.0 established that the document character set of HTML is ISO 10646 regardless of the *character encoding* used to transfer the document. (The character allocations in ISO 10646 track the allocations of Unicode [ISO10646][Unicode].) However, the internationalization of HTML 2.0 was not fully addressed in the HTML 2.0 specification itself, and a standards track RFC that extended HTML 2.0 to address internationalization issues was published in late 1997 [RFC2070].

2.1.5 HTML 3.0

To keep the Web unified amidst product development by various competing vendors, an industry consortium called The World Wide Web Consortium (W3C) was founded in 1994 to develop specifications for the Web. [Weaving]

Dave Raggett – this time representing the W3C – edited a specification called HTML 3.0, which carried forward the ideas of HTML+ [Raggett]. To support the use of style sheets, HTML 3.0 introduced the `STYLE` element and the `CLASS` attribute, which lived on in HTML 4 [Raggett]. An HTML 3.0 draft was published through the IETF as an Internet Draft [HTML30].

Meanwhile, Netscape extended HTML on its own. In particular, its extensions included presentational features instead of adopting style sheets.

HTML 3.0 did not match what was being implemented in browsers. The draft specification was abandoned and never reached the RFC status [Raggett]. Even though HTML 3.0 as a whole was dropped, a specification for HTML tables (as an extension to HTML 2.0) was published as an experimental RFC. [RFC1942]

2.1.6 HTML 3.2

In November 1995, representatives of browser vendors and the W3C formed an HTML working group at the W3C. The following month, the IETF HTML working group was disbanded. [Raggett]

In January 1997, the W3C published the specification for HTML 3.2 as a Recommendation. Unlike HTML 3.0, HTML 3.2 documented actual practice that had grown as extensions to HTML 2.0. The specification itself stated: "HTML 3.2 aims to capture recommended practice as of early '96 and as such to be used as a replacement for HTML 2.0 (RFC 1866)." [HTML32]

HTML 3.2 continued to say that HTML was an application of SGML: "HTML 3.2 is an SGML application conforming to International Standard ISO 8879 – Standard Generalized Markup Language. As an SGML application, the syntax of conforming HTML 3.2 documents is defined by the combination of the SGML declaration and the document type definition (DTD)." [HTML32] However, even the specification itself admitted that SGML-compliance of user agents was not part of the actual practice as of early '96 by noting: "Note that some user agents require attribute minimisation for the following attributes: `COMPACT`, `ISMAP`, `CHECKED`, `NOWRAP`, `NOSHADE` and `NOHREF`. These user agents don't accept syntax such as `COMPACT=COMPACT` or `ISMAP=ISMAP` although this is legitimate according to the HTML 3.2 DTD." [HTML32]

In documenting the actual practice, HTML 3.2 included presentational features, such as the `FONT` element, that would later be deprecated.

2.2 Contemporary HTML

The versions of HTML discussed above are of historical interest and are not in active use for creating new documents. The versions in current use start with HTML 4.

2.2.1 HTML 4

HTML 4.0 was published as a W3C Recommendation in December 1997 [HTML40]. The specification formalized existing features that had been introduced by browser vendors but also introduced new features of its own. HTML 4.0 was revised without incrementing the version number, and the revision was published in April 1998 [HTML40rev]. Another revision called HTML 4.01 became a W3C Recommendation in December 1999 [HTML401].

Again, the specification said, "HTML 4 is an SGML application conforming to International Standard ISO 8879 – Standard Generalized Markup Language." [HTML401] Yet, the specification acknowledged the reality that user agents in general are not conforming SGML systems: "SGML systems conforming to [ISO8879] are expected to recognize a number of features that aren't widely supported by HTML user agents. We recommend that authors avoid using all of these features." [HTML401]

HTML 4 deprecated presentational features such as the `FONT` element that had made its way to a W3C Recommendation less than a year before the first version of HTML 4. In principle, HTML 4 tried to backpedal on the point of presentational features to where HTML 2.0 had been – with the intent of leaving presentation to style sheets such as Cascading Style Sheets (CSS) [Cascading], which had been published as a W3C Recommendation [CSS1] the year before.

HTML 4 without the deprecated features was termed "Strict" and HTML 4 with the deprecated presentational features was termed "Transitional". In practice, the deprecated features continue to be used nine years later even though CSS has been very successful both in terms of acceptance by Web authors and in terms of implementations.

HTML 4 included features for adding more structure to tables, for adding more structure to forms, and for marking up insertions and deletions. HTML 4 adopted the model proposed in the experimental RFC on HTML tables [RFC1942] dropping a few presentational attributes. Internationalization features, including support for bidirectional text (e.g. for Hebrew and Arabic), were adopted from the standards track RFC on the internationalization of HTML [RFC2070].

HTML 4 introduced the `OBJECT` element, which was supposed to eventually replace `IMG`, `APPLET` and the Netscape `EMBED` elements. `EMBED` did not fit together with an SGML DTD, because it could take arbitrary attributes. However, in practice, browsers continued to support `EMBED`, and even today browsers do not fully support `OBJECT` as designed.

HTML 4 formalized frames, which had been introduced by Netscape and were discredited [Frames] even before their inclusion in HTML 4. Additionally, HTML 4 included `IFRAME` from Microsoft.

2.2.2 ISO HTML

In 2000, ISO standardized its own version of HTML by referencing a subset of HTML 4.0 as defined by the W3C but also making changes other than merely subsetting in the DTD [ISO15445]. A technical corrigendum changed the references to HTML 4.01 [ISO15445TC1].

In practice, ISO HTML is only of curiosity value, since Web authors have largely ignored it.

2.2.3 XHTML 1.0

Extensible Markup Language (XML) 1.0 [XML] was published as a W3C Recommendation in February 1998 [AXML]. XML is a simplification of SGML that stands alone without making a normative reference to SGML. Since HTML was defined as an application of SGML and the W3C now had its own replacement for SGML, the W3C decided to swap the markup language framework from underneath HTML. The result was XHTML 1.0 – a reformulation of HTML 4 in XML. XHTML 1.0 became a Recommendation in January 2000 [XHTML10].

XHTML 1.0 includes the features that were deprecated in HTML 4. That is, XHTML 1.0 has three versions just like HTML 4: Strict, Transitional and Frameset.

Appendix C. To be compatible with existing HTML user agents, the XHTML 1.0 specification included compatibility guidelines commonly known as "Appendix C". Appendix C limits the syntactic sugar permitted by XML 1.0 so that an XHTML 1.0 document that adheres to Appendix C could be processed by existing HTML user agents if served as `text/html` [RFC2854] media type. Appendix C relies on the fact that browsers do not actually process `text/html` as SGML.

Appendix C made it seem that XHTML 1.0 was succeeding by being adopted immediately. Obviously, since the browsers gained no new capabilities, using XHTML 1.0 could not actually deliver any true benefits over HTML 4 in user agents designed for HTML. No XML processor was involved despite the XHTML 1.0 being a reformulation in XML. In fact, the HTML WG of the W3C gave an explicit opinion that browsers should not try to process documents served as `text/html` using a real XML processor [Sniffing].

There are experts close to the development of browser engines who have discredited the practice of serving XHTML as `text/html`, because authors are not actually invoking any new kind of processing but end up making documents that rely on error handling and would not work with the new kind of processing (e.g. [Harmful] and [Understanding]).

Processing as XML. Later on, Mozilla, Opera and Apple (three of the top four browser vendors after the demise of Netscape) took the XML nature of XHTML seriously and implemented support for XHTML 1.0 using a real XML processor. A real XML processor is used when the document is served using the `application/xhtml+xml` [RFC3236] media type (instead of `text/html`).

Serving pages as `application/xhtml+xml` has not become popular among Web authors for three reasons. First, since XHTML 1.0 is a reformulation of HTML 4 on top of another markup language framework, it (alone) does not enable new interesting things in the browser. This means there is not a compelling technical advantage to be gained from using XHTML 1.0 served as `application/xhtml+xml` over HTML 4.01 served as `text/html`. Second, the browser engine with the largest desktop market share (Trident, the engine of Microsoft's Internet Explorer) still does not support `application/xhtml+xml`. (Browser market share is difficult to define and measure, but the global usage share of Internet Explorer is estimated to be above 80% in early 2007 [OneStat][TheCounter].) Third, when a real XML processor is used, an error is reported if the document violates the well-formedness constraints of XML. Often, the document is not displayed at all if it violates these syntactic constraints. This means that a small authoring error breaks the document completely. In contrast when content is served as `text/html`, browsers try to recover from markup errors.

Moreover, there are subtle differences in the ways Cascading Style Sheets [CSS2] and the Document Object Model [DOM2] API exposed to JavaScript interact with `text/html` and `application/xhtml+xml`. Differences involve issues such as case-sensitivity and whether elements are in a namespace [MozFAQ]. In addition, `document.write()`, which allows scripts to insert data into the character stream being parsed, does not work in XML. In practice, scripts written naïvely for XHTML served as `text/html` do not work when the document is served as `application/xhtml+xml`.

2.2.4 Modularization

The W3C decided to abandon the development of the old non-XML HTML and to only develop XHTML. After the reformulation of HTML 4 in XML, which became XHTML 1.0, the W3C HTML working group proceeded to modularize XHTML. Modularization meant dividing XHTML into logical parts such as Hypertext Module and Image Module and rewriting the previously monolithic DTD as multiple files following the logical module partitioning.

The rationale for the modularization was based on a view that one size of XHTML did not fit all client platforms. In the words of the specification itself: "This modularization provides a means for subsetting and extending XHTML, a feature needed for extending XHTML's reach onto emerging platforms." [M12N] The foremost "emerging platforms" were mobile phones, which were thought to be unable to host a browser for full HTML. The rationale for modularization implicitly assumes a walled garden-style world view of the owners of mobile phone networks where a client platform design can dictate a language subset used on the network. Such a view assumes a separate “Mobile Web”, because – quite obviously – the World Wide Web would still use full HTML or XHTML.

XHTML Basic. *XHTML Basic*, published in late 2000, defines a baseline for XHTML languages built on top of the Modularization. XHTML Basic is a subset of XHTML 1.0. The specification itself lists “mobile phones, PDAs, pagers, and settop boxes” as target devices. [XHTMLBasic]

XHTML 1.1. XHTML 1.1 [XHTML11], published in 2001, was the first (X)HTML specification since HTML 4 that introduced a new feature. XHTML 1.1 includes the XHTML modules that correspond to XHTML 1.0 Strict. Additionally, XHTML 1.1 includes the XHTML Ruby Annotation module [Ruby] for expressing a type of text annotations used in East Asia.

Microsoft’s Internet Explorer for Windows 5.0 (and later) supports a draft version of Ruby markup when used in `text/html` documents [RubyIE]. However, the most notable browsers that support `application/xhtml+xml` do not support Ruby. Therefore, XHTML 1.1 has failed to make a significant practical impact.

XHTML Mobile Profile. In 2001, WAP Forum – a consortium of mobile phone manufacturers – defined a superset of XHTML Basic called *XHTML Mobile Profile* [XHTML-MP]. The profile did not follow the prescribed XHTML module boundaries.

The specification defined `application/vnd.wap.xhtml+xml` as the media type for XHTML Mobile Profile documents [XHTML-MP], but this media type has not been officially registered. The profile has not made a notable impact on the World Wide Web.

2.3 HTML5

The above review explains the context in which HTML5 was born.

The prior versions of HTML had officially been applications of SGML, but browsers were actually using special-purpose HTML parsers rather than SGML parsers. The SGML basis only gave guidance on what document tree was expected when a document was conforming. There was no realistic specification for parsing HTML when the input was erroneous (which it most often is). Browser vendors had to reverse engineer the behavior of the current market leader. This has caused interoperability problems.

Moreover, significant new features had not been introduced in years as the work had focused on reformulating the syntax as XML. Yet, documents purporting to use the reformulated XML syntax were still served as `text/html`, so browsers kept using the same special-purpose parsers as before. The usage of `application/xhtml+xml` had failed to take off.

There was demand for new features for HTML and demand for the recognition of the fact that `text/html` content was parsed neither as SGML nor as XML but had a syntax of its own.

2.3.1 The Mozilla/Opera Joint Position Paper

The balance of power in the W3C had shifted from traditional desktop browser vendors to various other interest groups such as makers of software for mobile walled gardens and developers of "rich client" technologies that could be deployed on intranets but that were not used by the general public on the Web. This had led to a situation where the focus was more on the "Semantic Web", "Web Services" and "Mobile Web" than on what is usually considered "the Web". As a result, the development of the Web itself had been neglected.

In June 2004, the W3C held a workshop on Web Applications and Compound Documents. The Mozilla Foundation and Opera Software – the two most active browser vendors in the W3C at the time – submitted a joint position paper noting the "rising threat of single-vendor solutions" and calling for seven principles to be followed in the design of Web Applications Technologies [JointPosition]. (At the time Microsoft – a notable browser vendor itself – was pushing a single-vendor solution code named Avalon [MS-WebApps] and Apple was catching up having entered the market only recently.)

The first one of the seven principles in the Mozilla/Opera position paper was "Backwards compatibility, clear migration path" [JointPosition]. The transition from HTML 4 to XHTML 1.0 had not worked out smoothly as discussed earlier (page 10). In addition, XForms [XForms], the W3C's successor for HTML forms, did not provide backwards compatibility or a clear migration path. Moreover, the HTML working group was working on XHTML 2.0 [XHTML20], which was designed to be incompatible with XHTML 1.x, even though the transition to XHTML 1.x served as `application/xhtml+xml` was not complete.

The position paper called for well-defined error handling – something that had never been addressed for HTML. The paper took a position in favor of graceful recovery (as in CSS [CSS2]) and against the Draconian error policy of XML.

The paper called for every feature to be backed by a practical use case and for the specification process to be open. This was in contrast with including features that are "nice to have" and making decisions on the W3C's member-only mailing lists.

The paper took a position against device-specific profiles. This was in direct contrast with the *Modularization of XHTML* (page 11) [M12N] as well as mobile profiles of other W3C deliverables such as Scalable Vector Graphics (SVG [SVG]). The paper also took a position more favorable to scripting (JavaScript [JavaScript] in practice) than what has been the general line in the W3C.

The paper stated two design principles for compound documents (documents that mix different XML vocabularies): "Don't overuse namespaces" and "Migration path". The latter was related to the problems with the HTML to XHTML migration discussed above. The position paper was dismissive of schema languages.

The paper went on to list specific features that a Web application host environment should provide. It made several references to XBL, which has been a very politicized language (but is now on track to become a W3C Recommendation [XBL2]).

2.3.2 The WHATWG is Formed

The proposal presented by Opera Software and the Mozilla Foundation was not well received at the W3C. At the end of the second day of the workshop, a poll was held on the topic of the joint position paper: whether the W3C should develop extensions to HTML, CSS and the DOM as proposed. Of the 51 attendees of the workshop, 8 voted in favor of the motion and 11 voted against. When the motion was slightly reformulated, 14 voted against. [cdf-ws-minutes2]

Two days after the vote at the workshop, The Web Hypertext Applications Technology Working Group (WHATWG) and its public mailing list were publicly announced. The group was described as "a loose, unofficial, and open collaboration of Web browser manufacturers and interested parties". The stated intent was creating specifications for implementation in "mass-market Web browsers, in particular Safari, Mozilla, and Opera". [WHAT-Ann]

The initial (invite-only) membership of the WHATWG consisted of individuals affiliated with Apple, Mozilla and Opera Software. (Ian Hickson, the editor of the WHATWG specifications, later moved to Google.) However, in the view of the Web held by the WHATWG, there is also a fourth mass-market browser: Microsoft's Internet Explorer – the leader in market share. Microsoft has not been participating in the WHATWG despite having been invited. The publicly stated reason was that the WHATWG lacked a patent policy [Wilson]. Dean Edwards, a Internet Explorer expert not affiliated with Microsoft, joined the WHATWG later [NewMember].

Even though the group of WHATWG members is invite-only, anyone is allowed to join the WHATWG mailing list and contribute technically, which makes the process open. The editor acts as a benevolent dictator who writes the specifications taking into account the contributions. The WHATWG members "provide overall guidance" [WHAT-Charter], which means the power to impeach and replace the editor of the specifications.

Microsoft is not expected to implement the WHATWG specifications in Internet Explorer in the near term. Instead, the implementations of the WHATWG specifications for IE are expected to be built by teams not affiliated with Microsoft using the extensibility mechanisms provided by Microsoft in IE. [IEcompat]

I share the view of the Web that holds WebKit, Presto, Gecko and Trident (the engines of Safari, Opera, Mozilla/Firefox and IE, respectively) to be the most important browser engines.

2.3.3 The WHATWG Specifications

The WHATWG has two specifications in development and another two that are expected in the future. [WHAT-Charter]

The two specifications being developed are *Web Forms 2.0* [WebForms2] and *Web Applications 1.0* [WebApps]. Web Forms 2.0 is an update to HTML 4.01 forms. Web Applications 1.0 is a re-specification of HTML that both constrains and extends HTML. The language specified by Web Forms 2.0 and Web Applications 1.0 taken together is referred to as (X)HTML5. It is expected that Web Forms 2.0 will be eventually be folded into the Web Applications 1.0 specification.

The two expected future specifications are *Web Controls 1.0* for creating new widgets and *CSS Rendering Object Model* for defining programmatic access to the CSS rendering tree. [WHAT-Charter]

Web Forms 2.0. Web Forms 2.0 extends HTML forms with new features. The HTML forms as of HTML 4.01 are considered "Web Forms 1.0". Web Forms 2.0 is not a standalone specification. Instead, it specifies updates to HTML 4.01 and the DOM. The choice of updates is based on what has been identified as common needs and what can be implemented as a script-based library for Internet Explorer [IEcompat].

The most obviously visible new features are new input field types. For example, there are new inputs for dates that can be implemented in browsers by popping up a platform specific calendar widget. The new input types are backwards compatible in the sense that unknown input types degrade into text inputs in legacy browsers. Simple constraints on the values of the input field can be declared and checked by the browser without the form author having to resort to scripting. For complex restrictions, new integration points for scripts are provided.

In addition to the new input types, there is also a repetition model for adding and removing repeating sets of fields from the form without scripting. A new XML form submission format in introduced. The format can also be used for pre-loading values into the form fields.

Web Forms 2.0 is the most mature part of the new features of HTML5. It has already been implemented and shipped in the Opera 9 browser [Opera9].

Web Applications 1.0. Web Applications 1.0 is the main specification for HTML5. The name of the specification highlights the Web application focus of the new features. An XML-based parallel language called XHTML5 is specified alongside HTML5.

The specification has two general areas that are intertwined. On the one hand, new features are specified. On the other hand, existing features are specified in detail that was absent from previous specifications. When existing features are respecified, the behavior of the four notable mass-market browsers is reverse engineered and the specification is made compatible with the existing practice. New features are based on expected use cases.

As the name of the specification suggests, there are new features aimed for Web applications. New application-oriented markup includes `canvas` for establishing a canvas onto which scripts can draw, `menu` and `command` for building menus, `meter` and `progress` for representing gauges and progress indicators, `datagrid` for complex data display widgets, `details` for additional information that can be hidden and `event-source` for indicating that the page listens to server-sent remote events.

In addition to new elements, the specification includes a number of scripting APIs for Web applications, but they are outside the scope of this thesis as documents are checked without executing scripts.

The addition of new elements for document structure is based on an analysis of common `class` attribute values as used on the Web [Stats]. New elements for document structure include `section` for document sections, `nav` for identifying page navigation, `article` for marking up standalone parts of page content, `aside` for tangential notes, `header` for complex headers and `footer` for page footers.

There are also new elements that do not quite fit to the groups given above: `figure` for grouping figures with captions, `time` for associating a machine-readable point in time with human-readable text designating a point in time (e.g. for use in conjunction with microformats [Microformats]) and `m` for marking highlighted text. The `embed` element for (typically) plug-in-rendered content is legitimized.

Finally, HTML5 specifies a `text/html` parsing algorithm in meticulous detail. The algorithm is designed to work with existing erroneous Web content in a way that is compatible with existing browser behavior. In general, the algorithm strives to be compatible with the behavior of Microsoft Internet Explorer to the extent possible while still keeping the resulting data structure as a DOM [DOM2] *tree* that does not have hidden annotations.

The requirement of a tree that does not have hidden annotations fits into the architecture of all contemporary browsers and is the design already used in Gecko and WebKit (the engines of Firefox and Safari). The way the HTML5 parsing algorithm deals with misnested tags is based on the behavior of WebKit. Presto (the engine of Opera) appears to have hidden annotations in the tree when the input was malformed [SoupDOM]. Trident (the engine of Internet Explorer) does not guarantee a tree model [SoupDOM]. Since Trident uses a non-tree data structure but the other engines assume a tree, the behavior of the leader in market share cannot be adopted exactly without requiring drastic architectural changes in the other engines.

Chapter 3

Schema Languages

In this chapter, XML schema languages are reviewed in order to put the choice of RELAX NG and Schematron in context.

A schema is a formal definition that partitions the set of all possible XML documents into two disjoint sets so that each document is in either set: documents that are valid according to the schema and documents that are not valid according to the schema. A computer language for expressing a schema is called a schema language. The process of checking whether a given XML document is valid according to a schema is called validation.

The notion of a schema could be generalized to mean any abstract partitioning of the set of all possible XML documents into two disjoint sets. However, in common usage "schema" means a formal definition expressed in a schema language. This less abstract notion is used in this thesis.

There are two main classes of schema languages: grammar-based schema languages and schema languages that are not grammar-based. The properties of grammar-based schema languages in terms of mathematics and formal language theory as well as the related validation algorithms are discussed in [Taxonomy]. In the article, three types of grammar-based languages are identified: local, single-type and regular (from least expressive to more expressive). The article identifies three classes of schema languages in addition to grammar-based languages: special-purpose languages dedicated to a particular kind of information that may be represented as XML, languages for representing identity constraints and languages for namespace-based validation dispatching.

3.1 DTDs

XML has a built-in schema language that consists of DTDs [XML]. DTD is short for Document Type Definition. However, for reasons explained below, it does not actually define the type of the document in a useful way. Unlike many later schema languages for XML, DTDs are not expressed as XML elements and attributes. Rather, the DTD syntax is separate from the syntax for elements and attributes. (It is sometimes said that DTDs do not have XML syntax, but this is misguided in the sense that the DTD syntax is part of XML.)

DTDs are grammar-based. The expressiveness of DTDs is relatively weak compared to other schema languages. In [Taxonomy], DTDs are classified to be of the type “local”, which is the weakest grammar type. A grammar-based schema language is classified as “local” if there are no competing grammar productions. Two non-terminal productions compete if they have the same terminal on the right-hand side of the productions. The practical consequence is that the content model of an element cannot depend on the context of the element in the document tree.

SGML DTDs were slightly more expressive than XML DTDs. XML removed some of the more complex DTD features. In addition, XML made it possible to parse documents without the parser knowing the DTD grammar for the document. Unlike with XML, SGML parsers needed to know the DTD grammar in order to be able to parse the document.

3.1.1 Infoset Augmentation

DTD-based validation is intertwined in the process of parsing XML. Instead of merely checking that the structure of the document meets the constraints of the grammar, DTDs have features that augment the infoset. Infoset means the information structure embodied in an XML document [Infoset]. Infoset augmentation means that the validation process adds some information that is reported to the application compared to the situation where the document is parsed without any validation formalism between the document and the application. Infoset augmentation is problematic in three ways.

First, the DTD is usually included in the document by reference but the XML 1.0 specification makes processing such external references optional. As a result, the application sees different data depending on whether the DTD is being processed or not.

Second, being able to attach datatypes to attributes requires that there not be two derivations for a given document that would assign conflicting datatypes to a given attribute. To avoid this problem, the grammars expressed as DTDs are required to be unambiguous. This requirement is restrictive and would be unnecessary for pure non-infoset-augmenting validation.

Third, since one of the key features of XML is that a document can be parsed even if it does not have a DTD and since the DTD is supplied by the document itself, an application cannot trust documents to supply DTDs with particular infoset augmentation features. For example, a consuming application cannot trust incoming documents to declare a particular attribute to have the type `ID` or to declare default values for particular attributes. Therefore, applications cannot rely on DTD-based infoset augmentation taking place, which severely limits the usefulness of such infoset augmentation.

3.1.2 Datatyping

XML DTDs allow a primitive form of attribute datatyping. The possible datatypes are `CDATA`, `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NMTOKEN` and `NMTOKENS`. (SGML DTDs have more datatypes.) Additionally, the value of an attribute can be constrained to be one of several enumerated tokens. `CDATA` means an unconstrained string. Cross-references whose integrity is checked are represented using the `ID`, `IDREF` and `IDREFS` types. (There is a single document-wide namespace for IDs established by the `ID` type. A particular reference cannot be constrained to point to elements of a certain type only.) The only datatypes that constrain the allowed lexical space of the attribute value without involving any referential semantics are `NMTOKEN` and `NMTOKENS`. `NMTOKEN` is merely a token that satisfies a particular, relatively arbitrarily specified, grammar production. `NMTOKENS` is a white space-separated list of these. These datatypes are rather useless unless the desired datatype constraint happens to match the definition of `NMTOKEN`.

3.1.3 Other Problems with DTDs

It was noted above that the document supplies its own DTD. This is one of the key problems. It means that DTD-based validation cannot be used for determining if a document belongs to a class of documents (the "type" of the document) that the recipient expects to receive. That is, if two parties have agreed to exchange documents in a particular format, the recipient cannot use normal DTD-based validation to find out whether a given document is in the pre-agreed format, because the sender can use another DTD in the document. DTD-based validation only shows whether a document conforms to the grammar that the document *declares for itself*. Moreover, since DTDs require that the schema is included in the document itself, DTDs effectively require the document to be contaminated with schema-specific syntax.

There are implementations that allow validation against an application-supplied DTD, but this is not a standard part of XML 1.0 processing. Other schema formalisms (RELAX NG in particular) provide superior features in such a scenario.

Lastly, DTDs do not work properly with namespaces, because the *Namespaces in XML* specification [XMLNS] layers the namespace processing on top of XML 1.0 processing and, therefore, DTD-based validation takes place underneath the namespace layer – not on top of it.

DTDs were not used in this project, because DTDs are not expressive enough, are not namespace-aware and would allow smuggling of grammar productions by the document that is being checked.

3.2 W3C XML Schema

The W3C XML Schema [XSD] (sometimes abbreviated WXS but more often XSD for XML Schema Definition) is a schema language defined by the W3C in response to shortcomings of DTDs. XSD is a grammar-based schema language. In [Taxonomy], XSD is classified as "single-type". In a single-type language, competing productions within a content model and competing start symbols are prohibited. The practical consequence is that the grammar is required to be unambiguous. (A grammar is ambiguous if there is more than one derivation for a given document tree.)

XSD provides a richer repertoire of datatypes than DTDs. In fact, data typing is seen as one of the major improvements over DTDs. XSD validation very much involves infoset augmentation: instead of merely checking whether a document satisfies the schema, the XSD validation process yields a Post-Schema Validation Infoset (PSVI) which is the infoset of the validated document augmented with datatype information.

The concept of PSVI stems from the data-orientedness – as opposed to document-orientedness – of XSD. XSD is biased towards use cases that involve serializing objects or database items as XML on one hand and, on the other hand, databinding which involves doing the reverse. Despite the datatype focus of XSD, the datatype system is not extensible. The schema author has to get by with the datatypes that the specification provides. For this reason it is not uncommon to see schemata that do not constrain the data type of a given attribute even when the attribute has a very specific format.

XSD is rather verbose and, therefore, inconvenient to write. After a compact syntax for RELAX NG (page 22) was developed and found useful, a compact syntax for XSD was developed by Kilian Stillhard [CompactXSD]. However, this format is not standardized and appears not to have gained wide acceptance.

The W3C held a workshop on XML Schema 1.0 User Experiences and Interoperability [SchemaUE] where XSD users and vendors of implementations reported on their experiences. There had been problems with the complexity of the specification and the resulting implementation inconsistencies. The reports about the problems faced with XSD were summarized by Rick Jelliffe on the xml-dev mailing list in [Freddy].

The problems with XSD are discussed in more detail in [IntroXML].

XSD was not used in this project due to its reputation of implementation problems, its verbosity, its data-orientedness and relative lack of acceptance in document-oriented tasks.

3.3 Document Structure Description

Document Structure Description (DSD) [DSD] is a schema language developed at the University of Aarhus and AT&T Labs Research. In [Taxonomy], DSD 1.0 [DSD1] is classified as "single-type" and DSD 2.0 [DSD2] is classified to be able to represent any regular tree grammar.

DSD was not used in this project due to the wider acceptance of RELAX NG and Schematron and the better availability of tools for RELAX NG and Schematron. DSD has largely been sidelined in the marketplace by XSD, RELAX NG (discussed below) and Schematron (page 23).

3.4 TREX, RELAX, XDuce and DDML

There have been various also-ran schema languages in the quest for a replacement for DTDs. Perhaps the most famous ones are TREX by James Clark [TREX] and RELAX by Makoto Murata [RELAX]. These languages were the basis of RELAX NG (discussed below), which has superseded them. Document Definition Markup Language (DDML) was published as a W3C Note [DDML] but was abandoned in favor of XSD. XDuce [XDuce] has not gained wide acceptance.

3.5 RELAX NG

RELAX NG [RNG] is a grammar-based schema language for XML. In [Taxonomy], it is classified to be of the most powerful kind of tree grammars: "regular". It was developed from the basis of TREX and RELAX in OASIS (Organization for the Advancement of Structured Information Standards) with James Clark and Makoto Murata (the developers or TREX and RELAX respectively) as the editors of the specification. RELAX NG has also been subsequently standardized as Part 2 of the Document Schema Definition Language (DSDL) family [ISO19757-2].

James Clark describes RELAX NG as an evolution and generalization of XML DTDs based on experience from both SGML and XML. Design patterns used for writing DTDs can be applied to RELAX NG. Moreover, DTDs can be programmatically converted into RELAX NG. [RNGdesign]

RELAX NG treats elements and attributes in a uniform way to the maximum extent possible. This means that co-occurrence constraints between attributes and the content model of an element are possible.

RELAX NG is strictly for validation. No infoset augmentation is performed. Since there is no need to ambiguously assign datatypes to information items based on the derivation in the grammar, ambiguous grammars are allowed. A document is valid according to a RELAX NG schema if there is at least one derivation for the document in the grammar expressed by the schema. It does not matter if there are multiple derivations. Allowing ambiguous grammars makes RELAX NG schemata easier to write than DTDs or XSD.

3.5.1 Datatyping

RELAX NG has only two built-in datatypes: `string` and `token`. A schema may enumerate permissible datatypes and typed literals. The `string` type uses strict code point for code point comparison when comparing the literal and a value from the document being validated. The `token` datatype normalizes white space before the comparison. When a literal is not given, both types allow all strings that are legal in XML. Additionally, there is a `list` pattern that allows datatypes and literals to be used in white space-separated lists.

In addition to the built-in datatypes, RELAX NG has a framework for pluggable datatype libraries. A datatype library makes it possible to use a Turing-complete programming language for checking whether a string conforms to a particular datatype. In formal terms, a datatype (with given parameters) is a formal language and an equivalence relation for strings of the language. Each possible string of XML characters either belongs in the language of the datatype or does not belong in the language. For example, a datatype for dates could accept strings that represent valid Gregorian dates in the W3C-DTF notation [W3C-DTF] and reject all other strings.

A datatype also defines an equivalence relation for valid values. At minimum, each string needs to be equivalent with an identical string (reflexivity). However, the equivalence relation may be more lax as long as it is transitive and symmetric. For example, a datatype might accept all possible strings and define the equivalence relation as case-insensitive comparison.

It is important to note that the RELAX NG notion of datatypes only concerns classifying strings. Unlike the non-extensible XSD type system and the PSVI concept, RELAX NG datatyping is not about converting the strings to datatypes of a programming language (integers, floats, date objects, etc.). Special-purpose tools built on top of RELAX NG could use datatyping for databinding (if they restrict grammar ambiguity), but such usage is not sanctioned by the RELAX NG specification.

The RELAX NG defines syntax for using datatype libraries within a schema. However, the RELAX NG specification does not specify an API for interfacing a datatype library implementation with a RELAX NG validator, because such an API needs to be specific to the programming language used for implementation and RELAX NG does not require any particular programming language. However, for Java there is a de facto standard datatype library API developed by James Clark and Kohsuke Kawaguchi [DatatypeAPI]. The Java API has also been adapted to other languages – C# in particular.

There are two well-known datatype libraries: the XSD datatype library [RNG-XSD] and the DTD compatibility datatype library [DTDCompat]. The former brings the datatypes from [XSDDatatypes] to RELAX NG. The latter brings the datatyping features of DTDs to RELAX NG. RELAX NG validators often have built-in support for these two datatype libraries.

3.5.2 Compact Syntax

RELAX NG is defined as an XML vocabulary. However, since the XML syntax is designed for marking up text, it is not particularly convenient to write or even read in cases where there's almost no text and a lot of markup.

To address this problem, RELAX NG has an alternative Compact Syntax [Compact]. The compact syntax is vastly more human-friendly than the XML syntax and is intuitive to anyone familiar with the customary notation for regular expressions and the Backus–Naur Form. A tutorial of the Compact Syntax is given in [RNC-tutorial]. As with RELAX NG proper, the Compact Syntax was specified by OASIS and has subsequently been adopted as an amendment to the ISO standard [ISO19757-2Amd1].

3.5.3 Use in This Project

I chose RELAX NG as the main schema language for this project because of its status as the schema language of choice for document-oriented tasks and because a schema project [HTML5RNG] and a validator project [ValidatorAbout] were already in place. Elika Etemad had already chosen the Compact Syntax for the schema project. This was a good choice because of the human-friendliness of the Compact Syntax.

3.6 Schematron

Schematron [Schematron15] is an assertion-based schema language. In [Taxonomy], it is classified as a language for expressing identity constraints. Schematron was developed by Rick Jelliffe at the Academia Sinica Computing Centre (ASCC). A newer version of Schematron has been standardized as Part 3 of the Document Schema Definition Language (DSDL) family [ISO19757-3]. The ISO version of Schematron is incompatible with processors for the ASCC versions of Schematron.

A Schematron schema consists of assertions. In practice, an assertion is an XPath [XPath] expression and its context expression. The XPath expression is tested for evaluation to either true or a non-empty node set. This condition can either be considered required (false or an empty node set constitute an error) or an error (true or a non-empty node set constitute an error) depending on what the assertion is designed to test.

This is significantly different from grammar-based schema languages. In order for a document to conform to grammar-based schema, there has to be a derivation for the document in the grammar. Therefore, in a grammar-based schema, by default, everything is forbidden and only the constructs that can be derived from the grammar are allowed. In Schematron, however, everything is allowed by default and each assertion makes a specific restriction.

3.6.1 Using RELAX NG and Schematron Together

Adding specific restrictions without having to take a stance on the document as a whole makes Schematron ideal for refining a cruder schema written in another language. Rick Jelliffe, the creator of Schematron, has characterized Schematron as "a feather duster for the furthest corners of a room where the vacuum cleaner cannot reach" [SchematronOld]. Indeed, in the last three years, a pattern of using RELAX NG and Schematron together has emerged: a slightly over-permissive RELAX NG grammar is used for the bulk of the schema and Schematron assertions are used to tighten corner cases. For example, the schemata for the Atom syndication format [RFC4287] and DocBook 5.0 [DocBook] are RELAX NG schemata that are refined with Schematron assertions.

The RELAX NG schema and the Schematron schema can be separate or combined. If they are separate, the document is simply validated against both the

RELAX NG schema and the Schematron schema separately and considered valid if it passes both validations.

In the combined case, the Schematron assertions are written inside the RELAX NG schema. The validation phases are still separate, but the Schematron assertions can be organized so that they appear in the relevant element context in the RELAX NG schema for a human reader. An implementation may use the RELAX NG schema context of an assertion to establish the XPath context in Schematron. [Relaxtron]

3.6.2 Use in This Project

Following the example of Atom and DocBook, I chose Schematron as the secondary schema language in this project for expressing details that are inconvenient or impossible to express in RELAX NG.

I chose Schematron 1.5 [Schematron15] instead of ISO Schematron [ISO19757-3] due to lack of tool support for the ISO version – in particular lack of support in the Jing engine (page 37) [Jing].

Chapter 4

Prior Work on Markup Checking

The service presented in this thesis is not by any means the first markup checking service on the Web. In this chapter, notable other markup checking services are reviewed.

4.1 The W3C Markup Validation Service

The W3C Markup Validation Service [W3Cvalidator] (better known as the W3C Validator), originally written by Gerald Oskoboiny, is probably the best known of the markup checking services reviewed here. For many users, it is *the* validator. It has been in use since the late 1990s.

The W3C Validator is a Perl CGI front end for OpenSP. OpenSP [OpenSP] is an SGML parser based on James Clark's SP [SP]. OpenSP performs DTD-based validation according to SGML. That is, the input document is validated against the DTD that the document declares for itself. The front end allows the user to override the DTD declared by the document, in which case the front end modifies the document accordingly before passing it to OpenSP. The HTML 4.01 and XHTML 1.0 specifications come with normative DTDs. Typically documents include one of the normative DTDs by reference, but a document can include any DTD.

The W3C Validator sticks strictly to the SGML validity formalism. It is often argued that it would be inappropriate for a program to be called a "validator" unless it checks exactly for validity in the SGML sense of the word – nothing more, nothing less. Markup language specifications virtually always contain conformance requirements that cannot be expressed in an SGML DTD. Those requirements are simply not checked for at all. For example, in HTML 4.01 Strict, the value of the `datetime` attribute is required to be a date in the W3C datetime format [W3C-DTF], but since SGML DTDs cannot express this constraint, any string passes as a valid value for `datetime`.

Another problem is related to XML support. XML 1.0 was designed to be compatible with SGML in the sense that an XML document that is valid according to its DTD when treated as XML is also a valid SGML document when the Annex K of the SGML standard [ISO8879TC2] is in effect. The opposite is not always true, however: A document can be valid for the purposes of SGML without even being well-formed from the XML point of view. As a trivial example, SGML does not require white space between attributes but XML does. As a result, tools designed for SGML are not suitable for checking XML correctness. When giving results for XML documents, the W3C Validator states briefly “Note: The Validator XML support has some limitations.” [W3Cvalidator]

The SGML validation process requires a different SGML declaration for XML than for HTML 4.01. The choice of SGML declaration is external to the document. However, to avoid asking the user the rather esoteric question of which SGML declaration to use, the W3C Validator uses heuristics to decide which SGML declaration to use.

I did not choose the code of the W3C Validator as a starting point for the software discussed is this thesis, because the W3C Validator is a DTD-based validator for SGML documents written in Perl and C while the software discussed in this thesis started out as a RELAX NG-based validator for XML documents with key libraries written in Java.

4.2 WDG HTML Validator

The Web Design Group HTML Validator [WDG], developed by Liam Quinn, is very similar to the W3C Validator. It too has been in use since the late 1990s, has James Clark’s SP [SP] inside and has a Perl front end.

Originally, the WDG HTML Validator was differentiated from the W3C Validator by better error messages, by support for hexadecimal character references and by support for more character encodings than just ISO-8859-1 [WDG1998]. The W3C Validator has later added all these features as well. Currently, the WDG Validator is differentiated from the W3C Validator by warning about certain SGML markup minimization features that do not work in real-world browsers, by warning about character references to C1 control characters (characters U+0080…U+009F) and by using a different SGML declaration with custom DTDs [WDG2007]. In addition, the WDG HTML Validator can spider a site and validate multiple pages on one invocation.

Like the W3C Validator, the WDG HTML Validator is restricted to the SGML validity formalism.

I did not choose the code of the WDG Validator as a starting point for the software discussed in this thesis due to the same technical reasons that apply to the code of the W3C Validator.

4.3 Page Valet

Page Valet is a DTD-based validator developed by Nick Kew. Page Valet uses OpenSP for validating HTML as SGML. However, unlike the W3C and WDG validators, Page Valet uses a real XML parser – Xerces-C – by default for validating XML. (The option to use OpenSP for XML is offered.) [Valet]

Page Valet has an experimental option for turning on XSD-based validation in Xerces-C [XercesC]. However, there is no user interface for providing a schema separately from the document. That is, it is up to the document to specify its own schema.

For SGML-based validation, Page Valet provides three parse modes: Strict, Web and Fussy. The Strict Mode does what the W3C Validator does. It adheres strictly to the de jure formalism even when the results are impractical considering browsers. The Web Mode is described to be similar to what the WDG HTML Validator does. That is, SGML markup minimization features that do not work in real browsers are flagged. The Fussy Mode is described to "add further checks over and above Web Mode". [ValetMode]

Unlike the W3C and WDG validators, Page Valet is not a Perl CGI program. It is implemented as a C-language Apache module called mod_validator [mod_validator]. I did not use mod_validator as a starting point for the software discussed in this thesis, because I considered a managed runtime to be a safer choice than C for Web-facing services and I had more experience with Java. Moreover, using RELAX NG within a C program would have required using libxml2 [libxml2] instead of Xerces-C in practice.

4.4 The Schneegans XML Schema Validator

The XML Schema Validator (formerly XHTML Schema Validator) by Christoph Schneegans validates XML documents against XSD schemata. It does not validate HTML documents. [Schneegans]

While the three DTD-based validators discussed above use any DTD that the document declares for itself, the XML Schema Validator has a closed list of built-in XSD schemata. The user can choose a schema from a list manually or request the validator to choose a built-in schema based on the `xsi:schemaLocation` or the doctype. The schemata offered include the three variants of XHTML 1.0, XHTML 1.1, XHTML 1.0 Basic, MathML 2.0, XSD itself and Google Sitemaps.

The schemata for the variants of XHTML as well as XSD itself come from the W3C. The schemata for XHTML 1.0 were published as a W3C Note [XHTML10XSD]. The schemata for modularized XHTML were published as a Proposed Recommendation [M12N11] (later changed back to Working Draft [M12N11WD]).

The XML Schema Validator is written in Visual Basic .NET and is based on the XML tool chain provided as part of the Microsoft .NET Framework 2.0. The source code is not available.

4.5 Relaxed

Relaxed [RelaxedValidator] by Petr Nálevka validates documents against RELAX NG and Schematron schemata. The main advantage of Relax compared to the W3C Validator is the ability to check for requirements that cannot be expressed in DTDs. Relaxed was originally presented in Czech in Nálevka's bachelor's thesis [Validace], which I have been unable to read beyond the English abstract. Relaxed has subsequently been described in English in a paper co-written by Nálevka and his thesis instructor Jirka Kosek [Relaxed].

Relaxed builds upon James Clark's *Modularization of XHTML in RELAX NG* [M12N-RNG]. The schemata developed by Clark have been further refined and augmented with Schematron assertions. In addition to Schematron assertions based on the conformance requirements of XHTML 1.0, Relaxed offers optional Schematron-based checks for some of the *Web Content Accessibility Guidelines 1.0* [WCAG] requirements. In later updates after the initial release, schemata for compound documents that embed SVG [SVG] and MathML [MathML] in XHTML have been added.

Relaxed is written in Java and JSP. It uses the Sun Multi-Schema Validator (MSV) by Kohsuke Kawaguchi [MSV] as its validation engine. Even though there is a plug-in for MSV that enables support for Schematron assertions that are embedded in RELAX NG, Relaxed uses a separate XSLT-based [XSLT] solution. The Schematron part is first extracted from RELAX NG using XSLT. Then the resulting Schematron schema is compiled into an XSLT script using another XSLT transformation. Finally, the resulting XSLT script is run against the input document.

Relaxed offers a list of preset schemata. Custom schemata are not allowed. Preset schemata are provided for XHTML 1.0, optionally with SVG and MathML. The schemata for XHTML 1.0 can also be used for HTML 4.01. For XHTML 1.0 and HTML 4.01 without SVG or MathML, partial WCAG checks are available. A separate user interface is provided for a prerelease version of DocBook 5.0 [RelaxedDocBook].

The HTML support in Relaxed is based on the idea that HTML 4.01 can be mapped to XHTML 1.0 before applying XML validation technologies. Relaxed uses a patched version of John Cowan's TagSoup [TagSoup] for the conversion.

Of the validation services reviewed in this chapter, Relaxed is the closest to the system described in this thesis. The reason why I did not use the Relaxed code base as the starting point for this project is that I had already developed a comparable online validation service user interface around the Jing RELAX NG engine [Jing], which I found preferable over MSV (page 37), by the time Relaxed was announced in July 2005 [RelaxedAnn]. (I had already deployed mine in the spring of 2005.) Moreover, I prefer using a SAX pipeline as the output generation method (page 35) instead of JSP, because it is easier to guarantee the correctness of the output when a SAX pipeline is used.

Even though Relaxed and the service described in this thesis do not share Java code, I have used the schemata from the Relaxed project for the pre-HTML5 functionality of my validation service.

4.6 The Feed Validator

The Feed Validator [FeedValidator] by Sam Ruby, Mark Pilgrim, Joseph Walton, and Phil Ringnalda is notably different from the services reviewed above. Whereas the other services focus on HTML and/or XHTML, the Feed Validator focuses on Atom and RSS feeds, which are rather different from HTML and XHTML as markup languages. Also, the methodology used in the Feed Validator is significantly different.

The Feed Validator does not use any schema formalism. Instead, it uses handcrafted SAX handlers written in Python [Dive]. The main SAX `ContentHandler` (page 31) maintains a stack of element-specific delegates. Each element-specific delegate (inheriting from a common base) can check the conformance requirements pertaining to its element.

The approach taken in the Feed Validator has two benefits over schema-based validation. First, the Feed Validator is not limited by the capabilities of any schema formalism. Since Python is a full programming language, any requirement that is machine checkable in principle is checkable by a Python program. Second, since the emission of error messages is programmed by humans on a case-by-case basis, the messages can be as good and informative as the human developers care to make them. In the case of grammar-based schema languages in particular, error messages are generated by the validation engine in which case context-sensitive advice is generally not provided except by perhaps applying guesswork outside the validation engine.

It should be noted that feeds are different from (X)HTML in the sense that there is a greater focus on string values adhering to certain formats and the nesting structures of elements are less complex than in (X)HTML. The Feed Validator does not validate (X)HTML content embedded in feeds.

I did not choose the Feed Validator methodology for this project, because it was assumed that using a domain-specific non-programming language – RELAX NG – would be more manageable in terms of effort and malleability during the development process of HTML5. However, in the hybrid approach that I did choose, the non-schema-based checkers (page 49) are similar to the `ContentHandler` delegates used by the Feed Validator.

4.7 Validome

Validome by Thomas Mell, Vadim Konovalov, Alex Leporda, Olivier Duffez, Eduard Schlein and Dirk Klar checks both (X)HTML and feeds. Additionally, Validome offers generic XML validation against a DTD or an XSD schema declared by the document itself. Validome also offers to check DTDs and XSD schemata for syntax errors. [Validome]

In contrast to the other services that are English-only or also provide messages in French, Validome supports German and English as the user interface languages. The (X)HTML facet of Validome also offers French and Russian as user interface languages.

Validome uses SGML DTDs for HTML and XSD for XHTML. Additionally, Validome performs non-schema checks, which makes it (in at least some areas) more comprehensive than the other (X)HTML validators discussed above. [ValidomeStaff]

It is difficult to review what exactly Validome does, because its inner workings are not publicly documented and the source code is not available. Still, it is worth emphasizing that Validome goes beyond schema formalisms when specifications have conformance requirements that cannot be expressed as schemata.

Chapter 5

Implementation

This chapter starts the second part of this thesis which focuses on the software implemented as the experimental part of the thesis project.

5.1 The SAX API

The implementation uses extensively the Simple API for XML (SAX) version 2 [SAX]. Therefore, a brief introduction to the API is in order. SAX is discussed at length in [SAX2].

SAX is a community-created parser API for XML parsers written in Java. It has later been adopted as an official part of the standard Java class library. The API has also been adapted to other programming languages.

SAX is a streaming API. That is, data is reported to the application as the parse progresses as opposed to the parser handing a full document tree to the application after the parse. SAX is a push API. This means that the parser owns the main processing loop and reports data to the application through callbacks instead of the application requesting data from the parser.

The callbacks are organized in various handler interfaces that application classes implement in order to receive parse events (i.e. callback calls). It is necessary to discuss only the main interface, `ContentHandler`, here. `ContentHandler` has the following methods that the parser calls:

`startDocument()`
: Reports the start of the document.

`endDocument()`
: Reports the end of the document.

`startPrefixMapping(String prefix, String uri)`
: Reports the start of a namespace prefix mapping as per [XMLNS]. When namespace processing is enabled, `xmlns` attributes show up in SAX as prefix mapping and do not show up as attributes.

`endPrefixMapping(String prefix)`
: Reports the end of a namespace prefix mapping.

`startElement(String uri, String localName, String qName, Attributes atts)`
: Reports an element start. In XML, an element start always corresponds to a start tag or an empty-element tag. In HTML, element starts may be implied by other markup.

`endElement(String uri, String localName, String qName)`
: Reports an element end. In XML, an element end always corresponds to an end tag or an empty-element tag. In HTML, element ends may be implied.

`characters(char[] ch, int start, int length)`
: Reports text in element content. A single run of text may be split across multiple method calls. An efficient implementation allows the application to read data straight from the buffer the parser itself is reading.

`ignorableWhitespace(char[] ch, int start, int length)`
: Reports white space that is ignorable according to the DTD of the document (if a DTD is processed). For the purpose of this thesis, this method can be considered a special-case alias for `characters()`.

`skippedEntity(String name)`
: Reports that the parser skipped an entity reference, because the parser had not processed the external DTD and had not seen the possible definition of the entity.

`processingInstruction(String target, String data)`
: Reports a processing instruction. Processing instructions are of the form `<?target data?>`.

`setDocumentLocator(Locator locator)`
: Sets up source line and column reporting.

The `ContentHandler` interface does not expose comments, which an XML processor is not required to report to the application, nor syntactic sugar like CDATA section boundaries.

I chose GNU Ælfred2 [GNUJAXP] as the XML SAX parser for this project. Modifications to the parser were necessary for normalization checking (page 53) and for other character encoding-related checking (page 55). I also considered Xerces-J 2 but I deemed it more difficult to modify. For HTML, I developed a special-purpose SAX parser.

5.2 The HTML Parser

I implemented an experimental HTML parser to enable checking of `text/html` with XML tools. I developed the parser speculatively before the HTML5 parsing algorithm was published. The implementation sufficiently demonstrates the feasibility of the chosen approach for the purpose of this thesis project. Modifying the parser to implement the HTML5 parsing algorithm was deemed to be out of the scope of the thesis project and is discussed as possible future work.

5.2.1 HTML5 as an Alternative Infoset Serialization

The fundamental idea underlying the `text/html` support of the conformance checker is that HTML5 can be treated as an alternative infoset serialization for a subset of possible XML infosets [Infoset]. An HTML parser can appear to the XML tooling as an XML parser parsing XHTML.

HTML5 and XHTML5 are not defined in terms of the *XML Information Set* specification [Infoset] but in terms of the DOM [DOM2], which is a tree API for both HTML and XML. Regardless of definitional details, HTML5 and XHTML5 are considered alternative serializations that parse into a single kind of document tree. Consider the following two documents:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>Foo</p>
  </body>
</html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>Foo</p>
  </body>
</html>
```

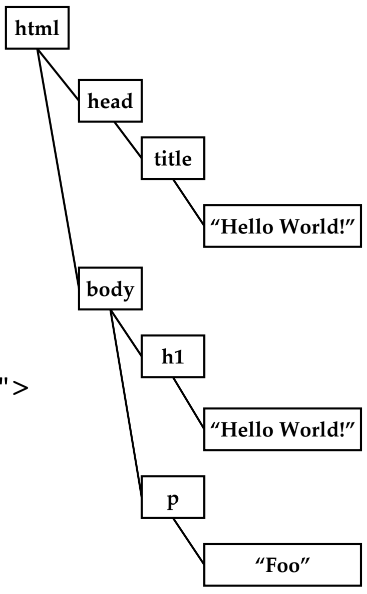

Figure 1: Document tree

The first one is an HTML5 document. The second one is an equivalent XHTML5 document. In order to make XML tools applicable to HTML5, the HTML5 parser needs to emit SAX parse events as if it were an XML parser parsing an equivalent XHTML5 document. When the HTML parser developed in this project parses the first document, it reports the parse events that an XML parser would report when parsing the second document.

5.2.2 TagSoup

The approach of making the HTML parser appear to be an XML parser was inspired by John Cowan's TagSoup [TagSoup]. I deemed TagSoup itself unsuitable for this project, however. TagSoup is designed to perform fix-ups on horribly malformed markup and to never report a parse error of any kind. A parser must report low-level errors and preserve high-level errors to be suitable for conformance checking. Preserving high-level errors means that the kind of errors that the RELAX NG layer, the Schematron layer and the non-schema checkers are designed to find are not hidden. Tokenization errors are low-level errors that need to be reported on the parser level.

After inspecting the TagSoup code base, I deemed it easier and less conflicting with the goal of the TagSoup project to develop a parser specifically for HTML conformance checking than to modify TagSoup. (However, Petr Nálevka later chose the route of modifying TagSoup for Relaxed.) I assessed the applicability of parser generators briefly, but it was apparent that both the nature of HTML and the requirements of the SAX API favored a hand-crafted parser.

5.2.3 Parser Design

The parser consists of a tokenizer that emits SAX events and SAX filters that add `endElement` events for empty elements and perform tag inference for optional tags. The tokenizer only emits `startElement` and `endElement` events for start tags and end tags, respectively. As a result, the SAX events emitted by the tokenizer may violate the SAX API contract. An empty element filter adds an `endElement` event for every `startElement` event belonging to an element that does not have an end tag in HTML. A tag inference filter adds `startElement` and `endElement` events where HTML 4.01 Strict would allow tags to be omitted. In most cases, the optional tags are end tags. Start tags that imply the end tag can be enumerated in a simple lookup table. For implicit start tags, case-by-case checks are used.

Since I developed the parser before the HTML5 parsing algorithm was specified, it did not make sense to implement error recovery. Instead, the parser treats most tokenization errors as fatal. Also, by design, the parser is not concerned with nesting errors that are known to be caught on the RELAX NG level even though conventional HTML parsers have to deal with fixing certain nesting errors.

5.2.4 Minor Problems

There are some minor problems related to mapping HTML5 to XHTML5.

- XHTML5 does not allow the character encoding to be declared using the `meta` element. Therefore, reporting an HTML5 document faithfully to XML tools as XHTML5 and converting an HTML5 document into XHTML5 are subtly different cases. In the former case one would want the `meta` element that declares the character encoding to be reported, but in the latter case the element would be omitted and the character encoding would be declared in the XML declaration instead. Likewise, in the latter case the `href` attribute on the `base` element would need to be converted to an `xml:base` attribute on the root element.
- XML places rather arbitrary restrictions on the characters that are permitted in element and attribute names. Fortunately, these restrictions are not a problem for conforming HTML5 documents. Unfortunately, XML also restricts the

characters permitted in content more strictly than HTML5 does. For example, HTML5 allows the form feed character. In the current implementation, the XML restrictions apply. This issue needs to be addressed if HTML5 continues to allow control characters that XML does not. The solution would be replacing the forbidden characters with permitted characters that have a similar syntactic role (e.g. replacing the form feed with a space).

- Finally, the permitted contents of comments differ in a subtle way between XML and HTML5. This is not a problem because the parser does not report comments to the validation layer.

5.3 Front End

I reused the front end that I had developed before the start of this thesis project as the front end for a service that allowed XML document to be validated against RELAX NG schemata.

The front end is a rather straight-forward Java servlet that handles form input, runs the back end code and produces output using a SAX pipeline. The front end code emits SAX events as if an XML parser were parsing an XHTML5 document corresponding to the output. The output markup is produced by a an HTML5 serializer. The serializer is an unparser that takes SAX events corresponding to an XHTML5 document and that writes HTML5 markup based on the events. For example, the call `endElement("http://www.w3.org/1999/xhtml", "a", "a");` causes the serializer to print `</a>`. Hence, the approach taken with output is the reverse of the approach taken with HTML5 input. Messages from the back end are written straight away to the output SAX pipeline, so the view and the controller are effectively conflated. There is no need for a separate data model, as the front end is only a thin wrapper for the back end code.

I chose this method of producing markup because isolating the markup text generation in a single serializer class makes it easier to get markup generation right, compared to more traditional methods where snippets of markup text are generated ad hoc in multiple places in

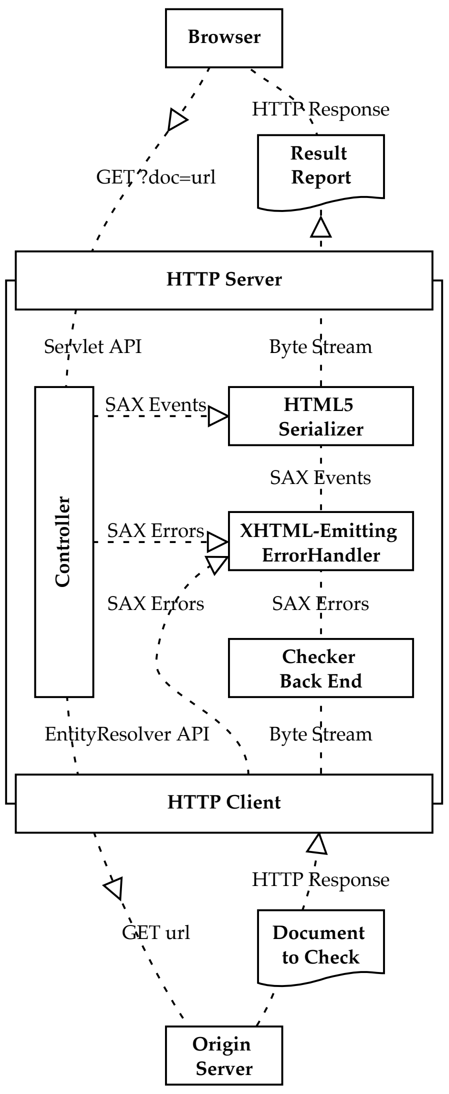

Figure 2: Overall data flow

the program. In particular, using an isolated serializer eliminates a whole class of markup generation problems: forgetting to escape markup-significant characters or accidentally escaping them twice. [ProducingXML]

All parts of the conformance checker report to a SAX `ErrorHandler`. The `ErrorHandler` implementation emits SAX events for the error messages formatted as XHTML list items. There is no intermediate data model. The XHTML list items are streamed out as the back end progresses on processing the input document.

Expressing an XML document by writing Java method calls is not as convenient as writing tags. To avoid hand-coding large unchanging parts of the user interface markup as Java method calls, had I developed a code generator called SaxCompiler [SaxCompiler] when I developed the front end for the RELAX NG validation service. SaxCompiler is a SAX `ContentHandler` that produces Java source for a class that will call the `ContentHandler` methods in the same sequence as the methods of SaxCompiler itself were called. The resulting class plays back SAX method calls recorded from a real XML parse. SaxCompiler supports the insertion of method calls back to the application that uses the generated classes and supports recoding XML fragments.

5.4 Back End Design

The basic architecture of the back end of the conformance checker is very simple: A parser consumes the document byte stream and emits SAX parse events. An arbitrary number of SAX handlers can listen to the events. The SAX handlers do not form a pipeline with one consumer emitting events to the next. Instead, the handlers are placed side-by-side and each SAX event is repeated to every handler by a chain of splitters.

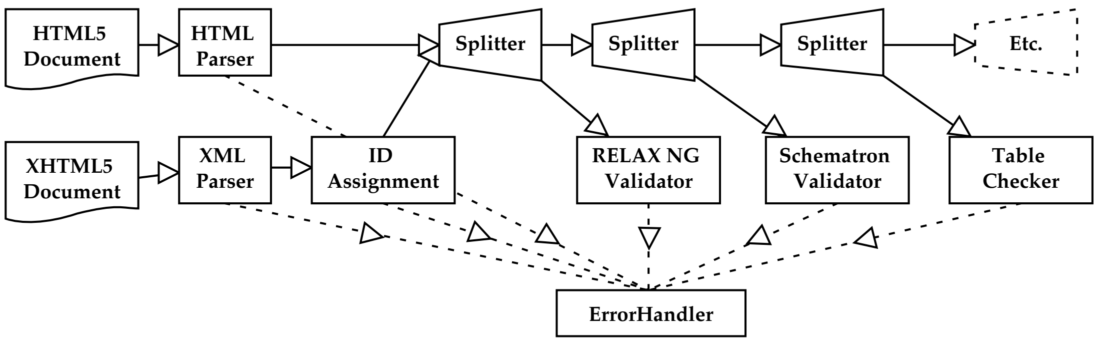

Figure 3: Data flow inside the back end

The handlers can be schema-based validators or custom code. The consumers receive SAX parse events that affect the meaning of the document. The choices of syntactic sugar are not exposed. For example, comments are not reported and it is not exposed if and how characters were escaped in the source.

The meaningful SAX events are the ones reported to the SAX `ContentHandler` and `DTDHandler` interfaces. However, in the events reported to `DTDHandler` (notifications of notations and unparsed entity declarations) are of no interest for an (X)HTML5 conformance checker, so in practice the SAX handlers only listen to the `ContentHandler` events. Moreover, the processing is genuinely namespace-aware and, hence, qualified names are not used by event consumers. Instead, the consumers observe namespace URIs and local names.

`LexicalHandler` is not listened to, because it exposes syntactic details that do not affect the meaning of the document and it would be inappropriate to tie conformance on the higher layer to the choice of syntactic sugar on the lower layer. `DeclHandler` is not listened to, because it exposes declarations that are intended to be expanded by the XML processor and, thus, should not be managed by the application.

The main SAX event handler is an instance of the Jing validation engine that has been instantiated with a RELAX NG schema for HTML5 or XHTML5.

5.5 The Jing Validation Engine

There are two well-known Open Source RELAX NG validation engines for Java: Jing by James Clark [Jing] and the Sun Multi-Schema XML Validator (MSV) by Kohsuke Kawaguchi.

I chose Jing as the RELAX NG and Schematron validation engine. Before starting this thesis project, I tried using MSV [MSV] in place of Jing [Jing] as the engine of my generic validation service that allows user-supplied schemata. However, I found that MSV was easier to crash with stack overflow with user-supplied schemata. Also, Jing supports the RELAX NG Compact Syntax but MSV does not.

The rationale behind my choice was based on the requirements of the generic validation service. In retrospect, the rationale is not strictly valid for the HTML5 conformance checker that has a particular fixed schema. With MSV, the stack size of the Java runtime could be configured to be sufficient for the HTML5 schema. Also, the schema could be converted from the Compact Syntax (page 22) to the XML syntax of RELAX NG as part of the build process of the software. Therefore, considering HTML5 conformance checking only, it would have made more sense to compare validation engines on their other merits, such as the quality of diagnostic messages. MSV was chosen for the Relaxed project due to quality of its error messages [Kosek].

Jing implements Clark's derivative algorithm [Derivative] for RELAX NG validation. For Schematron, Jing provides a wrapper for an XSLT engine. Jing supports SAXON and Xalan. I used the SAXON XSLT engine by Michael Kay [SAXON] for this project, because Clark chose to bundle SAXON with Jing.

5.6 The RELAX NG Schema

The bulk of what is allowed in HTML5 is encoded in a RELAX NG schema. The schema customizability framework and the core of the schema were developed by Elika Etemad [HTML5RNG]. I extended the schema with definitions for Web Forms (both 1.0 and 2.0) and with definitions for elements introduced in HTML5. I also integrated the schema with a custom datatype library.

For the most part, the element nesting conformance requirements for HTML5 are defined in terms of simple parent-child relationships. That is, it is defined that a given element may have children of a given type. These sorts of requirements map trivially to a grammar.

DTDs allow only one grammar production per element name. HTML5, however, requires different grammar productions in different contexts. For example, the attributes allowed for list items depend on the kind of list the items are in. Fortunately, RELAX NG decouples grammar productions from element names. There can be an arbitrary number of grammar productions for a given element name. This makes it possible for elements to have different attributes or content models depending on where the elements occur in the document. For example, the `form` element has an empty content model when it occurs as a child of the `head` element whereas it can have child elements when it occurs as a descendant of the `body` element.

In HTML5, some elements are defined to take either block-level children or inline-level children but not both at the same time. Such content models are called bimorphic. In HTML 4.01, those elements generally allowed a mix of block and inline content, because DTDs cannot express the kind restriction demanded by HTML5. Fortunately, RELAX NG can deal with bimorphic content models. When a RELAX NG validator sees an element, the element can have multiple pending derivations in the grammar. Therefore, adjacent subtrees can influence each other in validation.

The vast majority of the conformance criteria related to the document structure can be expressed as a RELAX NG grammar. In the implementation, I have favored the use of RELAX NG whenever practical. Still, many conformance criteria are not conveniently expressible in a RELAX NG grammar. Schematron and custom Java code are used for those criteria.

Expressing constraints related to ancestry turned out to be impractical in RELAX NG, even though it is theoretically possible. Balancing the needs of the main conformance checking use case and the expected reuse of the RELAX NG schema, for e.g. guiding auto-completion in an editor (page 63), turned out to be a problem. From the conformance checking point of view, it makes sense to handle more things in Schematron, because special cases of constraints on element ancestry are trivially expressed in Schematron with precise error messages whereas "remembering" ancestry in a grammar generally requires duplicating grammar productions. This would lead to unmanageable growth of the number of grammar productions. Eventually, I favored Schematron at the expense of the applicability of the schema to RELAX NG-only use cases.

5.6.1 The General Schema Design

The schema is divided into modules. The module division is motivated both by grouping similar elements together and by making it possible to easily subset the schema in ways deemed reasonable by the schema authors. The schemata for different subsets include the modules that are enabled for the particular subset.

5.6.2 Common Definitions

A module called `common.rnc`, developed by Elika Etemad, includes definitions for the schema framework. It contains switches for parameterizing the schema, initial definitions for common content models, definitions for common attributes and definitions for common datatypes. It also designates the grammar start symbol, i.e. the root element.

Common Content Models. The `common.rnc` module initializes the definition for classes of elements as follows:

```
common.elem.strict-inline =
  ( notAllowed )

common.elem.struct-inline =
  ( notAllowed )

common.elem.block =
  ( notAllowed )

common.elem.embedded =
  ( notAllowed )
```

Other modules then add to these definitions. For example, the following makes the `p` element allowed where block elements are allowed.

```
common.elem.block |= p.elem
```

The |= connector redefines the named pattern on the left-hand side to be the choice between right-hand side and the previous definition of the left-hand side.

The `common.rnc` module also defines common content models using the definitions for common element classes:

```
common.inner.strict-inline =
  ( text & common.elem.strict-inline* )

common.inner.struct-inline =
  ( text & common.elem.struct-inline* )

common.inner.block =
  ( common.elem.block* )

common.inner.bimorphic =
  (  (  text
     &  (  common.elem.struct-inline
        |  common.del.block
        )*
     )
  |  (  common.elem.block
     |  common.del.struct-inline
     )*
  )
```

The bimorphic case is more complex, because the structured inline alternative has to allow deleted block content and vice versa. The definitions for common content models are not augmented directly by other modules.

Common Attributes. HTML5 defines a handful of attributes that apply to all HTML elements. A pattern called `common.attrs` is defined in `common.rnc`. The pattern serves as a reusable pattern for element definitions and provides an extension point for modules that add attributes which apply to all elements. For example, including the module for scripting adds event handler attributes to all elements.

Common Datatypes. Many attributes in HTML5 take values that are required to conform to a specific format. In RELAX NG, such formats are known as datatypes. As discussed earlier (page 21), datatypes in RELAX NG only constrain the space of allowable string values and do not imply infoset augmentation or data binding. Except for datatypes related to HTML forms and enumerated string values, the datatypes used to constrain attribute values are defined in `common.rnc`.

When possible, the W3C XML Schema Datatypes [RNG-XSD] are used in order to make the schema more easily portable to different RELAX NG validators. In particular, the regular expression facet of the W3C XML Schema Datatypes is used. For example, percentage values are defined as follows:

```
common.data.percent =
  xsd:string {
    pattern = "(100)|([1-9]?[0-9](\.[0-9]+)?)%"
  }
```

`[0-9]` is used in the example above because in XSD regular expression the `\d` shorthand matches any character classified as Nd (decimal digit) in Unicode.

When the permissible lexical space of a datatype does not form a regular language or when the lexical space would in theory form a regular language but writing it down as a regular expression would be impractical, datatypes from the custom-built HTML5 Datatype Library (page 43) are used. For example, datatypes involving dates and IRIs are handled using the custom-built library (`w` is bound to the HTML5 Datatype Library).

```
common.data.datetime =
  w:datetime-tz

common.data.uri =
  string "" | w:iri-ref
```

It is important to note that using a custom datatype library makes the schema less portable. To use the schema the RELAX NG validator needs to have an implementation of the datatype library available to it. An alternative less precise version of the schema could be made if portability was appreciated over correctness.

Parameter Switches. RELAX NG has two special patterns that can be used to implement Boolean feature switches. These patterns are `empty` and `notAllowed`. The `empty` pattern takes no attributes or element content to satisfy. The `notAllowed` is never satisfied. Hence, interleaving `empty` with another pattern is equivalent to the other pattern alone and interleaving `notAllowed` with another pattern makes the interleave unsatisfiable as a whole. Thus, a named pattern can be used in a schema and the effect of the pattern can be changed by changing the definition of the named pattern from `empty` to `notAllowed` as needed. RELAX NG makes this easy by allowing a schema file that includes another to override definitions in the file that is being included.

For example, the `p` element has a more versatile content model in XHTML5 than in HTML5. In XHTML5, structured inline children – that is, inline mixed with select primarily block elements such as `ul` – are allowed. (The HTML5 serialization cannot allow elements that were block elements in HTML 4.01 as children of `p` due to backwards compatibility considerations.)

This distinction is handled using a switch called `nonHTMLizable`. It is defined in `common.rnc` as `nonHTMLizable = empty` which is what is needed for XHTML5. To flip the switch for HTML5, `common.rnc` is included as follows:

```
include "common.rnc" {
  nonHTMLizable = notAllowed
}
```

The switch pattern is then used like this:

```
p.inner =
  (  common.inner.strict-inline
  |  (  common.inner.struct-inline
     &  nonHTMLizable
```

```
      )
  )
```

When `nonHTMLizable` expands to `notAllowed`, the right-hand size of the | connector becomes unsatisfiable causing the `p.inner` production to become equivalent to `common.inner.strict-inline`.

When `nonHTMLizable` expands to `empty`, `p.inner` becomes equivalent to `common.inner.struct-inline` because `common.inner.strict-inline` is defined to be a subpattern of `common.inner.struct-inline`.

5.6.3 Examples of Elements

The RELAX NG implementation for a typical element looks like this:

```
blockquote.elem =
  element blockquote { blockquote.inner & blockquote.attrs }
blockquote.attrs =
  ( common.attrs
  & blockquote.attrs.cite?
  )
  blockquote.attrs.cite =
    attribute cite {
      common.data.uri
    }
blockquote.inner =
  ( common.inner.block )
```

The element is given a named definition: `blockquote.elem`. The definition simply expands to an `element` pattern for the element in question. The content model is defined to be an interleaving of two named patterns: one for element content (`blockquote.inner`) and another for attributes (`blockquote.attrs`).

The named pattern for attributes is defined as the interleaving of common attributes and element-specific attributes. In this case, the only element-specific attribute (`cite`) is optional, hence the `?` quantifier. Next, the named pattern (`blockquote.attrs.cite`) for the element-specific attribute is defined. The datatype for the attribute refers to a common datatype definition.

The named pattern for the element content (`blockquote.inner`) is merely defined to map to the common content model for elements that accept only block children (`common.inner.block`).

Other elements are defined analogously. Of course, the definitions for other elements tend to become more complex. For example, this is the definition for `datetime` form controls:

```
input.datetime.elem =
  element input { input.datetime.attrs }
input.datetime.attrs =
  ( common.attrs
  & common-form.attrs
  & input.datetime.attrs.type
  & common-form.attrs.accesskey?
  & input.attrs.autocomplete?
  & common-form.attrs.autofocus?
  & input.attrs.list?
  & input.datetime.attrs.min?
  & input.datetime.attrs.max?
  & input.attrs.step.float?
  & common-form.attrs.readonly?
  & input.attrs.required?
  & input.datetime.attrs.value?
  )
  input.datetime.attrs.type =
    attribute type {
      string "datetime"
    }
  input.datetime.attrs.min =
    attribute min {
      form.data.datetime
    }
  input.datetime.attrs.max =
    attribute max {
      form.data.datetime
    }
  input.datetime.attrs.value =
    attribute value {
      form.data.datetime
    }

input.elem |= input.datetime.elem
```

The notable detail in this more complex example is that the reference to `input.datetime.attrs.type` is not quantified. Hence, the `type` attribute with the value "`datetime`" is required. This definition for the `input` element is combined with the other definitions with different `type` attributes using the choice connector (`input.elem |= input.datetime.elem`). This means that the value of the `type` attribute serves as a discriminator for determining which patterns apply even though the name of the element remains the same.

5.7 The HTML5 Datatype Library

I discovered that a custom datatype is needed in the following cases:

- When the lexical space of a datatype is not a regular language or when it is but formulating it as a regular expression would be particularly hard or inconvenient.
- When the lexical space of a datatype is a regular language, but the grammar would require a lot of literal strings (e.g. registered language codes or character encoding names) embedded into it.
- When checking the value requires calendar calculations.

I developed a custom datatype library to address these cases. I also wrote a draft specification for the abstract library [HTML5Datatypes], so that the library could be implemented in other programming languages without having to inspect the Java source to see what it does.

5.7.1 Dates

On the surface, it would seem that the last case is unnecessary considering that the XSD datatypes [XSDDatatypes] include a `dateTime` type that can be constrained through its regular expression facet. However, the XSD `dateTime` is inappropriate for HTML5 in a subtle way. With the XSD type, leading and trailing white space is discarded before the pattern is matched, so there is no way of forbidding surrounding white space. The Web Forms 2.0 date and time types do not allow surrounding white space. Hence, custom types are needed.

Discarding surrounding white space as part of the data type makes the data type library less flexible. Any data type that does not allow white space at all (neither surrounding the meaningful value nor inside the value) can be used in a way that allows surrounding white space by wrapping it in the RELAX NG list pattern so that a single-token `list` of white space-separated tokens results.

5.7.2 IRIs

In addition to dates, it turns out that custom types for IRIs and language tags are called for, even though the repertoire of XSD types includes data types for these.

The XSD `anyURI` datatype is not useful. Its definition has changed with each edition and version of the specification. In the first edition of version 1.0 [XSDDatatypesFE], which predated the IETF IRI specification [RFC3987], the definition implied that not all strings are valid `anyURI` values. Indeed, the Jing [Jing] implementation does not allow all possible strings. In the second edition of version 1.0 [XSDDatatypes], which was finalized after the Jing implementation had been released, the definition suggested that different implementation levels could treat different lexical spaces valid. In a February 2006 working draft of the 1.1 version [XSDDatatypes11WD], the definition conceded that any finite string is a valid `anyURI`.

For these reasons, I developed custom datatypes for IRIs (only absolute) and IRI references (either relative or absolute). The major problem with deciding whether a string is a conforming IRI is that the generic IRI syntax is not restrictive. Instead, individual IRI schemes, such as `http`, `mailto` and `ftp`, are allowed to specify their own syntax. An implementation working only on the generic IRI level would pass just about any string as a valid IRI reference. On the other hand, it would be impossible to implement scheme-specific knowledge of future or private IRI schemes. Even implementing support for all the IANA-registered IRI schemes would be impractical. In fact, these problems were the reason (in addition to the XSD datatypes predating the IRI specification) for the fuzzy definition of `anyURI`. Still, it would be a pity not to help authors with scheme-specific issues related to the most commonly used schemes – the `http` scheme in particular.

The obvious solution is to implement scheme-specific checking for the most common schemes and apply only generic processing to the rest. However, leaving the decision on what scheme-specific checking to support to implementations of the abstract datatype library specification would make implementations uninteroperable. As a compromise, I wanted to include scheme-specific knowledge of the schemes that are specifically covered by the Web Forms 2.0, but not all of them were covered within this thesis project. I used the Jena IRI library [Jena] to provide scheme-aware support for the `http` [RFC2616] `https` [RFC2818], `ftp` [RFC1738] and `file` [RFC1738] schemes. However, the Jena IRI library does not have scheme-specific knowledge about the `mailto` [RFC2368], `data` [RFC2397] and `javascript` [javascriptURI] schemes. Support for the `javascript` scheme would require using the Rhino JavaScript library [Rhino] for syntax checking instead of using the Jena IRI library. I did not implement support for the `mailto`, `data` and `javascript` schemes within the scope of this thesis project. I left support for them as possible future work (page 65).

5.7.3 Language Tags

The HTML `lang` attribute and the XML `xml:lang` attribute take a language tag identifying a human language as their value. Until recently, language tags were defined by [RFC3066]. Now that specification has been made obsolete by [RFC4646] and [RFC4647]. A language tag consists hyphen-separated subtags.

Different subtags have different lengths. Moreover, the permissible list of values is restricted to private use subtags (starting with `x-`) and values that are in the Internet Assigned Numbers Authority (IANA) subtag registry [LangTagRegistry]. The XSD definition for `language` is significantly more coarse: the lexical space is defined to be 1–8 ASCII letters followed by zero or more hyphen-separated subtags consisting of 1–8 ASCII letters and/or digits [XSDDatatypes].

A custom datatype can do much better if it implements checking against the syntax defined in [RFC4646] and also contains data from the IANA registry. I chose and partially implemented this approach. However, the implementation was not finished within the scope of this thesis project.

Making the lexical space of a datatype dependent on a registry that changes over time poses a versioning problem. However, as long as implementations document when their IANA registry snapshot was taken, introducing new language tags will not be a significant problem. Introducing new values to the registry will expand the valid lexical space without making any previously conforming documents non-conforming. This is in contrast with the IRI scheme issue. Introducing support for a previously unsupported IRI scheme would narrow the valid lexical space.

5.7.4 ECMAScript Regular Expressions

Web Forms 2.0 introduces an attribute called `pattern` for form controls that accept textual input. The attribute specifies a regular expression that the value of the form control must match. Web Forms 2.0 does not define its own regular expressions. Instead, ECMAScript regular expressions [ECMA262] are used. When `^(?:` is prepended to the value of the `pattern` attribute and `)$` is appended to it, the resulting string is required to be a conforming ECMAScript regular expression. The datatype library handles the checking of ECMAScript regular expression syntax by wrapping the regular expression parser of Rhino [Rhino].

5.8 The Schematron Schema

A Schematron schema was used where RELAX NG did not work conveniently and where using custom Java code was not strictly necessary. Surprisingly, the use cases for Schematron turned out to be narrower than originally expected.

5.8.1 Exclusions

While grammars are very good at enforcing parent-child relationships, they are not good at enforcing ancestor-descendant relationships, because in the absence of intersections and negations, some of the productions would have to "remember" what kind of ancestors of interest there have been. Suppose element A is not allowed to have element B as a descendant. In this case, all the elements that can occur on the ancestry path from B to A would need to have two grammar productions: one that can be derived from the root without an intervening A element and one that cannot. Moreover, if there were more such exclusions, the number of parallel grammar productions per neutral element would double for each exclusion rule. Obviously, this could be a serious maintainability problem.

Fortunately, exclusions are extremely easy to express in Schematron. The forbidden descendant element is used as the context node for an assertion, which then states that the context node must not have as an ancestor an element that forbids the context element as its descendant.

For example, this Schematron pattern causes `blockquote` elements that are descendants of `header` elements to be reported as errors.

```
<rule context="h:blockquote">
  <report test="ancestor::h:header">
    The blockquote element cannot appear as a
    descendant of the header element.
  </report>
</rule>
```

The Schematron wrapping of the XPath expressions is rather verbose, but the expressions themselves are simple. The expression `h:blockquote` matches the `blockquote` element in the XHTML namespace. (The prefix `h` is bound to the XHTML namespace.) The rest of the rule is only applied if the context expressions matches, and in that case, the `blockquote` element is used as the XPath context for the second expression `ancestor::h:header`. This expression matches `header` elements (in the XHTML namespace) that are also ancestors of the context node. If the set of matching nodes is non-empty, the natural-language error message is reported.

5.8.2 Required Ancestors

Opposite to exclusions, there are also checks for required ancestors. Specifically, the Web Forms 2.0 repetition model requires form inputs for moving and removing repetition blocks to have a repetition block or a repetition template as an ancestor.

The checks are of the following form:

```
<rule context='h:input[@type=move-down]'>
  <assert test='ancestor::h:*[@repeat]'>
    An input element of type="move-down"
    must have a repetition block or a
    repetition template as an ancestor.
  </assert>
</rule>
```

The context matches `input` elements in the XHTML namespace that have an attribute named `type` that has the value `move-down`. The assertion test matches ancestors of the context node that are in the XHTML namespace and have an attribute named `repeat`. This rule is correct for pure (X)HTML5 documents but is not sufficient for compound documents. Support for compound documents (mixing XHTML with e.g. SVG and MathML with the Web Forms 2.0 repetition model) was left outside the scope of this master's thesis project.

5.8.3 Referential Integrity

It turns out that RELAX NG is not good for enforcing referential integrity. Enforcing referential integrity means checking cases where an attribute value is required to be a reference to the ID of another element. RELAX NG has an optional extension called *RELAX NG DTD Compatibility* [DTDCompat]. This extension makes it possible to check that values designated as ID references actually refer to IDs in the same document. However, that is all it can check. Moreover, enabling this extension places restrictions on the ambiguity of the schema, which makes schemata harder to write in some cases. Due to the limitations of RELAX NG DTD Compatibility, I moved all ID-related checking away from RELAX NG. Most cases are handled in Schematron.

In Schematron, referential integrity checking builds on the XPath `id()` function. The argument of the function is coerced into a string that is split on white space. The return value is a node set containing the elements that have an ID equal to any of the tokens split from the argument.

The use of the `id()` function has two crucial differences compared to testing equality against the values of two attributes. First, the function operates on IDness and not on the names of attributes. Second, the argument is split on white space to produce a list of tokens, so the function works for the case where multiple IDs are referenced (IDREFS in DTD terms).

The concept of IDness is part of XML 1.0 itself. The IDness is established by processing a DTD that declares certain attributes to have the type ID. The DTD-based type of attributes is data that is exposed through the core SAX interface. More precisely, it is exposed via the `getType()` methods of the `Attributes` interface. The XPath implementation determines which attributes should be considered to be of type ID by calling one of the `getType()` methods.

In the case of the HTML5 serialization, there is no DTD processing whatsoever involved. The parser simply assigns IDness to the attribute named `id` and exposes this through SAX.

In the case of the XML serialization (XHTML5), IDness that is not based on DTD processing is assigned between the parser and the validation SAX handlers. Immediately after the parser in the pipeline, there is a SAX filter that constitutes an "xml:id processor" as per [xmlid]. Immediately after the xml:id processor, another SAX filter performs similar ID assignment on attributes named `id` that are not in a namespace and belong to an element that is in the XHTML namespace. This second filter could be called an "XHTML id processor".

With ID assignment performed before the Schematron stage, Schematron can be used to check that referents are of the right kind. For example:

```
<rule context='h:input[@list]'>
  <assert test='id(@list)/self::h:datalist or
               id(@list)/self::h:select'>
    The list attribute of the input element must
    refer to a datalist element or to a select element.
  </assert>
</rule>
```

5.9 The Non-Schema-Based Checkers

It turns out, as expected, that HTML5 has conformance requirements that cannot be expressed in RELAX NG or Schematron or that would be inconvenient to express in RELAX NG or Schematron. Checking for such requirements is handled by non-schema-based checkers.

Conceptually, a non-schema-based checker listens to parse events and does whatever is necessary and computable to identify the kind of conformance requirement violations that the checker is designed to handle. In practice, a non-schema-based checker is a Java class that implements the SAX `ContentHandler` interface and reports to a SAX `ErrorHandler`. Since non-schema-based checkers are implemented in a full programming language, they can check for any machine-checkable conformance requirement.

To make the non-schema-based checkers fit into the Jing-based architecture, I wrote a wrapper class that implements the Jing `Validator` interface. With this wrapper class, the non-schema-based checkers conform to the same interface as the schema-based validators and can be combined into the same validation pipeline. Every non-schema-based checker is also given a URI so that they can be instantiated by URI in the extended validation user interface just like validators instantiated from schemata identified by URIs.

The organization of checks for different requirements into different non-schema-based checkers is a matter of software design. After all, the specification does not mandate a particular code organization. If every requirement is implemented as a separate checker, there will be a lot of code duplication, since many checkers need to do similar things. On the other hand, implementing everything in a single checker would make the code unmaintainable. The organization below reflects my design decisions.

Developing checkers to cover all of HTML5 was deemed to be out of the scope of this thesis project. Instead, I implemented a selection of prototype checkers as a proof of concept.

5.9.1 Table Integrity Checker

A table integrity checker was chosen as the main proof of concept non-schema-based checker. I deemed table integrity the most complex of the requirements that call for a non-schema-based checker making it ideal for proving by implementation that the approach works. Also, purely schema-based validators are incapable of checking table integrity and table integrity has notable relevance to table rendering in browsers, so table integrity checking makes for a good demo. The source code of the table integrity checker is presented in the appendix (page 93).

HTML Tables. Since the table model for (X)HTML5 was only being specified when the checker was prototyped in November 2006, the checker was speculatively based on the HTML 4.01 table model [HTML401] and browser behavior. The differences from HTML 4.01 are that `colspan='0'` is treated as `colspan='1'` and that `headers` must refer to `th` cells as per [HixieTables]. The top left corner of cells is placed in the first available slot on the row, which is browser-compatible but different from what the CSS2 specification [CSS2] says.

An HTML table has a Cartesian grid of slots. A cell can span multiple slots. Subsequent cells are moved to the right until a free slot for the top left corner of the cell is found. When cells span multiple rows, this slot allocation policy can lead to overlapping cells.

Class Division. The table integrity checker consists of seven classes: `TableChecker`, `Table`, `RowGroup`, `Cell`, `ColumnRange`, `HorizontalCellComparator` and `VerticalCellComparator`. The `TableChecker` class is the non-schema-based checker class used by other code and the rest of the classes are internal to the checker.

`TableChecker` maintains a stack of `Table` instances. When a `startElement` event for the `table` element is seen, a new `Table` instance is pushed onto the stack. Likewise, the stack is popped upon seeing an `endElement` event for `table`. The rest of table-significant events (`startElement` and `endElement` for the `col`, `colgroup`, `thead`, `tbody`, `tfoot`, `tr`, `td` and `th` elements) are delegated to the `Table` object at the top of the stack.

`Table` maintains the state of where in the table markup (e.g. in table row inside an explicit row group like `tbody`) the parse currently is. The `Table` object only sees table-significant parse events. If an event occurs out of the permitted sequence (e.g. a cell start occurs when the state is not in a table row) the whole subtree of misplaced elements is silently ignored at that point by counting starts and ends. Reporting this as an error is left to the RELAX NG validator.

`Table` maintains a linked list `ColumnRanges`. A `ColumnRange` represents a contiguous range of columns that has been established e.g. by the `col` element but does not yet have any cells starting in the range. A `ColumnRange` knows its start and end column slot indexes. In contrast to e.g. an array of column slots, the memory usage of this data structure is proportional to the number of ranges rather than the size of the ranges. Therefore, the memory usage can be throttled down by limiting the number of elements that are processed, which can indirectly be throttled by limiting the size of documents (in bytes). A malicious content author cannot make the checker allocate excessive amounts of memory by declaring a column group that spans an immense number of column slots.

`Table` also has the responsibility of instantiating `RowGroups`. The `thead`, `tfoot` and `tbody` elements establish explicit row groups represented as `RowGroup` objects. In XHTML, `tr` elements may occur as children of the `table` element. This case is treated as an implicit row group also represented by `RowGroup`. (In HTML, the parser infers a wrapping `tbody`.) Since table cells are assigned to slots on a per-row group basis in (X)HTML, the `Table` only instantiates `Cell` objects and gives them to the current `RowGroup` for slot allocation.

Slot Allocation. A `Cell` knows the column index where it starts and where it ends horizontally. It also knows the row until which it spans vertically. `RowGroup` maintains a set of cells that span more than one row and that are still in effect – that is, the first row onto which they do not span has not passed yet. As with `ColumnRanges`, the data structure does not involve allocating a two-dimensional array of slots. Therefore, the memory requirements are bounded in proportion to the number of cell elements rather than the number of slots that cells are declared to span.

When a `Cell` spans more than one row, it is added to the set of `Cells` that are in effect. When a row ends, `Cells` that are no longer in effect are culled from the set. When a new row starts, the `Cells` that are in effect are sorted according to their start column index. A new `Cell` on the row get its top left corner assigned to the first free slot on the row. Slots occupied by `Cells` that are still in effect (i.e. span down from earlier rows) are skipped. When a `Cell` is allocated to particular slots, this is reported back to `Table`, so that the list of `ColumnRanges` without cells can be adjusted accordingly. Because cells cannot span across row groups, if the set of `Cells` in effect is not empty when the row group ends, an error is reported for each `Cell` still in the set. If there are `ColumnRanges` left in the linked list when the table ends, the ranges without cells starting in them are reported.

Detected Conditions. The checker emits both warnings and errors. Since *Web Applications 1.0* is still a draft, it is too early to tell which conditions should be treated as errors eventually. The rest will be just warnings.

The following conditions are detected:

- A table cell is overlapped by a later table cell.
- A table cell overlaps an earlier table cell. (A single overlap is reported in both directions to show source location for both cells.)
- A table cell spans past the end of its row group.
- A row has no cells starting on it.
- The column count on a table row is greater than the column count established by `cols/colgroups`.
- The column count on a table row is less than the column count established by `cols/colgroups`.
- The column count on a table row is greater than the column count established by the first row in the absence of `cols/colgroups`.
- The column count on a table row is less than the column count established by the first row in the absence of `cols/colgroups`.
- The `headers` attribute does not point to th elements in the same table. (This feature was based on speculative information and will likely have to be revisited as the specification matures.)
- A column range has no cells starting on it.

- The value of a `colspan` attribute exceeds 1000, which is a magic number in Gecko (and according to comments in Gecko source, also in IE and Opera) [TableCell].
- The value of a `rowspan` attribute exceeds 8190, which is a magic number in Gecko [TableCell].
- A `col` element causes a `span` attribute to be ignored on the parent `colgroup`. (Conforming in HTML 4 / XHTML 1.0; non-conforming in (X)HTML5. With (X)HTML5, there is also a schema-level error.)

5.9.2 Checking the Text Content of Specific Elements

The new `time`, `meter` and `progress` elements have a specific format for their text content. HTML5-aware user agents are required to parse the text content but the content serves as a fallback in legacy user agents. RELAX NG makes it possible to constrain the text content of an element if the element only has text content and no child elements. In HTML5, the elements with a specific text content format are allowed to have child elements. Hence, merely developing a custom datatype and using it from the RELAX NG schema would not work exactly the right way.

A Java-based RELAX NG-aware XML editor may make it possible to use custom datatype libraries but is unlikely to allow non-schema-based checkers. In such a case, text content checking may be valued higher than child elements. In anticipation of this use case, I implemented checkers for the particular text content formats as classes that implement the `Datatype` interface from the Java API for custom RELAX NG datatypes [DatatypeAPI]. The non-schema-based checker then uses these `Datatype` implementations to check the text content of particular elements.

The Java API for custom RELAX NG datatypes supports streaming reporting of text content to a `Datatype` implementation. The `Datatype` interface has a method called `createStreamingValidator` which returns an object that implements an interface called `DatatypeStreamingValidator`. Character data can be reported to a `DatatypeStreamingValidator` in chunks the same way as the SAX interface handles reporting of character data. The non-schema-based checker simply maintains a stack of active `DatatypeStreamingValidators` and reports the SAX character data to every `DatatypeStreamingValidator` on the stack.

The instantiation rules for the `DatatypeStreamingValidators` are hard-coded. The non-schema-based checker could in theory be generalized to accept parameters from the outside stating which elements require which `Datatype` to be instantiated. However, hard-coding these rules was simple and sufficient to meet the objectives of this project.

When a `DatatypeStreamingValidator` on the stack is about to be popped – that is, the end of the element is seen – the `DatatypeStreamingValidator` is queried whether it accepts the reported content or not. If it does not accept the reported content, the non-schema-based checker reports an error.

5.9.3 Checking for Significant Inline Content

HTML5 introduces a concept of significant inline content. Significant inline content consists of embedded content (such as an `img` element) or significant text. Significant text is defined to be text that contains any character that is not in the Unicode categories Zs, Zl, Zp, Cc and Cf.

HTML 4.01 stated: "We discourage authors from using empty `P` elements. User agents should ignore empty `P` elements." [HTML401] As a result, markup generators started to generate paragraphs containing a single NO-BREAK SPACE. In HTML5, a NO-BREAK SPACE does not count as significant text, since it is in the Unicode category Zs. I believe that this kind of conformance definition and workaround arms race is not productive. However, since the requirement was in the specification draft, I developed a checker. This a simple example of a non-schema-based checker. Having this checker also makes it possible to test the requirement of significant inline content with existing Web pages to see how realistic the requirement is.

The checker maintains a stack corresponding to open elements. The stack keeps track of which currently open elements have significant inline content. Character data is tested against an ICU4J `UnicodeSet` [ICU4J] that represents the characters in the above-mentioned Unicode categories. When characters or elements that constitute significant inline content are encountered the current stack nodes are marked as having significant inline content.

The checker for significant inline content demonstrates that non-schema-based checkers for particular requirements can be very simple and straightforward.

5.9.4 Unicode Normalization Checking

In order to develop a prototype checker for a potential requirement that needs checking not only in the SAX handlers but also in the parser, I prototyped a checker for Unicode normalization.

Character Model for the World Wide Web 1.0: Normalization [CharmodNorm] is still in the Working Draft state. Nonetheless, a checking for compliance was prototyped as if the normalization specification was a normative part of (X)HTML5. This way, the feasibility of the requirements of *Character Model for the World Wide Web 1.0: Normalization* could be evaluated.

Requirements. The definition for Fully-normalized Text involves checking normalization before and after parsing. That is, the source text is required to be in Unicode Normalization Form C [UAX15]. After parsing the "constructs" parsed out of the source are required to be in Unicode Normalization Form C and are required not to start with a "composing character" (which is not exactly the same as a "combining character" in Unicode).

In order to integrate normalization checking of the unparsed character stream into Ælfred2, special-case decoding for US-ASCII, ISO-8859-1, UTF-8, UTF-16 and

UTF-32 was removed and all character decoding was unified to use the java.nio.charset framework [NIO].

The definition involves peeking underneath the parser, which might be considered a violation of abstraction layers. However, the requirement of checking the unparsed source text does have the benefit that if the source is in Unicode Normalization Form C, the document cannot be accidentally broken by editing it in a normalizing text editor, as a text editor touches the unparsed text.

Interpretation. *Character Model for the World Wide Web 1.0: Normalization* does not define what "constructs" are in the context of XML 1.0 or HTML5. However, XML 1.1 [XML11] does define what "relevant constructs" are, so that the definition might be generalizable to XML 1.0 and HTML5. Unfortunately, XML 1.1 defines relevant constructs in terms of the grammar productions of XML itself instead of the significant information items that an XML processor reports to the application. As a result, the XML 1.1 definition is not particularly useful in practice.

Since XML 1.0 and HTML5 do not have a definition for "constructs", a definition that makes sense was devised for the purpose of prototyping. [C14N] and the SAX `ContentHandler` interface (page 31) were used as indicators of what the meaningful constructs in XML are once the differences in syntactic sugar are out of the way. This gave the following definition of constructs:

- Local names of elements
- Local names of attributes
- Attribute values
- Declared namespace prefixes
- Declared namespace URIs
- Processing instruction targets
- Processing instruction data
- Concatenations of consecutive character data between element boundaries and processing instructions ignoring comments and CDATA section boundaries.

Implementation. As with the checker for significant inline content, the implementation turns out to be rather simple. The checker outsources most work to ICU4J [ICU4J]. An ICU4J `UnicodeSet` is used for testing whether a character is a composing character. The ICU4J `Normalizer` class is used for testing Unicode normalization.

Constructs that are exposed as Java `String`s in the SAX API are very simple to check. The first character is checked against the above-mentioned `UnicodeSet` and the whole string is passed to the `Normalizer` for normalization checking.

Character data, however, is checked in a piecewise manner. Most complexity in the checker is due to trying to avoid buffering as much as possible while still using the ICU4J API unmodified. Also, dealing with the halves of a surrogate pair falling into different UTF-16 code unit buffers causes complexity by roughly doubling the lines of code compared to an implementation that was not supplementary plane-aware. (UTF-16 is the character encoding used for strings in Java.)

The checker tries to check as much character data as possible by passing runs of the SAX-provided buffer to ICU4J. However, normalization-sensitive data may continue over the buffer boundary, so the checker copies potentially normalization-sensitive data near the buffer boundaries to its internal buffer which it later passes to ICU4J. The ability to test for composing characters is used for finding safe points for slicing buffers. By definition, it is always safe to slice buffers so that in piecewise normalization checking each buffer slice being tested starts with a character that is not a composing character.

Unfortunately, the column and line numbers reported on errors are very inaccurate due to buffering. Due to the design of the SAX API, accurate column and line positions are unavailable within a particular buffer of character data.

The normalization checking of the source text is performed by making the parsers (both the HTML parser and the XML parser) instantiate the checker on the parser level. Both parsers decode the incoming byte stream into a UTF-16 character stream, which is then parsed. The buffers are passed to the checker as a side effect of the parser reading the character stream.

5.10 Character Model Checking

Web Forms 2.0 requires documents to conform to *Character Model for the World Wide Web 1.0: Fundamentals* [Charmod]. Therefore, checking for the conformance requirements that the character model places on content was investigated.

Since the recommendations of *Character Model for the World Wide Web 1.0: Fundamentals* deal with issues related to character encoding, the best opportunity for checking whether a document conforms to specification is in the parser. Hence, the checks were implemented in the parsers and not in the validation pipeline.

Some of the requirements were not machine-checkable and, thus, had to be omitted. For example, the requirement "Specifications, software and content MUST NOT require or depend on a one-to-one correspondence between characters and the sounds of a language." [Charmod] is not machine-checkable, because the checking software cannot tell what the author expects of correspondence to sounds and if the expectation is relied upon.

Many recommendations deal with the identification of the character encoding and the use of preferred IANA-registered names. These checks were implemented as part of the code that sets up the input stream decoding in the parsers.

In addition to the recommendations related to identifying the character encoding, there were only three other machine-checkable recommendations. However, all three are problematic in terms of reporting errors.

I deemed the recommendation "Publicly interchanged content SHOULD NOT use codepoints in the private use area." [Charmod] worth a warning (only once per document), because human judgment is needed in order to decide when the private use area is used in a legitimate way. For example, it is legitimate to use the private use are to encode scripts that have not yet been assigned official Unicode code points or that will not included in Unicode as a matter of policy (such as Tengwar or Klingon which have been constructed to support particular works of fiction). Moreover, another recommendation in [Charmod] denies forbidding the use of the private use area.

I ignored the two other recommendation as too impractical when considered in the light on real-world HTML authoring practice.

The first ignored recommendation reads: “Escapes SHOULD only be used when the characters to be expressed are not directly representable in the format or the character encoding of the document, or when the visual representation of the character is unclear.” [Charmod] The second one is: “Content SHOULD use the hexadecimal form of character escapes rather than the decimal form when there are both.” [Charmod]

Using the five predefined entities in XML, using the HTML5 entities from the specification or using numeric character references is harmless when it comes to the parsed document tree. In XML decimal and hexadecimal character references work equally well. In HTML, the decimal form actually works better in very old browsers. Enforcing these recommendations would mean proclaiming a prevalent authoring practice non-conforming on the grounds of the aesthetic preferences pertaining to source markup even when there is no difference in the post-parse document tree. Moreover, *Character Model for the World Wide Web 1.0: Fundamentals* does not give a solid machine-checkable definition for characters whose visual representation is unclear.

Chapter 6

Shortcomings

The developed service has some shortcomings. The serious shortcomings pertain to error messages. The less important shortcomings pertain to the software not being quite as efficient in terms of performance as it could theoretically be.

6.1 Non-Ideal Error Messages

The foremost shortcoming of the conformance checker is the lack of useful detail in the error messages emitted upon violations of the RELAX NG schema. This problem was anticipated before starting the project. Also, the users of development versions have identified the messages generated by the Jing validation engine as a notable problem.

Jing has a handful of different messages for RELAX NG validation failures. They are all very concise. For example: "Element *name* not allowed in this context.", "Attribute *name* not allowed at this point; ignored." and "Bad value for attribute *name*." The error messages are always correct, but they do not help the user understand why something went wrong.

6.1.1 Bimorphic Content Models

When the permissibility of an element depends on something more complex than the parent element, the error messages may confuse the user. For example, some elements take either inline or block children but not both. Moreover, in HTML 4.01 Transitional these elements were generally allowed to take a mix of inline and block children. Consider this fragment:

```
<div><em>foo</em><p>bar</p></div>
```

When the validation engine sees the `p` element, it has already committed to a derivation in the grammar that allows `em` as a child of `div`. That derivation is the inline branch of the bimorphic content model. Hence, the `p` element is not allowed in the derivation and the validation simply states that the element `p` is not allowed there.

This may be confusing, because `div` does allow `p` children in other situations. A better error message would state that `div` only takes either inline or block children.

One way of addressing this problem would be allowing a mix of inline and block children in the schema and using a non-schema-based checker for detecting and reporting the mixed use of inline and block content as children of elements that have a bimorphic content model.

6.1.2 Lack of Datatype Diagnostics

When the value of an attribute is not permissible according to the datatype of the attribute, the validation engine emits a message simply stating that the attribute had a "bad" value. No hint is given on why the value is bad.

The RELAX NG datatype API allows a datatype implementation to communicate diagnostic messages to the validation engine in exception messages. However, Jing does not expose these messages to the user. In theory, an ambiguous grammar could cause the validation engine to test a single attribute value against multiple datatypes, so there would not be a single diagnostic message. However, in practice with many schemata it would be quite helpful to provide the diagnostic message from at least one (usually the only) datatype that did not accept the value.

Even if diagnostic information were reported, there would be a further complication. In many cases the datatype is defined using a regular expression. If the regular expression does not match, there is no useful natural-language explanation available.

6.1.3 Erroneous Source Is Not Shown

The error messages do not show the erroneous markup. For this reason it is unnecessarily hard for the user to see where the problem is.

6.2 Poor Localizability

Although the conformance checker is carefully internationalized in the sense that it correctly handles input documents in any language and supports supplementary characters in addition to the Basic Multilingual Plane of Unicode, the messages that the conformance checker reports are available in English only and there is no mechanism in place for supporting other languages.

There are a number of problems related to the translatability of the messages emitted by the software. Various libraries are used, so the ways in which messages originate is not unified (except that all libraries emit English-language messages by default). Some libraries have no localization facilities whatsoever. Other libraries do have localization mechanisms but the mechanisms assume that the user interface language is a property of the entire Java process and discovered from the environment of the process. However, this assumption does not hold for Web applications. The locale of the server is uninteresting and instead Web applications should be able to vary the user interface language on a per-HTTP response basis. In addition to these problems, the Schematron schema directly contains messages that are exposed to the end user.

Instead of modifying the libraries themselves, an alternative approach to localization would be reverse templating. The English messages would be matched against known patterns that would allow the variable parts to be extracted. The variable parts could then be plugged into a translated message corresponding to the matched pattern.

In order to focus on HTML5 conformance checking instead of solving the translatability problems discussed above, translatability of the user interface was left out of the scope of this project.

6.3 Opportunities for Optimization

Some shortcomings relate to the implementation not being as efficient in terms of performance as theoretically possible. These shortcomings are not necessarily practical problems and the achievable improvements may not be worth the effort that would be required.

6.3.1 RELAX NG

The Jing RELAX NG engine took its current form in 2003. Back then, it was designed to be compatible with Java 1.1. Dropping support for Java 1.1 opens up opportunities for performance optimizations by replacing thread-safe classes with classes that perform the same tasks but do not use thread synchronization features.

When Java 1.0 and 1.1 were designed, all the classes in the standard library were made thread-safe as a matter of policy. In retrospect, this has turned out to be a bad policy. Often, a given object is only accessed from one thread, which makes synchronized monitor entry and exit useless. When an object is shared between the threads, it is likely that more than one standard class library object needs to be mutated in an atomic operation, which means that the application needs to manage the synchronization anyway. Even with modern virtual machine designs that can make monitors biased towards one thread so that access from that thread does not actually cause real inter-thread synchronization, class implementations that do not use synchronization perform better in scenarios where thread-safety is not necessary.

For compatibility with Java 1.1, Jing uses the `Hashtable` class instead of the `HashMap` class introduced in Java 1.2 as part of the Collection API [CollectionsAPI]. Profiling the conformance checker with the NetBeans profiler [NetBeansProfiler] shows that `Hashtable$Entry` is the second most common object (after `char[]`) in the memory allocation statistics in terms of number of live objects of a given type. While this statistic does not indicate how often the methods of `Hashtable` are called, it is still reasonable to expect the `Hashtable` class to be used a lot. Therefore, replacing occurrences of the `Hashtable` class with the API-compatible non-synchronized `HashMap` class would likely make RELAX NG validation slightly faster.

The two other typical candidates for replacement with non-synchronized counterparts are the `Vector` class and the `StringBuffer` class. They could be replaced with `ArrayList` (introduced in Java 1.2) and `StringBuilder` (introduced in Java 5), respectively. Even though the instances of these classes are not as common as instances of `Hashtable` and `Hashtable$Entry`, it would make sense to use the non-synchronized counterparts in these cases as well.

6.3.2 Schematron

The Schematron implementation in Jing is based on XSLT [XSLT]. This is natural considering that Schematron has been designed to be easily implementable as an XSLT transformation on the document being validated. Inside the transformer, a tree is built from the SAX parse events. The Schematron assertions fire when the entire document has been reported to the XSLT transformer.

Also, Jing creates a short-lived helper thread that sleeps when the main thread runs for fitting together API calls whose blocking behavior makes it impossible to use them from one thread. The helper thread pretends to call into an XML parser and then blocks itself and unblocks the main thread and allows the parse events from the main thread to be reported as if coming from the XML parser that the helper thread pretended to call.

This implementation approach has many points where it could be improved.

First, the helper thread can be eliminated by using an API that exposes the XSLT engine as a SAX `ContentHandler` instead of insisting on the XSLT engine initiating the parse. I prototyped this approach and, indeed, it was possible to eliminate the helper thread and the overhead associated with creating and destroying it. Cursory testing locally without network suggested that this improved the throughput (number of requests per unit of time) of the system by about 1%, but due to the variation between test runs the figure should be considered inaccurate.

Second, even though Schematron is designed to be implementable in XSLT, running an XSLT transformation on a full XSLT implementation is more complex than what would be minimally required to implement Schematron. The XSLT transformation spends time creating a report document which is then converted to calls to the SAX `ErrorHandler`. A Schematron implementation without XSLT could evaluate XPath expressions on a tree model and produce error messages as necessary without creating a report document.

Third, Schematron implemented by evaluating XPath expressions on a full document tree causes all the messages to appear after the entire document has been parsed and the tree built. In some cases, messages could logically be triggered much sooner. For example, in the case of exclusions as soon as an element is seen with a forbidden parent on the stack of open elements, an error message could be produced. XPath expressions are classified based on when they can be evaluated in [SchematronHeuristic]. In theory, a streaming XPath matcher could both produce error messages during the parse and consume less memory. However, implementing such a streaming matcher for this project in particular would have been (and still would be) too great a distraction from the main goals of the project. If a streaming Schematron implementation was available off-the-shelf, using it would be worthwhile.

Fourth, the project ended up using Schematron only for two simple purposes: exclusions and referential integrity checking. For just these two purposes, Schematron in general and XSLT-based Schematron in particular is unnecessarily heavy. An extremely simple hand-crafted non-schema-based checker could replace the Schematron part of the system. A rough estimate based on the benchmarking the throughput of the system with and without the Schematron part suggests that the throughput of the system could increase by about 5% if the Schematron part was replaced with a hand-crafted non-schema checker. Moreover, such a checker would make it extremely easy to emit errors related to exclusions as soon as logically possible.

In summary, the best way to optimize the performance of the Schematron part would be to treat it as a rapid prototype and replace it with hand-crafted code once the HTML5 language requirements have stabilized and there is less need for the conformance checker to be easily modifiable.

Chapter 7

Applicability in Other Contexts

Even though the focus of this thesis project was conformance checking, other applications for the schemata and Java code are briefly considered.

7.1 Auto-completion

Early on in the project I assumed that the RELAX NG schema would be directly usable in RELAX NG-aware XML editors for guiding the auto-completion. Auto-completion gives element or attribute name suggestions to the user based on the permissible names at a given point according to the schema and based on the start of the name already typed by the user. RELAX NG-aware editors include nxml-mode for Emacs [nxml-mode], oXygen XML [oXygen] and Etna [Etna]. Unfortunately, during the course of the project it became apparent that in some cases RELAX NG could in theory express a constraint but expressing it in Schematron or in custom code would be easier and would provide better error messages. Exclusions are the foremost class of constraints for which this is the case. Moreover, the use of a custom datatype library makes the schema less portable.

If RELAX NG were able to combine patterns with intersection and negation connectors, writing a schema for auto-completion would be easier. However, it would not solve the problem that it would be hard for a generic RELAX NG validator to generate better error messages for exclusions than what hand-crafted messages in a Schematron schema can provide trivially.

7.2 Content Management Systems

Many content management systems in use today do not properly check the input they accept. This leads to a situation commonly referred to as “garbage in garbage out”. The content management systems serve erroneous markup if erroneous markup has been entered into the system. The back end of the conformance checker implemented in this project could be used in content management systems to check input.

Content management systems written in Java could easily integrate the back end of the conformance checker. However, to support other programming languages the conformance checker would need to expose a remote interface that could be used from other programming languages with minimal client code. In practice, it would make sense to implement such an interface as a Web service following the REST architectural style.

Chapter 8

Future Work

Developing a full HTML5 conformance checker is too broad a task for a master's thesis project. For this reason, I developed the software to a point where the feasibility of implementation is demonstrated in all areas, but I did not push every area to completion.

8.1 Open Up

Even though the software developed in this project is Free Software / Open Source, it has not been developed in a way that would make it easily approachable for potential contributors. Perhaps the most pressing need for change in order to move the software forward after the completion of this thesis is moving the software to a public version control system and making building and deploying the software easy.

8.2 The HTML5 Parsing Algorithm

I implemented the HTML parser speculatively before the HTML5 parsing algorithm had been defined. The parser needs to be revised to implement the HTML5 parsing algorithm.

The parser is designed for the SAX API, which is a streaming API. The HTML5 parsing algorithm has error recovery features that require the parser to mutate parts of the parse tree that have already been built in earlier stages of the parse. This is incompatible with SAX. In order not to compromise streamability, the revised parser would have to treat errors that require SAX-incompatible recovery as fatal errors. This is allowed by the specification.

To make the parser reusable as a general-purpose HTML5 parser for other Java programs, it would be desirable to also implement the full HTML5 parsing algorithm including the SAX-incompatible parts. This would require a tree building layer as an alternative to the tag inference filter that does not build an in-memory tree. There could be an interface for pluggable tree builders. Tree builders could be provided for DOM [DOM2], XOM [XOM] and a special-purpose tree designed for efficient SAX event replay. A tree designed for SAX replay could store attributes as objects implementing the SAX `Attributes` interface and could store character data in `char` arrays as opposed to `Strings`.

A preliminary review of the HTML5 parsing algorithm indicates that the tokenizer would not need to be completely rewritten even though the tokenizer maintains its state implicitly in the runtime stack and the HTML5 parsing algorithm maintains state explicitly. At the first sight, the HTML5 parsing algorithm appears to allow state transitions that do not appear to correspond to normal returns on the runtime stack. However, on a closer inspection the abnormal state transitions are always abrupt returns to the main loop and could be implemented as an exception caught in the main loop.

8.3 Tracking the Specification

Since the conformance checker and HTML5 itself have been developed in parallel, the conformance checker has been almost constantly more or less out of sync with the specification. Ideally, future development should track the specification on a near-daily basis instead of major synchronization work every few months.

Moreover, during this project, various small issues with no clear answer in the specification were discovered. Once the issues are clarified in the specification, the software needs to be updated accordingly. This list of known issues at the time of the publication of this thesis is at [ValidatorAbout2007].

8.4 RELAX NG Message Improvements

The foremost problem with the RELAX NG-based approach to HTML5 conformance checking is that the error messages that a generic validator can realistically generate cannot be as good as messages written by a human being for specific situations.

The first potential improvement is to expose the diagnostic messages from datatype libraries to the end user. The second, also relatively simple, improvement would be to mention the parent element of the element that is deemed forbidden in a given context. With RELAX NG, the parent is not sufficient for explaining the situation, but with HTML5 it usually is when exclusion are handled in Schematron. No assessment was made to determine the feasibility of these changes, but the changes do seem simple.

If the validation engine reported the misplaced element and its parent, an HTML5-aware error message decorator could add natural-language descriptions about the content model of the parent and the allowed contexts of the child.

The third more complex change would involve giving a hint of what kind of elements would have been allowed in place of the element that was not allowed. Even if querying the schema in this way were possible to add to Jing, there would be the additional complication that RELAX NG-based expectations could still be wrong. For example, exclusions expressed in Schematron could forbid some elements that the RELAX NG schema would allow. For this reason, it may be better to focus on the first two improvement ideas.

Alternatively, the suitability of MSV [MSV] as the RELAX NG engine for HTML5 conformance checking should probably be reassessed with the attention on the quality of diagnostic messages, even though Jing appears to be more robust (page 37) is for use cases that involve arbitrary user-supplied schemata.

8.5 Completion of the Datatype Library

I did not complete the datatypes for IRIs and language tags within the scope of this thesis project. Support for the `data`, `mailto` and `javascript` IRI schemes needs to be added. Also, the implementation for the language tag datatype needs to be completed.

8.6 More Non-Schema-Based Checkers

The table integrity checker was the most complex non-schema-based checker that is needed. Therefore, there is not work of comparable complexity left to be implemented. However, in terms of quantity, there are many simple and small requirements scattered around [WebApps] and [WebForms2] that need to be addressed using non-schema-based checkers or alternatively, in some cases, Schematron. Making sure that all these requirements have been identified, writing test cases for the requirements and implementing checking for each requirement may well end up being more time-consuming than the development of the table integrity checker.

Identifying all these requirements calls for particular attention, as the requirements are scattered around the specifications and many times are only briefly mentioned in passing. The following were identified as unimplemented features requiring non-RELAX NG checking:

- Emit a warning if the `accept-charset` attribute is used on a `form` element that uses the XML submission format.
- Emit a warning if there is no selected radio button in a radio button group.
- Emit an error if there are more than one radio button selected in a radio button group.
- Emit an error if a form field name starts with `Ecom_` and the name is not listed in [RFC3106].
- Emit an error if a `form` attribute is non-empty but does not point to a `form` element by ID.
- Emit an error if a `label` element has more than one form control descendant or if a `label` element has the `for` attribute and a form control as a descendant.

- Emit an error if the form submission method is `get`, the form is designated to submit to an `http` URI, and the `accept-charset` attribute designates an encoding other than `US-ASCII` or `UTF-8` .
- Emit an error if there are non-unique term definitions using the `dfn` element.
- Emit an error if there is an `abbr` element that has neither a `title` attribute nor a corresponding `dfn` element.
- Emit an error if the attribute values on the `meter` element do not satisfy the expected inequalities.
- Emit an error if the `rect` coordinates on the `area` element do not satisfy the expected inequalities.
- Emit an error if the value of the `value` attribute on the `progress` element is greater than the value of the `max` attribute.
- Emit an error if IDs are not unique.
- Support checking for `rel` and `class` attribute values in a way that updates the registered permissible values from an external service dynamically.

Of the features listed above, the last one will likely take the most effort to implement.

Additionally, the functionality provided by the Schematron schema could be implemented as a non-schema-based checker that fires errors as soon as logically possible.

A relatively complex component similar to a non-schema-based checker is needed for showing the document outline. However, strictly speaking, such a component would not be checking machine-checkable conformance requirements and, therefore, is not included in the list above.

8.7 Assistance for Checking Human-Checkable Requirements

The conformance checker is unable to check for conformance requirements that require human judgment. However, the conformance checker could make it easier for human users to check such requirements.

For example, a machine cannot check if the document outline produced by the HTML5 outline algorithm makes sense. However, a machine could generate the outline and show it to the human user for review.

8.8 Web Service

To support the use of the conformance checker back end from other applications (non-Java applications in particular), a Web service would be useful. A Web service interface following the REST architectural style [REST] could be added relatively easily. Ideas about such an interface are presented in [WebServiceIdeas].

8.9 Embedded MathML and SVG

Browsers from the vendors involved with the WHATWG are adding (partial) support for SVG 1.1 or have already done so. The Gecko engine also supports MathML. To support the use of these features, support for SVG and MathML in XHTML5 host documents could be added to the conformance checker.

8.10 Showing the Erroneous Source Markup

The error messages give the line and column for errors, but the output does not show the actual erroneous part of the source markup. This makes it harder to see where the problems are.

Future development could include a class for collecting the character-decoded and line-identified source code. This would require changes to both the HTML and XML parsers. The parsers would need to report the unparsed source to a data holder class at the point where the byte stream has already been decoded to UTF-16 and the line boundaries have been identified.

The data holder class could be used for extracting markup snippets corresponding to each message. Also, the data could be used for formatting the entire source markup in the conformance checker output in a way that allowed linking to the points that contain errors.

Chapter 9

Conclusions

In this thesis the implementation of an HTML5 conformance checker was examined. A conformance checker was built around a RELAX NG validator.

9.1 Correct Expectations

My prior expectation related to the expressiveness of RELAX NG was that RELAX NG alone would not be sufficient but would have to be augmented with Schematron and hand-crafted custom code. However, I expected RELAX NG to be more convenient to write and change than Java code. Also, I expected that the convenience of using RELAX NG would have the cost of making error messages not as good as than what they could be if they were hand-crafted on a case-by-case basis.

My prior expectation related to the `text/html` serialization of HTML5 was that it could be treated as an alternative infoset serialization for a subset of possible XML infosets. Therefore, XML tools would be applicable if the parser for `text/html` exposed the result of the parse in a way an XML parser would expose the result of parsing an equivalent XML document.

I found, through implementation experience, that these prior expectations were correct.

9.2 Incorrect Expectations about RELAX NG

I expected that it would make sense to use RELAX NG for expressing virtually all HTML5 conformance requirements that could theoretically be expressed in RELAX NG. This expectation turned out to be incorrect. I found that especially in cases of exclusions implementation using Schematron (or custom Java code) was by far easier than expressing the exclusions in the RELAX NG grammar. Moreover, expressing exclusions in Schematron (or custom Java code) also made it possible to give more sensible error messages than what a RELAX NG validator would give.

For purposes of validation, when Schematron (or hand-crafted code) can be used alongside RELAX NG, RELAX NG should be used for expressing the general rules whereas Schematron should be used for expressing the exceptions to the general rules. Trying to include the exceptions to the general rules in the RELAX NG schema is bad both for schema maintainability and for the error messages that are generated. That is, the full formal expressiveness of RELAX NG cannot be put to use, because the generation of useful error messages cannot keep up.

I expected *RELAX NG DTD Compatibility* extension to be applicable for this project. However, I discovered that the problems caused by the extension outweighed its benefits. Its level of referential integrity checking is inadequate. The type of referent elements cannot be constrained. On the other hand, the restrictions *RELAX NG DTD Compatibility* places on the grammar in order to avoid type ambiguity turned out to be a significant annoyance. Schematron is much better suited for checking referential integrity.

The decisions not to use RELAX NG for exclusion checking and referential integrity checking has reusability implications for the RELAX NG schema. Using the RELAX NG schema alone in, for example, an editing system that only supports RELAX NG would mean losing some exclusion and referential integrity checking features which theoretically could be expressed in RELAX NG. Therefore, in such cases it may be worthwhile using code generation together with the schema from this project to produce a schema that incorporates the exclusions in RELAX NG.

9.3 Unexpected Discoveries about Schematron

I did not expect the Schematron part to end up to be limited mainly to exclusions and referential integrity checking. Thinking of the Schematron part only as a rapid prototype of certain kinds of non-schema-based checkers emerged relatively late in the project.

Because Jing did not support Schematron embedded in RELAX NG, I developed the Schematron schema as a file that is separate from the RELAX NG schema. I was surprised to find that I felt no need for embedding Schematron assertions into RELAX NG. On the contrary, it seems that mixing the two would have been bad for maintainability. I conclude that embedded Schematron is overrated.

9.4 Overall Assessment

Even though I overestimated the applicability of schema languages before the project, the overall hybrid implementation approach worked out very well. The integration of the HTML with the XML tools was a success. Therefore I consider the project to be a success.

References

[AXML]
The Annotated XML 1.0 Specification. Tim Bray, Jean Paoli and C. M. Sperberg-McQueen, editors. O'Reilly Media, Inc., 1998.
http://www.xml.com/pub/a/axml/axmlintro.html (referenced: 2007-03-04)

[C14N]
Canonical XML Version 1.0. John Boyer. W3C, 2001.
http://www.w3.org/TR/2001/REC-xml-c14n-20010315

[Cascading]
Cascading Style Sheets. Håkon Wium Lie. PhD thesis, University of Oslo, 2005.
http://people.opera.com/howcome/2006/phd/ (referenced: 2007-02-26)

[cdf-ws-minutes2]
W3C Workshop on Web Applications and Compound Documents (Day 2) Jun 2, 2004. Leigh Klotz, editor. W3C, 2004.
http://www.w3.org/2004/04/webapps-cdf-ws/minutes-20040602.html (referenced: 2006-10-18)

[Charmod]
Character Model for the World Wide Web 1.0: Fundamentals. Martin J. Dürst, François Yergeau, Richard Ishida, Misha Wolf and Tex Texin, editors. W3C, 2005.
http://www.w3.org/TR/2005/REC-charmod-20050215/

[CharmodNorm]
Character Model for the World Wide Web 1.0: Normalization, working draft. François Yergeau, Martin J. Dürst, Richard Ishida, Addison Phillips, Misha Wolf and Tex Texin, editors. W3C, 2005.
http://www.w3.org/TR/2005/WD-charmod-norm-20051027/

[CollectionsAPI]
Annotated Outline of Collections Framework. Sun Microsystems, Inc..
http://java.sun.com/j2se/1.4.2/docs/guide/collections/reference.html (referenced: 2007-03-23)

[Compact]
RELAX NG Compact Syntax. James Clark, editor. OASIS, 2002.
http://relaxng.org/compact-20021121.html (referenced: 2007-04-23)

[CompactXSD]
A Compact Syntax for XML Schema. Kilian Stillhard. Master's thesis, Swiss Federal Institute of Technology Zurich, 2003.
http://dret.net/netdret/docs/da-ws2002-stillhard.pdf (referenced: 2007-04-23)

[Computable]
On computable numbers, with an application to the Entscheidungsproblem. Alan M. Turing. In *Proceedings of the London Mathematical Society*, series 2, volume 42, pages 230–265. London Mathematical Society, 1937.
http://www.turingarchive.org/browse.php/B/12 (referenced: 2007-03-03)

[CSS1]
Cascading Style Sheets, level 1. Håkon Wium Lie and Bert Bos. W3C, 1996.
http://www.w3.org/TR/REC-CSS1-961217

[CSS2]
Cascading Style Sheets, level 2. Bert Bos, Håkon Wium Lie, Chris Lilley and Ian Jacobs, editors. W3C, 1998.
http://www.w3.org/TR/1998/REC-CSS2-19980512/

[DatatypeAPI]
RELAX NG Datatype Interface. James Clark and Kohsuke Kawaguchi.
http://relaxng.sourceforge.net/datatype/java/apiDocs/
(referenced: 2007-04-23)

[DDML]
Document Definition Markup Language (DDML) Specification, Version 1.0. Ronald Bourret, John Cowan, Ingo Macherius and Simon St. Laurent, editors. W3C, 1999.
http://www.w3.org/TR/1999/NOTE-ddml-19990119

[Derivative]
An algorithm for RELAX NG validation. James Clark. 2002.
http://www.thaiopensource.com/relaxng/derivative.html
(referenced: 2007-02-27)

[Dive]
Inside the RSS Validator. Mark Pilgrim. O'Reilly Media, Inc., 2003.
http://www.xml.com/pub/a/2003/02/26/dive-into-xml.html
(referenced: 2007-03-03)

[DocBook]
The DocBook Schema, working draft. Norman Walsh, editor. OASIS, 2007.
http://www.docbook.org/specs/docbook-5.0CR3-spec-wd-01.html
(referenced: 2007-05-03)

[DOM2]
Document Object Model (DOM) Level 2 Core Specification. Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion and Steve Byrne, editors. W3C, 2000.
http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/

[DSD]
DSD: A schema language for XML. Nils Klarlund, Anders Møller and Michael I. Schwartzbach. In *FMSP '00: Proceedings of the third workshop on Formal methods in software practice*, pages 101–111. ACM Press, 2000. ISBN: 1-58113-262-X.
http://doi.acm.org/10.1145/349360.351158

[DSD1]
Document Structure Description 1.0. Nils Klarlund, Anders Møller and Michael I. Schwartzbach. AT&T and BRICS, 1999.
http://www.brics.dk/DSD/dsddoc.html (referenced: 2007-03-03)

[DSD2]
Document Structure Description 2.0. Anders Møller. BRICS, 2005.
http://www.brics.dk/DSD/dsd2.html (referenced: 2007-03-03)

[DTDCompat]
RELAX NG DTD Compatibility. James Clark and Makoto Murata, editors. OASIS, 2001.
http://relaxng.org/compatibility-20011203.html (referenced: 2007-04-23)

[EarlyHistory]
The Early History of HTML. Sean B. Palmer.
http://infomesh.net/html/history/early/ (referenced: 2006-10-05)

[ECMA262]
ECMA-262 3rd ed., *ECMAScript Language Specification*. ECMA, 1999.
http://www.ecma-international.org/publications/files/ecma-st/
ECMA-262.pdf (referenced: 2007-03-04)

[Etna]
Etna, a wysiwyg XML RELAX NG- and Gecko-based editor. Daniel Glazman and Laurent Jouanneau. IDEAlliance Inc., 2006.
http://xtech06.usefulinc.com/schedule/paper/84 (referenced: 2007-03-04)
Proceedings of the XTech 2006 conference

[FeedValidator]
Feed Validator for Atom and RSS. Sam Ruby, Mark Pilgrim, Joseph Walton and Phil Ringnalda.
http://feedvalidator.org/ (referenced: 2007-03-04)

[Frames]
Why Frames Suck (Most of the Time). Jakob Nielsen. 1996.
http://www.useit.com/alertbox/9612.html (referenced: 2006-10-09)

[Freddy]
Can Blind Freddy see a pattern here?. Rick Jelliffe. 2005.
http://lists.xml.org/archives/xml-dev/200507/msg00057.html
(referenced: 2007-03-05)

[Generalized]
A Generalized Approach to Document Markup. Charles F. Goldfarb. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, pages 68–73. ACM Press, 1981. ISBN: 0-89791-050-8.
http://doi.acm.org/10.1145/800209.806456

[GNUJAXP]
The GNU JAXP Project. Free Software Foundation, Inc., 2006.
http://www.gnu.org/software/classpathx/jaxp/ (referenced: 2007-04-02)

[Handbook]
The SGML Handbook. Charles F. Goldfarb. Oxford University Press, 1991. ISBN: 0-19-853737-9.

[Harmful]
Sending XHTML as text/html Considered Harmful. Ian Hickson.
http://www.hixie.ch/advocacy/xhtml (referenced: 2006-10-14)
Originally written in 2002; revised in 2006.

[HixieTables]
[whatwg] Table integrity and conformance. Ian Hickson. 2006.
http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2006-October/007430.html (referenced: 2007-03-26)

[HTML30]
HyperText Markup Language Specification Version 3.0, working draft. Dave Raggett, editor. IETF, 1995.
http://www.w3.org/MarkUp/html3/html3.txt (referenced: 2007-04-23)

[HTML32]
HTML 3.2 Reference Specification. Dave Raggett. W3C, 1997.
http://www.w3.org/TR/REC-html32

[HTML40]
HTML 4.0 Specification. Dave Raggett, Arnaud Le Hors and Ian Jacobs, editors. W3C, 1997.
http://www.w3.org/TR/REC-html40-971218/

[HTML401]
HTML 4.01 Specification. Dave Raggett, Arnaud Le Hors and Ian Jacobs, editors. W3C, 1999.
http://www.w3.org/TR/1999/REC-html401-19991224/

[HTML40rev]
HTML 4.0 Specification. Dave Raggett, Arnaud Le Hors and Ian Jacobs, editors. W3C, 1998.
http://www.w3.org/TR/1998/REC-html40-19980424/
Revised without incrementing the version number.

[HTML5Datatypes]
RELAX NG Datatype Library for HTML5 Datatypes, working draft. Henri Sivonen. 2006.
http://hsivonen.iki.fi/html5-datatypes/2006-04-27 (referenced: 2007-03-04)

[HTML5RNG]
RELAX NG Schema for (X)HTML 5. Elika Etemad and Henri Sivonen. 2007.
http://syntax.whattf.org/ (referenced: 2007-02-27)

[HTMLplus]
HTML+ (Hypertext markup format), working draft. Dave Raggett. 1993.
http://www.w3.org/MarkUp/HTMLPlus/htmlplus_1.html
(referenced: 2007-04-23)

[ICU4J]
International Components for Unicode.
http://www.icu-project.org/ (referenced: 2007-04-02)

[IEcompat]
[whatwg] Questions: IE 6 Compatibility, HTML 5, Spec Timeframe, and Implementation Timeframe. Ian Hickson. 2005.
http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2005-April/003818.html (referenced: 2007-03-23)

[IIIR-HTML]
Hypertext Markup Language (HTML), working draft. Tim Berners-Lee and Daniel Connolly, editors. IETF, 1993.
http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt

[Infoset]
XML Information Set (Second Edition). John Cowan and Richard Tobin, editors. W3C, 2004.
http://www.w3.org/TR/2004/REC-xml-infoset-20040204/

[IntroXML]
An Introduction to XML and Web Technologies. Anders Møller and Michael I. Schwartzbach. Addison-Wesley, 2006. ISBN: 0321269667.
http://www.brics.dk/ixwt/ (referenced: 2007-04-23)

[ISO10646]
ISO/IEC 10646:2003(E), *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*. ISO, 2003.
http://standards.iso.org/ittf/PubliclyAvailableStandards/c039921_ISO_IEC_10646_2003(E).zip

[ISO15445]
ISO/IEC 15445:2000(E), *Information technology – Document description and processing languages – HyperText Markup Language (HTML)*. ISO, 2000.
http://purl.org/NET/ISO+IEC.15445/15445.html (referenced: 2007-04-23)
URL reference to version corrected in 2003.

[ISO15445TC1]
ISO/IEC 15445:2000(E) TC1, *Information technology – Document description and processing languages – HyperText Markup Language (HTML), Technical Corrigendum 1*. ISO, 2002.
http://purl.org/NET/ISO+IEC.15445/TC1.html (referenced: 2007-04-23)

[ISO19757-2]
ISO/IEC 19757-2:2003(E), *Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG*. ISO, 2003.
http://standards.iso.org/ittf/PubliclyAvailableStandards/
c037605_ISO_IEC_19757-2_2003(E).zip

[ISO19757-2Amd1]
ISO/IEC 19757-2:2003/Amd 1:2006(E), *Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar-based validation – RELAX NG – Amendment 1: Compact Syntax*. ISO, 2006.
http://standards.iso.org/ittf/PubliclyAvailableStandards/
c040774_ISO_IEC_19757-2_2003_Amd_1_2006(E).zip

[ISO19757-3]
ISO/IEC 19757-3:2006(E), *Information technology – Document Schema Definition Language (DSDL) – Part 3: Rule-based validation – Schematron*. ISO, 2006.
http://standards.iso.org/ittf/PubliclyAvailableStandards/
c040833_ISO_IEC_19757-3_2006(E).zip

[ISO8879]
ISO 8879:1986, *Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*. ISO, 1986. Published in [Handbook].

[ISO8879TC2]
ISO 8879 TC2. Charles F. Goldfarb, editor. 1997.
http://www1.y12.doe.gov/capabilities/sgml/wg8/document/1955.htm
(referenced: 2007-03-03)

[JavaScript]
Core JavaScript 1.5 Reference. Mozilla Foundation, 2007.
http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference
(referenced: 2007-04-02)

[javascriptURI]
The 'javascript' resource identifier scheme, working draft. Björn Höhrmann. IETF, 2006.
http://ietfreport.isoc.org/all-ids/draft-hoehrmann-javascript-scheme-00.txt (referenced: 2007-03-01)

[Jena]
Jena – A Semantic Web Framework for Java.
http://jena.sourceforge.net/ (referenced: 2007-04-02)

[Jing]
Jing – A RELAX NG validator in Java. James Clark. Thai Open Source Software Center Ltd, 2003.
http://www.thaiopensource.com/relaxng/jing.html (referenced: 2007-03-23)

[JointPosition]
Position Paper for the W3C Workshop on Web Applications and Compound Documents. The Mozilla Foundation and Opera Software, 2004.
http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html (referenced: 2006-10-16)

[Kosek]
Private communication with Jirka Kosek. 2007.

[LangTagRegistry]
Language Subtag Registry. The Internet Corporation for Assigned Names and Numbers, 2007.
http://www.iana.org/assignments/language-subtag-registry (referenced: 2007-04-02)

[libxml2]
The XML C parser and toolkit of Gnome. Daniel Veillard.
http://xmlsoft.org/ (referenced: 2007-04-02)

[M12N]
Modularization of XHTML™. Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer and Ted Wugofski, editors. W3C, 2001.
http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/

[M12N-RNG]
Modularization of XHTML in RELAX NG. James Clark. Thai Open Source Software Center Ltd, 2003.
http://www.thaiopensource.com/relaxng/xhtml/ (referenced: 2007-04-23)

[M12N11]
XHTML™ Modularization 1.1. Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, Ted Wugofski, Daniel Austin, Subramanian Peruvemba and Masayasu Ishikawa, editors. W3C, 2006.
http://www.w3.org/TR/2006/PR-xhtml-modularization-20060213/

[M12N11WD]
XHTML™ Modularization 1.1, working draft. Daniel Austin, Subramanian Peruvemba, Shane McCarron, Masayasu Ishikawa, Mark Birbeck, Murray Altheim, Frank Boumphrey, Sam Dooley, Sebastian Schnitzenbaumer and Ted Wugofski, editors. W3C, 2006.
http://www.w3.org/TR/2006/WD-xhtml-modularization-20060705/

[MathML]
Mathematical Markup Language (MathML) Version 2.0 (Second Edition). David Carlisle, Patrick Ion, Robert Miner and Nico Poppelier, editors. W3C, 2003.
http://www.w3.org/TR/2003/REC-MathML2-20031021/

[Microformats]
Microformats: a pragmatic path to the semantic web. Rohit Khare and Tantek Çelik. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 865–866. ACM Press, 2006. ISBN: 1-59593-323-9.
http://doi.acm.org/10.1145/1135777.1135917

[mod_validator]
mod_validator. Web Thing.
http://apache.webthing.com/mod_validator/ (referenced: 2007-01-30)

[MozFAQ]
Mozilla Web Author FAQ. Henri Sivonen. 2005.
http://www.mozilla.org/docs/web-developer/faq.html
(referenced: 2006-10-18)

[MS-WebApps]
Paper for participation in the W3C Workshop on Web Applications and Compound Documents. Alex Hopmann and Michael Wallent. Microsoft, 2004.
http://www.w3.org/2004/04/webapps-cdf-ws/papers/microsoft.html
(referenced: 2006-10-16)

[MSV]
Sun Multi-Schema Validator. Kohsuke Kawaguchi.
https://msv.dev.java.net/ (referenced: 2007-04-02)

[NetBeansProfiler]
The NetBeans Profiler Project.
http://profiler.netbeans.org/ (referenced: 2007-04-02)

[NewMember]
[whatwg] New WHATWG member. Ian Hickson. 2004.
http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2004-June/
000491.html (referenced: 2007-03-23)

[NIO]
New I/O APIs. Sun Microsystems, Inc., 2002.
http://java.sun.com/j2se/1.4.2/docs/guide/nio/ (referenced: 2007-04-02)

[nxml-mode]
nXML mode. James Clark. Thai Open Source Software Center Ltd, 2004.
http://www.thaiopensource.com/nxml-mode/ (referenced: 2007-03-23)

[OneStat]
Microsoft's Internet Explorer global usage share is 85.81 percent according to OneStat.-com. OneStat.com, 2007.
http://www.onestat.com/html/
aboutus_pressbox50-microsoft-internet-explorer-7-usage.html
(referenced: 2007-03-22)

[OpenSP]
OpenJade Distribution Page.
http://openjade.sourceforge.net/ (referenced: 2007-04-02)

[Opera9]
Web Specifications Supported in Opera 9. Opera Software ASA.
http://www.opera.com/docs/specs/opera9/ (referenced: 2007-03-03)

[oXygen]
<oXygen/> XML Editor & XSLT Debugger. SyncRO Soft Ltd, 2007.
http://oxygenxml.com/ (referenced: 2007-03-23)

[ProducingXML]
HOWTO Avoid Being Called a Bozo When Producing XML. Henri Sivonen. 2005.
http://hsivonen.iki.fi/producing-xml/ (referenced: 2007-02-22)

[Raggett]
Raggett on HTML 4. Dave Raggett, Jenny Lam, Ian Alexander and Michael Kmiec. Addison Wesley Longman, 1998. ISBN: 0-201-17805-2.
http://www.w3.org/People/Raggett/book4/ch02.html
(referenced: 2007-04-23)

[RELAX]
RELAX (Regular Language description for XML). Makoto Murata. 2002.
http://www.xml.gr.jp/relax/ (referenced: 2007-03-03)

[Relaxed]
Relaxed: on the way towards true validation of compound documents. Jirka Kosek and Petr Nálevka. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 427–436. ACM Press, 2006. ISBN: 1-59593-323-9.
http://doi.acm.org/10.1145/1135777.1135841

[RelaxedAnn]
Relaxed - new HTML validation service based on RELAX NG + Schematron. Jirka Kosek. 2005.
http://lists.xml.org/archives/xml-dev/200507/msg00241.html
(referenced: 2007-03-11)

[RelaxedDocBook]
"RELAXED" DocBook Validator. Petr Nálevka and Jirka Kosek.
http://relaxed.vse.cz/docbookvalidator/ (referenced: 2007-03-04)

[RelaxedValidator]
"RELAXED" the HTML validator. Petr Nálevka.
http://relaxed.vse.cz/ (referenced: 2007-03-04)

[Relaxtron]
Combining RELAX NG and Schematron. Eddie Robertsson. O'Reilly Media, Inc., 2004.
http://www.xml.com/pub/a/2004/02/11/relaxtron.html
(referenced: 2007-04-23)

[REST]
Architectural Styles and the Design of Network-based Software Architectures. Roy Thomas Fielding. PhD thesis, University of California, Irvine, 2000.
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
(referenced: 2007-02-28)

[RFC1738]
RFC 1738, *Uniform Resource Locators (URL)*. Tim Berners-Lee, Larry Masinter and Mark McCahill, editors. IETF, 1994.
http://ietf.org/rfc/rfc1738

[RFC1866]
RFC 1866, *Hypertext Markup Language - 2.0*. Tim Berners-Lee and Dan Connolly, editors. IETF, 1995.
http://ietf.org/rfc/rfc1866

[RFC1942]
RFC 1942, *HTML Tables*. Dave Raggett. IETF, 1996.
http://ietf.org/rfc/rfc1942

[RFC2070]
RFC 2070, *Internationalization of the Hypertext Markup Language*. François Yergeau, Gavin Thomas Nicol, Glenn Adams and Martin J. Dürst. IETF, 1997.
http://ietf.org/rfc/rfc2070

[RFC2368]
RFC 2368, *The mailto URL scheme*. Paul E. Hoffman, Larry Masinter and Jamie Zawinski. IETF, 1998.
http://ietf.org/rfc/rfc2368

[RFC2397]
RFC 2397, *The "data" URL scheme*. Larry Masinter. IETF, 1998.
http://ietf.org/rfc/rfc2397

[RFC2616]
RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*. Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach and Tim Berners-Lee. IETF, 1999.
http://ietf.org/rfc/rfc2616

[RFC2818]
RFC 2818, *HTTP Over TLS*. Eric Rescorla. IETF, 2000.
http://ietf.org/rfc/rfc2818

[RFC2854]
RFC 2854, *The 'text/html' Media Type*. Daniel W. Connolly and Larry Masinter. IETF, 2000.
http://www.ietf.org/rfc/rfc2854

[RFC3066]
RFC 3066, *Tags for the Identification of Languages*. Harald Tveit Alvestrand. IETF, 2001.
http://ietf.org/rfc/rfc3066

[RFC3106]
RFC 3106, *ECML v1.1: Field Specifications for E-Commerce*. Donald E. Eastlake and Ted Goldstein. IETF, 2001.
http://www.ietf.org/rfc/rfc3106

[RFC3236]
RFC 3236, *The 'application/xhtml+xml' Media Type*. Mark A. Baker and Peter Stark. IETF, 2002.
http://www.ietf.org/rfc/rfc3236

[RFC3987]
RFC 3987, *Internationalized Resource Identifiers (IRIs)*. Martin Dürst and Michel Suignard. IETF, 2005.
http://www.ietf.org/rfc/rfc3987

[RFC4287]
RFC 4287, *The Atom Syndication Format*. Mark Nottingham and Robert Sayre, editors. IETF, 2005.
http://ietf.org/rfc/rfc4287

[RFC4646]
RFC 4646, *Tags for Identifying Languages*. Addison Phillips and Mark Davis, editors. IETF, 2006.
http://ietf.org/rfc/rfc4646

[RFC4647]
RFC 4647, *Matching of Language Tags*. Addison Phillips and Mark Davis, editors. IETF, 2006.
http://ietf.org/rfc/rfc4647

[Rhino]
Rhino: JavaScript for Java. Norris Boyd, editor. Mozilla Foundation, 2007.
http://www.mozilla.org/rhino/ (referenced: 2007-03-26)

[RNCtutorial]
RELAX NG Compact Syntax Tutorial, working draft. James Clark, John Cowan and Makoto Murata, editors. OASIS, 2003.
http://relaxng.org/compact-tutorial-20030326.html (referenced: 2007-03-04)

[RNG]
RELAX NG Specification. James Clark and Makoto Murata, editors. OASIS, 2001.
http://relaxng.org/spec-20011203.html (referenced: 2007-03-03)

[RNG-XSD]
Guidelines for using W3C XML Schema Datatypes with RELAX NG. James Clark and Kohsuke Kawaguchi, editors. OASIS, 2001.
http://relaxng.org/xsd-20010907.html (referenced: 2007-04-23)

[RNGdesign]
The Design of RELAX NG. James Clark.
http://www.thaiopensource.com/relaxng/design.html (referenced: 2007-04-23)

[Ruby]
Ruby Annotation. Marcin Sawicki, Michel Suignard, Masayasu Ishikawa, Martin Dürst and Tex Texin, editors. W3C, 2001.
http://www.w3.org/TR/2001/REC-ruby-20010531/

[RubyIE]
RUBY Element | ruby Object. Microsoft Corporation.
http://msdn.microsoft.com/workshop/author/dhtml/reference/objects/ruby.asp (referenced: 2007-04-25)

[SAX]
SAX. David Megginson and David Brownell.
http://www.saxproject.org/ (referenced: 2007-03-03)

[SAX2]
SAX2. David Brownell. O'Reilly, 2002. ISBN: 0-596-00237-8.
http://safari.oreilly.com/0596002378

[SaxCompiler]
SaxCompiler. Henri Sivonen. 2005.
http://hsivonen.iki.fi/saxcompiler/ (referenced: 2007-02-22)

[SAXON]
About SAXON. Michael H. Kay. Saxonica Limited, 2005.
http://saxon.sourceforge.net/saxon6.5.5/ (referenced: 2007-04-02)

[Schematron15]
The Schematron Assertion Language 1.5. Rick Jelliffe. Academia Sinica Computing Centre, 2002.
http://xml.ascc.net/resource/schematron/Schematron2000.html
(referenced: 2007-04-23)

[SchematronHeuristic]
Optimizing Time-Performance of Streaming Schematon. Rick Jelliffe. 2002.
http://www.topologi.com/resources/pdfs/SchematronHeuristic.pdf
(referenced: 2007-03-02)

[SchematronOld]
The Schematron – An XML Structure Validation Language using Patterns in Trees. Rick Jelliffe. Academia Sinica Computing Centre, 2001.
http://xml.ascc.net/resource/schematron/old-index.html
(referenced: 2006-09-25)

[SchemaUE]
W3C Workshop on XML Schema 1.0 User Experiences and Interoperability. W3C, 2005.
http://www.w3.org/2005/03/xml-schema-user-program.html
(referenced: 2007-04-23)

[Schneegans]
XML Schema Validator. Christoph Schneegans.
http://schneegans.de/sv/ (referenced: 2007-03-04)

[Several]
Re: [whatwg] several messages about HTML5. Ian Hickson. 2007.
http://lists.whatwg.org/pipermail/whatwg-whatwg.org/2007-February/
009517.html (referenced: 2007-02-27)

[Sniffing]
Re: Sniffing XHTML sent as text/html. Steven Pemberton. 2000.
http://lists.w3.org/Archives/Public/www-html/2000Sep/0024.html
(referenced: 2007-03-01)

[SoupDOM]
Tag Soup: How UAs handle <x> <y> </x> </y>. Ian Hickson. 2002.
http://ln.hixie.ch/?start=1037910467&count=1 (referenced: 2007-03-23)

[SP]
SP. James Clark.
http://www.jclark.com/sp/ (referenced: 2007-04-02)

[Stats]
Web Authoring Statistics. Google, 2005.
http://code.google.com/webstats/ (referenced: 2007-02-26)

[SVG]
Scalable Vector Graphics (SVG) 1.1 Specification. Jon Ferraiolo, Jun Fujisawa and Dean Jackson, editors. W3C, 2003.
http://www.w3.org/TR/2003/REC-SVG11-20030114/

[TableCell]
nsHTMLTableCellElement.cpp. Mozilla Foundation, 2006.
http://mxr.mozilla.org/seamonkey/source/content/html/content/src/nsHTMLTableCellElement.cpp (referenced: 2007-04-02)

[TagSoup]
TagSoup: A SAX parser in Java for nasty, ugly HTML. John Cowan. 2004.
http://home.ccil.org/~cowan/XML/tagsoup/tagsoup.pdf
(referenced: 2007-03-03)
Presented at Extreme Markup Languages 2004.

[Taxonomy]
Taxonomy of XML schema languages using formal language theory. Makoto Murata, Dongwon Lee, Murali Mani and Kohsuke Kawaguchi. In *ACM Trans. Inter. Tech.*, volume 5, number 4, pages 660–704. ACM Press, 2005. ISSN: 1533-5399.
http://doi.acm.org/10.1145/1111627.1111631

[TheCounter]
Browser Stats. Jupitermedia Corporation, 2007.
http://www.thecounter.com/stats/2007/February/browser.php
(referenced: 2007-03-22)

[ToBeDeleted]
<draft-ietf-iiir-html-01.txt, .ps> to be deleted.. W. Eliot Kimber. 1994.
http://1997.webhistory.org/www.lists/www-talk.1994q1/0573.html
(referenced: 2007-04-23)

[TREX]
TREX – Tree Regular Expressions for XML. James Clark.
http://www.thaiopensource.com/trex/ (referenced: 2007-03-03)

[UAX15]
Unicode Normalization Forms. Mark Davis and Martin Dürst. Unicode, Inc., 2006.
http://www.unicode.org/reports/tr15/tr15-27.html (referenced: 2007-03-03)

[Understanding]
Understanding HTML, XML and XHTML. Maciej Stachowiak. 2006.
http://webkit.org/blog/?p=68 (referenced: 2006-10-14)
The surname of the author isn't mentioned in the article itself.

[Unicode]
The Unicode Standard, Version 5.0. The Unicode Consortium. Addison-Wesley, 2006. ISBN: 0-321-48091-0.
http://www.unicode.org/versions/Unicode5.0.0/ (referenced: 2007-03-03)
Version 5.0 not yet online at the time of the publication of this thesis.

[Valet]
Page Valet. WebThing Ltd.
http://valet.webthing.com/page/ (referenced: 2007-04-02)

[ValetMode]
Parse Modes - Page Valet Help. Web Thing.
http://valet.webthing.com/page/parsemode.html (referenced: 2007-01-30)

[Validace]
Doplňková validace HTML a XHTML dokumentů. Petr Nálevka. Bachelor's thesis, University of Economics, Prague, 2005.
http://relaxed.sourceforge.net/thesis_cz.html (referenced: 2007-04-23)

[ValidatorAbout]
About the Validation Service. Henri Sivonen. 2007.
http://hsivonen.iki.fi/validator-about/ (referenced: 2007-02-27)

[ValidatorAbout2007]
About the Validation Service. Henri Sivonen. 2007.
http://hsivonen.iki.fi/validator-about/2007-05-07 (referenced: 2007-05-07)

[Validome]
Validome HTML / XHTML / WML / XML Validator.
http://www.validome.org/ (referenced: 2007-03-04)

[ValidomeStaff]
Private communication with Validome Staff. 2006.

[W3C-DTF]
Date and Time Formats. Misha Wolf and Charles Wicksteed. W3C, 1998.
http://www.w3.org/TR/1998/NOTE-datetime-19980827

[W3Cvalidator]
The W3C Markup Validation Service v0.7.4. W3C.
http://validator.w3.org/ (referenced: 2007-01-24)

[WaterlooGML]
Waterloo SCRIPT GML User's Guide. University of Waterloo, 1988.
http://www.uga.edu/~ucns/stddocs/script-gmlref-tso.txt
(referenced: 2007-04-23)

[WCAG]
Web Content Accessibility Guidelines 1.0. Wendy Chisholm, Gregg Vanderheiden and Ian Jacobs, editors. W3C, 1999.
http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/

[WDG]
WDG HTML Validator. Liam Quinn.
http://www.htmlhelp.com/tools/validator/ (referenced: 2007-01-25)

[WDG1998]
What Makes the WDG HTML Validator Special. Liam Quinn. 1998.
http://web.archive.org/web/19990128203022/htmlhelp.com/tools/validator/differences.html (referenced: 2007-04-23)

[WDG2007]
How the WDG HTML Validator differs from others. Liam Quinn.
http://www.htmlhelp.com/tools/validator/differences.html.en
(referenced: 2007-01-25)

[Weaving]
Weaving the Web. Tim Berners-Lee. HarperBusiness, 2000. ISBN: 0-06-251587-X.

[WebApps]
Web Applications 1.0, working draft. Ian Hickson, editor. WHATWG, 2006.
http://whatwg.org/specs/web-apps/current-work/

I used three primary snapshots of the specification draft in this project. I took the first snapshot at the start of the thesis project. The snapshot was dated February 24 2006. I took the second snapshot in November 2006. The snapshot was dated November 1 2006. In March 2007, I took a third snapshot. It was dated March 9 2007.

[WebForms2]
Web Forms 2.0, working draft. Ian Hickson, editor. WHATWG, 2006.
http://whatwg.org/specs/web-forms/current-work/

I used two primary snapshots of the specification draft in this project. I took the first snapshot at the start of the thesis project. The snapshot was dated January 10 2006. I took the second snapshot was taken in November 2006. The snapshot was dated October 12 2006. In March 2007, a third snapshot was not necessary, because the specification had not changed.

[WebServiceIdeas]
Validator Web Service Interface Ideas. Henri Sivonen. 2006.
http://hsivonen.iki.fi/validator-ws-ideas/ (referenced: 2007-03-23)

[WHAT-Ann]
WHAT open mailing list announcement. Ian Hickson. WHATWG, 2004.
http://whatwg.org/news/start (referenced: 2006-10-18)

[WHAT-Charter]
Web Hypertext Application Technology Working Group Charter. WHATWG.
http://whatwg.org/charter (referenced: 2006-10-19)

[Wilson]
Jon Udell: Chris Wilson on IE7, Ajax, and web standards. Jon Udell and Chris Wilson. Microsoft, 2007.
http://channel9.msdn.com/podcasts/MSConversations_wilson_ch9.mp3
(referenced: 2007-02-26)

[XBL2]
XML Binding Language (XBL) 2.0, working draft. Ian Hickson, editor. W3C, 2007.
http://www.w3.org/TR/2007/WD-xbl-20070117/

[XDuce]
XDuce: A statically typed XML processing language. Haruo Hosoya and Benjamin C. Pierce. In *ACM Trans. Inter. Tech.*, volume 3, number 2, pages 117–148. ACM Press, 2003. ISSN: 1533-5399.
http://doi.acm.org/10.1145/767193.767195

[XercesC]
Xerces C++ Parser. The Apache Software Foundation.
http://xml.apache.org/xerces-c/ (referenced: 2007-04-02)

[XForms]
XForms 1.0. Micah Dubinko, Leigh L. Klotz, Jr., Roland Merrick and T. V. Raman, editors. W3C, 2003.
http://www.w3.org/TR/2003/REC-xforms-20031014/

[XHTML-MP]
XHTML Mobile Profile. WAP Forum, 2001.
http://www.openmobilealliance.org/tech/affiliates/wap/
wap-277-xhtmlmp-20011029-a.pdf (referenced: 2007-04-23)

[XHTML10]
XHTML™ 1.0: The Extensible HyperText Markup Language. Steven Pemberton et al. W3C, 2000.
http://www.w3.org/TR/2000/REC-xhtml1-20000126/

[XHTML10XSD]
XHTML™ 1.0 in XML Schema. Masayasu Ishikawa, editor. W3C, 2002.
http://www.w3.org/TR/2002/NOTE-xhtml1-schema-20020902/

[XHTML11]
XHTML™ 1.1 – Module-based XHTML. Murray Altheim and Shane McCarron, editors. W3C, 2001.
http://www.w3.org/TR/2001/REC-xhtml11-20010531/

[XHTML20]
XHTML™ 2.0, working draft. Jonny Axelsson, Beth Epperson, Masayasu Ishikawa, Shane McCarron, Ann Navarro and Steven Pemberton, editors. W3C, 2003.
http://www.w3.org/TR/2003/WD-xhtml2-20030506/

[XHTMLBasic]
XHTML™ Basic. Mark Baker, Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Ted Wugofski and Toshihiko Yamakami, editors. W3C, 2000.
http://www.w3.org/TR/2000/REC-xhtml-basic-20001219/

[XML]
Extensible Markup Language (XML) 1.0 (Fourth Edition). Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler and François Yergeau, editors. W3C, 2006.
http://www.w3.org/TR/2006/REC-xml-20060816/

[XML11]
Extensible Markup Language (XML) 1.1 (Second Edition). Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau and John Cowan, editors. W3C, 2006.
http://www.w3.org/TR/2006/REC-xml11-20060816/
The document has been edited in place on 29 September 2006.

[xmlid]
xml:id Version 1.0. Jonathan Marsh, Daniel Veillard and Norman Walsh, editors. W3C, 2005.
http://www.w3.org/TR/2005/REC-xml-id-20050909/

[XMLNS]
Namespaces in XML 1.0 (Second Edition). Tim Bray, Dave Hollander, Andrew Layman and Richard Tobin, editors. W3C, 2006.
http://www.w3.org/TR/2006/REC-xml-names-20060816/

[XOM]
XOM. Elliotte Rusty Harold. 2006.
http://xom.nu/ (referenced: 2007-03-23)

[XPath]
XML Path Language (XPath). James Clark and Steve DeRose, editors. W3C, 1999.
http://www.w3.org/TR/1999/REC-xpath-19991116

[XSD]
XML Schema Part 1: Structures Second Edition. Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn, editors. W3C, 2004.
http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
(referenced: 2007-03-03)

[XSDDatatypes]
XML Schema Part 2: Datatypes Second Edition. Paul V. Biron and Ashok Malhotra, editors. W3C, 2004.
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[XSDDatatypes11WD]
XML Schema 1.1 Part 2: Datatypes, working draft. David Peterson, Paul V. Biron, Ashok Malhotra and C. M. Sperberg-McQueen, editors. W3C, 2006.
http://www.w3.org/TR/2006/WD-xmlschema11-2-20060217/

[XSDDatatypesFE]
XML Schema Part 2: Datatypes. Paul V. Biron and Ashok Malhotra, editors. W3C, 2001.
http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[XSLT]
XSL Transformations (XSLT) Version 1.0. James Clark, editor. W3C, 1999.
http://www.w3.org/TR/1999/REC-xslt-19991116

Appendix: Table Integrity Checker

```
public final class TableChecker extends Checker {

  private Table current;

  /** Grows from the tail. */
  private final LinkedList<Table> stack = new LinkedList<Table>();

  private void push() {
    if (current != null) {
      stack.addLast(current);
    }
    current = new Table(this);
  }

  private void pop() throws SAXException {
    if (current == null) {
      throw new IllegalStateException("Bug!");
    }
    current.end();
    if (stack.isEmpty()) {
      current = null;
    } else {
      current = stack.removeLast();
    }
  }

  public void startElement(String uri, String localName, String qName,
      Attributes atts) throws SAXException {
    if ("http://www.w3.org/1999/xhtml".equals(uri)) {
      if ("table".equals(localName)) {
        push();
      } else if (current != null) {
        if ("td".equals(localName)) {
          current.startCell(false, atts);
        } else if ("th".equals(localName)) {
          current.startCell(true, atts);
        } else if ("tr".equals(localName)) {
          current.startRow();
        } else if ("tbody".equals(localName)
            || "thead".equals(localName)
            || "tfoot".equals(localName)) {
          current.startRowGroup(localName);
        } else if ("col".equals(localName)) {
          current.startCol(AttributeUtil.parseNonNegativeInteger(atts.getValue(
              "", "span")));
        } else if ("colgroup".equals(localName)) {
          current.startColGroup(AttributeUtil.parseNonNegativeInteger(atts.getValue(
              "", "span")));
        }
      }
    }
  }

  public void endElement(String uri, String localName, String qName)
      throws SAXException {
    if ("http://www.w3.org/1999/xhtml".equals(uri)) {
      if ("table".equals(localName)) {
        pop();
      } else if (current != null) {
        if ("td".equals(localName)) {
          current.endCell();
        } else if ("th".equals(localName)) {
          current.endCell();
        } else if ("tr".equals(localName)) {
          current.endRow();
        } else if ("tbody".equals(localName)
            || "thead".equals(localName)
            || "tfoot".equals(localName)) {
          current.endRowGroup();
        } else if ("col".equals(localName)) {
          current.endCol();
        } else if ("colgroup".equals(localName)) {
          current.endColGroup();
        }
      }
    }
  }

  public void reset() {
    stack.clear();
    current = null;
  }
}

final class Table {

  /** An enumeration for keeping track of the parsing state of a table. */
  private enum State {

    /** The table element start has been seen. No child elements have been seen.
     *  A start of a column, a column group, a row or a row group or the end of
     *  the table is expected. */
    IN_TABLE_AT_START,

    /** The table element is the open element and rows have been seen. A row in
     *  an implicit group, a row group or the end of the table is expected. */
    IN_TABLE_AT_POTENTIAL_ROW_GROUP_START,

    /** A column group is open. It can end or a column can start. */
    IN_COLGROUP,

    /** * A column inside a column group is open. It can end. */
    IN_COL_IN_COLGROUP,

    /** A column that is a child of table is open. It can end. */
    IN_COL_IN_IMPLICIT_GROUP,

    /** The open element is an explicit row group. It may end or a row may start. */
    IN_ROW_GROUP,

    /** A row in a an explicit row group is open. It may end or a cell may start. */
    IN_ROW_IN_ROW_GROUP,
    /** A cell inside a row inside an explicit row group is open. It can end. */
    IN_CELL_IN_ROW_GROUP,

    /** A row in an implicit row group is open. It may end or a cell may start. */
    IN_ROW_IN_IMPLICIT_ROW_GROUP,

    /** The table itself is the currently open element, but an implicit row group
     *  been started by previous rows. A row may start, an explicit row group may
     *  start or the table may end.  */
    IN_IMPLICIT_ROW_GROUP,

    /** A cell inside an implicit row group is open. It can close. */
    IN_CELL_IN_IMPLICIT_ROW_GROUP,

    /** The table itself is the currently open element. Columns and/or column groups
     *  have been seen but rows or row groups have not been seen yet. A column, a
     *  column group, a row or a row group can start. The table can end. */
    IN_TABLE_COLS_SEEN
  }

  private State state = State.IN_TABLE_AT_START;

  private final TableChecker owner;

  /** The number of suppressed element starts. */
  private int suppressedStarts = 0;

  /** Indicates whether the width of the table was established by column markup. */
  private boolean hardWidth = false;

  /** The column count established by column markup or by the first row. */
  private int columnCount = -1;

  /** The actual column count as stretched by the widest row. */
  private int realColumnCount = 0;

  /** A colgroup span that hasn't been actuated yet in case the element has
   *  col children. The absolute value counts. The negative sign means that
   *  the value was implied. */
  private int pendingColGroupSpan = 0;

  /** A set of the IDs of header cells. */
  private final Set<String> headerIds = new HashSet<String>();

  /** A list of cells that refer to headers (in the document order). */
  private final List<Cell> cellsReferringToHeaders = new LinkedList<Cell>();

  /** The current row group (also implicit groups have an explicit object). */
  private RowGroup current;

  /** The head of the column range list. */
  private ColumnRange first = null;

  /** The tail of the column range list. */
  private ColumnRange last = null;

  /** The range under inspection. */
  private ColumnRange currentColRange = null;

  /** The previous range that was inspected. */
  private ColumnRange previousColRange = null;
```

```
/** Constructor.
 * @param owner reference back to the checker */
public Table(TableChecker owner) {
  super();
  this.owner = owner;
}

private boolean needSuppressStart() {
  if (suppressedStarts > 0) {
    suppressedStarts++;
    return true;
  } else {
    return false;
  }
}

private boolean needSuppressEnd() {
  if (suppressedStarts > 0) {
    suppressedStarts--;
    return true;
  } else {
    return false;
  }
}

void startRowGroup(String type) throws SAXException {
  if (needSuppressStart()) {
    return;
  }
  switch (state) {
    case IN_IMPLICIT_ROW_GROUP:
      current.end();
    // fall through
    case IN_TABLE_AT_START:
    case IN_TABLE_COLS_SEEN:
    case IN_TABLE_AT_POTENTIAL_ROW_GROUP_START:
      current = new RowGroup(this, type);
      state = State.IN_ROW_GROUP;
      break;
    default:
      suppressedStarts = 1;
      break;
  }
}

void endRowGroup() throws SAXException {
  if (needSuppressEnd()) {
    return;
  }
  switch (state) {
    case IN_ROW_GROUP:
      current.end();
      current = null;
      state = State.IN_TABLE_AT_POTENTIAL_ROW_GROUP_START;
      break;
    default:
      throw new IllegalStateException("Bug!");
  }
}

void startRow() {
  if (needSuppressStart()) {
    return;
  }
  switch (state) {
    case IN_TABLE_AT_START:
    case IN_TABLE_COLS_SEEN:
    case IN_TABLE_AT_POTENTIAL_ROW_GROUP_START:
      current = new RowGroup(this, null);
      // fall through
    case IN_IMPLICIT_ROW_GROUP:
      state = State.IN_ROW_IN_IMPLICIT_ROW_GROUP;
      break;
    case IN_ROW_GROUP:
      state = State.IN_ROW_IN_ROW_GROUP;
      break;
    default:
      suppressedStarts = 1;
      return;
  }
  currentColRange = first;
  previousColRange = null;
  current.startRow();
}

void endRow() throws SAXException {
  if (needSuppressEnd()) {
    return;
  }
  switch (state) {
    case IN_ROW_IN_ROW_GROUP:
      state = State.IN_ROW_GROUP;
      break;
    case IN_ROW_IN_IMPLICIT_ROW_GROUP:
      state = State.IN_IMPLICIT_ROW_GROUP;
      break;
    default:
      throw new IllegalStateException("Bug!");
  }
  current.endRow();
}

void startCell(boolean header, Attributes attributes) throws SAXException {
  if (needSuppressStart()) {
    return;
  }
  switch (state) {
    case IN_ROW_IN_ROW_GROUP:
      state = State.IN_CELL_IN_ROW_GROUP;
      break;
    case IN_ROW_IN_IMPLICIT_ROW_GROUP:
      state = State.IN_CELL_IN_IMPLICIT_ROW_GROUP;
      break;
    default:
      suppressedStarts = 1;
      return;
  }
  if (header) {
    int len = attributes.getLength();
    for (int i = 0; i < len; i++) {
      if ("ID".equals(attributes.getType(i))) {
        headerIds.add(attributes.getValue(i));
      }
    }
  }
  String[] headers = AttributeUtil.split(attributes.getValue("",
      "headers"));
  Cell cell = new Cell(
      Math.abs(AttributeUtil.parsePositiveInteger(attributes.getValue(
          "", "colspan"))),
      Math.abs(AttributeUtil.parseNonNegativeInteger(attributes.getValue(
          "", "rowspan"))), headers, header,
      owner.getDocumentLocator(), owner.getErrorHandler());
  if (headers.length > 0) {
    cellsReferringToHeaders.add(cell);
  }
  current.cell(cell);
}

void endCell() {
  if (needSuppressEnd()) {
    return;
  }
  switch (state) {
    case IN_CELL_IN_ROW_GROUP:
      state = State.IN_ROW_IN_ROW_GROUP;
      break;
    case IN_CELL_IN_IMPLICIT_ROW_GROUP:
      state = State.IN_ROW_IN_IMPLICIT_ROW_GROUP;
      break;
    default:
      throw new IllegalStateException("Bug!");
  }
}

void startColGroup(int span) {
  if (needSuppressStart()) {
    return;
  }
  switch (state) {
    case IN_TABLE_AT_START:
      hardWidth = true;
      columnCount = 0;
    // fall through
    case IN_TABLE_COLS_SEEN:
      pendingColGroupSpan = span;
      state = State.IN_COLGROUP;
      break;
    default:
      suppressedStarts = 1;
      break;
  }
}

void endColGroup() {
  if (needSuppressEnd()) {
    return;
  }
  switch (state) {
    case IN_COLGROUP:
      int right = columnCount + Math.abs(pendingColGroupSpan);
      Locator locator = new LocatorImpl(owner.getDocumentLocator());
      ColumnRange colRange = new ColumnRange("colgroup", locator,
          columnCount, right);
      appendColumnRange(colRange);
      columnCount = right;
      realColumnCount = columnCount;
      state = State.IN_TABLE_COLS_SEEN;
      break;
    default:
      throw new IllegalStateException("Bug!");
  }
}

void startCol(int span) throws SAXException {
  if (needSuppressStart()) {
    return;
  }
  switch (state) {
    case IN_TABLE_AT_START:
      hardWidth = true;
      columnCount = 0;
    // fall through
    case IN_TABLE_COLS_SEEN:
      state = State.IN_COL_IN_IMPLICIT_GROUP;
      break;
    case IN_COLGROUP:
      if (pendingColGroupSpan > 0) {
        warn("A col element causes a span attribute with value "
            + pendingColGroupSpan
            + " to be ignored on the parent colgroup.");
      }
      pendingColGroupSpan = 0;
      state = State.IN_COL_IN_COLGROUP;
      break;
    default:
      suppressedStarts = 1;
      return;
  }
  int right = columnCount + Math.abs(span);
  Locator locator = new LocatorImpl(owner.getDocumentLocator());
  ColumnRange colRange = new ColumnRange("col", locator,
      columnCount, right);
  appendColumnRange(colRange);
  columnCount = right;
  realColumnCount = columnCount;
}

private void appendColumnRange(ColumnRange colRange) {
  if (last == null) {
    first = colRange;
    last = colRange;
  } else {
    last.setNext(colRange);
    last = colRange;
  }
}

void warn(String message) throws SAXException {
  owner.warn(message);
}

void err(String message) throws SAXException {
  owner.err(message);
}

void endCol() {
  if (needSuppressEnd()) {
    return;
  }
  switch (state) {
    case IN_COL_IN_IMPLICIT_GROUP:
      state = State.IN_TABLE_COLS_SEEN;
      break;
    case IN_COL_IN_COLGROUP:
      state = State.IN_COLGROUP;
      break;
    default:
      throw new IllegalStateException("Bug!");
  }
}

void end() throws SAXException {
  switch (state) {
    case IN_IMPLICIT_ROW_GROUP:
      current.end();
      current = null;
      break;
    case IN_TABLE_AT_START:
    case IN_TABLE_AT_POTENTIAL_ROW_GROUP_START:
    case IN_TABLE_COLS_SEEN:
      break;
    default:
      throw new IllegalStateException("Bug!");
  }

  // Check referential integrity
  for (Iterator<Cell> iter = cellsReferringToHeaders.iterator(); iter.hasNext();) {
    Cell cell = iter.next();
    String[] headings = cell.getHeadings();
    for (int i = 0; i < headings.length; i++) {
      String heading = headings[i];
      if (!headerIds.contains(heading)) {
        cell.err("The \u201Cheaders\u201D attribute on the element \u201C"
            + cell.elementName()
            + "\u201D refers to the ID \u201C"
            + heading
            + "\u201D, but there is no \u201Cth\u201D element with that ID"
            + " in the same table.");
      }
    }
  }

  // Check that each column has non-extended cells
  ColumnRange colRange = first;
  while (colRange != null) {
    if (colRange.isSingleCol()) {
      owner.getErrorHandler().error(
          new SAXParseException("Table column " + colRange
              + " established by element \u201C"
              + colRange.getElement()
              + "\u201D has no cells beginning in it.",
              colRange.getLocator()));
    } else {
      owner.getErrorHandler().error(
          new SAXParseException("Table columns in range "
              + colRange + " established by element \u201C"
              + colRange.getElement()
              + "\u201D have no cells beginning in them.",
              colRange.getLocator()));
    }
    colRange = colRange.getNext();
  }
}
```

```
  /** Reports a cell whose positioning has been decided back to the table
   *  so that column bookkeeping can be done. (Called from
   *  <code>RowGroup</code>--not <code>TableChecker</code>.) */
  void cell(Cell cell) {
    int left = cell.getLeft();
    int right = cell.getRight();
    // first see if we've got a cell past the last col
    if (right > realColumnCount) {
      // are we past last col entirely?
      if (left == realColumnCount) {
        // single col?
        if (left + 1 != right) {
          appendColumnRange(new ColumnRange(cell.elementName(), cell, left + 1, right));
        }
        realColumnCount = right;
        return;
      } else {
        // not past entirely
        appendColumnRange(new ColumnRange(cell.elementName(), cell, realColumnCount,
                          right));
        realColumnCount = right;
      }
    }
    while (currentColRange != null) {
      int hit = currentColRange.hits(left);
      if (hit == 0) {
        ColumnRange newRange = currentColRange.removeColumn(left);
        if (newRange == null) {
          // zap a list item
          if (previousColRange != null) {
            previousColRange.setNext(currentColRange.getNext());
          }
          if (first == currentColRange) {
            first = currentColRange.getNext();
          }
          if (last == currentColRange) {
            last = previousColRange;
          }
          currentColRange = currentColRange.getNext();
        } else {
          if (last == currentColRange) {
            last = newRange;
          }
          currentColRange = newRange;
        }
        return;
      } else if (hit == -1) {
        return;
      } else if (hit == 1) {
        previousColRange = currentColRange;
        currentColRange = currentColRange.getNext();
      }
    }
  }

  int getColumnCount() { return columnCount; }
  void setColumnCount(int columnCount) { this.columnCount = columnCount; }
  boolean isHardWidth() { return hardWidth; }
}


/** Represents a contiguous range of columns that was established by a single
 *  element and that does not yet have cells in it. */
```

```
final class ColumnRange {

  /** The locator associated with the element that established this column range. */
  private final Locator locator;

  /** The local name of the element that established this column range. */
  private final String element;

  /** The leftmost column that is part of this range. */
  private int left;

  /** The first column to the right that is not part of this range. */
  private int right;

  /** The next range in the linked list of ranges. */
  private ColumnRange next;

  /** Constructor
   * @param element the local name of the establishing element
   * @param locator a locator associated with the establishing element;
   * <em>must be suitable for retaining out-of-SAX-event!</em>
   * @param left the leftmost column that is part of this range
   * @param right the first column to the right that is not part of this range */
  public ColumnRange(String element, Locator locator, int left, int right) {
    super();
    this.element = element;
    this.locator = locator;
    this.left = left;
    this.right = right;
    this.next = null;
  }

  /** Hit testing.
   * @param column column index
   * @return -1 if the column is to the left of this range,
   * 0 if the column is in this range and
   * 1 if the column is to the right of this range */
  int hits(int column) {
    if (column < left) {
      return -1;
    } if (column >= right) {
      return 1;
    } else {
      return 0;
    }
  }

  /** Removes a column from the range possibly asking it to be destroyed or
   * splitting it.
   * @param column a column index
   * @return <code>null</code> if this range gets destroyed,
   * <code>this</code> if the range gets resized and
   * the new right half range if the range gets split */
  ColumnRange removeColumn(int column) {
    // first, let's see if this is a 1-column range that should
    // be destroyed
    if (isSingleCol()) {
      return null;
    } else if (column == left) {
      left++;
      return this;
    } else if (column + 1 == right) {
      right--;
      return this;
    } else {
      ColumnRange created = new ColumnRange(this.element, this.locator,
          column + 1, this.right);
      created.next = this.next;
      this.next = created;
      this.right = column;
      return created;
    }
  }

  boolean isSingleCol() { return left + 1 == right; }

  public String toString() {
    if (isSingleCol()) {
      return Integer.toString(right);
    } else {
      return (left + 1) + "\u2026" + (right);
    }
  }

  ColumnRange getNext() { return next; }
  void setNext(ColumnRange next) { this.next = next; }
  String getElement() { return element; }
  Locator getLocator() { return locator; }
}

/** Represents a row group (explicit or implicit) for table integrity checking. */
final class RowGroup {

  /** Runtime type constant. */
  private final Cell[] EMPTY_CELL_ARRAY = {};

  /** Keeps track of the current slot row of the insertion point. */
  private int currentRow = -1;

  /** The column slot of the insertion point. */
  private int insertionPoint = 0;

  /** The index of the next uninspected item in <code>cellsOnCurrentRow</code>. */
  private int nextOldCell = 0;

  /** The owning table. */
  private final Table owner;

  /** Cells from previous rows that are still in effect extending downwards. */
  private final SortedSet<Cell> cellsIfEffect = new TreeSet<Cell>(
      VerticalCellComparator.THE_INSTANCE);

  /** A temporary copy of <code>cellsIfEffect</code> sorted differently. */
  private Cell[] cellsOnCurrentRow;

  /** Whether the current row has had cells. */
  private boolean rowHadCells;

  /** The local name of the element that established this row group or
   *  <code>null</code> if this is an implicit row group. */
  private final String type;

  RowGroup(Table owner, String type) {
    super();
    this.owner = owner;
    this.type = type;
```

```
}

public void cell(Cell cell) throws SAXException {
  rowHadCells = true;
  findInsertionPoint();
  cell.setPosition(currentRow, insertionPoint);
  owner.cell(cell);
  if (cell.getBottom() > currentRow + 1) {
    cellsIfEffect.add(cell);
  }
  insertionPoint = cell.getRight();
  for (int i = nextOldCell; i < cellsOnCurrentRow.length; i++) {
    cellsOnCurrentRow[i].errOnHorizontalOverlap(cell);
  }
}

private void findInsertionPoint() {
  for (;;) {
    if (nextOldCell == cellsOnCurrentRow.length) {
      break;
    }
    Cell other = cellsOnCurrentRow[nextOldCell];
    int newInsertionPoint = other.freeSlot(insertionPoint);
    if (newInsertionPoint == insertionPoint) {
      break;
    }
    nextOldCell++;
    insertionPoint = newInsertionPoint;
  }
}

public void end() throws SAXException {
  for (Cell cell : cellsIfEffect) {
    cell.errIfNotRowspanZero(type);
  }
}

public void endRow() throws SAXException {
  if (!rowHadCells) {
    owner.err("Row "
        + (currentRow + 1)
        + " of "
        + (type == null ? "an implicit row group"
            : "a row group established by a \u201C" + type
              + "\u201D element")
        + " has no cells beginning on it.");
  }

  findInsertionPoint();
  cellsOnCurrentRow = null;

  int columnCount = owner.getColumnCount();
  if (owner.isHardWidth()) {
    if (insertionPoint > columnCount) {
      owner.err("A table row was "
          + insertionPoint
          + " columns wide and exceeded the column count established using column"
          + " markup ("
          + columnCount + ").");
    } else if (insertionPoint < columnCount) {
      owner.err("A table row was "
          + insertionPoint
          + " columns wide, which is less than the column count established using"
```

```
            + " column markup ("
            + columnCount + ").");
      }
    } else if (columnCount == -1) {
      // just saw the first row
      owner.setColumnCount(insertionPoint);
    } else {
      if (insertionPoint > columnCount) {
        owner.warn("A table row was "
            + insertionPoint
            + " columns wide and exceeded the column count established by the"
            + " first row ("
            + columnCount + ").");
      } else if (insertionPoint < columnCount) {
        owner.warn("A table row was "
            + insertionPoint
            + " columns wide, which is less than the column count established"
            + " by the first row ("
            + columnCount + ").");
      }
    }

    // Get rid of cells that don't span to the next row
    for (Iterator<Cell> iter = cellsIfEffect.iterator(); iter.hasNext();) {
      Cell cell = iter.next();
      if (cell.shouldBeCulled(currentRow + 1)) {
        iter.remove();
      }
    }
  }

  public void startRow() {
    currentRow++;
    insertionPoint = 0;
    nextOldCell = 0;
    rowHadCells = false;
    cellsOnCurrentRow = cellsIfEffect.toArray(EMPTY_CELL_ARRAY);
    // the array should already be in the right order most of the time
    Arrays.sort(cellsOnCurrentRow, HorizontalCellComparator.THE_INSTANCE);
  }

}


final class Cell implements Locator {

  private static final int MAX_COLSPAN = 1000;
  private static final int MAX_ROWSPAN = 8190;

  /** The column in which this cell starts. (Zero before positioning.) */
  private int left;

  /** The first row in the row group onto which this cell does not span.
   * (rowspan before positioning)
   *
   * <p>However, <code>Integen.MAX_VALUE</code> is a magic value that means
   * <code>rowspan=0</code>. */
  private int bottom;

  /** The first column into which this cell does not span.
   * (colspan before positioning.) */
  private int right;
```

```
/** The value of the <code>headers</code> attribute split on white space. */
private final String[] headers;

/** Whether this is a <code>th</code> cell. */
private final boolean header;

/** Source column. */
private final int columnNumber;

/** Source line. */
private final int lineNumber;

/** Source public id. */
private final String publicId;

/** Source system id. */
private final String systemId;

/** The error handler. */
private final ErrorHandler errorHandler;

Cell(int colspan, int rowspan, String[] headers, boolean header,
    Locator locator, ErrorHandler errorHandler) throws SAXException {
  super();
  this.errorHandler = errorHandler;
  if (locator == null) {
    this.columnNumber = -1;
    this.lineNumber = -1;
    this.publicId = null;
    this.systemId = null;
  } else {
    this.columnNumber = locator.getColumnNumber();
    this.lineNumber = locator.getLineNumber();
    this.publicId = locator.getPublicId();
    this.systemId = locator.getSystemId();
  }
  if (rowspan > MAX_ROWSPAN) {
    warn("A rowspan attribute has the value " + rowspan
        + ", which exceeds the magic Gecko limit of " + MAX_ROWSPAN
        + ".");
  }
  if (colspan > MAX_COLSPAN) {
    warn("A colspan attribute has the value " + colspan
        + ", which exceeds the magic browser limit of "
        + MAX_COLSPAN + ".");
  }
  if (rowspan == Integer.MAX_VALUE) {
    throw new SAXException(
        "Implementation limit reached. Table row counter overflowed.");
  }
  this.left = 0;
  this.right = colspan;
  this.bottom = (rowspan == 0 ? Integer.MAX_VALUE : rowspan);
  this.headers = headers;
  this.header = header;
}

public void warn(String message) throws SAXException {
  if (errorHandler != null) {
    errorHandler.warning(new SAXParseException(message, publicId,
        systemId, lineNumber, columnNumber));
  }
}
```

```
  public void err(String message) throws SAXException {
    if (errorHandler != null) {
      errorHandler.error(new SAXParseException(message, publicId,
          systemId, lineNumber, columnNumber));
    }
  }

  /** Emit errors if this cell and the argument overlap horizontally. */
  public void errOnHorizontalOverlap(Cell laterCell) throws SAXException {
    if (!((laterCell.right <= left) || (right <= laterCell.left))) {
      this.err("Table cell is overlapped by later table cell.");
      laterCell.err("Table cell overlaps an earlier table cell.");
    }
  }

  public void setPosition(int top, int left) throws SAXException {
    this.left = left;
    this.right += left;
    if (this.right < 1) {
      throw new SAXException(
          "Implementation limit reached. Table column counter overflowed.");
    }
    if (this.bottom != Integer.MAX_VALUE) {
      this.bottom += top;
      if (this.bottom < 1) {
        throw new SAXException(
            "Implementation limit reached. Table row counter overflowed.");
      }
    }
  }

  public boolean shouldBeCulled(int row) { return row >= bottom; }

  public int freeSlot(int potentialSlot) {
    if (potentialSlot < left || potentialSlot >= right) {
      return potentialSlot;
    } else {
      return right;
    }
  }

  public void errIfNotRowspanZero(String rowGroupType) throws SAXException {
    if (this.bottom != Integer.MAX_VALUE) {
      err("Table cell spans past the end of its "
          + (rowGroupType == null ? "implicit row group"
              : "row group established by a \u201C" + rowGroupType + "\u201D element")
          + "; clipped to the end of the row group.");
    }
  }

  public String elementName() { return header ? "th" : "td"; }

  public int getBottom() { return bottom; }
  int getLeft() { return left; }
  int getRight() { return right; }
  public int getColumnNumber() { return columnNumber; }
  public int getLineNumber() { return lineNumber; }
  public String getPublicId() { return publicId; }
  public String getSystemId() { return systemId; }
  public String[] getHeadings() { return headers; }
  public boolean isHeader() { return header; }
}
```

```
final class HorizontalCellComparator implements Comparator<Cell> {

  public static final HorizontalCellComparator THE_INSTANCE =
    new HorizontalCellComparator();

  public final int compare(Cell cell0, Cell cell1) {
    if (cell0.getLeft() < cell1.getLeft()) {
      return -1;
    } else if (cell0.getLeft() > cell1.getLeft()) {
      return 1;
    } else {
      throw new IllegalStateException("Two cells in effect cannot start on the"
        + " same column, so this should never happen!");
    }
  }
}

final class VerticalCellComparator implements Comparator<Cell> {

  public static final VerticalCellComparator THE_INSTANCE =
    new VerticalCellComparator();

  public final int compare(Cell cell0, Cell cell1) {
    if (cell0.getBottom() < cell1.getBottom()) {
      return -1;
    } else if (cell0.getBottom() > cell1.getBottom()) {
      return 1;
    } else {
      if (cell0.getLeft() < cell1.getLeft()) {
        return -1;
      } else if (cell0.getLeft() > cell1.getLeft()) {
        return 1;
      } else {
        throw new IllegalStateException("Two cells in effect cannot start on the"
          + " same column, so this should never happen!");
      }
    }
  }
}
```