Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 9, 2011

H. Tschofenig
Nokia Siemens Networks
B. Aboba
Microsoft Corporation
J. Peterson
NeuStar, Inc.
D. McPherson
Verisign

March 8, 2011

Trends in Web Applications and the Implications on Standardization
draft-tschofenig-post-standardization-00.txt

Abstract

Advancements in the design of web browsers have introduced fundamental changes to the architecture of application protocols. The widespread availability and growing sophistication of JavaScript interpreters in browsers enables web servers to push to browsers all of the application logic required to implement a client-server protocol. Consequently, many client-server applications that once required an installed client on a host computer now can rely simply on a modern browser to act as a client for the purposes of a particular application. For example, where once email clients required a custom application to access an inbox, increasingly a web browser can serve this purpose as well as the purpose-built applications of the past. Similarly, HTTP with the assistance of JavaScript can subsume the functions performed by the protocols like POP3 and IMAP. The need for Internet standards beyond HTTP to implement an email inbox application consequently diminishes - why author standards and worry about interoperability of clients and servers when the server can simply push to the client all the code it needs to be interoperable?

Many client-server applications on the Internet could potential migrate to this post-standardization environment. In this environment, there is of course still a role for the IETF to play: existing working groups like HyBi and OAuth are examples of areas where standards work is still required to support this application development paradigm. Collectively, we need to identify areas where the standardization is unlikely to be relevant in the future, and focus our efforts on those areas where our application designs will remain impactful.

Status of this Memo

This Internet-Draft is submitted in full conformance with the

provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 9, 2011.

Copyright Notice

Table of Contents

1. Introduction

The generic nature of the personal computer has enabled the information technology community to develop applications that move functionality available in the offline world into the digital world. This flexibility has introduced security issues, since it is difficult for end users to judge the trustworthiness of downloaded programs in any reasonable way. Consequently, many users are very suspicious about any download they are asked to accept. However, an important aspect for ensuring speed of innovation is to reach widespread deployment of a new technology, which to a large extent requires the ability to run new code on end devices. With operating system updates happening less frequently and the acceptance for software downloads decreasing the browser with its limited capabilities was seen by many as an ideal platform for running code that could not cause harm. In particular, JavaScript has found widespread deployment in browsers 'under the radar' of many companies and is now referred as the 'assembly language of the Internet'. JavaScript was initially perceived as being quite limited in functionality. This perception has changed over the last couple of years when it became the scripting language implemented in the majority of browsers.

For application developers writing code running on Web servers as well as for applications that are downloaded to the end device the desire was always to develop the application once without having to consider all the different runtimes (operating systems or browsers). Now, with the PC and the cellular phone segments getting increasingly blurry this desire is stronger than ever considering the increased

number of obstacles that have to be dealt with. For example, it is highly unlikely that an application will work on various different devices even if all the devices were produced by a single mobile phone vendor. As a consequence, writing cross-platform applications that can be deployed on a variety of target devices has always been difficult. Getting users to download new applications, and to install software updates also leaves software developers in a difficult situation.

How can software be developed so that it can (1) be updated instantly when a new version becomes available, (2) be used across a wide range of devices, and (3) be as powerful as regular desktop applications? This sounds almost impossible but with the increased capabilities of Web browsers and in particular JavaScript it seems that the Internet community has gotten a couple of steps closer to achieve this goal.

User experience has changed largely due to global coverage of high speed cellular networks, a range of new end devices (such as netbooks, smart phones, and Internet tablets), lower costs for

Internet access (often bundled with flatrate tariffs), and the shift of the industry towards moving storage and computation into the network. The younger generation of Internet users today has a very different Internet experience than users 10 years ago. They just click to a Web page of an application service provider and let the browser execute JavaScript without any need to install new applications. A positive side effect is that lower configuration requirements are imposed on the user.

Today, many of the features previously available only with dedicated browser plugins, such as Adobe Flash [1] and Microsoft's Silverlight [2], will become widely available as standardized versions with HTML5 [3]. The expectation therefore is that new versions of browsers will be shipped with these increased functionalities, thereby empowering Web application developers.

This document focuses on the impact for the standardization community, and to provide some recommendations.

2. Impact for the Standardization Community

In the application area communication protocols often follow the pattern where an end host utilizes some application service provider for communication setup and sometimes also for message routing towards the other communication end point. In this article we call this communication legs, or legs for short. This is true for the Post Office Protocol (POP) [4] and for the Internet Message Access Protocol (IMAP) [5], as well as for real-time communication protocols like the Session Initiation Protocol (SIP) [6] and the Extensible Messaging and Presence Protocol (XMPP) [7]. The same can be observed also at lower layers in the protocol stack, such as in various tunneling and mobility protocols where the 'rendezvous service' provides the initial contact point for communication (and may even remain on the end-to-end communication path for the duration of the

communication). Standardization efforts often assume a model where all legs should or need to be standardized, namely the end host to application service provider and the cross domain interaction.

While many standardization efforts in the IETF have considered the possibility for using proprietary protocols along the end host to application service provider leg, this has usually been considered as exception or a transition case. With few exceptions it was assumed that the desired end state is to move from a proprietary protocol to the standardized alternative in the long run, which allows client software vendors to interact with all forms of application service providers. Such an approach increases the need for standardization considerably and requires far more interoperable network elements to exist. This attitude is not particularly surprising given that many standardization participants in the real-time communication area look back to a regime that exactly follows a highly standardised eco-system, namely the telecommunication business.

Email functionality offered by IMAP4 and POP3 is already being replaced by an Ajax-based browser experience. Asynchronous JavaScript and XML (Ajax) [8] is usually referred to as a combination of tools that allows a developer to retrieve data from a Web server in real-time. Interactions demanding real-time communication, such as instant messaging and chat, are possible without any additional plug-ins. Voice over IP and video support that requires access to microphone and cameras is available in browsers but still requires plug-in support today.

Allowing application developers to write code that is downloaded to the end host when a user initially accesses the application service is attractive and allows for fast innovation cycles. Within the IETF the areas that are most impacted by this trend in the Web application domain are quite naturally the 'Applications Area' as well as the

'Real-Time Applications and Infrastructure Area'. Implications for security and transport layer mechanisms are to be expected as this possibility becomes a reality.

Inevitably, the functions of many real-time communications protocols will migrate to web browsers, for the simple reason that they match the modalities of web communication: the Session Initiation Protocol, for example, is essentially a rendezvous protocol implemented over an

interactive messaging layer, and the flows of that messaging layer are easily replicated by web sockets. The challenge, however, is that real-time communications protocols do not map onto the client/server paradigm as easily as POP3 or IMAP. Email relies on client/server protocols that allow users to interact with their local domain, but uses SMTP, a separate interdomain protocol, to send mail between domains. Similarly, real-time communications protocols tend to have a client/server component that interacts with the local domain and a server-to-server component n in the case of SIP, the same protocol serves both functions. While it is clear that the web can subsume the client/server function, could mail delivery similarly move to some sort of interdomain server-to-server variant of HTTP? Thusfar, that has prevailed in the mail world. When examining real-time communications protocols, one must similarly ask what protocols will be used to cross domain boundaries outside of the client/server realm, and what are the implications for security, capability negotiation, and similar core protocol functions when one introduces interworking at the domain boundary.

3. Challenges

Even though a number of new building blocks are being made available at rapid speed, such as HTML5 and various JavaScript extensions, there are still a number of limitations in today's browser environment that prevent a broad range of applications from being executed in the browser. We list a couple of those challenges, some of which will be resolved in a year or two, while others will remain a challenge for a long time.

3.1. Performance Limitations

JavaScript was not designed with high-performance in mind. Indeed, over many years very little attention was paid to boost the performance until recently when the Google JavaScript engine V8 [9] started to compile JavaScript code directly into machine code when it is first executed. More details about the design can be found at [10].

A more serious limitation is the graphics capabilities in browsers. Efforts are under way to enhance the API capabilities, for example WebGL [11] bringing 3D graphics to the browser with features similar to OpenGL ES 2.0 that can be used in HTML5 canvas elements but expensive computations on the end host need to migrate from the Central Processing Unit (CPU) to the Graphics Processing Unit (GPU) for proper performance. Simple 3D games (similar to the recently demonstrated Quake II port to HTML5 [12] utilizing JavaScript, the WebSocket API [13] and the Web Storage API [14]) can now be implemented but state-of-the-art games and virtual worlds are out of reach. The problem is with the number of polygons that many games and virtual worlds need to process and display. Games, like Quake, use a limited number of textures, and the complexity of the scene graph is small.

In comparison to virtual worlds where the content is put together by users, in many games the playing field is carefully designed by experts. This has implications for the complexity of the scene graph. On the other hand, most virtual worlds do not rely on rapid communication updates in the same way that many action and tactic games do. Joshua Bell illustrated this with an example of 'a quiet scene with a single user running around in SecondLife [15]. A teleport to a region can easily have a scene graph with 2000 nodes, a couple hundred 3D textures, 4000 vertexes, and 20 MByte of vertex data. This corresponds to the maximum a graphics developer would typically like to have in a state-of-the-art game. In a busy scene with lot of user generated content and avatars the volume easily jumps up by a factor of five.' [16]. The size of the game itself (often due to the high quality textures) and software updates is

impressive; often reaching beyond several 100 Mbytes. Utilizing persistent storage and caching in combination with more aggressive client-server interactions demands a different style of programming and therefore also puts different constraints on the protocol design. This might also stress the current Mbyte limits for Web storage.

Initial work to deal with more sophisticated graphics computation has started already, as described in the recently published article [17] about elevating JavaScript performance through offloading processing to the GPU. As stated in the announcement of the Jetpack 0.5 contest [18]: 'By giving webpages and add-ons easy access to the raw processing power available on most computers, the range of abilities that the web can have greatly increases.'.

3.2. Transport Protocol Limitations

In [19] Jonathan Rosenberg argued that the new waist of the Internet hourglass is UDP and TCP, rather than IP as in the initial design. Today, application protocol designers may, however, get the impression that tunneling inside HTTP or even HTTPS is required to get an application running in a large number of environments, especially to reach a customer base that is connected to the Internet through an enterprise network. Needless to say that more complex tunneling leads to more complexity, the data transport adds overhead and the initial environment sensing phase adds delays. This is certainly true for the VoIP context where the payload data is comparatively small to the overall header size (including the TCP/HTTP headers). The work on Interactive Connectivity Establishment (ICE) [20] is relevant for the sensing phase and this functionality may need to be replicated in the browser environment. Worse than inefficiency is that some real-time applications do not behave well with the retransmission behavior of TCP. For real-time voice and video applications, for virtual worlds, and for many games it is acceptable to loose video and voice frames from time to time without waiting for retransmission.

Adding the support for UDP to browsers again adds complexity, as the experience with Voice over IP showed, particularly when the protocols are not multiplexed together, so that it is necessary to identify multiple working end-to-end paths for the traversal of Network Address Translators (NATs) and firewalls. With the increased IPv6 usage the number of NATs is likely to increase during a long transition period. Furthermore, in many cases it might be desired to perform route optimization for data traffic and to exchange it directly between the two endpoints whenever possible to reduce the financial costs and the added delay of using an anchor point. For

example, Google Talk only requires the involvement of relays for 8% of their calls, as reported in [21] by utilizing ICE.

It should be noted that audio and video streaming capabilities have been available in the browser for a while with plug-in support. More sophisticated audio support, such as tagging audio with x/y positions for 3D audio, is not even possible with the Adobe Flash application today. The challenge with video support in browsers is based on the lack of universal support of a specific video codec. The lack of hardware support is secondary although relevant for increased performance and lower energy consumption. Naturally, supporting different codecs makes the work of web developers and content distributors difficult.

3.3. Security, Privacy, and Cryptographic Processing Limitations

Many protocol mechanisms have several built-in cryptographic primitives and and the same capabilities must be available in the browser in order to move migrate applications that use these capabilities. For example, JavaScript allows cryptographic operations to be implemented (see [22] for a JavaScript AES or other cryptographic functions [23] implementation) but access to hardware crypto-processors, smart cards [24] or to key storages from JavaScript is still at an early stage. The authors are also not aware of a shared authorization policy store that allows a number of websites to benefit from a central user preferences store, such as settings regarding the distribution of location information. It is quite likely that users might prefer to control their privacy settings in one location, given a specific context, and have those settings applied to all running Web applications to avoid private information leakage.

The security model of JavaScript is rather weak in comparison to those of Widgets [25] (available with different platforms/operating systems, such as Mac OS X (via the dashboard), Windows 7, Opera, etc.). JavaScript code does not declare what operations it is intended to perform. Even with Widgets the question is who will verify any of these privileges. It can hardly be assumed that the end user will be bothered with such a responsibility (due to the lack of his or her expertise in making reasonable decisions). Furthermore, the semantic of end-to-end security is challenged when the distinct communication legs support protocols with different

semantics, and dissimilar encodings. Imagine a browser that sends location data encoded in JSON [26], for example using [27], to a web server, which converts it to XML, for example into the PIDF-LO format [28] to interoperate with another application service provider. Consequently, this server then uses XMPP to deliver notifications to its users, for example using [29]. No two of these encodings offer the same privacy mechanisms nor security properties.

From the work in the W3C Geolocation [30] and the W3C Device API and

Policy [31] working group it was observable in recent years that attempts to incorporate privacy mechanisms beyond notice and choice are hard to accomplish with community consensus. While many of these user-interface indications barely work on a PC with a large screen, keyboard and mouse, it remains to be seen how successful they will be on devices with display and input constraints. For a more detailed discussion of privacy challenges related to initial implementations of the Geolocation API see [32].. Further privacy related information can be found with the recent 'W3C Workshop on Privacy for Advanced Web APIs' [33].

The privacy implications of a heavily JavaScript-centered Web environment are not yet well understood. For example, the SIP privacy mechanisms, described in [34], [35], and [36]) rely to a large degree on the end point to select independent RTP/SRTP relays, and to obfuscate important header fields based on the context provided by the user. When the executable code itself is provided by the application service provider (rather than an independent software client vendor) then the privacy functionality for data minimization can change at any point in time with little possibility that the user will notice.

3.4. Source Code Hiding Limitations

In many commercial environments it is not desirable to make source code available to the public. With JavaScript the source code is sent from the server to the browser and only compression and obfuscation tools are available [37]. However, the only way to protect code is to not expose it to observers, instead leaving the important code on the server-side and have a minimal public Javascript code segment use asynchronous message exchanges with the server. Developers in the past rarely had to worry about such a

design criteria; how it will impact protocol design?

4. Recommendations

This section lists a couple of questions for protocol authors. We hope that in answering these questions honestly a thought process will be triggered that may lead you to re-consider your design before starting a year-long standardization effort that may not lead to successful deployment. Note: We use the term 'protocol' below to refer to a protocol extension, a protocol, or to a complete protocol suite, or an entire architecture.

1. Does your standardization effort fall priminarily into the client-to-server interaction described in this document? If the answer is "yes", is there a story how the involved stakeholders can innovate at the same speed as in the architecture described in this document? If you do not have a credible answer to the latter question you will run into trouble. If your answer is "no", then you may skip the rest of the questions. Your protocol may, for example, focus on backend server interactions or dealing with lower-layer interactions.

2. Are you attempting to offer functionality typically found at the application layer at the lower layers (such as network layer)? If so, have you carefully investigated the cost vs. benefit tradeoff?

3. Does your protocol design involve stakeholders who are not aligned with the goals of your envisioned deployment, i.e. for successful deployment do you require cooperation of stakeholders who may have disincentives (or unclear incentives) to deploy your protocol? Are there other architectural variants that allow innovation to happen at a higher speed?

4. When designing your protocol have you considered the Web application environment? Do you understand Web development or do you have experts from the Web development community involved in your work? If the answer to this question is "no" then might miss some important concepts.

5. Are there ways for your protocol to be carried on top of HTTP/HTTPS? If the answer is "no", do you understand that a certain user class will not be able to use your protocol?

6. Are your protocol requirements not met by the current Web framework? (For the limits of the current Web framework see Section 3?) If the answer is "no" then you may have a few years time before the functionality is available even thought plug-ins would allow your desired functionality to be deployed today already. Since your requirements may be special already you are

targeting a small community. Is a several year standardization effort justified to satisfy the need of that community or are there other ways to serve your user base?

7. Have you implemented your protocol in a tyipcal Web development programming langage? If the answer is "no" then you might not know whether there are challenges with the usage in a Web context. You may be using, for example, an encoding that is foreign to Web application developers or you may demand functionality that requires browser extensions.

8. Is your protocol deployed already? If the answer is "no", who is going to deploy it? Particularly interesting is the case when none of the standardization participants have a substantial impact on deployment. In that case you are hoping that someone else finds it useful and you could as well publish an academic paper instead.

5. Conclusions

This document to highlight recent trends in Web application communities with impact to Internet standardization. In a nutshell, there is a certain class of applications for which the standardization need is diminishing: chances are good that your standardization work will not be relevant relevant in such an environment.

A lot of this change is driven by JavaScript and HTML5 executed on the end host (typically in the Web browser) while server-to-server

communication is not yet impacted. We are, however, already seeing server-side JavaScript implementations. NodeJS [38] is such an example that is built on top of the V8 JavaScript engine. It runs multiple concurrent JavaScript execution engines in one thread allowing to develop a massively concurrent Web server in JavaScript, addressing a typical pain point for server developers when implementing distributed systems. As another example, CommonJS [39] defines APIs that handle many common application needs, including those that go beyond the usage in Web browsers (such as regular command line programs).

Hence, just as the barriers for rapidly deploying code have dropped on the client side; the server side will likely follow.

Even if there are challenges for standardization there are other areas where work is needed:

- The development of of protocol mechanisms to support a larger range of applications will have an important role to play in the future. Examples of such efforts include the currenly ongoing work on 'BiDirectional or Server-Initiated HTTP' in the HYBI working group [40]. For future work on improving the performance of the Web, for example [41], improvements in HTTP, or common security functionality for the Web as standardized in the Web Security working group [42].

- In those areas where application islands want to interact with larger eco-systems the need for cross-domain communication arises. Often, this is done in a proprietary way but for larger distributed systems and for common functions standardized solutions are valuable. This can be observed today within the VoIP environment, although much slower than expected, in the case of Voice over IP peering but also in the Internet identity management community under the umbrella of 'data portability' [43]. As recent IETF work in this area the Open Authentication Protocol (oauth) [44] working group could be referenced. OAuth deals with more sophisticated security protocol interactions that

require multiple parties to participate in an interoperable way.

- Everyone knows that protocol design is hard regardless whether it happens inside a standards developing organization, like the IETF

or W3C, or in some other less structured community. For Web developers the standardization results are often only visible if they appear in form of rich JavaScript libraries and development frameworks, such as JQuery [45], the Prototype JavaScript Framework [46], MooTools [47], YUI [48] and Narwahl [49]. In order to have an impact in the Web community it is essential for working groups participants to think about how to their protocols can be deployed in a Web environment, for by making JavaScript implementations available. The desire in the standards developing community, including the IETF, to be programming language agnostic and to avoid API standardization may need to be re-visited in light of these recent developments. Extending JavaScript may, for example, require new Document Object Models (DOMs) [50] and these could serve as a valuable contribution.

Offering almost unlimited capabilities to JavaScript/HTML running in a browser (in the same style as native applications run in an operating system environment) will raise security concerns and will consequently require countermeasures (such as 'deep inspection' and blocking). This in turn will sparkle new ideas to bypass limitations introduced, for example by utilizing new scripting languages with different capabilities, etc. This is an arms race that the IT industry is already able to observe already with deep packet inspection firewalls and peer-to-peer networks during the last few years.

It is unavoidable to get the impression that the hard problems, particularly to security concerns regarding the distribution of new software in whatever form, have not been tackled. Instead, the browser becomes the new operating system, inherits the same weaknesses and is likely to share the same fate.

6. Security Considerations

This document includes discussions related to security.

7. IANA Considerations

This document does not require actions by IANA.

8. Acknowledgements

The authors would like to thank Gonzalo Camarillo, Robert Sparks, Alissa Cooper, Blaine Cook, Alexey Melnikov, Peter Saint-Andre, Jonathan Rosenberg, Lisa Dusseault, Joshua Bell, John Hurliman, Meadhbh Hamrick, Mark Nottingham, Anders Rundgren, Markus Isom[/1000]ki, Spencer Dawkins, Jan Kall, Jan Ignatius and Thomas Roessler.

9. Informative References

[1] "Adobe Flash Player", Sep 2010.

[2] "Microsoft Silverlight", Sep 2010.

[3] "W3C HTML Working Group Charter", Sep 2010.

[4] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.

[5] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[6] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[7] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[8] "Ajax (programming)", Sep 2010.

[9] "V8 JavaScript Engine", Sep 2010.

[10] "V8 JavaScript Engine - Design Elements", Sep 2010.

[11] "WebGL", Sep 2010.

[12] "Quake II Google Web Toolkit (GWT) Port", Sep 2010.

[13] "The WebSocket API", Sep 2010.

[14] "Web Storage", Aug 2010.

[15] "Second Life", Sep 2010.

[16] "Private communication between Joshua Bell, Hannes Tschofenig and Jon Peterson about browser performance limitations", Aug 2010.

[17] "Elevating JavaScript Performance Through GPU Power", Jan 2010.

[18] "Jetpack 0.5 Contest: A Winner", Nov 2009.

[19] Rosenberg, J., "UDP and TCP as the New Waist of the Internet Hourglass", draft-rosenberg-internet-waist-hourglass-00 (work in progress), February 2008.

[20] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[21] "Google Talk for Developers: Important Concepts", Sep 2010.

[22] "JavaScript Implementation of AES Advanced Encryption Standard in Counter Mode", Sep 2010.

[23] "crypto-js: JavaScript implementations of standard and secure cryptographic algorithms", Sep 2010.

[24] "JavaScript Crypto", Sep 2010.

[25] "W3C Web Applications (WebApps) Working Group", Sep 2010.

[26] "JavaScript Object Notation (JSON)", Sep 2010.

[27] "The GeoJSON Format Specification", Jun 2008.

[28] Peterson, J., "A Presence-based GEOPRIV Location Object Format", RFC 4119, December 2005.

[29] "XEP-0080: User Location", Sep 2009.

[30] "W3C Geolocation Working Group", Sep 2010.

[31] "Device APIs and Policy Working Group", Sep 2010.

[32] Doty, N., Mulligan, D., and E. Wilde, "Privacy Issues of the W3C Geolocation API, UC Berkeley School of Information Report 2010-038", Feb 2010.

[33] "W3C Workshop on Privacy for Advanced Web APIs", Jul 2010.

[34] Peterson, J., "A Privacy Mechanism for the Session Initiation Protocol (SIP)", RFC 3323, November 2002.

[35] Munakata, M., Schubert, S., and T. Ohba, "Guidelines for Using the Privacy Mechanism for SIP", RFC 5379, February 2010.

[36] Munakata, M., Schubert, S., and T. Ohba, "User-Agent-Driven Privacy Mechanism for SIP", RFC 5767, April 2010.

[37] Crockford, D., "(JavaScript) Minification v Obfuscation", Mar 2006.

[38] "nodeJS", Sep 2010.

[39] "CommonJS", Sep 2010.

[40] "IETF BiDirectional or Server-Initiated HTTP (hybi) Working Group Charter", Mar 2011.

[41] "Let's make the web faster", Sep 2010.

[42] "IETF Web Security (websec) Working Group Charter", Mar 2011.

[43] "Data Portability Project: Share and Remix Data using Open Standards", Sep 2010.

[44] "IETF Open Authentication Protocol (oauth) Working Group Charter", Sep 2010.

[45] "jQuery: The Write Less, Do More, JavaScript Library", Sep 2010.

[46] "Prototype JavaScript framework: Easy Ajax and DOM anipultion for dynamic web applications", Sep 2010.

[47] "MooTools - a compact javascript framework", Sep 2010.

[48] "Yahoo! User Interface Library 3", Sep 2010.

[49] "Narwhal - A general purpose JavaScript platform", Sep 2010.

[50] "Document Object Model", Sep 2010.

Authors' Addresses

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6

Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Bernard Aboba
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: bernarda@microsoft.com

Jon Peterson
NeuStar, Inc.
1800 Sutter St Suite 570
Concord, CA 94520
US

Email: jon.peterson@neustar.biz

Danny McPherson
Verisign
US

Email: danny@tcb.net