Bridging the Browser and the Server

Miguel Raposo and José Delgado,

Instituto Superior Técnico, Universidade Técnica de Lisboa, Av. Prof. Cavaco Silva, Porto Salvo, Portugal
miguelfernandoraposo@gmail.com, jose.delgado@ist.utl.pt

Abstract. Web applications are now built on the principle that users interact with them through a generic, universal browser. The paradigm, client-server, is essentially limited to one-way interactions, with the client as the sole entity with real initiative. Also, server-based applications often do not guarantee information privacy, resulting in reluctance in its usage. This paper presents the Browserver as a means to give users the ability to be service providers, not mere consumers, and to avoid storing data at central servers. We describe an architectural approach and a technological solution for the union of a browser and a server for the development of a Browserver using existing technologies.

Keywords: Browser, Server, User Interface, Services, Peer-to-Peer

1 Introduction

In the early Internet days, applications were made with specific client and server side components (Fig. 1) and specific protocols, with interactions limited by the existence of the specific client on each user's machine. Nowadays, the browser constitutes a generic, universal client component capable of accessing all of the ever-growing Web applications (Fig. 2). Web users are seen as information generators, not merely as consumers. Although services already constitute the main paradigm at enterprise integration and the Internet of Services [17] is already a discussion subject, the Web is still centered around content and not on services, with the client-server paradigm limiting the interaction patterns with humans by requiring these to initiate the interaction by navigating to some page through a URL. If a user is involved in some business process, there is no direct way to interact with him through the browser so, the email is now the most used tool to contact and request someone's services, having become a nightmare and not practical for many persons nowadays.

To reduce this limitation, AJAX, polling and long-lived HTTP connections (Comet) [4] have been introduced to simulate server requests to the client, enabling more dynamic processes. Web Sockets [1, 2] are promising real bi-directional connections between the browser and the server, enabling better and faster communication between browser and server than AJAX. Nevertheless, the browser remains as a simple client, in the same paradigm, and business processes still depend on user's will to initiate the first interaction. This way, the email remains as an indispensable tool to connect people.

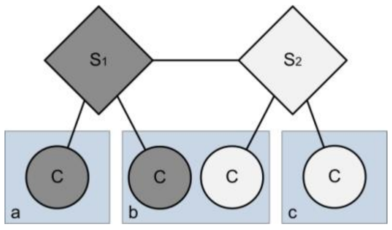

Fig. 1 A Specific client for each specific application.

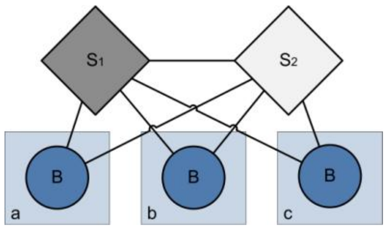

Fig. 2 The Web browser, a universal client for Web applications.

Using the browser, interactions between people in the Web always have a server as an intermediary, which offer the services that enable information sharing and collaboration. These intermediaries can take the information and use it for their own purposes. Users (or an agent on their behalf) should be able to provide electronic services themselves and be first class peers in web interactions and business or generic processes without the need for applicational intermediaries. We propose to overcome the limitations of the client-server paradigm by endowing each user not with a browser but with a Browserver (a browser (*B*) and a server (*S*)), as represented in Fig. 3. Interactions can be made directly between peers (*a*, *b*, *c*) equipped with a Browserver. Remote servers (*S1*, *S2*) can also be accessed as usual but are not as crucial. Direct, P2P interactions now become the norm instead of having to resort to centralized application servers for user interactions.

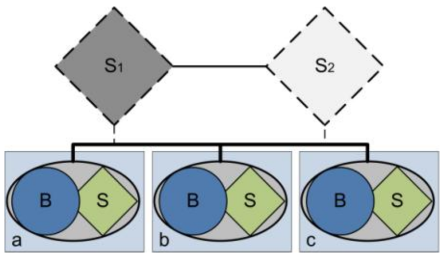

Fig. 3 The Browserver, the union of a universal client and a universal server on P2P interactions.

This entails a paradigm change for web usage, from client-server to peer-to-peer, and not just for file sharing. Applications such as email, instant messaging (IM), social networks, collaborative document edition and workflow systems can be implemented without necessarily depending on some central server system.

We conceptually present the Browserver in Sect. 2, a technological solution to it in Sect. 3, the related work in Sect. 4 and draw some conclusions in Sect. 5.

2 The Browserver

We consider a service as a capacity exhibited by an entity (e.g. a user or system) which can be offered by him, as provider and used by other(s), as consumer(s). The

Browserver provides a platform for service interactions involving users either as providers and/or consumers, including both:

- A Web browser. A generic and universal browser (e.g. Internet Explorer, Firefox, Chrome);
- A Web server. A generic and universal application server, enabling the user to provide services to other entities (e.g. Sun Glassfish, Apache Tomcat).

Although each user is able to create content and resources [13] that others can use, he is still positioned in the edge, and not in the center of the Web. Security issues limit browser's connections to be made only with the originating server of the Web page that the user is navigating, making impossible to build applications for direct collaboration through the browser. Each user acts as the ultimate consumer of services made available by other users or organizations on remote servers, that act as interaction intermediaries and have full access to information shared between peers even if private. The Browserver sets P2P (Fig. 5), as its paradigm for interactions, instead of the classic client-server model of the Web (Fig. 4).

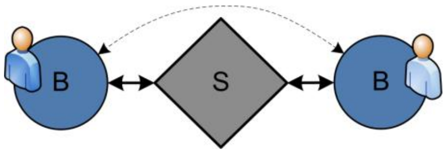

Fig. 4 Client-server model. There is always an intermediary offering services to each entity.

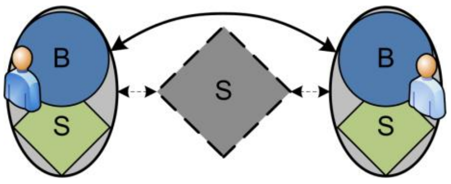

Fig. 5 Peer-to-peer model. Entities can interact directly, consuming each other's services.

The Browserver aims at giving each user a really active role in the Web, minimizing the need for intermediaries, and turning each user to be seen as an entity fully capable of providing services, rather than a mere consumer of information and services. The browser acts as a user interface for locally hosted services that can be made available to the Web as well as to remote services that need to interact with the user. Each public service of the user can be directly consumed (called, requested) by the entity who needs to. Everyone becomes a service provider and the Web becomes service centric instead of content centric.

To a business process, a person with a Browserver is seen as the set of invocable services that he provides. Also, services otherwise located at centralized servers may now exist in each user's computer. This entails:

- More information privacy, by putting services locally to each Browserver and directly consuming other's services in a peer-to-peer fashion;
- A complete service paradigm on building applications from which enterprise applications and customer relationship management can benefit from;
- New enterprise and personal relationships, and new tools for collaboration.
- Interactions with users can be proactive and not only reactive to user's actions.
- The email and other communication platforms became accessory and not mandatory for communications and interactions involving persons in the Web.
- Offline work, which can be granted by having the needed services and resources for an application executing at the local server.

3 A Technological Approach

In this section we present a high-level description of one solution for a technological implementation of the Browserver which is part of a work in progress on the subject. Although privacy requires security, that is not the focus of this article. The main focus on this architectural design goes to the connection of a browser and a server and automatic UI generation for services.

This approach intends to demonstrate the use of existing technologies to build a Browserver. Given that services are the paradigm of the Browserver, Web Services are chosen for its expressiveness and widespread use at organizational level and Java is chosen for its full support on the technology. However, the Browserver is not limited to a specific language or protocol.

3.1 Browser and Server

To unite a browser and a server some alternatives arise:

a) Develop from scratch a new fully integrated Web browser and server.
b) Develop a standards compliant browser frontend as an application running on the server.
c) Connect a local browser to a local server using existing solutions.

In the solution presented in this article, we opt for the last option. This gives the user the option to use the browser of its choice, while empowering him with the features of a Browserver. It also allows normal Web navigation, making the Browserver network a parallel Web to the existing one. Another advantage comes with the possibility of physical separation of both components. On user's will or necessity, its private server could be located remotely (at his home or office) and the browser could be on his mobile device (less computational capable).

The server must be compliant with Web Services [18] standards. Being Java the programming language, Java Servlets are used in the implementation, therefore Glassfish is the choice as it meets the requirements, with the integrated Metro web service stack [10]. Tomcat or other compliant server could have also been chosen.

To actively make requests to the user, Comet and Reverse-Ajax [4] help to overcome the limitations of the client-server model. Comet refers to long-lived HTTP connections, enabling low-latency communication between browser and server. Reverse-Ajax uses continuous *polling* from the client to the server for changes or server *pushing* to the client using Comet connections enabling a server to send data to the client without it without having been explicitly requested.

Direct Web Remoting [9], offers a framework for browser-server interaction based on Reverse-Ajax. Complementing with a strong Javascript library, like JQuery [16] full manipulation of a Web page displayed on the browser can be made. Fig. 6 shows:

- DWR Javascript library at the client side.
- DWR Java Servlet at the server side
- Browserver auxiliary and structural Javascript for UIs at the client side.
- Browserver Plain Old Java Classes (POJOs), Servlets and Beans, composing the Browserver architecture at server side.

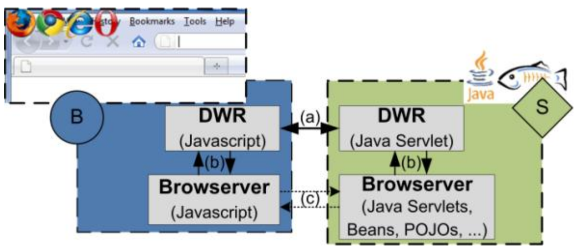

Fig. 6 Connecting browser (B) and server (S) through the DWR [9] framework.

The DWR framework exposes classes and methods on the Server that can be called from the client, being reverse Ajax used to connect both ends (*a*) through pull and push based techniques. At the client side, server methods are called (*c*) through the DWR framework (*b*), being the returned result obtained through a callback function.

The server also acts as a proxy to the browser, allowing navigation on the Internet. Fig. 7 shows a request (*a*) to a remote resource (a Web 2.0 site), going through a local proxy at the server that executes the request (*b*) and sends the response to the client. The response can be parsed, filtered and modified, if a service with such properties exists in the server, enabling the system to act as in [8].

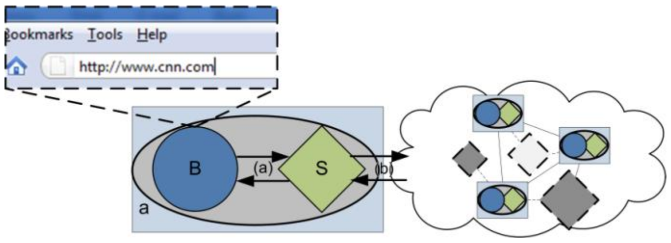

Fig. 7 The server (*S*) acting as a Web proxy to the browser (*B*).

3.2 Architectural Logical View

Fig. 8 presents a simplified logical view of the Browserver with two main parts: the *Browser (B)* and the *Server (S)*. In the context of this solution, the development leads to a single application deployed and running on the server. The *Browser* part of the system is responsible for creating and managing UIs for services and the connection with the browser. The *Server* part of the system has responsibility of managing services and the network of the Browserver. Each service has its own unique identifier, compliant with the URI syntax [11]. In the *Browser* part:

- The *BrowserManager*, coordinates the creation of *Containers* and *ContainerUnits*, and is responsible for sending the full container UI for the specific browser that requires it through the *Proxy*, as well as creating new UI units from *UIData* sent by the *ServiceManager,* using the *UnitBuilder*.
- A *UnitBuilder* takes the XML definition of an UI and builds a *ContainerUnit* representing that UI. The *BrowserManager* can then add it to a *Container*.

- There can be one or more *Container*s available and each holds multiple *ContainerUnits*, being *Portal*,*Portlets* and *ControlPanel* realizations of these. These elements produce code understandable by the browser like HTML and Javascript. The *Container* also updates the UI at the browser through the DWR whenever it changes internally.
- A *DataHandler* has the ability to handle user input from the browser. A *ContainerUnit* must handle this data, sending it to the *BrowserManager*, who forwards it to the *ServiceManager*.
- A Interface Unit can be:
  - A *SimpleUnit*, which cannot hold any other units inside (e.g. a *SimpleText* is used to present text without any special format).
  - A *ComplexUnit*, which can hold other units (e.g. in HTML, a <*div*> element plays this role).
  - A *DataUnit*, which is a *ComplexUnit* and *DataHandler* that collects data from the user (e.g. a <*form*> element in HTML corresponds to a DataUnit).
  - A *ContainerUnit*, which is a *DataUnit* that holds the whole UI for a service and handles input data from the browser, redirecting it to the corresponding service at the Server part.

Different browsers are supported by *Containers, ContainerUnits* and *UnitBuilders* that aim the specificities of each one. For a mobile device, a simple new *Container* that extends an existing one and converts the output using XSLT could be a solution.

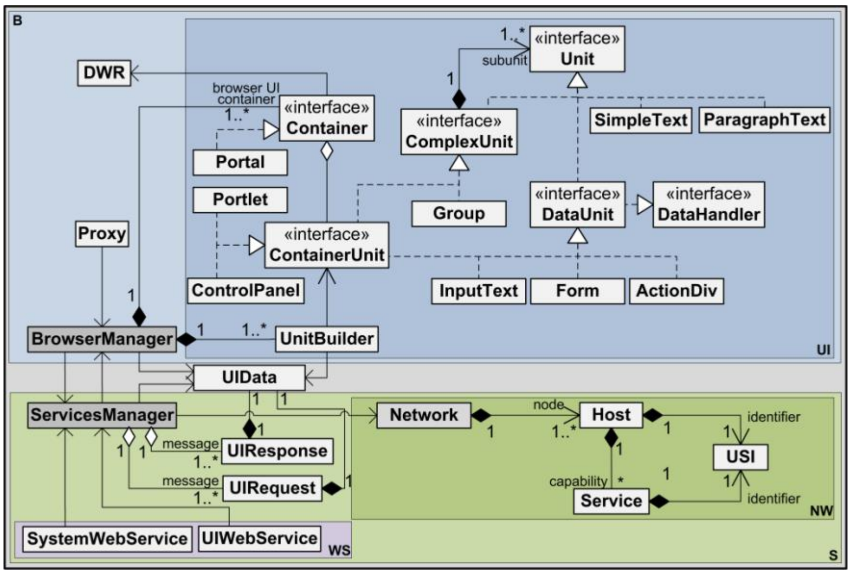

Fig. 8 Simplified logical view of the Browserver.

In the *Server* part:

- The *Server* part (S) is divided into the services part and the network (NW) part. In the services part, are the externally accessible Web Services (*WS*).

- The *ServicesManager* is responsible to manage incoming requests and outgoing responses for UI data.
- The *Network* consists of at least one *Host* (the local host) and all the known remote Browserver hosts that can have any number of associated *Service*s.
- The *Network* provides a means to remotely register local services. A remote service directory is used to publicize the services of the Browserver.
- The UIWebService is an externally accessible Web Service for external entities to request UIs to the local Browserver. *SystemWebService* is an externally accessible Web Service for external entities to make system requests. Its operations include *deployment* and *undeployment* of services. SOAP-based [22] implementations of these services offer great interoperability with existing systems.

3.3 Services

Services can be developed either to be local only or remotely accessible. The service is deployed on the local application server and registered in the Browserver, through the *SystemWebService*. The user has full control on whether the service is remotely published or not. Services can be composed of other services, promoting reusability.

Asynchronous communication is a crucial requirement on processes involving users so the Browserver *UIWebService* uses only one-way message exchange patterns. This decision is due to the nature of the behavior of users. A reply might be made immediately, or after weeks so, bidirectional communication channels can't be assured.

To receive replies to UI requests, the requester must provide a specific endpoint that is able to receive, process and correctly deliver SOAP messages, using WS-Addressing [22]. This is a limitation of existing communication channels, such as HTTP, which is the basis of the Browserver communication, as it is application-agnostic and can easily pass through firewalls.

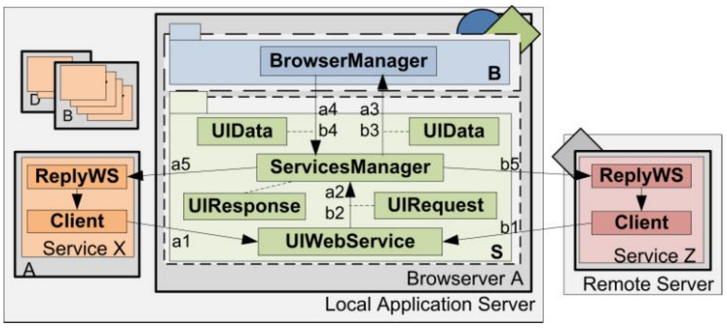

Fig. 9 Services activity on user interface request.

Fig. 9 presents a simplified activity on UI creation from services point of view:

1) A local service *X* and a remote service *Z*, request a UI to the Browserver, through the *UIWebService*, using SOAP messages (*a1,b1*) with UI definitions

compliant with the schema presented in Fig. 10. The messages include WS-Addresing headers indicating where to send the reply.

2) The *UIWebService* builds a *UIRequest* object with information provided by the sender and a *UIData* object representing the UI definition, sending it to the *ServiceManager* (*a2,b2*).
3) The *ServiceManager* dispatches requests, sending the *UIData* to the *BrowserManager*, that will generate and present the UI to the user (*a3, b3*).
4) The data submitted by the user is forwarded (*a4,b4*) by the *BrowserManager* to the *ServiceManager,* that builds a *UIResponse* with the data needed for the reply.
5) The *ServiceManager* sends the data (*a5,b5*) to the endpoint previously indicated by the requester.
6) The requester *Service* parses the data and act accordingly to its business rules.

To maintain context on successive service interactions, the *messageId* and *relatesTo* elements of the WS-Addressing headers are used. A user data response contains an ID that can be used on a later request, to indicate the relationship. To make a service publicly available, the user can indicate the Browserver to publish it in a service directory, like UDDI. A distributed solution for this is described in [21].

3.4 User Interface Generation and User Data Handling

Fig. 10 presents a simplified XML schema for UI definition for the Browserver. Upon receive a request, the *Browser* object uses the *UnitBuilder* to get a new *ContainerUnit* for that request. This *ContainerUnit* is then added to the *Container*, which has the responsibility to update the UI view at the browser, through DWR and JQuery.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:complexType name="interfaceType">
    <xs:group ref="iGroup" minOccurs="0" maxOccurs="unbounded"/>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="formType">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="inputText" type="inputTextType"/>
        <xs:element name="text" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="inputTextType">
    <xs:attribute name="name" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:group name="iGroup">
    <xs:sequence>
      <xs:choice>
        <xs:element name="p" type="xs:string"/>
        <xs:element name="text" type="xs:string"/>
        <xs:element name="form" type="formType"/>
        <xs:element name="group">
          <xs:complexType>
            <xs:group ref="iGroup" minOccurs="0" maxOccurs="unbounded"/>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:sequence>
  </xs:group>
</xs:schema>
```

Fig. 10 Simplified user interface definition XML schema.

User input is gathered by Javascript (JQuery) and submitted to the *Browser* object through DWR as a JSON string object. No POST or other HTTP actions are activated at the browser. The JSON data is then converted to XML and sent to the *DataHandler* associated with the UI unit. The *DataHandler*, primarily the *ContainerUnit*, parses the data, deciding whether it will be redirected to a smaller unit to handle or to the requesting service as a data response message, whose schema is simplified in Fig. 11.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="data">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="input" type="dType" minOccurs="0"
maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 11 Simplified Data Response XML Schema.

3.5 Universal Service Identifier

The Universal Service Identifier (USI) is a Browserver approach that is not mandatory for the Browserver, as it is possible to implement the Browserver with other unique identifier scheme. We consider it to be a valuable means for unique service identification. A USI is a subset of the URI [11], with the following syntax:

urn:bs:[Browserver][Service][Operation] (1)

[Browserver] = [name]@[subdomain].[domain] (2)

The Service and Operation parts (1) are optional. When consisting only of the Browserver part (2), the it refers to the default Browserver service (the *UIWebService*). The schema of the Browserver services URNs can be exemplified:

- bs:mike@ist.pt/
- bs:mike85@ist.pt/id/Accounting
- bs:john@chunk.us/MathService/squareOp

The Universal Service Identifier has no existing implementations, so, compliant solutions would have to be developed. A solution comprising a distributed hierarchical architecture, like the DNS (that could eventually be adapted), providing services for naming and locating services, and Ad-UDDI [21] would fulfill the needs.

The USI offers a naming schema that can be used to fully resolve a service location, provided that the Web supports it with the necessary systems.

4 Related Work

There is currently no known work which is conceptually closely related to the Browserver, although there are some attempts to give the user the ability to provide services or resources using the browser.

Opera Unite [20] couples a browser and a server, giving the user the possibility of providing some services and resources to other users, but not in a direct fashion. Opera servers are always in the middle, and there is no continuous presence for services or resources in the Web.

In [19], the author tries to get the browser to be seen as web services server, but the method results in sending some notification (email, SMS) to the user with a URL to follow, instead of making a direct request to the browser which the user can fulfill.

Smart Browser [8], intends to provide more processing power, enabling background processing that can change the way things are presented, but doesn't give service provider capabilities to the user, who remains a mere consumer.

Most of current efforts to improve the Web are centered at the user experience as a consumer. The HTML5 draft enables more interactive content, by extending the dynamic UI creation to meet the standards. Anyhow, many capabilities it will bring to the browsers can be done by the local server and, for greater user interaction, also Flash can be used, so the choice isn't limited.

Still in a draft state of a standard protocol [2] and API [1], Web Sockets promise to enable seamless bi-directional communication and, consequently, much lower latency in connections between browser and server, even through intermediary proxies and firewalls (if encryption is used). The Browserver might eventually benefit from the use of such technology for communication, although the direction the technology is heading does not put the user in a provider position, as the services still remain at the servers.

5 Rationale and Conclusions

This work intends to be a first approach to the development of the Browserver, and instigate discussion over the best solutions to it as no system today implements its features. An implementation of the Browserver is under development as a demonstration of the concept, with the architectural design presented by this article mostly implemented and functional.

The Browserver is intended to be a platform for the Internet of Services and can change the way Web applications are designed. People, the leaves of the current Web, can be invoked as if they were Web Services. Workflows can be implemented by knowing that each participant is able to perform a task and to provide a service, directly requested (as in a real business process) and not relying on the user's willingness to follow an URL.

Nowadays, collaborative work is made mostly using central servers. Most companies prefer using their own infrastructure as a security and privacy measure. As in [15, 7], the Browserver eliminates the intermediaries in communication, therefore providing a platform for more secure and private collaboration environments. The e-mail is one of the applications that can be redesigned to send messages directly to the addressee or to feed them through some trusted third-party with user defined encryption mechanisms.

New and existing large-scale applications can be built or redesigned by knowing that the client has the ability to perform server-side tasks, lowering the load on the application servers. New P2P social networks are also a targeted application area. We can maintain a social network by keeping the URNs of all our connections, instead of having them all stored in some server.

Not all the current technologies are well suited for the Browserver. NAT constitutes an obstacle to P2P networks like the one the Browserver intends to build, and the existing solutions are not optimal. While Web Services are still the most used standard technology to implement the service paradigm, their complexity and sluggish performance constitute an opportunity for alternatives that best suit performance and scalability, such as WOA and REST [14]. However, expressiveness is not the strongest point in REST. The convergence of the two approaches is now the focus of study and development [17]. Peer-to-Peer networks using Web Services have already been addressed by [6, 3, 5, 12].

References

[1] The WebSocket API. W3C Working Draft, June 2010. http://dev.w3.org/html5/-websockets/, last access on 2010-07-14.

[2] The WebSocket protocol. IEFT Draft, May 2010. http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00, last access on 2010-07-14.

[3] Conrad, M., Dinger, J., Hartenstein, H., Schöller, M., and Zitterbart, M.: Combining service-orientation and peer-to-peer networks. In KiVS Kurzbeiträge und Workshop, p. 181–184, 2005.

[4] Crane, D. and McCarthy, P.: Comet and Reverse Ajax: The Next-Generation Ajax 2.0. Apress, Berkely, CA, USA, 2008.

[5] Galatopoullos, D.G., Kalofonos, D.N., and Manolakos, E.S.: A P2P SOA enabling group collaboration through service composition. In ICPS '08: Proceedings of the 5th international conference on Pervasive services, pages 111–120, New York, NY, USA, 2008. ACM.

[6] Harrison, A., and Taylor, I.: Dynamic web service deployment using WSPeer. In Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, pages 11–16. Louisiana State University, February 2005.

[7] Kortuem, G., Schneider, J., Preuitt, D., Thompson, T. G., Fickas, S., and Segall, Z.:When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. Peer-to-Peer Computing, IEEE International Conference on, 0:0075, 2001.

[8] Lin, D., Jin, J., and Xiong, Y.. Smart Browser: A framework for bringing intelligence into the browser. volume 7540, Tower A505 SP Tower, Tsinghua Science Park, HaiDian District, Beijing, China, 100084, 2010.

[9] Marginian, D. and Walke, J.: Direct Web Remoting - easy Ajax for Java, 2010. http://directwebremoting.org/, last access on 2010-06-07.

[10] Sun Microsystems. Metro, open source web service stack, 2009.

[11] T. Berners-Lee, Fielding, R., and Masinter, L.: RFC 3986, Uniform Resource Identifier (URI): Generic syntax. Request For Comments (RFC), 2005.

[12] Mondejar, R., Garcia, P., Pairot, C., and Skarmeta, A.F.G.: Enabling wide-area service oriented architecture through the p2pweb model. In WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pages 89–94, Washington, DC, USA, 2006. IEEE Computer Society.

[13] Tim O'Reilly: What is web 2.0: Design patterns and business models for the next generation of software. MPRA Paper 4578, University Library of Munich, Germany, March 2007.

[14] Pautasso, C., Zimmermann, O., and Leymann, F.: Restful web services vs. "big"' web services: making the right architectural decision. In WWW '08: Proceeding of the 17th international conference on World Wide Web, pages 805–814, New York, NY, USA, 2008. ACM.

[15] Reif, G., Kirda, E., Gall, H., Picco, G.P, Cugola, G., and Fenkam, P.: A web-based peer-to-peer architecture for collaborative nomadic working. In 10th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (Wetice), pages 334–339. IEEE Computer Society Press, 2001.

[16] John Resig. JQuery: The write less, do more, javascript library, 2010.

[17] Schroth, C. and Janner, T.: Web 2.0 and SOA: Converging concepts enabling the internet of services. IT Professional, 9:36–41, 2007.

[18] W3C: Web services architecture, February 2004. http://www.w3.org/TR/ws-arch/, last access on 2010-07-04.

[19] Waldorf, J.A., Lu, Y., and Demetriades, A.: Web browser as web service server in interaction with business process engine. Patent US 2005/0182768 A1, Aug 2005.

[20] Opera: Opera Unite. http://unite.opera.com/, last access on 2010-07-14.

[21] Du, Z., Huai, J., and Liu, Y. Ad-UDDI: An active and distributed service Registry. In C. Bussler and M.-C. Shan, editors, 6th VLDB Int'l Workshop on Technologies for E-Services, volume 3811 of LNCS, pages 58–71. Springer, 2006.

[22] Weerawarana, S., Curbera, F., Leymann, F., Storey, T, and Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.