Treefic: bridging the gap between XML and plain text

Olivier Aubert <olivier.aubert@liris.cnrs.fr>
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France

Pierre-Antoine Champin <pierre-antoine.champin@liris.cnrs.fr>
Université de Lyon, CNRS
Université Lyon 1, LIRIS, UMR5205, F-69622, France

Abstract

XML has become a *de facto* industry standard for exchanging and managing structured documents and data. As a conceptual model, XML is the core of a set of standard and widely available technologies. As a syntax, on the other hand, XML is not suitable for all applications, being considered too generic or too verbose. In this paper, we propose the use of *specialized textual syntaxes* as a valid alterative, in some contexts, to the XML syntax. With our implemented approach *Treefic*, we show that those syntaxes can be straighforwardly mapped to an XML tree. By bridging that gap, we aim at both advocating textual syntaxes to XML supporters, and promoting XML technologies to its detractors.

1 Introduction

For more than a decade, XML has provided a unifying set of concepts and tools for exchanging, transforming, storing and querying documents and data [10, 16, 13, 28, 6]. Being based on lessons learned from earlier web technologies (such as HTML) and mature standards for document representation (SGML), it has very naturally become an industry standard.

However XML also has its detractors; the main concern raised every now and then is about its verbosity, making it hard to read or edit manually. This verbosity is sometimes simply due to bad design of XML vocabulary [2]. But it is also inherently due to XML aiming at a high level of *genericity*, and some degree of self-containment (well-formed-ness can be checked without any knowledge of the DTD or schema). These properties may not be critically needed in some contexts, and so there are cases where those criticisms are well founded. Despite this fact, XML is often used in situations where alternatives are not even considered, or hastily dismissed under the assumption (sometimes akin to superstition) that XML is intrinsically better than other solutions.

In this work, we advocate the use of *specialised textual syntaxes* (STS) as a valid and viable alternative to XML in many situations, including industrial-grade or high-profile applications. Our argument is that such syntaxes can be parsed as XML trees, allowing to benefit from most of the advantages of XML, while eschewing its flaws. We named our approach Treefic, because it aims at making a tree ("treefication") out of any text.

In the next section we motivate this work by analysing the benefits of using XML, and surveying a number of cases where an alternative syntax was successfully used. Section 3 then describes the core principles of our approach. In Section 4, we extend those principles into the specifications of an implementation presented in Section 5. The next section is dedicated to comparing ours to similar approaches, then we conclude and discuss future work in Section 7.

2 Motivation and case studies

2.1 What XML is and is not

Although it is usually presented as a syntax, XML is a two-sided coins: a syntax and a *conceptual model*. Historically, the abbreviation XML described the syntax [10] while the conceptual model was first introduced as the Document Object Model (DOM) [35]. Other variants of the model underlying XML documents were then proposed in XPath [13], the XML Information Set [16] and the XQuery/XPath data model [21]. This lack of unique reference for the conceptual model may account for the fact that, even nowadays, what people recall from XML is primarily its syntax and the famous angle bracket.

We argue that this is a misconception and that the most important feature of XML is the tree-shaped structure it imposes on documents, rather than the way this structure is serialised into a character string. Indeed, most XML technologies are described as operating at the conceptual level (as defined by one of the documents cited above), independently of the bracket-based syntax: XML-namespaces allow to unambiguously name nodes in the tree [28]; XML-Schemas restrict the shape of the structure for a class of documents [20]; XSL-T specifies how to transform a tree into another tree [28]. Canonical XML [8] distinguishes "logical equivalence" of XML documents which may differ in their "physical representations". The very notion of "physical representation" is a clear sign that the syntax is secondary, while the conceptual structure of the tree is what matters.

Finally, the fact that now and then, alternative syntaxes for XML have been proposed, both outside [5, 32] and inside the W3C [34], confirms our point: XML is not defined by its syntax, but rather by the tree structure encoded by that syntax —or others.

2.2 Succesful non-XML syntaxes

There are a number of examples of popular languages eschewing XML-based syntaxes. The Relax-NG compact syntax [12] is a text-based syntax for representing schemas constraining XML documents. Despite the attempt of the W3C to deprecate SGML-based HTML in favour of XML-based XHTML, HTML 5 [24] has been advocated by a number of web companies, arguing against the verbosity brought by XML-compliance. Even HTML has sometimes been often considered too complicated to be edited by hand, leading to wiki syntaxes [31] and other simplified syntaxes [23, 19]. In the realm of data exchange, JSON [18] has largely dethroned XML in many Web 2.0 applications.

Even some W3C-recommended languages have a text-based alternative syntax: the abstract syntax of OWL [25], the presentation syntax of RIF [7], or the N3 syntax for RDF [3]. Except for the latter, those syntaxes are not considered as exchange syntaxes to be used by machines, but merely for human consumption. It is especially clear in the case of RIF, where the presentation syntax is described both in mathematical English and with a formal grammar, but only the former is normative.

It is interesting to notice that, for all of these languages but JSON, the underlying model either is an XML tree (HTML and wiki syntaxes) or has a standard XML serialisation (RDF, RIF, Relax NG...). This demonstrates that text-based syntax and XML-based syntax can be used in a complementary way, when both considered as the expression of a common model.

2.3 When text matters

When raising the issue of the complexity of XML-based syntaxes, one is often retorted with the "GUI argument": XML-based syntaxes are not meant to be directly visualised or edited by the end-user, but rather hidden behind a specialised and friendly graphical user interface (GUI). Of course this argument holds in a number of situations —think for example of a graphical editor for SVG. However, there is some value in allowing end-users to handle the data directly, and making it easy for them to read and edit it.

In [33], Eric Raymond points out the importance of text-based syntaxes in UNIX history. Text can easily be handled through versatile tools (in the command line) or components (in a graphical environment), which require no further training for the user, and are usually robust and mature. A specialised editor, on the other hand, requires specific coding, debugging and learning. Problems can arise not only in the programm processing the data, but also in the editor or at the interface between them. Readable text-based syntaxes therefore increase transparency of how a processing program works, and make it easy for users (and obviously developers as well) to detect and fix errors. This also fosters adoption and reuse of that program.

In the domain of personal information management [4], knowledge acquisition [29] and querying [14], the importance of controlled languages has also been emphasised. They are usually considered as a good tradeoff towards nat-

ural language interfaces, which are still an open challenge for computer science. Furthermore, text-based interfaces are more suited when the environment is constrained, either by device limitations (*e.g.* mobile devices) or by users' disabilities (*e.g.* screen readers or braille displays).

In fact, the simplification advocated by the GUI argument can as well be applied to the syntax itself. In all the scenarios presented above, one can consider that the specialised textual syntax *is* a user-friendly interface to the underlying model.

2.4 How we managed before XML

Specifying the structure of textual (or non-textual) data has been done long before XML was here. XML is itself a descendant of SGML [22], but the seminal work in that domain is that of Noam Chomsky on generative grammars [11]. Chomsky proposed the notion of *grammar* to capture the structural constraints of a particular language. A grammar is described as a set of *production rules*. Depending on the kind of rules one is allowed to write, Chomsky distinguished four types of grammars of decreasing complexity, from type 0 (unconstrained) to type 3 (regular grammar). While type 0 and type 1 grammars need a full-fledged Turing machine to be checked, type 2 or *context free* grammars (CFG) only need a stack machine, and type 3 or *regular grammars* only need a finite state automaton. The last two are interesting from a computer science perspective, as they require less complex algorithms.

Regular grammars have been popularised by *regular expressions* which are character strings representing a regular grammar in a very compact way. They have been normalised by POSIX, but extensions introduced by the Perl programming language have now become a *de facto* standard. Note that some of these extensions bring features from *contextual* (type 1) grammars (a feature that we will use in Section 3). Because of their compact syntax, regular expressions are not well suited to describe complex formats, but are rather used to check the structure of relatively short strings or mine into textual data.

For describing more complex formats, context-free grammars (CFG) have also been widely used in the pre-XML era (and after). They offer a good trade-off, being at the same time quite expressive, (relatively) unexpensive to parse, and (relatively) easy to implement. They have been especially used under the Backus-Naur Form (BNF), proposed by those authors in [1], or one of its variants, e.g. [17, 26]. Another popular use of CFG is fostered by tools like `yacc` [27] and its successors. Those tools are meant to ease the programming of parsers. They provide high-level programming constructs allowing to express the rules of the grammar in an abstract way, and automatically convert them to operational code.

3 Core principles of Treefic

Following a long tradition (see Section 2.4), we propose to use context-free grammars to describe the structure of an specialised textual syntaxes. Such a grammar is defined as a set of production rules. The body of a rule is a sequence of symbols, which can be *terminal* (*i.e.* symbols appearing in the text) or *non-terminal*. Each non-terminal symbol appears as the head of one or several rules, and one of them (usually the head of the first rule) is the *initial* symbol. A text matches the CFG if and only if, starting with the initial symbol and recursively replacing (deriving) non-terminal symbols with the body of one of their rules, one can build a sequence of terminal symbols that equals the text. Another way to look at it is that the text can be *abstracted* to the initial symbol by recursively replacing a part of the text matching the body of a rule by the head of that rule, until one gets the initial symbol only.

Parsing a text according to a CFG amounts to building an ordered *parse tree*, where leaves are labelled with a terminal symbol, intermediate nodes are labelled with a non-terminal symbol and the root is labelled with the initial symbol. For each node with children, the children's labels correspond to one of the rules having the parent node's label as its head.

The idea of Treefic originated with the observation that such a parse tree can be straightforwardly represented as an XML tree. Indeed, only a subset of XML is needed: element nodes (to represent intermediate nodes, labelled with non-terminal symbols), and text nodes (to represent leaves, labelled with terminal symbols). See for example Figure 1.

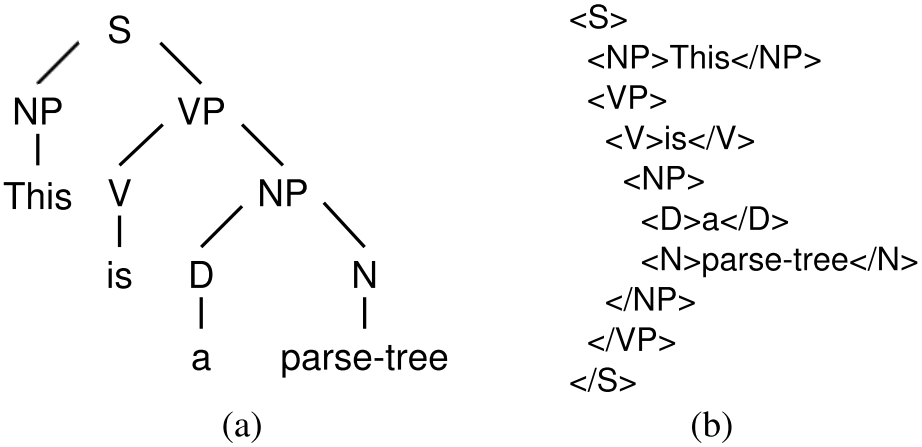

Figure 1: A typical parse-tree (a) and its straightforward XML representation (b)

So the minimal work flow bridging the gap between XML and specialised textual syntaxes is the one described in Figure 2. Provided with a document as text and a CFG, it produces the parse tree of the document as an XML tree.

Once this is done, standard XML tools can be used to process it: transform it, check it against a schema, query it... Transforming it back to the text-based syntax is one of the many applications offered by XSL-T stylesheets.

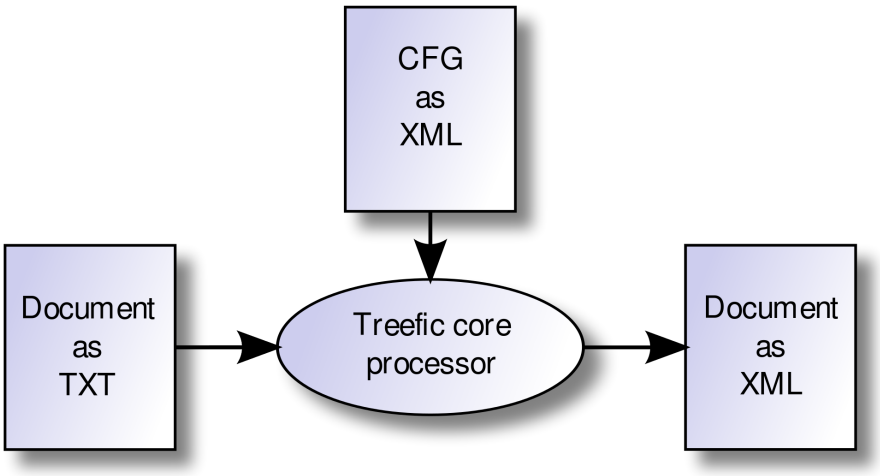

Figure 2: The core work flow of the Treefic approach

Of course, one needs to be able to describe the CFG. For this, we propose an XML-based syntax described by a Relax-NG schema[1]. This follows our philosophy of relying on standard XML technologies: a CFG expressed in our syntax can easily be checked against the schema, parsed with off-the-shelf libraries, *etc.*

It is worth mentioning that our language for describing CFG is not limited to the minimal construct described above. Following the BNF and its variant, we provide syntactic sugar as additional constructs that do not extend the expressive power of the grammars but make them easier to write and read: alternative, repetitions and optional elements.

Terminal symbols in our language are unicode strings or regular expressions. The use of unicode is a must-have to be able to handle international textual syntaxes. The use of regular expressions, however, deserves some explanation, as they could in principle be expressed by a set of rules in the CFG (recall that regular grammars are a subset of context-free grammars). First, doing so would make the grammar much more complicated, and may introduce additional non-terminal symbols, while regular expressions represent a more compact and widely known alternative. Second, it is common to distinguish two levels in a CFG: the lexical level (describing the "words"), and the syntactical level (describing the "sentences"). The former most often relies on regular grammars, while context-free features are usually required by the latter. Thus, by embed-

[1] available at http://liris.cnrs.fr/silex/2010/treefic

ding the lexical "rules" inside regular expressions, the author of a grammar can clearly distinguish both levels. Third, regular expression engines implement limited support for *contextual* features (look-ahead and look-behind expressions). Those features therefore extend (locally) the expressive power of our grammars, making regular expression slightly more than syntactic sugar.

We illustrate this principles with a simple grammar for attribute-value pairs. An example document is given in Figure 3. The CFG is given in Figure 4. The resulting XML tree is given in Figure 5.

```
name       =Doe
first-name =John
email      =john@doe.org
address    =123 Second Street
```

Figure 3: Example document containing attribute-value pairs

```
<?xml version="1.0" encoding="utf-8"?>
<grammar
 xmlns="http://liris.cnrs.fr/silex/2010/treefic#">
  <rule name="av-pairs">
    <repetition min="0">
      <rule ref="pair"/></repetition></rule>
  <rule name="pair">
    <sequence>
      <rule ref="_"/>
      <rule ref="key"/>
      <rule ref="equal"/>
      <rule ref="value"/>
      <rule ref="eol"/></sequence></rule>
  <rule name="key">
    <regexp flags="i">[a-z_][a-z0-9_.-]*</regexp></rule>
  <rule name="equal">
    <sequence>
      <rule ref="_"/>
      <terminal>=</terminal>
      <rule ref="_"/></sequence></rule>
  <rule name="value">
    <regexp flags="">[^\n]*</regexp></rule>
  <rule name="_">
    <regexp flags="">[ \t]*</regexp></rule>
  <rule name="eol">
    <regexp flags="">\n+</regexp></rule>
</grammar>
```

Figure 4: A CFG for attribute-value pairs

```
<av-pairs><pair><_/><key>name</key>
  <equal><_>        </_>=<_/></equal>
  <value>Doe</value><eol>
</eol></pair>
<pair><_/><key>first-name</key>
  <equal><_> </_>=<_/></equal>
  <value>John</value><eol>
</eol></pair>
<pair><_/><key>email</key>
  <equal><_>       </_>=<_/></equal>
  <value>john@doe.org</value><eol>
</eol></pair>
<pair><_/><key>address</key>
  <equal><_>     </_>=<_/></equal>
  <value>123 Second Street</value><eol>
</eol></pair></av-pairs>
```

Figure 5: The XML tree generated for the example of Figure 3 with the CFG in Figure 4 (indentation added for readability). This is a straightforward dump of the raw parse tree.

4 Advanced principles of Treefic

4.1 Textual grammars

Of course, while we argued about the interest to map our CFG to the conceptual model of XML, we would *also* like to be able to represent them in a more textual and compact way, in the fashion of BNF or its variants. Fortunately, the core approach described in the previous section provides the only building block we need to achieve that. All we need is to provide a *meta-grammar* describing how the textual representation of the CFG can be mapped to the standard XML representation expected by the core parser. This meta-grammar is nothing more than a specific CFG described using the same XML-based syntax. This leads us to the extended work flow described in Figure 6.

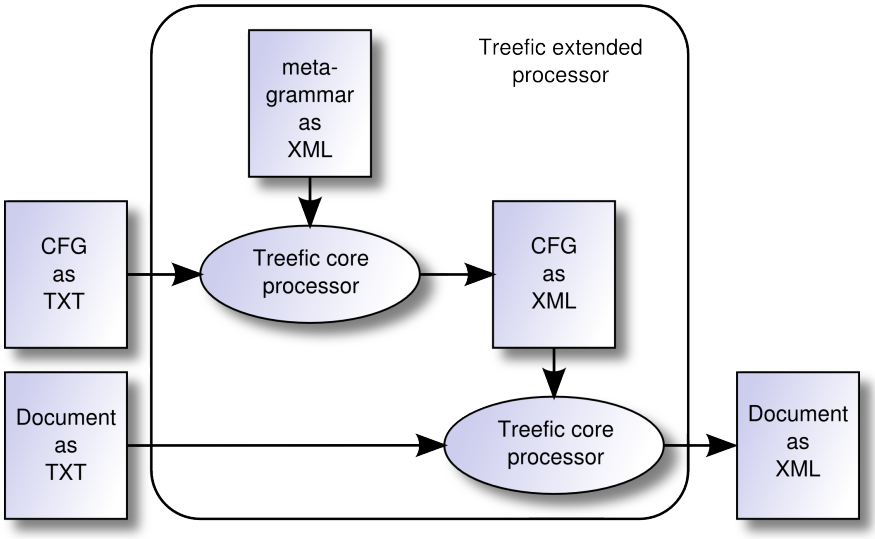

Figure 6: The extended workflow of the Treefic approach

The immediate benefit, as advocated in Section 2, is that CFG are much easier to read and write. As an illustration, Figure 7 is the textual version of the CFG given in Figure 4. But besides convenience, this extended framework allows to immediately "treefy" any text whose structure is described by a CFG. The only requirement is a meta-grammar for the BNF variant used to specify that CFG, which is a reasonable requirement as only a few of those variants are commonly used.

```
av-pairs ::= pair*
    pair ::= _ key equal value eol
     key ::= /[a-z_][a-z0-9_.-]*/i
   equal ::= _ "=" _
   value ::= /[^\n]*/
       _ ::= /[ \t]*/
     eol ::= /\n+/
```

Figure 7: The textual version of the CFG given in Figure 4

4.2 Decorators

Once textual data has been "lifted" into an XML tree, one can use standard XML technologies to process it. Such processing will typically include a transformation from the structure induced by the CFG parse tree into a more common schema. Indeed, the structure of the XML tree in Figure 5 is not exactly how

one would naturally represent attribute-value pairs in XML. One would rather expect representation like the one in Figure 9.

In principle, that transformation can of course be performed with XSL-T. But in practice, we found it tedious to use an XSL stylesheet for the sole purpose of cleaning the parse tree into a more usable XML tree. One ends up copying and adapting the same XSL-T rules, as the number of operations required for the cleaning is relatively limited, like:

- ignoring some terminal symbols only used as delimiters (as the equal symbol in our attribute-value pairs);
- considering a non-terminal node as an attribute rather than an element (*e.g.* the `key` part of a pair becoming an attribute of the parent `pair` element);
- collapsing a node with its parent (*e.g.* the text of `value` node becoming the text of the `pair` element);
- renaming a node (useful when the target XML structure uses the same name in different *contexts*, requiring different non-terminal symbols in a context-free grammar);
- adding an attribute with a fixed value (useful when the target XML structure uses an attribute rather than an element name to distinguish between different cases; *e.g.* the `version` attribute of the `av-pairs` element);
- changing the namespace of all or one node(s).

As those operations apply to specific non-terminal symbols, and as they are tightly linked to the rule corresponding to that symbol, it seemed a better practice to specify the cleaning operation side-to-side with the rule itself. We therefore extended our CFG language with the notion of *decorator*. Decorators specify a special processing for the nodes generated by the decorated non-terminal symbol (rule decorator), or for all nodes (grammar decorator). Figure 8 gives the decorated version of the CFG in Figure 7, which results in the XML tree of Figure 9. Note that rule decorators start with `@` and appear *before* the decorated rule; grammar decorators start with `@@` and appear at the beginning of the grammar. Of course, our XML syntax has corresponding constructs.

It is important to understand that decorators are in no way meant to provide a general purpose transformation language. This would contradict our goal to rely as much as possible on standard technologies. They are designed as a lightweight convenience for cleaning the parse tree into a more useful XML tree, and as such are kept limited to the small set of operations listed above. The only drawback, compared to the use of XSL-T or another existing transformation language, is that the raw parse tree can not be generated anymore (as the cleaning instructions are integrated into the grammar). However, if the parse tree is valuable in itself, then it needs no cleaning, and then the grammar should not be decorated (but an external stylesheet should be used instead).

```
@@xmlns=http://example.com/ns

@add-attribute version="1.0"
av-pairs ::= pair*

pair ::= _ key equal value eol

@as-attribute
key ::= /[a-z_][a-z0-9_.-]*/i

@prune
equal ::= _ "=" _

@no-tag
value ::= /[^\n]*/

@prune
_ ::= /[ \t]*/

@prune
eol ::= /\n+/
```

Figure 8: A decorated CFG for attribute-value pairs

```
<av-pairs version="1.0"
 xmlns="http://example.com/ns">
  <pair key="name">Doe</pair>
  <pair key="first-name">John</pair>
  <pair key="email">john@doe.org</pair>
  <pair key="address">123 Second Street</pair>
</av-pairs>
```

Figure 9: The clean XML tree generated for the example of Figure 3 with the decorated CFG in Figure 8 (indentation added for readability)

5 Implementation

Treefic has been implemented as an open-source python library and command line tool[2]. When used as a library, it produces an element-tree, a standard data structure used in python to represent XML trees. When used on the command line, it writes the resulting XML on its standard output, making it easy to combine with other XML tools such as an XSL-T processor.

[2] http://liris.cnrs.fr/~pchampin/treefic/

The command line tool tries to automatically guess whether the grammar is expressed using our XML language (parsed directly as in Figure 2) or a textual syntax (parsed using a meta-grammar as in Figure 6). It is currently shipped with only one meta-grammar. Although it would be useful in the future, as suggested in Section 4.1, to provide meta-grammars for different variants of BNF such as [17, 26], we started with a different approach. The main rationale was that we needed specific features (regular expressions, decorators) that none of these variants provided.

The textual CFG syntax supported by the current implementation aims to support as much as possible of the features of the most common BNF variants. For example, the separator between the head and the body of a rule can be "::=", ":=", "=", "->" or "→". The operator for alternative can be either "|" or "!". Element in a sequence can be separated by "," or simply juxtaposed. As some variants have mutually incompatible features, it is not possible to have a syntax which would be a strict superset of all of them. However, only a few modifications are required to convert existing BNF to our expressive syntax. We have successfully experimented this with the ABNF of [18]; the resulting decorating CFG is available at the Treefic website[2].

6 Related work

We have presented in Section 2 a number of work pursuing a goal similar to ours. Alternative syntaxes for XML [5, 32] are significantly different, however, as they try to be as generic as the original XML syntax. Even if they manage to be less verbose (for example by not repeating the name of a tag when closing it), they are still not as flexible as the *specialised* textual syntaxes (STS) that we advocate.

We also mentioned a number of existing STS in Section 2.2. Those syntaxes usually rely on *ad-hoc* implementations to be converted to the corresponding XML tree. Since they are typically specified by a BNF variant, they could benefit from Treefic: the conversion process can be described in a declarative way, either by decorating the grammar (see Section 4.2) or by specifying an XSL-T transformation from their parse tree to the target XML tree.

A hybrid approach is taken by the Regular Fragmentation [30] project. This project uses XML syntaxes, but uses regular expressions to describe the hidden structure of textual contents (for example a date). This allows a parser to "see" a subtree of elements while the user would only see text. However, the user still has to edit that text inside an XML document.

The work most similar to ours is XSugar[3], described in [9]. The goal of this approach is to provide XML syntaxes with an alternative STS; the main focus of the authors is to guarantee the *reversibility* of the transformation between the two syntaxes. To achieve this, they propose a specific language describing at the same time the CFG, the structure of the XML tree, and the equivalence relations between them. This complex specification relies on a new underlying model (the unifying syntax tree), which enables the translation between the two

syntaxes.

XSugar is theoretically more advanced than Treefic, as it guarantees that the the XML and textual syntaxes can unambiguously be converted to each other. On the other hand, it relies on a more complex language, playing altogether the role of an XML schema, a decorated CFG, and XSL-T stylesheet for transforming both ways. We believe that, by relying on existing technologies, and allowing to reuse existing BNF grammars rather than having to specify them from scratch, Treefic is prone to wider adoption, even if it provides less theoretical guarantees.

7 Conclusion and perspectives

In this paper, we advocated for the use of *specialised textual syntaxes* as a valid alternative to XML-based syntaxes. We furthermore proposed that those syntaxes can be considered as expressions of XML trees, and can as such benefit from standard XML technologies.

We presented the Treefic approach, consisting of the following contributions:

- an XML-based syntax for representing context-free grammars (CFG);
- an implemented processor for converting the parse tree of a CFG to an XML tree, optionally using *decorators* in the CFG to clean the generated tree;
- a meta-grammar allowing to use BNF-like grammars instead of their XML counterpart.

There are several perspectives opened by this work. We are currently working on providing stylesheets to automatically generate the RelaxNG schema corresponding to a decorated CFG. It would also be interesting to generate the XSL stylesheet transforming the XML tree back to the textual syntax; since it may not exist for every CFG, we need to investigate the theoretical work of [9].

As we already mentionned, we consider providing additional meta-grammars corresponding exactly to existing BNF variants. This would make it even easier to use existing STS with Treefic, but with limitations as those syntaxes do not support all the specific features offered by Treefic. A way to circumvent those limitations would be to hide Treefic features (such as decorator) inside comments in the host syntax.

Another interesting perspective would be to integrate Treefic with GRDDL [15]. The goal of GRDDL is to glean semantic descriptions from HTML and XML-based languages. With Treefic, the scope of GRDDL could be enlarged to other syntaxes, and the Semantic Web in general could benefit from information currently buried in textual syntaxes.

[3] http://www.brics.dk/xsugar/

References

[1] J. W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich ACM-GAMM conference. page pp. 125–132. UNESCO, 1959.

[2] R. Berjon. Designing XML/Web languages: A review of common mistakes. Prague, CZ, Mar. 2009.

[3] T. Berners-Lee. Notation3 (N3) a readable RDF syntax. http://www.w3.org/DesignIssues/Notation3, Mar. 2006.

[4] M. Bernstein, M. V. Kleek, M. Schraefel, and D. R. Krager. Evolution and evaluation of an information scrap manager. Florence, Italy, Apr. 2008.

[5] V. Birk. YML - why a markup language?! http://fdik.org/yml/, Aug. 2010.

[6] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML query language. W3C recommendation, W3C, Jan. 2007.

[7] H. Boley, G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynolds. RIF core dialect. W3C proposed recommendation, W3C, Oct. 2009. http://www.w3.org/TR/rif-core/.

[8] J. Boyer. Canonical XML version 1.0. W3C recommendation, W3C, Mar. 2001.

[9] C. Brabrand, A. M\oller, and M. I. Schwartzbach. Dual syntax for XML languages. *Information Systems*, 33(4-5):385–406, 2008.

[10] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. W3C recommendation, W3C, Feb. 1998.

[11] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[12] J. Clark. RELAX NG compact syntax. OASIS committee specification, OASIS, Nov. 2002.

[13] J. Clark and S. DeRose. XML path language (XPath). W3C recommendation, W3C, Nov. 1999.

[14] C. Comparot, O. Haemmerlé, and N. Hernandez. An easy way of expressing conceptual graph queries from keywords and query patterns. In *Conceptual Structures: From Information to Intelligence*, pages 84–96, Kuching, Sarawak, Malaysia, July 2010.

[15] D. Connolly. Gleaning resource descriptions from dialects of languages (GRDDL). W3C recommendation, W3C, Sept. 2007.

[16] J. Cowan and R. Tobin. XML information set (Second edition). W3C recommendation, W3C, Feb. 2004.

[17] D. Crocker and P. Overell. Augmented BNF for syntax specifications: ABNF. RFC 5234, IETF, Jan. 2008.

[18] D. Crockford. The application/json media type for JavaScript object notation (JSON). RFC 4627, IETF, July 2006.

[19] Docutils. reStructuredText. http://docutils.sourceforge.net/rst.html, Sept. 2010.

[20] D. C. Fallside and P. Walmsley. XML schema part 0: Primer second edition. W3C recommendation, W3C, Oct. 2004.

[21] M. Fernández, A. Malhotra, J. Marsh, M. Nagy, and N. Walsh. XQuery 1.0 and XPath 2.0 data model (XDM). W3C recommendation, W3C, Jan. 2007.

[22] C. F. Goldfarb and Y. Rubinsky. *The SGML handbook*. Oxford University Press, USA, 1990.

[23] J. Gruber. Markdown. http://daringfireball.net/projects/markdown/, 2010.

[24] I. Hickson. HTML5: a vocabulary and associated APIs for HTML and XHTML. W3C working draft, W3C, Oct. 2010.

[25] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 web ontology language primer. W3C recommendation, W3C, Oct. 2009.

[26] ISO. Information technology - syntactic metalanguage - extended BNF. International Standard 14977, ISO, Dec. 1996.

[27] S. C. Johnson. YACC: yet another Compiler-Compiler. In *Unix Programmer's Manual*, volume 2b. 1979.

[28] M. Kay. XSL transformations (XSLT) version 2.0. W3C recommendation, W3C, Jan. 2007.

[29] T. Kuhn. AceWiki: a natural and expressive semantic wiki. In *Proceedings of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges*. CEUR Workshop Proceedings, 2008.

[30] S. S. Laurent. Regular fragmentations. http://simonstl.com/projects/fragment/, 2001.

[31] B. Leuf and W. Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet.* Addison-Wesley Professional, Apr. 2001.

[32] S. McGrath. Pyxie, Mar. 2000.

[33] E. S. Raymond. *The art of unix programming.* Pearson Education, 2003.

[34] J. Schneider and T. Kamiya. Efficient XML interchange (EXI) format 1.0. W3C candidate recommendation, W3C, Dec. 2009.

[35] L. Wood. Document object model (DOM) level 1 specification. W3C recommendation, W3C, Oct. 1998.