DECLARATIVE PUSH ON WEB

Mikko Pohja
Department of Media Technology, Helsinki University of Technology
P.O. Box 5400, FI-02015 HUT, Finland

Keywords: Push technology, REX, SSE, Comet, XForms, XBL.

Abstract: Push technology is an essential component of modern WWW applications. With the ability of sending relevant information to users in reaction to new events, enables highly interactive applications on WWW. What has been present in desktop applications for decades is only now coming online. Actually, the push is usually emulated using the pull technology, since, with the HTTP protocol alone, it is not possible to make a real push. A declarative approach widely used on WWW has not yet adopted for the push methods. In this paper are reviewed four declarative methods for the push-update on WWW. The methods are defined by combining existing and upcoming Web technologies. The scope of this paper is a targeting the update and modifying the document on client-side. To evaluate the methods, a use case is designed and implemented with all the methods.

1 INTRODUCTION

Modern WWW applications are based on information push and asynchronous updates in addition to the traditional information pull. Examples of such applications include chats, stock tickers, news services etc. The common factor for all these is a server's ability to push new information on to user's screen whenever it appears on the server. Often, the document is modified partially, i.e., only the changed parts are updated. That has brought WWW applications closer to desktop applications.

Even though, there are already numerous of these highly interactive WWW applications, the push technology is not really sophisticated at the moment. In reality, the information push is emulated using information pull by polling at constant time intervals or keeping the connection open until next update is available. Some research-oriented push systems (Kanitkar and Delis, 1998) and (Xin et al., 2005) and push-pull combinations (Deolasee et al., 2001) are introduced in the literature, but they are not widely adopted.

The server push is comprised of the following five phases (cf. Figure 1): 1. Data changes on server, 2. Application server is notified, 3. Update event is created, 4. Event is delivered into the client, and 5. User Interface (UI) is modified. The operations are independent from each other. That is, they can be realized by totally separate components. When a data changes for instance in a data base, an application server must notified. An example solution is discussed in (Bry and Pătrânjan, 2005). The application server creates the update event according to the change. The update event, which is expected to be in XML format in this paper, can be delivered for the user agent in several ways. The delivery is independent from the actual Document Object Model (DOM) modification method. However, it is noteworthy that the traditional WWW architecture using HTTP connections must be augmented somehow to achieve the server push.

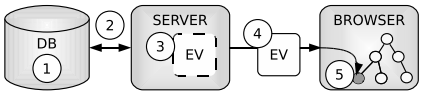

Figure 1: Server push phases.

This paper studies options to make the server push for Web documents. The focus is in the final DOM modification method and how it is defined, i.e., the phase 5 in the Figure 1. The modification phase can be further divided into two independent phases: targeting the event to the correct element and the actual UI modification. In the current systems, the both operations are implemented by ECMAScript (ECMA, 1999). It provides diverse means to control those operations. However, the trend on WWW has been to define declarative technologies to replace the most used ECMAScript functionality. That can be done also for the push-updates. There are methods to define both the targeting and the UI modifications in a declarative manner. In addition, some of them enable an automatic evaluation of the content of an update.

Pohja M. (2008).
DECLARATIVE PUSH ON WEB.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 201-207
DOI: 10.5220/0001525002010207

The push methods reviewed in this paper are combined from existing Web technologies. The UI semantics are provided by XHTML, CSS, XForms (Dubinko et al., 2003), or XML Binding Language 2.0 (XBL) (Hickson, 2007) or combinations of them. Remote Events for XML (REX) (Berjon, 2006) is used for the targeting the updates. The methods are evaluated by implementing the use case described in the next Section. The main contributions of this paper are the following:

- Four declarative push methods for WWW documents are defined.
- A use case is designed and implemented with all the four methods.
- The declarative push methods are evaluated based on the use case implementations.

The paper is organized as follows. After the use case description, a background to the topic is given. Sections 4 and 5 discuss the push-update methods and the implementations, respectively. The methods are evaluated in Section 6, a discussion is presented in Section 7 and Section 8 concludes the paper.

2 USE CASE

Consider the following airline scenario as a use case for push-updates. Joe is embarking on a business trip. He is currently driving to the airport. His mobile terminal beeps and he receives a message, which is opened by the browser on his terminal. The message notifies him that the plane is leaving on time and asks if he wants to check in for the flight.

When Joe has confirmed the check-in, he receives an electronic boarding pass. The boarding pass includes information on the flight, such as the seat, the gate, and the boarding time. The boarding pass is a Web document, which can receive updates about the flight status.

At the airport, Joe's mobile terminal beeps again. The electronic boarding pass alerts him about the new gate number. Later, the airline has published a new departure time for his flight. Joe is proactively notified by the electronic boarding pass about the delay and the new departure time. Finally, Joe is notified that the boarding has started and that he should now proceed to the gate.

Since the flight was delayed, Joe will miss his connecting flight to the final destination. The airline disruption management center is looking for an alternative routing for him. During the flight, Joe's mobile terminal is connected to the airplane's WLAN network. On the flight, he receives another notification. It proposes a new routing for his trip. When he accepts the routing, he will receive a confirmation about the traveling arrangements including the flight numbers and the departure and the arrival times.

The above scenario introduces an airline system, which differs considerably from the legacy systems. The fundamental difference is the ability to push information to the travelers' Web browsers in their mobile terminals. The scenario introduces two main information types that are pushed for the travelers: whole documents and piece of information that augments the existing document. Figure 2 gives examples of both of those types in this scenario.

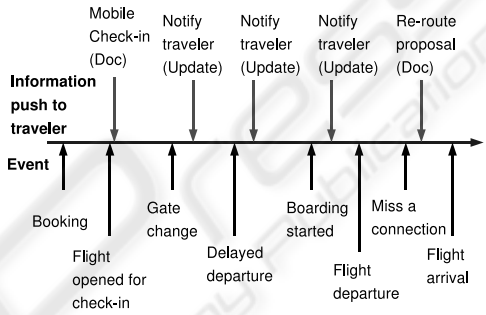

Figure 2: Push-updates in the airline scenario.

3 BACKGROUND

This paper studies options to make the server push for Web documents. The focus is in the final DOM modification method and how it is defined. That is, the whole document push mentioned in the previous Section is out of scope. It could be realized by pushing the URL of a document or the whole document. In this Section is given a background what is needed to realize the server push on WWW. In Figure 3 is depicted a legacy Web application architecture and the additions it needs to support the server push. The Eventing Service reacts on changes and notifies the Notification Service. That, in turn, creates the events and pushes them to the user agent either directly or via Web Server. That is discussed in detail in next the Subsection. Further below is explained the concept how to target the DOM event for correct node in the document to enable the modification.

3.1 Server Push

The HTTP connection provides information only through a pull method. That is, a user agent asks for the information from the server. Even if the content changes, it is not delivered for the user agent unless it asks for it. Server push is possible to implement with

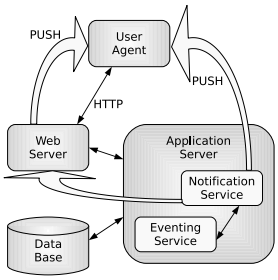

Figure 3: Legacy Web application architecture augmented with the server push components.

a privileged code and it can be emulated in many ways.

At its simplest, Server push can be emulated by polling the server at a certain time interval. That can be done for instance by reloading the document or with Ajax (Garrett, 2005) by asking incremental updates. That causes a lot of extra net traffic, especially the reloading, and there is always trade-off between the latency and the polling frequency.

Comet (Russell, 2006) combines two existing frameworks to emulate the push. It uses Ajax to realize asynchronous updates and Multipart MIME type (Levinson, 1997) for long-lived HTTP connections. With Multipart, Ajax updates can be sent whenever an event occurs on server-side. The downside is that the server must keep connections open to all clients it is to update. The central server must be able to distribute the communication properly (Resig, 2006).

HTML 5 specification (Hickson and Hyatt, 2007) defines an approach called Server Sent Events (SSE). With SSE, an author can define a source, from which the browser is listening for the incoming connections. This technique requires privileged code in a user agent and it is not widely implemented in the current browsers. The HTML 5 specification is at work-in-progress stage at World Wide Web Consortium (W3C) at the moment.

Another real Server push is achieved using a Server-Centric Interaction Architecture (SCIA). It uses Session Initiation Protocol (SIP) to establish a connection between the client and the server. The client still fetches the Web documents through HTTP but receives push-updates for the documents via SIP. In this paper, SCIA is used to deliver the updates for the client. SCIA was developed as a part of WeSAHMI[1] project. An article about SCIA is being prepared at the time of writing.

[1]WeSAHMI Project, http://www.tml.tkk.fi/Research/wesahmi/

3.2 Remote DOM Events

In addition to the delivery of the update, one needs a method to target the event to a node and define the modification. That can be done in a declarative manner or via a scripting language. An XML update language called XML-RL is represented in (Liu et al., 2003). It treats the XML document as a data base and provides methods to update it according to the changes. However, it does not provide means for the remote update. Another declarative proposal is REX. W3C has produced a working draft of it, but at the present moment has finished the specification work because of patent issues[2]. In spite of all, REX represents the declarative concept to describe the event. In this paper, that concept is called Remote DOM Events (RDE).

3.3 XBL

XBL provides methods to enhance DOM nodes and alter their sub trees. That is realized by attaching functions, event handlers, style declarations, and sub trees into the nodes. The complements are called bindings. XBL is designed to augment the user experience of a document and not be a primary method to describe the semantics. That is, the default UI semantics must be defined by some other technology.

3.4 XForms

XForms 1.0 Recommendation is the next-generation Web forms language, designed by the W3C. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative mark-up to describe the most common operations in form-based applications (Cardone et al., 2005). It can use any XML grammar to describe the content of the form (the instance data). Thus, it also enables to create generic editors for different XML grammars with XForms. It is possible to create complex forms with XForms using declarative mark-up, without resorting to scripting.

4 PUSH-UPDATE METHODS

The push-update methods consist of three operations, which are mostly independent from each other and even their order may vary depending on the used technologies. The operations and the implementation options are:

[2]Report of the REX PAG, http://www.w3.org/2006/rex-pag/rex-pag-report.html

Delivering the Update. An update must be pushed to the client whenever it occurs on a system. Current browsers support Comet-method; other proposals include SSE and SCIA architecture.

Targeting the Update. When an update arrives on client-side, there must be a way to target the new UI mark-up for correct elements in the existing document. In addition to RDE, that is possible also with ECMAScript.

Adding the UI Semantics. By default, a system provides its updates in a raw data, which has to be transformed into UI declarations. That can be done on both server and client side. On server-side, there must a dedicated component for that, whereas on client-side the transformation can be realized by CSS, XForms, XBL, or ECMAScript.

This paper focuses on the last two operations of the push-update and expects that whatever the update delivery method is, the payload is in XML format. Once the update is received, ECMAScript can be used to target it to the correct element and give the UI semantics. However, this paper focuses on the declarative technologies. By combining the technologies mentioned above, we get the following options to realize the push-update on the Web documents: RDE combined with a User Interface Markup Language (RDE+UIML), RDE+XForms, RDE+CSS, and RDE+XBL. All these are discussed in the following Subsections and summarized in Table 1.

Table 1: The push-update method summary.

Update Method	Data Transform	UI Semantics
RDE+UIML	On Server	UIML
RDE+XForms	On Client	XForms
RDE+CSS	On Client	CSS
RDE+XBL	On Client	CSS+XBL

4.1 RDE+UIML

In RDE+UIML, the data is transformed into a UIML on server-side. That is, if the document to be updated is, for instance, in XHTML or in Scalable Vector Graphics (SVG) format, the content of the RDE event is respectively a piece of XHTML or SVG mark-up, too. In this method, there needs to be only RDE interpreter on client-side, which targets the event to the correct node. On server-side, there must be a component which can transform the data into the UI mark-up. It must also be aware which elements the mark-up will replace or modify on client-side. That can be automated by retaining the client-side document also on the server-side, even if it was created by a Servlet, PHP, or any equivalent technology. When something changes which affect to the document, an XML differencing (XML diff) (Lindholm et al., 2006) is applied. Its result is automatically in the client-side format and can be sent to the client within RDE.

4.2 RDE+XForms

The RDE+XForms method is based on the XForms technology, which enables to use the Model-View-Controller paradigm in the Web forms. The benefit is that the data of a form is separated from its presentation. In other words, the data can exist in same form in both the client and the server-side. The outfit of the data is defined in an XForms form. To update the XForms instance data with RDE events, the RDE interpreter must update the instance document instead of the UI document and call the *refresh* method for the instance in the end. After the refresh, the update appears on the UI. The relation of XForms instance data with the actual UI DOM is represented in Figure 4. As can be seen, the instance is completely separate from the UI. There is a two-way connection between an instance node and its UI node.

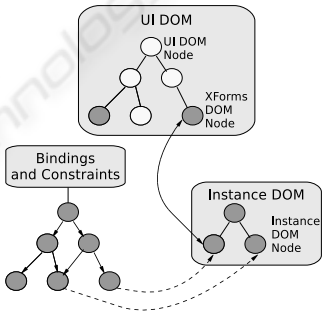

Figure 4: The XForms instance in relation to the UI document and the form constraints.

XForms can be used also to automatically evaluate the content of the update. The XForms *bind* element can retrieve the content and make calculations on it or compare its value to other values in the instance. The result can be made visible on the UI, too. Figure 4 depicts also a dependency graph through which the XForms engine control the values of the data instance. The bind elements affect to the graph among the others.

4.3 RDE+CSS

RDE+CSS is used in a similar way as RDE+XForms. Using CSS, the whole document on the client-side or part of it is a sort of data instance. CSS role is to define UI semantics of the data elements. It cannot provide automatic evaluation of an update.

4.4 RDE+XBL

RDE+XBL is an extension to the RDE+CSS method. XBL can be used to define the UI semantics a lot more diverse than CSS. With XBL, the data instance can be transformed into the desired UIML (e.g., XHTML or SVG). Nevertheless, to not to abuse XBL, it cannot be used to primarily define the semantics of the elements in a user agent. That is why CSS is used to describe the default semantics of the data instance in this paper. In addition to the UI semantics, XBL can define functions for the updated element. The functions can evaluate and react to the updated content like the XForms *bind* element. To react to the updated data instance element, an *xblEnteredDocument()* function must defined within the XBL *implementation* element.

5 USE CASE IMPLEMENTATIONS

The push-update methods were tested by implementing the use case described in Section 2. SCIA architecture was used to create the updates and notify the application server. In Figure 3, SCIA locates in the application server. An open source XML browser called X-Smiles (Vuorimaa et al., 2002) was used as a user agent on the client-side. REX was used as the RDE technology and it was implemented for X-Smiles. In this Section is described as an example how a new departure time of a flight can be updated with the push-update methods. The main differences of the methods are discussed.

In RDE+UIML, the update's UI semantics is defined on server-side. In the use case implementation, the UI description was in XHTML. Below is a sample event which replaces a *span* element in the document. The element is recognized by an *id* attribute. The REX interpreter in a user agent automatically does the operation defined in *event* element when it receives the event.

```
<rex xmlns='http://www.w3.org/ns/rex#'>
  <event target='id("edt")'
           name='DOMNodeRemoved'>
    <span id='edt'>16:15</span>
  </event>
</rex>
```

To highlight the changed departure time, text "DELAYED" is added after the updated time. That was targeted to a different element. The REX event was as follows:

```
<rex xmlns='http://www.w3.org/ns/rex#'>
  <event target='id("edtmsg")'
           name='DOMNodeRemoved'>
    <span id='edtmsg'>DELAYED</span>
  </event>
</rex>
```

The boarding pass after the updates is depicted in Figure 5.

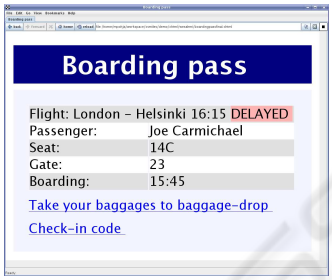

Figure 5: The boarding pass after the push-update.

In the other methods, the UI semantics is given on the client-side. Thus, the REX event can be a bit simpler. Below is an event which updates a data instance on the client-side. In this case, the element's name describes the data it contains.

```
<rex xmlns='http://www.w3.org/ns/rex#'>
  <event target='//edt' name='DOMNodeRemoved'>
    <edt>16:15</edt>
  </event>
</rex>
```

In RDE+XForms, the *edt* element resided within the XForms *instance* element. The REX interpreter must be able to notice that and updated instance document instead of the UI document. After the update, the instance document had to be refreshed to update the UI. The XForms *bind* element compared the updated time with the scheduled departure time using *if()* function and set an *edtmsg* element's content as "DELAYED" if departure time was later than scheduled. The *edtmsg* element was referenced by another *output* element in the form. In contrast to RDE+UIML, there was no need for two update events because the "DELAYED" message could be produced by the XForms *bind* element. The final output was exactly similar to the RDE+UIML implementation (cf. Figure 5).

The RDE+CSS method requires that there is a CSS declaration for the *edt* element to visualize the data update. In the update like the one above, that is usually the case because there already was another *edt* element, which was replaced. In this case, the CSS declaration was:

```
edt { display: inline; }
```

CSS does not provide methods to evaluate the content of the update like XForms does. Thus, the "DELAYED" message must be sent separately as

with the RDE+UIML method above. That could be avoided using the RDE+XBL method. An XBL binding bound to an *edt* element could define a function which checked if the new time differed from scheduled departure time and added the message if it did. The binding is depicted below.

```
<xbl xmlns="http://www.w3.org/ns/xbl">
 <binding element="edt">
  <implementation>
   ({
    xblEnteredDocument: function () {
      var edt = this.boundElement.firstChild
             .nodeValue;
      var sdt = document
             .getElementsByTagName('sdt')
             .item(0).firstChild.nodeValue;
      if (edt != sdt) {
        var msg = document
              .createTextNode('DELAYED');
        var edtmsg = document
              .getElementsByTagName('edtmsg')
              .item(0);
        edtmsg.replaceChild(msg,
              edtmsg.firstChild);
      }},})
  </implementation>
 </binding>
</xbl>
```

6 EVALUATION

The push-update methods are compared to each other against the following criteria.

Maintainability of the UI Description. What have to be taken into account, if an author wants to modify the UI afterwards?

Automatic Evaluation of the Update. How the methods can react on the content of the update?

Versatility. What constraints the methods have regarding the UI description formats.

Maintainability of UI description is simpler with the methods in which UI description resides only on the client-side. In RDE+UIML, the update's UI description is defined on server-side, which requires making modifications on two sites. However, it is noteworthy, that if UIML is used with CSS, the UI can partially be modified through CSS which does not affect on the UI mark-up. Another way to avoid the modifications on two sites with RDE+UIML is to use XML diff (cf. Section 4.1) on the server-side.

As discussed in previous Section, the RDE+XForms and RDE+XBL methods provide means to react to the content of the update. With other methods, extra updates and possibly server-side logic are needed to realize the same functionality.

The RDE+UIML and RDE+XBL can be used with any UI description language as long as it is in XML format. The same goes with RDE+XForms presuming that the format can be combined with XForms. RDE+CSS can use only CSS to layout the data.

7 DISCUSSION

The RDE+XForms and the RDE+XBL methods survived best from the evaluation above. They both are easy to maintain, provides automatic evaluation of the content of the updates, and work well with other UI technologies. The RDE+XBL offers more diverse evaluation of the content of the update than XForms since it is defined by ECMAScript. That, on the other hand, is against the declarative approach used in this paper.

RDE+UIML loses on the maintainability and on the content evaluation for the two above. However, it provides the updates in the final format, which may make the user agent simpler. The UIML can be basically anything supported by the user agent.

The RDE+CSS method's power is its simplicity. It is limited on CSS's ability to define UI, but uses a data instance on the client-side, which makes it simple to use.

A notable aspect on the push-updates is that they suit well only for UI elements which do not allow user input. That is, for instance, if the XForms *input* element could receive updates, there should be logic to prevent the update override a possible user input. User may have provided her input but not sent it to the server when the update arrives. For example, a movie theater can offer a service where user can select a seat from a seat map. The seat map shows both free and reserved seats. All the users see the same map and the reservations are updated for everyone. There must be a way to avoid double booking, for instance by locking few seats for a short period for each user etc.

There is a huge amount of spam sent everyday by e-mail. Undoubtedly, spammers are willing to use also push-updates to deliver their messages. That can be prevented by allowing the updates only from the same source as the original document. In addition, it might be necessary to ask a user if he wants to get updates from a certain source. That way he can for instance deny advertisements but accept information updates.

8 CONCLUSIONS

In this paper, four declarative push-update methods were studied, namely RDE+UIML, RDE+XForms, RDE+XBL, and RDE+CSS. The methods were defined combining existing and upcoming Web technologies. To evaluate the methods, a use case, which included a lot of updates, was designed and it was implemented with all the four methods. The implementations and the implementation experience were reported in this paper.

It was easiest to implement the use case with RDE+XForms and RDE+XBL. They both provide a data instance on the client-side and a powerful UI control. They can also be combined with other Web technologies easily. RDE+UIML and RDE+CSS are simpler methods, but also with them it was possible to implement the use case. However, they required more update messages to complete the same task. Anyway, even the RDE+CSS method proved to be useful in simple use cases.

Along with the adoption of these methods, the WWW still needs a proper method to deliver the updates. Nowadays, the push is emulated by the Comet technology or even by polling the server. SSE seems to be a promising technology for delivery, but comparison of delivery technologies is left as a future work. Another item to work further is the updates of the input elements. As discussed in the previous Section, there must some kind of control if several users can update a single data. The current solution fits only for centralized updates.

REFERENCES

Berjon, R. (2006). Remote Events for XML (REX) 1.0. Working draft, W3C.

Bry, F. and Pătrânjan, P.-L. (2005). Reactivity on the web: paradigms and applications of the language XChange. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1645–1649, New York, NY, USA. ACM.

Cardone, R., Soroker, D., and Tiwari, A. (2005). Using XForms to simplify web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224, New York, NY, USA. ACM Press.

Deolasee, P., Katkar, A., Panchbudhe, A., Ramamritham, K., and Shenoy, P. (2001). Adaptive push-pull: disseminating dynamic web data. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 265–274, New York, NY, USA. ACM.

Dubinko, M., Klotz, L. L., Merrick, R., and Raman, T. V. (2003). XForms 1.0. W3C Recommendation.

ECMA (1999). Standard ECMA-262 - ECMAScript Language Specification. Standard, ECMA.

Garrett, J. J. (2005). Ajax: A New Approach to Web Applications. Technical report, Adaptive Path.

Hickson, I. (2007). XML Binding Language (XBL) 2.0. Candidate Recommendation, W3C.

Hickson, I. and Hyatt, D. (2007). HTML 5. Editor's draft, W3C.

Kanitkar, V. and Delis, A. (1998). Real-Time Client-Server Push Strategies: Specification and Evaluation. In *RTAS '98: Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, pages 179–188, Washington, DC, USA. IEEE Computer Society.

Levinson, E. (1997). The MIME Multipart/Related Content-type. RFC, RFC Editor, United States.

Lindholm, T., Kangasharju, J., and Tarkoma, S. (2006). Fast and simple XML tree differencing by sequence alignment. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 75–84, New York, NY, USA. ACM Press.

Liu, M., Lu, L., and Wang, G. (2003). A Declarative XML-RL Update Language. In *22nd International Conference on Conceptual Modeling, ER'03*, volume 2813/2003 of *Lecture Notes in Computer Science*, pages 506–519. Springer-Verlag.

Resig, J. (2006). *Pro JavaScript™ Techniques*, chapter 14, pages 287–304. Springer-Verlag, New York, NY, USA.

Russell, A. (2006). Comet: Low Latency Data for the Browser. Weblog. Available online: http://alex.dojotoolkit.org/?p=545.

Vuorimaa, P., Ropponen, T., von Knorring, N., and Honkala, M. (2002). A Java based XML browser for consumer devices. In *17th ACM Symposium on Applied Computing*, pages 1094–1099, Madrid, Spain.

Xin, Z., Zhao, J., Chi, C., and Sun, J. (2005). Information Push-Delivery for User-Centered and Personalized Service. In *Second International Conference on Fuzzy Systems and Knowledge Discovery, FSKD'05*, volume 3613/2005 of *Lecture Notes in Computer Science*, pages 594–602. Springer-Verlag.