Using XSL, XForms and UBL together to create complex forms with visual fidelity

Klaas **Bals**

Abstract

This paper will explain how XSL-FO, XSLT, XForms and UBL can be used together (and how the implementation in Scriptura XBOS is done). Each technology contributes its own strengts to the total solution. XSL-FO for page oriented layout with a visual fidelity, XForms for advanced and flexible forms, and UBL to represent the business data.

Together they allow to create UBL documents such as invoices in a very powerful and flexible way, all with open standards.

Several challenges are explored. Typically, XSL-FO is used for paginated output, but not for user interaction, where user actions can change the output. XForms is typically used in combination with XHTML to create rich web forms.

Also problems encountered when trying to use these technologies together are explained. For example the way XForms uses CSS to apply dynamics in contrast with the fact that XSL uses XSLT to apply dynamics. Or the limations that the built-in Dynamic Effects elements in XSL-FO have in relation with XForms.

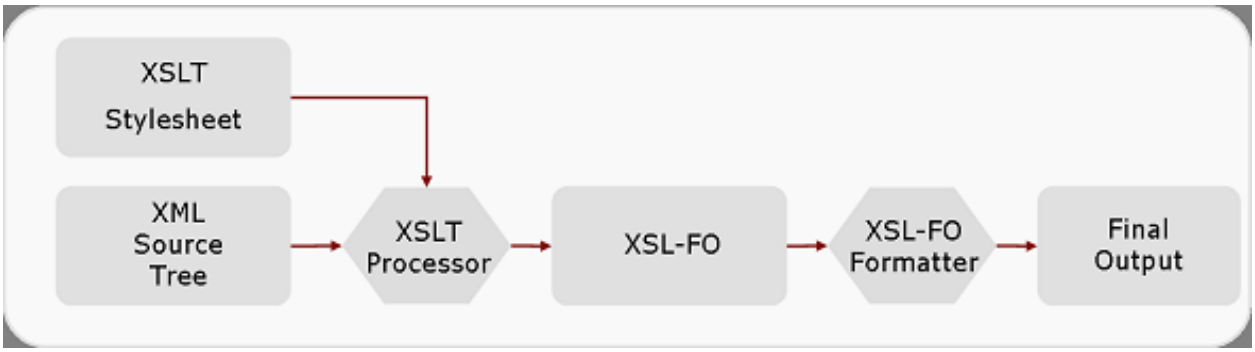
In the end, a working prototype implementation will be shown that proves that these three technologies really can be used together, and that they provide capabilities comparable to Adobe PDF Forms, but then in an open standard way.

Table of Contents

1. eXtensible Stylesheet Language

XSL is composed out of XSLT and XSL-FO. XSLT (with T for Transformations) is the language that is used in XSL to transform an XML document into an XSL-FO document. This XSL-FO (with FO for formatting objects) document is then rendered into a final output format on screen, in PDF format, PostScript etc.

XSL-FO is a language to express formatting semantics. XSL-FO is the core of Scriptura XBOS, a product from Inventive Designers, where it is used to represent page oriented layout throughout the complete process. Apart from XSL-FO, Scriptura uses a variety of other open standards, such as SOAP Web Services (with WSDL, security, ...), SVG, OpenOffice Charts, WebDAV, XForms, XSLT, etc...



The XSLT processor takes the XSLT stylesheet and the XML instance as its input. It generates an XSL-FO result tree. This XSL-FO tree is then transformed into a final output format by an XSL-FO formatter. This figure shows XSLT and XSL-FO as two distict steps, but of course in the real world they can occur at the same time in a streaming process, or even by a piece of code that is both an XSLT and XSL-FO processor.

Figure 1. A schematic overview of the XSL process

At time of this writing, XSLT 2.0 is a Candidate Recommendation, and XSL-FO 1.1 is a Last Call Working Draft.

2. Universal Business Language (UBL)

UBL is a set of schema's for XML business documents, such as purchase orders, invoices, order cancellations etc. Ken Holman has developed XSLT+XSL-FO stylesheets that can render UBL XML instances to XSL-FO and thus PDF and other formats in the UN Layout. This is using XSL-FO 1.0 and it also shows the power of XSL-FO 1.0 right now.

Currently it can be found at http://www.cranesoftwrights.com/resources/ublss/ [http://www.cranesoftwrights.com/resources/ublss/]

In this paper, UBL will be used as an example of how a purchase order can be represented in an interactive form.

A typical representation of a purchase order has multiple pages. The first page typically has a big header containing the address information. The last page has a big footer containing the delivery terms, and all the pages in between have a small header and small footer. If there is only one page, it has a big header and a big footer.

Also, the visual representation contains a page number, and a reference to the page with the delivery terms.

XSL is the perfect language to satisfy these constraints, and to define a stylesheet to transform the UBL instance into a visual representation of the purchase order.

PURCHASE ORDER Page 1 of 3

Order No: 20031234-1

Issue Date: 01-23-03

From: Bills Microdevices
413 Spring St
Elgin, IL 60123

To: Joes Office Supply
32 W. Lakeshore Dr
Chicago, IL 60022

Contact: George Tirebiter

Deliver to: 413 N Spring St
Elgin, IL 60123

Requested Delivery Date: 14-02-03

Line Num	*Part Number*	*Description*	*Qty*	*Unit Price*	*Extended Amount*
1	32145-12	Pencils, box #2 red	5	$ 2.50	$ 12.50
2	78-697-24	Photocopy Paper- case	10	$ 30.00	$ 300.00
3	091356-3	Pens, box, blue finepoint	10	$ 5.00	$ 50.00
4	543-165-1	Tape, 1in case	3	$ 12.50	$ 37.50
5	984567-12	Staples, wire, box	10	$ 1.00	$ 10.00
6	091344-5	Pens, box red felt tip	5	$ 5.00	$ 22.50
7	21457-3	Mousepad, blue	12	$ 0.50	$ 6.00
8	32145-12	Pencils, box #2 red	5	$ 2.50	$ 12.50
9	78-697-24	Photocopy Paper- case	10	$ 30.00	$ 300.00
10	091356-3	Pens, box, blue finepoint	10	$ 5.00	$ 50.00
11	543-165-1	Tape, 1in case	3	$ 12.50	$ 37.50
12	984567-12	Staples, wire, box	10	$ 1.00	$ 10.00
13	091344-5	Pens, box red felt tip	5	$ 5.00	$ 22.50
14	21457-3	Mousepad, blue	12	$ 0.50	$ 6.00

Order continued on next page.
See the last page (page 3) for the Delivery Terms.

An example of a visual representation of a UBL purchase order generated using XSLT and XSL-FO (with Scriptura XBOS). [Page 1]

Figure 2. UBL Representation page 1

PURCHASE ORDER Page 2 of 3

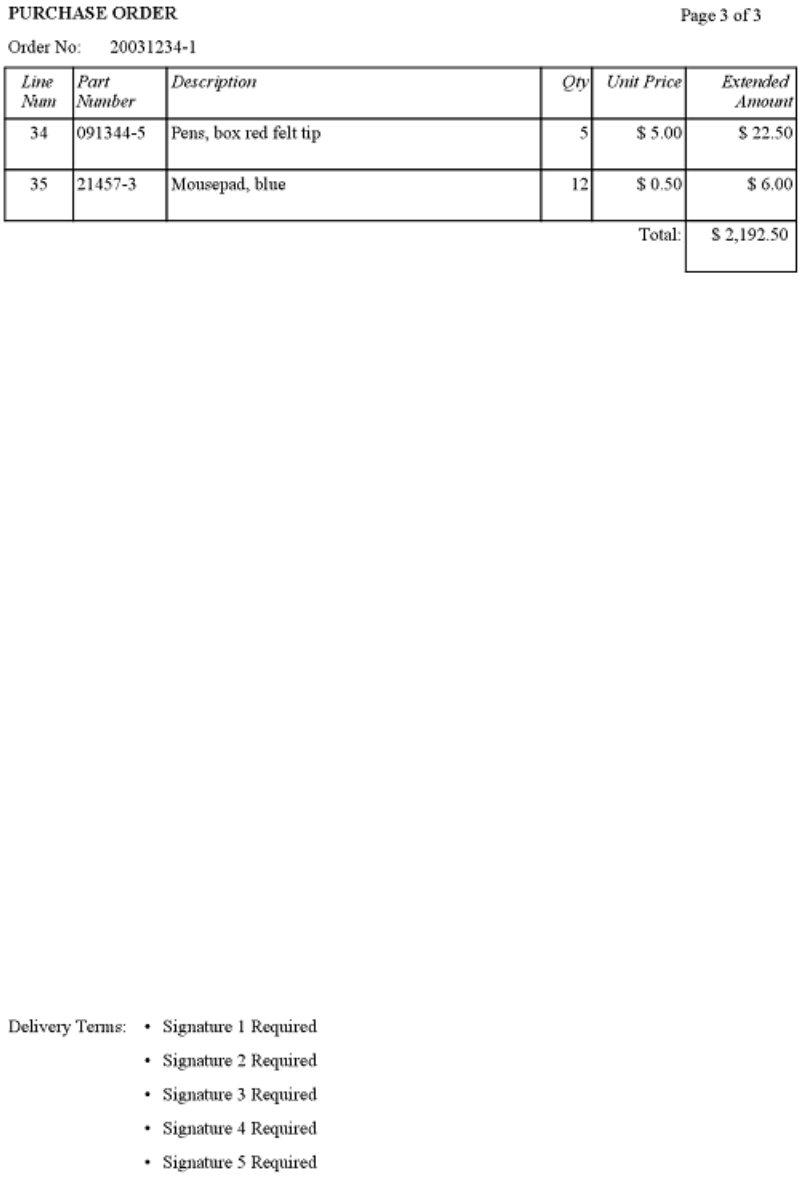
Order No: 20031234-1

Line Num	*Part Number*	*Description*	*Qty*	*Unit Price*	*Extended Amount*
15	32145-12	Pencils, box #2 red	5	$ 2.50	$ 12.50
16	78-697-24	Photocopy Paper- case	10	$ 30.00	$ 300.00
17	091356-3	Pens, box, blue finepoint	10	$ 5.00	$ 50.00
18	543-165-1	Tape, 1in case	3	$ 12.50	$ 37.50
19	984567-12	Staples, wire, box	10	$ 1.00	$ 10.00
20	091344-5	Pens, box red felt tip	5	$ 5.00	$ 22.50
21	21457-3	Mousepad, blue	12	$ 0.50	$ 6.00
22	32145-12	Pencils, box #2 red	5	$ 2.50	$ 12.50
23	78-697-24	Photocopy Paper- case	10	$ 30.00	$ 300.00
24	091356-3	Pens, box, blue finepoint	10	$ 5.00	$ 50.00
25	543-165-1	Tape, 1in case	3	$ 12.50	$ 37.50
26	984567-12	Staples, wire, box	10	$ 1.00	$ 10.00
27	091344-5	Pens, box red felt tip	5	$ 5.00	$ 22.50
28	21457-3	Mousepad, blue	12	$ 0.50	$ 6.00
29	32145-12	Pencils, box #2 red	5	$ 2.50	$ 12.50
30	78-697-24	Photocopy Paper- case	10	$ 30.00	$ 300.00
31	091356-3	Pens, box, blue finepoint	10	$ 5.00	$ 50.00
32	543-165-1	Tape, 1in case	3	$ 12.50	$ 37.50
33	984567-12	Staples, wire, box	10	$ 1.00	$ 10.00

Order continued on next page.
See the last page (page 3) for the Delivery Terms.

An example of a visual representation of a UBL purchase order generated using XSLT and XSL-FO (with Scriptura XBOS). [Page 2]

Figure 3. UBL Representation page 2



PURCHASE ORDER

Page 3 of 3

Order No: 20031234-1

Line Num	*Part Number*	*Description*	*Qty*	*Unit Price*	*Extended Amount*
34	091344-5	Pens, box red felt tip	5	$ 5.00	$ 22.50
35	21457-3	Mousepad, blue	12	$ 0.50	$ 6.00
				Total:	$ 2,192.50

Delivery Terms:

- Signature 1 Required
- Signature 2 Required
- Signature 3 Required
- Signature 4 Required
- Signature 5 Required

An example of a visual representation of a UBL purchase order generated using XSLT and XSL-FO (with Scriptura XBOS). [Page 3]

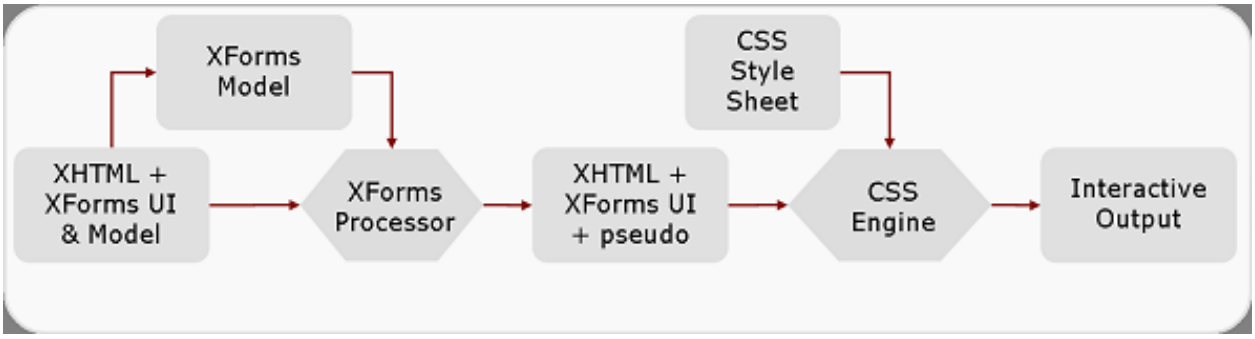
Figure 4. UBL Representation page 3

3. XForms

XForms is a representation of a form in an XML environment. XForms is not a language on itself, it needs a host language, such as XHTML, SVG or XSL-FO. Up to this point, most usages of XHTML are combined with XHTML, and it will even become an integrated part of XHTML 2.0.

XForms has a very clean design to represent the form in separate components. It separates the model, the instance data and the user interface. XForms provides the model and the instance data, and the host language must take care of the user interface. It has support for strong typing, resulting in fast client side validation.

At the time of this writing, XForms 1.1 is a Working Draft.



The XForms processor maintains a data-model consisting of values and states (e.g. validity and relevance). It also maintains the state and processing of the XForms UI, e.g. repeats, cases and any bound element. XHTML embeds the XForms model and UI in its content which results in a semantic representation of a page with one or more forms. The CSS engine will then layout and display the semantic XHTML+XForms page. The XForms UI will expose its states as pseudo-classes to allow the CSS engine to style the XForms controls accordingly to their states. The XForms UI will also expose a set of pseudo-elements which enables the CSS engine to layout the concrete rendering of the XForms controls.

Figure 5. A schematic overview of the XForms process in combination with XHTML and CSS

To allow for the dynamically changing states of the controls (valid/invalid/required etc), a set of pseudo-classes and pseudo-elements were added as extensions to CSS by the XForms working group.

3.1. Code Example of a CSS file using pseudo-*

```
/* Display a red background on all invalid form controls */
*:invalid { background-color:red; }

/* Display a red asterisk after all required form controls */
*:required::after { content: "*"; color:red; }

/* Do not render non-relevant form controls */
*:disabled { visibility: hidden; }
```

4. Combining XForms and XSL-FO

The XSL-FO instance also needs to contain XForms elements. Several XForms elements are embedded using a different approach.

4.1. Embedding the XForms model in XSL-FO

The XForms `xforms:model` element is embedded in the XSL-FO `fo:declarations` element.

This is an example of how the XForms model is embedded in the XSL-FO `fo:declarations` element. Please note that this is an generated example and that's why some identifiers are not very human-friendly.

```
<fo:declarations>
  <xforms:model id="Rm9ybTogRm9ybQ____">
    <xforms:instance id="instance">
      <!-- here is the UBL instance -->
    </xforms:instance>
    <xforms:bind type="xs:base64Binary" required="false()" relevant="true()"

                 readonly="false()" nodeset="DeliveryTerms"
                 id="2319213325121437696" constraint="true()"/>
    <xforms:bind type="xs:unsignedLong" required="true()" relevant="true()"
                readonly="false()" nodeset="BuyersID" id="755313862053666816"

                 constraint="true()"/>
      <xforms:bind nodeset="OrderLine" id="__query_T3JkZXJMaW5l__">
      <xforms:bind type="xs:decimal" required="false()" relevant="true()"
                   readonly="true()" nodeset="FormInput1"
                   id="__query_T3JkZXJMaW5l___706076128828153856"
                   constraint="true()"
                   calculate="count(../preceding-sibling::OrderLine)+1"/>
      <xforms:bind type="xs:string" required="false()" relevant="true()"
                  readonly="false()" nodeset="ItemSellersItemIdentificationID"

                   id="__query_T3JkZXJMaW5l___889669368973639680"
                   constraint="true()"/>
      <xforms:bind type="xs:string" required="false()" relevant="true()"
                   readonly="false()" nodeset="ItemDescription"
                   id="__query_T3JkZXJMaW5l___3789927607584666624"
                   constraint="true()"/>
      <xforms:bind type="xs:string" required="false()" relevant="true()"
                   readonly="false()" nodeset="ItemQuantity"
                   id="__query_T3JkZXJMaW5l___5055423852316362752"
                   constraint="true()"/>
      <xforms:bind type="xs:string" required="false()" relevant="true()"
                   readonly="false()" nodeset="ItemBasePricePriceAmount"
                   id="__query_T3JkZXJMaW5l___3368271152911538176"
                   constraint="true()"/>
      <xforms:bind type="xs:string" required="false()" relevant="true()"
                   readonly="true()" nodeset="LineExtensionAmount"
                   id="__query_T3JkZXJMaW5l___7884086665032749056"
                   constraint="true()"
                  calculate="../ItemQuantity * ../ItemBasePricePriceAmount"/>

    </xforms:bind>
    <xforms:bind type="xs:string" required="false()" relevant="true()"
                 readonly="true()" nodeset="LineExtensionTotalAmount"
                 id="5527481283448094720" constraint="true()"
                 calculate="sum(../OrderLine/LineExtensionAmount)"/>
    <xforms:submission replace="none" method="" id="4081033177616557056"
                       action=""/>
```

```
  </xforms:model>
</fo:declarations>
```

4.2. Embedding XForms Controls in XSL-FO

The XForms Controls elements, such as input, are embedded in the XSL-FO instance using the `fo:instream-foreign-object` element. XSL-FO attributes are used on the XForms elements to specify the layout of the controls.

This is an example of how an `xforms:input` control is embedded:

```
<fo:block-container fo:width="135.6pt" fo:top="50.85pt" fo:left="118.65pt"
                    fo:block-progression-dimension.minimum="16.95pt"
                    fo:absolute-position="fixed">
  <fo:block>
    <fo:instream-foreign-object fo:content-width="135.6pt"
                                fo:content-height="16.95pt">
     <xforms:input st:stylingClass="C755313862053666816" fo:font-size="12.0pt"

                    fo:font-family="Arial" fo:color="#000000"
                    bind="755313862053666816">
       <xforms:label>BuyersID</xforms:label>
      </xforms:input>
    </fo:instream-foreign-object>
  </fo:block>
</fo:block-container>
```

4.3. XForms Repeat

The XForms Repeat constructs are allowed anywhere in the XSL-FO document, as in shown in this example:

```
<fo:table fo:table-layout="fixed">
  <fo:table-column fo:column-width="45.2pt"/>
  <fo:table-header fo:font-style="italic" fo:font-size="12.0pt"
                   fo:font-family="Verdana">
    <fo:table-row>
      <fo:table-cell fo:text-align="center" fo:border-width="1.0pt"
                     fo:border-style="solid">
        <fo:block fo:line-height="normal">Line Num</fo:block>
      </fo:table-cell>
    </fo:table-row>
  </fo:table-header>
  <fo:table-body>
    <xforms:repeat id="__query_T3JkZXJMaW5l___REPEAT"
                   bind="__query_T3JkZXJMaW5l__">
      <fo:table-row>
        <fo:table-cell fo:border-width="1.0pt" fo:border-style="solid">
          <fo:block fo:start-indent="0pt" fo:end-indent="0pt">
            <fo:instream-foreign-object fo:content-width="44.2pt"
                                        fo:content-height="32.9pt">
              <xforms:input fo:font-size="12.0pt"
                            fo:font-family="Times New Roman"
```

```
                              bind="__query_T3JkZXJMaW5l___706076128828153856">

                <xforms:label>FormInput1</xforms:label>
              </xforms:input>
            </fo:instream-foreign-object>
          </fo:block>
        </fo:table-cell>
      </fo:table-row>
    </xforms:repeat>
  <fo:table-body>
<fo:table>
```

4.4. Rich Text Component

To be able to allow form-users to add content to the document, including formatted content with bold, italics, etc, support for a 'Rich Text Component' was added. An `xforms:upload` element is used to add the content of the Rich Text Field, which will be an XSL-FO fragment, to the instance. This means that the instance will contain XSL-FO elements and attributes.

```
<fo:block>
  <fo:instream-foreign-object fo:content-width="423.75pt"
                              fo:content-height="101.7pt">
    <xforms:upload fo:font-size="12.0pt" fo:font-family="Times New Roman">
      <xforms:label>DeliveryTerms</xforms:label>
      <xforms:mediatype ref="@mediatype"/>
    </xforms:upload>
  </fo:instream-foreign-object>
</fo:block>
```

4.5. XForms Label

In XForms, the `xforms:label` element must be a child of the XForms Control. It is also necessary that the label is displayed so that it is clear that it is related to the XForms control. This is done for reasons of accessiblity.

In XSL-FO, you want to have the possibility to have the label appear anywhere you want, for example in a different column of a table. This means that it is not possible to embed the `xforms:label` in the control, without doing something different.

Inventive Designer proposed to the XForms Working Group to have the possibility to have the `xforms:label` element appear ourside of the control, and just makes a reference to the control (or the other way around with a reference from the control to the label). This suggestion did not get enough support. This is an example of what was suggested:

```
<fo:table>
  <fo:table-row>
    <fo:table-cell>
      <xforms:label idref="mylabel"/>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block>
        <fo:instream-foreign-object>
    <xforms:input bind="__query_T3JkZXJMaW5l___706076128828153856">
            <xforms:label id="mylabel">My Label</xforms:label>
```

```
          </xforms:input>
  </fo:instream-foreign-object>
      </fo:block>
    </for:table-cell>
  </fo:table-row>
</fo:table>
```

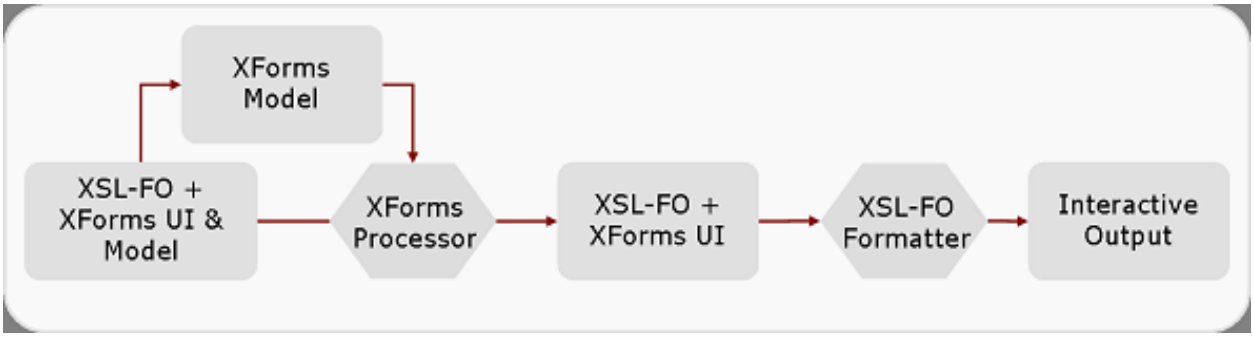
For that reason, it is probably best to use an extension. Another alternative is to just repeat the label contents outside of the control (without any semantic link with the control), and set the fo:visibility attribute to false on the label in the control, as was done in the example:

```
<fo:table>
  <fo:table-row>
    <fo:table-cell>
      <fo:block>My Label:</fo:block>
    </fo:table-cell>
    <fo:table-cell>
      <fo:block>
        <fo:instream-foreign-object>
    <xforms:input bind="__query_T3JkZXJMaW5l___706076128828153856">
           <xforms:label fo:visibility="false">My Label</xforms:label>
         </xforms:input>
  </fo:instream-foreign-object>
      </fo:block>
    </for:table-cell>
  </fo:table-row>
</fo:table>
```

5. XForms aware XSL-FO Formatter

Generating the final output format from XSL-FO is a unidirectional process. After the output has been generated, no interaction is possible (apart from a couple of interactive features that XSL-FO supports). To be able to combine XForms and XSL-FO, the XSL-FO formatter needs to be adapted so that it can cope with the changes that are happening in the document.

When using XSL-FO as a host language, the XSL-FO document contains XForms elements. The way we implemented this, the XSL-FO instance (that now contains XForms) is still generated using an XSLT stylesheet. This means that the form can be dynamically built, for example the purchase order can have a list of products the form-user can select, and these can be extracted from a product catalog when the form is generated. This is shown in this figure.

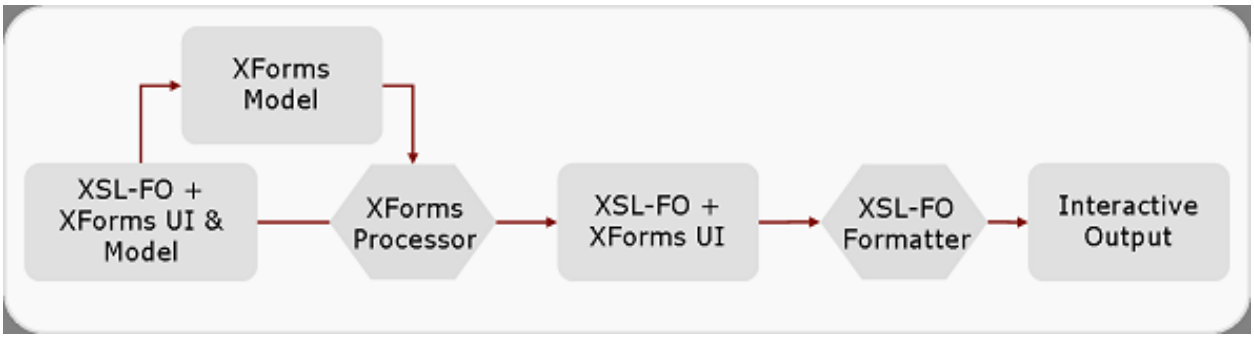


The XSL-FO+XForms document is still generated using an XSLT stylesheet.

Figure 6. A schematic overview of the XForms process in combination with XSL

The XForms aware formatter will show the controls and allow interaction with them. When values are changed, the forms XML instance is updated (and the XML source tree is not updated), and also the screen must be updated to show the change!

If we forget the generation of the XSL-FO+XForms document for now, and focus on the XForms aware XSL-FO Formatter, we will get this figure.



The details of the XForms-aware XSL-FO formatter.

Figure 7. A schematic overview of the XForms-aware XSL-FO Formatter

6. Specify apprearance of XForms controls (and their states)

We the values are updated, its model item properties (required, valid/invalid, relevat) might be updated. This means the XSL-FO Formatter must also show this updated state. This means we need to find a way to express how the controls should look depending on the model item properties. (Thus we need to find an alternative for the CSS pseudo-classes and -elements.) There are different approaches to fullfil this requirement. The example that will be shown illustrates the typical requirement of making the background of a control red when the value is invalid, and making it yellow when the control must be filled in.

There are three alternatives that I will discuss

6.1. fo:multi-properties

XSL-FO has a concept called `fo:multi-properties`, that are intended to provide user interaction on (screen-)output. The element `fo:multi-properties` has an attribute called 'active-state' that can have values 'visited', 'hover', 'focus', etc and depending on the state of the object, the properties related to that state are used on that object.

To be able to use this in the XForms context, new values would need to be introduced, for example 'active-state', 'valid', 'invalid', 'relevant', etc or perhaps better 'xfx:active-state', 'xfx:valid', ...

The good thing about this approach is that it is a know XSL-FO concept, and very nicely integrated into XSL-FO.

But there are also downsides to this approach. One of the problems is the visibility to hide shapes when a control (or the XML element in the XML forms instance) becomes non-relevant. The visibility property in XSL-FO can not completely hide an object. If it is hidden, it still takes up space in the output, and that is not what you want of a non-relevant control. (The reason that XSL-FO is not really removing the object is that it just wasn't necessary, because if you didn't want an object to appear, you just don't produce the elements in XSLT. XSL-FO assumed hiding would happen by XSLT process by just removing the objects from the tree.)

Another downside of this approach is that it is bot possible to change appearance of objects depending on contents, for example make the text red if the entered amount is negative. This feature is not supported with the CSS pseudo-classes and -elements either, but it would be nice to have support for it.

This is an example of how fo:multi-properties would look when combined with XForms:

```
<fo:multi-properties text-decoration="underline">
  <fo:multi-property-set active-state=“xfx:required"
                         background-color=“yellow"/>
  <fo:multi-property-set active-state=“xfx:invalid"
                         background-color="red"/>
  <xforms:input  fo:background-color="merge-property-values()"
                bind=“bind-name">
    <xforms:label>Buyers ID</xforms:label>
  </xforms:input>
</fo:multi-properties>
```

6.2. fo:multi-switch

XSL-FO has a concept of switching the appearance of certain blocks. An `fo:multi-switch` always shows exactly one of its `fo:multi-case` children, and a user/reader can change the case that's displayed by clicking an `fo:multi-toggle`.

To use this in an XForms context, you would need to add a property to every `fo:multi-case` to indicate when this case must be displayed. For example you can add a new property 'xfx:active' with values 'valid', 'invalid', 'relevant', ...

Using this approach, (just as with the fo:multi-properties) it is also not possible to change appearance of objects depending on contents.

Another limitation is that you can not combine cases, so expressing that the control must have a red border when it is invalid and a yellow background when it is required will never show these two appearances at the same time, while this is actually required. This could be solved by allowing multiple values for the xfx:active property, but still it is not very convenient.

The advantage of this approach is that is is more powerful than fo:multi-properties (is can hide objects when non-relevant), and it is also very nicely integrated in XSL-FO

6.3. Extra XSLT stylesheet

The last alternative of changing the appearance of controls depending on their states uses an extra XSLT stylesheet. The extra XSLT stylesheet assumes that the XForms processor that is a part of the XForms-aware XSL-FO Formatter adds extra information to those controls about their state, such that the Formatter can render them with the appropriate appearance.

For example, if the XForms processor find that a control is non-relevant, it can add an attribute or element to the control in a separate namespace. In the XForms solution implemented in Scriptura, the open source Chiba processor is used, and thus elements and properties in the chiba namespace are added. This is an example of a control that is invalid:

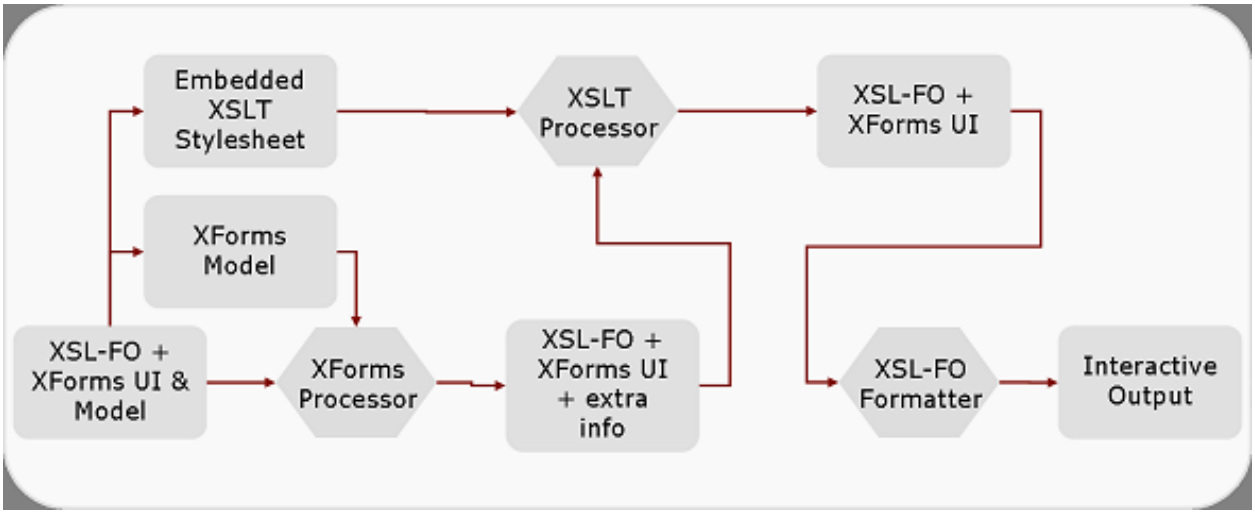
```
<xforms:input bind="xyz">
  <xforms:label>My Label</xforms:label>
  <chiba:data chiba:valid="false"/>
</xforms:input>
```

This means that the XSLT stylesheet should match those elements. This is a fragment of such a stylesheet:

```
<xsl:template match="xforms:input[chiba:data/@chiba:valid = ‘false']">
  <xforms:input fo:background-color=“red” bind=“{@bind}”>
  <xsl:apply-templates/>
  </xforms:input>
</xsl:template>

<xsl:template match="xforms:input[chiba:data/@chiba:required = 'true']">
  <xforms:input fo:background-color=“yellow” bind=“{@bind}”>
  <xsl:apply-templates/>
  </xforms:input>
</xsl:template>
```

This figure shows the detailed operation of the XForms-aware XSL-FO Formatter including the extra XSLT stylesheet.



The details of the XForms-aware XSL-FO formatter including the XSLT Stylesheet.

Figure 8. A schematic overview of the XForms-aware XSL-FO Formatter in detail

The major advantage of this approach is that it is very flexible. Any type of condition can be evaluated to determine the appearance of controls, and even conditions that use the controls values can be done, for example to make the text red in the amount is negative. It is also possible to have alternating row colors in a dynamic table for example.

Of course this approach also has a downside. Apart from being a new concept, which means it is not very nicely integrated in XSL-FO, there also is a performance penalty to executing this stylesheet. And as this stylesheet has to be executed every time a control's value changes, the user may notice the delay for big forms.

7. Strengths of XForms + XSL

The major advantage of using XForms and XSL-FO together is that you can create forms with visual fidelity. This feature is very important for legal and business reasons and is a funcamental step in using electronic forms.

The advantages of XForms are still valid: Strong typing, repeated structures, XML data model, client side validation, etc.

When this is combined with other features, you can dynamically update barcodes and even charts (pie charts, bar charts, etc). Also rich text area's enable people to add content even with italics, underline etc.

It has all of the advantages of PDF Forms, but then really using open standards, were PDF Forms are actually only using XML and don't use open standards for the form or layout specification.

Biography

Klaas **Bals**
CTO
Inventive Designers [http://www.inventivedesigners.com]
Sint-Bernardsesteenweg 552
Antwerp
2660
Belgium

Klaas Bals is Chief Technology Officer of Inventive Designers [http://www.inventivedesigners.com] where he is responsible for Scriptura XBOS, a document design, generation, delivery and forms solution using XSL-FO and XForms, including the WYSIWYG designer.

He studied Computer Science at the University of Antwerp, Belgium. He is a member of the XSL working group at the W3C. He can be reached at the following email address: 'Klaas_Bals (at) inventivedesigners.com'.