Benefits and Pitfalls of Using HTML5 APIs for Online Experiments and Simulations

http://dx.doi.org/10.3991/ijoe.v8iS3.2254

P. Garaizar, M.A. Vadillo and D. López-de-Ipiña
University of Deusto, Bilbao, Spain.

***Abstract*—The most recent advances in the architecture of the Web allow using it as an excellent platform to deliver experiments and simulations over the Internet. However, there are still some challenges related to the animations' accuracy, to user input collection or to real-time communications that have to be accomplished to properly port native application- based experiments and simulations to the Web. The limitations of the standards preceding HTML5 have forced web developers to embed non-HTML objects using a wide range of non-standard plugins and causing an extremely fragmented execution environment where features must be implemented several times in different programming languages to guarantee full compliance with every user-agent. As HTML5 provides a standard -yet fully-featured- environment to develop and execute applications, web user-agents are now more similar to application players than to simple Internet browsers. In this paper we analyze the benefits and pitfalls of these new Application Programming Interfaces (APIs), providing examples of both good and bad instances of research-related use.**

***Index Terms*—Application programming interface, Best practices, Internet, Standards.**

I. INTRODUCTION

During the last two decades, the Web has evolved from the transfer of read-only and static hypertextual data to the "Read/Write Web", achieving one of the goals initially proposed by its designers [1]. Real-time responsive interfaces are nowadays the standard way of interaction in the Web. Current users do not only browse (i.e., inspect something leisurely and casually), but also play applications on the Web (e.g., Gmail for e-mail, Google Docs for word processing, Youtube for multimedia content sharing). This shift from native to web-based applications has also changed education. On the one hand, webinars and Massive Open Online Courses (MOOCs) [2] are becoming more and more popular as an easy way to deliver learning materials (i.e., declarative knowledge). On the other hand, web-based remote laboratories provide platforms to practice (i.e., procedural knowledge).

From a technical point of view, the limitations of the first versions of the HTML standard [4] have forced web developers to combine a wide range of non-standard technologies (e.g., Java, Flash, Silverlight) to achieve their goals. This fragmented execution environment for web applications caused multiple implementations of the same feature to avoid incompatibilities. A fully-featured and standard execution environment was needed. HTML5 and related standards (e.g., WebGL. SVG, CSS3) [5] try to solve this issue providing native-like user experiences within the web browser. In this paper we analyze the benefits and pitfalls of using them regarding online experimentation and simulations.

II. ANIMATIONS

There are several technologies and APIs related to the HTML5 standard that can be used to generate animations. Some of them are part of the HTML5 specification (e.g., Canvas 2D API [5]), and some others are defined in different specifications (e.g., WebGL [6]). Although this distinction is clear, many web browser vendors (i.e., Microsoft, Apple, Google, Mozilla, Opera) usually refer to all of them as HTML5-related technologies, and so we do in the rest of this paper.

Creating animations in HTML5 is not just adding multimedia content to a HTML document via <video> elements. It also involves other features like Document Object Model (DOM) [7] manipulation over time, event handling, human interaction, synchronization, or real-time data processing. Coding in ECMAScript [8] is needed to run the animation in most of the cases, but some animations can be purely declarative. Both procedural and declarative animations have their pros and cons. Declarative animations are usually easier to create. Developers define what they want (e.g. change the width of a box from 100 pixels to 800 in 5 seconds), but not how the browser should achieve it. Procedural animations are more flexible and less limited to the functionality provided by the specifications [9, 10, 11, 12]. However, developers have to describe how to implement all the changes given in the animation programmatically (e.g. create a loop to increase the width of a box progressively, adding a constant value to the current width of that box each time). As web-based procedural animations usually depend on ECMAScript timers, they could interfere with other interactions using the event queue. Generally, declarative animations are preferred over procedural ones due to their simplicity and the browser optimizations that can be done to run them faster (e.g., GPU acceleration).

A. *Declarative Animations*

Currently there are two standards to define declarative animations in a user-agent (i.e., browser) execution environment: Scalar Vector Graphics (SVG) Animation [12] and Cascading Style Sheets (CSS) Animations [9, 10, 11].

SVG Animation is closely related to Synchronized Multimedia Integration Language (SMIL) Animation [13]. SVG is a host language in terms of SMIL Animation. This means that SVG supports animation elements defined in the SMIL Animation specification (i.e., animate, set, animateMotion, animateColor), but also includes compatible extensions to it (i.e., animateTransform, path, mpath, keyPoints, rotate). Figure 1 shows an example of a

declarative animation of a 200 x 200 px black rectangle being expanded to 800 px of width in 5 seconds, beginning at 00:00:01s and ending at 00:00:06s.

```
<?xml version="1.0" encoding="UTF-8" ?>
<svg xmlns="http://www.w3.org/2000/svg">
<rect id="box" x="100" y="100"
 width="200" height="200" fill="black">
<animate attributeName="width" begin="1s"
 dur="5s" from="200" to="800" />
</rect>
</svg>
```

Figure 1. Declarative animation defined in SVG Animation (SMIL).

There are many benefits of using SVG Animation: a) Accurate and easy to define and synchronize animation; b) No need to use ECMAScript timers; c) No overload of the ECMAScript event queue; d) Standalone SVG files that can be played outside HTML documents. The biggest pitfall is that browser's compliance with these specifications may be incomplete [14].

CSS Animations are defined using several specifications. CSS Transforms API [9] allows transforming CSS-styled elements in a two-dimensional or three-dimensional space using transformation functions (e.g., scaleX, skewY, rotate3d) and related properties (e.g., perspective, transform-origin, transform-style). CSS Transitions API [10] is used to change CSS properties from one value to another smoothly over a given period of time (i.e., using a transition timing function like "ease", "linear", "ease-in", "ease-out", "ease-in-out", or "cubic-bezier"). Finally, CSS Animations API [11] allows declaring a set of keyframes with different transitions (i.e., property, duration, timing function, iteration count, etc.) between them. Figure 2 shows an example of a declarative animation using CSS Animations.

```
@keyframes change {
  0% {
    width: 200px;
  }
  50% {
    width: 800px;
  }
}
div#box {
  animation-name: change;
  animation-duration: 5s;
  animation-iteration-count: infinite;
  animation-timing-function: linear;
}
```

Figure 2. Declarative animation defined using CSS Animations.

The benefits and pitfalls of using CSS Animations are almost the same as those of SVG Animation, except for the ability of SVG files to be played outside HTML documents. However, CSS Animations are more likely to be used by web developers because more HTML elements can be animated using them. Therefore, it is recommended to use CSS Animations rather than SVG Animation as they are more likely to be implemented, upgraded and optimized by browser vendors.

B. Procedural Animations

There are two main reasons for using procedural animations rather than declarative animations: a) The lack of support of the browser, and b) The complexity of the animation precluding its definition in a declarative manner. In any of those cases, ECMAScript timers must be used to change properties over time (see Figure 3). However, timer delay is not guaranteed using ECMAScript timers. In order to achieve the highest frame rate in their animations, some developers use a 0 ms delay timer to redraw them as fast as possible. However, delays smaller than the limit defined by the specification [15] are forced to the use at least the minimum delay (i.e., 4 ms for browsers released in 2010 and onward, 10 ms for previous versions). Moreover, considering that all ECMAScript in a browser window executes on a single thread, asynchronous events (e.g., mouse clicks or timers) are only dispatched when the event queue is free [16]. For this reason, Web workers [17] can be used when running scripts independently in the background is needed (e.g., to calculate the next coordinates of a complex animation in a physics engine), leaving the window event queue free. Using the cross-document messaging API (i.e., postMessage) designed with Web workers in mind, there are cross-browser implementations of zero-delay timers [18]. The ECMAScript closure shown in Figure 5 enables an immediate call to the event handler of the timer, in a similar way to the setImmediate method proposed by Microsoft in the Efficient Script Yielding API [19]. Nevertheless, Web workers have some drawbacks. They are not able to change DOM elements or properties, and they are relatively expensive to create and should not be used in large numbers.

```
// 60 Frames Per Second
var interval = 1000 / 60;

function start() {
  //Define initial status
  setTimeout(animate, interval);
}

function animate() {
  //Change properties
  setTimeout(animate, interval);
}
```

Figure 3. Procedural animation stub using ECMAScript timers.

Considering all these issues, a new API was created to write procedural animations where the browser controls the update rate of the animation [20]. Using the Timing control for script-based animations API developers request the browser schedule an animation frame update, instead of trying to figure out when is the best moment to do it. As long as the browser keeps control of all the running animations using this API, it is in a better position to determine the frame rate such that all of the animations will run as smoothly as possible (see Figure 4). There is also another advantage of using this API regarding power consumption, as it will set up a very low frame rate for animations currently invisible. Using this API for procedural animations is highly recommended, but it is still not widely implemented [21].

Regarding the content, not only CSS properties and SVG files can be animated using scripts, but also new HTML5 related APIs such as Canvas 2D Context [22] and WebGL 2D/3D Context [6]. ECMAScript code can be seamlessly embedded into SVG files to change SVG properties, create new shapes or animate them over time. HTML Canvas 2D Context API works in a different way. It provides a blank canvas (i.e., a blank bitmap) to draw shapes, text or images, apply transformations (i.e., scale, rotate, translate, transform), and change compositing or shadow attributes. Similarly, WebGL enables a 2D/3D context for the canvas element. It is derived from OpenGL ES 2.0 [23] and provides an immediate mode 3D rendering API where OpenGL-like resources (i.e., textures, buffers, framebuffers, renderbuffers, shaders and programs) are represented as DOM objects. Most of the times, choosing the proper API for a specific scenario is not trivial. Table 1 shows their advantages and disadvantages and both good and bad instances of their use in terms of complexity (i.e., nearly every animation can be generated using any of these APIs, but some are easier to program when using one of them and not the others).

```
// 60 Frames Per Second
var interval = 1000 / 60,
    last;

function start() {
  //Define initial status
  last = Date.now();
  requestAnimationFrame(animate);
}

function animate(time) {
  if (time >= last + interval) {
    //Change properties
    last = time;
  }
  requestAnimationFrame((animate);
}
```

Figure 4. Procedural animation stub using Timing control for script-based animations API.

III. USER INPUT COLLECTION

Online experiments and simulations often need to collect user input in real-time. As stated on its specification, in ECMAScript time is measured in milliseconds since 01 January, 1970 UTC (i.e., UNIX epoch) [8]. The same happens with DOM events' timeStamp property, used to specify the time at which the event was created [24]. However, these timestamps are gathered from the system clock and therefore may not have millisecond accuracy in some Operating Systems [25, 26]. Considering this limitation, some developers have included a Java Applet to expose Java's nanosecond timing function to ECMAScript [27, 28] (see Figure 6).

Fortunately, the Java's nanoTime trick is not necessary when High Resolution Time (HRT) API [29] is available. This API provides a monotonically increasing timing function with sub-millisecond resolution not subject to system clock skew or adjustments (see Figure 7).

```
// 60 Frames Per Second
var interval = 1000 / 60,
    last;

(function() {
  var timeouts = [];
  var messageName = "zero-timeout-message";

  function setZeroTimeout(fn) {
    timeouts.push(fn);
    window.postMessage(messageName, "*");
  }

  function handleMessage(event) {
    if (event.source == window &&
      event.data == messageName) {
      event.stopPropagation();
      if (timeouts.length > 0) {
        var fn = timeouts.shift();
        fn();
      }
    }
  }

  window.addEventListener("message",
    handleMessage, true);

  window.setZeroTimeout = setZeroTimeout;
})();

function start() {
  //Define initial status
  last = Date.now();
  setZeroTimeout(animate);
}

function animate() {
  time = Date.now();
  if (time >= last + interval) {
    //Change properties
    last = time;
  }
  setZeroTimeout(animate);
}
```

Figure 5. Procedural animation stub using postMessage based setZeroTimeout [16].

```
import java.applet.Applet;
public class nano extends Applet {
  public long nanoTime() {
    return System.nanoTime();
  }
}
```

Figure 6. Java Applet to expose nanosecond timing function to ECMAScript (excerpt from BenchmarkJS's nano.java).

TABLE I.
COMPARISON OF PROCEDURAL ANIMATION APIS

SVG			
Advantages	Disadvantages	Good Scenarios	Bad Scenarios
Standalone	Slow with a big number of shapes	Conceptual animations with a small number of objects	Very detailed objects with complex animations
Vectorial	Cannot be exported to an image	Scalable animations with controls to zoom-in and zoom-out	Highly responsive and fast games or simulations
Easy to redraw / move shapes		Accessible applications, providing an alternative description of the animation if needed	
		Data charts and plots	

HTML CANVAS 2D CONTEXT			
Advantages	Disadvantages	Good Scenarios	Bad Scenarios
Fast	Redraws	Realistic animations with a big number of objects involved	Drag & Drop authoring tools with events attached to objects
Similar performance with a big number of concurrent elements	Poor text-rendering support.	Interactive graphs and real-time image processing (e.g. fractals)	Animations where text readability is important
Can be exported to an image (`toDataURL`)	Resolution dependent	Videogames	

WEBGL			
Advantages	Disadvantages	Good Scenarios	Bad Scenarios
3D	Lack of support (browser / graphics card)	Real-time rendered 3D visualizations	Accessible applications working out-of-the-box in any device
Extremely fast (3D acceleration / Web Workers)	Problems exporting to an image.	Sophisticated animations 2D simulations with zooming, shading or lighting effects	Low-power consuming applications (e.g., mobile applications)
Easy to port to / from OpenGL		Immersive videogames	

On the whole, dealing with ECMAScript timing functions to collect user input accurately depends on the API support of the user agent. Undoubtedly, HRT API is the best option when available. Relying on Java's nanosecond timing function may be an option when Java support is available. Unfortunately, that setup is uncommon nowadays. Finally, ECMAScript timing functions perform differently depending on the underlying platform [30]. This should not be overlooked by researchers.

```
partial interface Performance {
  DOMHighResTimeStamp now();
};
```

Figure 7. The DOMHighResTimeStamp type and the now method of the Performance interface in High Resolution Time API.

IV. REAL-TIME COMMUNICATIONS

Delivering a simulation or experiment over the Internet usually involves real-time communications. During the last decade, Asynchronous JavaScript And XML (AJAX) [31] based interfaces have become the norm. AJAX enables clients to asynchronously poll for server-side events, but polling is not strictly a real-time communication as it is bounded to the polling interval.

Comet tried to solve this issue, defining mechanisms to allow the server to send information to the user agent without prompting from a client (i.e., push instead of pull). However, Comet implementations are not standard and often they are not interoperable either [32]. Consequently, long polling connections (i.e., asking for a new AJAX connection with an extremely long timeout when the last update is received from the server) are often used as a workaround for the lack of server pushing support, but it is a far from satisfactory solution.

The HTML5 WebSocket specification [33] allows pushing and pulling information through a single-socket full-duplex connection between the browser and the server. It can provide a 500:1 reduction in unnecessary HTTP header traffic and 3:1 reduction in latency [34]. Unfortunately, the WebSockets API is still under development, some security problems have been discovered [35], and there may be compatibility issues across different implementations. Figure 8 shows how these mechanisms can be used to establish a connection that enables server-side notifications.

Finally, another web standard for real-time communications is currently being developed: WebRTC [36]. WebRTC provides real-time communications between browsers (e.g., audio / video conferencing) without requiring proprietary plugins, downloads or installs. WebRTC current implementation relies on two APIs, PeerConnection API [36] and GetUserMedia API [37]. PeerConnection API is able to cover most of the use cases involved in peer-to-peer connections natively from the user-agent (i.e., Network Address Translation using ICE [38] and STUN [39] or TURN [40] servers, and session management using SDP [41]). GetUserMedia API is used to get access from JavaScript to local devices that can generate multimedia stream data (e.g., webcams or microphones). Although there is still a long roadmap ahead implementing WebRTC, working proofs of concept have been released for Google Chrome and Mozilla Firefox browsers. GetUserMedia API is already released for Opera, and announced for Internet Explorer 11 [42].

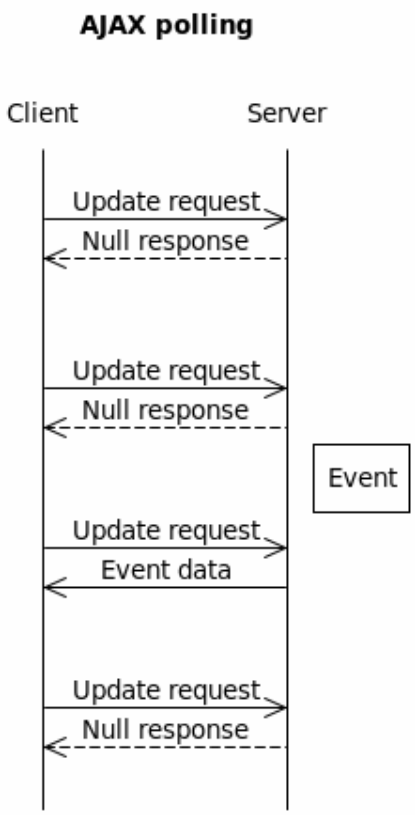

Figure 8a. Server-side notifications mechanisms, AJAX polling.

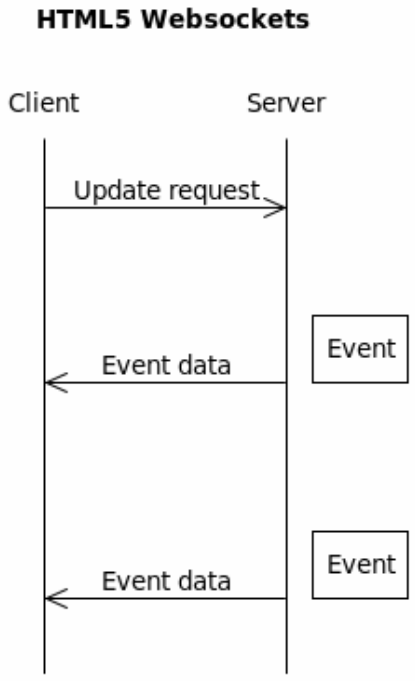

Figure 8c. Server-side notifications mechanisms, HTML5 Websockets.

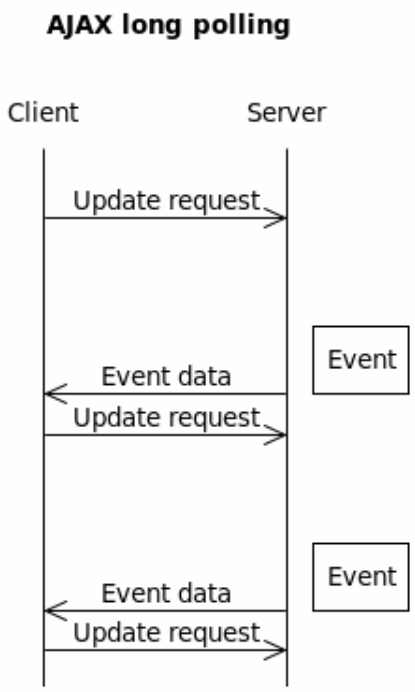

Figure 8b. Server-side notifications mechanisms, AJAX long polling.

V. CONCLUSIONS

Twenty years on from the invention of the Web, its use as a platform for delivering experiments and educational simulations has increased dramatically and a comprehensive range of standard and patent-free specifications are now widely available for researchers and content providers. Nevertheless, there are still some unsolved issues. Many of these specifications are still under heavy development and it is not easy to find living instances of research-related use. The purpose of the present paper was to provide a roadmap to this new land of opportunity for HTML5 based e-research, focusing on three main aspects: a) Animations, b) User input, and c) Real-time communications. For each of them, several recommendations of API usage have been provided related to the special needs of online experimentation. To conclude, benefits of using HTML5 APIs for online experiments and simulations clearly exceed the costs, although related pitfalls and caveats should not be overlooked by researchers.

REFERENCES

[1] T. Berners-Lee, and R. Cailliau, *"WorldWide-Web: Proposal for a hypertexts project"*. Retrieved from http://www.w3.org/Proposal.html

[2] A. McAuley, B. Stewart, G. Siemens and D. Cormie.*"The MOOC Model for Digital Practice."* Retrieved from http://davecormier.com/edblog/wp-content/uploads/MOOC_Final.pdf

[3] J. Garcia-Zubia, P. Orduña, D. Lopez-de-Ipiña, and G.R. Alves, *"Addressing software impact in the design of remote laboratories". IEEE Transactions on Industrial Electronics, 56, pp: 4757-4767, 2009.* http://dx.doi.org/10.1109/TIE.2009.2026368

[4] D. Ragget, *"HTML 3.2 Reference Specification, W3C Recommendation"*, 14-Jan-1997, Retrieved from http://www.w3.org/TR/REC-html32

[5] I. Hickson, *"HTML5. A vocabulary and associated APIs for HTML and XHTML"*. W3C Working Draft 20 April 2012. Retrieved from http://www.w3.org/TR/2012/WD-html5-20120320/

[6] C. Marrin,*"WebGL Specification, Version 1.0"*, 10 February 2011. Retrieved from https://www.khronos.org/registry/webgl/specs/1.0/

[7] P. Le Hégaret, R. Whitmer, and L. Wood, *"Document Object Model (DOM)"*, 2009/01/06. Retrieved from http://www.w3.org/DOM/

[8] Ecma International, *"Standard ECMA-262. ECMAScript Language Specification. Edition 5.1 (June 2011)"*. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[9] S. Fraser, D. Jackson, D. Hyatt, C. Marrin, E. O'Connor, D. Schulze, and A. Gregor, *"CSS Transforms"*, W3C Working Draft 28 February 2012, Retrieved from http://www.w3.org/TR/2012/WD-css3-transforms-20120228/

[10] D. Jackson, D. Hyatt, C. Marrin, and L.D. Baron, *"CSS Transitions Module Level 3"*, W3C Working Draft 01 December 2009. Retrieved from http://www.w3.org/TR/2009/WD-css3-transitions-20091201

[11] D. Jackson, D. Hyatt, and C. Marrin, *"CSS Animations Module Level 3"*, W3C Working Draft 20 March 2009. Retrieved from http://www.w3.org/TR/2009/WD-css3-animations-20090320

[12] E. Dahlström, P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Schepers, and J. Watt, *"Scalable Vector Graphics (SVG) 1.1 (Second Edition)"*, W3C Recommendation 16 August 2011. Retrieved from http://www.w3.org/TR/2011/REC-SVG11-20110816/

[13] P. Schmitz and A. Cohen, *"SMIL Animation"*, W3C Recommendation 04-September-2001. Retrieved from http://www.w3.org/TR/2001/REC-smil-animation-20010904/

[14] A. Deveria, *"When can I use... SVG SMIL animation"*, Retrieved from http://caniuse.com/#feat=svg-smil

[15] I. Hickson, *"WHATWG HTML Living Standard, Last Updated 13 March 2012, 7.3 Timers"*. Retrieved from http://www.whatwg.org/specs/web-apps/current-work/multipage/timers.html#timers

[16] J. Resig, *"How JavaScript Timers Work"*. Retrieved from http://ejohn.org/blog/how-javascript-timers-work/

[17] I. Hickson, *"WHATWG HTML Living Standard, Last Updated 13 March 2012, 9 Web Workers"*. Retrieved from http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html

[18] D. Baron, *"setTimeout with a shorter delay"*, 2010-03-09, Retrieved fron http://dbaron.org/log/20100309-faster-timeouts

[19] J. Mann and J. Weber, *"Efficient Script Yielding"*, Editor's Draft July 28th, 2011. Retrieved from http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/setImmediate/Overview.html

[20] J. Robinson and C. McCormack, *"Timing control for script-based animations"*, W3C Working Draft 21 February 2012. Retrieved from http://www.w3.org/TR/2012/WD-animation-timing-20120221/

[21] A. Deveria, *"When can I use... requestAnimationFrame"*. Retrieved from http://caniuse.com/#feat=requestanimationframe

[22] I. Hickson, *"HTML Canvas 2D Context"*, W3C Working Draft 20 March 2012. Retrieved from http://www.w3.org/TR/2012/WD-2dcontext-20120320/

[23] A. Munshi and J. Leech,*"OpenGL® ES Common Profile Specification Version 2.0.25"*, November 2010. Retrieved from http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf

[24] T. Pixley, *"Document Object Model Events"*. Retrieved from http://www.w3.org/TR/DOM-Level-2-Events/events.html

[25] M. Russinovich, *"ClockRes v2.0"*, Microsoft Technet, 2009. Retrieved from http://technet.microsoft.com/en-us/sysinternals/bb897568.aspx

[26] N. Stewart, *"Millisecond accuracy video display using OpenGL under Linux", Behavior Research Methods, Volume 38, Number 1, 142-145, 2006.* http://dx.doi.org/10.3758/BF03192759

[27] jsPerf. Retrieved from http://jsPerf.com

[28] Benchmark.js. Retrieved from http://benchmarkjs.com

[29] J. Mann, *"High Resolution Time"*, W3C Working Draft 13 March 2012. Retrieved from http://www.w3.org/TR/2012/WD-hr-time-20120313/

[30] J. Resig, *"Accuracy of JavaScript Time"*. Retrieved from http://ejohn.org/blog/accuracy-of-javascript-time/

[31] J. J. Garrett, *"Ajax: A New Approach to Web Applications"*. Retrieved from http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[32] Comet Daily, *"Comet Maturity Guide"*, Version: 0.5 (Dec 12, 2009). Retrieved from http://cometdaily.com/maturity.html

[33] I. Hickson, *"The WebSocket API"*, Editor's Draft 13 March 2012. Retrieved from http://www.w3.org/TR/websockets/

[34] P. Lubbers and F. Greco, *"HTML5 Web Sockets: A Quantum Leap in Scalability for the Web"*. Retrieved from http://websocket.org/quantum.html

[35] L. Huang, E. Y. Chen, A. Barth, E. Rescorla, and C. Jackson, *"Talking to Yourself for Fun and Profit", in Proceedings of W2SP, 2011.*

[36] A. Bergkvist, D.C. Burnett, C. Jennings, and A. Narayanan, *"WebRTC 1.0: Real-time Communication Between Browsers"*, W3C Editor's Draft 16 March 2012. Retrieved from http://dev.w3.org/2011/webrtc/editor/webrtc.html

[37] D. Burnett, and A. Narayanan. getusermedia: Getting access to local devices that can generate multimedia streams 22 December 2011. W3C Editors draft (Work in progress.) Retrieved from http://dev.w3.org/2011/webrtc/editor/getusermedia.html

[38] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. April 2010. Internet RFC 5245. Retrieved from http://tools.ietf.org/html/rfc5245

[39] J. Rosenberg, R. Mahy, P. Matthews, D. Wing. Session Traversal Utilities for NAT (STUN). October 2008. Internet RFC 5389. Retrieved from http://tools.ietf.org/html/rfc5389

[40] P. Mahy, P. Matthews, J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). April 2010. Internet RFC 5766. Retrieved from http://tools.ietf.org/html/rfc5766

[41] J. Rosenberg, H. Schulzrinne. An Offer/Answer Model with the Session Description Protocol (SDP). June 2002. Internet RFC 3264. Retrieved from http://tools.ietf.org/html/rfc3264

[42] H.Alvestrand, and S. Håkansson. Status of W3C WEBRTC. March 2012. IETF 83 Paris - RTCWEB meeting. Retrieved from http://www.ietf.org/proceedings/83/slides/slides-83-rtcweb-7.pdf

AUTHORS

P. Garaizar is with Deusto Institute of Technology (DeustoTech), University of Deusto, Avda. Universidades 24, 48007, Bilbao, Spain (e-mail: garaizar@deusto.es).

M.A. Vadillo is with the University of Deusto, School of Psychology and Education, Avda. Universidades 24, 48007, Bilbao, Spain (e-mail: mvadillo@deusto.es).

D. López-de-Ipiña is with Deusto Institute of Technology (DeustoTech), University of Deusto, Avda. Universidades 24, 48007, Bilbao, Spain (e-mail: dipina@deusto.es).

This work was supported in part by Grants IT363-10 and IT458-10 from the Education, Universities, and Research Department of the Basque Government.