Building Rich Web Applications using XForms 2.0

Nick van den Bleeken
<nick.van.den.bleeken@inventivegroup.com>

Abstract

XForms is a cross device, host-language independent markup language for declaratively defining a data processing model of XML data and its User Interface. It reduces the amount of markup that has to be written for creating rich web-applications dramatically. There is no need to write any code to keep the UI in sync with the model, this is completely handled by the XForms processor.

XForms 2.0 is the next huge step forward for XForms, making it an easy to use framework for creating powerful web applications.

This paper will highlight the power of these new features, and show how they can be used to create real life web-applications. It will also discuss a possible framework for building custom components, which is currently still missing in XForms.

Table of Contents

1. Introduction

Over the last 2 years there is a trend of moving away from browser plug-in frameworks (Adobe Flash, JavaFX, and Microsoft silverlight) in favor of HTML5/Javascript for building rich web-applications. This shift is driven by the recent advances in technology (HTML5 [HTML5], CSS [CSS] and Javascript APIs) and the vibrant browser market on one hand, and the recent security problems in those plug-in frameworks on the other hand.

Javascript is a powerful dynamic language, but a potential maintenance nightmare if one is not extremely diligent. Creating rich web-applications using javascript requires a lot of code. There are a lot of frameworks (like Dojo [DOJO] and jQuery [JQUERY]) that try to minimize the effort of creating user interfaces. Dojo even goes one step further by allowing you to create model-view-controller applications, but you still have to write a lot of javascript to glue everything together.

XForms is a cross device, host-language independent markup language for declaratively defining a data processing model of XML data and its User Interface. It uses a model-view-controller approach. The model consists of one or more XForms models describing the data, constraints and calculations based upon that data, and submissions. The view describes what controls appear in the UI, how they are grouped together, and to what data they are bound.

XForms reduces the amount of markup that has to be written for creating rich web-applications dramatically. There is no need to write any code to keep the UI in sync with the model, this is completely handled by the XForms processor.

XForms 2.0 is the next huge step forward for XForms, making it an easy to use framework for creating powerful web applications. This paper will first discuss the most important improvements in this new version of the specification, followed by an analysis of possible improvements.

2. XForm 2.0

This section will discuss the most important improvements of XForms compared to its previous version. Those improvements make it easier to create powerful web applications that integrate with data available on the web.

2.1. XPath 2.0

XPath 2.0 [XPATH-20] adds a much richer type system, greatly expands the set of functions and adds additional language constructs like 'for' and 'if'. These new language features make it much easier to specify constraints and calculations. At the same time it makes it easier to display the data the way you want in the UI.

Example 1. XPath 2.0: Calculate order price

The folowing XPath expression calculates the sum of the multiplication of the pricae and quantatiy of each item in the order:

```
sum(for $n in order/item return $n/price * $n/quantity)
```

2.2. Attribute Value Templates

Attribute Value Templates [AVT] allow you the use dynamic expressions virtually everywhere in the markup. They are not limited to the XForms elements, but are supported on most host language attributes. Attribute Value Templates enable even more powerful styling of your form based on the data. As an example, a form author can now easily highlight rows in a table based on certain data conditions (overdue, negative values, or complex conditions). In HTML5, this feature enables the form author to declaratively specify when certain css-classes apply to an element.

Example 2. AVT: Higlight overdue jobs

```
<xf:repeat ref="job">
 <tr class="{if (current-dateTime() > xs:dateTime(@due))
     then 'over-due' else ''}">
  ...
 </tr>
</xf:repeat>
```

2.3. Variables

Variables [VAR] make it possible to break down complex expressions into pieces and make it easier to understand the relationship of those pieces, by using expressive variable names and documenting those individual pieces and their relationships.

Variables also facilitate in de-duplication of XPath expressions in your form. In typical forms the same expression is used multiple times (e.g.: XPath expression that calculates the selected language in a multi-lingual UI).

Example 3. Variables

```
<xf:var name="paging" value="instance('paging')"/>
<xf:group>
 <xf:var name="total"      value="$paging/@total"/>
 <xf:var name="page-size"  value="$paging/@page-size"/>
 <xf:var name="page-count" value="($total + $page-size - 1)
     idiv $page-size"/>
 <xf:output value="$page-count">
  <xf:label>Number of pages</xf:label>
 </xf:output>
</xf:group>
```

2.4. Custom Functions

Custom functions [CUST_FUNC] like variables allow form authors to simplify expressions and prevent code duplication without using extensions.

Example 4. Custom Functions: Fibonacci

```
<function signature="my:fibonacci($n as xs:integer) as xs:integer">
   <var name="sqrt5" value="math:sqrt(5)"
   <result value="(math:power(1+$sqrt5, $n) - math:power(1-$sqrt5, $n))
       div (math:power(2, $n) * $sqrt5)" />
</function>
```

2.5. Non-XML data

Because XForms' data model is XML it can consume data from a lot of sources with little effort (SOAP services, XML data bases using XQuery, REST XML services, ...). Starting from XForms 2.0, XForms can natively consume JSON data [JSON]. As more and more services on the web are starting to deliver JSON today this is an important feature. XForms implementations may support other non-XML file formats like CSV, vCard, ...

The form author can use all normal XForms constructs (binds, UI controls, actions,...) independant from the data format of the external source. The XForms processor will build an XPath data model from the recieved data.

2.6. Miscellaneous improvements

Other interesting new features are:

- Model based switching/repeats allows form authors to capture the state of the switch/repeat in the model, which makes it possible to save and restore the actual runtime state of the form.
- The iterate attribute makes it much easier to execute actions for all nodes matching a certain condition.
- Ability to specify the presentation context for the result of a replace-all submission. This feature makes it possible to display the result of a submission in another frame or tab, when you using HTML as your host language.
- MIP functions (e.g.: valid()) which can be used in actions and the user interface. They can for example be used to conditionally show content based on the value of a MIP on a data node.

3. Possible features for a future version of the specification

There are a lot of possible enhancements that could be made to the current specification. Ranging from improving UI events, better expression of conditional default values, structural constraints to better integration with other web platform technologies (Javascript, geo-location, Web Storage, Crypto, ...).

But in my opinion the most important thing that is still missing in the current version of the specification, is a framework for defining custom components that can be re-used in multiple forms/applications. This section will discuss possible requirements for such a framework and a proposal of a possible custom components framework which is heavily based on the framework that is currently implemented by Orbeon Forms [ORBEON].

4. Custom Components

Custom components should be easily re-usable over multiple forms/applications. They should feel and behave the same way as native XForms UI controls. It should also be possible to strongly encapsulate their internal data and logic. The following section will go into more depth about these requirements and how they could be implemented.

4.1. Using Custom Components

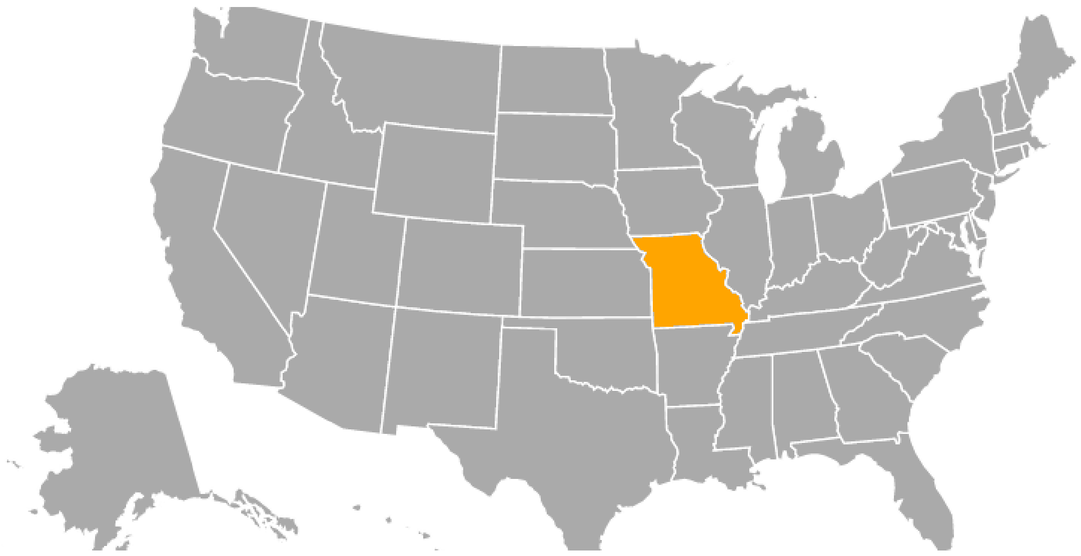
This section will discuss the aspects of a custom component that are relevant to the user of the component and demonstrate how custom controls can be used by the form author in an XForms document.

In general there are two different categories of custom components:

1. Components that are just a different appearance of an existing XForms control. An example of this is a graphical state selection component.

   Example 5. Custom appearance: Graphical state selection component

   ```
   <xf:select1 ref="state" appearance="cc:us-states">
     <xf:itemset ref="instance(‘states’)">
       <xf:label ref="@name"/>
       <xf:value ref="@code"/>
     </xf:itemset/>
   <xf:select1>
   ```

2. Advanced and/or complex user interface controls that are not an appearance of an existing XForms User Interface control. An example of such a control is a donut chart:

   Example 6. Custom element: Donut chart

   ```
             <cc:chart>
               <cc:slice-serie ref="instance('accounts')/account">
                 <cc:label value="@label"/>
                 <cc:name value="@id"/>
                 <cc:value value="@amount"/>
               </cc:slice-serie>
             </cc:chart>
   ```

As shown in the above example, the markup to use custom components is similar to the markup for native XForms controls. To use the first category of controls the form author just has to specify a custom appearance. For the second category new element names should be used, but the same constructs as for native form controls are used (ref-attribute for specifying the repeat sequence, value-attribute for the values, a structure similar to xf:itemset is used for cc:slices).

4.1.1. Container control

Some custom components, such as tabview, act like a container controls (xf:group, xf:switch, xf:repeat). Those controls have typically one or multiple insertion points, to which the custom control host’s children are transposed. The transposed children can contain any host language and XForms content, which will be visible from the “outside” (e.g.: IDs are also visible to actions in the host document outside of this custom control).

The following example creates a tab view with two tabs (Host language and xforms markup can be used under the tab elements):

Example 7. Custom Container Control: Tab view

```
<cc:tabview>
  <cc:tab>
    <xf:input ref="foo">...</xf:input>
  </ cc:tab>
  <cc:tab>...</cc:tab>
<cc:tabview>
```

4.1.2. Events

XForms uses XML events and XForms actions to execute various tasks during the form execution. The appropriate xforms events are dispatched to the custom control (e.g.: the data node related events like xforms-value-changed and xforms-reaonly are sent to the control if the control has a data binding). The form author can attach event listeners to all custom controls just like he does for native XForms controls.

The following example attaches a value change listener to the custom pie chart control:

Example 8. Events: Attach handler to Custom Control

```
<cc:pie-chart ref="account">
  <cc:slices ref="instance('accounts')/account">
    <cc:label value="@label"/>
    <cc:name value="@id"/>
    <cc:value value="@amount"/>
  </cc:slices>

  <xf:action ev:event="xforms-value-changed">
      ...
  </xf:action>
</cc:pie-chart>
```

Custom events, which can be handled by the custom control, can be dispatched to the custom control using the xf:dispatch action.

The following example dispatches an event my-event to the control with id foo-bar when the trigger is activated:

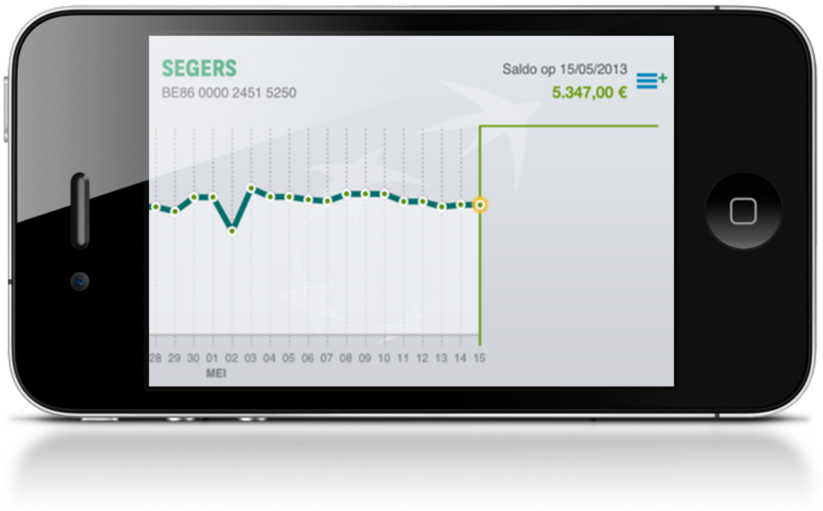
Example 9. Events: Dispatch event to Custom Control

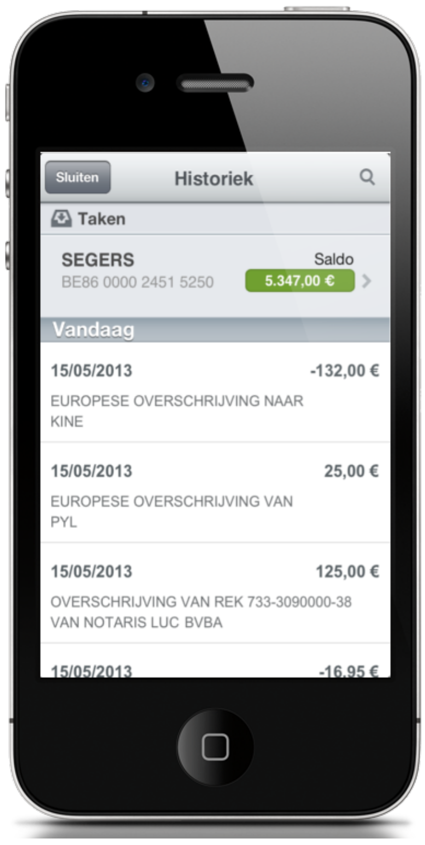
```
<cc:foo-bar id="foo-bar">
...
<xf:trigger>
  <xf:label>Do it</xf:label>
  <xf:dispatch ev:event="DOMActivate" name="my-event" targeted="foo-bar"/>
</xf:trigger>
```

4.1.3. Responsive Design

When targeting a wide variety of devices with different capabilities (screen size/resolution, mouse/touch, …) and usages, it might be desirable to change the appearance of a control depending on the used device and or environment in which it is used. Examples of this are desktop versus mobile, but also landscape versus portrait on a tablet. This is currently not supported but it is something that should be considered for the next version of XForms, and might be related to the custom controls framework.

Example 10. Responsive Design: different appearence depending on orientation





SEGERS Saldo
BE86 0000 2451 5250 5.347,00 €

Vandaag

15/05/2013 -132,00 €
EUROPESE OVERSCHRIJVING NAAR KINE

15/05/2013 25,00 €
EUROPESE OVERSCHRIJVING VAN PYL

15/05/2013 125,00 €
OVERSCHRIJVING VAN REK 733-3090000-38 VAN NOTARIS LUC BVBA

15/05/2013 -16,95 €

4.2. Creating Custom Components

Implementing a custom component is typically done using a mixture of XForms and host language specific markup. There are a lot of possibilities on how to specify this implementation. A possibility is to extend the work done for XBL 2.0, but because this specification is no longer maintained it is probably better to specify something that is a bit more tailored to the XForms requirements.

A simple custom component that just wraps an input control, has a data binding and supports LHHA (label, hint, help and alert) might look like this:

Example 11. Custom Control: Implementation

```
<xf:component xmlns:xhtml=http://www.w3.org/1999/xhtml
       xmlns:xforms=http://www.w3.org/2002/xforms
       xmlns:ev=http://www.w3.org/2001/xml-events
       xmlns:cc=http://www.sample.com/custom-controls
         id="foo-bar-id"
         element="cc:foo-bar"
         mode="data-binding lhha">
    <xf:template>
        <xf:input ref="xf:binding('foo-bar-id')"/>
    </xf:template>
</xf:component>
```

In the above example the mode attribute on the component element ensures that the custom control will support the data binding attributes (ref, context and bind) and supports the LHHA-elements.

4.3. Data and logic encapsulation

A custom component should be able to have private models to abstract and encapsulate their internal data and logic. This implies that a component can define its own instances, binds and submissions.

4.4. Event encapsulation

Events that are sent from and are targeted to elements inside the component should not be visible to the user of that component. But it should be possible to send events to, and receive events from, the user of the component.

To fulfill these requirements the elements of the custom control will reside in its own ‘shadow’ tree. But events dispatched to, and listener attached to, the root element of the custom control will be re-targeted to the host element.

4.5. Component lifecycle

Standard XForms initialization events (xforms-model-construct and xforms-model-construct-done and xforms-ready) and destruction events (xforms-model-destruct) will be sent to the models in the custom control when the custom control is created and destroyed respectively. A custom control can be created either when the form is loaded or when a new iteration is added to an xf:repeat. A custom control is destroyed when the XForms Processor is shutdown (e.g.: result of load action or submission with replace all) or if an iteration in an xf:repeat is removed.

The events are sent to the implementation of the custom control and therefore, not traverse any of the host document elements.

4.6. Styling

By default, styling should not cross the boundary between the host document and the component. In other words, the styling rules from the host document should not impact with the styling rules from the component and vice versa. But it should be possible to style parts of the custom control from within the host document that are explicitly specified as being style able by the custom controls' implementation. When using CSS as a styling language it is recommended to use custom pseudo elements, just like defined in Shadow DOM [SHADOW_DOM].

5. Conclusion

The new features in XForms 2.0 like XPath 2.0, Attribute Value Templates and variables make it easier to create web applications with XForms. The support of non-XML data sources ensures that the technology can be used to consume data from a variety of sources. One of the biggest strengths of XForms is its abstraction by declaratively defining its data processing model (dependencies , validations, calculations and data binding). But it is currently missing a standardized way for abstracting re-usable high level components, that can be used to build rich forms/applications. Hopefully such a frame is something that can be added in the next version of XForms.

References

[AVT] http://www.w3.org/TR/xforms20/#avts .

[CSS] http://www.w3.org/TR/CSS/ .

[CUST_FUNC] http://www.w3.org/TR/xforms20/#The_function_Element .

[DOJO] http://dojotoolkit.org/ .

[DOJO_DECL] http://dojotoolkit.org/documentation/tutorials/1.8/declarative/ .

[HTML5] http://www.w3.org/TR/html51/ .

[JQUERY] http://jquery.com/ .

[JSON] http://www.json.org/ .

[ORBEON] http://www.orbeon.com/ .

[SHADOW_DOM] http://www.w3.org/TR/shadow-dom/ .

[VAR] Variables: http://www.w3.org/TR/xforms20/#structure-var-element .

[XFORMS-20] http://www.w3.org/TR/xforms20/ .

[XPATH-20] http://www.w3.org/TR/2012/WD-xforms-xpath-20120807/ .