Authoring, Deploying And Consuming Dynamic Web Applications Using Mixed-Namespace XML Documents

T. V. Raman
IBM

Abstract

This position paper outlines a mechanism by which one can create dynamic Web applications from documents authored using multiple XML namespaces. We investigate the syntax, semantics and run-time behavior of such mixed-namespace XML documents, and describe how the standardized W3C DOM2 eventing model may be used to create consistent interaction behaviors when working with such mixed-namespace documents. In doing so, we identify a number of open issues that need a standardized solution in order to author, deploy and consume Web applications using XML.

Table of Contents

1. Authoring Mixed-Namespace XML Documents

The syntactic validation rules for authoring mixed-namespace XML documents can be defined along the lines of XHTML Modularization. Issues concerning the definition of XML Schemas for such modularization profiles are presently being addressed within the HTML-WG in the context of the work on XHTML 2.0. Notice that when complete, this only defines the syntactic rules whereby different namespaces may be mixed; the runtime semantics and interaction behavior are not directly addressed by simply creating a modularized XML schema.

1.1. Open Issues

Content-Type

> What should the `content-type` of mixed-namespace documents be?
>
> `application/xml` gives too little information.
>
> Creating a `content-type` for each possible combination of namespaces causes a combinatorial explosion.

2. Loading Mixed Namespace Documents

Given a mixed-namespace document, what is the behavior at `document.load` time in an XML-aware browser? Assuming that we solve the `content-type` problem identified in the previous section, how should an XML browser discover, load and associate appropriate software components needed to consume the different namespaces being used in the document? The present-day Web has invented a few solutions to these problems in the absence of a single standardized one --- e.g., using `object` elements to associate content-specific plugins. However such solutions quickly cause XML content to become user-agent specific.

2.1. Open Issues

DOM:HasFeature

> How can one bootstrap an XML browser using DOM3 interfaces --- `DOM:HasFeature`[1]

Declaring Required Components

> How should a mixed-namespace document declare the software components that are required by a browser to process that document? Notice that depending on the user task being performed, the software component to be loaded for processing a given namespace will vary: Thus, given an XHTML+SVG document, the component that processes the [SVG] component is different depending on whether one is editting or viewing the document.
>
> View: Component creates SVG rendering
>
> Edit: Component presents an editable view

Locating Components

> How should an XML browser locate the required software components needed to consume a given namespace, and what should the fall-back behavior be when one fails to locate an appropriate component?

3. Component Interfaces

Once the browser has identified the software components needed to consume a

mixed-namespace document, these components need to be loaded and initialized. What common interfaces should a component that is designed to operate in such a framework implement?

3.1. Open Issues

Lifecycle

> Need to define the component lifecycle interfaces for software components that wish to participate in this framework.

Resource Allocation

> Components that are primarily responsible for output e.g., visual or auditory presentation, will need access to appropriate resources. Thus, an SVG or MathML component that is hosted inside an XHTML page needs access to a portion of the visual canvas; a voice output element will need access to the audio output device. Such resource access needs to be coordinated by the container. Similarly, components that handle user input, e.g., [VoiceXML] dialogs that handle voice input, or event handlers for keyboard/pen input will need access to the appropriate user input channels.
>
> Issues concerning the orchestration of visual output were identified in the W3C workshop on Component Extensions [CX].
>
> The DOM2 Events specification defines a consistent eventing model for Web documents. This eventing model can be exposed to XML authors via a consistent syntax defined in [XML-Events 1.0]. The above can be used to advantage when implementing multimodal interaction by Registering for the appropriate DOM events and relying on the DOM2 Eventing loop to dispatch events down the hierarchy. This has been shown to solve many of these issues for user input e.g.,see [XHTML+Voice (X+V)].

4. User Interaction --- Dispatching Events In Mixed Namespace Documents

In a browser that uses DOM2 Events to dispatch user interaction events down the hierarchy, components can create rich user experiences by registering appropriate event handlers during the `capture`, `target` or `bubble` phases of DOM2 Event propagation. This enables different components on a given `page` to react to a given user interaction event in a coordinated manner. As an example, consider a mixed-namespace document that uses XHTML and SVG for visual output, a set of pen-input events for capturing pen input, [SMIL] for time syncrhonization ,and VoiceXML for spoken interaction. In addition, assume that system environment changes, e.g., availability of a microphone, are signaled by raising appropriate events. Using XML-Events, an author can create DOM2 Event bindings that enable a multiplicity of rich user interaction scenarios enumerated below:

- Turn off microphone when the user starts writing:

  The containing document can listen for pen-input events, and dispatch an appropriate `microphone-off` event to the voice component. Notice that in this case the voice and pen components are unaware of one another; yet, DOM2 Event propagation enables the author of the hosting document to coordinate user interaction with these two separate modalities.

- Provide spoken confirmation of keyboard or pen input

  Consider an XHTML+XForms document where user input is available through the [XForms] data instance. Aural confirmation of keyboard or pen-input can be achieved by binding appropriate event handlers to the user-input event. When the entered value is *finalized* to the XForms Model, the event handler for producing spoken output can access the value from the XForms data-model and render it appropriately to the user. Notice that coordinating data access through the XForms model *automatically* synchronizes the information presented to the user in the visual and aural interaction modality.

- Provide visual confirmation of spoken input.

  Given an XHTML+XForms document, one can bind voice-input handlers --- voice dialogs authored in VoiceXML. Voice dialogs can be bound to individual user input controls to collect single values; alternatively, richer *mixed-initiative* VoiceXML dialogs that permit the user to specify multiple values in a single utterance can be bound to a *group* of controls. The VoiceXML handlers upon interpreting spoken input can *bind* the results into the XForms model. XForms processing automatically synchronizes the visual presentation, with the result that the user sees immediate visual confirmation of spoken input. Using the XForms data model for synchronizing across multiple views avoids the need to make the aural and visual views aware of one another; by having each view bind to a centralized model in the host document, synchronizing multiple views becomes scalable.

- Notice that synchronizing across multiple views is not specific to multimodal interaction; a rich user interface might choose to display complex data using a multiplicity of synchronized visual views, e.g. a bar-chart generated via SVG and a table of numbers generated via XHTML; by accessing the underlying data from a single centralized XForms model, these multiple views can be automatically synchronized.

5. Packaging

A complex Web application typically consists of more than a single document --- in general, an application may be made up of a collection of resources that may be thought of as multiple *content streams*. Deploying and archiving such applications requires an interoperable packaging scheme that allows for unambiguous resolution of cross-references among the component content streams. To date there is no

standardized packaging mechanism with an interoperable means of encapsulating metadata about the resources comprising such applications. In the absence of such a single universal solution, the industry has adopted defacto archival filetypes along with `Manifest` files that hold the metadata; however this leads to platform-dependent cross-referencing schemes.

5.1. Open Issues

XML Packaging

> Viewing the collection of content streams as a *forest* of XML documents might lead to a possible solution that creates an *XML Package* scheme that creates an umbrella container to holds the individual content streams and enables cross-references among the components.

XML Fragments

> As a dual to the *XML Package* approach, one might instead view each component resource as an addressable *XML Fragment* and solve the problem of cross-referencing among a collection of such fragments.

Bibliography

[xevents] XML Events. *An Event Syntax For XML*.

[cx] CX. *Component Extensions API Requirements* .

[xv] X+V . *XHTML+Voice 1.0* .

[xforms] XForms . *XML Powered Web Forms*.

[vxml] VoiceXML . *Voice Extensible Markup Language*.

[svg] SVG. *Scalable Vector Graphics*.

SMIL . *Synchronized Multimedia Integration Language*.

---

[1] Issue originally raised by Mark Birbeck at the 2004 Tech Plenary