Position Paper for Origo Services Ltd on Web Applications and Compound Documents

Origo

Origo Services Ltd is the b-to-b, XML message standards body for the UK Life Insurance Industry. In 1999 it began to define its own forms markup language, and the management services necessary to support the deployment of forms based applications. Forms are executed either server or client-side.

Experience

Origo has been represented in the XForms WG for the last year, and is now working on migrating the UK Life Insurance Industry to XForms based applications. Part of this work involves looking at how aggregations of forms and other resources can be combined as applications.

Applications may consist of many discrete forms, and other resources, chained together in defined sequences.
Some of the resources in an application might be compound. It is possible that an entire application might consist of one compound document.

From our perspective, the intent of a Web application is defined in one or more XML instance that is interpreted at runtime. Clearly such an application must be hosted by an application, that in turn may call a number of components (plug-ins).

Web Application Requirements

From Origo's perspective, these are some of the most important requirements for Web Applications, in no particular order:

1. Intent based definition

The workshop is intended to focus on web applications that are executed on the client. However, experience shows that some classes of application may be executed either client or server-side, depending on circumstances.

Organisations represented by Origo often have to redevelop entire applications for multiple deployment platforms. This should not be necessary.

It is therefore important to be able to define as much of the logic, or intent of an application as possible in a deployment agnostic manner.

For the same reasons any ui controls defined should clearly separate function/intent from representation.

2. Reduced need for procedural code

Organisations represented by Origo have been attempting to reduce the need to represent application logic as procedural code for a number of years. Procedural code, in the context of Web applications, tends to be difficult to debug and maintain, lengthy, and does not lend itself to reuse.

On the other hand, declarative code often has a more obvious relationship with the logic they represent, is more concise and lends itself to reuse.

3. The ability to act upon XML data

Web applications need rule based, read/write access to XML instances.

4. Accessibility

It must be easy to define Web applications that meet legal accessibility requirements.

Web Applications and XForms

Origo feels that XForms lays a solid foundation for building a platform for Web applications. Given that all applications are sets of rules for event based transformation of data, with, or without some kind of representation of the current state of data to a human, the XForms model would appear to be useful outside the obvious context of traditional forms.

Indeed any application that requires interaction, either with a human, or not, can be considered as a form, as interaction is recorded as change to an underlying data model.

It is to be hoped that any W3C activity to define Web applications more fully will build on the work already done by the HTML and XForms Working Groups.

Compound Documents

In Origo's view there is not necessarily a difference between Web applications and compound documents. Below are thoughts on some of the questions asked in the call for participation.

Should there be a set of predefined compound document profiles (eg. XHTML Basic + SMIL Basic + SVG Tiny)?

Preferably not. It would be better to develop general methods to allow authors to make arbitrary combinations of document types as necessary.

What happens with event processing and style cascading across the boundaries of mixed content?

Regarding events, these need to be translated into events meaningful within each boundary. So, either everything understands one set of generic events, else the host environment has access to event mappings, which it manages for each document.
For styles, default behaviour should be that style cascades across boundaries. However, it should be possible for an author to block styles from cascading down.

What is needed from schema languages?

Treat differently namespaced parts of a document as separate documents for validation. It would be nice if XML languages did not have to be written specifically to allow other vocabularies to be mixed with them.
Need a compound document definition to set out what should be validated, in what order, and what to do if there are validation errors
Need to define multiple states (of validity?), and actions associated with them, in the way that XForms allows.
Bubble validation errors up through a compound document, until they handled somewhere.

How can application semantics from different markup languages be mixed in an interoperable way (e.g. using XBL)?

Something XBL-like sounds like a good approach to us.

Is there a need for a generic extension architecture? What is needed to allow extensions, such as plug-ins, to handle content that is not supported directly by the browser/host environment?

For current browsers, perhaps we need a plug-in plug-in. It should be enough that the host manage the definition of the compound document (expressed in something akin to the XForms model perhaps), and provide mechanisms for components of the document to be realised, as appropriate using plug-ins, and to interact with each other appropriately.

Given that vocabularies could be nested within each other an arbitrary number of times, could one end up with a situation where, effectively there are compound documents within compound documents.