The Essence of Declarative, XML-based Web Applications: XForms & XSLT

Abstract

The value of the browser as a universal platform for user interface interaction may seem a bit overstated with the recent intensity with which AJAX (and other related, but lesser appreciated mechanisms) has caught the limelight. Buried under the lure of javascript trickery and eye candy of various sorts are some developing user interface frameworks upon which a significant amount of automation can be built. In particular, XForms brings a rich history of browser-based data entry to bear in a comprehensive, declarative syntax meant to completely abstract the drudgery of widgets, controls, remote data management, host languages, and other related issues. XSLT has long been the technology of choice for the manipulation of XML content with emphasis where the nature of the manipulation is declarative, predictable and the end result is XML.

In this session, the author discusses best practices, common patterns and pitfalls in using XSLT as a host language for generating web-based user interfaces expressed in XForms. The XPath requirement imposed on both processors (XForms and XSLT) allows for some sophistication in how XPath contexts are setup for evaluation. This scarcely-mentioned combination results in a level of expressiveness without precedent in web-applications and demonstrates a very tangible value in leveraging XML technologies for this purpose.

Introduction

XSLT has a very well-known and well-articulated value proposition for content publishing. The ability to separate content from presentation in a very declarative way is a powerful idiom for minimizing the cost of maintenance and increasing re-usability. The value of XSLT to web-based content publishing in this regard is analogous to the value proposition of XSLT for generating XForms as well.

XForms Paradigm

The value of XForms to the general architectural style of rich, dynamic, expressive and asynchronous web applications is perhaps best articulated within the context of the recent W3C Coordination Group Note: *Rich Web Application Backplane*. It describes a set of common building blocks for submission, an XML data model, model-view binding, behavior, and web components or widgets that enable both declarative and imperative web application programming. These building blocks are the essential construction tools for the what is commonly referred to as Web 2.0 applications.

The architectural style that is Rich Web Application Backplane seeks to unify the expansive, fragmented landscape of browser adoption of XML by suggesting a common means for submitting XML content for both web programming styles (imperative and declarative). XForms is definitely more on the declarative side and it is this particular trait (in addition to the fact that it relies on XML as its core data model) that makes it amenable for use with XSLT. XForms is declarative because it uses

markup to express its organization and behavior. It is also declarative because it is just abstract enough to be independent of any particular browser. Finally, XForms is also *expressively* declarative and can be thought of as a Universal Turing Machine for web-based user interfaces.

XForms Challenges

XForm's primary challenge is authoring. XForms is a very recursive language. It is also very verbose due to its reliance on XML for it's syntax. Finally, XForms presents a very steep learning curve to developers mostly familiar with adhoc, scripting-oriented web programming.

In particular, XForms at its heart is built on bindings from the user interface to one or more *live* DOM trees. The bindings are facilitated both by run-time evaluation of XPath expressions on these live DOMs as well as the registration of listeners for XML Events triggered from them. The combination of the heavy reliance on the state of an XML tree at run-time and the collective learning curve of both XML Events and XPath presents a challenging set of technologies to tackle especially for a developer with only introductory experience in each.

XForms 1.0 suffers from a lack of programmatic expressiveness compared to its scripting counterparts (Javascript, E4X, etc.). It is not exactly turing complete and lacks certain programming constructs and idioms that are often associated with full blown programming languages. A significant proportion of this deficiency is addressed by the recent XForms 1.1 Working Draft which introduces (for instance) enhancements for the duplication and destruction of nodes within homogeneous collections.

Finally, XForms implementations suffer from blatant disparities in how controls are styled and CSS properties are bound to certain components of controls (their labels, the raw data field, etc.). Often the CSS written for the Mozilla XForms extension are very different from those written for the FormsPlayer Internet Explorer plugin. This is mostly due to the current poor state of CSS support across browsers than with XForms specifically.

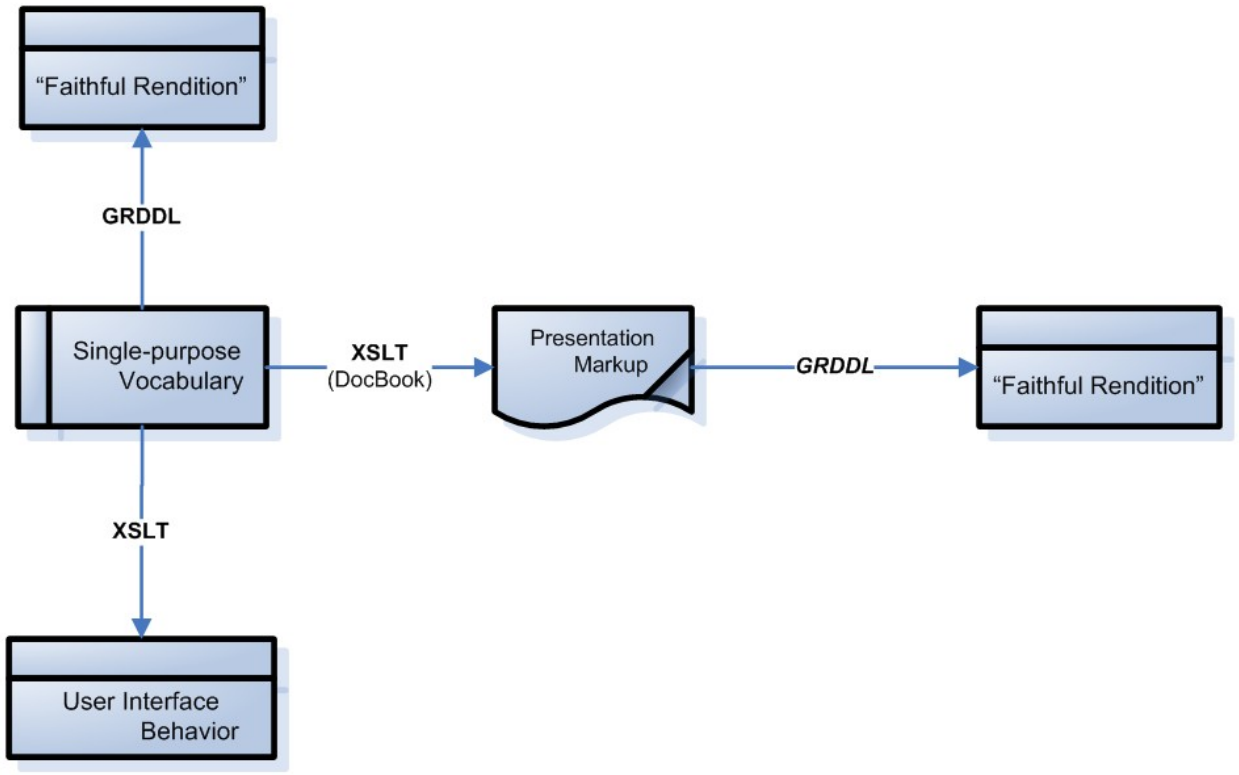
Facet Segregation

The term *Facet Segregation* is used in this paper to identify a methodology for separating and abstracting individual aspects of XHTML and its derivative, related dialects into single-purpose vocabularies and using XSLT to *compile* more expressive languages from them. It can be considered an application AspectXML for the composition of web dialects.

It extends the well-established method(s) of separating content from presentation (used by DocBook XSLT stylesheets, for instance) with both Gleaning Resource Description Dialects of Languages (**GRDDL**) and the value proposition presented in this paper. The GRDDL specification:

> *.. introduces markup for declaring that an XML document includes gleanable data and for linking to algorithms, typically represented in XSLT, for gleaning the resource descriptions from the document.*

It is within this larger picture that the value proposition of separating presentation markup from more specific application logic is proposed.



Editing Atom

Atom is the target payload vocabulary used to demonstrate the value proposition of XSLT for XForms. There are several reasons for this. Atom is also a very recursive XML format and lends itself well to a declarative approach for building user interfaces for data entry. It is also a very popular content model, specifically meant to meet requirements for syndication feeds but capable of being a general purpose exchange format for web content.

The other reason why Atom is a good fit is that it can be deployed over HTTP via the Atom Publication Protocol (**APP**) which lends itself well to an expressive submission mechanism. XForm's *submission* elements can be configured as the external port from which Atom content can be transmitted using the various HTTP methods - each of which have very well defined bindings within **APP**.

A Mix of UI Vocabularies

This paper also relies on the XML User Interface Language (**XUL**) to demonstrate the value of XSLT in generating XForms from a more abstract user interface language. XUL includes some overlap with XForms but has in its arsenal many more high-level components such as grids, menus, windows, and the like. XUL itself attempts to provide a clear separation from application logic and presentation and thus helps achieve the more broader goal in this regard.

The general idea is to ease authoring of XForms by abstracting collection of common user interface components (expressed in XUL – and other constructs we will introduce later) from which more

complicated XForms can be generated.

From XUL to XForms / XHTML

Below is more comprehensive mapping from XUL to XForms and XHTML than will be used for the Atom example:

- xul:button -> xforms:trigger
  - @image -> xforms:trigger/xhtml:img
  - @orient -> xhtml:img floating
- xul:label -> xforms:output? (embedding in controls via @control)
- xul:textbox -> xforms:input | xforms:secret | xforms:textarea (via @multiline)
- xul:radiogroup/xul:radio -> xforms:select1 (with @appearance)
- xul:listbox/xul:listitem -> xforms:select1/xforms:items
- xul:menulist/xul:menupopup/xul:menuitem -> xforms:select1 (@apprearance)
- xul:box[@orient | @align] -> table
- xul:groupbox -> fieldset
- xul:spacer -> div (empty)
- xul:tabbox -> xforms:switch
- xul:grid -> table (or css)

Abstract Vocabularies for User Interfaces

In addition to translations from XUL to XForms and XHTML, we will introduce two more abstract *patterns* that model very general but reusable user interface behavior for the components of the Atom vocabulary. For both patterns we will use the **ui** prefix which will be bound to the URI: urn:abstract-user-interface-patterns:v1/xml

Existential Block

A *ui:existential-block* is a widget which renders a (labeled) placeholder when absent, otherwise the content of its body is displayed. The general idea is to facilitate data entry with the consistent use of a mechanism for populating nodes (or nodesets) in their absence. The (XForms) template for transforming existential blocks is below:

```
<xsl:template match="ui:existential-block">
  <xf:group ref="current()[not({@node})]">
    <img src="add.gif" alt="Add {@node}" title="Add {@node}"/>
  </xf:group>
  <xsl:apply-templates select="*"/>
</xsl:template>
```

Here, we render an icon in the absence of the node associated with the existential-block (via the @node attribute). It is left as an exercise for the reader to modify the template with more specific XForms actions for *inserting* the missing node upon clicking the rendered icon.

Attribute Anchor

The other pattern described here is a simple widget for rendering form components for element

multiple attributes. This construct is used to model Atom *link* and *category* elements. Below is the template for transforming a ui:attribute-anchor into XForms components:

```
<xsl:template match="ui:attribute-anchor">
  <xf:group ref="{@node}">
    <xsl:for-each select="ui:attribute">
      <xf:input ref="{@attrName}">
        <xf:label><xsl:value-of select="@label"/></xf:label>
      </xf:input>
    </xsl:for-each>
  </xf:group>
</xsl:template>
```

Finally, we have a demonstration of using *both* constructs (as well as a mix of XForms) for Atom categories and links:

```
<ui:existential-block node="atom:category">
  <xf:repeat nodeset="atom:category">
    <ui:attribute-anchor node=".">
      <ui:attribute label="Scheme" attrName="@scheme"/>
      <ui:attribute label="Term"   attrName="@term"/>
    </ui:attribute-anchor>
  </xf:repeat>
</ui:existential-block>
```

```
<ui:existential-block node="atom:link">
  <xf:repeat nodeset="atom:link">
    <ui:attribute-anchor node=".">
      <ui:attribute label="Relation" attrName="@rel"/>
      <ui:attribute label="URL"      attrName="@href"/>
    </ui:attribute-anchor>
  </xf:repeat>
</ui:existential-block>
```

'Pushing' through the Vocabulary Mix

An obvious question to ask at this point is how such a mix of vocabularies can be properly processed in an organized, reusable fashion. The XSLT problem solving paradigm often breaks down along two approaches: *push* versus *pull*. Essentially, *push* takes a more declarative approach to template processing by triggering the template matching process from a given context rather than explicitly iterating over the same context.

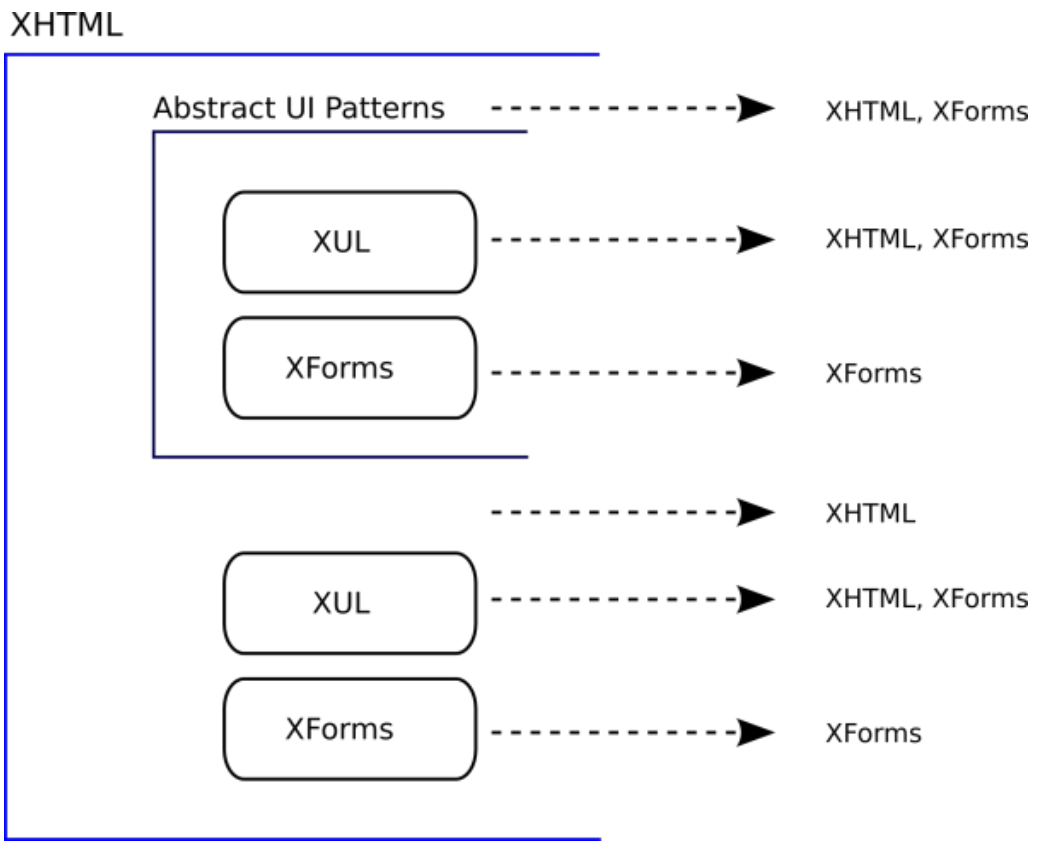
The reader will note that the existential block template invokes xsl:apply-templates to pass on further

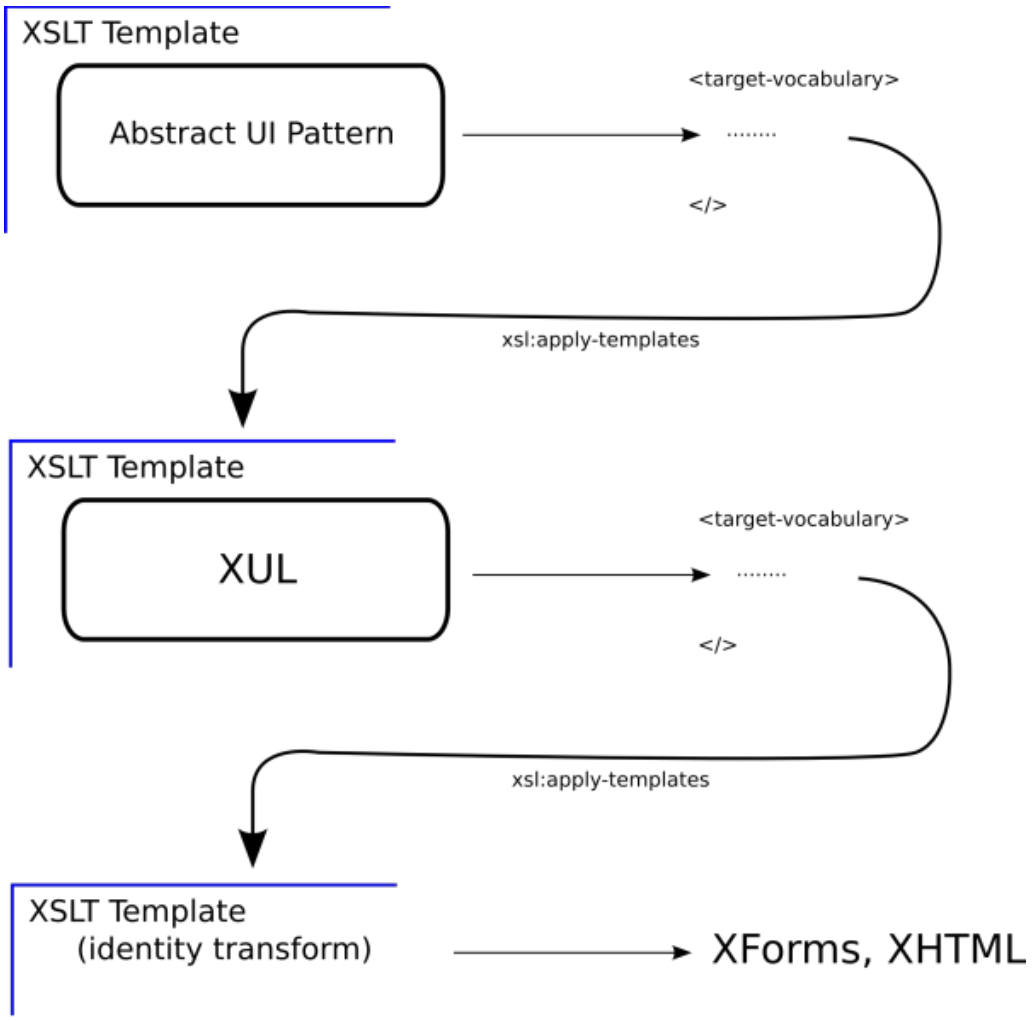
processing to other templates which can match XUL, XHTML, or XForms markup recursively:

```
<xsl:template match="ui:existential-block">
    .... snip ...
    <xsl:apply-templates select="*"/>
    .... snip ...
</xsl:template>
```

A final piece to processing this plethora of user interface vocabularies is the all common identity transform which is used to leave XHTML and XForms markup as is within the original source document:

```
<xsl:template match="@*|node()">
    <xsl:copy>
        <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
</xsl:template>
```





Conclusions

The use of push XSLT processing, facet segregation, and well-modeled, single-purpose vocabularies such as Atom and XUL provides a very powerful tool chest to ease the authoring headaches often associated with XForms and enable the next generation of XML-based web applications associated with the Rich Web Application Backplane architecture.

A reader with some familiarity with XForms implementation nuances can appreciate the other aspects of XForms that can be *throttled* via the XSLT transforms. For example the subtle difference between one XForms implementation and another (differing levels of conformance, perhaps) can be accommodated by using XSLT's import mechanism to override default behavior with behavior specific to a particular implementation.

In addition, a reader familiar with XUL *and* XForms might take note of the fact that both frameworks rely heavily on XML Events to manage user interface behavior. As a further exercise, such a reader might want to experiment with how the common reliance on XML Events can be leveraged by the XSLT transforms presented here. This paper really only scratches the surfaces of the clean, reusable, and expressive nature of using XSLT to generate XForms.

Listing

Complete XSLT Transform

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0"
                xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
                xmlns:xf="http://www.w3.org/2002/xforms"
                xmlns:ev="http://www.w3.org/2001/xml-events"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                xmlns="http://www.w3.org/1999/xhtml"
                xmlns:atom="http://www.w3.org/2005/Atom"
                xmlns:ui="urn:abstract-user-interface-patterns:v1/xml">
  <xsl:template match="ui:existential-block">
    <xf:group ref="current()[not({@node})]">
      <img src="add.gif" alt="Add {@node}"/>
    </xf:group>
    <xsl:apply-templates select="*"/>
  </xsl:template>
  <xsl:template match="xul:grid">
    <table>
      <tbody>
        <xsl:for-each select="xul:rows/xul:row">
          <tr>
            <xsl:for-each select="*">
              <xsl:variable name="curr-pos" select="position()"/>
              <xsl:element name="xhtml:td" namespace="http://www.w3.org/1999/xhtml">
                <xsl:for-each select="../../../xul:columns/xul:column[position() = $curr-pos]/@*">
                  <xsl:copy-of select="."/>
                </xsl:for-each>
                <xsl:apply-templates select="."/>
              </xsl:element>
            </xsl:for-each>
          </tr>
        </xsl:for-each>
      </tbody>
    </table>
  </xsl:template>
  <xsl:template match="xul:label">
    <span style="font-weight:bold;"><xsl:value-of select="text()"/></span>
  </xsl:template>
  <xsl:template match="xul:groupbox">
    <fieldset>
      <xsl:apply-templates select="*"/>
```

```
    </fieldset>
  </xsl:template>
  <xsl:template match="xul:caption">
    <legend>
      <xsl:apply-templates select="* | @*"/>
    </legend>
  </xsl:template>
  <xsl:template match="xul:caption/@label">
    <xsl:value-of select="text()"/>
   </xsl:template>
  <xsl:template match="ui:attribute-anchor">
    <xf:group ref="{@node}">
      <xsl:for-each select="ui:attribute">
        <xf:input ref="{@attrName}">
          <xf:label><xsl:value-of select="@label"/></xf:label>
        </xf:input>
      </xsl:for-each>
    </xf:group>
  </xsl:template>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Complete Atom User Interface

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
      xmlns:xf="http://www.w3.org/2002/xforms"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xhtml="http://www.w3.org/1999/xhtml"
      xmlns:atom="http://www.w3.org/2005/Atom"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="urn:abstract-user-interface-patterns:v1/xml">

  <head>
    <title>Editing Atom Feed</title>
    <xf:model>
      <xf:instance src="ongoing.atom"/>
    </xf:model>
   </head>
  <body>
    <h1>Editing Atom Feed</h1>
    <xf:group ref="/atom:feed">
```

```
<xul:grid>
   <xul:columns>
      <xul:column width="10%"><!-- Labels  --></xul:column>
      <xul:column width="90%"><!-- Content--></xul:column>
   </xul:columns>
   <xul:rows>
      <xul:row>
         <xul:label>Title</xul:label>
         <ui:existential-block node="atom:title">
            <xf:input ref="atom:title"/>
         </ui:existential-block>
      </xul:row>
      <xul:row>
         <xul:label>Identifier</xul:label>
         <ui:existential-block node="atom:id">
            <xf:input ref="atom:id"/>
         </ui:existential-block>
      </xul:row>
      <xul:row>
         <xul:label>Link(s)</xul:label>
         <ui:existential-block node="atom:link">
            <xf:repeat nodeset="atom:link">
               <ui:attribute-anchor node=".">
                  <ui:attribute label="Relation" attrName="@rel"/>
                  <ui:attribute label="URL"      attrName="@href"/>
               </ui:attribute-anchor>
            </xf:repeat>
         </ui:existential-block>
      </xul:row>
      <xul:row>
         <xul:label>Author</xul:label>
         <ui:existential-block node="atom:author">
            <xf:input ref="atom:author/atom:name"/>
         </ui:existential-block>
      </xul:row>
      <xul:row>
         <xul:label>Category(ies)</xul:label>
         <ui:existential-block node="atom:category">
            <xf:repeat nodeset="atom:category">
               <ui:attribute-anchor node=".">
                  <ui:attribute label="Scheme" attrName="@scheme"/>
                  <ui:attribute label="Term"    attrName="@term"/>
               </ui:attribute-anchor>
            </xf:repeat>
         </ui:existential-block>
```

```
            </xul:row>
          </xul:rows>
        </xul:grid>
      </xf:group>
      <xul:groupbox>
        <xul:caption label="Entries"/>
        <xf:repeat nodeset="atom:entry" >
          <xul:groupbox>
            <xul:caption>
              <xf:label ref="@title"/>
            </xul:caption>
            <xul:grid>
              <xul:columns>
                <xul:column width="10%"><!-- Labels  --></xul:column>
                <xul:column width="90%"><!-- Content--></xul:column>
              </xul:columns>
              <xul:rows>
                <xul:row>
                  <xul:label>Title</xul:label>
                  <ui:existential-block node="atom:title">
                    <xf:input ref="atom:title"/>
                  </ui:existential-block>
                </xul:row>
                <xul:row>
                  <xul:label>Identifier</xul:label>
                  <ui:existential-block node="atom:id">
                    <xf:input ref="atom:id"/>
                  </ui:existential-block>
                </xul:row>
                <xul:row>
                  <xul:label>Updated</xul:label>
                  <ui:existential-block node="atom:updated">
                    <xf:input ref="atom:updated"/>
                  </ui:existential-block>
                </xul:row>
                <xul:row>
                  <xul:label>Published</xul:label>
                  <ui:existential-block node="atom:published">
                    <xf:input ref="atom:published"/>
                  </ui:existential-block>
                </xul:row>
                <xul:row>
                  <xul:label>Link(s)</xul:label>
                  <ui:existential-block node="atom:link">
                    <xf:repeat nodeset="atom:link">
```

```
                    <ui:attribute-anchor node=".">
                      <ui:attribute label="Relation" attrName="@rel"/>
                      <ui:attribute label="URL"      attrName="@href"/>
                    </ui:attribute-anchor>
                  </xf:repeat>
                </ui:existential-block>
              </xul:row>
              <xul:row>
                <xul:label>Author</xul:label>
                <ui:existential-block node="atom:author">
                  <xf:input ref="atom:author/atom:name"/>
                </ui:existential-block>
              </xul:row>
              <xul:row>
                <xul:label>Category(ies)</xul:label>
                <ui:existential-block node="atom:category">
                  <xf:repeat nodeset="atom:category">
                    <ui:attribute-anchor node=".">
                      <ui:attribute label="Scheme" attrName="@scheme"/>
                      <ui:attribute label="Term"    attrName="@term"/>
                    </ui:attribute-anchor>
                  </xf:repeat>
                </ui:existential-block>
              </xul:row>
            </xul:rows>
          </xul:grid>
        </xul:groupbox>
      </xf:repeat>
    </xul:groupbox>
  </body>
</html>
```

References

- "*Rich Web Application Backplane*" http://www.w3.org/TR/backplane/
- "*XForms 1.1 Working Draft*" http://www.w3.org/TR/xforms11/
- "*GRDDL Working Draft*" http://www.w3.org/TR/grddl/
- "*XML User Interface Language (XUL)*" http://www.mozilla.org/projects/xul/
- "*XSL Transformations (XSLT) Version 1.0*" http://www.w3.org/TR/xslt
- "*The Atom Syndication Format*" http://tools.ietf.org/html/rfc4287
- "*Assets, Atom Feeds, and AspectXML*" http://www.oreillynet.com/xml/blog/2005/09/part_3_assets_atom_feeds_and_a.html
- "*Push vs Pull*" http://www.dpawson.co.uk/xsl/sect2/pushpull.html