Server Push with Instant Messaging

Mikko Pohja
Helsinki University of Technology
P.O. Box 5400, FI-02015 HUT, Finland
mikko.pohja@hut.fi

ABSTRACT

Server Push is an essential part of modern web applications. With the ability of sending relevant information to users in reaction to new events, enables highly interactive applications on the WWW. User interfaces of desktop applications have had a two-way communication with an underlying software since their advent, but web applications are reaching the same state only now. In addition, currently, the push is usually emulated using the pull technology, since, with the HTTP protocol alone, it is not possible to realize a real push. This paper evaluates how an instant messaging protocol, namely XMPP, can complement HTTP-based web applications. We present a communication paradigm of a push system and an implementation of it. To evaluate the implementation, a use case is designed and realized with the system.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.5.5 [**Computer System Implementation**]: Servers

General Terms

Design, Experimentation, Theory

Keywords

Instant Messaging, Server Push, XMPP

1. INTRODUCTION

Modern web 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, auction sites etc. The common denominator for all these is a server's ability to push updates to the clients whenever they occur on the server. Often, the document is modified partially, i.e., only the changed parts are updated. That has brought web applications closer to desktop applications.

Even though, the push technology has been studied already over decade and there are numerous of highly interactive web applications, the technology is not really sophisticated. An observation of Franklin and Zdonik [8] is still valid: most push systems are actually implemented using a periodic pull or its derivative. That is due to the fact that IP and HTTP protocols do not properly support push communication. Ajax [9] and Comet [22] frameworks are a step further compared to a simple polling, especially on performance-wise [1, 4]. Nevertheless, they also rely on client initiated updates and HTTP after all. In addition, an implementation effort is notable with them.

In the literature, it is widely suggested that web applications should be implemented using both pull and push protocols [6, 10], because of their dynamic nature. The pull protocol is naturally HTTP, but the push protocol has not been settled yet even though there have been some proposals, e.g., [18, 25]. The push updates are reactions on events and they are sent to receivers as messages. Hence, a web application running on a client compares with a subscriber in a centralized Instant Messaging (IM) application. This paper introduce a model how instant messaging can complement HTTP-based web applications. Extensible Messaging and Presence Protocol (XMPP) is used as the push protocol. XMPP was selected, because it is successively used in many IM applications and its numerous extensions enables diverse usage scenarios.

The main contributions of the paper are:

- Based on a literature and use cases, a set of requirements for an IM-based push system is derived.
- A communication paradigm for the push system is presented.
- A prototype implementation of the system as a proof of concept.
- The system is evaluated based on the derived requirements and a use case implementation.

The rest of the paper is organized as follows. The next Section reviews a background of the topic. Section 3 defines the scope of the paper and requirements. A communication paradigm and a proof of concept implementation are discussed in Sections 4 and 5, respectively. A use case implementation is described in Section 6 and results are evaluated in Section 7. Finally, Section 8 concludes the paper.

2. BACKGROUND

The push systems can be classified many ways. Franklin and Zdonik [8] classify both pull and push systems according to their data delivery policy. Depending on initialization, systems are either aperiodic or periodic. The subcategories are unicast and 1-to-N systems depending on number of receivers. The push enabled web applications fall into aperiodic 1-to-N category.

Kendall and Kendall [14] classify push systems based on data filtering. Simplest systems broadcast the content to everyone, Beta-push allows users to select channels they want to subscribe, Gamma-push sends messages complementing user's needs, and Delta-push analyzes user's behavior and evolves according to that. The system discussed in this paper is closest to the Gamma-push.

2.1 Related Work

There has been considerable research on generic push systems [2, 3, 5, 13, 26]. They do not operate in Web environment, but shares the common functions like reacting on changes e.g., in a database and creating push events based on changes.

At its simplest, Server push on Web can be emulated by polling the server at a certain time interval. That can be done for instance by reloading the document periodically or with Ajax [9] by asking incremental updates. That causes a lot of additional network traffic, especially reloading the whole document, and there is always trade-off between the latency and the polling frequency.

Comet [22] combines two existing frameworks to emulate the push. It uses Ajax to realize asynchronous updates and Multipart MIME type [16] 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 [21].

HTML 5 specification [11] defines an approach called Server Sent Events (SSE). With SSE, an author can declaratively define a source, from which the browser is listening for the incoming connections. The communication itself can be realized for instance with Comet. The HTML 5 specification is at work-in-progress stage at World Wide Web Consortium (W3C) at the moment.

Deolasee et al. [6] and Hauswirth and Jazayeri [10] discuss components and theory of the push systems on the Web. Both have examined the combination of pull and push and find it suitable for the Web. Deolasee et al. argue that adjustment between push and pull depends on use case. Sometimes only either one is needed. In Hauswirth and Jazayeri's system, the push is used to notify user agent to pull fresh information. Other push and pull systems are the mWeb presentation framework [18], which uses SRRTP and SRFDP protocols for push, and HTTP+RTP push/pull system by Trecordi and Verticale [25].

2.2 Publish/Subscribe

Publish/Subscribe (pub/sub) is an asynchronous messaging model. Publishers do not send their messages directly to subscribes but to channels. Subscribers receive messages from channels they have been subscribed. That makes the paradigm loosely coupled. Actually, pub/sub can be decoupled on three dimensions: space, time, and synchronization [7]. That is to say, senders and receivers do not have to know each other, the messages can wait on a channel for subscription, and the production and consumption of events do not block other activity on either end of the system. Removing dependencies between parties makes the paradigm well adopted in distributed environments, which are asynchronous by nature [12]. Pub/sub paradigm is an example of the push system as the publishers push data to the subscribers through the channels.

Subscription of the pub/sub system can be topic-, content- or event-based [7]. In topic-based systems, a receiver subscribes to a certain topic, from which it receives all the published messages. In content-based pub/sub model, subscriber defines filters on subscription. It will receive messages, which pass the filters. Event-based systems are founded on event types, to which receivers subscribe. In addition, pub/sub system can be a combination of the above-mentioned types.

2.3 XMPP

Extensible Messaging and Presence Protocol (XMPP) is a protocol for streaming Extensible Markup Language (XML) elements in near real time between any network endpoints [23]. It is initially developed as a protocol for IM applications [24]. The XMPP technology set comprises of core functionality and numerous extensions. The core defines how XML snippets are handled for instance in IM applications and the extensions utilize that base. Among the others, there is an extension for XMPP pub/sub [17] and HTTP binding for XMPP communications (BOSH) [19] that can be used if regular XMPP communication is blocked (e.g., by firewall).

BOSH uses Comet to mimic two-way communication between the Server and the browser. However, even if it is an emulation of push and not pure XMPP anymore, it has many benefits over plain Comet since it holds other characteristics of the XMPP. Those include persistent connections even if the underlying network connection is unreliable, prescribed messaging model and format, and authentication. It can be said that XMPP BOSH provides standardized communication model for Comet.

3. RESEARCH PROBLEM AND SCOPE

The research problem of the paper was to design a system, which can submit dynamic information for web applications. The goal was not to replace HTTP protocol, but define an additional protocol to support server push. The main focus is on one-to-many delivery scheme, even though broadcasting and one-to-one communication can be realized with the same technologies to some degree. In addition, delivering user input is out of scope of this paper. The next subsection introduces two possible use case of the system.

3.1 Use Cases

3.1.1 On-line Auction Tool

There are several goods on sale on an auction site. Users can set bids for the goods they like to purchase and a user with a winning bid win an item. To follow the bids from other users, the application must be able update the bidding history on users' screen in real time.

3.1.2 Score Service

A soccer league offers a live score service on their Web site. The service provides a live score and played minutes from each game on a match day. Each game has also an own page, which has additional information on the game in question. That includes scorers, bookings, substitutions etc. The main page provides access to these pages.

3.2 Requirements

The requirements of the system are derived from the literature and the use cases. The requirements are divided to the general system requirements and to the technology requirements.

3.2.1 System Requirements

R1: Protocol. System must support both push and pull protocols. Based on the literature, HTTP-based web applications benefit from an integration of a push protocol.

R2: Delivery Scheme. System must support one-to-many delivery scheme. That is, more fine grained than a broadcasting, but not a one-to-one dialog.

R3: Coherence. The data updates must be delivered in near real time.

R4: Flexibility. System must support asynchronous communication between loosely coupled components.

R5: Performance. The system must outperform HTTP polling in terms of latency and amount of a transferred data. Both of them are considered problems with polling.

3.2.2 Technology Requirements

R6: Ease of Authoring. The application development should not require any new programming languages to ease the adoption of the system. Declarative languages are commonly considered easier to author than procedural languages and can be used even by non-programmers. An application should consist of reusable software components.

R7: Web integration. Using existing technologies at client-side eases the adoption of a technology or a framework among the users.

4. COMMUNICATION PARADIGM

The communication of the system is based on *Push-and-Pull (PaP)* [6] algorithm. That is, a user agent fetches a document from the Web Server using HTTP protocol as usual. When the web application is running on the user agent, the dynamic data is pushed to the client whenever needed. In addition, the model allows degrading gracefully if the push component fails. The system can rely on pull based polling as a backup mechanism.

The system uses pub/sub-messaging paradigm with a combined topic- and content-based model. The user agents subscribe to the topics according to the location of their current resource. In other words, there is a one-to-one match between URLs on the Web Server and the topics on the Push Server. The Web Server communicates the correct topic to the user agent within the data sent over the HTTP. Thus, it can be said that the model supports the REST paradigm on which the WWW relies.

There is an Event Source for every topic on the Push Server. The Event Sources track the database changes, create update events according to the changes, and publish them on their topics. The Push Server distributes the updates to every user agent, which has subscribed to the corresponding topic. This is known as a multicast delivery scheme. The distributions are fine-grained adding content-based subscriptions. This way only relevant updates are filtered to each subscriber.

The Event Sources hold the information on what data on the database corresponds to a certain element in a DOM tree of a web application. The Event Sources track the changes of the pertinent data in the database and, whenever notified about a change, publish an update event, which modifies the corresponding node in a DOM tree on client-side.

As a summary, the communication between components of the system is shown in Figure 1. The user agent fetches a document form the Web Server via HTTP as usual (1). In addition to the document, it receives address and subscription details of the Push Server. With that data, the user agent connects to the Push Server and subscribes to the relevant topic (2). As a respond, Push Server submits the latest message of the topic to keep the user agent up to date (3). That ensures that a possible change, which occurs between Web Server's response and the subscription, is delivered to the client.

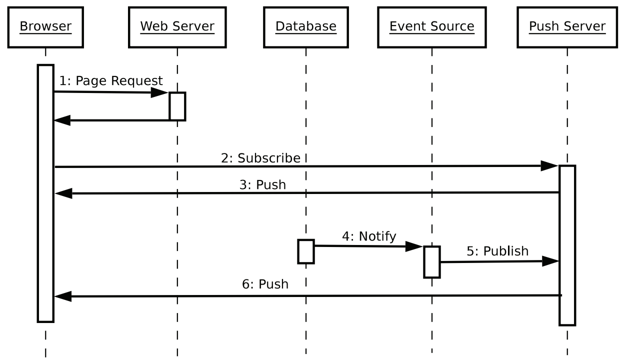


Figure 1: The communication between components of the system.

When a data is modified in the database, a trigger notifies the Event Source (4). The Event Source creates an update event and publishes it on the Push Server in the pertinent topic (5). The Push Server pushes the event to all subscribers of the topic (6).

5. IMPLEMENTATION

We have implemented a Push System conforming to the communication paradigm discussed in the previous Section. The system is based on the XMPP protocol. Main reasons for selecting XMPP were its widely usage in other XML streaming applications, its pure push nature, existing implementations, and possibility to use with legacy browsers and firewalls.

5.1 Components

The components of the system are depicted in Figure 2.

All the software is open source software and extended as needed. The Web Server is an Apache Tomcat Servlet container[1] and the database is an eXist-db XML database[2]. In addition, there are an XMPP Server and Event Sources in the server-side. The XMPP Server is an Openfire XMPP Server[3]. It has native support for XMPP pub/sub. Event Sources are system specific components, which use Smack XMPP client library[4] with su-smack[5] pub/sub extension for XMPP communication. User Agents of the system include a browser, the XMPP client, and an update handler. The XMPP Client can have native integration into the browser or it can be an ECMAScript library, which comes along with a web application. Thus, the system can be used with legacy browsers. As a proof of concept, we integrated the XMPP components into X-Smiles [27], which is a Java based open source browser. It uses the same libraries as the Event Sources for the XMPP communication. The update handler depends on format of the updates. That is discussed below.

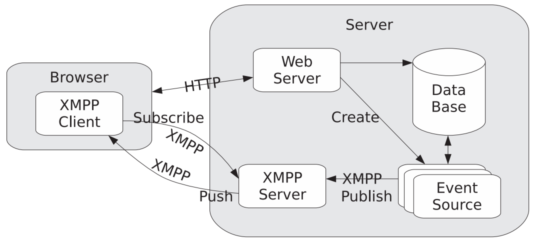


Figure 2: The components of the Push System.

5.2 Initialization

In addition to handling the Servlets, the Web Server initializes the Event Sources automatically. Each Servlet defines which data on the database its Event Source should monitor and to which element it corresponds in the document it provides. The Event Sources are parametrized via the Servlets. An author of an application includes corresponding database reference and element pointer pairs in a Servlet. The generic Event Source can create update messages according to the pairs. However, the author can also extend the generic Event Source to create custom messages. That might be necessary if, for instance, the author wants to modify or analyze content before submitting it to the clients through the Push Server. In addition, at this point the author can add keywords to the message to enable content-based filtering on the Push Server.

The Event Sources subscribe to the XMPP Server just like the User Agents (UAs). As pointed out in the previous Section, there is a one-to-one mapping between a web document, an Event Source, and a topic on the XMPP Server. The Event sources create their topics on the Server and publish update events on them.

UA receives the subscription details for the XMPP Server within the HTTP response of the Web Server. The subscription data is embedded in the header section of a document and is recognized by a namespace. The concept is equivalent to the aforementioned HTML 5's SSE. SSE also allows defining the event source, but not topics and content filters, which are required for the XMPP. Since XMPP messages are in XML format, the subscription data can be embedded as such in a document. The UA must be extended to handle the XMPP snippets in the documents. An alternative to the declarative subscription descriptions is to add an ECMAScript function, which subscribes to the XMPP Server using the dedicated ECMAScript library.

With XMPP, a client must first connect to an XMPP Server before it can subscribe to topics and receive messages. The system opens a new connection for every document it downloads and closes the connection when user moves to a next document to avoid unused open connections.

5.3 Operation

The operation of the system is described in Figure 3. The database has a mechanism to trigger an event when a collection in a database is modified. The Event Sources have registered themselves to the relevant triggers. The triggers notify registered Event Sources when database has been modified (1). The Event Sources publish the modified information on their topics in the XMPP Server (2), which further pushes the event to the subscribers (3). The browser handles the event and updates relevant node in a document (4).

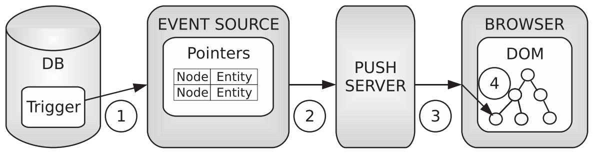


Figure 3: Conceptual model of the operation.

5.3.1 Database Triggers

The database triggers are built on the triggers provided by the eXist-db. The eXist-db triggers do not specify exactly which data has changed on the database, but only the modified collection. To find out the exact change, we use *xmldiff*[6] to compare the collection before and after the modification. The *xmldiff* returns the modification in XUpdate language [15]. Effectively, the XUpdate result consists of an XPath expression, which refers to the correct element in a database and the new content. To see if the expression relates to a certain Event Source, the Event Source's XPath references are compared to the expression. The comparison is done taking the intersection of the two expressions. If the intersection is not null, the modification relates to the Event Source and it has to publish an update event on the pertinent topic in the XMPP Server. The communication between the database triggers and the Event sources is realized with Java Messaging Service (JMS) technology.

Computing the intersection against every XPath expression in every Event Source is not an optimal solution. Designing structure of the database carefully can enhance the performance. What smaller collections are that lower number of Event Sources is tracking changes on each one of them. However, since XML processing methods are out of scope of this paper, it is left as a future work to find out the most scalable solution.

[1]Apache Tomcat, http://tomcat.apache.org/
[2]eXist-db, http://exist.sourceforge.net/
[3]Openfire XMPP Server, http://www.igniterealtime.org/projects/openfire/index.jsp
[4]Smack API, http://www.igniterealtime.org/projects/smack/index.jsp
[5]su-smack, http://static.devel.it.su.se/su-smack/
[6]XML differencing tool, http://www.logilab.org/859

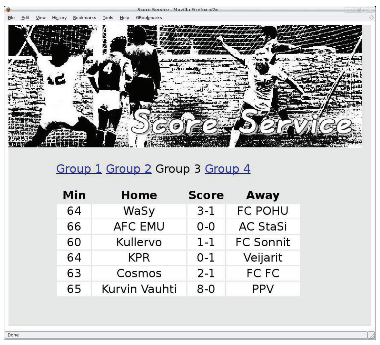
5.3.2 *Update Events*

The purpose of the push updates is to modify the DOM tree in the client-side. The modification methods are discussed in depth in a related research [20]. Since XMPP messages are in XML format, it is natural to use XML to describe the updates. W3C has produced a working draft of Remote Events for XML (REX), but at the present moment has finished the specification work because of patent issues[7]. In spite of all, REX represents the declarative concept to describe the update events. In this paper, that concept is called Remote DOM Events (RDE).

The Event Sources create the RDE events. An event defines a target element in the remote DOM tree and a modification on it. The target element is specified by an XPath expression and modification is gotten from a database trigger. The author of the application defines the XPath expression.

6. USE CASE

We implemented the Score Service use case discussed in Section 3.1. The service provides a main page, from which a user can select which group's games he or she wants to follow. A screen capture of the main page is shown in Figure 4. It shows played minutes and scores in real time. Clicking a score on the page opens a new page, which has detailed information on the game in question.



Score Service - Mozilla Firefox <2>
File Edit View History Bookmarks Tools Help GBookmarks
Score Service
Done

Group 1 Group 2 Group 3 Group 4

Min	Home	Score	Away
64	WaSy	3-1	FC POHU
66	AFC EMU	0-0	AC StaSi
60	Kullervo	1-1	FC Sonnit
64	KPR	0-1	Veijarit
63	Cosmos	2-1	FC FC
65	Kurvin Vauhti	8-0	PPV

Figure 4: Screen capture of the Score Service use case.

The score service consists of pages for each group and a general page to follow a single game. If a user follows a whole group, the user agent subscribes to a pertinent topic on the XMPP Server and receives all the updates of the topic. In the case of the single game, the user agent subscribes to a general topic, which receives all the updates from all the games. The subscription includes also content-based subscription, which in this case is a game id. Hence, the user receives updates only from game he or she has selected.

The implementation uses the generic Event Sources for the group pages, because all the updated content appears as such on the page. For the single game page, we implemented a custom Event Source, which adds the game id for each message to enable the content-based filtering.

7. EVALUATION

The implementation shows that it is possible to implement a system, which push database modifications all the way to the Web browser in near real time. Overall, the system fulfills the requirements well. XMPP with the pub/sub extension suits well in this kind of communication. Using XMPP pub/sub covers the requirements R1, R2, and R4. Decoupling between the core communicators, i.e., the user agents and Event Sources, has succeed well. They both communicate through a standard XMPP Server and do not have any dependencies between each other.

The model provides means to fulfill the coherence requirement (R3), but the current experimental implementation needs an optimization on database triggering to conform to the requirement well. The performance of the push communication is similar to Comet that has been shown to outperform legacy HTTP polling applications clearly [1, 4]. In addition, the XMPP-based system has some benefits over plain Comet as discussed above.

The push applications are easy to author with the system. An author just has to provide data reference pairs on server-side for the Event Sources. The pairs include a pointer to the database and a corresponding element in a client-side document. In addition, if content-based filtering is wanted on the Push Server, an author must customize the Event Sources to provide desired content. That requires knowledge on programming. The system specific components take care of the rest of the functionality. For the client-side functionality, the author just needs to add relevant ECMAScript libraries unless the client supports the technologies natively.

The requirement R7 presumes that the legacy clients should be able to use the system without additional components. That is fulfilled well in the system. As mentioned earlier, the ECMAScript libraries can handle all required client-side functionality and the XMPP BOSH extension enables the communication even through the firewalls.

The use case implementation shows that creating topics based on URL on the Push Server is a workable method. If the varying content on the page is same for all, the topic-based subscriptions are enough. If the content depends on a user, the subscriptions can be fine-grained with content-based subscription. Still, there is only one topic per URL.

The system requires still some optimization before it could be widely deployed. We have left as a future work to enhance the database triggers. An efficient method to find the correct Event Sources must be found. Another possible point of optimization is XMPP related. Now system requires the user agent to create a new connection to the XMPP server every time it fetches a new page. And, respectively, close the connection when leaving the page. That produces communication overhead, because web applications naturally consists of several pages. The connection to the Push server could be kept open if it remains the same between the previous and new page. The logic should be implemented on the user agent or provided with ECMAScript.

8. CONCLUSIONS

Server push is an integral part of present-day web applications. The push is usually emulated using HTTP-based technologies. That causes additional network load and increases implementation effort. This paper examined how to replace current HTTP-based push technologies with instant messaging. We propose that HTTP communication should be complemented with a push protocol, XMPP to be exact.

A communication paradigm of the instant messaging-based push system is presented in the paper. The paradigm is

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

based on the literature and the requirements, which we defined for the paper. In addition, we implemented a push system conforming to the communication paradigm and also a use case running on the system. The implementation shows that the paradigm is feasible and earlier research confirms that XMPP is scalable in large systems. The implementation still requires some optimization regarding database triggering. Otherwise, the implementation fulfills the predefined requirements well.

9. ACKNOWLEDGMENTS

The author would like to thank Jari Kleimola for valuable comments on the topic and Petri Vuorimaa, the leader of the research group.

10. REFERENCES

[1] M. Angelaccio and B. Buttarazzi. A Performance Evaluation of Asynchronous Web Interfaces for Collaborative Web Services. In *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, volume 4331/2006 of *Lecture Notes in Computer Science*, pages 864–872. Springer, November 2006.

[2] J. Bailey, A. Poulovassilis, and P. T. Wood. An Event-Condition-Action Language for XML. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 486–495, New York, NY, USA, 2002. ACM.

[3] A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 633–641, New York, NY, USA, 2001. ACM.

[4] E. Bozdag, A. Mesbah, and A. van Deursen. Performance Testing of Data Delivery Techniques for AJAX Applications. Technical report, Delft University of Technology, Delft, Netherlands, 2008.

[5] A. Delis and N. Roussopoulos. Management of updates in the enhanced client-server DBMS. In *Proceedings of the 14th International Conference on Distributed Computing Systems, 1994.*, pages 326–334, June 1994.

[6] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. 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, 2001. ACM.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[8] M. Franklin and S. Zdonik. "Data in Your Face": Push Technology in Perspective. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 516–519, New York, NY, USA, 1998. ACM.

[9] J. J. Garrett. Ajax: A New Approach to Web Applications. Technical report, Adaptive Path, 2005.

[10] M. Hauswirth and M. Jazayeri. A Component and Communication Model for Push Systems. *SIGSOFT Softw. Eng. Notes*, 24(6):20–38, 1999.

[11] I. Hickson and D. Hyatt. HTML 5. Working draft, W3C, January 2008.

[12] Y. Huang and H. Garcia-Molina. Publish/Subscribe in a Mobile Environment. *Wirel. Netw.*, 10(6):643–652, 2004.

[13] V. Kanitkar and A. Delis. 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, 1998. IEEE Computer Society.

[14] J. E. Kendall and K. E. Kendall. Information delivery systems: an exploration of Web pull and push technologies. *Communications of AIS*, 1(14), April 1999.

[15] A. Laux and L. Martin. XUpdate XML Update Language. Working draft, XML:DB Initiative, September 2000.

[16] E. Levinson. The MIME Multipart/Related Content-type. RFC, RFC Editor, United States, 1997.

[17] P. Millard, P. Saint-Andre, and R. Meijer. Publish-Subscribe. Standards Track 0060, XMPP Standards Foundation, March 2008.

[18] P. Parnes, M. Mattsson, K. Synnes, and D. Schefström. The mWeb presentation framework. *Comput. Netw. ISDN Syst.*, 29(8-13):1083–1090, 1997.

[19] I. Paterson, D. Smith, and P. Saint-Andre. Bidirectional-streams Over Synchronous HTTP (BOSH). Standards Track 0124, XMPP Standards Foundation, February 2007.

[20] M. Pohja. Declarative Push on Web. In *Proceedings of the 4th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 201–207. INSTICC, May 2008.

[21] J. Resig. *Pro JavaScript^{TM} Techniques*, chapter 14, pages 287–304. Springer-Verlag, New York, NY, USA, 2006.

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

[23] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Proposed standard, IETF, October 2004. http://www.ietf.org/rfc/rfc3920.txt.

[24] P. Saint-Andre. Streaming XML with Jabber/XMPP. *IEEE Internet Computing*, 9(5):82–89, 2005.

[25] V. Trecordi and G. Verticale. An architecture for effective push/pull Web surfing. *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, 2:1159–1163 vol.2, 2000.

[26] L. Vargas, J. Bacon, and K. Moody. Integrating Databases with Publish/Subscribe. In *ICDCSW '05: Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05)*, pages 392–397, Washington, DC, USA, 2005. IEEE Computer Society.

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