From templates to schemas: bridging the gap between free editing and safe data processing

Vincent Quint, Cécile Roisin, Stéphane Sire, Christine Vanoirbeek

From Templates to Schemas: Bridging the Gap Between Free Editing and Safe Data Processing

Vincent Quint
Cécile Roisin
INRIA, Grenoble, France
{vincent.quint, cecile.roisin}@inria.fr

Stéphane Sire
Christine Vanoirbeek
EPFL, Lausanne, Switzerland
{firstname.lastname}@epfl.ch

ABSTRACT

In this paper we present tools that provide an easy way to edit XML content directly on the web, with the usual benefit of valid XML content. These tools make it possible to create content targeted for lightweight web applications. Our approach uses (1) the XTiger template language, (2) the AXEL Javascript library for authoring structured XML content and (3) XSLT transformations for generating XML schemas against which the XML content can be validated. Template-driven editing allows any web user to easily enter content while schemas make sure applications can safely process this content.

Categories and Subject Descriptors

I.7 [**Document and Text Processing**]: Document Preparation—*Languages and systems, Markup languages*

General Terms

Design, Experimentation, Languages

Keywords

Document authoring, web editing, document language, XML

1. INTRODUCTION

Several intermediate levels of document structure have been proposed to bridge the gap between HTML tag soup and highly structured XML: microformats, semantic HTML, document templates, RDFa, to name only the most popular technologies. They pursue different goals and provide different benefits regarding structure improvement, but all of them aim at rigorously structuring some information in HTML documents. This structured information can then be extracted safely from its host document, and it can be further processed to use and re-use in many different ways the content of HTML documents, which otherwise could only be displayed by web browsers. However, most of these technologies apply only to some specific parts of a document and they still lack tools that could allow average web users to easily create structured information.

In this paper, we explore another approach which aims at producing full XML documents and data on the web while allowing authors to work in a familiar HTML environment, their favorite web browser. The goal is to bring the power of the usual XML processing chain to new application domains such as Web 2.0 applications, but based on valid XML content. Such web sites cannot afford complex authoring processes involving multiple languages and skilled specialists. Usually these sites rely heavily on Rich Text Editors or on Wiki-based input that restrict information reuse and computability. Instead, we are proposing a lightweight approach based on document templates for XML document authoring. However, to fully take advantage of XML structures, this simplification must not compromise the quality of the created documents and data. This has lead us to engineer a method to generate schemas from the document templates.

The next section briefly introduces the XTiger template language from which this work has begun. Section 3 presents the browser-based editor that was built to exploit this template language, originally designed to create XHTML documents. Section 4 explains how the language was extended to allow the editor to generate XML documents. The issue of the automatic generation of XML Schemas is discussed in section 5. Section 6 summarizes the advantages brought by these tools and compares them to related work.

2. TEMPLATES FOR XHTML

The starting point is a template language called XTiger [2] that was created for authoring different kinds of XHTML documents that are supposed to follow a given model. With XTiger, a template is a skeleton XHTML document that contains elements in the XTiger language (as both XHTML and XTiger are XML languages, they can easily be mixed). Each XTiger element expresses constraints about the local XHTML structure and its possible evolutions during editing.

Typical XTiger elements are:

- `xt:component` which defines a component, i.e. a piece of XHTML structure (possibly with XTiger elements) that can be inserted in some places in the document.
- `xt:use` which indicates, with its `types` attribute, what component(s) or basic content (text for instance), may be inserted at its location.
- `xt:repeat` which indicates that the structure it includes (XHTML with XTiger elements) may be repeated several times at its location.

DocEng2010, September 21–24, 2010, Manchester, United Kingdom.

Figure 1 shows the XTiger definition of component `authorComp`, which is used in the Article template (the template used for producing this paper, see Figure 2).

```
<xt:component name="authorComp">
  <p class="vcard">
    <xt:repeat minOccurs="1">
      <span class="fn">
        <xt:use types="text" label="name">
                Author name</xt:use>
      </span><br />
    </xt:repeat>
    <xt:repeat minOccurs="1" label="address">
      <span class="addr">
        <xt:use types="text" label="line">
                Address line</xt:use>
      </span><br />
    </xt:repeat>
    <span class="email">
      <xt:use types="text" label="email">
              email</xt:use>
    </span>
  </p>
</xt:component>
```

Figure 1: The author component template

This very simple language was first implemented in the Amaya [9] web editor. The XHTML editor in Amaya was extended to be driven by XTiger templates, i.e. to interpret and follow the hints given by the XTiger elements about the XHTML structure. It allowed authors to produce more easily better structured documents, but it was available only in Amaya and it could only create XHTML documents.

3. TEMPLATE-DRIVEN EDITING IN THE BROWSER

The next step was to make XTiger template-driven editing also usable on the widely available platform offered by web browsers. The goal was to make it easier for any web user to create content on line, by relieving the burden of installing new software and learning how to use a new editor. This was achieved by developing a Javascript library that runs in the browser (in any Javascript-enabled browser) and implements editing functions that follow XTiger templates.

The Adaptable XML Editing Library (AXEL) that resulted from this work dynamically generates an interactive editing application from a source template inside any web page [8]. The source template can be the web page itself, the content of an internal frame, or a file loaded with an Ajax call. The library also provides some functions to linearize the edited document into a string representation at any time. Thus it is easy to send it back to a server.

The design of the editing user interface aims at minimizing user's mouse movements. Hence, there is no menu bar at the top of the document, but only contextual menus that follow the document structure to give access to the editing functions (Figure 2). The system displays '+' and '-' icons next to the document parts that can be repeated as defined by the template. Similarly, the system displays a pop-up menu next to the document parts that can be selected among a choice of different component types. Finally, the system dynamically turns the text fragments corresponding to primitive editing components into text entry fields when the user clicks them.

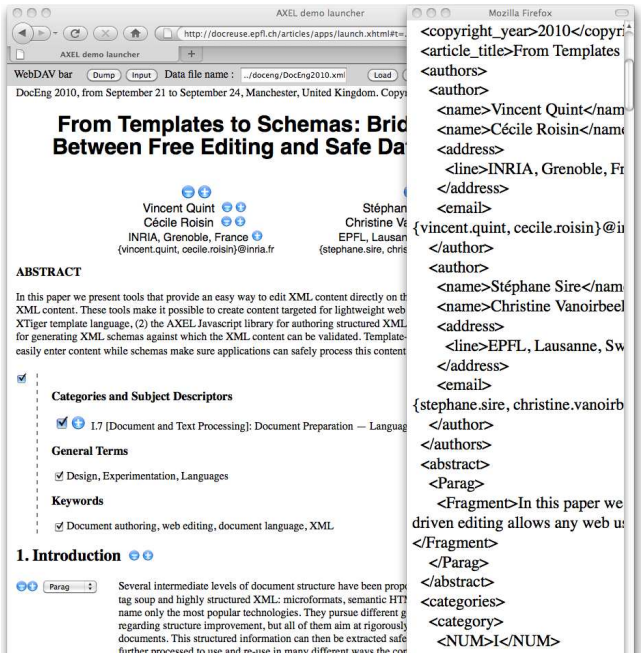

Figure 2: Editing an Article with AXEL

The user interface is highly customizable as a consequence of running inside a web browser. It is possible for instance to create variations of the editing style with CSS rules and Javascript functions. It is even possible to extend the primitive editing components for entering text with new editing components written in Javascript, or to create highly specialized editing component types to enter complex data types, such as a photo uploader. In this way AXEL differs from Amaya, as it allows to create domain specific document templates with document specific user interaction models.

The idea of implementing an editor in the web browser is not new, but AXEL has a few original features. Google Docs, for instance, offers an editor that runs in the browser. It also proposes a large library of templates, like most word processors do, but these templates are just ordinary documents that authors can use as a starting point and as a source of inspiration. They do not provide any guarantee about the structure and the content of the final result. With XTiger, the final document always conforms to the structural constraints expressed in the template. When the information contained in the document has to be further processed, this makes a difference.

4. GENERATING XML DOCUMENTS

An extension to the XTiger language was necessary to get one step further. This is a syntactically minor extension, but it brings a significant advantage by allowing the editor to map the XHTML document being edited to a target XML document. The `label` attribute was added for that purpose to most XTiger elements to indicate the name of a corresponding XML element to be created when the DOM tree built by AXEL is linearized in XML. This process is similar in intention with XForms for HTML [1].

The `label` attribute can be set on `xt:use` and `xt:repeat` elements to map their content model to a target XML element whose name is defined by the label. For instance, as the template used in Figure 2 declares the component inclusion `<xt:use label="author" types="authorComp"/>`, then the linearized content of this component will be placed in an XML sub-tree tagged `author`. The right hand part of Figure 2 shows the XML code generated for this article. The mapping algorithm is recursive. It stops at the inclusion of a few primitive component types, such as `text`, that produce only unparsed content which is entered with browser text entry fields as explained in the previous section.

We have also written an XML loading algorithm together with the linearization algorithm. It uses the same `label` information to load XML content into a compatible template and generates the corresponding XHTML document.

To some extent the use of a document template to produce a target XML content model is close to more data-centric form-based applications. For instance, this could be compared to Google Forms where the document template is a form document, and where the responses are collected inside a spreadsheet. However the range of XML documents reachable with AXEL goes beyond tables and columns and allows also document-centric data to be edited.

There are other browser-based editors for XML documents such as XOpus. In some respects, these tools are like many other XML editors: to edit a document authors must provide an XML Schema, a style sheet, and an XSLT transformation. With AXEL, as soon as an XTiger template is available, well structured XML documents can be edited. Although XML Schemas are available for some common applications, users with specific needs have to create their own schemas, style sheets and transformations for most XML editors. This is a different task than developing an XTiger template, for which users have just to create the XHTML skeleton of a typical document and to include a few XTiger elements that indicate how this skeleton may be changed. In addition, a CSS style sheet can be defined to set the graphical aspect of documents, but this is not mandatory.

5. GENERATING XML SCHEMAS

The editor outputs XML files, but the only claim we can make is that these files are well-formed and follow the constraints expressed by an XTiger template. That is not enough to fully exploit the XML structure. We need a schema against which the XML document could be validated and that can be used by XML applications that require a schema to process XML data.

The semantics of XTiger may be defined as an extension of XML Document Type Definitions (DTD), the extension consisting in more precise occurrence indicators (for this purpose, XTiger borrows the `minOccurs` and `maxOccurs` attributes from XML Schema). The semantics of XTiger is then a subset of the semantics of XML Schema. As a consequence, the constraints defined by an XTiger template can be expressed in the XML Schema language, and the translation of an XTiger template into an XML Schema can be done automatically.

We have developed an XSLT transformation to perform this translation. It works in two steps:

- First, the template is stripped from all XHTML code. The result is a pure XTiger file that contains only structure constraints and the names (`label` attribute) of the XML elements to be generated (Figure 3).
- The second step transforms this structure definition into an equivalent XML Schema (Figure 4).

```
<xt:component name="authorComp">
  <xt:repeat minOccurs="1">
    <xt:use types="text" label="name"/>
  </xt:repeat>
  <xt:repeat minOccurs="1" label="address">
    <xt:use types="text" label="line"/>
  </xt:repeat>
  <xt:use types="string" label="email"/>
</xt:component>
```

Figure 3: Stripped template

```
<xsd:complexType name="authorComp">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"
                 maxOccurs="unbounded"/>
    <xsd:element name="address">
      <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="line" type="xsd:string"
                      maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="email" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4: Schema for the author component

The resulting XML Schema defines the structure of the XML documents generated by AXEL when using a given XTiger template. Document templates designed for AXEL can now be used in any XML process that benefits from a schema, such as writing transformations or queries. Similarly, the documents edited with AXEL can now be used in any XML application that requires a schema, such as storing data in an XML database that enforces validation.

The transformation was validated on very different XTiger templates, including those available in [7]: they range from form-based documents, such as a restaurant menu or a resume, to more complex documents, such as scientific and technical documentation with hundreds of elements and a significant tree depth. For each template, several XML document instances were validated against the XML Schema produced by the transformation.

6. APPLICABILITY AND FUTURE WORK

Existing XML document production systems are usually employed in areas where well-established XML schemas have been defined. In this typical setting, several users play different roles: computer-skilled persons create schemas, style sheets, and transformations, whereas content producers use

the authoring tool. This process is basically the same whatever the kind of the editing system: stand-alone or browser-embedded. Adapting this process to a new document type is both heavy and time consuming. With the XTiger template language and the AXEL library, the process becomes simpler and faster. Moreover the transformation tool presented in the previous section provides a schema for every XTiger template. It has been developed once and it is now available for all XTiger templates, so that no more XSLT transformations are necessary. As a consequence, XML documents can be involved in new classes of applications where simpler approaches such as rich text editors or wiki text entry were privileged before. In particular, the schema has not to be completely defined before starting to create XML content, and a more flexible and incremental process is possible as it is the case with wikis [4].

This raises two classes of problems that we have encountered in real applications.

The first class of problems is to easily check whether a document template is compatible with an XML schema. We have encountered this issue when using document templates to create an editing application based on AXEL for complex specification documents that were initially defined by a XML schema and that were converted manually from Word to XML. Word was imposed by the specification authors as they were non XML savvy. In that case it was crucial that the XML documents edited with the XTiger template be valid, as they were further processed after editing. The ability to convert the XTiger template into an XML schema greatly simplifies comparison with the original XML schema. When a literal comparison does not make sense, we plan to use the tool presented in [5] to make sure that both schemas define exactly the same type of XML documents.

At some point we could also have imagined to go one step further and to generate the document template from the XML schema, but this kind of transformation can work only in the opposite direction. Indeed, given the many possibilities offered by the XML Schema language that have no equivalent in XTiger, there is no hope for an automatic transformation of a schema into a template. In addition, a template contains XHTML code that could not be created from a schema.

The second class of problems is to check whether a modified version of a template is backward-compatible with the original template (can the new template accept XML content originating from the original template?), and in case of discrepancy to automatically generate some transformations for migrating the corresponding data. This is a very important feature for allowing Web 2.0 applications to take advantage of XML processing chains, because these applications usually have evolving content models, at least until the web site finds its audience. Moreover, developers of these sites are accustomed to powerful MVC frameworks such as Ruby on Rails, where they directly write data migration paths with object relational mapping languages that generate the migrations for them. Convincing these developers to adopt XML languages requires equivalent migration tools for XML data. The capability to generate an XML schema for every template as presented in this paper is a step in that direction, that still needs to be complemented with versioning and schema comparison tools [6], [3], [5] to come up with similar support.

7. CONCLUSION

The approach presented in this paper provides a simple and safe way to produce XML content. Well structured XML documents can be edited directly on the web, using only a browser. As the corresponding XML schemas are available, these documents can be used in many different ways. Although the expressive power of XTiger is not at the same level as in the XML Schema language, its coverage is large enough to cope with a wide variety of XML contents. In addition, users do not need to learn XML and its associated technologies such as XSLT or XML Schema. Knowing a bit of HTML and learning how to use the six XTiger elements is enough to develop an environment for entering XML content on the web. This lightweight process enables new kinds of applications where web users can provide their own information in a well structured way.

8. ACKNOWLEDGMENTS

Early work on this project has been initiated in the framework of the PALETTE Integrated Project supported by the IST programme of the European Commission (DG Information Society and Media, no. 028038). Further developments of the XTiger language are currently supported by the Innovation Promotion Agency of Switzerland under grant No 10813.1 PFES-ES, a project in collaboration with MadeinLocal.com (www.madeinlocal.com).

9. REFERENCES

[1] J. Boyer. *XForms for HTML*. W3C Working Draft, http://www.w3.org/TR/XForms-for-HTML/, 19 December 2008.

[2] F. Campoy-Flores, V. Quint, and I. Vatton. Templates, microformats and structured editing. In D. Brailsford, editor, *Proceedings of the 2006 ACM Symposium on Document Engineering, DocEng 2006*, pages 188–197. ACM Press, Oct. 2006.

[3] A. B. Coates and D. Dui. "full impact" schema differencing. In *Proceedings of XML Prague 2010*, pages 65–86. Institute for Theoretical Computer Science, Mar. 2010.

[4] A. Di Iorio, F. Vitali, and S. Zacchiroli. Wiki content templating. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 615–624, New York, NY, USA, 2008. ACM.

[5] N. Layaïda and P. Genevès. Debugging standard document formats. In M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, pages 1269–1272. ACM, Apr. 2010.

[6] I. Mlýnková. Similarity of XML schema definitions. In *DocEng '08: Proceeding of the eighth ACM symposium on Document engineering*, pages 187–190, New York, NY, USA, 2008. ACM.

[7] S. Sire. *XTiger XML editing with AXEL demos*. http://media.epfl.ch/Templates/.

[8] S. Sire, C. Vanoirbeek, V. Quint, and C. Roisin. Authoring XML all the time, everywhere and by everyone. In *Proceedings of XML Prague 2010*, pages 125–149. Institute for Theoretical Computer Science, Mar. 2010.

[9] I. Vatton. *Amaya*. http://www.w3.org/Amaya/.