USING XSD INHERITANCE FOR XML-DRIVEN PRESENTATION LAYOUT IN WWW, ITV AND MULTI-CHANNEL PUBLISHING

Facilitating Large-Scale Publishing

Daniel Farré Giribet
Department of Technology, Pompeu Fabra University, Passeig de Circumval·lació, 8, Barcelona, Spain

Keywords: XML, XSD, XML Schema, XHTML, MHP, iTV, GUI, multi-channel publishing, XForms, USIXML, UIML, presentation layout.

Abstract: The usage of abstract XML layout languages has proven an effective technique of agile, adaptable and multi-channel multimedia publishing. This general strategy employs an XML definition file specifying graphical widget layout and relative disposition regardless of specific presentation medium. This strategy has been successfully applied for WWW, MHP, WAP, PDA and in many media platforms and there exist several existing proposals and implementations. However, not many general formal strategies have been developed to address some XML-based layout language design challenges such as general structure, validation and schema development. Additionally, there are some implementation issues that many such techniques fail to address properly. In this paper, a simple yet powerful general strategy of solving these issues is described: the XSD standard is leveraged to help in designing and validating XML layout languages as well as aiding in the development of GUI layout applications. This strategy is proposed as a practical approach valid in large-scale publishing environments.

1 XML PRESENTATION LANGUAGES

Multiplatform multimedia electronic publishing is one of the key challenges modern Content Management Systems (CMS) need to solve. The same content produced by a given organization should be easily published in any form in any plausible electronic end-user platform: Web browsers, iTV MHP Set-Top-Boxes (STBs), PDAs, WAP cell phones, smartphones, Rich Internet Applications (RIAs), SWT applications, MMS alerts, etc. Moreover, the list of electronic multimedia devices is constantly being expanded, so any publishing system needs a solid strategy of adding presentation layout and content selection on multiple devices. Albeit at a slower pace, new content delivery technologies and networks also appear (3G telephony, DVB-H wireless broadcast, etc.), having their very own limitations and idiosyncrasies.

One remarkable proposal to solve this general problem is made by the User Interface Markup Language UIML (Abrams, Phanouriou, Batongbacal, Williams & Shuster, 1999). Abrams et al discuss many of the presentation problems posed by device proliferation, focusing on Internet access (it should be noted that in this paper this is given a broader scope) and propose the UIML language as a solution. UIML describes an XML language used to describe generic interfaces that can be "rendered" for any target platform. It has a specific logical structure, defining elements in the interface, appliance capabilities, presentation styling and domain-dependent information. UIML is being proposed as a standard (Oasis Open, 2004) by the non-profit OASIS organization (with IBM and Sun amongst its members).

Also interesting is the analysis of XML layout languages made by Zdun (2001), he introduces the technique in the MHP domain as an example publication channel, though it can be applied to other domains such as "web engineering and content management". As Zdun states, interactive MHP

Farré Giribet D. (2007).
USING XSD INHERITANCE FOR XML-DRIVEN PRESENTATION LAYOUT IN WWW, ITV AND MULTI-CHANNEL PUBLISHING - Facilitating Large-Scale Publishing.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Web Interfaces and Applications*, pages 315-320
DOI: 10.5220/0001287603150320

applications use content from many different sources and that is just one of the publishing channels available to broadcasters (such as the WWW).

Following this trend, a similar this approach is also taken in the well-known XForms standard (W3C, 2006), its second edition in recommendation status as of this writing. Though it is a very flexible technology, a lot of emphasis is put on user-data entry forms, a common interaction case in the WWW. XForms uses an XML-based language that can be interpreted straightaway or converted on the fly to presentation languages such as HTML.

Using an XML language for presentation layout potentially allows content editors to design GUIs without any expertise of the final presentation technical details (HTML in the case of WWW deployment, Java in the case of MHP and so on).

2 XML PRESENTATION LANGUAGE STRATEGY

There are many variations and applications of the XML layout languages strategy, in this section, a specific 'implementation' is described to serve as basis for the proposal of section 5.

In targeting a new publishing medium, a content provider wishing to apply the XML presentation strategy can define a custom appliance-specific XML-based layout language. This language can define several presentation characteristics best described by example:

• Content Selection: information about the data that should appear on the interface, its general type and source selection, for instance "weather forecast images" and "latest international news headlines" and so on.

• Content disposition: the language includes information on content layout in a two-dimensional space or relative content disposition and nesting information, e.g. the "weather forecast section" is conceptually the parent of "weather images".

• Other information: some other custom attributes like sorting, number of elements, etc.

This language is usually defined in an ad-hoc fashion and can have a number of DTD or XML schema files to provide instance validation. This language can be then used to create instances of content units (e.g. "pages" or "screens") that are to be presented to viewers/users.

One example of such an instance related to weather information would be like:

```
<screen type="weather">
    <headlines n="3" type="news"/>
    <forecasts n="3">
        <forecast>
            <title/>
            <body image="no"/>
        </forecast>
    </forecasts>
</screen>
```

In this example, the content editors have created a content unit of weather news containing a number of headlines and a number of weather forecasts. The headlines are of a more relative importance to the forecasts and should come first or be displayed more prominently. The disposition and contents of forecasts are also specified, having a title and a body without any images (see fig. 1).

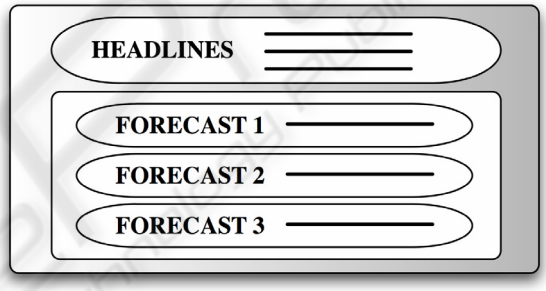

Figure 1: Example of simple layout instance.

It should be noted that the content itself is not specified in the XML layout instances, only pointers or placeholders (like *datasource* identifier strings or SQL query mnemonic id's). In this case, the content layout is independent of the destination channel and can be reused across publishing mediums. However, if the content editors need a fine-grained control over publishing presentation details, different "dialect" variations of the XML language can be defined for some of the presentation channels, for instance by employing optional attributes. It would be up to the system designers to find a proper balance depending on the amount of control desired, publishing channels, etc. In practice, however, usually for each presentation technology a different XML language is used and only *datasource* information and basic attributes are truly 'global'.

Once the publishing medium is defined and the necessary graphic presentation tools and design are completed, a *mapping* between the XML layout language and the presentation technology is built. Whenever content is to be published, the system can automatically parse layout instances, get the content

from the specified *datasources*, merge it with the layout nodes and finally translate to final presentation.

Whenever there is new content and there are no layout changes, the system just re-parses the layout, fetches any new content and produces new presentation material. On the other hand, if a content editor wants to change the layout of a content unit she only needs to alter the XML layout instance file.

This XML layout strategy has been applied successfully in dozens of sites at the CCRTV (the Catalan Government media corporation), several iTV applications, Flash RIA deployments and many other systems (including Microsoft's Media Center PCs). To provide a specific example application, the 3xl.net Flash-based site (CCRTVi, 2005) uses a generic XML layout language to present Flash-based content on the fly. The approach described in this section compares to other XML-based technologies of the next section in its emphasis on simplicity whilst maintaining generality. It is not the aim of this document to invalidate other approaches, instead, a valid working strategy is presented, along with some future relevant enhancements that can be useful to XML architecture designers. It should be noted that a helpful aid in any future improvements of this strategy would be the patterns described by Vogel & Zdun (2002), specially the "content format template" (pp. 216-219).

3 OTHER XML PRESENTATION LANGUAGE EXAMPLES

Zdun (2002) describes the technique of using XML presentation languages and captures the spirit of mapping XML languages to presentation details (with a big focus on simplicity). Besides that generic approach, there exist many implementations sporting a 'one size fits all' approach: mapping from a single XML layout language to one or many presentation technologies. The XSWT Eclipse plug-in (Dorme, 2005) is particularly well structured and defines an XML language targeting SWT (IBM's Java Standard Widget Toolkit). Another well-known single-mapping implementation is the cross-platform Mozilla XPToolkit (mozilla.org, 2005), aiming to "make UIs as easy to build as web pages".

All these single XML language technologies mainly focus on interactive application UI design (though related to the general philosophy described by Zdun) and do not try to target a wide range of presentation technologies. In contrast, the UIML effort tries to solve the problem in a generic way to allow for multi-channel publishing. However, UIML has proven too generic in many scenarios (Ali, Pérez-Quiñones, Abrams & Shell, 2002) as "the generic vocabulary is not sufficient to build interfaces for widely varying platforms" and the authors propose "a multi-step process" to make UIML more practical, acknowledging the fact that "differences between display layouts were found to be too significant to simply create one UIML file for one particular platform". Problems such as these have steered the approach described in section 2 and further in section 5 towards a simpler and arguably more practical model. Finally, there are other 'competing' generic model-based proposals that also tackle the original UI design issue and try to avoid the UIML problems, for a good example see the USer Interface eXtensible Markup Language (Limbourg, Vanderdonckt, Michotte, Bouillon & López-Jaquero, 2005).

4 COMMON DESIGN PITFALLS

Using XML layout languages to provide presentation has some known implementation risks:

- The XML-to-GUI mapping is built into the publishing engine and is not very resilient to presentation changes. Major presentation modifications usually require code and implementation tweaks in the publishing engine.
- XSLT templates are a common technology used to implement translation from XML to presentation (such as XHTML pages). Even though XSLT is perfectly adequate for XML document translation (e.g. from template units to final XHTML page files), the translation logic can become entangled and dispersed in many XSLT files.
- Even though ad-hoc XML presentation languages are usually fairly simple from a technical-savvy user's point of view, content editors and journalists can have problems editing XML files, leading to verification problems, lost productivity and error-prone publication. Even though well-known XML-validation editing tools do exist, they are usually tailored to programmers and not suited for end-user usage.
- A common approach to solve the difficulties that content editors have in editing the XML presentation instances is to build a custom XML layout editor. In this case, there is a significant risk of having the XML-presentation mapping embedded in the editor application code. This is like having the mapping embedded in the publishing engine, the

rigidity problems have just been shifted to a different subsystem of the whole architecture.

• When using XML schemas to perform some sort of validation on presentation units created by content editors, there is a risk of duplicating validation logic across many XSD files. Additionally, *datasources* are usually the same among the different publishing channels targeted by the content producer and this can lead to more logic duplication.

To avoid problems such as these outlined, the author suggests a structured approach (evolved from section 2), thoroughly described in the next section.

5 A LAYERED XSD APPROACH

Using XML Schema definition files to verify the presentation units of content editors is a natural approach to ensure correctness. On top of that, it strengthens the specification and definition of the language and any "dialects", also serving as a formal technical specification document. However, in the XSD documents there should not be any duplicate validation and specification logic. Basic data types, domain-specific information, *datasources* and any other shared logic should be defined only once. There is also the challenge of providing the content editors with productivity-enhancing GUI layout tools to generate presentation instances. There is also the extra hurdle of maintaining XSD validation files and the GUI tool internal validation and XML generation code.

The proposed technique is to use XML schemas having a layered structure to both solve the common design pitfalls and still serve as a GUI design tool implementation foundation.

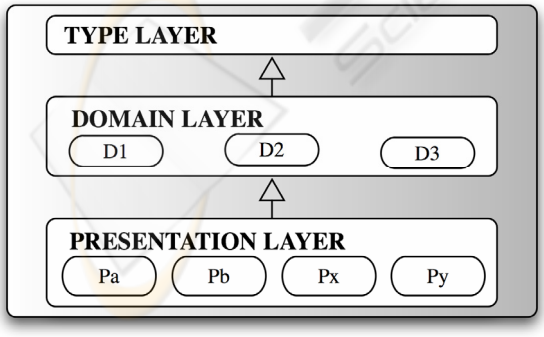

Figure 2: Proposed XSD layers.

The different layers go from the most abstract schema definitions to the more specific validation logic. There are three levels (see fig. 2) and the order from the most abstract to the more specific also corresponds to an increase of the number of definition elements in each layer. The most abstract one is the 'Type Layer, the middle layer is the 'Domain Layer' and finally the least general layer is named the 'Presentation Layer'.

5.1 Type Layer

Its responsibility is to define the XML schema elements considered global within the organization. Its contents and definitions are to be reused by all the other layers. The schema definitions are usually only <simpleTypes> but any other XML schema definition can be used (though organization-wide complex type definitions are usually scarce). Another property of this layer is that it is self-sufficient and self-defined (besides the XMLSchema namespace itself).

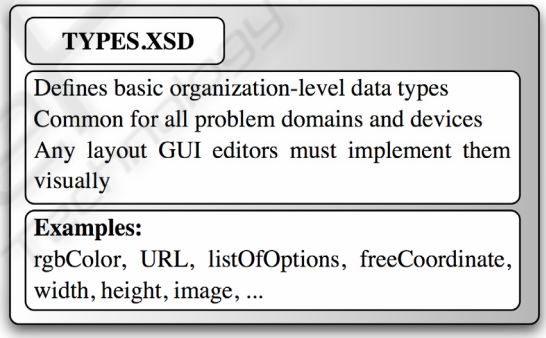

Figure 3: Type Layer summary.

Using a namespace for this layer is up to the system designers. Due to its expected low number of definitions, a single XSD file is usually enough. Types need not be defined once and for all, as global necessities grow, fresh new definitions can be added with minimal impact.

Types defined in this layer are to be shared amongst presentation technologies and layout languages. Examples of types defined in this layer are: 'rgbColor', 'URL', 'listOfOptions', 'yesNo', 'xy', etc. (see fig. 3), largely depending on the organization's context and its XML language definitions.

If the XSDs are used to build a GUI-building tool, it should implement all the basic types defined in this layer as 'visual controls' (this is the only layer tied to the GUI tool implementation). Then the

'rgbColor' definition can be presented to the user as a color picker, a 'yesNo' type as a checkbox, a 'listOfOptions' as a dropdown list, etc. Types that cannot be easily associated to traditional GUI widgets can be free text fields. A useful trick to apply in this association is to divide types in two groups: 'explicit' and 'implicit'. Explicit types are those directly edited by users (e.g., a checkbox is checked or unchecked), while implicit types are filled by the tools in response to changes made by the user manipulating interface elements (e.g., *xy* coordinates are updated in relation to movements in the GUI tool). This can increase the usability of the tool by having some data entered implicitly (e.g., not requiring input of *xy* values in a text box).

5.2 Domain Layer

In this layer, the types related to the content domains of the organization are defined, namely any *datasources* or general types falling under the semantic umbrella of the same content domain. An extra property is that any definitions given at this point should be independent of the publishing channel. In this manner, whenever a new channel is targeted the definition of its additional XML dialect has no impact on this layer. On the other hand, whenever the content organization adds a new source of data, it is quite clear where the first modification is to be made: on any domains that the new source is relevant (should the content need to be available to all domains, the Types Layer would be more appropriate).

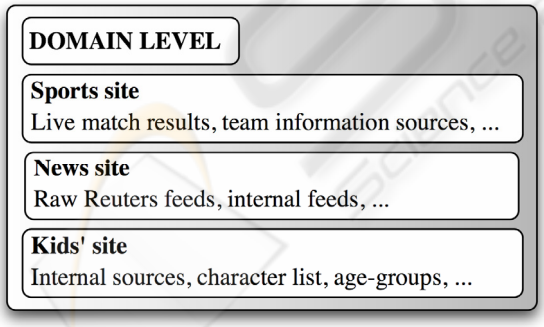

Figure 4: Examples of domain level definitions.

For example, if a new third-part sports content provider is added, a *datasource* id named "news.sports.providerfoo" is added to any domains eligible to publish that new content. No further modifications to any upper or lower layers are needed (e.g. an enumeration-style <simpleType> named 'thirdPartyNewsProviders' is provided that lists all third-party news providers, the new *datasource* is only an addition to this list).

Any types defined in this 'Domain Layer' need to be based on types defined in the 'Type Layer', the easiest way is to use Schema <restriction> inheritance of the abstract layer types. This allows GUI tools to present *datasources* and any domain-specific types to users in a visual manner. Following the example of a sports content provider, the list of third-party providers is a <restriction> enumeration of 'listOfOptions' and is presented to the user as a dropdown list.

Usually one XSD file per content domain is enough. All schema files in this layer need to <include> or inherit all the common definitions in the 'Types Layer'.

5.3 Layout Layer

In the most specific layer, all types belonging to the layout languages and dialects are defined. In this case Schema <complexTypes> are usually used. The complete logical presentation nodes and attributes are defined in this layer, which inherits all the definitions of the 'Domain' and 'Type' layers.

Examples of types defined in this layer are 'currentNewsForecast', 'headlineListing', 'news Ticker', 'optionsMenu' and so on. All those definitions include all the XML language definition and validation logic needed to generate and validate XML presentation instance files in full.

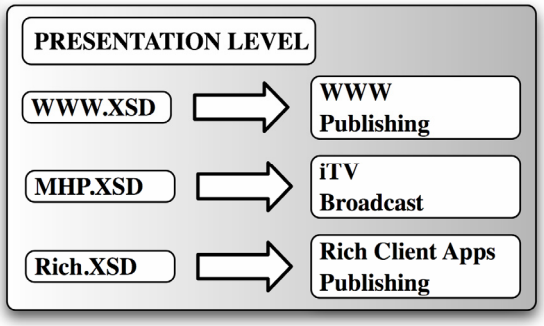

Figure 5: Examples of presentation level definition instances, independent of content domain.

Usually there is a schema file for each presentation style or appliance being targeted (see fig. 5). System designers can choose the number of files and final intended 'generality'.

A problem with this strategy is that due to the nature of XML Schemas, only syntactic information is described. Apart from the implicit association between the types defined in the 'Types Layer' and

GUI tool, no further semantic logic is present in the three XSD layers. If the strategy is used to parameterize GUI layout tools, they could greatly benefit from having more semantic information. Semantics useful to content editors could include defaults for values, mockup preview images for XML layout nodes and in general any useful editing helpers.

That semantic information can be added to the 'Domain' and 'Layout' layers in the form of ad-hoc XML files, its contents and structure dictated by the GUI layout tools implementation. In these XML files, the mapping of logical elements to the final channel presentation language could also be defined (e.g. XML nodes to XHTML snippets, Java code, etc.). If the GUI presentation tool is created having this layered approach in mind it can actually be designed to have the very XSDs as its configuration files. The tool needs to read the definition layer XSDs to configure itself, helped by parsing schema files as 'regular' XMLs. In fact, any changes in the two specific layers would not require any tool changes at all. Only changes to the basic general types would require the GUI tool to be adapted (global types are expected to remain stable for a long time, though).

6 CONCLUSIONS

When using the general strategy of multilayered XML schemas outlined in this paper, several benefits are gained:

- There is a common point of definition of the presentation languages throughout the whole architecture and the definition layers have precise and well-defined responsibilities.
- It is clear where changes are to be made in any situation: new global types or content domains are created, etc.
- The XSD files double up as a self-defined formal unambiguous technical documentation for the publishing architecture and can be used to validate presentation units in the server-side.
- Any changes to schema files (apart from basic types) are automatically gained by GUI tools.
- XML editors able to validate using XML schemas can be still used.
- Presentation channels and content domains can be added without sacrificing maintainability.

And finally, it should be noted that the technique described is completely standards-based and implementation-independent.

ACKNOWLEDGEMENTS

The schema strategy described in this paper has been made possible in part due to the funding and kind encouragement of CCRTV Interactiva (Spain).

REFERENCES

Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., & Shuster, J. (1999). UIML: An Appliance-Independent XML User Interface Language. *Proceedings of the World Wide Web Conference*. Retrieved, December 1, 2005 at http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html

Ali, M. F., Pérez-Quiñones, M. A., Abrams, M., & Shell, E. (2002). Ali, M.F., M.A. Pérez-Quiñones, M. Abrams, and E. Shell. Building Multi-Platform User Interfaces With UIML. *Proceedings of 2002 International Workshop of Computer-Aided Design of User Interfaces: CADUI'2002*. Valenciennes, France.

CCRTV Interactiva (2005). *3xl.net*. Retrieved November 15, 2005 from http://www.3xl.net

Dorme, D. J. (2005). *XSWT - XML/SWT page description language*. Retrieved December 1, 2005, from http://sourceforge.net/projects/xswt

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L. & López-Jaquero V. (2005). UsiXML: a Language Supporting Multi-Path Development of User Interfaces. *Engineering Human Computer Interaction and Interactive Systems*. (pp. 200-220) : Springer Berlin / Heidelberg.

OASIS Open (2004). *User Interface Markup Language (UIML) Specification*. Retrieved November 4, 2006, from http://www.oasis-open.org/committees/download.php/5937/uiml-core-3.1-draft-01-20040311.pdf

Mozilla.org (2005). *The XPToolkit Architecture*. Retrieved December 2, 2005, from http://www.mozilla.org/xpfe/xptoolkit/

Vogel, O., & Zdun, U. (2002). Content Conversion and Generation on the Web: A Pattern Language, *Proceedings of EuroPloP 2002*. (pp. 216-219). Irsee, Germany.

W3C (2006). *XForms 1.0 (Second Edition)*. Retrieved November 4, 2006, from http://www.w3.org/TR/2006/REC-xforms-20060314/

Zdun, U. (2002). XML-Based Dynamic Content Generation and Conversion for the Multimedia Home Platform. *Proceedings of the Sixth International Conference on Integrated Design and Process Technology (IDPT)*.