A Framework for Coordinated Multi-Modal Browsing with Multiple Clients

Alistair Coles, Eric Deliot, Tom Melamed
Hewlett-Packard Laboratories
Filton Road
Bristol
BS34 8QZ
UK
{alistair.coles,eric.deliot,tom.melamed}@hp.com

Kevin Lansard
INSA
Département Télécommunications
20, avenue Albert Einstein
69621 VILLEURBANNE cedex
France
kevin.lansard@insa-lyon.fr

ABSTRACT
As users acquire or gain access to an increasingly diverse range of web access clients, web applications are adapting their user interfaces to support multiple modalities on multiple client types. User experiences can be enhanced by clients with differing capabilities combining to provide a distributed user interface to applications. Indeed, users will be frustrated if their interaction with applications is limited to one client at a time.

This paper discusses the requirements for coordinating web interaction across an aggregation of clients. We present a framework for multi-device browsing that provides both coordinated navigation between web resources and coordinated interaction between variants, or representations, of those resources once instantiated in the clients. The framework protects the application from some of the complexities of client aggregation.

We show how a small number of enhancements to the XForms and XML Events vocabularies can facilitate coordination between clients and provide an appropriate level of control to applications. We also describe a novel proxy which consolidates HTTP requests from aggregations of clients and reduces the burden that multi-client browsing places on the application.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *frameworks.* H.5.2 [**Information Interfaces and Presentation]**: User Interfaces - *Graphical User Interface, prototyping*.

General Terms
Algorithms, Performance, Human Factors, Languages.

Keywords
Multi-modal browsing, web proxy.

1. INTRODUCTION
An increasing number of information, communication and entertainment services are becoming accessible to both fixed and mobile internet users. These users are equipped with an increasingly diverse array of devices which will be used to access web applications in an increasingly diverse range of contexts. In response to this, the user interface to web applications is evolving to embrace other modalities in addition to the traditional visual interface: voice portals already offer a hands-free and eyes-free alternative to the visual web browser.

In contrast to today's uni-modal, uncoordinated interfaces to web applications (e.g. a visual browser, or a voice portal), the user interface to future web services will exploit multiple coordinated modalities. The exact combination of modalities will seamlessly and continually adapt to the user's context and preferences. This will enable greater mobility, a richer user experience of the web application and a more flexible user interface.

In many situations a user's interaction with a web application will be by means of a single multi-modal access device such as a desktop PC or a powerful handheld PC. (We use the term 'interaction' to describe both user initiated events within a web page such as filling form fields, and also user initiated navigation between web pages.) However, multi-modal interfaces will not only be embodied in powerful access devices, but also in aggregations of multiple access devices that are brought together to form a distributed user interface. In some cases these aggregations will include multi-modal clients (for example, a visual browser implementing support for SALT [16]) alongside uni-modal clients.

The simplest multi-device multi-modal scenario is when a user begins an interaction using a particular modality on a first access device, then ceases to use the first device and completes the interaction using a different (or same) modality on another access device. For example, a visual browser on a desktop PC might be used to obtain a map of a campus and directions to a conference room from a virtual guide application. Subsequently a voice browser might be used to access the directions via a cell phone. The interaction spans multiple devices and modalities, but each modality is used independently during temporally separated phases of the interaction.

This simple scenario illustrates a first requirement for a multi-client browsing framework: the second device should be able to pick up an interaction at the point that the first device left off. This implies that interaction state (which includes current URI and any form data that the user has entered) is persistent beyond the duration of any one client's session, and that this state is available to other clients.

In practice users may transition between clients without such a clearly defined temporal separation. In our example scenario, the user may direct the voice browser to the virtual guide application whilst still interacting via the visual browser. Once session mobility between clients is enabled, users will expect their interaction state to be consistently represented simultaneously on multiple clients during periods of temporal overlap. Indeed it

WWW 2003, May 20-24, 2003, Budapest, Hungary.
ACM 1-58113-680-3/03/0005.

would be highly undesirable for users to have to explicitly suspend one client before activating an alternative client.

When the user's experience is enhanced by the combined capabilities of multiple clients then temporal overlap of clients will continue beyond these transition periods. For example, we envisage the following scenarios:

Scenario 1: The user is able to control the navigation of a slide presentation using a visual browser on a handheld personal computer with limited display capabilities while the slide graphics (without navigation controls) are rendered by a second browser on a large area display device.

Scenario 2: An interaction started using a voice browser accessible via a cellular phone is enhanced by introducing a visual browser available at a public kiosk. For example, the voice and visual browsers may be used simultaneously to make the query to the virtual guide application ("Where are the restrooms in this *{click}* building?").

Scenario 3: A multi-modal (aural and visual) browser on a handheld personal computer is supplemented by a network hosted voice browser. The network hosted voice browser is used during periods of an interaction that require more complex speech recognition tasks beyond the capabilities of the handheld computer.

A number of challenges are presented when deploying a multi-modal user interface across multiple access devices. In addition to the requirement for persistent interaction state available to all clients, temporal overlap also demands coordination of web resource navigation across multiple clients (e.g. ensuring that all clients navigate to the same URI) and coordination of interactions with a web resource across all clients.

The remainder of this paper is structured as follows. In Section 2 we discuss our objectives and assumptions for this work. Sections 3 and 4 detail our approach to navigation coordination and interaction coordination respectively. We describe our implementation of a multi-device multi-modal framework in Section 5. Other work related to this paper is discussed in Section 6, followed by a summary and discussion of potential further work in Section 7.

2. OBJECTIVES AND ASSUMPTIONS

The goal of our work has been to develop a framework to support the temporal overlap of multiple web clients during interactions with web based applications. In doing so we have sought to relieve the application of the burden of managing multiple aggregated client connections. At the same time we have attempted to provide the application with flexible control over the coordination of clients. Instead of designing a completely new markup language for multi-device multi-modal applications, our objective is to leverage as many existing or emerging XML vocabularies and enhance them where necessary.

2.1 Navigation Coordination

When multiple clients are simultaneously rendering the same web resource, navigation to another web resource may be initiated by user interaction with any one of the clients. In some cases this navigation event should trigger all clients to navigate to the same resource in unison. However, it might be that some hyperlinks should be followed by only a subset of the clients (for example, interactions with a control panel application in one browser result in a number of different pages being rendered in another browser). These alternatives should be supported by the framework and subject to the application's control through markup language.

2.2 Interaction Coordination

Many web resources solicit user input through the use of forms. When these resources are rendered on multiple clients it is highly desirable that the current state of form data is consistent on all clients. This requires form data elements to be synchronized across the clients, but as with navigation coordination, control of the synchronization should be exposed to the application through markup language.

We do not assume that all form data is replicated on all clients. Rather, we envisage that an application may distribute a form across multiple clients such that some elements of the form appear on all clients while other elements appear only on one client and therefore do not require synchronization. For example, a form to upload an image to an application may have some data bound to text fields where the user enters textual annotations (e.g. a description of the photograph). This part of the form is rendered in a visual browser on a handheld PC. At the same time another part of the form data is bound to a specialized control that gathers image data from a digital camera. Replicating the image data on the handheld PC would consume significant bandwidth.

Beyond synchronizing form data, interaction coordination also involves reflecting user interactions with one client on other clients. Selecting a particular form control for keyboard focus on a visual browser might cause the associated spoken prompt to be generated on a voice browser. Speaking *"help"* to a voice browser might cause some text to be rendered or highlighted on a visual browser. We seek to enable a flexible approach to specifying interaction coordination through markup language.

2.3 Assumptions

A significant aspect of client aggregation is how the aggregation is formed and managed, e.g., how clients discover each other, establish secure communications and authenticate themselves. Other work (for example [9]) addresses these problems and for the purposes of this paper we assume that clients can be aggregated and that clients within an aggregation are able to exchange messages in a secure manner. The exact mechanism for this message exchange is also beyond the scope of this paper, but an example might be to use a subscribe/notify scheme based on the Session Initiation Protocol (SIP) [15].

3. NAVIGATION COORDINATION

Previously reported schemes for synchronizing the navigation of an aggregation of browsers (e.g., [4][7]) have often relied on a third party (e.g., a proxy) to respond to requests from one browser by pushing copies of the requested resource to all browsers. This requires the third party to maintain knowledge of the clients in the aggregation, i.e., it must hold state. We have sought to avoid the use of a stateful third party by having the clients coordinate their individual requests to the resource.

Furthermore, in contrast to many previous schemes, we anticipate that rather than replicating the same view of the resource in all clients, each client will potentially load a different *variant* of the same *resource*. This is a subtle but important characteristic of the web: an HTTP resource, as described by a URL, does not represent static data, but rather an entity that serves data which may vary [12]. A single URL may therefore have many variants,

and the variant returned in response to a particular request may be determined by time of day, content negotiation headers or other factors. In the current problem all clients make requests to the same resource but receive different variants of that resource according to their capabilities and modalities.

The stateful proxy may be avoided as follows. When any one client in the aggregation initiates loading of a new resource it issues an HTTP request to that resource and at the same time sends a message to other clients which stimulates further HTTP requests to the resource. Each client may include in its request a description of its capabilities, for example through the use of CC/PP [18]. The resource responds to each client with a variant of itself that may have been adapted to suit the capabilities of that client.

As it stands this scheme has two significant problems. First, the resource server receives multiple requests every time a navigation event occurs. Not only does this increase the server load, but it also requires any application accessed via the resource URI to treat non-idempotent requests (e.g., HTTP POST's) as idempotent. For example, a POST to an e-commerce website might result in an item being added to a shopping cart. When multiple clients in an aggregation perform similar POST's the results will be unpredictable unless the application is aware of the client aggregation and able to consolidate the multiple requests into a single transaction.

Second, the resource server is able to adapt its response to each client's capabilities, but it is not able to adapt for a client within the context of the aggregation. For example, the server could not tailor an HTML variant on the basis of whether or not a voice browser is also available. To achieve this, the server would again need to correlate the multiple requests and construct an aggregation profile prior to responding to any request.

3.1 Consolidating Proxy

To overcome these problems we have introduced an intermediary function to our system which consolidates HTTP requests from multiple clients into a single HTTP request that propagates to the resource server. In doing so it also constructs an aggregation profile, i.e., a description of the composition of the aggregation and the capabilities of each client within it (see Section 3.2). The resource server now receives a single request and is able to construct variants for each client contextualized for the aggregation. These variants are returned in the HTTP response and distributed to the appropriate clients by the intermediary.

Although the intermediary function could be implemented as a front end for the resource server, we have chosen to implement it as a "consolidating proxy" (Figure 1) so that the functionality is associated with the client aggregation rather than the resource. This offloads the burden of request consolidation from every resource server. The resource server remains responsible for processing the aggregation profile and generating variants adapted to the device aggregation. An important feature of the consolidating proxy is that it is stateless apart from while handling a request. The proxy retains no knowledge of the individual clients or the composition of the aggregation beyond the lifetime of a request/response transaction.

To facilitate request consolidation, clients include a number of new HTTP headers in their requests to the proxy. These headers include the originating client's unique ID and capability profiles,

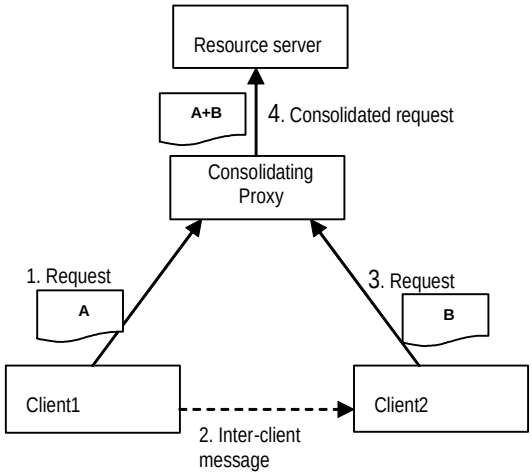

Figure 1. Consolidating proxy operation.

the ID of all other client's that will make an associated request and a unique request identifier. For example:

```
Client-ID: client1
Agg-Client-ID: client1, client2
Req-ID: abcde1234
Profile: "www.example.com/profile1"
```

These headers enable the proxy to identify requests that should be grouped together and to determine when the whole group has been received, at which point it constructs the aggregation profile from the set of client profiles and forwards a request to the resource. By including these headers in every client request we avoid maintaining persistent state at the proxy.

The consolidating proxy waits for the whole group of client requests to be received before issuing the consolidated request for two reasons. The first is that each client supplies its component profile for the aggregation profile. The second is that requests may contain form data, and as previously described this form data may be distributed across clients. When forms are distributed across clients then each client may have a fragment of the complete form data to submit within a HTTP request. (We describe later how fragmented submission can be specified by the application.) The consolidating proxy must gather form data from all client requests and include the amalgamated data in the consolidated request.

When there is no form data being submitted or when the form data is included in all client requests, we would like to avoid delaying the consolidated request. In an enhancement to our scheme, the first request to reach the proxy in these circumstances is immediately forwarded to the resource server. This implies that each client is made aware of the other clients' profiles via inter-client messaging and includes the aggregation profile in its request (this only needs to be updated when there is a change to the aggregation). In cases where that client has the complete form data, or there is no form data, its request is now sufficient for the consolidating proxy to immediately forward to the resource server. We therefore introduce another HTTP header `Request-Status` which indicates to the proxy whether a client request is complete or partial. If the consolidating proxy receives a partial request it waits for all other partial requests or a complete request to arrive before issuing the consolidated request. If a complete request is received and triggers the consolidated request, then

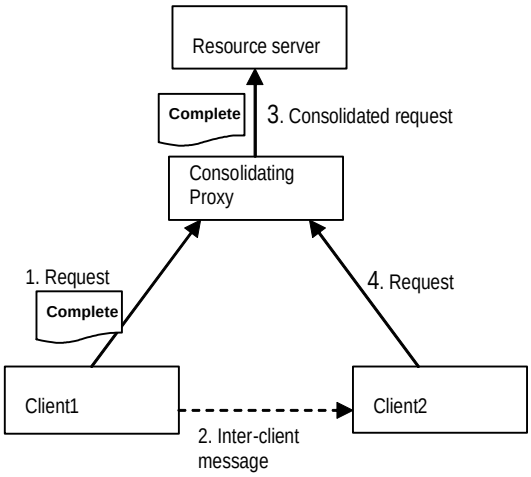

Figure 2. Consolidating proxy operation with a complete client request.

subsequent associated requests are stalled pending the response from the resource server (Figure 2).

3.2 Aggregation Profile

The aggregation profile must describe not only the combined capabilities of the aggregation but must also associate those capabilities with individual clients. For example, if the aggregation profile merely indicates that visual and aural interfaces are available then the application cannot distinguish between an aggregation comprising a visual-only client and an aural-only client, and an aggregation comprising a visual-plus-aural client and a visual-only client.

The aggregation profile should therefore include the constituent client profiles along with bindings between those profiles and the clients they match. This is achieved by the consolidating proxy first associating a tag with each unique profile. Since HTTP headers may be reformatted and consolidated by intermediaries, the tag should be associated with each component of a profile, as shown in the example below for the tag "yy". When more than one client has the same profile, only one tagged profile need be included in the aggregation profile. A Profile-Agg header may then be used to bind each unique client ID to a profile tag. This has the advantage of not repeating potentially verbose profile definitions for identical clients.

As an example, consider the case when three clients requests are received each including client specific headers:

```
Client-ID: client1
Profile: "www.example.com/profile1"

Client-ID: client2
Profile: "www.example.com/profile2",
         "www.example.com/profile3"

Client-ID: client3
Profile: "www.example.com/profile2",
         "www.example.com/profile3"
```

Note that client2 and client3 have identical profiles. From these headers the consolidating proxy is able to construct the following aggregation profile:

```
Profile-Agg: client1;xx, client2;yy, client 3;yy
Profile: xx;"www.example.com/profile1"
Profile: yy;"www.example.com/profile2",
         yy;"www.example.com/profile3"
```

This set of headers is robust to being reformatted by intermediaries with no loss of information, e.g.:

```
Profile-Agg: client1;xx, client2;yy, client 3;yy
Profile: yy;"www.example.com/profile2",
         xx;"www.example.com/profile1",
         yy;"www.example.com/profile3"
```

3.3 Response Demultiplexing

On receiving a consolidated request the resource generates a set of variants of itself adapted for each client and returns these in a multipart HTTP response. The mechanism for this is beyond the scope of this paper, but does imply enhancements to existing content adaptation frameworks to cope with aggregations and to automatically generate the new markup elements described below. Each part of the multipart response is labeled with a header that identifies the client(s) for which it is appropriate, for example:

```
--boundaryABCDE
Content-For: client1

…variant1…

--boundaryABCDE
Content-For: client2, client3

…variant2…
```

The `Content-For` header is used by the consolidating proxy to demultiplex the variants and return them in the HTTP responses to each client.

3.4 Markup for navigation coordination

To provide the application with flexible control of the navigation coordination we add a single new markup tag, `also`. This is used to modify the clients' behavior when executing hyperlink elements.

In our markup examples we use several XML vocabularies with the following namespace prefixes: `mu` for our new multi-device coordination markup, `xh` for XHTML [22] , `xf` for XForms [21] , `xl` for Xlink [24] and `ev` for XML Events [23] . For the sake of brevity we omit the namespace declarations from the examples.

An example of the use of the `also` tag is shown below:

```
<!-- in client1 -->
<xh:span id="myLink">Click here</xh:span>

<ev:listener ev:event="click" ev:target="myLink"
ev:handler="#myLoader"/>

<xf:load id="myLoader" xl:href="foo.html">
  <mu:also name="client2"/>
  <mu:also name="client3"/>
</xf:load>
```

Here XML Events markup is used to route click events on the `span` element to an XForms `load` element which loads a new resource. The `also` elements specify (using the `name` attribute) other clients with which the `load` operation should be coordinated. (By default `also` inherits the `href` attribute of its parent `load` element, but it could possibly be modified with a different `href` attribute.) When the load handler is executed in one client it triggers other clients via messages which include the URI for each client to load.

Since the unique client IDs are included in the HTTP request headers, the application has them available when dynamically generating the `also` elements. Of course, there remains the question of how this process is bootstrapped, i.e., how are the `also` elements generated by the first application to be loaded? One mechanism by which this may be achieved is to have a 'portal' web application which provides a mechanism for clients to join an aggregation and begin navigation.

A detailed description is beyond the scope of this paper, but in summary the portal would maintain a list of clients in the aggregation and is therefore always able to generate hyperlinks annotated with the appropriate `also` elements. It has the usual navigation controls (address entry field, back/forward) and also acts as a persistent store for navigation history. We are considering alternative schemes that would avoid holding aggregation related state at a portal application.

Navigation events also occur when a user submits a form. The example below shows how we enhance XForms markup to coordinate form submissions:

```
<!-- in client1 -->
<xf:model>
  <xf:instance>
    <data1>
  </xf:instance>
<xf:submission action="http://example.com/submit"
    method="post" ref="data1" id="subInfo1"
    mu:status="partial"/>
</xf:model>

<xf:submit submission="subInfo1">
  <mu:also name="client2" submission="subInfo2"/>
</xf:submit>

<!-- in client2 -->
<xf:model>
  <xf:instance>
    <data2>
  </xf:instance>
<xf:submission action="http://example.com/submit"
    method="post" ref="data2" id="subInfo2"
    mu:status="partial"/>
</xf:model>

<xf:submit submission="subInfo2">
  <mu:also name="client1" submission="subInfo1"/>
</xf:submit>
```

In this example the form submission is triggered by the user activating the submit control (declared by the XForms `submit` element). The details of form submission (HTTP method, which data should be submitted) are provided by the XForms `submission` element. We use the new `also` tag again to modify operation of the `submit` element by specifying that the form submission should be coordinated with another client. In the example the `also` element has its own `submission` attribute so that a different `submission` element may be specified for each client. This allows each variant to specify different elements of form instance data that should be submitted by its host client when a form is fragmented across clients. We have also added a new attribute, `status`, to the `submission` element that indicates whether the submitted form data is partial or complete. In the example above client1 submits the value of the `data1` element and client2 submits the value of the `data2` element. Both client requests are partial.

As with a load operation, the client initiating the submission uses inter-client messages to instruct other clients to submit and includes in those messages the identity of the appropriate `submission` element for each variant. Other clients then locate that `submission` element in order to construct their individual HTTP requests.

3.5 Inter-client locking

Any navigation event, whether it be a form submission or following a hyperlink, should occur atomically across all clients. Given finite latency in messages passing between clients, it might be possible for a user to activate one navigation operation on a first client and then activate a different navigation operation on another client, resulting in unpredictable outcomes as both clients attempt to trigger associated requests from the rest of the aggregation (not to mention a potential deadlock at the consolidating proxy). This is prevented by having a token scheme operate within the aggregation, such that only the client with the token may initiate a navigation event. Such schemes are well known [13] .

The client on which a navigation operation is initiated uses the inter-client messaging to request ownership of the navigation token. Once all other clients involved in the navigation have responded with messages agreeing that the first client can proceed with navigation, it then instructs the other clients to issue HTTP requests, and issues its own HTTP request. Any navigation operation initiated on another client during this process is blocked until the token is released by the first client, preventing the possibility of two unrelated sets of HTTP requests being generated. Before releasing the token the first client instructs all other clients to abort pending requests (again ensuring atomicity).

As with any distributed token system, the failure of one client may result in the token being "lost". To guard against this a timeout may be applied which, if reached, would alert the user and offer to remove the unresponsive client from the aggregation.

The need for atomic navigation events caused us to adopt the markup constructs described above which offload the details of navigation coordination to the client browsers. We considered an alternative scheme in which application events (e.g. "click", "submit") are explicitly passed between clients subject to markup declarations. For example, a click event could be routed to separate `load` handlers in each client, with each handler independently loading the new resource. However, there would be no way to guarantee atomic navigation in such a scheme without introducing far more complex markup.

A further benefit of offloading the navigation coordination to client browsers is that we are able to retain the ability for browsers to cancel loads in progress (e.g. using the "Stop" button), and again to ensure that the cancellation is atomic across all clients.

3.6 Summary of navigation coordination

In summary, a typical navigation operation proceeds as follows:

1. User selects (clicks) a hyperlink.
2. If load handler specifies other clients with also tag, client negotiates for navigation lock.
3. Client issues HTTP request to proxy and instructs other clients to do likewise.
4. Proxy consolidates requests, constructing aggregation profile and merging any form data.

5. Resource server dynamically generates variants for each client based on aggregation profile and returns in a multi-part response.
6. Proxy separate de-multiplexes multi-part response to individual clients.

4. INTERACTION COORDINATION

Coordination of interactions with multiple variants of a resource may be achieved by extending the events and event handler model that underpins "dynamic HTML" across an aggregation of clients. A good basis for this is XML Events [23], which provides a flexible means to declare the routing of events from event targets to event handlers within and between XML documents.

To illustrate the use of XML Events we consider the case of coordinating visual and aural interfaces to a form. The desired effect is for the user to be able to click (or otherwise select) a form field in a visual browser and to have the corresponding spoken prompt rendered by the speech browser. To simplify the illustration we assume that both browsers support modality independent XForms declaration of the form, but it should be noted that in principle the same result can be achieved using combinations of XHTML, VoiceXML [20] and SALT. The required markup is shown below:

```
<!-- client1, visual -->
<xf:input id="visualIp" xf:ref="data1">

<ev:listener ev:event="focus" ev:target="visualIp"
ev:handler="#myActions"/>

<xf:action id="myActions">
  <mu:fork mu:to="client2" ev:handler="#myFocus"/>
  <mu:fork ev:handler="#localHandler"/>
</xf:action>

<!-- client2, aural -->
<xf:setFocus id="myFocus" control="auralIp">
```

In this example, when the focus of the visual browser is moved to the `input` element, the `focus` event is fired and is routed to a handler in the resource variant hosted by client2, the aural browser. In response to the event the handler in client2 - an XForms `setFocus` element - sets the aural focus to the corresponding aural form control.

To facilitate the event routing between clients we have introduced a new event handler tag `fork` which provides a generic means to asynchronously route a copy of any event to another variant of the resource. It can also route the event to a local handler in which case the `to` attribute may be omitted. Whilst XML Events already provides for event handlers to be addressed in different documents through the use of URI's, there is no provision for routing events to handlers in different variants of the same document. This is presumably due to there being no clearly defined method for addressing variants of the same resource [12]. We therefore utilize the unique client ID as a means to address variants of a document and specify this as an attribute of the `fork` element. The introduction of the fork tag allows us to minimize inter-client event traffic: only those events with a declared remote handler are distributed. Combined with the fact that inter-client events are generated at the rate of human interaction, this means that the inter-client traffic does not represent a significant network load (which might also be confined to an ad-hoc personal area network).

Events are distributed by serializing the local event object and sending it to the receiving variant. Sufficient contextual information is embedded in the serialized event to avoid the need for callbacks to the event generating variant. For example, in a `focus` event we include the identity of the event target, which allows a generic remote focus event handler to modify its behavior on that basis.

4.1 Form Data Synchronization

In some circumstances the interaction coordination should be symmetric amongst clients. A good example of this is when elements of form data are replicated in several variants of the resource: changes to those elements in one client should be reflected in all other clients. We considered applying the generic event routing solution described above, for example:

```
<!-- client1 -->
<xf:instance id="myData">
  <dataX/>
</xf:instance>

<ev:listener ev:event="xforms-value-changed"
ev:target="myData" ev:handler="#publish"/>

<xf:action id="publish">
  <mu:fork mu:to="client2" ev:handler="#change"/>
</xf:action>

<xf:setValue id="change"/>

<!-- client2 -->
<xf:instance id="myData">
  <dataX/>
</xf:instance>

<ev:listener ev:event="xforms-value-changed"
ev:target="myData" ev:handler="#publish"/>

<xf:action id="publish">
  <mu:fork mu:to="client1" ev:handler="#change"/>
</xf:action>

<xf:setValue id="change" />
```

In this example when the instance data element `dataX` changes in either variant, an `xforms-value-changed` event is fired which is routed by the `listener` element to a publish handler. The handler contains a `fork` element which causes the event to be distributed to the other client where it should be routed to a change handler. The change handler is an XForms `setValue` action element which sets the local copy of the data to the new value. We assume that the change handler is capable of querying the event object for the data element path and the new value.

This approach presents two problems. First, change events will "rebound" to the originator due to the symmetric nature of the markup – execution of the "change" handler will trigger a local `xforms-value-changed` event that will trigger the "publish" handler. Second, due to event latencies, the scheme does not guarantee consistency across all clients when more than one client invokes its "publish" handler at or close to the same time.

The second problem can be solved by imposing universal event ordering or time-stamping. The first problem is harder to solve without introducing more complex markup constructs. Recognizing that data synchronization will be a commonly used feature we have therefore again chosen to offload this function to the client browsers and avoid further cluttering the application markup. The resulting markup is simply:

```
<!-- client1 -->
<xf:instance id="myData">
  <dataX/>
  <dataY>
    <dataZ/>
  </dataY>
</xf:instance>

<mu:also ref="dataX" name="client2, client3"/>
<mu:also ref="dataY/dataZ" name="client3"/>


<!-- client2 -->
<xf:instance id="myData">
  <dataX/>
</xf:instance>

<mu:also ref="dataX" name="client1, client3"/>
```

In this context the `also` element instructs the browser to synchronize the referenced data elements with their counterparts in other variants. We use the inter-client messaging subsystem to publish "data-change" events containing sufficient context for remote clients to update their copies of the data, i.e., the path to the data item within the instance data structure and the new and previous values. Given the possibility of quasi-simultaneous changes to data on multiple clients, consistency is ensured by including either a time-stamp or globally unique event number in each event so that all clients have the same view of the order of data changes. Note that this can be achieved without negotiating a lock and therefore does not degrade the responsiveness of the client with which the user is interacting.

Note that there can be more than one `also` element referring to sub-trees of the instance data that are to be synchronized with different subsets of the client aggregation. In fact this example illustrates one of the reasons we have been motivated to use XForms: the declaration of form data as structured XML allows us to easily bind the `also` element to the data, with the flexibility of binding to individual elements, sub-trees or the whole instance data structure.

4.2 Data locking

Some events may invoke handlers whose operation is dependent upon the state of the resource. For example, a handler that calculates the total amount for an online shopping cart will depend upon the quantity and prices of the items in the cart. As with any other distributed system it is necessary to "lock" elements of data during the execution of code with dependencies. We have therefore added a `lock` attribute to event handlers which specifies instance data that should be locked before the handler executes. Client browsers cooperate to lock the instance data (i.e., prevent any write operations) whenever a handler with a `lock` attribute is invoked. For example:

```
<xf:instance id="myData">
  <dataX>
</xf:instance>

<mu:also ref="dataX" name="client2, client3"/>

<xf:action mu:lock="dataX">
  <!-- some handlers/scripts -->
</xf:action>
```

This combination of the lock attribute and the `also` element that is bound to the specified instance data contains enough information for the client to negotiate a lock on the specified data when the handler is invoked. The mechanism by which a lock is obtained is similar to that used by clients to negotiate a token for initiating navigation operations.

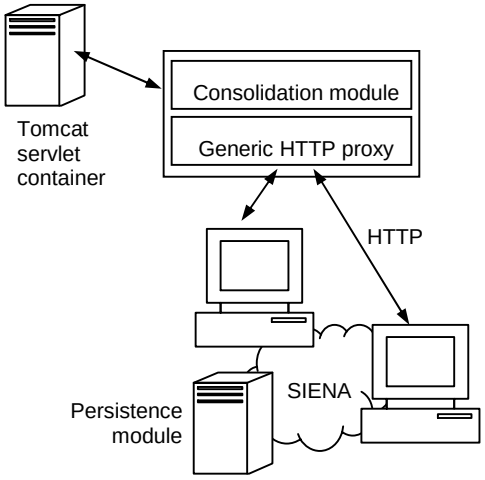

Figure 3. Prototype system

5. PROTOTYPE FRAMEWORK

We have developed a prototype system to demonstrate aspects of the multi-device browsing framework (Figure 3). This provided valuable experience that shaped the design of the new markup language tags.

We have implemented a number of clients that provide some of the navigation and interaction coordination features described above, including support for XForms declaration of forms. For visual browsing we have enhanced a Mozilla web browser and also implemented a limited set of capabilities for a Windows CE version of Internet Explorer running on a handheld PC. We have also integrated navigation and interaction coordination support into a proprietary VoiceXML browser. Combinations of these clients are capable of fully synchronized form filling and navigation. The clients communicate using the Siena [5] message passing system.

We have also built a prototype consolidating proxy which interoperates with our Mozilla based clients and a rudimentary application server. The consolidating proxy performs request consolidation, form data amalgamation and response de-multiplexing. The prototype system also has support for persistence of user sessions beyond the duration of any one client session: a persistent set of bookmarks is available to all clients, and form data is automatically reloaded when clients return to a previously visited resource.

6. RELATED WORK

The coordination of navigation across a number of browsers has been reported in the context of collaborative web browsing (e.g. [4][7]). The goal has usually been to replicate identical web pages on browsers belonging to several users. CoWeb [8] went one step further and enabled collaborative HTML form filling by replacing HTML elements in the source document with Java applets that communicated via a central control unit. WebSplitter [7] introduces the notion that each client might receive a different version of the resource according to its capabilities and other policies, but does not provide coordinated form filling across multiple clients.

In Websplitter, a single device-independent variant of the resource is adapted for specific clients on the basis of rules described in an associated policy file. Device independent authoring (DIA) is essential for enabling web application deployment across multiple clients, and other approaches have also been proposed (see [3] for a review). However, the focus of our work has been not on DIA itself but on providing a framework for simultaneous multi-client browsing that can be exploited by DIA.

Recently there has been considerable interest in multi-modal web browsing (e.g. [2][14][19]) and others have proposed architectures for multi-device multi-modal browsing (e.g. [1][10][11]). XForms is proposed as a tool for synchronizing form data in [10] but no details are given of markup language extensions to support synchronization. The architecture described in [1] differs from ours in that each modality or browser is assumed to access different resources rather than variants of the same resource. Resources for each modality are associated with each other using cross-referencing tags and there appears to be no provision for these resources to be dynamically adapted in response to an aggregation profile.

The need for interaction state persistence beyond the duration of interactions with any one client has been identified in [17] where a session state is persistently stored in response to user stimulus. In our prototype state persistence is a continuous background task not requiring user action.

Finally, the CATCH 2004 [6] project appears to have developed a multi-modal browser architecture, although at the time of writing we have not been able to access a detailed description.

7. SUMMARY AND FURTHER WORK

We have described a framework for multi-device multi-modal browsing. By enhancing the browsing infrastructure and introducing a small number of new markup tags we have enabled applications to simply coordinate user interaction and navigation across a client aggregation. We have found XML Events and XForms to be a good basis for the framework. Much of the required coordination is triggered by events and XML Events already provides most of the functionality required to route events between variants of a resource and to specify handlers for those events. The separation of instance data from user interface in XForms allows easy binding of multi-client synchronization functions to the data. The novel functionality of the consolidating proxy allows coordinated navigation without the need for a stateful third party (the proxy only maintains state during a request/response transaction), and also enables forms to be split across multiple clients for greater flexibility in user interface design.

Many of the concepts, including the consolidating proxy, have been implemented in our prototype multi-device browsing system which demonstrates the viability of web access via client aggregations. Some aspects of the framework remain to be implemented such as the inter-client locking and description of aggregation profiles based on CC/PP. We are investigating alternatives to the HTTP multipart response for returning multiple variants from the resource server. The framework does not currently support HTTPS and we recognize that SSL will significantly affect the consolidating proxy behavior. We would like to consider the implementation of inter-client message passing based on other event transport protocols such as SIP. We have yet to assess the performance of the system, in particular the user experience with varying degrees of network delay between clients.

As a generalization of our XForms instance data synchronization it would be interesting to explore the linking of any parts of resource variants e.g. synchronizing HTML nodes and sub-trees such that any modifications such as re-writing of a text node are reflected across multiple clients. XUpdate [25] may prove a useful tool for this. One important topic we have not yet addressed is server-side adaptation of content in response to aggregation profiles. This is likely to be significantly more complex than adaptation to a single client capability profile.

8. ACKNOWLEDGMENTS

We are grateful for illuminating discussions with Aled Edwards, Jim Gettys and Mark Butler of HP Laboratories.

9. REFERENCES

[1] Amann, N., Hue, L., Lukas, K. Position Statement for Multi-Modal Access. World Wide Web Consortium Multimodal Interaction Activity. See http://www.w3.org/2002/mmi/2002/siemens-26nov01.pdf

[2] Beckham, J., Di Fabbrizio, G., Klarlund, N. Towards SMIL as a Foundation for Multimodal, Multimedia Applications. Eurospeech 2001, 1363-1367, Aalborg, September 2001.

[3] Butler, M., Giannetti, F., Gimson, R., Wiley, T. Device Independence and the Web. IEEE Internet Computing, Vol. 6, No. 5, September 2002, 81-86.

[4] Cabri, G., Leonardi, L., Zambonelli, F. Supporting Cooperative WWW Browsing: a Proxy-based approach. 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira, Portugal, 1999, pp 138-145.

[5] Carzaniga, A., Rosenblum, D.S., Wolf, A.L. Design and Evaluation of a Wide-Area Event Notification Service. ACM Trans. Computer Systems, 19(3):332-383, August 2001.

[6] CATCH 2004. See http://www.catch2004.org.

[7] Han, R., Perret, V., Naghshineh, M., WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. ACM CSCW'2000 Conference Proceedings, Philadelphia, Dec. 2000.

[8] Jacobs, S., Gebhardt, M., Kethers, S., Rzasa, W. Filling HTML Forms Simultaneously: CoWeb - Architecture and Functionality. Proceedings of the Fifth International World Wide Web Conference, May 1996.

[9] Kumar, R., et al. User-Centric Appliance Aggregation. HP Labs Technical Report HPL-2002-277, October 2002.

[10] Maes, S. A Single Authoring Programming Model: the Interaction Logic. Proceedings of the IEEE Symposium on Applications and the Internet, Nara City, January 2002.

[11] Maes, S., Raman, T.V. Multi-modal Interaction in the Age of Information Appliances. Proceedings of IEEE ICME 2000, New York, July 2000.

[12] Mogul, J. Clarifying the Fundamentals of HTTP. WWW2002, Honolulu, May 2002.

[13] Mullender, S. (ed). Distributed Systems. ACM Press, New York, 343-346.

[14] Nitta, T., et al. XISL: An attempt to Separate Multimodal Interactions from XML Contents. Eurospeech 2001, 1197-1200, Aalborg, September 2001.

[15] Roach, A. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, IETF, June 2002.

[16] SALT Forum. Speech Application Language Tags (SALT). See http://www.saltforum.org/spec.asp.

[17] Song, H., Chu, H., Kurakake, S. Browser Session Preservation and Migration. WWW2002, Honolulu, May 2002.

[18] World Wide Web Consortium. Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. See http://www.w3.org/TR/CCPP-struct-vocab, March 2001. W3C Working Draft.

[19] World Wide Web Consortium. Multimodal Interaction Activity. See http://www.w3.org/2002/mmi.

[20] World Wide Web Consortium. Voice Extensible Markup Language (VoiceXML) Version 2.0. See http://www.w3.org/TR/voicexml20, April 2002. W3C Working Draft.

[21] World Wide Web Consortium. XForms, Version 1.0. See http://www.w3.org/TR/xforms, August 2002. W3C Working Draft.

[22] World Wide Web Consortium. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition). See http://www.w3.org/TR/xhtml1, January 2000. W3C Recommendation.

[23] World Wide Web Consortium. XML Events, An Events Syntax for XML. See http://www.w3.org/TR/xml-events, August 2002. W3C Working Draft.

[24] World Wide Web Consortium. XML Linking Language (XLink) Version 1.0. See http://www.w3.org/TR/xlink/, June 2001. W3C Recommendation.

[25] Laux, A., Martin, L. XUpdate Working Draft. See http://www.xmldb.org/xupdate/xupdate-wd.html, September 2000.