A Standards-based Virtual Machine

Author: Mark Birbeck
CEO and CTO
x-port.net Ltd.

Invited Expert on XForms and HTML Working Group

mailto:Mark.Birbeck@x-port.net
http://www.x-port.net/
http://www.formsPlayer.com/

x-port

Introduction

We need to define an open standards 'virtual machine'. The concept of a VM is of course not new - it lies at the heart of both Java and .NET, for example, as well as CPUs. However, the VM is usually insufficiently 'abstract'.

The VM we need is more a cross between the current DOM, and elements of pattern programming. But it needs to be far richer than the current DOM, expressed using mark-up rather than APIs, and needs to be centrally co-ordinated.

The VM could run on a user's desktop or a server.

Example Usage

We have an Internet Application that:

- takes an XML document from one server
- sums two of the nodes
- stores the result in a third node
- dispatches the resulting document to another server
- exits

The application looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xh2:html
 xml:lang="en"
 xmlns:xh2="XHTML 2.0"
 xmlns:css3="CSS 3.0"
 xmlns:xf11="XFORMS 1.1"
 xmlns:ev="EVENTS 1.0"
>
  <xh2:head>
    <xh2:title>Modify the data and close</xh2:title>
    <xh2:meta property="dc:creator">Mark Birbeck</xh2:meta>
    <xf11:model id="m1">
      <!--
        Initialise an instance with the source document
      -->
      <xf11:instance id="inst1" src="modify-and-close.instance.xml" />

      <!--
        Set up a submission ready to save the modified
        document
      -->
      <xf11:submission
       id="sub1"
       action="modify-and-close.instance.copy.xml"
      />

      <!--
        When the model is fully initialised:

        * set the value of one of the nodes in the instance data
        * invoke the submission
        * close the form
      -->
      <xf11:action ev:event="xforms-ready">
```

```
        <xf11:message level="ev:log">Received 'xforms-ready'</xf11:message>
        <xf11:setvalue ref="c" value="a + b" />
        <xf11:send submission="sub1" />
        <xf11:close />
      </xf11:action>
    </xf11:model>
  </xh2:head>

  <!—
    Note the empty ‘body’ ... there is no UI.
  -->
  <xh2:body />
</xh2:html>
```

Current technologies require that we build a specialist processor to handle the language used to express this application - the “when will IE/Mozilla/Opera support XHTML 2/SVG/XForms?” syndrome. However, the virtual machine approach says that we define how the instructions in our application above should be performed using a set of more basic instructions, expressed in one or more intermediate languages.

For example, the higher-level construct:

```
  <xf11:instance id="inst1" src="modify-and-close.instance.xml" />
```

could be specified as follows:

```
  <xf11:instance id="inst1">
    <xf11:action ev:event="load">
      <dom3:document id="dom1" />
      <dom3ls:load dom3:dom="dom1" href="modify-and-close.instance.xml" />
    </xf11:action>
  </xf11:instance>
```

(Note that this snippet won’t quite work since you need to put the retrieved DOM into the `instance` element’s shadow tree, but it should make the point.)

Now all we need is a virtual machine that understands how to create a DOM document:

```
      <dom3:document id="dom1" />
```

and how to load an XML document into it:

```
      <dom3ls:load dom3:dom="dom1" href="modify-and-close.instance.xml" />
```

This VM could be constructed using any language, such as Java, C++, C#, JavaScript, etc. But the clear advantage is that we can define higher-level languages much more quickly and accurately, since they must be expressed in terms of the more basic languages.

For example, the current XHTML 2 specification for <a> contains a large amount of information about relative and absolute URIs, behaviour when activated, and so on. Much of this is information that is also repeated in the SVG specification. However, we could more succinctly define the behaviour of an XHTML 2 ‘anchor’:

```
<xh2:a href="home-page.html">Home</xh2:a>
```

in terms of base features that we have defined must be provided by our VM:

```
<xf11:trigger appearance="minimal">
  <xf11:label>Home</xf11:label>
  <xf11:load
   resource="home-page.html"
   ev:event="DOMActivate"
   show="replace"
  />
</xf11:trigger>
```

In this case we have assumed that XForms is a base language. However, since VM definitions would be recursive, we can go further and define `xf11:load` in terms of XLink:

```
<xf11:load
 xlink:type="simple"
 xlink:href="home-page.html"
 xlink:show="replace"
 xlink:actuate="onRequest"
 ev:event="DOMActivate"
/>
```

From the standpoint of specification writing we have now moved all discussion about things like relative and absolute URIs out of the XHTML 2 spec.

But most importantly, we have created the ability for a processor that understands our 'virtual machine' constructs - in this case the XLink attributes, `xf11:label`, and `xf11:trigger` - to implement `xh2:a` without any specific knowledge of XHTML 2.

Key Modules

Some of the key modules of this virtual machine would be:

- an object broker module;
- a 'system' module;
- an object based around the XML infoset;
- an infoset item selection module;
- an events module;
- a 'decorator' module;
- a dynamic infoset module;
- a communications module;
- a validation module;
- a rendering module.

Architecture

Everything would be based around a core object, much like our current DOM. These objects can be initialised via URIs, and delivered to other

locations, via the communications module. The objects can also be validated.

All communication between modules would take place via events - for example when a document has just completed loading, or has become invalid - rather than APIs. Events would also allow inter-application communication, for example if an email arrives, or the user chooses an option from a system tray application.

All objects would be created by an object broker, which would make it easier to port the VM to other platforms.

A 'system' module is required to provide features such as closing an application, timers, and so on.

Perhaps the two most important modules are the decorator module and the dynamic infoset module. The decorator layer would allow the recursive definition of features required further up the hierarchy, ensuring that the VM remains completely dynamic. The dynamic infoset module allows the value of nodes to be determined on the basis of values and changes to other nodes.

The nodes being referred to by the decorator module and the dynamic infoset module would be addressed using the selection module.

Finally, a renderer object can allow interaction (if required) with a user. The renderer module would use abstract components and a styling module. Note that the state of any particular property would be maintained by the 'dynamic infoset' module, and communication with the renderer would be via events.

We would want to devise some standard renderers, for example an application's toolbar should be addressable using some well-known URI, so that it works on all platforms. As a first cut at this, we have implemented in formsPlayer the ability to load an XForms document into the Windows system tray, using the following mark-up:

```
<xf11:load
 resource="home-page.html"
 ev:event="DOMActivate"
 show="fp:systray"
/>
```

Current State-of-play

Many of the modules we would need are either already defined or are hinted at in existing/proposed W3C standards:

- DOM 3 Implementation Registry provides a very effective object broker module, although a few things need to be added;
- DOM 2 and DOM 3 provide an XML infoset object for the core;

- DOM 3 XPath provides a selection module which has the ability to add functions, although additional work is needed to hook these functions into the events system;
- DOM 2 Events provides much of the events module we need, and XML Events provides a declarative way to use it, although it is not possible to create listeners for things without IDs at the moment (such as the document object);
- XForms `bind` and CSS are very specific dynamic infoset modules which would not take much to generalise;
- DOM 3 Load and Save and XForms `submission` provide some aspects of a communications module;
- XForms has a validation module;
- rendering modules would be wrappers around objects that understand rendering languages such as HTML and SVG;
- XBL provides some aspects of a 'decorator' module, although much more is needed.

The 'system' module is the only one that would be completely new, but that is pretty straightforward.

Summary

We need a virtual machine that is defined using declarative mark-up, rather than any particular language. The processing model would rely heavily on events to communicate between the modules, and interpreting the mark-up would be recursive.

The aim would be to establish a VM that could execute as servers or clients, and so facilitate web applications as agents.