UNIVERSIDAD DE DEUSTO

LA WEB COMO PLATAFORMA DE EJECUCIÓN ADECUADA PARA LA PRESENTACIÓN EXACTA Y PRECISA DE CONTENIDOS AUDIOVISUALES Y EL REGISTRO DE LA INTERACCIÓN DEL USUARIO

Tesis doctoral presentada por Pablo Garaizar

dentro del Programa de Doctorado en Ingeniería Informática y Telecomunicación

Dirigida por Dr. Diego López-de-Ipiña

y Dr. Miguel Ángel Vadillo

UNIVERSIDAD DE DEUSTO

LA WEB COMO PLATAFORMA DE EJECUCIÓN ADECUADA PARA LA PRESENTACIÓN EXACTA Y PRECISA DE CONTENIDOS AUDIOVISUALES Y EL REGISTRO DE LA INTERACCIÓN DEL USUARIO

Tesis doctoral presentada por Pablo Garaizar

dentro del Programa de Doctorado en Ingeniería Informática y Telecomunicación

Dirigida por Dr. Diego López-de-Ipiña
y Dr. Miguel Ángel Vadillo

El doctorando | Los directores | Los directores

Bilbao, Enero de 2013

La Web como plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario

Autor: Pablo Garaizar
Director: Diego López-de-Ipiña
Director: Miguel Ángel Vadillo

Puede encontrarse información acerca de esta tesis doctoral y los trabajos derivados de la misma en la siguiente dirección web:
`http://paginaspersonales.deusto.es/garaizar/`
Impreso en Bilbao
Primera edición, enero de 2013

A Silvia, Peio y Lorea. Os debo cada minuto.

Abstract

During the last decade of the 20^{th} century and the first decade of the 21^{st} century, the Web has evolved into the most prominent platform for developing applications. Nowadays, it is difficult to find a service not provided by the Web. The apparent simplicity of an architecture based on a small set of well-known methods has been able to cover a myriad of different scenarios. The maturity of the Web has been made possible by the latests technological advances in its infrastructure and user agents.

The real-time, ubiquitous, and constantly updated Web is now a reality for most of the web services. However, not all of them have been properly ported to this new execution environment. There are applications not intended to run as fast as possible or instantaneously, but to meet strict temporal constraints. It is necessary to analyze the accuracy and precision of the timing mechanisms provided by the web techonologies in order to enable real-time computing on the Web.

The applications needed to conduct experiments in fields like vision, audition or implicit learning are a particular case of the real-time computing paradigm. Because of their strict temporal constraints in the presentation of multimedia content and the record of user interaction, and because of the lack of evidence for the real-time computing compliancy of the Web, these applications have been traditionally developed using offline technologies. Nevertheless, the recent advances in standarization and performance of web user agents would support the feasibility of developing this kind of applications.

With the aim of elucidating this issue, an analysis of the accuracy and precision of the web technologies has been performed. Both proprietary and standard web technologies have been compared with offline specialized software. As a result of this analysis, we conclude that a combination of the appropriate web technologies allows developing web applications under strict temporal constraints. Using this combination of standard technologies we have developed Labclock Web, a web version of the Libet's clock experimental paradigm, very popular among cognitive scientists. In order to have an offline equivalent of Labclock Web, we have developed Labclock, a Windows native application that provides the same functionality. The accuracy and precision of both applications was tested using the proper measurement equipment. Moreover, a large set of participants ($N > 300$) took part in 6 experiments studying the "temporal binding" effect conducted using Labclock and Labclock Web. Moreover, a set of tools designed to evaluate the accuracy and precision of extant timing mechanisms in a web execution environment is provided. The insight gathered through the previous analyses and the experiments is summarized in a brief list of best practices to develop web applications under strict temporal constraints.

Resumen

A lo largo de la última década del siglo XX y la primera del siglo XXI la Web ha evolucionado hasta postularse como la principal plataforma de ejecución de aplicaciones. La aparente sencillez de una arquitectura basada en un conjunto reducido de operaciones bien conocidas ha sido capaz de cubrir un rango tan amplio de escenarios que resulta difícil encontrar un servicio que la Web no pueda proporcionar. Esta evolución ha sido posible gracias a los avances tecnológicos tanto en su infraestructura como en lo que respecta a sus agentes de usuario.

La Web en tiempo real, ubicua y permanentemente actualizada es una realidad para la gran mayoría de servicios. Sin embargo, no todos ellos han sido convenientemente portadas a este entorno de ejecución. Existen aplicaciones en las que lo fundamental no es maximizar su velocidad de ejecución, sino cumplir con unos estrictos requisitos de exactitud y precisión temporal. Para que este tipo de aplicaciones pueda desplegarse en la Web, es necesario comprobar que las tecnologías de desarrollo disponibles son capaces de proporcionar los mecanismos de temporización adecuados, que permitan implementar no solo aplicaciones *en tiempo real*, sino también aplicaciones *de tiempo real*.

Un caso particular de este tipo de aplicaciones lo constituyen las aplicaciones que requieren la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario, propias de la experimentación en visión, audición o aprendizaje implícito, entre muchas otras áreas. Debido a la falta de evidencia favorable a su despliegue en la Web hasta la fecha, estas aplicaciones han sido mayoritariamente desarrolladas con tecnologías offline. Sin embargo, los recientes avances tecnológicos permiten plantear la viabilidad de implementar

una aplicación con estrictos requisitos de exactitud y precisión temporal mediante tecnologías web estándar.

Con el objetivo de dilucidar esta cuestión, se ha llevado a cabo un estudio acerca de la exactitud y precisión de este tipo de tecnologías, comparándolas tanto con las alcanzables a través de software especializado, como mediante el uso de tecnologías web propietarias. Como resultado de este estudio, se concluye que existe una combinación viable de tecnologías estándar que permite el desarrollo de aplicaciones web con estrictos requisitos temporales. Empleando esta combinación de tecnologías web estándar, se ha desarrollado una aplicación que implementa el paradigma experimental del reloj de Libet ampliamente utilizado en ciencias cognitivas, a través del cual es posible medir el efecto de «unión temporal». Con el fin de obtener una aplicación equivalente de referencia, se ha hecho lo propio a través de las tecnologías nativas adecuadas. El correcto funcionamiento de ambas aplicaciones se ha evaluado tanto mediante equipamiento de precisión como a través de seis experimentos en los que se contó con la participación de un nutrido grupo de sujetos experimentales ($N > 300$). A partir de los resultados obtenidos, se corrobora la viabilidad de la Web como plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario. Fruto de estos desarrollos y de los estudios previos, se proporciona a la comunidad de desarrolladores web un compendio de recomendaciones para el desarrollo de aplicaciones web con estrictos requisitos de exactitud y precisión temporal, así como un conjunto de herramientas destinadas a la evaluación de la exactitud y precisión de los mecanismos de temporización presentes en cada caso.

Laburpena

XX. mendearen azken hamarkadan eta XXI.aren lehengoaren zehar Web-a aplikazioen egikaritze-plataforma nagusia bilakatu arte garatu da. Oso ezagunak diren eragiketa multzo murriztuan oinarritutako arkitekturaren ageriko sinpletasuna agertoki tarte hain zabala betetzeko gai izan da, ezen Web-ak hornitu ez dezakeen zerbitzuren bat topatzea zaila gertatzen dela. Eboluzio hau, bai azpiegitura zein erabiltzaile agenteei dagokien aurrerakuntza teknologikoei esker izan da.

Nonahiko eta etengabe gaurkotutako denbora errealeko Web-a, zerbitzu gehienetarikoentzat errealitatea da. Hala ere, hauek guztiak ez dira egikaritze-ingurune honetara egoki eramanak izan. Bere egikaritze-abiadura maximizatu beharrean, denbora-zehaztasun eta doitasun baldintza zorrotzak betetzea lehenesten duten aplikazioak egon badaude. Aplikazio mota hau Web-ean zehar zabal dadin, garapen-teknologia erabilgarriak denboralizazio-mekanismo egokiak hornitzeko gai direla frogatu behar da, eta *denbora errealeko* aplikazioak ez ezik, *denbora errealerako* aplikazioak ere inplementatzea onar dezaketela.

Aplikazio mota honen kasu berezietako bat, ikus-entzunezko edukien aurkezpen zehatz eta doia zein erabiltzailearen erregistroa behar duten aplikazioak dira; arlo askoren artean, ikusmen, entzumen edo ikaste inplizituaren esperimentazioari dagozkionak. Orain arte, Web-ean hedatze aldeko nabaritasun eza dela eta, aplikazio hauek, gehienbat, offline teknologiez garatuak izan dira. Hala ere, berriki gertatutako aurrerakuntza teknologikoek, web teknologi estandarren bidezko denbora-zehaztasun eta doitasun baldintza zorrotzak betetzen dituen aplikazioaren inplementazioaren bideragarritasuna proposatzeko aukera ematen dute.

Auzi hau argitzeko asmoz, teknologia mota hauen zehaztasun eta doitasunari buruzko ikerketa burutu da, bai xede bereziko softwarearen bidez lortutakoekin zein web teknologia jabedunen erabilerarekin erkatuz. Ikerketa honen emaitzaren ondorioz, denbora-betekizun hertsidun web aplikazioen garapena ahalbidetzen duten teknologia estandarren eskema bideragarri bat dagoela ondorioztatzen da. Web teknologia estandar hauen eskema erabiliz, zientzia kognitiboetan asko erabiltzen den Libet erlojuaren paradigma esperimentala inplementatzen duen aplikazioa garatu da, zeinaren bidez «denbora-lotura» efektua neurtzea posiblea da. Erreferentziazko aplikazio baliokidea lortzeko helburuarekin, egokiak diren jatorrizko teknologien bidez berdin jokatu da. Aplikazio biren funtzionamendu zuzena ebaluatu da, bai zehaztasun ekipamenduaren bidez, bai sei esperimentuen bitartez, non subjektu esperimental-talde joriaren ($N > 300$) parte-hartzea izan zen. Lortutako emaitzetatik abiatuz, Web-a ikus-entzunezko edukien aurkezpen zehatz eta doirako eta erabiltzailearen interakzioaren erregistrorako egikaritze-plataforma egokia bezala bideragarria dela egiaztatzen da. Garapen hauen eta aurretiko ikerketen etekina dela medio, web garatzaileen komunitateari, denbora- zehaztasun eta doitasun baldintza zorrotzak bete behar dituzten web aplikazioen garapenerako gomendio sintesiaz hornitzen zaio, hala nola, kasu bakoitzaren denboralizazio-mekanismoen zehaztasun eta doitasunaren ebaluazioari zuzendutako tresna multzoa.

Agradecimientos

A pesar de que una tesis doctoral pueda parecer un trabajo muy individual, nunca es así. Para poder llegar a rozar las fronteras del conocimiento es necesario que alguien te indique el camino. En la mayoría de casos la ayuda no se limita a la mera indicación, sino que ese camino es compartido y, gracias a la compañía, enriquecido con pequeñas excursiones por los alrededores. Por este motivo, se me hace necesario dedicar unas líneas de agradecimiento a algunas de las personas que han hecho posible este trabajo.

En primer lugar, gracias a Diego y Miguel Ángel, mis guías de viaje, siempre dispuestos a discutir cualquier asunto relacionado con la tesis, a corregir y mejorar detalles, a ayudarme a convertirme en un investigador. Una de las pocas certezas que tengo a este respecto es el indudable acierto que supuso solicitar vuestra supervisión. No suelo comentarlo con vosotros, pero os estoy tremendamente agradecido.

En segundo lugar, a mis compañeros de viaje. Doctorandos y doctores de Labpsico y DeustoTech Learning que habéis hecho que esto sea realmente divertido. En la medida de lo posible, claro, que el título de doctor exige importantes renuncias y mucha dedicación. Echo la vista atrás y me doy cuenta de que haber coincidido con vosotros ha sido un pequeño regalo que la vida académica me ha hecho. Muchas gracias Helena, por haberme permitido formar parte del mejor grupo de investigación que he conocido. Muchas y divertidas anécdotas con vosotros harán que recuerde estos años con especial cariño.

En tercer lugar, a mi familia. Gracias a mi padre, por inculcarme su determinación y el valor del esfuerzo. Gracias a mi madre, por enseñarme

todo lo demás, ya que sus ganas de enseñar nunca han distinguido lugares ni momentos. Gracias a Juliana por demostrarme que las únicas fronteras que existen son las que cada uno dibuje en su mente. Gracias a Luzia por haberme acompañado en tantas locuras tecnológicas domésticas, desde Fidonet a GNU/Linux, pasando por el IRC y otras aventuras. El mundo es un lugar mucho más interesante después de todas las claves que me habéis dado para conocerlo mejor.

Por último, a Silvia, Peio y Lorea. Os he tenido muy presentes durante todo este tiempo. Las renuncias a las que hacía mención antes han sido compartidas, y eso me apena. La parte buena es que este esfuerzo conjunto nos ha supuesto una prueba y la hemos superado. Podría escribir muchas más cosas para agradeceros todo vuestra dedicación, pero creo que es mejor demostrároslo con hechos y no con palabras. Tenemos toda la vida por delante para ello.

Muchas gracias,

Pablo Garaizar

Enero de 2013

Índice general

Índice de figuras

Índice de tablas

Índice de listados

"In programming, the hard part isn't solving problems, but deciding what problems to solve"

Paul Graham

CAPÍTULO

1

Introducción

A lo largo de la última década, la Web se ha convertido en la plataforma preferente para el despliegue de aplicaciones. Son varios los avances que han propiciado esta paulatina transición de un ecosistema tecnológico centrado en el ordenador personal y las aplicaciones de escritorio a una total integración en la vida cotidiana de los usuarios de los servicios ofrecidos en la Web, con independencia de dispositivo, momento o lugar. Servidores, infraestructuras, plataformas y aplicaciones se ofrecen como servicios que pueden ser consumidos sin requerir el conocimiento de qué recursos utilizan o cómo están implementados. Esta abstracción de la complejidad subyacente ha liberado a los clientes Web de su implementación y ha permitido centrar sus esfuerzos en el desarrollo de nuevos estándares que equiparen su funcionalidad a la ofrecida por aplicaciones nativas de escritorio. Sin embargo, aún quedan varias cuestiones que dilucidar en este sentido. Una de ellas, la viabilidad de la Web como plataforma de ejecución de aplicaciones con altos requisitos temporales, es a la que se dedica el contenido de este estudio.

En la sección 1.1 del presente capítulo se describen la motivación y el contexto que dan origen a esta tesis doctoral. Su hipótesis de partida y objetivos son definidos en la sección 1.2. Asimismo, en la sección 1.3 se detalla la metodología de investigación empleada. Finalmente, en la sección 1.4 se muestra la organización de este documento.

1.1 Motivación y contexto

1.1.1 La Web como plataforma

La Web entendida como una plataforma define los servicios que presta a través de una Arquitectura Orientada a la Web (*Web Oriented Architecture*, WOA, ver Hinchcliffe, 2008), una metodología de diseño y modelado que extiende la Arquitectura Orientada a Servicios (*Service Oriented Architecture*, SOA) a las aplicaciones web (Erl, 2004). WOA representa la información como recursos que serán gestionados por agentes de usuario web (navegadores) y servidores web a través de Transferencias de Estado Representacional (*Representational State Transfer*, REST). A pesar de que la definición inicial de REST describía un conjunto de principios de arquitectura de software (Fielding, 2000), actualmente se emplea para describir cualquier interfaz simple basada en un lenguaje de marcas extensible (*eXtensible Markup Language*, XML) y el protocolo de transferencia de hipertexto (*Hyper-Text Transfer Protocol*, HTTP).

Simultáneamente a este desarrollo de la infraestructura de la Web, los agentes de usuario web han experimentado a su vez una gran transformación. En pocos años hemos pasado de una Web que puede ser navegada (mediante consultas hipertextuales y puntuales de información) a una Web que necesariamente ha de ser ejecutada. Los agentes de usuario web, por tanto, han dejado de ser navegadores de Internet y se han convertido en reproductores de aplicaciones web. Este cambio ha exigido que los principales creadores de agentes de usuario web (Google, Microsoft, Mozilla, Apple, Opera) proporcionen un entorno de ejecución acorde con las nuevas necesidades. Iniciativas como el *Web Hypertext Application Technology Working Group* (WHATWG) o el propio *World Wide Web Consortium* (W3C) están liderando la definición de nuevos estándares que den respuesta a este cambio de paradigma. El estándar HTML5 es el ejemplo más representativo de su trabajo, a través del cual se definen comportamientos propios de aplicaciones web tales como el almacenamiento local, el intercambio de mensajes entre documentos, la funcionalidad de «arrastrar y soltar» (*drag & drop*), la gestión del historial de navegación o el dibujado 2D en tiempo real, entre otras (Hickson, 2012a).

Además de los esfuerzos puestos en HTML5, tanto el WHATWG como el W3C están definiendo nuevas especificaciones que permitirán a los agentes de usuario

web competir de igual a igual con las aplicaciones nativas de escritorio. En este sentido, interfaces de programación de aplicaciones (*Application Programming Interfaces*, API) como Web Storage, Web Sockets o Server-sent Events facilitan el intercambio de información entre aplicaciones web a través del acceso al almacenamiento local y la comunicación asíncrona respectivamente. Otras API como *Web Workers* proporcionan la posibilidad de delegar cálculos complejos que no requieran la interacción con la interfaz de usuario a hilos de ejecución en segundo plano, superando la restricción del hilo único de ejecución propia de JavaScript.

Este contexto propicia la consideración de JavaScript como lenguaje de referencia para el desarrollo de aplicaciones web. No solamente debido a las nuevas funcionalidades disponibles desde JavaScript, sino también por la continua mejora del rendimiento que los desarrolladores de agentes de usuario web ofrecen en sus intérpretes, así como el creciente corpus de código JavaScript de calidad publicado. Las implementaciones del lado del servidor basadas en JavaScript (*Server-Side JavaScript*, SSJS) son la prueba fehaciente de que el lenguaje ha alcanzado su madurez y es capaz de ofrecer características distintivas.

No es casual, por tanto, que el estándar HTML5 fuera inicialmente conocido como Web Applications 1.0 por el comité que lo definió dentro del WHATWG, ya que representa una clara apuesta por concebir la Web como una plataforma de aplicaciones.

1.1.2 La ciencia en la Web

Desde su concepción a comienzos de los años 90, la Web pretendió aportar un entorno flexible y colaborativo en el que sus usuarios pudieran añadir nuevo contenido y enlazarlo libremente. El objetivo inicial fue el de ofrecer una plataforma de intercambio y generación de contenidos, una Web de lectura y escritura, la «*Read/Write Web*», más allá de la mera transferencia de información digital (Berners-Lee y Cailliau, 1990). Sin embargo, no ha sido hasta la llegada de los medios sociales a comienzos de los años 2000 que esa intención se ha hecho realidad, en lo que varios autores han denominado Web 2.0 (DiNucci, 1999; OReilly, 2007) o Web Social (Hoschka, 1998). La Web 2.0 supone la evolución desde una

red centrada en el intercambio de información («las superautopistas de la información») a un medio social en el que el contenido generado por los usuarios es crucial. Su éxito, con cientos de millones de usuarios compartiendo experiencias en las redes sociales más populares (v. gr., Facebook, Twitter, LinkedIn), está fuera de toda duda.

Este progresivo proceso de socialización de la Web ha generado también grandes expectativas en el ámbito científico. Muchos investigadores han comenzado a la divulgar los resultados de sus investigaciones en Facebook o Twitter con efectos muy positivos (Eysenbach, 2011). El creciente interés por las publicaciones *Open Access* está en sintonía con ese afán de compartir el conocimiento. Al mismo tiempo, las redes sociales de investigadores (Academia.edu, Mendeley, ResearchID, SciLink, ResearchGate) permiten hacer explícitas muchas relaciones dentro de la comunidad académica y fomentar la colaboración.

La Ciencia 2.0 no descarta el método científico tradicional pero propone completarlo con las posibilidades que brinda la Web 2.0 (Shneiderman, 2008). El carácter digital de la Web Social permite el registro pormenorizado de todas las interacciones en tiempo real propio del rigor experimental de un laboratorio y, al mismo tiempo, la interacción en un entorno «natural» en el que se da un amplio rango de relaciones. Los desarrolladores de las redes sociales proporcionan un gran número de API para que terceros puedan acceder de forma cómoda a esos datos e interactuar automáticamente. De la simple minería de datos cuantitativa pasamos a la minería de opiniones o sentimientos (Jansen, Zhang, Sobel, y Chowdury, 2009; Pang y Lee, 2008) mediante las que se pretende analizar cualitativamente lo que ocurre en las redes sociales (Reips y Garaizar, 2011). Si a esto le añadimos la posibilidad de conjugarlo con repositorios de datos públicos compartidos y relacionados sobre los que trabajar semánticamente (Berners-Lee, 2006), las inferencias automáticas que pueden realizarse serán más provechosas.

Además del procesamiento automatizado de grandes cantidades de datos, la Web 2.0 puede verse desde los ojos de un investigador como un enorme laboratorio de experimentación (Nosek, 2005). Con solo acceder a una pequeña fracción del abrumador número de personas que utilizan las redes sociales, los investigadores son capaces de paliar la falta de poder estadístico a la hora de analizar efectos en el laboratorio, o acceder a muestras que por sus características especiales resultarían

inaccesibles de otra forma (Mangan y Reips, 2007; Vernberg, Snyder, y Schuh, 2005). En esta misma línea, un enfoque muy interesante es el aportado por la Ciencia Ciudadana (*Citizen Science*, ver Hand y cols., 2010), que pretende aprovechar el potencial humano y tecnológico de aficionados y gente ajena a la ciencia para llevar a cabo investigaciones científicas. La idea no es nueva (v. gr., el proyecto SETI@home data de 1999), pero recientemente se ha aplicado a entornos similares a videojuegos (v. gr., FoldIt) con resultados asombrosos (Khatib y cols., 2011). Más allá del altruismo de la Ciencia Ciudadana, a través de sitios como *Amazon Mechanical Turk* los investigadores pueden contratar sujetos experimentales a bajo coste. A pesar de las reticencias iniciales que este enfoque pueda provocar (Downs, Holbrook, Sheng, y Cranor, 2010), son varios los estudios que equiparan su fiabilidad a los resultados obtenidos en el laboratorio (Buhrmester, Kwang, y Gosling, 2011; Paolacci, Chandler, y Ipeirotis, 2010; Snow, O'Connor, Jurafsky, y Ng, 2008; Sprouse, 2011).

La Web Semántica o Web de los Datos aporta a su vez nuevos escenarios para la ciencia. Si bien es cierto que la Web Semántica no ha conseguido el éxito esperado, propuestas más posibilistas como la de los «Datos Enlazados» (*Linked Data*, ver Berners-Lee, 2006) están logrando alcanzar una masa crítica de repositorios de datos públicos compartidos y relacionados sobre los que trabajar semánticamente. Actualmente son las ciencias de la vida (Biología y Medicina, principalmente) las que más provecho están sacando de estos repositorios compartidos (Pedersen, Pakhomov, Patwardhan, y Chute, 2007), pero las posibilidades son inmensas para otras ramas del saber.

En definitiva, la Web ha dejado de ser un mero canal de difusión de hallazgos científicos para pasar a ser una plataforma sobre la que poner a prueba nuevas teorías científicas o desplegar experimentos de forma masiva.

1.1.3 Precisión y exactitud en la Web

Considerando las enormes posibilidades que brinda la Web a la ciencia, parece razonable pensar que en los próximos años asistiremos a una migración gradual hacia la Internet de la experimentación científica. Sin embargo, así como muchos paradigmas experimentales son fácilmente replicables con tecnologías web, existen

otros cuyos requisitos son tan exigentes que dificultan su implementación en la Web. Tal es el caso de aquellos paradigmas que requieren la presentación exacta y precisa de contenidos audiovisuales o el registro ajustado de la interacción de usuario.

Figura 1.1: Metáfora de la diana para comprender los conceptos de exactitud, precisión y resolución.

Dado que habitualmente y de forma errónea son empleados como sinónimos, conviene definir varios conceptos relacionados con la precisión en la temporización: la exactitud (*accuracy*), la precisión (*precision*) y la resolución (*resolution*). La exactitud tiene que ver con la distancia o diferencia entre el valor real y el valor medido. La precisión tiene que ver con el mutuo acuerdo entre diferentes mediciones realizadas con el mismo mecanismo de medida. La resolución tiene que ver con la unidad de medida más pequeña que el temporizador puede representar. Así pues, utilizando el tiro con arco como símil, podríamos decir que un tirador dispara con gran exactitud si la distancia de sus flechas al centro de la diana es pequeña, a pesar de que puedan estar repartidas irregularmente. Otro tirador podría ser menos exacto (mayor distancia al centro de la diana), pero más preciso si todos sus disparos estuvieran separados entre sí por una distancia pequeña (alto acuerdo entre diferentes disparos). La resolución tendría que ver con el grosor de las flechas, ya que si consideramos que el centro de la diana tiene un diámetro de 10 mm, un tirador que dispare flechas de 20 mm de grosor no podrá dar en el blanco sin salirse del centro de la diana (ver figura 1.1). Volviendo al caso de los mecanismos de temporización, pueden darse todas las combinaciones, existiendo aquellos que son muy exactos y

precisos pero su baja resolución los descarta para mediciones de intervalos cortos; aquellos que tienen una resolución adecuada y buena precisión, pero debido a que su valor se actualiza poco frecuentemente, su exactitud es muy baja; o aquellos que teniendo la resolución adecuada y alta exactitud, no gozan de buena precisión debido a los costes de invocación asociados a su uso.

1.1.4 Dominio de aplicación

Dentro de los paradigmas experimentales propios de la psicología o las neurociencias es habitual encontrar aplicaciones con elevados requisitos en lo referente a la presentación exacta y precisa de contenidos audiovisuales o el registro ajustado de la interacción de usuario. Tal es el caso de las aplicaciones taquistoscópicas (Mapou, 1982; Mckinney, MacCormac, y Welsh-Bohmer, 1999) o del reloj de Wundt, posteriormente conocido como el reloj de Libet (Libet, Gleason, Wright, y Pearl, 1983; Moore y Obhi, 2012), que exigen presentaciones y animaciones de contenidos visuales con periodos de tiempo muy específicos.

En lo que respecta a su desarrollo, la llegada de la informática al laboratorio ha provocado la sustitución del instrumental experimental por software especializado (Stahl, 2006). Así, este tipo de procedimientos experimentales especializados han seguido la evolución del resto aplicaciones de propósito general a lo largo de los sucesivos cambios de paradigma tecnológico que se han ido produciendo, desde las soluciones basadas en microcomputadores (Mapou, 1982) hasta los sistemas operativos multitarea más comunes (Microsoft Windows, ver K. Forster y Forster, 2003; Apple Mac OS, ver MacInnes y Taylor, 2001; o GNU/Linux, ver Stewart, 2006), pasando por los sistemas operativos monotarea (Segalowitz, 1987). Sin embargo, las limitaciones de la Web para proporcionar un entorno de ejecución de cliente apropiado, entre las que destacan la falta de precisión, exactitud y resolución de los mecanismos de temporización disponibles, han dificultado hasta la fecha su despliegue en el marco de este nuevo paradigma computacional.

Nos encontramos, por tanto, ante un cruce de caminos interesante. Por un lado, los últimos avances en infraestructura y estándares han convertido a la Web en la plataforma de ejecución por excelencia. La experimentación científica no ha sido ajena a estos avances y está aprovechando las posibilidades que brinda la Web en

este sentido. Por otro lado, las limitaciones previas de las tecnologías web clásicas en cuanto a exactitud, precisión y resolución han dificultado el despliegue en la Web de aplicaciones con elevados requisitos temporales. La reciente eclosión de las tecnologías web estándar y su continua y decidida optimización por parte de los proveedores de agentes de usuario web animan a pensar que la Web ha alcanzado un grado de madurez suficiente como para que este tipo de aplicaciones pueda desplegarse en Internet con garantías.

1.2 Hipótesis y objetivos

Teniendo en cuenta lo comentado en la sección anterior, la hipótesis de este trabajo de investigación se enuncia de la siguiente manera:

Hipótesis. *«La Web es una plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario».*

Con el objetivo de poder validar esta hipótesis, se hace necesario dar respuesta a las siguientes cuestiones de investigación:

Cuestión 1. *¿Cuál es el grado de exactitud y precisión en la presentación de contenidos audiovisuales empleando software especializado?*

Cuestión 2. *¿Cuál es el grado de precisión y exactitud en la presentación de contenidos audiovisuales empleando tecnologías web propietarias sobre agentes de usuario modernos?*

Cuestión 3. *¿Cuál es el grado de precisión y exactitud en la presentación de contenidos audiovisuales empleando tecnologías web estándar sobre agentes de usuario modernos?*

Cuestión 4. *¿En qué medida estos tres conjuntos de tecnologías presentan diferencias en cuanto a la exactitud y precisión en la presentación de contenidos audiovisuales?*

Cuestión 5. *¿Cuál es el grado de precisión y exactitud que puede lograrse en el registro de la interacción mediante tecnologías web estándar?*

Cuestión 6. *A tenor de los resultados obtenidos anteriormente, ¿existen aplicaciones con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario que puedan implementarse empleando tecnologías web estándar?*

Cuestión 7. *Si la cuestión anterior resulta ser cierta, ¿es una aplicación de este tipo implementada mediante tecnologías web estándar capaz de obtener resultados equivalentes a una aplicación offline implementada a través de los mecanismos de temporización óptimos disponibles en el sistema?*

Cuestión 8. *Finalmente, en el caso de que exista evidencia experimental que corrobore la cuestión anterior, ¿sería posible definir unas recomendaciones de desarrollo para la implementación de una aplicación con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario empleando tecnologías web estándar?*

Para responder estas cuestiones, y por tanto poder validar la hipótesis enunciada, es necesaria la consecución de los siguientes objetivos operativos:

Objetivo Operativo 1. *Analizar el grado de exactitud y precisión real en la presentación de contenidos audiovisuales del software especializado más empleado en experimentación (E-Prime, DmDX y PsychoPy) mediante el equipamiento de precisión adecuado.*

Objetivo Operativo 2. *Analizar el grado de exactitud y precisión real en la presentación de contenidos audiovisuales de las tecnologías web propietarias más comúnmente utilizadas (GIF89a, Java, Adobe Flash y Microsoft Silverlight).*

Objetivo Operativo 3. *Analizar el grado de exactitud y precisión real en la presentación de contenidos audiovisuales de las tecnologías web estándar (CSS, SMIL, SVG, Canvas, WebGL, HTML5, Web Audio, Audio Data).*

Objetivo Operativo 4. *Seleccionar las tecnologías web que optimicen la exactitud y la precisión en la presentación de contenidos audiovisuales, tratando de minimizar el consumo de recursos y requerir el menor número posible de complementos.*

Objetivo Operativo 5. *Analizar el grado de exactitud y precisión real en el registro de la interacción del usuario de las tecnologías web estándar.*

Objetivo Operativo 6. *Definir el grado de exactitud y precisión alcanzable a través de tecnologías web estándar en la presentación de contenidos audiovisuales y el registro de la interacción del usuario.*

Objetivo Operativo 7. *Desarrollar una aplicación con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario empleando los mecanismos de temporización óptimos disponibles en el sistema.*

Objetivo Operativo 8. *Desarrollar una aplicación equivalente a la descrita en el objetivo operativo 7 empleando el conjunto de tecnologías web estándar definido en el objetivo operativo 6.*

Objetivo Operativo 9. *Validar las aplicaciones desarrolladas en los objetivos operativos 7 y 8, tanto mediante el uso de sistemas automatizados de medición como a través de la experimentación psicológica gracias a la colaboración de sujetos experimentales.*

Objetivo Operativo 10. *Definir unas recomendaciones de desarrollo para la implementación de aplicaciones con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario empleando tecnologías web estándar.*

Objetivo Operativo 11. *Desarrollar las herramientas auxiliares necesarias para que las recomendaciones definidas en el objetivo operativo 10 puedan ser generalizadas a un conjunto de escenarios y dominios más amplio.*

Estos objetivos operativos describen las principales actividades que conducen a la validación de la hipótesis de esta investigación. Hemos empleado los aparatos de medición más precisos que estén a nuestro alcance para tratar de maximizar el rigor y la validez de la evidencia empírica recogida. Gracias a un estudio pormenorizado de las opciones disponibles en cada tecnología analizada podemos extraer las conclusiones necesarias para hallar una respuesta a las cuestiones de partida. En la siguiente sección se detalla la metodología de investigación que hemos seguido para alcanzar los objetivos descritos.

1.3 Metodología de investigación

La metodología de investigación empleada para dar respuesta a las cuestiones que permitan validar la hipótesis planteada puede describirse a través de los procedimientos descritos a continuación (ver figura 1.2).

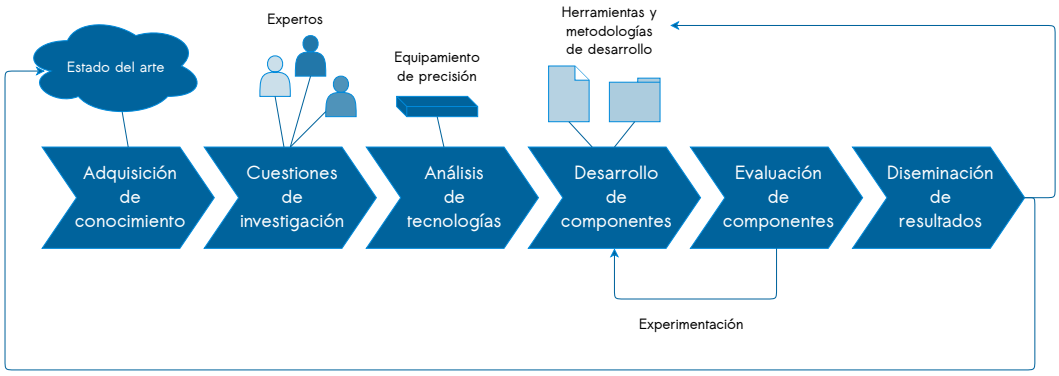

Figura 1.2: Metodología de investigación.

1. **Adquisición de conocimiento**
   El estudio inicial de los antecedentes en lo referente a la precisión y exactitud en la presentación de contenido audiovisual y el registro de la interacción de usuario mediante tecnologías offline y tecnologías web a través de la revisión exhaustiva de la literatura del área. Resulta fundamental para determinar cuáles son los últimos avances llevados a cabo y qué retos es preciso afrontar.

2. **Formulación de cuestiones de investigación**
   Gracias a la revisión bibliográfica realizada en el paso anterior es posible formular las cuestiones de investigación pertinentes que deberán ser respondidas para validar la hipótesis planteada.

3. **Análisis de las diferentes tecnologías**
   El análisis de las diferentes tecnologías consideradas a través de procedimientos rigurosos permite recabar la evidencia empírica suficiente para extraer conclusiones acerca de las mismas.

4. **Desarrollo de nuevos componentes**
   A partir del análisis de las diferentes tecnologías consideradas puede acometerse el desarrollo de nuevos componentes que pongan a prueba la validez de la hipótesis planteada.

5. **Evaluación de los diferentes componentes**
   Una vez desarrollados los nuevos componentes, su validez debe ser evaluada de forma rigurosa a través de la metodología experimental.

Posteriormente a la finalización de esta investigación, los resultados serán divulgados en congresos internacionales y revistas científicas especializadas en el área, con el objetivo de validarlos y contribuir al corpus de estudios existentes.

1.4 Estructura del documento

A continuación se detalla la estructura del resto de este documento. El capítulo 2 describe los estudios previos en torno a la precisión y exactitud de los mecanismos de temporización involucrados en la presentación de contenido audiovisual y el registro de la interacción del usuario a través de la revisión de la literatura existente. El capítulo 3 recoge la información referente a los procedimientos, resultados y conclusiones de cada uno de las tecnologías estudiadas (software especializado, tecnologías web propietarias y tecnologías web estándar). El capítulo 4 resume las distintas aproximaciones previas a la generación de la interacción de usuario y detalla el funcionamiento de las herramientas desarrolladas para la generación automatizada de la interacción de usuario, así como su registro mediante tecnologías offline y online. El capítulo 5 explica el paradigma experimental del reloj de Libet, describe la implementación de la versión offline de la aplicación desarrollada (Labclock) y de la versión online (Labclock Web) basada en tecnologías web estándar, y presenta los resultados de los seis experimentos en los que se pone a prueba validez de ambas aplicaciones. El capítulo 6 recoge un conjunto de recomendaciones de desarrollo una para el uso de la Web como plataforma de ejecución para el tipo de aplicaciones contempladas. Por último, el capítulo 7 proporciona un resumen de las principales aportaciones de esta investigación, sus dominios de aplicación, sus limitaciones y los trabajos futuros que a partir de ella podrían realizarse.

"The heavy work is getting the problem statement right"

Daniel E. Geer

CAPÍTULO

2

Antecedentes

ANTES de repasar los estudios previos en relación a la precisión y exactitud de las tecnologías disponibles desde el entorno de ejecución de un agente de usuario web, conviene recordar que los diferentes elementos o sistemas involucrados conforman una entidad propia. En ocasiones esta cuestión es pasada por alto. Sin embargo, de poco sirve una mejora parcial en uno de estos elementos si tiene un efecto nulo o incluso negativo en el resultado final del sistema en su conjunto. Este enfoque está presente de manera continua a lo largo de este estudio, en el que las mediciones se realizan mediante sistemas externos que puedan evaluar sus entradas y salidas reales, sin verse afectados por ellas.

En el marco de una situación de interacción entre un usuario y una aplicación web pueden distinguirse las entidades mostradas en la figura 2.1 y detalladas a continuación.

A **Usuario**

En el extremo más externo de la interacción se encuentra el usuario. Durante esta investigación no se han tenido en cuenta las diferencias fisiológicas o de estado que pueden darse entre usuarios (Carbonell, Elkind, y Nickerson, 1968; Lockhart, 1967; Schiff y Thayer, 1968; Shneiderman, 1980; Treisman, 1963), ni los diferentes agentes externos que pueden afectar a su capacidad de

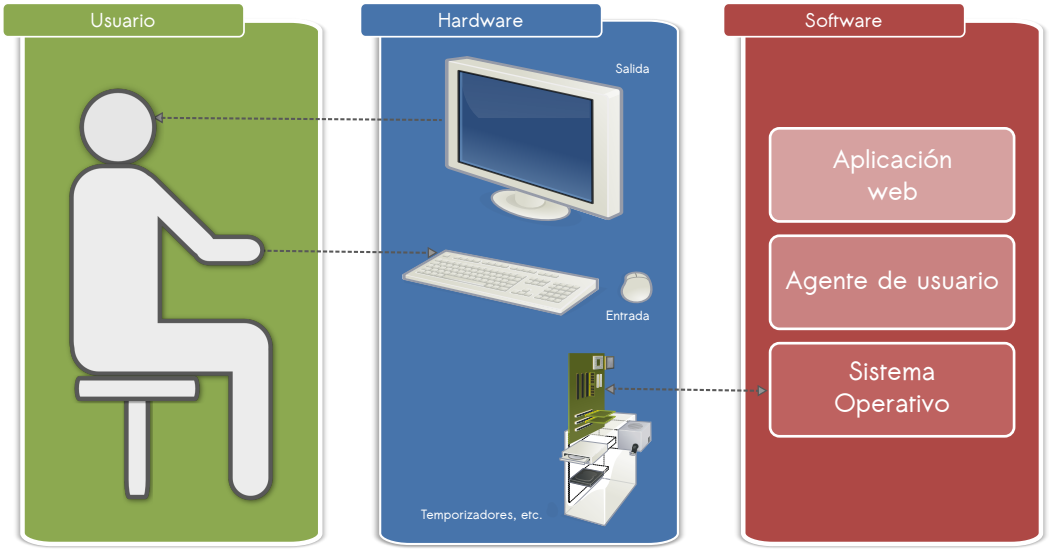

Figura 2.1: Entidades en una situación de interacción entre un usuario y una aplicación web.

atención o reacción frente a los estímulos presentados por la aplicación web (Gorn, Chattopadhyay, Sengupta, y Tripathi, 2004). En las ocasiones en las que deseamos evaluar con precisión las diferentes tecnologías, reemplazamos a los usuarios por aparatos de medición especializados (Plant, Hammond, y Turner, 2004). Por el contrario, solicitamos la colaboración de participantes voluntarios en las ocasiones en las que se precisa la validación externa de la tecnología a través de paradigmas experimentales con personas.

B Hardware

1 Periféricos de entrada

En los orígenes de la Web, durante la década de los 90, resultaba razonable asumir que los periféricos de entrada para una aplicación web se limitaban al teclado y al ratón. A pesar de que esto sigue siendo cierto en muchos casos, conviene no pasar por alto el significativo aumento del uso de dispositivos móviles para acceder a la Web. Cada una de las características implicadas en el uso de un dispositivo de entrada (conexión cableada o inalámbrica, tipos de bus y conectores, sensibilidad, tiempo de respuesta, etc.) puede afectar notablemente a la precisión y exactitud

del registro de la interacción de usuario (Plant y Turner, 2009), por lo que no debería asumirse lo contrario.

2 **Periféricos de salida**

A pesar de las diferencias subyacentes a las tecnologías empleadas para mostrar el contenido visual, todas ellas se encuentran con dificultades similares a la hora de mostrar animaciones continuas debido a sus limitaciones físicas (Elze, Tanner, Lochmann, y Becker, 2007). Por lo tanto, resulta conveniente conocer estas limitaciones en detalle para no dedicar recursos y esfuerzos a mejorar otros elementos del sistema más allá del límite impuesto por ellas, ya que no tendrán ningún efecto en el resultado final.

3 **Temporizadores hardware**

Desde el chip Intel 8253 de los IBM PC originales hasta los modernos temporizadores HPET (*High Precision Event Timer*), los dispositivos empleados para ejecutar aplicaciones web han sido equipados con temporizadores hardware. La exactitud, precisión y resolución de estos temporizadores ha ido variando a lo largo del tiempo. Lo mismo sucede en cuanto a las funcionalidades que ofrecen, ya que no todos operan de la misma manera ni están influidos por los mismos factores externos. A pesar de la abstracción que supone el uso de un sistema operativo y las distintas API disponibles, es conveniente conocer qué temporizadores están disponibles en cada plataforma hardware con el propósito de tomar las decisiones adecuadas en capas superiores.

C **Software**

1 **Sistema operativo**

Gracias a los controladores (*drivers*) y las API de los sistemas operativos, las aplicaciones de usuario son capaces de ejecutarse conforme a sus requisitos establecidos. En lo referente a los dispositivos de entrada y salida que habitualmente se emplean en aplicaciones web, el grado de ajuste en la configuración (*tunning*) y la funcionalidad proporcionada

por los controladores pueden tener un impacto notable en su desempeño. De igual manera, no todos los sistemas operativos son capaces de emplear todos los mecanismos de temporización hardware disponibles (v. gr., temporizadores HPET en Microsoft Windows XP SP1), por lo que una misma aplicación web podría comportarse de manera diferente en función del sistema operativo empleado, aún cuando el equipamiento hardware sea idéntico.

2 **Agente de usuario web**
Los tiempos en los que un único agente de usuario web era empleado por más del 90 % de los usuarios han pasado a la historia (ver figura 2.2). En la actualidad, la amplia oferta de plataformas hardware y software unida a los esfuerzos dedicados por los desarrolladores de agentes de usuario en mejorar sus características han contribuido a fragmentar su cuota de uso. Este hecho, lejos de suponer un problema, está contribuyendo notablemente a la progresiva adopción de los nuevos estándares web por parte de los desarrolladores de aplicaciones web.

3 **Aplicación web**
Por último, la propia aplicación web, corriendo dentro del entorno de ejecución proporcionado por el agente de usuario. A este nivel, el uso de bibliotecas, metodologías o patrones de diseño adecuados puede suponer un incremento en la exactitud y precisión de la presentación de contenido audiovisual y el registro de la interacción del usuario.

A continuación, se presenta una revisión pormenorizada de los estudios previos en relación a cada uno de estos elementos.

2.1 Consideraciones acerca de los usuarios de la Web

De entre todos los elementos que participan en la interacción entre un usuario y una aplicación web, el factor humano resulta, debido a su complejidad y gran variabilidad, el más complicado de evaluar y controlar. En esta sección se analizan

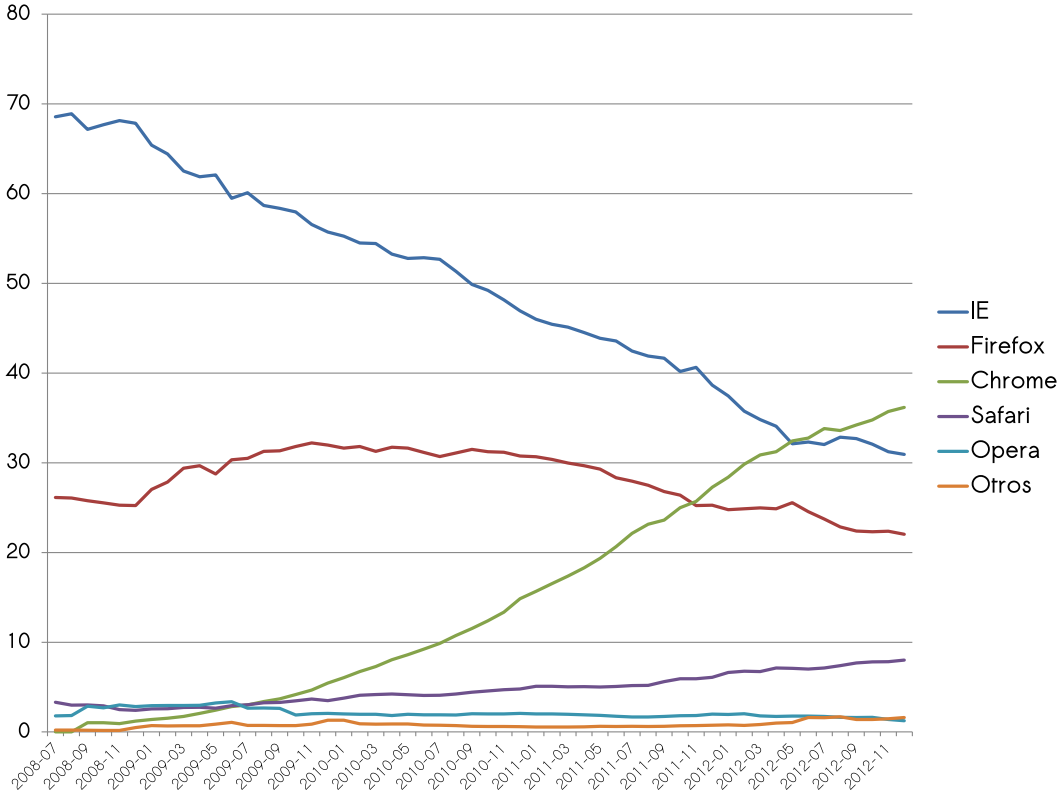

Figura 2.2: Evolución del porcentaje en cuota de mercado de los principales agentes de usuario web entre 2008 y 2012 (fuente: StatCounter).

los múltiples factores que pueden afectar a la estimación del tiempo por parte de personas.

A la hora de hablar de las estimaciones temporales realizadas por personas, conviene recordar la diferenciación entre los juicios de estimación temporal prospectivos y retrospectivos (Block y Zakay, 1997). En el caso de los juicios prospectivos de estimación temporal, también conocidos como juicios de duración experimentada, el sujeto es consciente de que se le solicitará una estimación de la duración más adelante. Por contra, en el caso de los juicios retrospectivos, también conocidos como juicios de duración recordada, el sujeto no es consciente de que se le solicitará la estimación temporal hasta que dicho periodo de tiempo ya ha sucedido. En su metaanálisis, Block y Zakay (1997) encontraron que a medida que la complejidad de la tarea incrementaba, la exactitud de las estimaciones en los juicios prospectivos decrecía, sugiriendo que los participantes disponían de menos atención disponible

para dedicarla a la monitorización temporal. Este efecto de la complejidad de la tarea, por contra, no se dio en la condición retrospectiva. Así pues, parece razonable asumir que la estimación temporal requiere la dedicación de recursos cognitivos que no podrán emplearse a otras tareas sin un perjuicio en su exactitud y precisión. Esta conclusión está alineada con lo avanzado previamente por R. Miller (1968) y otros investigadores (Carbonell y cols., 1968; Shneiderman, 1980), al plantear varias situaciones en las que el tiempo de respuesta del usuario se ve afectado, principalmente por limitaciones en la memoria a corto plazo.

Sin embargo, la estimación temporal por parte del usuario no solamente se ve perjudicada por sus limitaciones cognitivas, sino que otros factores externos pueden tener también un impacto notable. Varios estudios han encontrado evidencias de la influencia de aspectos como la temperatura ambiente (Lockhart, 1967), la valencia esperada del estímulo (Schiff y Thayer, 1968), su color (Gorn y cols., 2004) o la música que acompaña a una animación visual. Asimismo, la propia velocidad de la presentación de los contenidos audiovisuales (Shneiderman, 1984) o de la interactividad percibida en un una aplicación web (Teo, Oh, Liu, y Wei, 2003) pueden influir en el rendimiento del usuario.

Además de los factores comentados, la exactitud y precisión de la presentación de contenidos audiovisuales y el registro de la interacción del usuario se ven afectadas por un proceso retroalimentado en el que los posibles retardos de la aplicación web influyen en el estado emocional y provocan una merma en el rendimiento del usuario que, a su vez, puede generar mayores retardos y una estimación mayor de la duración temporal.

Cuál es el tiempo a partir del cuál este proceso de retroalimentación negativa se desencadena ha sido una cuestión recurrente en el último medio siglo (Galletta, Henry, McCoy, y Polak, 2004; Nah, 2004). R. Miller (1968) sugirió un retardo máximo de 2 s para que una interacción humano-máquina no pierda su naturaleza conversacional y un retardo óptimo de 0,5 s para maximizar el flujo de la conversación entre ambas entidades. Más adelante, Nielsen (1994) estableció tres límites importantes en relación a los tiempos de respuesta de un sistema y su usabilidad: a) 0,1 s como límite en el que el usuario percibe que el sistema reacciona de manera instantánea, por lo que no es preciso emplear ningún tipo de retorno más allá del propio resultado esperado; b) 1 s como el límite en el que el usuario percibe un flujo

continuo en la comunicación con el sistema; y c) 10 s como el límite en el que el sistema pierde la atención del usuario focalizada en la presente comunicación. Si se dieran retardos mayores, el usuario preferiría realizar otras tareas mientras espera a que el sistema termine, por lo que sería muy recomendable que el sistema informara al usuario de este tipo de retardos. A pesar de que otros autores han sugerido valores más conservadores para este último límite, se observa una paulatina disminución en su estimación. Ramsay, Barbesi, y Preece (1998) lo situaron en torno a los 41 s, mientras que Selvidge (1999) lo fijó alrededor de 30 s. Hoxmeier y DiCesare (2000) encontraron un impacto en la satisfacción del usuario con respecto al sistema a partir de los 12 s de retardo, y más recientemente (Galletta y cols., 2004) afirman que si bien el rendimiento y el comportamiento se ve afectado a partir de los 4 s de retardo, las actitudes con respecto al sistema no cambian hasta pasados los 8 s de retardo.

En lo referente a la Web, se dan otros problemas con relación a los usuarios. A pesar de la imposibilidad de calibrar su equipamiento al emplear una aplicación web, los usuarios se adaptan con rapidez y facilidad a las condiciones del entorno (v. gr., el brillo de un monitor), por lo que estas variables, que cambian de un usuario a otro, suelen tener un impacto pequeño en el resultado final (Krantz, 2001). Sin embargo, las discrepancias entre el laboratorio e Internet no se limitan a aspectos tecnológicos, sino que mayoritariamente se centran en factores metodológicos (Birnbaum, 2004). Los principales problemas detectados en la experimentación en Internet se pueden agrupar en torno a los siguientes factores: la participación repetida, el abandono y los sesgos en la muestra de participantes, las respuestas ofrecidas y los experimentadores. Existen varias soluciones para evitar o detectar la participación múltiple de una misma persona en un experimento en Internet (sistemas de identificación basados en direcciones IP, *cookies*, autenticación por contraseña; análisis de los registros en el servidor, etc.) ya que se trata de un problema técnico. No ocurre lo mismo con la tasa de abandono, que puede afectar de manera diferente al grupo experimental frente al grupo control y suponer un sesgo de selección de participantes. Aún en el caso de que afectara por igual a ambos grupos, podría ocasionar que se hallaran resultados opuestos a lo que realmente está ocurriendo (Birnbaum y Mellers, 1989). Por esta razón, Reips (2000, 2002) sugirió el empleo de las técnicas de «precalentamiento» y «valla alta» con la intención de

que el abandono se produzca antes de la asignación del participante a alguna de las condiciones experimentales. Una ventaja de la experimentación en Internet es que minimiza el sesgo del experimentador al no encontrarse presente, pero puede aumentar el sesgo de respuesta si no se proporcionan suficientes opciones de respuesta o las instrucciones del experimento no están claras, puesto que en ausencia del asistente de laboratorio, no habrá nadie para resolver posibles ambigüedades en el procedimiento.

Tal y como hemos podido constatar, son tantos los factores que afectan a la exactitud y precisión de las estimaciones temporales de los usuarios que no es razonable confiar en su criterio a la hora de valorar cada una de las tecnologías web analizadas en esta tesis. Por esta razón, solamente se cuenta con su participación para la validación de las aplicaciones desarrolladas, como demostración de su utilidad práctica dentro de áreas de investigación que requieren una gran exactitud y precisión en la presentación de contenido audiovisual y el registro de la interacción de usuario.

2.2 Análisis de la precisión y exactitud del hardware empleado

2.2.1 Teclados

El teclado ha sido el dispositivo preferente para la entrada de datos por parte del usuario durante décadas. Si bien es cierto que los progresos en el desarrollo de interfaces gráficas de usuario han reducido su presencia, aquellos dispositivos que por diseño deciden prescindir de él, se ven obligados a emularlo en un gran número de ocasiones. Es esta sección analizaremos las principales tecnologías relacionadas con los teclados, repasando sus características y limitaciones, así como los resultados de las mediciones a los que se han visto sometidos.

Dado que la intención de este análisis es la de identificar los posibles factores que influyen en la exactitud y precisión en la interacción entre un usuario y una aplicación web, las tecnologías previas a la invención de la Web (Berners-Lee y Cailliau, 1990) no serán estudiadas. Teniendo esto en cuenta, podemos identificar diferentes tipos de teclados en función del mecanismo empleado para distinguir

las teclas pulsadas (teclados mecánicos, de membrana, magnéticos, capacitivos, etc.), de los conectores y buses de comunicación empleados (serie, PS/2, USB, Bluetooth, infrarrojos, inalámbrico a 2.4 GHz, etc.), de su disposición (QWERTY, AZERTY, Dvorak, etc.) y otros factores que pudieran intervenir a la hora de evaluar su rendimiento.

Considerando que la influencia de la disposición del teclado en su rendimiento solamente tiene sentido en la medida en que se empleen diferentes teclas para interactuar con la aplicación, limitaremos nuestro análisis a la pulsación repetida de una sola tecla para eliminar el posible efecto de este factor. Además, la práctica totalidad de los teclados disponibles sufren en mayor o menor medida el «efecto *ghosting*» o teclas fantasma, debido a las limitaciones de su diseño. Este efecto provoca que ante determinadas pulsaciones combinadas de tres teclas, el teclado registre la pulsación de una cuarta tecla que realmente no está siendo pulsada (tecla fantasma). Algunos fabricantes emplean una técnica conocida como interferencia (*jamming*) para evitar este problema, que consiste en ignorar la pulsación de la tercera tecla para prevenir la aparición de la tecla fantasma. Otros fabricantes, por el contrario, ofrecen la funcionalidad NKRO (*N-Key Roll Over*) mediante la que se permite la pulsación simultánea de N teclas sin la aparición de teclas fantasma. Conviene señalar que un teclado con un valor NKRO de 8, por ejemplo, permitirá alguna combinación de 8 teclas sin que se dé el efecto *ghosting*, pero no de cualquier pulsación de 8 teclas (normalmente se redundan y optimizan los mecanismos implicados para las combinaciones más típicas como W-A-S-D y los cursores para teclados especializados en videojuegos). Así pues, con la intención de evitar estos inconvenientes, el resto del análisis supondrá la pulsación repetida de una misma tecla.

Las limitaciones de la tecnología subyacente a cada uno de los tipos de teclados habitualmente empleados para recoger la interacción del usuario tiene un impacto notable en su exactitud y precisión. Por esta razón, conviene recordar qué detalles intervienen en el proceso de lectura de la entrada por teclado en las diferentes tecnologías disponibles.

En el caso de los teclados con interfaz AT-PS/2, se produce de la siguiente manera: 1) el microcontrolador de teclado (Intel 8042) recibe un código de tecla válido y lo sitúa en el *buffer* de entrada; 2) se fija el *flag* IBF (*Input Buffer Full*) a 1 y se

activa la petición de interrupción IRQ 1; 3) el microcontrolador de teclado inhibe la recepción de más códigos de tecla hasta que el *buffer* de entrada no haya sido vaciado; 4) la IRQ 1 activa el controlador de teclado, apuntado por el vector de interrupciones (posición 0x09); 5) el controlador lee el código de tecla del puerto 0x60 y actualiza el área de memoria RAM que el sistema operativo ha destinado para la entrada por teclado (Chapweske, 2003b). Aunque no es habitual, la gestión de la entrada de este tipo de teclados puede hacerse mediante la consulta repetida (*polling*) en lugar de emplear interrupciones. Para ello es preciso desactivar la interrupción y consultar repetidamente el *flag* IBF. Cuando ese *flag* se encuentre a 1, se leerá el valor del código de tecla del puerto 0x60 y se volverá a fijar a 0 empleando el puerto 0x64, bit 1. El listado 2.1 muestra este proceso de consulta repetida en teclados AT-PS/2.

```
1 kbRead:
2   WaitLoop:
3         in   al, 64h   ; Read Status byte
4         and  al, 10b   ; Test IBF flag (Status<1>)
5         jz   WaitLoop  ; Wait for IBF = 1
6         in   al, 60h   ; Read input buffer
```

Listado 2.1: Consulta repetida en teclados AT-PS/2 por Chapweske (2003b).

Los teclados USB (*Universal Serial Bus*) siguen la definición de clase de dispositivo HID (*Human Interface Devices*). En esta especificación se indica que un dispositivo puede enviar o recibir una transacción en cada trama USB (1 trama / ms). Esta transacción puede estar compuesta de múltiples paquetes, pero está limitada en su tamaño. En la versión 1.1 de la especificación estos límites se sitúan en 8 bytes para dispositivos de baja velocidad y 64 bytes para dispositivos de alta velocidad (Forum, 2001). Considerando que cada pulsación de tecla precisa dos transacciones (una para indicar la pulsación de la tecla y otra para indicar que ha dejado de ser pulsada), esto limita la tasa de pulsaciones a 500 teclas / s. Por cuestiones de rendimiento y ahorro de energía, algunos sistemas operativos limitan el envío de paquetes a intervalos de 8 tramas, por lo que si se diera el caso, una pulsación de una tecla requeriría de al menos 16 ms para ser transmitida y se alcanzaría

una tasa de pulsaciones de 62,5 teclas por segundo.

A pesar de que las capacidades descritas puedan parecer más que suficientes para la tasa de pulsaciones por segundo alcanzable por una persona utilizando el teclado, no debe pasarse por alto la existencia de la funcionalidad de repetición tipomática (*typematic repeat*), capaz de generar un elevado número de pulsaciones de teclas por segundo de forma automática. Para activar la repetición tipomática es necesario mantener la tecla pulsada más allá del intervalo fijado (*typematic delay*). Una vez activada, se enviarán tantas pulsaciones por segundo como se haya configurado en la tasa de repetición tipomática (*typematic rate*) hasta que se deje de pulsar esa tecla o se pulse otra. En el caso de los teclados AT-PS/2, este comportamiento está definido en el protocolo. En los dispositivos USB, por contra, es el sistema operativo el encargado de implementarlo, empleando la tasa de envío de paquetes y el número de paquetes recibidos para determinar durante cuánto tiempo se ha pulsado una tecla.

Una vez conocidos los detalles propios de los diferentes estándares de teclados, conviene repasar los estudios en los que se han medido su exactitud y precisión de forma práctica. Graves y Bradley (1987) encontraron un retardo de 18,4 ms (desviación típica, DT: 4,3 ms) en la medición del tiempo de respuesta realizada con un teclado IBM PC estándar, mientras que este retardo alcanzó los 36,7 ms (DT: 2,9 ms) para el caso de un teclado clónico. Estos resultados fueron contrastados con los obtenidos mediante el registro de la pulsación de los botones de un joystick (modelo GravisMk IV), con el que obtuvieron un retardo de 0,55 ms (DT: 0,51 ms). Más adelante, Segalowitz y Graves (1990) detectaron una variabilidad de $\pm$ 7,5 ms en teclados IBM XT y concluyeron que puede incrementar la varianza del error y anular la detección de posibles efectos. Aprovechando la popularización de los teclados USB, Shimizu (2002) comparó los retardos ocasionados por el uso de varios teclados con los obtenidos al utilizar un joystick. Sus resultados muestran un claro perjuicio en la medición del tiempo de respuesta al emplear teclados USB (32,75 ms de intervalo de sondeo y DT: 9,453 ms en el peor de los casos) frente a teclados PS/2 (2,90 ms de intervalo de sondeo y DT: 0,836 ms en el mejor de los casos, 10,88 ms y DT: 3,140 ms en el peor). Además de los resultados presentados, Shimizu afirma que se debería prestar atención a tres factores a la hora de medir el retardo de un teclado: 1) el retardo físico, o tiempo transcurrido desde

que el usuario toca la tecla hasta que la tecla activa el mecanismo de envío; 2) el retardo de sondeo, o tiempo transcurrido desde que la señal se activa hasta que es detectada por el controlador del teclado; y 3) el retardo de codificación, o el tiempo transcurrido desde que se recibe la señal hasta que se convierte en un código de tecla, además del tiempo empleado para que esa señal llegue a su destino. Este último retardo es constante y puede ser fácilmente compensado, mientras que los dos primeros son variables y determinan la varianza del error. Siete años después, Plant y Turner (2009) evaluaron la exactitud y precisión de cuatro teclados y obtuvieron resultados que equiparaban el retardo en teclados USB (18,30 ms de retardo y DT: 1,29 ms) al de los teclados PS/2 (19,94 ms y DT: 0,83 ms en el mejor de los casos, 33,73 ms y DT: 3,08 ms en el peor). Estos ajutores encontraron diferencias significativas entre los cuatro teclados analizados. J. Forster (2007), encontró una DT de 5,154 ms que consideró inaceptable, recomendando evitar el uso del teclado como dispositivo de entrada cuando la exactitud y precisión en la medición de la interacción del usuario sean críticas. Finalmente, Neath, Earle, Hallett, y Surprenant (2011) evaluaron la precisión de dos teclados Apple USB sobre el sistema operativo Mac OS X en diferentes condiciones y encontraron que su precisión variaba entre 2,5 y 10 ms, por lo que concluyeron que pueden llevarse a cabo experimentos que detecten diferencias en torno a los 5–10 ms con equipamiento convencional de este fabricante.

A pesar de esta aparente falta de exactitud de los teclados analizados, Damian (2010) considera que esta no tiene un efecto tan negativo en los resultados analizados estadísticamente si se la compara con la inherente variabilidad del rendimiento humano entre diferentes personas. Es decir, a pesar de que sea deseable contar con equipamiento experimental de la máxima exactitud y precisión, no resulta un requisito *sine qua non* para la mayoría de los casos. Como ya se ha comentado anteriormente, el uso de un dispositivo impreciso tiene dos efectos con respecto al tiempo real transcurrido: incrementa la varianza del error e introduce un retardo. Dado que este retardo es constante, puede ser compensado. Además, este error no tiene influencia a la hora de estudiar las diferencias relativas entre condiciones. La varianza añadida al error podría provocar falsos negativos, evitando que el investigador replicara el efecto buscado. Más aún, esta falta de exactitud y precisión podría paliarse con el debido poder estadístico –aumentando el número de

observaciones–, tal y como han sugerido otros autores anteriormente (Ulrich y Giray, 1989).

2.2.2 Ratones

En lo que respecta al ratón como dispositivo de entrada, seguimos el mismo criterio que en la sección anterior y solamente estudiamos las tecnologías empleadas desde el nacimiento de la Web. Así pues, analizamos la precisión y exactitud de ratones con conexiones serie, PS/2 y USB, así como las posibles diferencias entre ratones mecánicos y ópticos.

La idoneidad de los ratones serie como dispositivos de respuesta en paradigmas de tiempos de reacción fue estudiada por varios investigadores (Crosbie, 1990; Segalowitz y Graves, 1990), que encontraron un retardo sistemático en torno a los 31–33 ms, por lo que sugirieron su compensación por parte de los experimentadores. Esta estimación fue confirmada posteriormente por J. Beringer (1992).

Un ratón serie transfiere bloques de información de 3 bytes de tamaño a 1200 baudios, con un tamaño de palabra de 7 bits y un bit de inicio y otro de parada, conformando un total de 27 bits (9×3 bits) que a 1200 baudios deberían precisar 22,499 ms para su transmisión. Teniendo en cuenta que el microcontrolador del ratón incluye un umbral de 7 ms para enviar la pulsación de un botón, la suma de estos dos tiempos ($22{,}499 + 7$ ms) se aproxima al rango de los 31–33 ms citado. Dado que este retardo está ocasionado por un cuello de botella en la tasa de trasferencia entre el ratón y el ordenador, el movimiento del ratón contribuye notablemente a este retardo, puesto que compite por el ancho de banda disponible con la pulsación del botón del ratón. Por este motivo, estos investigadores aconsejan eliminar la bola del ratón en ratones mecánicos si se van a emplear sus botones como dispositivo de respuesta.

El protocolo PS/2 para dispositivos de ratón es muy similar al empleado en los teclados. Existen cuatro modos de funcionamiento: 1) *reset* (inicialización y autodiagnóstico); 2) *stream* (envío de datos, modo de funcionamiento por defecto); 3) *remote* (el *host* hace un sondeo continuo para recibir datos del ratón); y 4) *wrap* (diagnóstico en el que el ratón reenvía todo comando enviado por el equipo al que está conectado). En el modo *stream* el ratón envía datos si detecta un cambio en su

posición o el estado de los botones. La tasa máxima de envío de datos es de 200 muestras / s, con un valor por defecto de 100 muestras / s (Chapweske, 2003a).

Los ratones USB son también dispositivos que siguen la definición de clase de dispositivo HID, por lo que podrán enviar o recibir una transacción en cada trama USB (1 ms, 1000 Hz), aunque resulta habitual que el estado del ratón se sondee cada 8 ms (125 Hz). En función del sistema operativo empleado, este valor podrá ajustarse, aunque en ningún caso podrá sobrepasar el límite impuesto por la tasa de envío de tramas USB.

Tras analizar 8 ratones diferentes, Plant, Hammond, y Whitehouse (2003) encontraron una gran variabilidad entre ellos. El retardo típico se situó en torno a los 6,55–61,60 ms, con grandes diferencias en cuanto a la tecnología empleada (32,20–38 ms para ratones serie, 6,55–61,60 ms para ratones PS/2 y 13,60 ms para el ratón USB analizado). En cuanto al tiempo empleado en la transmisión de los datos, las diferencias entre tecnologías se mantuvieron (24–33 ms para ratones serie, 3,5–6,16 ms para ratones PS/2 y 0,10 ms para el ratón USB). Otro hallazgo importante de estos autores tuvo que ver con los adaptadores PS/2 a USB incluidos en algunos dispositivos. El comportamiento de un mismo ratón empleando un puerto PS/2 y un puerto USB a través del adaptador varió notablemente. Estas diferencias son debidas a que el firmware que controla cada uno de los protocolos de comunicación con el *host* es diferente. En un estudio posterior, Plant y Turner (2009) analizaron otros 11 ratones de diferentes tecnologías y encontraron diferencias significativas entre todos ellos. El ratón con un retardo menor (Kensington Mouse) añadió 9,52 ms (DT: 2,63 ms) al tiempo medido, mientras que el retardo fue de 48,96 ms (DT: 3,19 ms) en el peor de entre los analizados (Chic Wireless Optical Mouse PS/2). Es interesante resaltar que el ratón que menor varianza registró en sus mediciones empleó el puerto USB (Microsoft Wheel Mouse Optical USB, 17,80 ms de retardo y DT: 0,70 ms).

Otro aspecto importante que conviene recordar es que la medición y calibración de los dispositivos de entrada empleados debe hacerse individualmente, ya que es habitual que los fabricantes modifiquen la circuitería de los dispositivos sin variar el nombre de producto ni su aspecto externo, por lo que podrían darse diferencias significativas entre ratones aparentemente idénticos y alterar gravemente los resultados de experimentos que no hayan contrabalanceado este factor.

2.2.3 Pantallas táctiles

Gracias a la popularización de los teléfonos móviles inteligentes (*smartphones*) y, sobre todo, a la proliferación de una nueva generación de tabletas a partir de la comercialización del Apple iPad (Samsung Galaxy Tab, RIM Playbook, Motorola Xoom, HP Slate, Microsoft Surface, Google Nexus 7, entre muchas otras), las pantallas táctiles se han convertido en pocos años en un dispositivo de entrada muy presente en la informática de consumo. Sin embargo, no se trata de un concepto novedoso, puesto que los primeros diseños datan de mediados de la década de los 60 del siglo XX (Johnson, 1965, 1967; Orr y Hopkin, 1968).

A pesar de su aparente similitud, las diferentes tecnologías empleadas de pantallas táctiles se basan en fenómenos físicos muy distintos. Así, podemos encontrar: a) pantallas táctiles infrarrojas, basadas en sensores infrarrojos situados a los bordes de la pantalla formando una matriz capaz de detectar objetos (típicamente el dedo del usuario) dentro de ella; b) pantallas resistivas, basadas en dos capas transparentes capaces de conducir la electricidad situadas sobre la pantalla (cuando el usuario ejerce presión sobre ellas, se ponen en contacto y la posición se estima en función de la resistencia del circuito cerrado por el contacto entre ambas capas); c) pantallas capacitivas, basadas en la creación de un campo electrostático sobre la pantalla que provoca la creación dinámica de un condensador cuando un elemento conductor (el dedo del usuario o un puntero adecuado) lo toca (la posición se estima en función del cambio en la capacitancia del campo); d) pantallas táctiles de onda acústica superficial, basadas en la emisión y recepción de ondas acústicas fuera del espectro audible por humanos (la posición se estima por los cambios en la señal emitida provocados por la presencia de un objeto dentro del área de la pantalla); e) pantallas de imagen óptica, basadas en el uso de emisores infrarrojos y sensores de imagen situados a los bordes de la pantalla capaces de detectar la sombra proyectada por un objeto dentro del campo de visión; y f) pantallas piezoeléctricas, basadas en la detección de la piezoelectricidad generada debida a las pulsaciones sobre la pantalla.

En lo relativo a su exactitud y precisión a la hora de registrar la interacción del usuario, existen diferencias en función de la tecnología empleada. Los tiempos de respuesta máximos de las pantallas resistivas se sitúan en torno a los 10 ms para las

pantallas de 4 hilos y en torno a los 15 ms para las pantallas de 5 hilos. En el caso de las pantallas acústicas este tiempo es de 10 ms, mientras que para las capacitivas está en torno a los 15 ms y llega hasta 20 ms en las pantallas infrarrojas (Bhalla y Bhalla, 2010).

Además de las limitaciones técnicas, también hay otros factores que pueden afectar al rendimiento de la interacción del usuario (D. Beringer y Peterson, 1985; D. Beringer, 1989; Hall, Cunningham, Roache, y Cox, 1988). Tanto Gould, Greene, Boies, Meluson, y Rasamny (1990) como Sears (1991) estudiaron el rendimiento de usuarios empleando teclados físicos y teclados virtuales operados mediante ratones o pantallas táctiles. Como cabe esperar, encontraron que el rendimiento de los usuarios es mejor en el caso de los teclados físicos (58,2 palabras por minuto), y es mayor para las pantallas táctiles (25,4 palabras por minuto) que para el ratón (17,1 palabras por minuto). Sears y Shneiderman (1991) propusieron una adaptación a la Ley de Fitts (1954) para el cálculo del tiempo necesario para interactuar con un elemento de la pantalla (2.1) que tuviera en cuenta la interacción en pantallas táctiles (2.2):

$$T = a + b \times \log\left(\frac{2D}{W}\right) \tag{2.1}$$

Donde D es la distancia, W la anchura de la pantalla, y a y b son parámetros definidos en función del dispositivo concreto.

$$T = a + b \times \log\left(\frac{cD}{W}\right) + d \times \log\left(\frac{e}{W}\right) \tag{2.2}$$

Donde D es la distancia, W la anchura de la pantalla, y a, b, c, d, y e son parámetros definidos en función del dispositivo concreto.

La interacción mediante pantallas táctiles ha pasado de la mera pulsación al uso de complejos gestos multitáctiles capaces de generar comportamientos sofisticados más allá de los ofrecidos por otros dispositivos de entrada (Forlines, Wigdor, Shen, y Balakrishnan, 2007).

Teniendo en cuenta estos avances en la interacción humano-ordenador y las ventajas en cuanto a ubicuidad y acceso que proporcionan los dispositivos móviles,

no es raro que algunos investigadores hayan decidido adaptar sus laboratorios y experimentos online a estas plataformas (Dufau y cols., 2011; Orduña, García-Zubia, Irurzun, López-de-Ipiña, y Rodríguez-Gil, 2011).

2.2.4 Monitores

Antes de analizar su impacto en la exactitud y precisión de la presentación de contenido visual, resulta conveniente recordar algunos conceptos relacionados con las tecnologías de visualización empleadas en los monitores de ordenadores personales.

En la actualidad, los monitores empleados habitualmente en los ordenadores de los laboratorios de experimentación se pueden clasificar en dos categorías en función de la tecnología que utilizan: monitores de tubo de rayos catódicos (*Cathode Ray Tube*, CRT) y monitores de cristal líquido (*Liquid Crystal Display*, LCD). Tradicionalmente los monitores CRT han sido considerados como más adecuados para la presentación precisa y exacta de contenido visual debido a sus relativamente altas tasas de refresco (en torno a los 85–100 Hz, aunque existen modelos que pueden alcanzar los 240 Hz) y sus menores tiempos de excitación (*rise time*) y decaimiento (*decay time*), permitiendo un mayor número de marcos por segundo (*Frames Per Second*, FPS) y cambios más abruptos de un marco a otro, respectivamente (Cowan, 1995). Sin embargo, estas consideraciones han dejado de ser ciertas, como detallaremos a continuación.

La tecnología subyacente a los monitores CRT y LCD es completamente diferente, pero hay algunos aspectos intermedios que son comunes (Castellano, 1992). En un monitor CRT, un cañón de electrones situado en la parte trasera apunta hacia una pantalla de cristal recubierta de fósforo situada en la parte delantera. Para dibujar una imagen, el cañón comienza por la esquina superior izquierda y va realizando barridos horizontales hasta llegar a la esquina inferior derecha. Una vez ahí, el cañón de electrones se apaga, vuelve a la posición de partida y comienza de nuevo (ver figura 2.3). Este proceso está controlado por la señal `V-SYNC` y el tiempo de refresco de un monitor CRT depende de cuánto de rápido se dispare esta señal. Al periodo entre dos señales `V-SYNC` se le denomina *tick* y su duración está en función del tiempo de refresco (v. gr., a 60 Hz un *tick* dura 16,667 ms, a

100 Hz un *tick* dura 10 ms). Conviene tener en cuenta que el cañón de electrones permanece menos de 1 ms sobre cada punto de la pantalla, por lo que la imagen se hace visible al ojo humano gracias a que el tiempo de permanencia o decaimiento es lo suficientemente prolongado.

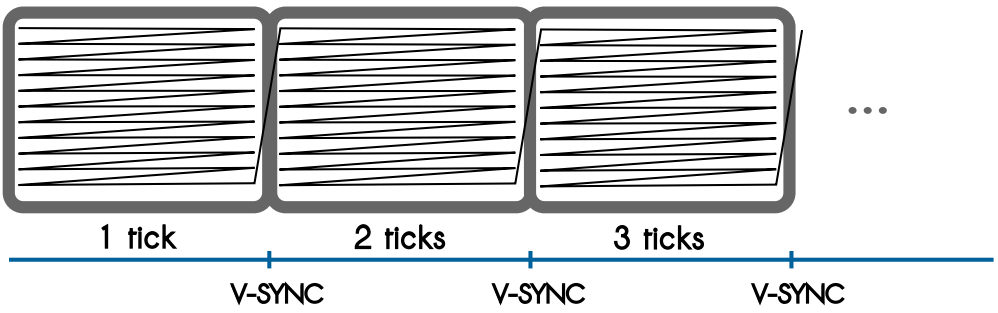

Figura 2.3: Tiempo de refresco y señal `V-SYNC`.

Un monitor LCD, por contra, emplea una matriz de cristales líquidos situada entre dos cristales polarizados. La imagen se muestra en la pantalla debido a los cambios que sufre al atravesar esa matriz de cristales líquidos una luz permanentemente encendida y situada en la parte trasera. Por lo tanto, los monitores LCD no detienen la emisión de luz en cada refresco, sino que ajustan la estructura de la matriz de cristales líquidos para reflejar los cambios en la imagen. El tiempo de respuesta del monitor se determina en función de cuánto de rápido puedan realizar estos cambios. El tiempo de respuesta no está relacionado con el tiempo de refresco de forma directa. Sin embargo, al configurar un monitor LCD en un PC es necesario fijar el tiempo de refresco como si de un monitor CRT se tratara. El motivo de que esto sea así está relacionado con la compatibilidad. Las tarjetas gráficas fueron diseñadas teniendo en mente a los monitores CRT y emplean una técnica denominada doble-*buffer*. En uno de los *buffers* de la tarjeta gráfica se encuentra la imagen que actualmente está siendo dibujada en la pantalla, mientras que en el otro está el siguiente marco que está siendo dibujado por el software. Cuando se dispara la señal `V-SYNC`, ambos *buffers* se intercambian y el monitor comienza a dibujar la nueva imagen mientras que el software comienza a pintar una nueva sobre el otro *buffer*.

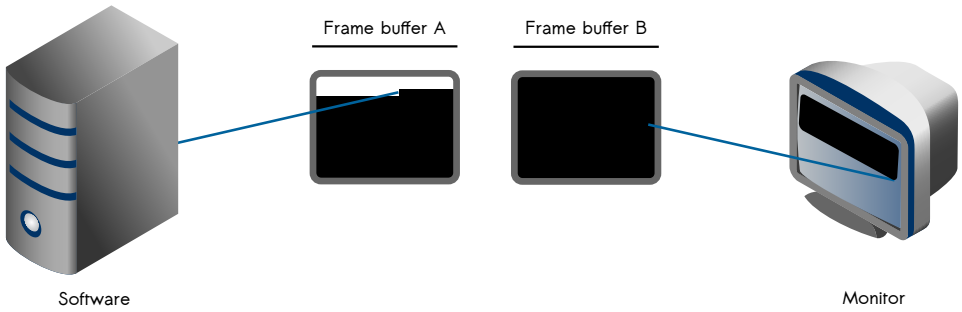

Figura 2.4: Proceso de dibujado con doble-*buffer*.

Esta es la razón por la que la configuración del tiempo de refresco influye incluso en monitores LCD. En el caso de no utilizar la señal de sincronización, se escribe directamente sobre la memoria que almacena una copia de la imagen de pantalla y es posible que la imagen aparezca cortada horizontalmente, mostrando la mitad del marco anterior y la otra mitad del marco siguiente. En el caso de emplear la señal de sincronización, no será posible mostrar cambios en la pantalla más rápidamente que lo marcado por el tiempo de refresco. Los intervalos de las animaciones se encontrarán «cuantizados» en relación a esta tasa. Es decir, los cambios se podrán producir en «cuantos» de un número entero de *ticks* de duración (v. gr., a 60 Hz podrá haber cambios cada 16,667 ms o cada 50 ms pero no cada 40 ms).

En lo relativo a la exactitud y precisión temporal, Lincoln y Lane (1980) detectaron tres fuentes potenciales de error a la hora de medir tiempos de reacción mediante el uso de monitores CRT: 1) la persistencia de la pantalla de fósforo; 2) el retardo por el proceso de refresco; y 3) el retardo por la transmisión de la imagen. Si tenemos en cuenta que el tercer factor únicamente tiene sentido al emplear tecnologías tan rudimentarias como las analizadas por estos autores, solamente las conclusiones relativas a los dos primeros factores tienen vigencia actualmente. Acerca de la persistencia de la pantalla de fósforo, sugieren que debería medirse el tiempo de persistencia o decaimiento y ajustar la duración de la imagen para compensarlo. En cuanto al retardo debido al proceso de refresco, consideran que el error al mostrar una imagen con una duración arbitraria está sujeta a un error distribuido uniformemente desde 0 ms hasta la duración de un *tick*. Podría realizarse, por lo tanto, una estimación no sesgada del tiempo de reacción medido compensando ese

valor con la duración de medio *tick* (v. gr., 8,333 ms a 60 Hz). Por lo tanto, esta fuente de error tiene como efecto una reducción del poder estadístico de la medida. La varianza de una distribución uniforme continua para $a \leqslant x \leqslant b$ es:

$$\frac{(b-a)^2}{12} \tag{2.3}$$

Por lo que la varianza del error producido por el retardo en el proceso de refresco será de:

$$\frac{(8{,}333-(-8{,}333))^2}{12} \approx 23{,}15 \text{ ms} \tag{2.4}$$

Este es el incremento en la varianza del error si solamente se obtiene a una muestra de la variable dependiente. Si se tratara de k muestras, el error sería $23{,}15/k$ ms, por lo que disminuiría en función del número de muestras tomadas y podría reducirse así hasta los parámetros deseados. Más adelante, Sutter y Tran (1992) definieron diferentes factores de corrección para esta fuente de error.

A pesar de que Gofen y Mackeben (1997) aportaron evidencias de la idoneidad de los ordenadores personales para la presentación exacta y precisa de contenido visual sobre diferentes versiones de Microsoft Windows empleando monitores CRT, Bach, Meigen, y Strasburger (1997) describieron las limitaciones de resolución temporal de este tipo de monitores, incidiendo en cuestiones ya comentadas en esta sección como la tasa de refresco o el tiempo de decaimiento de la pantalla de fósforo, e introduciendo otras nuevas como las interacciones espacio-temporales en animaciones periódicas de elementos visuales. Según estos autores, muchas pantallas de fósforo tienen funciones de decaimiento complejas, siendo este rápido inicialmente y más paulatino posteriormente (ver también Cowan y Rowell, 1986). Mollon y Baker (1995) coinciden al afirmar que este efecto puede ocasionar problemas en estudios científicos. En cuanto a las interacciones entre los límites espaciales y temporales en animaciones periódicas, es preciso establecer un límite máximo para el producto entre la velocidad del objeto en la animación periódica (medida en grados por segundo) y la frecuencia de repetición (medida en ciclos por grado) que esté en función del tiempo de refresco del monitor:

$$v_{max} \times f_{max} = \frac{f_{frame}}{4} \tag{2.5}$$

La razón por la que la tasa de refresco en Hz del monitor tiene un 4 como denominador obedece a las limitaciones espaciales y temporales (para la identificación de la dirección de movimiento es preciso que el desplazamiento máximo dure menos de $1/2$ del periodo).

Además de estas recomendaciones, Krantz (2001) sugiere emplear imágenes con un alto contraste con el fondo (por encima de 3:1), así como el uso de variaciones simples de color y luminancia y evitar imágenes que sean excesivamente pequeñas o detalladas, con el objetivo de reducir distintos efectos adversos, tales como interacciones espaciales entre píxeles en monitores CRT, altos tiempos de respuesta en monitores LCD, falta de constancia en la luminancia (mayor en el centro en monitores CRT y lo opuesto en monitores LCD) o la necesidad de un tiempo de calentamiento (en torno a 30 min) para obtener medidas estables en monitores CRT. Estos consejos son especialmente relevantes en aplicaciones Web donde la calibración fina del hardware y el software no es posible (Krantz, Ballard, y Scher, 1997).

Plant y Turner (2009) analizaron los retardos de 2 monitores CRT (LG Studioworks 775E y Sony Trinitron 110ES), 6 LCD (Acer AL718, Viglen EZX15F, AOC LM720A, MiTAC MT-15LXA, Dell E156FPf, LG Flatron L1715SM) y 3 proyectores multimedia (Dell 2300MP, Hitachi CPL850E, Hitachi CP-X328) a la hora de presentar contenido visual, y encontraron que el retardo se situaba en torno a los 20 ms en los dos monitores CRT (la duración de un *tick* a 75 Hz más la mitad de un *tick* para alcanzar el fotosensor situado en mitad de la pantalla: $13{,}333 + 6{,}666$ ms). El retardo en los monitores LCD varió en el rango de los 25 (LG Flatron L1715SM) a los 54 ms (MiTAC MT-15LXA) y fue coherente con la información proporcionada por el fabricante (i. e., los monitores con menores tiempos de respuesta en sus especificaciones sufrieron retardos menores en las pruebas). Los retardos en los proyectores multimedia se situaron entre los 29,79 ms del más rápido (Hitachi CP-X328) y los 35,30 ms (Hitachi CPL850E) del más lento.

Hasta 2011 existía cierto consenso acerca de la superioridad de los monitores CRT sobre los monitores LCD en cuanto a la temporización precisa de contenido visual. Las estimaciones previas situaban el tiempo necesario para que un monitor LCD adquiriera la máxima luminancia en torno a los 150 ms (Liang y Badano, 2007), haciendo que fuera inadecuado para animaciones con rápidos cambios de

este parámetro. Sin embargo P. Wang y Nikolić (2011) trataron de encontrar un monitor LCD entre los disponibles en el mercado que tuviera capacidades de temporización y estabilidad similares a las ofrecidas por monitores CRT, y lo lograron. El tiempo de inicio del contenido visual empleando este monitor LCD (Samsung 2233RZ 120 Hz LCD 22") fue estable a lo largo de todos las repeticiones, con un error inferior a 0,01 ms. La luminancia también fue constante a lo largo de las repeticiones (error inferior al 0,04 %) y las pruebas con diferentes duraciones (error inferior al 3 %), por lo que se sitúa a la par –o incluso por delante– de algunos monitores CRT y puede considerarse adecuado para presentaciones exactas y precisas de contenidos visuales. Lagroix, Yanko, y Spalek (2012) obtuvieron a su vez resultados muy favorables para los monitores LCD frente a los monitores CRT. A través de un conjunto de pruebas, hallaron que animaciones rápidas de imágenes blancas sobre fondo negro ocasionan una persistencia discernible en monitores CRT mientras que los monitores LCD no se ven afectados. Las animaciones de imágenes negras sobre fondo blanco no provocan persistencia en monitores CRT ni LCD. Asimismo, confirmaron la mejoría de la tecnología LCD en cuanto a la precisión temporal, encontrando tiempos de excitación mucho más cortos que los previamente estimados (de 1 a 6 ms, frente a los 20–150 ms de estimaciones previas) y comparables a los propios de monitores CRT (en torno a 1 ms). Su conclusión, sorprendente para el área, es que los monitores LCD son preferibles a los monitores CRT cuando la persistencia visual es importante, salvo en los casos de animaciones de negro sobre blanco en los que ambas tecnologías son adecuadas.

Sin embargo, la realidad es que ambas tecnologías sufren diversas limitaciones que las hacen inapropiadas para la presentación de contenido de muy corta duración (Elze y Tanner, 2009; Elze, 2010). Por un lado, conviene recordar que la luminancia de cada punto de una imagen mostrada por un monitor CRT no es constante, sino que depende de dos factores: la tasa de refresco y el tiempo de decaimiento. Este último factor depende del tipo de fósforo empleado y suele situarse en torno a los 1,5 y 6 ms para el tipo de fósforo más utilizado (P-22). Así, los investigadores que emplean monitores CRT en sus experimentos no deberán asumir una señal cuadrada con una duración del número de *ticks* configurados, sino una sucesión de pulsos discretos separados en el tiempo por la duración de un *tick* alargados momentáneamente lo que dure el tiempo de decaimiento. Por otro lado, como ya hemos avanza-

do anteriormente, el tiempo de respuesta de una pantalla LCD indica cuánto tiempo es necesario para pasar de un nivel de luminancia a otro. Sin embargo, los tiempos proporcionados por los fabricantes no corresponden al funcionamiento habitual, ya que aprovechan el uso de mecanismos de compensación (*overdrive*) para mejorar únicamente determinadas transiciones. Además, estos mecanismos pueden afectar muy negativamente a la presentación síncrona de contenido visual compuesto de varios niveles de luminancia (Elze y cols., 2007).

2.2.5 **Audio**

El audio ha sido uno de los elementos a los que menos atención se ha prestado desde los inicios de la Web. Tanto es así que hoy en día todavía no se dispone de un estándar web unificado y abierto para la generación de audio. En lo que respecta a la presentación exacta y precisa de contenido auditivo, el objetivo de un sistema diseñado para ser usado por personas debería ser no superar los 5–10 ms de latencia, umbral en torno al que se sitúa el consenso actual sobre la idoneidad de un equipo de reproducción y grabación de audio (Y. Wang, Stables, y Reiss, 2012).

Independientemente de la adecuación del hardware de audio a los requisitos temporales exigidos, los sistemas operativos constituyen habitualmente un factor que limita estas capacidades, sobre todo en el caso de que no se empleen sistemas operativos *de tiempo real*. Llegados a este punto, conviene recalcar la distinción entre los sistemas *en tiempo real* y los sistemas *de tiempo real*.

Un sistema *en tiempo real* pretende trabajar con información lo más actualizada posible, proporcionando una sensación de novedad, rapidez y de continuidad temporal al usuario, tratando de lograr este objetivo de la mejor forma posible en función de las limitaciones de hardware y ancho de banda. Podríamos citar como ejemplos de sistemas *en tiempo real* los videojuegos o las simulaciones audiovisuales, donde la alta responsividad ante las interacciones del usuario confieren una elevada sensación de realidad; o las plataformas de ofimática colaborativa, donde diferentes usuarios pueden contribuir simultáneamente en la edición de un documento y tener constancia casi instantánea de los cambios incorporados por otros editores. A pesar de que es deseable que estos sistemas respondan con rapidez a los cambios, suelen estar diseñados con una política de «mejor esfuerzo» (*best-effort*),

es decir, hacen lo que está en su mano para maximizar el rendimiento global (*throughput*) en términos estadísticos y no de conformidad con requisitos temporales estrictos.

Por otro lado, los sistemas *de tiempo real* son aquellos en los que para que el funcionamiento del sistema se considere correcto no basta con que las acciones sean correctas, sino que tienen que ejecutarse dentro del intervalo de tiempo especificado. Dentro de esta categoría, en función la flexibilidad de sus restricciones temporales, se pueden distinguir: 1) sistemas con plazos estrictos (*hard real-time*), en los que el incumplimiento de un solo plazo tiene consecuencias inaceptables; 2) sistemas con plazos flexibles (*soft real-time*) en los que resulta aceptable que eventualmente se incumpla algún plazo; y 3) sistemas con plazos firmes (*firm real-time*) en los que resulta aceptable incumplir algún plazo, pero una respuesta tardía no tiene valor. A partir de estas definiciones podría entenderse que los sistemas de tiempo real han de ser necesariamente rápidos, pero no tiene por qué ser así. El sistema de guiado de un barco puede constituir un sistema *de tiempo real* en el que las decisiones sobre su rumbo se deban tomar bajo unas severas restricciones temporales o podría producirse un accidente. Sin embargo, teniendo en cuenta las bajas velocidades que se dan en el mar, se dispone de largos periodos de tiempo (del orden de minutos) para tomar esas decisiones de control. Un sistema con una política de «mejor esfuerzo» podría controlar este sistema de guiado con mucha mayor granularidad en la mayoría de los casos, pero si existiera la posibilidad de que, dada su naturaleza no determinista, en algún caso no se cumpliera el plazo definido, la solución sería inaceptable. Por contra, un sistema *de tiempo real* menos eficiente pero que siempre cumpla sus plazos proporcionaría una solución aceptable.

Dicho esto, es importante señalar que los sistemas operativos más usados para ejecutar aplicaciones web no son sistemas *de tiempo real* (Microsoft Windows XP/Vista/7/8, Mac OS X, GNU/Linux). Las buenas noticias son que, a pesar de que el cerebro humano es capaz de detectar desviaciones en torno a los 20 μs entre cada oído como clave para determinar la posición espacial del origen del sonido (Pierce, 1999), diversos estudios afirman que la sensibilidad consciente de las diferencias temporales se sitúa en umbrales mucho más altos, en torno a los 30–50 ms (Aschersleben, 2002; Rasch, 1979).

A lo largo de las últimas décadas, varios grupos de investigadores han medido la latencia en el hardware de audio de ordenadores personales. Brandt y Dannenberg (1998) estudiaron la latencia mínima obtenible empleando sistemas operativos de propósito general (SGI Irix 6.4, Microsoft Windows 95, 98 y NT 4.0). Sus resultados no fueron del todo favorables para Windows 95 y NT 4.0, pero sí en el caso de Irix 6.4, en el que se cumplieron los requisitos planteados. Lago y Kon (2004) aportaron una extensa revisión bibliográfica acerca de los límites de la percepción humana en la latencia y su variabilidad (*jitter*), así como algunos detalles técnicos que conviene tener en cuenta a la hora de diseñar un sistema de audio preciso (v. gr., la separación de los altavoces mediante cableado extenso, los algoritmos de Procesado Digital de la Señal, etc.). Thom y Nelson (2004) realizaron un amplio conjunto de pruebas, analizando 4 tarjetas de sonido diferentes (Midiman MidiSport 2x2, USB; MOTU Fastlane, USB; EgoSys Miditerminal 4140, puerto paralelo; Creative Labs SoundBlaster Live! 5.1, PCI) sobre 5 sistemas operativos distintos (Linux 2.4 en una distribución Debian GNU/Linux con ALSA 0.9.4 y núcleo 2.4.20 con parches de baja latencia; Linux 2.6 con ALSA 0.9.7 y núcleo 2.6.0; Mac OS X 10.3.2 "Panther" con CoreMIDI; Microsoft Windows 2000 SP4 con WinMME; Microsoft Windows XP) y tres ordenadores diferentes (HP Pavilion 751n, Apple Mac G4, IBM Thinkpad T23). En todos los casos, la DT de la latencia se situó por debajo de 1,3 ms y su valor absoluto varió entre los 2,3 ms del mejor de los casos (Linux 2.6 con tarjeta SoundBlaster sobre HP Pavilion) y los 25,7 ms del peor (Linux 2.4 con tarjeta Midiman sobre HP Pavilion). Plant y Turner (2009) analizaron varios sistemas de reproducción de sonido (Acer AL718, Encore P-905U, Jazz J1321, Labtec LCS1060, Philips A1-2FPP005, Viglen EZX15F, Yamaha YST-M20DSP) empleando una tarjeta de sonido Creative Labs Sound Blaster Live Value! PCI con los drivers WHQL (*Windows Hardware Quality Laboratories*) para Windows XP. Tal y como era de prever, los altavoces que peor rendimiento obtuvieron fueron aquellos que emplearon la interfaz USB para la comunicación (10,12 ms de latencia frente a 3,21 ms del mejor de los casos). Por su parte, Y. Wang y cols. (2012) lograron latencias de 1,68 ms empleando Ardour sobre Linux con los drivers ALSA y el software de conexión de audio JACK, un valor comparable a las consolas digitales de audio profesionales.

2.2.6 Temporizadores

Los temporizadores constituyen el elemento de hardware fundamental para asegurar la precisión y exactitud del sistema. Sin una referencia estable, precisa y ajustada, el resto de mecanismos de temporización podrían sufrir desplazamientos temporales difícilmente corregibles. Por esta razón, desde el primer modelo de IBM PC compatible, el ordenador personal ha contado con dispositivos hardware de temporización. Tal es el caso del temporizador programable de intervalos (*Programmable Interval Timer*, PIT), un chip Intel 8253/8254 que consiste en un oscilador, un predivisor de frecuencia y tres divisores de frecuencia independientes con un canal de salida cada uno. El oscilador tiene una frecuencia de funcionamiento de 1,193182 MHz y una precisión de $\pm$ 1,73 s por día. El primer canal de salida puede emplearse para generar una interrupción de reloj (IRQ 0) periódica, cuya frecuencia suele fijarse en 18,2065 Hz (una IRQ cada 54,9254 ms), aunque puede dispararse con frecuencias mayores; o bien para generar interrupciones aperiódicas que se dispararán tras un periodo breve de tiempo (tiene que ser inferior a $1/18$ s). Es decir, de esta forma el sistema es capaz de fijar temporizadores periódicos y aperiódicos (*one-shot timers*) a través del PIT. Sin embargo, dado que este temporizador requiere la comunicación mediante puertos de entrada/salida, el coste de invocación de un temporizador aperiódico es tan alto que en la práctica no tiene sentido utilizar el PIT con este propósito. El segundo canal se empleaba antiguamente para refrescar la memoria RAM, pero en la actualidad no se utiliza y es posible que no esté implementado. Finalmente, el tercer canal se emplea para fijar la frecuencia de los sonidos producidos por el PC-Speaker (Walling, 2012).

En 1984, el IBM-AT incorporó el primer reloj de tiempo real (*Real Time Clock*, RTC) junto al chip Intel 8254. Se trataba de un chip Motorola MC146818A que, además del temporizador de tiempo real, incluía 64 bytes de RAM no volátil para almacenar información de estado (tipo del disco duro, cantidad de memoria, etc.). La frecuencia por defecto del RTC es de 32.768 Hz y dado que el divisor de frecuencia está activado inicialmente, se dispara una interrupción a una frecuencia de 1024 Hz (una IRQ cada 0,9765625 ms), aunque puede modificarse. El RTC puede gestionar 15 tasas de interrupción entre 2 Hz y 32.768 Hz de forma independiente,

aunque en algunas plataformas no es posible superar los 8000 Hz. Un detalle importante al emplear el RTC es que es necesario desactivar el soporte de interrupciones no enmascarables (*Non-Maskable Interrupts*, NMI) ya que si se diera el caso de que se produjera una interrupción no enmascarable durante la programación del RTC, no bastaría con reiniciar el sistema, puesto que este chip no se inicializa durante el arranque, sino que dispone de una alimentación eléctrica propia (Earls, 2009). A pesar de ser un temporizador con más funcionalidades que el PIT, no ha tenido el éxito que se esperaba. Hay varias razones por las que esto ha podido suceder, pero la más influyente es que no está disponible en todos los ordenadores personales, por lo que es necesario detectarlo previamente y esto incita a los programadores a seguir utilizando el PIT (Riemersma, 1994).

Con la llegada de los sistemas de multiprocesamiento simétrico (*Symmetric Multi-Processing*, SMP), los viejos controladores programables de interrupciones (*Programmable Interrupt Controller*, PIC) como el Intel 8259 fueron reemplazados por el estándar APIC (*Advanced Programmable Interrupt Controller*). La arquitectura APIC emplea dos componentes: el APIC local (LAPIC) y el APIC de entrada/salida. Existe un LAPIC por cada procesador, y un APIC de entrada/salida por cada bus en el sistema. Dentro del LAPIC hay un temporizador, el APIC *timer*. El gran beneficio de este temporizador es que se encuentra dentro del procesador, por lo que es mucho más fácil acceder a él. La principal desventaja es que no tiene un oscilador a una frecuencia fija, sino que emplea una de las frecuencias del procesador. Esto plantea dos problemas: 1) hay que consultar cuál es la frecuencia del temporizador antes de usarlo, y 2) la frecuencia del procesador puede variar dinámicamente, lo que complica excesivamente el uso de este temporizador como reloj. En cuanto a su funcionalidad, permite tres modos de funcionamiento: 1) periódico, 2) aperiódico, y 3) límite de contador de marcas temporales (*Time Stamp Counter*, TSC). El primero de ellos permite fijar un valor inicial que el temporizador irá decrementando periódicamente hasta llegar a cero, momento en el que se producirá una interrupción y se fijará de nuevo el contador al valor inicial. La frecuencia a la que se decrementa el contador depende de la frecuencia de bus del procesador y del valor del registro de configuración de la división de frecuencia de APIC (*Divide Configuration Register*). Así, por ejemplo, si creamos un temporizador periódico con el valor 123.456 para un procesador a 2,4 GHz con una

frecuencia de bus de 800 MHz y un valor del registro de configuración de la división de frecuencia de 4, el temporizador APIC lo decrementará a una frecuencia de 200 MHz, lo que provocará una interrupción cada 617,28 μs (Sondaar, 2007). El modo aperiódico funciona de manera muy similar, salvo que el contador no se restablece una vez la interrupción ha sido generada. El uso sin precaución de este modo podría dar lugar a condiciones de carrera (*race conditions*), pero a su vez permite un control más ajustado de los temporizadores. En cuanto al modo de límite de TSC, el funcionamiento es diferente. En lugar de fijar un contador que será decrementado por el temporizador, el sistema define un tiempo límite y el temporizador genera una interrupción cuando el contador de *ticks* del procesador (no confundir con los *ticks* del refresco de pantalla) alcanza ese valor. A pesar de que funcionalmente es muy similar al modo aperiódico, hay diferencias importantes de implementación, como por ejemplo que la frecuencia de estos temporizadores no es la del bus, sino la frecuencia nominal del procesador. Aún teniendo en cuenta todas estas funcionalidades, en la práctica no se emplea demasiado debido a los problemas que ocasiona su dependencia de la frecuencia del procesador.

A finales de 1996 se publicó la primera versión del estándar ACPI (*Advanced Configuration and Power Interface*, no confundir con APIC citado previamente), para la gestión de la eficiencia energética en dispositivos hardware. Para controlar la duración de los tiempos de reposo del sistema de manera independiente a la frecuencia del procesador esta especificación incluye el temporizador de gestión de energía (*Power Management Timer*, PMP), un reloj que funciona a una frecuencia constante de 3,579545 MHz y emplea un contador de 24 o 32 bits (configurable a través de un valor dentro de la tabla de configuración de ACPI). El modo de acceso y programación de este temporizador está implementado en hardware para incrementar su exactitud y precisión (Hewlett-Packard, Intel, Microsoft, y Phoenix Technologies, 2004).

A partir de la quinta generación de procesadores Intel (Pentium) se introdujo la instrucción `RDTSC` con la que es posible acceder al valor del TSC. Este contador almacena el número de ciclos de procesador desde el arranque. A pesar de que la lectura de este valor es extremadamente rápida (del orden de decenas de ciclos de procesador) y que su elevada frecuencia permitiría en principio medir periodos de tiempo muy breves, la baja calidad del oscilador de los procesadores y la gran

influencia que tiene la variabilidad de la temperatura en su estabilidad pueden suponer un prejuicio a su exactitud y precisión. En el listado 2.2 se muestra el uso de la instrucción `RDTSC` a través de un pequeño programa en ensamblador que devuelve el valor de los bits menos significativos del TSC al llamar a la petición al sistema `SYS_EXIT` en GNU/Linux.

```
1 global _start
2 _start:
3   rdtsc
4   xchg        eax, ebx
5   inc         eax
6   int         80h
```

Listado 2.2: Lectura del TSC en NASM (*The Netwide Assembler*) para GNU/Linux.

En sistemas multiprocesador existe un TSC por cada uno de los núcleos y su valor no tiene por qué estar sincronizado, por lo que es muy recomendable consultar siempre el valor de un mismo núcleo para evitar problemas. Además, desde el Intel Pentium Pro se admite la ejecución de instrucciones fuera de orden, por lo que una llamada a `RDTSC` podría ejecutarse más tarde de lo previsto y alterar la duración del intervalo medido. Para evitar este problema puede emplearse la instrucción `CPUID`, diseñada para identificar el núcleo en el que se ejecuta un programa, que permite la serialización de las instrucciones. En el listado 2.3 puede verse un fragmento de código ensamblador destinado a medir el tiempo transcurrido mediante el TSC al hacer una división de coma flotante que emplea `CPUID` para evitar estos problemas.

```
1 cpuid
2 rdtsc
3 mov         time, eax
4 fdiv
5 cpuid
6 rdtsc
7 sub         eax, time
```

Listado 2.3: Comprobación del tiempo transcurrido al hacer una división de coma flotante (Intel, 1998).

En procesadores más avanzados (Intel Core i7, AMD Athlon 64 X2 o superiores) está disponible la instrucción `RDTSCP`, variante serializada de `RDTSC`.

Finalmente, el esfuerzo conjunto de Intel y Microsoft entre 1999 y 2004 produjo la publicación de la especificación HPET (*High Precision Event Timer*), que pretendía solventar los problemas detectados en los dispositivos de temporización previos. Esta especificación exige la existencia de un contador monotónicamente creciente con una frecuencia de al menos 10 MHz, además de un conjunto de comparadores (al menos 3 y hasta 256) de 32 y 64 bits. Estos comparadores pueden emplearse para crear temporizadores periódicos o aperiódicos. El HPET se emplea a través de un acceso de entrada/salida mapeado en memoria (situado en una dirección definida en la tabla ACPI HPET). No todos los sistemas operativos ofrecen soporte para HPET (Windows XP, Windows 2003 Server, y versiones anteriores de Windows, así como versiones anteriores a Linux 2.6 carecen de soporte) y puede darse el caso de que no se esté usando –en favor de otros temporizadores más antiguos– porque el soporte esté deshabilitado (puede comprobarse empleando la herramienta `WinTimerTester` y corroborando que la frecuencia de `QueryPerformanceFrequency` está por encima de los 10 MHz, típicamente a 14,31818 MHz).

Sin embargo, no todo son ventajas al usar el HPET. El principal inconveniente derivado de su uso tiene que ver con el funcionamiento de los comparadores. En lugar de comprobar si el valor actual del contador es igual o mayor que el valor de un comparador, el HPET solamente evalúa si son iguales, por lo que puede darse el caso de que un programador fije un valor muy próximo al actual dentro de un comparador y que, por motivos de concurrencia, este cambio en el comparador llegue más tarde que su valor indicado. En este caso, en lugar de generarse una interrupción inmediata, habrá que esperar a que el HPET dé la vuelta completa al contador hasta alcanzar de nuevo el valor fijado (proceso que puede requerir hasta 306 segundos). Esta limitación en su diseño obliga a los programadores a combinar su uso con otros temporizadores de menor precisión que sirvan como sistemas de respaldo para evitar estos problemas.

2.3 Análisis de la precisión y exactitud de los mecanismos de temporización disponibles

Una vez detallados los límites y funcionalidades relacionadas con el equipamiento hardware empleado, llega el turno de analizar la precisión y exactitud de los mecanismos de temporización disponibles en los diferentes sistemas operativos utilizados. Con el ánimo de evitar confusiones innecesarias, este análisis se dividirá en dos partes. En la primera de ellas, se repasan los temporizadores disponibles en sistemas Microsoft Windows y compatibles con UNIX, es decir, los mecanismos que permiten la invocación de procedimientos demorados. La segunda parte está dedicada al análisis de las diferentes funciones de tiempo disponibles en estos sistemas, es decir, los mecanismos destinados a la obtención de marcas temporales.

2.3.1 Temporizadores

Al igual que ocurre con su implementación en hardware, los sistemas operativos brindan un amplio abanico de posibilidades para la creación de temporizadores. En la tablas 2.1 y 2.2 pueden verse sendos resúmenes de las principales API de temporización para sistemas compatibles con Windows y compatibles con UNIX (*UNIX-like*) respectivamente.

API	**Llamadas**
Standard Win32 Timers	`SetTimer`, `KillTimer`
Multimedia Timers	`timeBeginPeriod`, `timeSetEvent`, `timeKillEvent`
Waitable Timers	`CreateWaitableTimer`, `SetWaitableTimer`, `CancelWaitableTimer`
Queue Timers	`CreateTimerQueueTimer`, `WaitOrTimerCallback`, `DeleteTimerQueueTimer`

Tabla 2.1: Resumen de las API para la creación de temporizadores en sistemas Windows.

Si un programador necesita ejecutar un procedimiento al cabo de un periodo de tiempo en Windows y no tiene como requisito una elevada exactitud temporal, es probable que se decante por los temporizadores estándar de la API Win32. Estos temporizadores tienen la ventaja de funcionar en cualquier versión de Windows desde el modo usuario, aunque también sufren algunas limitaciones notables en cuanto a su exactitud y la responsividad del sistema (el procedimiento demorado lanzado por este tipo de temporizadores se ejecutará en el hilo dedicado a la interfaz de usuario). Para crear un temporizador de este tipo es necesario emplear la función `SetTimer`, incluyendo como argumentos el identificador de la ventana (puede estar vacío), el identificador de temporizador (un valor distinto de cero), el tiempo en el que expirará el temporizador (en ms) y un puntero a una función que gestionará los mensajes generados por el temporizador (habitualmente está vacía). De esta manera, se enviará un mensaje `WM_TIMER` a la ventana definida (o a la cola de aplicación, si no se definió ninguna ventana al crear el temporizador) periódicamente (según el periodo indicado al crear el temporizador) y la ventana o la aplicación será la encargada de ejecutar el código deseado al recibir este tipo de mensajes. Para eliminar el temporizador se emplea la función `KillTimer`, que requiere el identificador de ventana y el del temporizador como argumentos. Debido al sistema de colas de mensajes empleado en este tipo de temporizadores, las latencias en un sistema ocupado pueden ser grandes, pero su uso requiere muy pocos recursos del sistema y en muchos casos su exactitud es suficiente (v. gr., un temporizador encargado de comprobar si existen actualizaciones en un cliente de una red social).

Cuando se precisa una mayor exactitud y precisión, es necesario descartar los temporizadores estándar de la API Win32 y optar por otras opciones, entre las que se encuentran: 1) los temporizadores multimedia, 2) los temporizadores con tiempo de espera, y 3) los temporizadores de colas.

A pesar de que algunos autores desaconsejan su uso en favor de los temporizadores de colas (Osterman, 2005), los temporizadores multimedia proporcionan una exactitud y precisión cercana a 1 ms y no dependen de colas, sino que la función que se lanzará tras finalizar el temporizador podrá disponer de un hilo propio para su ejecución. La granularidad de este tipo de temporizadores ha de configurarse a nivel de todo el sistema empleando la función `timeBeginPeriod`, que recibe

como argumento el periodo de comprobación de los temporizadores multimedia (en ms). El valor original se restablece a través de la función `timeEndPeriod`. Para crear un temporizador multimedia se utiliza la función `timeSetEvent`, que recibe como argumentos el tiempo en el que expirará el temporizador (en ms), la resolución del evento de temporización (en ms), un puntero a la función que se llamará al finalizar el temporizador, información sobre el usuario y el tipo de temporizador (`TIME_PERIODIC` o `TIME_ONESHOT`, para temporizadores periódicos y aperiódicos respectivamente). Para eliminar el temporizador se emplea la función `timeKillEvent`, que recibe el identificador de temporizador como argumento.

Los temporizadores con tiempo de espera están disponibles desde Windows 98 / NT 4.0 y se emplean para bloquear la ejecución de hilos que necesitan realizar una espera. Internamente son objetos de sincronización en el núcleo del sistema cuyo estado se cambia a señalizado cuando el temporizador expira. Existen dos tipos de temporizadores con tiempo de espera: de reinicio manual (permanecen en el estado señalizado hasta que no vuelva a establecerse un tiempo de espera) y de sincronización (permanecen señalizados hasta que el hilo complete la espera). Ambos tipos pueden emplearse para crear temporizadores periódicos o aperiódicos. Estos temporizadores se crean a través de la función `CreateWaitableTimer`, que recibe como argumentos un puntero al descriptor de seguridad del temporizador, el tipo de temporizador (reinicio manual o sincronización) y un nombre de temporizador (puede estar vacío). Una vez creado, podrá llamarse a la función `SetWaitableTimer` para establecer una espera (en ms) y fijar qué función será llamada una vez esta espera finalice. El temporizador puede eliminarse a través de la función `CancelWaitableTimer`, indicando qué temporizador quiere cancelarse. Un detalle importante es que el hilo que quiera emplear este tipo de temporizadores debe configurarse con el estado de `alertable` o la función de finalización nunca será invocada.

Los temporizadores de colas fueron introducidos con Windows 2000 y, al igual que los temporizadores con tiempo de espera, emplean objetos del núcleo que residen en colas de temporizadores. Las funciones configuradas para ser llamadas al expirar los temporizadores de este tipo serán ejecutadas por un hilo del conjunto de hilos de Windows. Para crear un temporizador de este tipo es necesario emplear la función `CreateTimerQueueTimer`, que recibe como argumentos la cola de

temporizadores a la que se añadirá este temporizador (si no se establece ninguna, se añadirá a la cola por defecto de Windows), un puntero a una función que será invocada al finalizar el temporizador, los argumentos que se pasarán a esta función, el periodo de tiempo que deberá transcurrir la primera vez que se inicia este temporizador (en ms), el periodo de tiempo que deberá transcurrir el resto de veces (en ms, o cero para que solamente se lance una vez) y un conjunto de valores de configuración acerca de cómo se ejecutará la función llamada. Para eliminar un temporizador de este tipo se emplea la función `DeleteTimerQueueTimer`. La principal ventaja de este tipo de temporizadores es que introducen muy poca sobrecarga al sistema y son razonablemente precisos. A pesar de que Osterman (2005) defiende su uso frente a los temporizadores multimedia, Lee (2009) ha aportado datos que contradicen estas recomendaciones y cuestionan su precisión (retardos de hasta 10 ms para temporizadores fijados a 100 ms y mayores retardos para periodos más cortos). Zhang, Lu, Zhang, y Lian (2009) han corroborado la precisión en torno 1 ms de los temporizadores multimedia, y recomiendan evitar los temporizadores estándar de la API Win32 (basados en `SetTimer`) tanto de forma directa como a través de las funciones correspondientes en Visual Basic 6, Microsoft .Net o Delphi.

API	**Llamadas**
Interval Timers	`getitimer`, `setitimer`
POSIX Timers	`timer_create`, `timer_getoverrun`, `timer_settime`, `timer_gettime`, `timer_delete`
POSIX Threads	`pthread_cond_timedwait`
High-Resolution Timers	`hrtimer_init`, `hrtimer_rebase`, `hrtimer_start`, `hrtimer_forward`, `hrtimer_cancel`, `hrtimer_try_to_cancel`, `hrtimer_get_remaining`, `hrtimer_get_res`

Tabla 2.2: Resumen de las API para la creación de temporizadores en sistemas compatibles con UNIX (*UNIX-like*).

En el caso de sistemas compatibles con UNIX (v. gr., Free/Open/Net/BSD, GNU/Linux, Mac OS X, entre muchos otros), existe un amplio conjunto de API de temporización comunes a todos ellos, a pesar de que los detalles de implementación puedan tener grandes diferencias. Este hecho está motivado por la intención de adecuarse a las especificaciones POSIX (*Portable Operating System Interface*) del IEEE (*Institute of Electrical and Electronics Engineers*) en el diseño de estos sistemas para maximizar la compatibilidad entre ellos. Así, desde la primera versión del estándar POSIX se definieron los temporizadores de intervalo, lo más sencillos, basados en el envío de señales. En la siguiente versión de POSIX (POSIX.1b) se definieron las extensiones de tiempo real, entre las que se encuentra la especificación de relojes y temporizadores. Además, a través de la programación de hilos POSIX (*pthreads*) es posible implementar temporizadores en algunas versiones de UNIX (v. gr., Mac OS X), y sistemas compatibles como GNU/Linux tienen sus propias implementaciones de temporizadores de alta resolución (*High Resolution Timers*).

Los temporizadores de intervalo generan señales que se envían periódica o aperiódicamente al proceso cuando el temporizador expira. Cada temporizador de intervalo se describe por dos parámetros: 1) la frecuencia a la que se emitirán estas señales (o el valor cero para temporizadores aperiódicos) y 2) el tiempo que resta para que el temporizador expire. Es importante reseñar que estos temporizadores garantizan que la señal no será enviada al proceso antes de que terminen, pero una vez expirado el temporizador, no es posible conocer con exactitud cuándo llegará la señal. Para definir un temporizador de intervalo se emplea la llamada al sistema `setitimer`, que recibe como primer argumento la política que seguirá el temporizador. Estos valores pueden ser: 1) `ITIMER_REAL`, el valor del temporizador se actualiza en función del tiempo real transcurrido y el proceso recibirá señales `SIGALRM`; 2) `ITIMER_VIRTUAL`, el valor del temporizador se actualiza en función del tiempo transcurrido por el proceso en modo usuario y el proceso recibirá señales `SIGVTALRM`; y 3) `ITIMER_PROF`, el valor del temporizador se actualiza en función del tiempo transcurrido por el proceso en modo núcleo y el proceso recibirá señales `SIGPROF`. El segundo argumento de `setitimer` es una estructura de tipo `itimerval` que contiene el valor inicial del temporizador (en

s y ns) y el valor que tendrá cuando se renueve (cero para temporizadores aperiódicos). El tercer argumento es opcional y se trata de un puntero a otra estructura `itimerval` que será rellenada por la llamada al sistema con los valores previos. La implementación en GNU/Linux de este tipo de temporizadores con la política `ITIMER_REAL` precisa del uso de temporizadores dinámicos, puesto que el núcleo debe enviar señales al proceso incluso cuando no esté en ejecución (Bovet y Cesati, 2005).

En relación a los temporizadores dinámicos de GNU/Linux, conviene recordar que la implementación original de los temporizadores en este sistema estaba basada en el incremento de un contador propio del núcleo (*jiffies*) en cada interrupción de reloj. Este valor marcaba el tiempo de conmutación de procesos y otras operaciones importantes. La frecuencia de interrupciones de reloj se definía por la constante `HZ`, que tenía un valor diferente para cada plataforma (v. gr., en Linux 2.4 para la plataforma i386, `HZ` valía 1000, por lo que se producía una interrupción cada 1 ms; a partir de Linux 2.6.13, `HZ` se fijó en 250 puesto que se consideró que 1000 era demasiado alto). Al definir un temporizador dinámico, se establecía su valor de *jiffies* y se añadía a la «rueda de temporizadores». Esta rueda era una estructura que permitía al núcleo añadir, eliminar y comprobar de forma muy eficiente si los temporizadores dinámicos habían expirado a través de listas enlazadas parcialmente ordenadas (Molnar, 2005).

A partir de Linux 2.6 este tipo de temporizadores se implementa de otra manera, mediante una función postergable, a través de la política `TIMER_SOFTIRQ`. Sin embargo, la especificación POSIX 1003.1-2008 considera obsoleto el uso de este tipo de temporizadores y recomienda el uso de la nueva API de temporización (ver tabla 2.3).

Estos nuevos temporizadores pueden crearse empleando diferentes relojes del sistema como referencia. En función del sistema operativo y su versión estarán disponibles unos u otros, pero el estándar define al menos dos: `CLOCK_REALTIME` y `CLOCK_MONOTONIC`. Adicionalmente, en GNU/Linux se permite emplear otros relojes de referencia como `CLOCK_MONOTONIC_RAW` (dependiente del hardware de temporización), `CLOCK_PROCESS_CPUTIME_ID` (tiempo por proceso) o `CLOCK_THREAD_CPUTIME_ID` (tiempo por hilo). En la tabla 2.4 se describen las implicaciones de la elección de cada uno de ellos.

Función	Descripción
`clock_settime()`	Fijar un temporizador a un valor determinado
`clock_gettime()`	Obtener el valor de un temporizador
`clock_getres()`	Obtener la resolución de un temporizador
`clock_nanosleep()`	Demorar la ejecución del hilo actual con una espera de alta resolución
`timer_create()`	Crear un temporizador
`timer_delete()`	Eliminar un temporizador
`timer_settime()`	Establecer un temporizador
`timer_gettime()`	Obtener el valor del tiempo pendiente de un temporizador
`timer_getoverrun()`	Obtener el valor del tiempo excedido por un temporizador

Tabla 2.3: Funciones de la especificación POSIX 1003.1-2008 para relojes y temporizadores.

Pese a que no es un mecanismo de temporización propiamente dicho, es posible configurar la ejecución demorada de procedimientos a través de hilos POSIX. Para ello puede utilizarse la función `pthread_cond_timedwait`, que permite que un hilo espere a una condición para continuar con su ejecución. Sin embargo, en algunas implementaciones (principalmente en sistemas multiprocesador) puede darse el caso de que varios hilos retomen su ejecución cuando el cumplimento de la condición se comunica simultáneamente a varios procesadores. Por lo tanto, es recomendable evaluar de nuevo la condición cuando la espera termina y el hilo continúa, puesto que puede darse el caso de que se haya recibido un aviso equivocado.

Finalmente, en algunas implementaciones de sistemas operativos compatibles con UNIX se dispone de una API de temporizadores de alta resolución a nivel de núcleo. En el caso de GNU/Linux (a partir de la versión 2.6.21), estos temporizadores funcionan de forma independiente a los temporizadores dinámicos previamente descritos. En lugar de emplear *jiffies* como medida de referencia, emplean el hardware de temporización de alta resolución para obtener una granularidad de ns. Otra diferencia reside en el hecho de que se emplea una estructura de árbol rojo-negro en el núcleo para optimizar su rendimiento y minimizar los retardos en lugar de

añadirse a la «rueda de temporizadores» (Jones, 2010). Además de las funciones disponibles desde el propio núcleo, la activación de este sistema de temporizadores de alta resolución se incorpora a las API ya comentadas a nivel de usuario (temporizadores de intervalo y temporizadores POSIX).

Reloj de referencia	Descripción
`CLOCK_REALTIME`	Representa el reloj de tiempo real del sistema. En GNU/-Linux se actualiza aproximadamente cada ms (999,848 ns).
`CLOCK_MONOTONIC`	Reloj monotónicamente creciente que no puede configurarse y mide el tiempo a partir de un momento no especificado en el pasado que no cambia en cada reinicio del sistema.
`CLOCK_MONOTONIC_RAW`	Reloj monotónicamente creciente no influido por ninguna configuración externa.
`CLOCK_PROCESS_CPUTIME_ID`	Reloj que mide el tiempo de procesador empleado por el proceso (incluye el tiempo de usuario y del sistema de todos sus hilos).
`CLOCK_THREAD_CPUTIME_ID`	Reloj que mide el tiempo de procesador empleado por el presente hilo.

Tabla 2.4: Relojes de referencia disponibles para la creación de temporizadores POSIX.

2.3.2 Funciones de tiempo

Si bien el uso de temporizadores adecuados puede contribuir a la presentación exacta y precisa de contenido audiovisual, la disponibilidad de funciones de tiempo con la resolución, exactitud y precisión necesarias es un requisito necesario para la obtención ajustada de la entrada del usuario. En este sentido, resulta razonable comparar este escenario con lo que sucede en un sistema *de tiempo real* con una planificación dirigida por eventos (*event-driven*), puesto que la respuesta del usuario será capturada por el sistema en forma de suceso o evento, al que habrá que

asignar una marca de tiempo lo más ajustada posible al momento en el que se produjo.

Una vez más, las opciones disponibles en los principales sistemas operativos son variadas y se corresponden con la evolución producida en la arquitectura hardware sobre la que operan. En la tabla 2.5 puede verse un resumen de las principales API para el uso de funciones de tiempo en sistemas compatibles con Windows y UNIX respectivamente.

S.O.	Funciones de tiempo
Windows	`timeGetTime` (influida por `timeBeginPeriod`)
	`GetSystemTime` (basada en `GetTickCount`)
	`QueryPerformanceCounter` y `QueryPerformanceFrequency`.
UNIX-like	`gettimeofday`
	`clock_gettime`
	`mach_absolute_time` y `mach_timebase_info` (Mac OS X)
	`ktime_get` (GNU/Linux)

Tabla 2.5: Resumen de las API para el uso de funciones de tiempo en sistemas Microsoft Windows y compatibles con UNIX.

Las obligaciones adquiridas por los sistemas operativos de Microsoft en aras de la compatibilidad con versiones previas han provocado que sus API de funciones de tiempo estén disgregadas, ofreciendo diferentes mecanismos con muy diversas resoluciones y exactitudes. Quizá pueda resultar chocante que la función de tiempo general del sistema (comúnmente conocida como *wall clock*) sólo se actualice cada 15 ms aproximadamente (15,625 ms en Windows XP, 15,600100 ms en Windows Vista y Windows 7; ver Krishnan, 2008; Russinovich, Solomon, y Ionescu, 2009) debido a que el periodo del *tick* del sistema es constante y la llamada a la API de Windows `GetSystemTime` se apoya en `GetTickCount`, que precisamente devuelve el número de *ticks* desde el arranque del sistema multiplicados por el tamaño de su periodo. Este intervalo de tiempo también es importante,

ya que en versiones anteriores a Windows Vista el planificador del sistema utiliza múltiplos de él para asignar los tiempos de ejecución de cada hilo (*quantum*). Como ya hemos comentado con anterioridad, si se utilizan las llamadas a la API de Windows `timeBeginPeriod` o `NtSetTimerResolution` con una resolución mayor que el periodo del *tick*, los valores devueltos por `timeGetTime` y `GetTickCount` diferirán, puesto que la primera utilizará temporizadores de mayor resolución y exactitud. Esta configuración tiene un impacto significativo en el sistema, ya que el número de interrupciones de temporizador se verá incrementado sustancialmente. Por ello, para obtener funciones de tiempo de alta resolución a través de la API de Windows lo más recomendable es apoyarse en dos llamadas relacionadas entre sí, `QueryPerformanceCounter` (para leer el valor del contador) y `QueryPerformanceFrequency` (para conocer la frecuencia de ese contador), que elegirán el mecanismo hardware subyacente más apropiado en cada arquitectura. En condiciones favorables, `QueryPerformanceCounter` puede ofrecer resoluciones por debajo de 1 μs, aunque su coste de invocación se sitúa por encima de ese valor (Kuperberg, Krogmann, y Reussner, 2009). Como puede apreciarse en la tabla 2.6, esto supone una ventaja de hasta cuatro órdenes de magnitud frente a otras llamadas al sistema (Wilson, 2003). Estos valores coinciden con los aportados por Zhang y cols. (2009), que además de evaluar las llamadas al sistema de forma directa, comprobaron su uso por llamadas de más alto nivel en Visual Basic 6 y Microsoft .Net 2.0.

Función de tiempo	Documentada	Win98	WinNT4	WinNT4 (SMP)	Win2000	WinXP
`GetTickCount`	1000	1000	10.000	15.000	10.000	15.000
`timeGetTime`	1000	1000	10.000	15.000	10.000	15.000
`GetSystemTimeAsFileTime`	0,1	50.000	10.000	15625	10.000	15.625
`QueryPerformanceCounter`	Hardware	5	2	1	1	1
`GetThreadTimes`	0,1	N/A	10.000	15.625	10.000	15.625
`GetProcessTimes`	0,1	N/A	10.000	15.625	10.000	15.625

Tabla 2.6: Resolución (en μs) de funciones de tiempo en sistemas Microsoft Windows (Wilson, 2003).

En el caso de sistemas operativos compatibles con UNIX como Apple Mac

OS X o GNU/Linux, la función de tiempo más común es la llamada al sistema `gettimeofday`, que devuelve la hora actual con una resolución de μs y un coste de invocación en torno a 1 μs. Si se utilizan los mecanismos de planificación adecuados, su exactitud y precisión son elevadas (DT del orden de decenas de μs en el peor caso), pero pueden verse perjudicadas severamente si la carga del sistema es alta y no se utilizan dichos mecanismos (Finney, 2001). Otro problema grave que afecta a `gettimeofday` es su falta de monotonicidad. Un temporizador monotónico –o monotónicamente creciente– siempre devuelve valores al menos iguales o superiores al valor previo. Esta característica es de vital importancia para medir intervalos temporales, puesto que su ausencia puede provocar mediciones con valores negativos. Para evitar esto, es recomendable utilizar la llamada al sistema `clock_gettime` con el parámetro `CLOCK_MONOTONIC`, que está soportada por la mayoría de sistemas compatibles con UNIX y nos asegura la monotonicidad del temporizador. No obstante, `clock_gettime` también tiene sus contrapartidas, entre las que destacan la falta de soporte en algunas plataformas (Apple Mac OS X entre ellas) y la falta de estabilidad entre reinicios del sistema (no conviene utilizarlo como mecanismo de obtención de marcas de tiempo para generar registros históricos del sistema). Para sustituir a `clock_gettime` en Apple Mac OS X se dispone de dos funciones, `mach_absolute_time` y `mach_timebase_info`, que se emplean de forma muy similar a las llamadas `QueryPerformanceCounter` y `QueryPerformanceFrequency` de la API de Windows, respectivamente.

Como forma de solventar algunos de los problemas mencionados, algunos autores han sugerido el uso de métodos estadísticos para obtener medidas de alta resolución empleando temporizadores de baja resolución (Beilner, 1988; Danzig y Melvin, 1990; Lambert y Power, 2008). Sin embargo, estos enfoques no podrán aplicarse al caso del registro de la interacción del usuario en aplicaciones web, ya que requieren alta resolución en una sola medida, que difícilmente podría repetirse para que convergiera a un valor de mayor resolución.

2.4 Análisis de la precisión y exactitud del software especializado

El número disponible de paquetes de software especializado en la presentación exacta y precisa de contenido audiovisual para la experimentación es realmente elevado. Debido a esta amplia oferta, los investigadores se ven a menudo obligados a aprender un gran número de opciones de configuración y lenguajes de programación propios en los que no todas las funcionalidades son coincidentes (i. e., la sintaxis de configuración de un programa puede tener menos opciones que la de otro programa, pero incluir un pequeño conjunto de funcionalidad extra que el programa con más opciones no implementa). Si a este escenario añadimos el hecho de que la mayoría de investigadores en el campo de la visión, la audición o la experimentación neurocientífica en general no tiene unos profundos conocimientos de programación, las posibilidades de cometer errores técnicos se incrementan. En la tabla 2.7 puede verse un resumen de las características de algunos de estos paquetes de software especializado.

Nombre	GUI	Libre	Scripting	Plataforma	Referencia
DirectRT	Sí	No	Custom	Windows	Stahl (2006)
DMDX	No	Gratis	Custom	Windows	K. Forster y Forster (2003)
E-Prime	Sí	No	E-Basic	Windows	Schneider, Eschman, y Zuccolotto (2002); Stahl (2006)
Experiment Builder	Sí	No	Python	Windows	SR Research
Inquisit	Sí	No	Custom	Windows	Stahl (2006)
Matlab Psychophysics toolbox	No	Sí	Matlab	Win/Mac/Lin	Brainard (1997)
MEL	Formularios	No	Custom	IBM PC	Schneider (1988)
PEBL	No	Sí	Custom	Win/Mac/Lin	Mueller (2010)
Presentation	Sí	No	Custom	Windows	Neurobehavioral Systems
PsychoPy	Sí	Sí	Python	Win/Mac/Lin	Peirce (2007)
PsyScope	Sí	Sí	Custom	Mac OS	Cohen, MacWhinney, Flatt, y Provost (1993)
PsyToolkit	No	Sí	Custom	Linux	Stoet (2010)
PyEPL	No	Sí	Python	Mac/Lin	Geller, Schleifer, Sederberg, Jacobs, y Kahana (2007)
SuperLab	Sí	No	Custom	Windows	Stahl (2006)
Tscope	No	Sí	C/C++	Win/Mac/Lin	Stevens, Lammertyn, Verbruggen, y Vandierendonck (2006)
Vision Egg	No	Sí	Python	Win/Mac/Lin	Straw (2008)
OpenSesame	Sí	Sí	Python	Win/Mac/Lin	Mathôt, Schreij, y Theeuwes (2012)

Tabla 2.7: Características de los principales paquetes de software especializado en experimentación (Mathôt y cols., 2012).

Ante esta abrumadora oferta, muchos investigadores optan por emplear una herramienta que ofrezca un buen compromiso entre sencillez de uso y funcionalidades, además de contar con el aval de haber sido utilizada en muchos otros estudios previos. La herramienta que mejor encaja en este perfil es E-Prime (Schneider y cols., 2002). Más que un único programa de experimentación, E-Prime es un conjunto de aplicaciones que se combinan para cubrir todos los pasos implicados en la creación y despliegue de un experimento por ordenador. Así, dentro de E-Prime se distinguen los siguientes componentes: 1) E-Studio, una interfaz de usuario gráfica para el diseño de experimentos; 2) E-Basic, un lenguaje de *scripting* propio (prácticamente idéntico a Visual Basic for Applications de Microsoft); 3) E-Run, un intérprete de E-Basic capaz de asegurar que los experimentos se ejecutan con una precisión de 1 ms en la presentación, sincronización y recogida de datos; 4) E-Merge, una herramienta para la combinación de ficheros de datos; 5) E-DataAid, una utilidad de filtrado, edición, análisis y exportación de datos; y 6) E-Recovery, un sistema de recuperación de datos experimentales desde ficheros de datos corruptos o experimentos terminados inesperadamente. Una vez que un investigador ha diseñado con E-Studio su experimento, éste generará un fichero en E-Basic que será ejecutado por E-Run. Al finalizar el experimento, los datos serán eventualmente tratados por E-Merge o E-DataAid, y podrán ser reparados mediante E-Recovery. Si bien la ejecución de un experimento con E-Run puede hacerse desde cualquier ordenador que tenga instalado E-Prime (o incluso desde un ejecutable preparado para ello que no requiere la instalación de E-Prime), el uso de E-Studio está limitado a aquellos que hayan adquirido una licencia de uso. La empresa desarrolladora de esta herramienta (Psychology Software Tools) pone especial cuidado en que esto sea así a través de una llave criptográfica USB que se entrega con el software.

Tal vez sea el hecho de que no todos los investigadores están decididos a pagar una licencia de uso de E-Prime lo que ha motivado que, incluso para Windows –única plataforma en la que se ejecuta E-Prime–, se hayan desarrollado numerosas alternativas gratuitas. De entre todas estas alternativas gratuitas para Windows, probablemente la más empleada sea DMDX (K. Forster y Forster, 2003). La principal razón no es su facilidad de uso, sino la gran comunidad de usuarios que desde 1975 venían utilizando DMASTR, un conjunto de utilidades de experimentación desarrolladas en la Monash University de Australia. Este software fue ampliado a

lo largo de los años por diversos autores y portado a Windows 95 / 98 / Me por J. Forster en 1997. A pesar de las reticencias iniciales que provocaron la llegada de Windows NT y 2000 en relación al acceso directo al hardware y la exactitud y precisión de los mecanismos de temporización disponibles, K. Forster y Forster (2003) idearon la manera de migrar el antiguo DMASTR a Windows 2000 / XP y superiores a través del uso de DirectX y temporizadores multimedia. Este es el motivo de su nombre, que es la suma de DMASTR y DirectX: DMDX.

Aunque DMDX es una aplicación con una interfaz gráfica, el proceso para diseñar, definir y llevar a cabo un experimento sigue muy orientado al trabajo con comandos. Este proceso comprende las siguientes fases: 1) el investigador define la configuración de su experimento mediante un fichero RTF (*Rich Text File*) empleando un editor estándar para ello (v. gr., WordPad, Open/LibreOffice, Microsoft Word); 2) DMDX analiza todos los comandos definidos en el fichero anterior y prepara la lista de ítems que serán presentados (i. e., desordena la lista si fuera necesario, genera los ítems pseudoaleatorios que se hayan requerido, etc.); 3) si todo es correcto, DMDX inicia la preparación de la presentación del experimento, que comienza con la petición a DirectX de que cambie al modo de pantalla indicado (previamente se deberá haber calibrado empleando la herramienta TimeDX) y la creación de tantos *buffers* de pantalla (i. e., DirectDraw Surfaces de DirectX) como sean necesarios; 4) se inicializa el temporizador multimedia; 5) se crea un hilo que estará a la espera de la señal de redibujado de pantalla; si al cabo del tiempo de redibujado definido en TimeDX no hubiera recibido esa señal, entendería que la ha perdido y volvería a sincronizarse; 6) se crea un hilo para el control de la entrada del usuario; y 7) si hubiera elementos de audio definidos en el fichero de configuración, se crearía un hilo para controlar su reproducción.

```
1 -001 "+" %60 / "guerra" %15 / * "muerte" %120 / ;
2 +002 "+" %60 / "bomba" %15 / * "alegria" %120 / ;
```

Listado 2.4: Definición de items en un fichero de configuración para DMDX / DMASTR.

A nivel de usuario, la mayor ventaja de DMDX (i. e., su compatibilidad con DMASTR) también es su principal problema, puesto que la sintaxis de los ficheros de configuración dista mucho de ser intuitiva para quien se enfrenta por primera vez a la herramienta. En el listado 2.4 se muestra un ejemplo de definición de dos ítems. El primer carácter de cada línea indica si la respuesta correcta será positiva (+) o negativa (−), también pueden configurarse ítems que no requieran respuesta o que cualquier respuesta sea correcta. Después, se indica un valor numérico que identifica al ítem. Si dos ítems tienen el mismo valor numérico, solamente se almacenará la respuesta del último. Si se indica el valor cero como número de ítem, se entenderá que es un mensaje de instrucciones y no un ítem en sí. Posteriormente se definen tantos elementos de tipo marco (*frame*) como sean necesarios, delimitándolos con el separador `/`. Cada uno de estos marcos puede contener texto (delimitado por comillas dobles) o referencias a ficheros de imágenes, vídeo o audio. Además, puede configurarse la duración en *ticks* de cada uno de ellos empleando el carácter `%`. El carácter `*` indicará el punto a partir del cual se comenzará a medir la respuestas del usuario. La definición de un ítem finaliza con el carácter `;`. Así, en el ejemplo mostrado se definen dos ítems en los que se quiere influir en el rendimiento del usuario a la hora de clasificar una palabra como positiva o negativa mostrando muy brevemente otra palabra. El primer ítem (`001`) requiere pulsar la tecla definida como negativa y comenzará mostrando el símbolo + en el centro de la pantalla durante 60 *ticks* (1000 ms a 60 Hz), seguidamente mostrará la palabra `guerra` durante 15 *ticks* (250 ms a 60 Hz) y en el siguiente marco activará el registro de la respuesta del usuario (mediante `*`) y mostrará durante 120 *ticks* (2000 ms a 60 Hz) la palabra `muerte`. Por último, el ítem terminará con un marco vacío. El segundo ítem (`002`) es muy similar, salvo en lo que respecta a la tecla definida como correcta (positiva en este caso). Como podemos observar, la definición de cada ítem requiere el conocimiento de una sintaxis especial de complejidad creciente cuando se requieren funcionalidades avanzadas como saltos condicionales o la generación automática de ítems, entre otras.

Además de estas dos herramientas, dentro de la oferta para Windows hay muchas otras empresas que disputan el mercado comercial a E-Prime. Tal es el caso de las desarrolladoras de paquetes de software como DirectRT, SuperLab o Inquisit, descritos con detalle por Stahl (2006), entre muchos otros.

Otros desarrolladores, por contra, optaron desde el principio por compartir sus herramientas con el resto de la comunidad (Brainard, 1997; Cohen y cols., 1993; Geller y cols., 2007; Mathôt y cols., 2012; Mueller, 2010; Peirce, 2007; Stevens y cols., 2006; Stoet, 2010; Straw, 2008). Algunos de estos proyectos han aprovechado el auge del lenguaje Python (Tiobe, 2012) para apoyarse en bibliotecas ya desarrolladas y no tener que empezar de cero. Además, se dan otras ventajas por el uso de Python: 1) la naturaleza inherentemente multiplataforma de este lenguaje proporciona la posibilidad de ejecutar estas herramientas tanto en Windows como en Mac OS y GNU/Linux; 2) la claridad de su sintaxis permite su uso como lenguaje de *scripting* para los usuarios de cada herramienta, evitando emplear un lenguaje propio de la misma; y 3) en los casos en los que el rendimiento sea crítico, podrá compilarse para agilizar su ejecución. Estas razones, unidas al desarrollo de una interfaz gráfica de usuario muy similar a la de E-Prime, han hecho de PsychoPy una alternativa libre y multiplataforma con una creciente comunidad de usuarios (Peirce, 2007).

Una vez descritas sus características, resulta procedente cuestionar su validez en la presentación de contenido audiovisual y el registro de la interacción del usuario de forma exacta y precisa. Tal y como afirman MacInnes y Taylor (2001), el software comercial no suele validarse por terceros, principalmente debido a que no se dispone de acceso a su código fuente, e incluso en algunos casos no es posible su ejecución sin una licencia de uso. Con el objetivo de diseñar un sistema de validación de paradigmas experimentales en ordenadores personales, Plant, Hammond, y Whitehouse (2002) desarrollaron un conjunto de hardware y software capaz de medir de forma externa el comportamiento de cualquier software de experimentación. Esta propuesta sería perfeccionada más adelante en lo que denominaron el *Black Box Toolkit* (Plant y cols., 2004) (explicado con detalle en el capítulo 3).

Los desarrolladores del software de experimentación SuperLab afirman en su documentación que su programa es capaz de registrar tiempos de respuesta con una precisión de 1 ms, tanto en Windows como en Mac OS X, a pesar de que recuerda que esta precisión esta supeditada a la que pueda ofrecer el dispositivo de entrada y a los temporizadores empleados. Por esta razón, el uso de los temporizadores multimedia es la opción recomendada en la versión para Windows (ver figura 2.5).

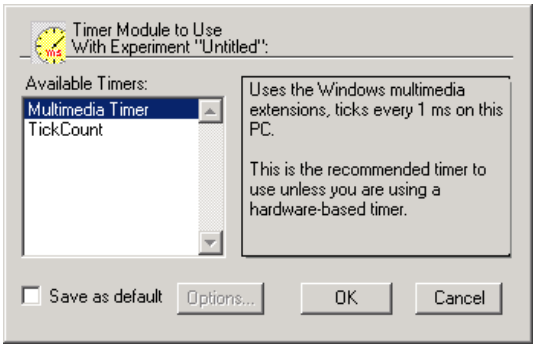

Figura 2.5: Configuración de temporizadores en SuperLab.

Además de los tiempos de respuesta, es necesario asegurar la precisión y exactitud de los tiempos de presentación. En este tipo de experimentos hay una serie de momentos en los que la precisión es crítica y otros momentos –típicamente los intervalos entre las diferentes presentaciones– que pueden aprovecharse para preparar el contenido crítico. Por este motivo, la mayoría de software especializado sigue una estrategia de «preparar y ejecutar», en la que todo el contenido que se presentará de forma precisa se preparará de antemano para evitar cualquier posible retardo cuando llegue el momento (Mathôt y cols., 2012). DMDX utiliza tantos buffers de DirectX como sea posible para este cometido y los prepara antes de que dé comienzo el experimento. E-Prime emplea un procedimiento llamado `PreRelease`, a través del que la siguiente presentación se comienza a preparar antes de que termine la presentación anterior, minimizando los retardos entre ellas. De esta forma, sus desarrolladores afirman que E-Prime puede asegurar una precisión de 1 ms, entendiendo como tal que la DT de los tiempos medidos no superará los 0,5 ms (Schneider y cols., 2002). Asegurarse con total certeza que ninguna de las presentaciones o registros de respuesta sufrirá algún retardo superior a 1 ms a lo largo de un experimento no es posible si se emplea un sistema operativo con multitarea con desalojo como Microsoft Windows. De cualquier manera, E-Prime guarda un registro de todos los retardos con una alta exactitud y precisión (error $< 0{,}5$ ms), por lo que los experimentadores pueden decidir si filtrar o no los ensayos en los que ha habido algún problema.

Como ya hemos comentado con anterioridad, tanto SuperLab como DMDX se apoyan en los temporizadores multimedia para la ejecución demorada de procedi-

miento. En el caso de E-Prime no es posible conocer con detalle qué mecanismo se emplea, pero un análisis de las llamadas a las API de Win32 de cada uno de sus componentes obtenida a través de la herramienta `WinAPIOverride32` nos confirma el uso de funciones de tiempo (`QueryPerformanceCounter` junto con `QueryPerformanceFrequency`) de alta precisión. El empleo de los temporizadores estándar de Windows (funciones `SetTimer` y `KillTimer`) solamente se muestra en componentes que no afectan a E-Run, por lo que no se utilizan para gestionar la presentación del contenido audiovisual sino en otras tareas menos criticas. En la tabla 2.8 se recoge un listado de las llamadas a las API de Win32 relacionadas con mecanismos de temporización que se encuentran en componentes de E-Prime.

DLL	**Llamada a la API**
KERNEL32.DLL	Sleep
KERNEL32.DLL	SetThreadPriority
KERNEL32.DLL	GetThreadPriority
KERNEL32.DLL	SetPriorityClass
KERNEL32.DLL	QueryPerformanceFrequency
KERNEL32.DLL	QueryPerformanceCounter
KERNEL32.DLL	GetTickCount
KERNEL32.DLL	GetSystemTimeAsFileTime
KERNEL32.DLL	GetSystemTime
KERNEL32.DLL	GetLocalTime
KERNEL32.DLL	FileTimeToSystemTime
KERNEL32.DLL	FileTimeToLocalFileTime
USER32.DLL	SetTimer
USER32.DLL	KillTimer
E-RUNTIME.DLL	ebxSetRealTimeClock
E-RUNTIME.DLL	ebxGetRealTimeClock

Tabla 2.8: Llamadas a las API de Win32 de E-Prime 2.0.

Sus propios autores midieron la validez de DMDX, reportando cero errores tras 100 tests. En cuanto a los tiempos de respuesta medidos, encontraron una DT de 0,53 ms (K. Forster y Forster, 2003). Este alto grado de precisión se logra gracias a dos soluciones diferentes, explicadas a continuación. Para minimizar los errores en la presentación de contenidos, DMDX se apoya en el reloj de tiempo real del sistema, que es independiente de la carga del mismo. Si DMDX no consigue detectar la señal de refresco de pantalla en el tiempo configurado (a través de la calibración con TimeDX), entenderá que ha perdido esa señal pero que el refresco se ha dado, por lo que ajusta su contador de acuerdo a este hecho. Dado que los tiempos de espera son conocidos, el hilo puede estar ocioso hasta poco antes de que se dé el refresco de pantalla, momento en el que comprobará activamente si el refresco se ha producido. Para el caso de los posibles errores en el registro de la interacción del usuario, DMDX realiza una consulta activa de los dispositivos cada ms, a través de un temporizador multimedia.

La versión de PsychoPy para Windows también emplea funciones de tiempo de alta precisión (la habitual combinación de `QueryPerformanceCounter` junto con `QueryPerformanceFrequency`), mientras que las versiones para GNU/Linux y Mac OS X se apoyan en `gettimeofday`, por lo que la exactitud y precisión del registro temporal están garantizadas en todas las plataformas soportadas. En cuanto a la presentación de contenidos, PsychoPy ofrece la posibilidad de definir la duración de cada elemento de la presentación en ms, por lo que aquellos requisitos temporales que no sean un múltiplo del intervalo de refresco no podrán ser cumplidos sin demora por el hardware de visualización. Sin embargo, PsychoPy puede emplearse tanto como herramienta de autor como paquete de software sobre el que desarrollar *scripts* en Python que importen sus funcionalidades. Si este es el caso, PsychoPy ofrece al desarrollador la posibilidad de sincronizar su presentación con el refresco de pantalla y ajustar al máximo la exactitud y precisión de la misma.

2.5 Análisis de la precisión y exactitud de tecnologías web

Dentro de las soluciones para la temporización precisa de estímulos mediante tecnologías web podemos distinguir entre aquellas que están basadas en animaciones declarativas y aquellas basadas en animaciones procedimentales implementadas en un lenguaje de programación adaptado a un agente de usuario web. Su diferencia radica en que mientras las primeras describen qué es lo que se desea que suceda, las segundas detallan cómo va a obtenerse el resultado deseado. Ambos enfoques tienen sus ventajas e inconvenientes.

En el caso de las animaciones declarativas, su principal ventaja es que se abstraen totalmente de los detalles de implementación a bajo nivel necesarios para llevar a cabo lo que solicitan. Sin embargo, dada la flexibilidad de los estándares empleados para definir este tipo de animaciones, las solicitudes pueden resultar poco realistas en función del software o hardware en el que se procesen, ocasionando la producción de animaciones significativamente diferentes a las diseñadas (v. gr., una animación que solicite una transición de color cada 200 ms procesada por un dispositivo con un temporizador cuya granularidad sea de 1 s).

Las animaciones procedimentales parten con la ventaja de conocer los detalles de más bajo nivel, pudiendo decidir entre diferentes aproximaciones para implementar la animación deseada en función de las características del software o hardware en el que está siendo procesada. Por el contrario, su principal desventaja es que resulta bastante costoso contemplar toda la casuística existente en cuanto a entornos de ejecución de animaciones (i. e., diferentes versiones de agentes de usuario, intérpretes de lenguajes, sistemas operativos, plataformas hardware, etc.) y el código desarrollado puede sufrir un sobreajuste a las condiciones de ejecución del momento en el que fue implementado, funcionando incorrectamente en futuras condiciones no contempladas.

Las aproximaciones realizadas hasta la fecha para la temporización de estímulos a través de animaciones declarativas han empleado principalmente los estándares GIF89a, SVG (*Scalable Vector Graphics*) y SMIL (*Synchronized Multimedia Integration Language*). En 2001, Schmidt evaluó la precisión de diferentes métodos de animación web, entre los que se encontraban las animaciones basadas en

GIF89a (Schmidt, 2001). Este estándar tiene la ventaja de estar soportado por la práctica totalidad de los agentes de usuario web, pero presenta graves problemas relacionados con su temporización. El más importante tiene que ver con la imposibilidad de sincronizar las animaciones basadas en GIF89a con otros eventos, lo que las descarta para aplicaciones que requieran exactitud y precisión en la presentación de varios elementos audiovisuales o el registro de la interacción del usuario en función al estado de la animación. Otro problema reside en la escasa granularidad en las definiciones de intervalos temporales dado que se utilizan centésimas de segundo en lugar de ms (CompuServe, 1990). Esta limitación resulta incomprensiblemente enmascarada por muchos generadores de animaciones en GIF89a que permiten configurar las duraciones de cada imagen en ms, a pesar de que finalmente se redondearán a centésimas de segundo (v. gr., GIMP – The GNU Image Manipulation Program). De forma similar, algunos agentes de usuario web modifican silenciosa y automáticamente los intervalos fijados en las animaciones en GIF89a si éstos son demasiado exigentes (v. gr., en todas las versiones de Microsoft Internet Explorer, si el intervalo es inferior a 50 ms., automáticamente se fija a 100 ms). El último problema corresponde a la precisión con la que se muestran las animaciones declaradas en GIF89a, ya que tras la evaluación de Schmidt se estimó que no era recomendable definir animaciones con intervalos por debajo de los 200 ms.

En lo referente a SVG y SMIL, muchas de sus características recomiendan su uso en este contexto: 1) su naturaleza declarativa facilita su reproducción en diferentes plataformas, a la vez que aporta información semántica al agente de usuario; 2) los intervalos temporales son definidos en segundos, pero pueden utilizarse tantas cifras decimales como se desee, dejando al agente de usuario la decisión de con cuánta resolución será posible mostrar los cambios en la animación; y 3) al tratarse de dialectos de XML, son fácilmente modificables tanto por humanos como por sistemas de información. Sin embargo, a pesar de los estudios teóricos sobre su semántica temporal (Chung y Pereira, 2005) y los aportes para dotarlos de un carácter más reactivo (King, Schmitz, y Thompson, 2004), no se disponen de datos sobre la exactitud en el cumplimiento de los intervalos temporales fijados en SMIL bajo ninguno de los reproductores más usados en el pasado para visualizar este tipo de animaciones, entre los que destacan QuickTime Player de Apple, Windows Media Player de Microsoft y RealPlayer de RealNetworks, además de los propios

agentes de usuario web que, bien de forma nativa o mediante complementos, son capaces de mostrar animaciones descritas en SVG y SMIL.

El uso de CSS (*Cascading Style Sheets*) para la definición de animaciones declarativas es la principal apuesta del W3C y de los desarrolladores de los agentes de usuario web mayoritarios, que ya incorporan esta funcionalidad en sus últimas versiones, a pesar de que su especificación todavía no haya alcanzado el estado de estándar (Jackson, Hyatt, Marrin, Galineau, y Baron, 2012b). Quizá sea este el motivo por el cual se carece de un estudio de la exactitud y precisión de esta tecnología.

Las soluciones basadas en animaciones procedimentales pueden diferenciarse a su vez en dos categorías: aquellas que utilizan tecnologías basadas en estándares (v. gr., JavaScript, SVG y DOM, principalmente) y aquellas que se ejecutan en máquinas virtuales (v. gr., Flash, Java, Silverlight, Authorware, etc.) o entornos restringidos de ejecución de código nativo como Microsoft ActiveX (Chappell, 1996), Gecko NPAPI (Oeschger, 2002) o Google NativeClient (Yee y cols., 2009), que es preciso asociar a los agentes de usuario web mediante complementos o *plugins*.

Dado que el objetivo principal es maximizar la exactitud y precisión de las temporizaciones de las animaciones, podría parecer que lo más apropiado sería utilizar código nativo a través de los complementos del agente de usuario que lo hagan posible. Sin embargo, tanto el uso de Microsoft ActiveX como de Gecko NPAPI ha demostrado que confiar en mecanismos manuales para conceder privilegios de ejecución al código nativo desde el agente de usuario no es una buena política de seguridad y actualmente su utilización se restringe a *intranets* corporativas donde esos problemas de seguridad pueden ser monitorizados y aislados. Más aún, ambas tecnologías están circunscritas a agentes de usuario concretos (Microsoft Internet Explorer en el caso de ActiveX, y la familia de navegadores Gecko, donde Mozilla Firefox es el ejemplo más conocido, en el caso de NPAPI) por lo que distan de ser soluciones multiplataforma. La tercera opción para la ejecución de código nativo, Google Native Client, promete solucionar ambos problemas proporcionando un entorno de ejecución protegido automáticamente y multiplataforma, pero no se disponen de datos que lo confirmen al tratarse de una tecnología aún en desarrollo.

Las tecnologías basadas en máquinas virtuales como Java o Flash han sido ampliamente utilizadas para controlar la temporización de contenido audiovisual a lo

largo de sus más de diez años de existencia. Schmidt (2001) evaluó su exactitud y precisión, llegando a la conclusión de que se requerían intervalos de al menos 100 ms para que fueran mostrados correctamente mediante Java y de 250 ms para el caso de Flash. Sin embargo, las sucesivas mejoras implementadas en ambas máquinas virtuales hacen necesaria la actualización de esos datos, así como el análisis de otras tecnologías menos populares como Silverlight.

Por último, el uso de JavaScript como lenguaje para la generación de animaciones procedimentales en la Web ha sido la opción preferente de la mayoría de desarrolladores web, fundamentalmente porque no exige para su ejecución la instalación de ningún complemento en los principales agentes de usuario web (i. e., Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Apple Safari, Opera). Sin embargo, esta solución tampoco está exenta de problemas. El más importante de ellos es su velocidad de ejecución. No ha sido hasta la popularización de aplicaciones con interfaces web sofisticadas (v. gr., Google Maps, GMail) que se ha elevando el grado de exigencia acerca del rendimiento de los intérpretes de JavaScript de los agentes de usuario.

Además del empleo de conocidos tests de rendimiento (v. gr., WebKit SunSpider, Mozilla Dromaeo y Google V8 Benchmark Suite), Ratanaworabhan, Livshits, y Zorn (2010) estudiaron su rendimiento en aplicaciones web reales concluyendo que todavía existe margen de mejora, por lo que los distintos equipos de desarrollo de intérpretes JavaScript están aplicando técnicas como la compilación JIT (*Just-In-Time*, ver Gal y cols., 2009), optimizaciones a nivel de *bytecode*, uso de memorias caché en línea polimórficas o máquinas virtuales a sus últimas propuestas (Mozilla JägerMonkey, Safari Nitro, Google V8, Microsoft Chakra u Opera Carakan). A pesar de estos esfuerzos, la definición del estándar ECMAScript en el que se basa JavaScript dificulta el uso de temporizadores precisos debido a dos factores: 1) no pueden configurarse para ser disparados en un periodo inferior a 4 ms, y 2) la gestión temporal de eventos de JavaScript impide el control preciso de las animaciones en situaciones de alta concurrencia (Ecma, 2011). El primero de estos factores no es más que el resultado de una decisión arbitraria, y por lo tanto tiene fácil solución. De hecho, la limitación para temporizadores aperiódicos (a través de `setTimeout`) ya fue rebajada de 10 a 4 ms en la última revisión. Pero ir más allá tendría un coste significativo, ya que existe una gran cantidad de aplicaciones

web que aprovechan esta limitación arbitraria para fijar intervalos menores (de 0 ms típicamente) que serán atendidos en el menor tiempo posible sin afectar al rendimiento del intérprete, por lo que cualquier cambio en este sentido podría romper la compatibilidad del intérprete con una gran base de aplicaciones web desplegadas (Belshe, 2010). El segundo de los factores que afectan a las funciones para crear temporizadores periódicos y aperiódicos (`setInterval` y `setTimeout`) implica de manera más profunda al diseño del propio intérprete de JavaScript y su gestión temporal de la cola de eventos a través de una política sin prioridades ni desalojo, que puede originar esperas inadmisibles en el caso de `setTimeout` o descartes en intervalos en el caso de `setInterval`.

Dentro del conjunto de pruebas que realizó Schmidt (2001) se encuentran las relativas a animaciones procedimentales en JavaScript. Sus conclusiones, que deben entenderse en su contexto, son que JavaScript no debería emplearse para animaciones procedimentales con una duración menor a 120 ms. En 2010, Adam comparó animaciones procedimentales programadas en JavaScript frente a Flash y encontró diferencias entre ambas en favor de Flash (Adams, 2010). Sin embargo, en los últimos años se ha producido una verdadera revolución en el desarrollo de nuevas API en torno al estándar HTML5 y su adopción por los principales desarrolladores de agentes de usuario web, en especial en lo que respecta a Google Chrome y Mozilla Firefox, por lo que conviene actualizar estos resultados.

En lo referente a la medición exacta y precisa de la interacción del usuario, diversos investigadores han analizado a lo largo de la última década distintas tecnologías web, abundando los estudios relativos a Java y Flash.

En 2001, Eichstaedt describió un filtro para descartar las mediciones poco precisas de los tiempos de respuesta en applets Java utilizando un hilo de control con un temporizador como mecanismo para detectar la influencia de la carga del sistema operativo en el proceso de medida (Eichstaedt, 2001). Keller, Gunasekharan, Mayo, y Corley (2009) propusieron un mecanismo de filtrado de las medidas inexactas similar al Eichstaedt (2001) dentro de su suite de experimentación online WebExp, también implementada en Java, obteniendo unas DT en torno a 1 ms después del filtrado de las medidas inexactas.

A su vez, McGraw, Tew, y Williams (2000) compararon la precisión de los tiempos de respuesta obtenidos utilizando Macromedia Authorware (una herramienta

de autor, ya descontinuada, para la creación de aplicaciones que podían ejecutarse tanto de forma nativa como desde el agente de usuario web) frente al conocido software de experimentación taquistoscópica E-prime, variando agentes de usuario, carga del sistema y tráfico de red. Estos autores llegaron a la conclusión de que la precisión era similar en ambos para 150, 200, 250, y 1000 ms, a excepción de las condiciones en las que la carga del sistema era elevada. Los resultados de Schmidt (2001) también situaron en torno a los 100 ms el umbral a partir del que Authorware deja de ser confiable.

Más adelante, Reimers y Stewart (2007) analizaron la capacidad de Flash para medir de forma precisa tiempos de respuesta en experimentación online y determinaron que las medidas realizadas mediante Flash en condiciones de laboratorio sufrían un retardo de entre 10 y 40 ms respecto a las tomadas por una aplicación nativa en Linux (Stewart, 2006). Además, esta diferencia era entre 30 y 40 ms mayor en las pruebas realizadas fuera del laboratorio. Al año siguiente, los mismos investigadores estudiaron la versión móvil de Flash (Adobe Flash Lite) y encontraron que los tiempos de respuesta medidos eran significativamente mayores en la versión móvil (entre 60 y 80 ms) con respecto a la versión para PC. Asimismo, constataron que la distribución de estas diferencias varió entre los dispositivos empleados (Reimers y Stewart, 2008).

En el caso de JavaScript, tal y como se detalla en el estándar (Ecma, 2011), el tiempo se mide con una resolución de ms relativa al 1 de enero de 1970 UTC, por lo que en ningún caso podrá obtenerse una resolución mayor empleando la función de tiempo estándar, incluso cuando el código esté ejecutándose lo suficientemente rápido como para lograrlo. En cuanto a su exactitud y precisión, las funciones de tiempo de JavaScript delegan su trabajo en el sistema operativo, por lo que sus características varían en función de la plataforma de ejecución subyacente. Así, en algunos sistemas operativos es posible que los valores devueltos por la llamada a `Date` en JavaScript estén «cuantizados» en relación al mecanismo de temporización del sistema operativo (e.g. Internet Explorer sobre Microsoft Windows XP muestra valores «cuantizados» en 10–16 ms). Sin embargo , no ocurre lo mismo en Mac OS X, como demostraron (Neath y cols., 2011). Estos autores comprobaron diversas tecnologías web (Flash, Java y JavaScript) en el agente de usuario Safari sobre dos equipos Apple (PowerMac 4.2 e iMac 8.1) y hallaron que en el

caso de Flash la DT del tiempo medido se situaba entre 5,338 y 11,232 ms, en Java entre 5,803 y 7,596 ms, mientras variaba entre 5,857 y 6,108 ms durante los test en JavaScript. Los resultados de estas tres tecnologías web estuvieron lejos de los obtenidos por estos mismos autores empleando el Psychophysics Toolbox (un conjunto de funciones libres para Matlab y GNU/Octave diseñado para la investigación en visión, ver Brainard, 1997), cuya DT varió entre 2,666 y 2,732 ms en las diferentes pruebas.

En definitiva, tal y como se resume en la tabla 2.9, los problemas aquí presentados apuntan a la posible falta de mecanismos adecuados para garantizar la exactitud y precisión en la temporización de animaciones multimedia y el registro de la interacción del usuario como factores principales que impiden el adecuado desarrollo, despliegue y ejecución de este tipo de aplicaciones mediante tecnologías web. Dado que ambos factores están fuertemente relacionados con el grado de exactitud y precisión de los mecanismos de temporización disponibles, tanto a nivel del sistema operativo como a nivel de hardware, parece razonable creer que las recientes mejoras a este respecto pueden trasladarse al entorno web. El objetivo principal de esta tesis será confirmar o descartar esta hipótesis.

2.6 Conclusiones

A lo largo del presente capítulo hemos podido comprobar cómo existe un gran número de elementos involucrados en la presentación de contenido audiovisual y el registro de la interacción del usuario de manera exacta y precisa por parte de una aplicación web. La revisión de la literatura en torno a cada uno de ellos ha puesto de manifiesto que todavía quedan interrogantes por responder en esta área.

En lo referente al usuario, a pesar de que la percepción consciente de instantaneidad se sitúe en torno a los 0,1 s, muchos paradigmas experimentales requieren ir más allá. Tal es el caso, por ejemplo, del Test de Asociación Implícita (*Implicit Association Test*, IAT, ver Greenwald, McGhee, y Schwartz, 1998), el *priming* (Fazio, Sanbonmatsu, Powell, y Kardes, 1986; Neely, 1977; Ratcliff y McKoon, 1978), las tareas *Go/NoGo* (Nosek y Banaji, 2001), o los experimentos acerca de la «unión temporal» (*temporal binding*) empleando el reloj de Libet (Libet y cols., 1983), entre muchos otros.

Tecnología	Limitaciones
GIF89a	Resolución en centésimas de segundo.
	Animaciones no interactivas.
	Diferencias de comportamiento entre agentes de usuario.
Java	Requiere la instalación de complementos en los agentes de usuario.
	Alta resolución, pero exactitud y precisión medias.
	Actualmente en desuso (no portado a muchas plataformas móviles).
Flash	Requiere instalación de complementos no disponibles en todas las plataformas.
	Exigente en capacidad de cómputo.
	Baja exactitud en entornos móviles.
	Actualmente en declive (escaso soporte móvil).
Silverlight	Requiere instalación de complementos no disponibles en todas las plataformas.
	Muy baja base de instalación.
SVG y SMIL	No hay datos sobre la exactitud de su temporización.
HTML5	Estándar todavía en fase de definición.
	Diferentes implementaciones en los distintos agentes de usuario web.
CSS	No hay datos sobre la exactitud de su temporización.
JavaScript	Resolución cuantizada en algunos sistemas operativos.
	Gestión de eventos no determinista.
	Limitaciones de diseño en el estándar ECMAScript.
Google Native Client	En fase de definición y desarrollo.
	No hay datos sobre su rendimiento en las diferentes plataformas soportadas.
	Problemas de seguridad derivados de la ejecución de código nativo desde la Web.
Tecnologías obsoletas	Requieren la instalación de complementos no disponibles en todas las plataformas.
(ActiveX, Authorware, etc.)	En desuso actualmente.

Tabla 2.9: Resumen de las limitaciones de las principales tecnologías web.

El equipamiento hardware empleado tiene también una fuerte influencia en la exactitud y precisión del sistema. Las desventajas en este sentido de la Web sobre el laboratorio son evidentes. La enorme variabilidad de dispositivos desde los que se utilizan las aplicaciones web imposibilita su correcta calibración. Los dispositivos de entrada introducen retardos variables, difíciles de estimar con pocas mediciones.

Las pantallas se comportan de forma muy diferente en función de la tecnología subyacente, la calidad del equipamiento, las condiciones externas o las opciones de configuración. Sin embargo, son varios los autores que afirman que todas estas fuentes de error pueden compensarse si se toman las suficientes mediciones y se procesan estadísticamente (Damian, 2010; K. Forster y Forster, 2003; Schneider y cols., 2002; Ulrich y Giray, 1989).

La mayoría del software especializado emplea mecanismos de temporización adecuados (i. e., `QueryPerformanceCounter` y Multimedia Timers en el caso de Windows, `gettimeofday` y temporizadores POSIX en el caso de sistemas compatibles con UNIX) y sus desarrolladores aseguran que su exactitud y precisión es elevada. No obstante, se echa en falta más evidencia experimental de la adecuación de estas aplicaciones a los altos requisitos temporales por parte de investigadores ajenos a su proceso de desarrollo, así como equivalentes implementados mediante tecnologías web estándar.

Por último, la principal conclusión que se extrae de la revisión bibliográfica en torno a la exactitud y precisión de las diferentes tecnologías web es que es preciso actualizar muchos de los datos que se dan todavía por ciertos aún teniendo más de 10 años de antigüedad. Asimismo, la gran mayoría de las tecnologías más recientes (v. gr., HTML5 Canvas, WebGL, CSS Animations) no han sido todavía analizadas a través de procedimientos similares a los estudios anteriores, por lo que se desconoce su grado de adecuación a los requisitos temporales.

Por lo tanto, a partir de este punto de partida establecemos un plan de trabajo destinado a demostrar que la Web es una plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario, según el esquema mostrado en la figura 2.6.

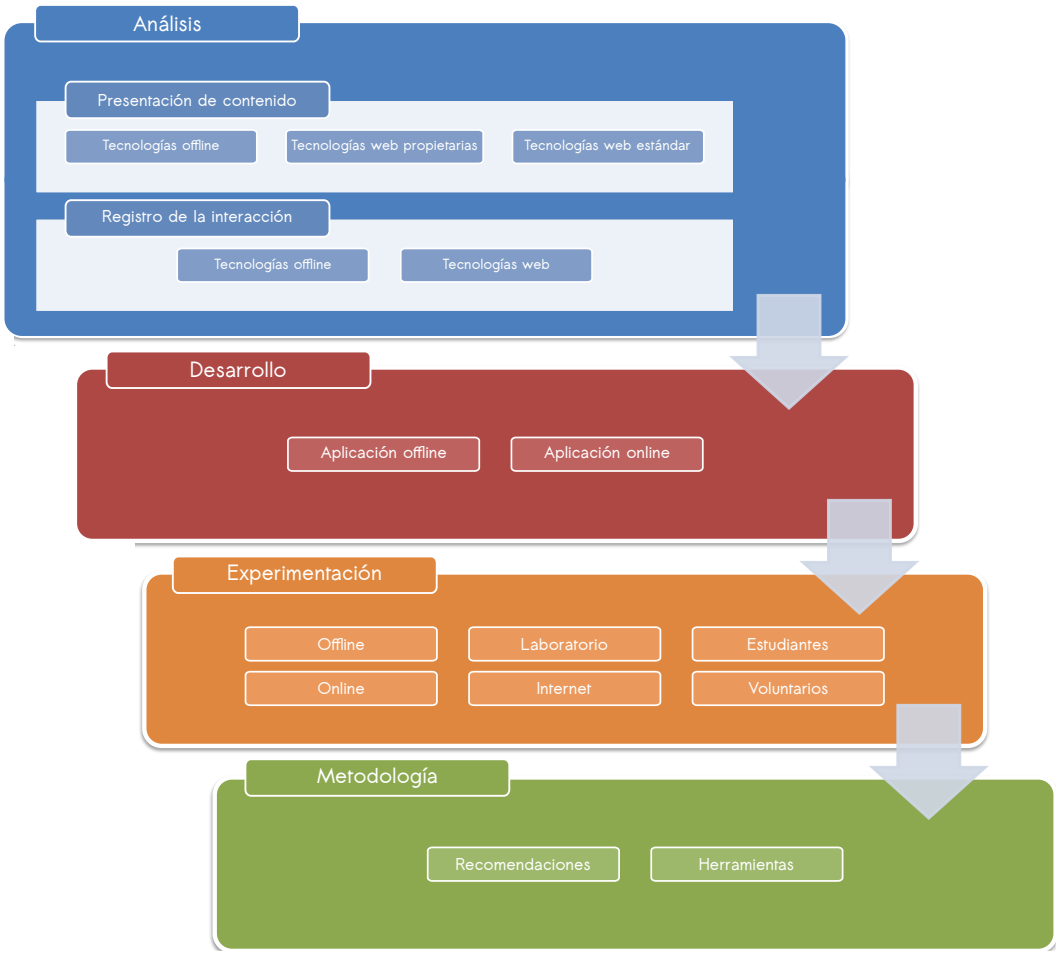

Figura 2.6: Plan de trabajo para demostrar la hipótesis de partida.

"Given enough eyeballs, all bugs are shallow"

Eric S. Raymond

CAPÍTULO

3

Análisis de tecnologías para la presentación exacta y precisa de contenidos audiovisuales

En el presente capítulo analizamos la exactitud y precisión de distintas tecnologías a la hora de presentar contenidos audiovisuales. Comenzamos fijando nuestra atención en la presentación del contenido visual y posteriormente abordamos el estudio de la presentación de contenido auditivo. Tal y como hemos avanzado en el capítulo anterior, el objetivo de estos análisis es el de disponer de un conjunto de datos empíricos suficientes para extraer conclusiones sobre cada tecnología. Estos datos se obtienen mediante sistemas externos a los evaluados, para evitar que la propia medición pudiera influir en el rendimiento de la tecnología evaluada.

El equipamiento, la metodología y el procedimiento de medición han sido elegidos cuidadosamente para facilitar la comparación de los resultados obtenidos en cada una de las pruebas, si bien en algunos casos concretos pueden encontrarse ligeras diferencias motivadas por detalles de implementación. Gracias al análisis de

estos datos es posible estimar el grado de exactitud y precisión de cada tecnología y, por lo tanto, seleccionar aquella que más se ajuste al desarrollo de una aplicación web con exigentes requisitos temporales.

3.1 Análisis del contenido visual

3.1.1 Metodología y equipamiento

Lamentablemente, los extraordinarios avances que se han dado en las últimas décadas en los ordenadores personales en cuestión de frecuencia de reloj no han traído consigo un incremento en su exactitud y precisión temporal.

Teniendo en cuenta los posibles retardos que se pueden dar a la hora de presentar contenido visual mediante un ordenador personal, el propio registro por parte del software evaluado no aporta las suficientes garantías como para asegurar que una presentación programada para un determinado momento y con una duración concreta se haya mostrado sin ningún retraso y con la duración apropiada. Por esta razón, resulta necesario el uso de sistemas externos que puedan registrar mediante sus sensores el momento preciso en el que el contenido comienza a mostrarse (*onset time*) y en el que deja de mostrarse (*offset time*). A pesar de que existen explicaciones de cómo obtener este tipo de sistemas de medición en la literatura (Schmidt, 2001), las pruebas detalladas a continuación se han llevado a cabo mediante el uso del *Black Box Toolkit* (BBTK) como sistema de medición externo para tratar de evitar problemas provocados por posibles errores en el ensamblaje de estos dispositivos o en el software encargado de la comunicación con ellos (Plant y cols., 2004).

El BBTK fue diseñado para comprobar un paradigma experimental *in situ* y sin ninguna modificación, a través de un «humano programable» capaz de detectar estímulos audiovisuales y generar respuestas con extrema exactitud y precisión. Conceptualmente ofrece una funcionalidad similar a un generador de señales digitales de 4 canales y un osciloscopio digital de 4 canales. Su tasa de muestreo es superior a 1 ms empleando una aplicación de Microsoft Windows con prioridad de tiempo real ejecutándose en un ordenador personal estándar. En estas circunstancias, el sistemas operativo otorga a la aplicación todos los recursos disponibles

y el ordenador no puede emplearse para ninguna otra tarea. La conexión desde el BBTK al ordenador se hace a través de un puerto IEEE 1284 (puerto paralelo) en modo EPP (*Enhanced Parallel Port*) o ECP (*Extended Capability Port*), lo que proporciona un ancho de banda de más de 2 MB/s. La latencia de comunicaciones entre el BBTK y el puerto IEEE 1284 es inferior a 100 ns.

En la figura 3.1 puede observarse la parte frontal del BBTK. Las líneas 1 y 2 son líneas de entrada digitales alimentadas con +5 v, pensadas para recibir información de un dispositivo externo (v. gr., un ratón, un teclado) al que se le ha acoplado un cable alimentado que detecta cuándo se ha activado uno de sus actuadores y envía la señal al BBTK mediante un conector *mini-jack* de 3,5 mm macho. Las líneas 3 y 4 son similares a las líneas 1 y 2, puesto que también son líneas digitales de entrada, pero vienen equipadas con sendos potenciómetros para ajustar la sensibilidad de los sensores conectados a ellas mediante un conector *mini-jack* de 2,5 mm macho. El escenario típico de empleo de estas líneas es la conexión de dos fotosensores y su posterior ajuste en función de las condiciones de luminosidad del ambiente y del dispositivo bajo medición (v. gr., el valor del brillo en un monitor CRT). Las líneas 5, 6, 7 y 8 son de salida. Las dos primeras (5 y 6) tienen un conector *mini-jack* de 3,5 mm hembra, mientras que el de las dos segundas (7 y 8) es un *mini-jack* de 2,5 mm hembra. Todas ellas pueden emplearse para conectar el generador de tonos digitales de BBTK, o cualquier otro dispositivo TTL (*Transistor–Transistor Logic*) que precise ser activado.

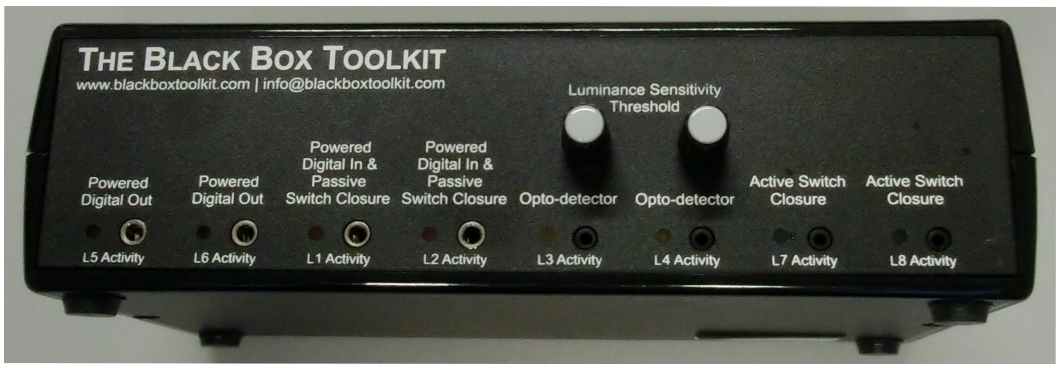

Figura 3.1: *The Black Box Toolkit* (Plant y cols., 2004), conectores delanteros.

En su parte posterior el BBTK dispone del mencionado puerto IEEE 1294 DB-25 macho para la conexión con el PC, un conector DE-9 hembra para emplearlo

con la tableta de botones de respuesta de BBTK, y un conector para la entrada de alimentación de 7,5–9 v y 250 mA de corriente continua.

Todo el equipamiento tiene una latencia muy inferior a la máxima admisible. La tasa de muestreo de las 4 líneas de entrada está por encima de las 50.000 muestras / s. La latencia de las entradas y salidas –fotosensores incluidos– con el puerto paralelo es inferior a los 100 ns. El generador de tonos tiene una latencia inferior a los 50 μs entre el puerto paralelo y lo detectado por un micrófono externo.

En lo que respecta al hardware en el que se prueban las diferentes tecnologías, se ha decidido emplear un ordenador Apple con el objetivo de comparar los diferentes sistemas operativos mayoritarios (Microsoft Windows, GNU/Linux y Apple Mac OS) sobre la misma plataforma física. Este ordenador es un Apple MacBook Pro A1211, con un procesador Intel Core 2 Duo T7600, una tarjeta gráfica ATI Radeon Mobility X1600, una tarjeta de sonido SigmaTel 9220 A1 High Definiton Audio, y un monitor LCD de 1440x900 píxeles de resolución nativa a 60 Hz. La adecuación de este equipamiento a las necesidades de exactitud y precisión en la presentación de contenido visual se probó a través de dos test diferentes. El primero de ellos fue la versión de 6 horas del `RefreshClockTest` de E-Prime, con resultado favorable. El segundo de ellos fue el test `timeByFrames` de PsychoPy, cuyo resultado también resultó favorable. En este test se recogieron los resultados mostrados en la figura 3.2, donde si bien se pueden observar fluctuaciones aisladas en la tasa de refresco, el valor central se sitúa en torno a $1000/60$ ms, como cabría esperar.

Sobre esta plataforma hardware se han instalado los siguientes sistemas operativos: 1) Microsoft Windows 7 Professional 32-bit edition con Service Pack 1; 2) Ubuntu Linux 10.04 LTS “Lucid Lynx” 32-bit con el núcleo Linux 2.6.33-29-realtime; y 3) Apple Mac OS X 10.7.3 “Lion”. Estas han sido las versiones elegidas por ser las versiones estables de cada uno de los sistemas operativos mencionados en el momento de realizarse las pruebas. Las tecnologías software analizadas han sido instaladas sobre los sistemas operativos soportados. Así, E-Prime o DMDX solamente se han instalado en Windows 7, mientras que PsychoPy se ha analizado tanto en Windows 7 como en Ubuntu Linux sobre un núcleo de tiempo real. En lo que respecta a tecnologías web, se han empleado los agentes de usuario compatibles con los 3 sistemas operativos mencionados: Google Chrome y Mozilla Firefox.

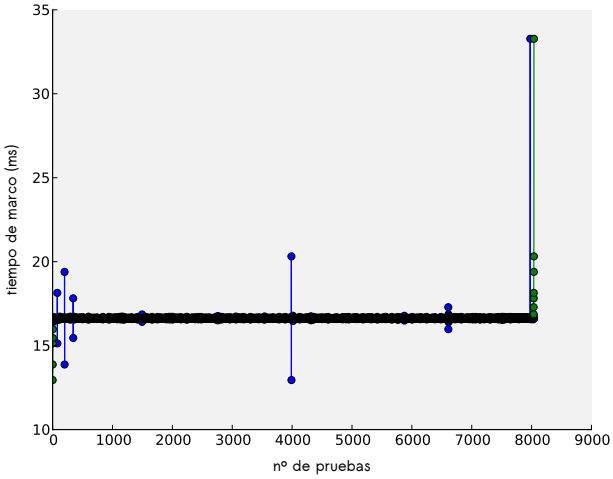

Figura 3.2: Resultados del test `timeByFrames` en PsychoPy.

Las versiones de estos agentes de usuario que han sido empleadas también corresponden con el momento en el que fueron realizados los test (Google Chrome 17 y Mozilla Firefox 10). Para algunas tecnologías web (GIF89a, Java, Adobe Flash o Microsoft Silverlight) también se ha utilizado Internet Explorer 9 sobre Microsoft Windows 7.

3.1.2 Tecnologías offline

En esta sección se comprueba la precisión y exactitud de algunas de las alternativas más populares de software experimental para la presentación de estímulos visuales (E-Prime, DMDX, PsychoPy), tratando de replicar las condiciones en las que estos programas se ejecutan habitualmente (monitores con baja tasa de refresco, sistemas operativos que no son de tiempo real). Para ello, se analizan las discrepancias encontradas entre los tiempos de presentación definidos por el investigador en la configuración del software experimental y los tiempos de activación (*onset*) y desactivación (*offset*) de los contenidos visuales detectados por un fotosensor conectado a un sistema de medición externo independiente (BBTK, ver Plant y cols., 2004).

3.1.2.1 Descripción de las tecnologías analizadas

Los programas de experimentación especializados en una presentación precisa y exacta de contenido audiovisual estudiados son los siguientes: E-Prime 2.0.8.90 (Schneider y cols., 2002), DMDX 4.0.4.8 (K. Forster y Forster, 2003) y PsychoPy 1.64.00 (Peirce, 2007) sobre Microsoft Windows 7 32-bit Professional edition; y PsychoPy 1.64.00 sobre Ubuntu Linux 10.04 LTS con un núcleo Linux 2.6.33-29-realtime.

Como se ha comentado con anterioridad, estos programas fueron ejecutados sobre un portátil Apple MacBook Pro con un procesador Intel Core 2 Duo T7600 y una tarjeta gráfica ATI Radeon Mobility X1600. La resolución nativa de su monitor LCD es de 1440x900 píxeles a 60 Hz.

Para realizar las mediciones se emplearon los fotosensores del BBTK, capaces de enviar los cambios detectados al puerto paralelo en menos de 100 ns. Esta herramienta ha sido específicamente diseñada por especialistas para realizar este tipo de mediciones. Las muestras recogidas por los fotosensores (1 muestra cada 20 μs) fueron transmitidas en tiempo real con una latencia inferior a los 100 ns a un ordenador auxiliar (AMD Sempron 2200 con Microsoft Windows XP 32-bit Professional edition) y almacenadas para su posterior análisis. En la figura 3.3 se muestra un esquema de conexión del equipamiento descrito.

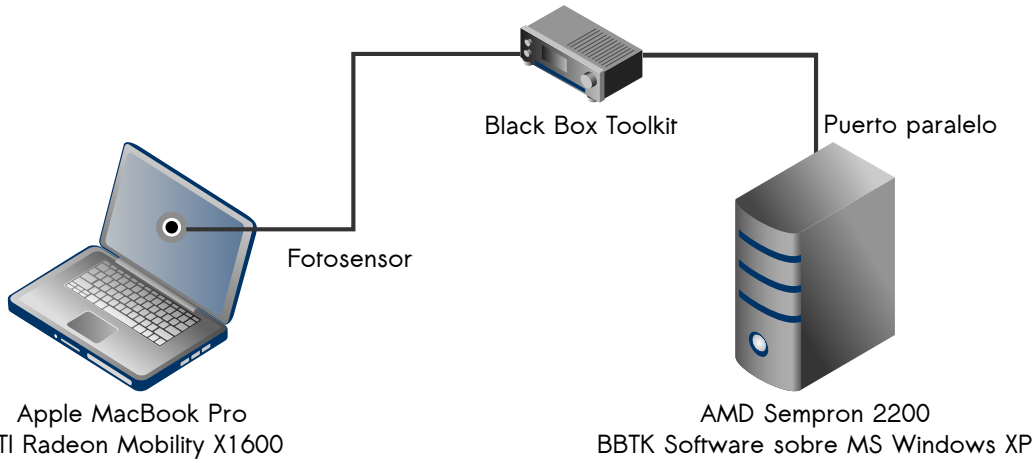

Figura 3.3: Elementos de prueba y medición en el análisis del contenido visual a través del *Black Box Toolkit*.

3.1.2.2 **Procedimiento**

Para cada una de las combinaciones de software y sistema operativo anteriormente mencionadas se prepararon varias animaciones a pantalla completa con transiciones no graduales de negro a blanco repetidas en las que se varió la duración de cada marco, probándose las duraciones de 1000, 500, 200, 100, 50 y 16,667 ms (60, 30, 12, 6, 3 y 1 *ticks* a 60 Hz, respectivamente). Para cada una de estas duraciones se registraron 5 series independientes de 60 s de duración cada una (ver figura 3.4). Para calcular el número de marcos perdidos se empleó el conjunto total de medidas para cada condición (i.e., aproximadamente 300, 600, 1500, 3000, 6000 y 18.000 para los intervalos de 1000, 500, 200, 100, 50 y 16,666 ms respectivamente). Para comparar todas las condiciones se emplearon únicamente las 50 primeras medidas de cada test, sumando un total de 250 medidas por cada duración en cada combinación de software y sistema operativo.

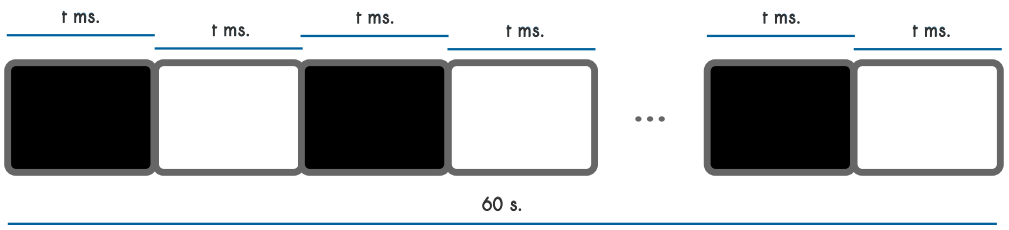

Figura 3.4: Transiciones no graduales de negro a blanco repetidas empleadas en las pruebas para t = 1000, 500, 200, 100, 50 y 16,667 ms.

Las causas que motivan este procedimiento son las siguientes: 1) el uso de animaciones compuestas de transiciones no graduales de negro a blanco son habituales en este tipo de estudios (Keller y cols., 2009; Reimers y Stewart, 2007; Schmidt, 2001) y facilitan la detección de los cambios en la pantalla cuando se emplean fotosensores digitales; 2) el decremento gradual en la duración del intervalo de transición desde 1 s al menor intervalo configurable (i. e., 1 *tick*, 16,667 ms a 60 Hz) va destinado a detectar umbrales a partir de los que la tecnología empleada deja de comportarse como es requerido; 3) la limitación de la duración de cada serie de medición a 60 s responde a una restricción en el software de captura de datos del BBTK, para el que resulta incapaz almacenar en disco más de 64 s cuando se

configura para emplear la máxima tasa de muestreo posible; y 4) la repetición de cada prueba a lo largo de 5 series responde, como ya hemos comentado, tanto a criterios estadísticos (mayor muestra, menor peso de posibles valores anómalos) y temporales (evaluar la posible degradación a lo largo del tiempo de la tecnología estudiada).

Para la preparación de las pruebas utilizando E-Prime, los intervalos temporales han sido ajustados en ms y no en función de la duración de un *tick*, pues esta es la manera de hacerlo en este software. Además de esto, es importante activar la sincronización con la señal `V-SYNC` para evitar los problemas derivados de la falta de sincronismo con el refresco de la pantalla. Tras analizar los resultados de unos test preliminares con esta configuración, llegamos a la conclusión de que la manera de maximizar la exactitud y precisión de la presentación empleada en E-Prime es emplear el modo de temporización *Event Timing Mode* de E-Prime, activar la sincronización con la señal `V-SYNC` tanto para el inicio de la presentación (*onset*) como para su finalización (*offset*) y sustraer la duración de un *tick* de la duración total de cada intervalo (v. gr., 1000 ms - 16,667 ms = 983,333 ms). Esta configuración está en línea con la aproximación sugerida en la documentación de E-Prime de restar 10 ms a la duración total (Schneider y cols., 2002). Sin embargo, en lugar de sustraer un valor arbitrario, estimamos más oportuno que este valor esté en función de la tasa de refresco del monitor y del tiempo de preparación del siguiente elemento a visualizar (dada la simplicidad de la animación propuesta, no consideramos la influencia de este segundo factor).

Los distintos modos de temporización de E-Prime se emplean para indicar si, en el caso de que se produjera un retraso, se prefiere acumular ese retraso al final del intervalo sin reducir su duración (*Event Timing Mode*), o si es preferible que los cambios se produzcan en los tiempos establecidos aunque eso suponga que una presentación dure menos que el intervalo fijado (*Cumulative Timing Mode*). E-Prime proporciona la opción de habilitar un modo de temporización personalizado (*Custom Timing Mode*) basado en una fórmula numérica que puede emplear los diferentes registros temporales que se obtienen durante la ejecución de un experimento en E-Prime (v. gr., `Probe.CustomOffsetTime = Probe.CustomOnsetTime` + 100). En la figura 3.5 puede verse el efecto causado por elegir el *Event Timing*

Mode y el *Cumulative Timing Mode* cuando se producen retardos en la presentación.

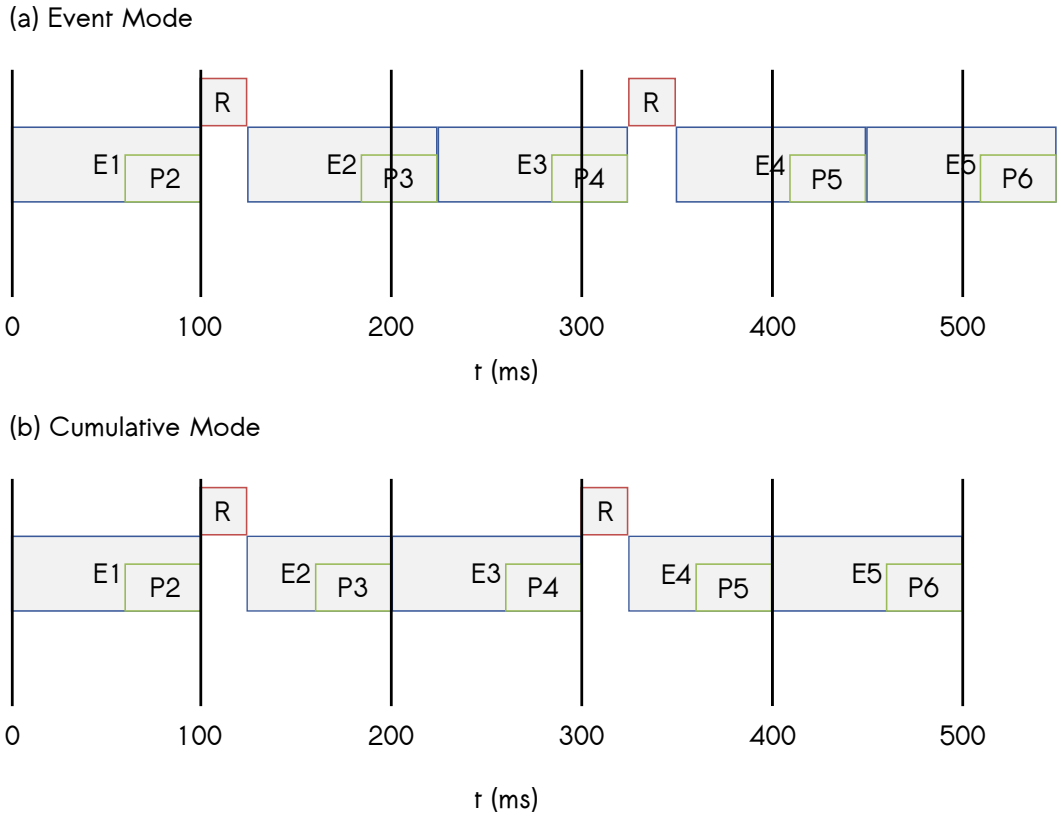

Figura 3.5: Modos de temporización en E-Prime y su efecto en presentaciones con retardos (E_n: estímulos, P_n: tiempos de preparación, R: retardos). Adaptado de Schneider y cols. (2002).

La configuración de las pruebas realizadas en DMDX se define usando las directivas de DMASTR (DisplayMaster). Dado que el objetivo es minimizar los tiempos de preparación de cada pantalla, la animación se ha programado evitando el uso de contadores y saltos condicionales, desenrollando el bucle principal que define cada transición de negro a blanco. En el listado 3.1 puede verse un ejemplo de esta configuración.

Como se puede observar, el fichero de configuración comienza con una cabecera encerrada entre las etiquetas `ep` (*extended parameters*) y `eop` (*end of parameters*). Dentro de esta cabecera tenemos las siguientes etiquetas: 1) `azk` (*Azkii*

```
<ep> <azk> <rcot> <n 2> <cr> <d 0> <fd 57>
<nfb> <vm 640,480,8,60> <id keyboard> <t 500> <eop>
0 "Press space bar to start";
^1 * <bc 255255255> /!;
^2 * <bc 0> /!;
^3 * <bc 255255255> /!;
[... 58 lineas omitidas]
^62 * <bc 0> /!;
^63 * <bc 255255255> /!;
^64 * <bc 0> /!;
0 "End";
```

Listado 3.1: Configuración de la transición no gradual de negro a blanco para DMDX.

responses keyword), empleada para definir el formato de fichero del registro temporal como AZK, es decir, con registros de 32 bits y hasta 4,2 millardos de ítems; 2) `rcot` (*record clock on time*), empleada para registrar el tiempo inicial del registro; 3) `n` (*number of items*), empleada para indicar el número de respuestas posibles; 4) `cr` (*continuous run*), empleada para ejecutar el experimento en modo de ejecución continua, sin pausas entre ítems; 5) `d` (*delay*), empleada para indicar el retardo en *ticks* entre ítems; 6) `fd` (*frame duration*), empleada para indicar la duración por defecto en *ticks* de un ítem (obsérvese que dado que el tiempo mínimo de preparación entre ítems es de 3 *ticks*, se ha fijado a 57 *ticks* para que cada ítem sea mostrado durante 60 *ticks*, 1000 ms a 60 Hz); 7) `nfb` (*no feedback*), empleada para indicar que no se desea proporcionar *feedback* tras cada ítem; 8) vm (*video mode*), empleada para indicar el modo de vídeo para DirectX (ancho, alto, profundidad de color y tasa de refresco); 8) `id` (*input device*), empleada para indicar el dispositivo de entrada; y 9) `t` (*timeout*), empleada para indicar el tiempo en ms en el que se registrará la respuesta del usuario. Seguidamente hay un pseudoítem con el número 0, por lo que se trata de un texto informativo que requiere de la acción del usuario para proseguir. El resto de ítems están numerados con valores distintos de cero y precedidos de un acento circunflejo, lo que indica que cualquier respuesta es considerada correcta, incluso la ausencia de ella. El asterisco se emplea para iniciar el temporizador correspondiente a cada ítem. Finalmente, una etiqueta `bc` (*background color*) define el color de fondo de cada ítem (`0` para negro, `255255255`

para blanco). El fichero de configuración termina con otro pseudoítem informativo que señala el final del experimento.

En las configuraciones iniciales de las pruebas para DMDX se detectó que a pesar de que el retardo entre ítems fuera fijado a cero, existe un retardo mínimo de 3 *ticks* que ha de ser compensado. Sin embargo, esta solución no es válida para los intervalos inferiores a los 3 *ticks* de duración, puesto que no se pueden configurar valores de tiempo negativos. Tras una comunicación personal con K.I. Forster, uno de los autores de DMDX, nos confirmó que las secuencias rápidas de contenidos visuales deben mostrarse como marcos separados dentro de un mismo ítem. Esta recomendación permite solucionar este problema, pero puede ocasionar otros en los casos en los que el comienzo de un ítem está sincronizado con equipamiento externo, tales como generadores de tonos, mediciones externas de tiempos de respuesta, aparatos de ERP (*Evoked Response Potential*) o fMRI (*functional Magnetic Resonance Imaging*), entre otros.

En lo referente a PsychoPy, cada una de las pruebas se define mediante dos ficheros: un documento XML de configuración general y un documento OOXML (*Office Open XML*, creado con Microsoft Excel) con la lista de ítems. El documento de configuración general es convertido a un programa en Python antes de ser ejecutado, por lo que las pruebas también pueden definirse mediante este programa en Python y el documento OOXML con la lista de ítems (durante la inicialización, el programa en Python importa este fichero a través de la función `importTrialList`). La estructura del documento XML de configuración es la siguiente: 1) la raíz la constituye el elemento `PsychoPy2experiment`, en el que se define la versión y la codificación de caracteres empleada; 2) en el elemento `Settings` se definen los parámetros de configuración general (`Show mouse`, `Full-screen window`, `Save log file`, etc.); 3) en el elemento `Routines` se definen cada una de las rutinas que se emplearán en el experimento (rutina de inicio, rutina de ítem, rutina de finalización, etc.); y 4) en el elemento `Flow` se define el flujo de ejecución del experimento, haciendo referencia a las rutinas anteriormente definidas y creando bucles cuando es necesario. El listado 3.2 muestra el contenido parcial de un fichero de configuración en PsychoPy correspondiente al control del flujo de ejecución.

Tras las pruebas iniciales con PsychoPy, detectamos otro suceso no esperado: en cada transición de marco en las animaciones se introduce un marco en negro de un *tick* de duración. Esto provoca que la duración de los marcos en negro se vea incrementada en 2 *ticks* (uno antes del marco y otro después), mientras que la duración de los marcos en blanco es la esperada. Para compensar este hecho, las animaciones se configuraron para que la duración de los marcos en negro fuera 2 *ticks* inferior a la de los marcos en blanco (v. gr., $1000 - 2 \times 16{,}667 = 966{,}667$ ms). Para las animaciones con una duración de un *tick*, se configuró una animación únicamente con marcos blancos de un *tick* de duración, que fueron intercalados por sendos marcos en negro por PsychoPy.

```
<Flow>
  <Routine name="start"/>
  <LoopInitiator loopType="TrialHandler" name="loop">
    <Param name="endPoints" val="[0, 1]" valType="num"
      updates="None"/>
    <Param name="name" val="loop" valType="code"
      updates="None"/>
    <Param name="loopType" val="sequential" valType="str"
      updates="None"/>
    <Param name="nReps" val="320" valType="code"
      updates="None"/>
    <Param name="trialList"
      val="[{u'windowColor': u'black'}, {u'windowColor': u'white'}]"
      valType="str"
      updates="None"/>
    <Param name="trialListFile" val="trials.xlsx" valType="str"
      updates="None"/>
  </LoopInitiator>
  <Routine name="trial"/>
  <LoopTerminator name="loop"/>
  <Routine name="end"/>
</Flow>
```

Listado 3.2: Control del flujo de ejecución de un experimento en PsychoPy definido en XML.

3.1.2.3 **Resultados**

Teniendo en cuenta los detalles acerca de los dispositivos de visualización mencionados con anterioridad, se estimó que los errores en los tiempos de presentación se concentrarían en torno a los múltiplos del tiempo de *tick* (ver figura 3.6a). Sin embargo, tras analizar los datos se obtuvo una distribución de los errores de medición diferente, en la que los errores se concentran en torno a dos picos: uno ligeramente anterior al momento de sincronización vertical y otro ligeramente posterior. En la figura 3.6b se puede observar una representación ideal de los datos recogidos, mientras que la figura figura 3.6c muestra un ejemplo concreto de estos resultados (i. e., PsychoPy 1.64.00 sobre Linux 2.6.33-29-realtime con 1000 ms de intervalo).

Esta discrepancia entre los datos reales y los esperados es debida a que el tiempo de respuesta del monitor no es instantáneo, por lo que los tiempos de excitación y decaimiento al mostrar un cambio en la pantalla provocan que la duración de las pantallas en negro sea ligeramente superior a lo esperado, mientras que las pantallas en blanco tengan una duración ligeramente inferior a la esperada. Dado que esto no es un error atribuible al software utilizado sino una particularidad de la tecnología empleada para mostrar los estímulos visuales, se decidió no contabilizar los errores de temporización absolutos sino los marcos perdidos (*missed frames*), asignando cada error de temporización absoluto al marco más próximo, mediante la siguiente fórmula:

$$Marcos\,perdidos = \begin{cases} \left\lfloor ETM - \frac{tick}{2} \right\rfloor, ETM < 0 \\ \left\lfloor ETM + \frac{tick}{2} \right\rfloor, ETM \geq 0 \end{cases} \tag{3.1}$$

Donde ETM corresponde al error de temporización medido y un *tick* tiene el valor de 16,667 ms a 60 Hz.

En la tabla 3.1 se muestran los marcos perdidos totales correspondientes a las pruebas realizadas sobre el software descrito en los diferentes intervalos (desde 1000 hasta 16,667 ms), una vez realizados los ajustes mencionados en la sección anterior. Los valores positivos representan intervalos mayores que los configurados, mientras que los valores negativos corresponden a intervalos más cortos que los deseados. El valor cero indica que el intervalo tuvo la duración esperada.

Como ya se ha mencionado anteriormente, para realizar el análisis inferencial tomamos únicamente las primeras 50 muestras de cada condición. Los estadísti-

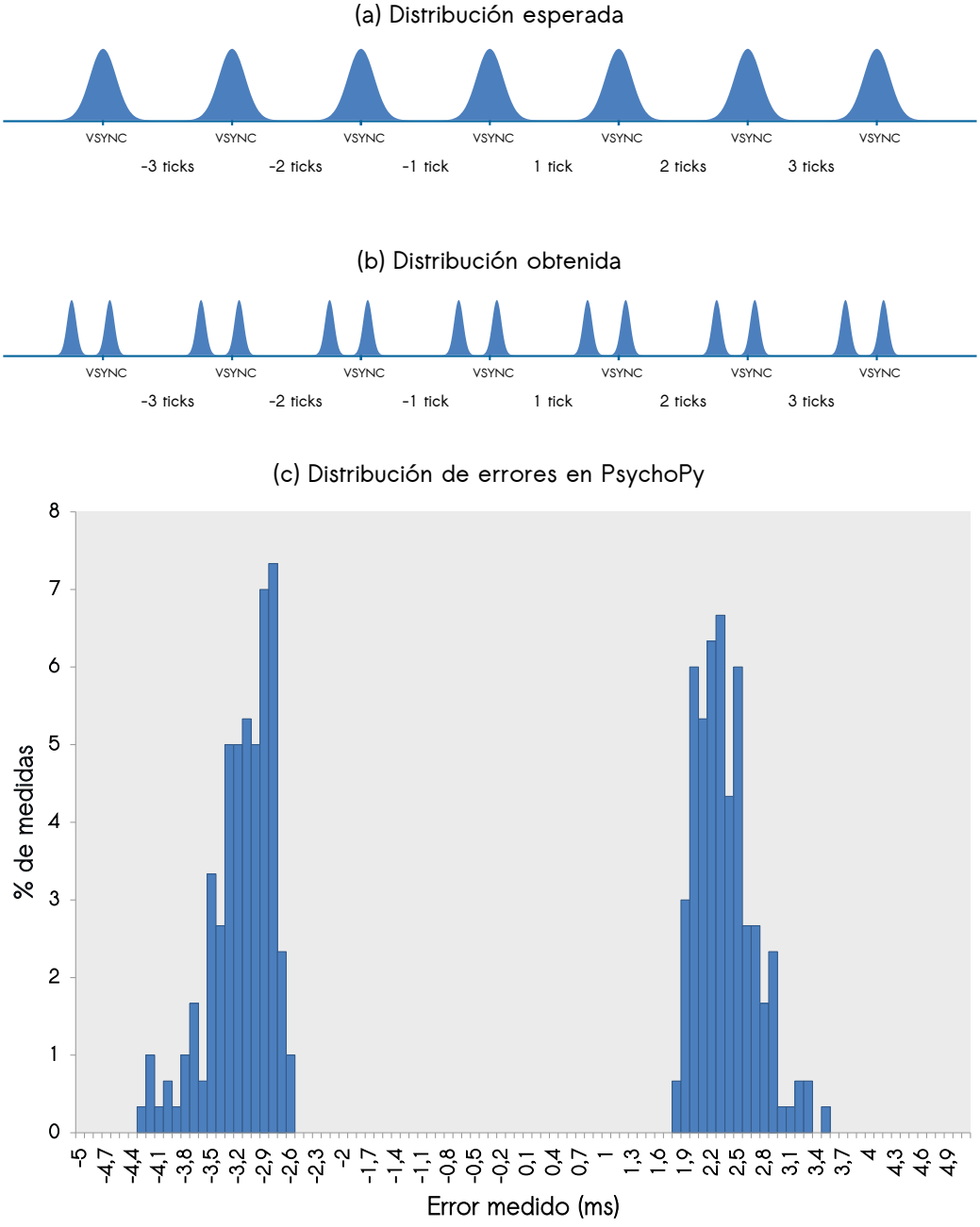

Figura 3.6: Distribución de los errores de temporización medidos. (a) Distribución esperada. (b) Distribución obtenida. (c) Distribución de los errores de temporización obtenidos al medir una animación de 1000 ms de intervalo en Psychopy 1.64.00 sobre Linux 2.6.33-29-realtime (en ms).

Software / SO	**Intervalo (ms)**	**Valor esperado (marcos)**	**Marcos perdidos**					
			-2	-1	0	1	2	≥ 3
DMDX	1000	300	0	0	300	0	0	0
/ Windows 7	500	600	0	0	600	0	0	0
	200	1500	0	0	1500	0	0	0
	100	3000	0	0	2999	1	0	0
	50	6000	0	0	6000	0	0	0
	16,667	18.000	0	0	17.998	1	0	0
			-2	-1	0	1	2	≥ 3
E-Prime	1000	300	0	0	299	1	0	0
/ Windows 7	500	600	0	0	600	0	0	0
	200	1500	0	0	300	0	0	0
	100	3000	0	0	3000	0	0	0
	50	6000	0	0	5997	1	1	0
	16,667	18.000	0	0	17.857	28	20	12
			-2	-1	0	1	2	≥ 3
PsychoPy	1000	300	0	0	300	0	0	0
/ Windows 7	500	600	0	0	600	0	0	0
	200	1500	0	2	1496	2	0	0
	100	3000	0	1179	634	1187	0	0
	50	6000	0	0	2572	2573	0	0
	16,667	18.000	0	0	5981	5980	0	0
			-2	-1	0	1	2	≥ 3
PsychoPy	1000	300	0	0	300	0	0	0
/ Linux RT	500	600	0	9	582	9	0	0
	200	1500	2	13	1447	13	0	0
	100	3000	2	25	2889	32	2	0
	50	6000	0	0	5113	575	0	0
	16,667	18.000	0	0	17.908	14	3	10

Tabla 3.1: Marcos perdidos por condición en las pruebas de tecnologías offline.

cos descriptivos de este conjunto de datos están resumidos en la tabla 3.2. Tras realizar un análisis factorial 4 (Software: DMDX, E-Prime, Psychopy-Windows, Psychopy-Linux) x 6 (Intervalo: 1000, 500, 200, 100, 50, 16,667 ms) x 5 (Serie: 1-

Software	Intervalo	N	M	ETM
DMDX /	1000	250	0	0
Windows 7	500	250	0	0
	200	250	0	0
	100	250	0	0
	50	250	0	0
	16,667	250	0	0
E-Prime /	1000	250	0	0
Windows 7	500	250	0	0
	200	250	0	0
	100	250	0	0
	50	250	0	0
	16,667	250	0,5	44
Psychopy /	1000	250	0	0
Windows 7	500	250	0	0
	200	250	0	0
	100	250	-0,01	61
	50	250	0,5	32
	16,667	250	0,5	32
Psychopy /	1000	250	0	0
Linux RT	500	250	0,00	14
	200	250	0,00	9
	100	250	0,00	0,11
	50	250	0,10	21
	16,667	250	0	0

Tabla 3.2: Estadísticos descriptivos de las 50 primeras medidas de cada una de las 5 series de medición (250 en total por cada condición).

5) de la varianza del número de marcos perdidos, encontramos efectos principales de Software, $F(3, 5880) = 106{,}31$, $p < 0{,}001$, e Intervalo, $F(5, 5880) = 62{,}63$, $p < 0{,}001$, así como en la interacción Software x Intervalo, $F(15, 5880) = 44{,}66$, $p < 0{,}001$. El resto de efectos principales e interacciones no fueron significativos. Los estadísticos descriptivos recogidos en la tabla 3.2 sugieren que el gran número de marcos perdidos en los intervalos más cortos al emplear PsychoPy –tanto en Windows como en GNU/Linux– pueden explicar la interacción Software x Intervalo. Para explorar con más detalle este efecto, realizamos un ANOVA 2 (SO: Windows, Linux) x 6 (Intervalo: 1000, 500, 200, 100, 50, 16,667 ms) de los marcos perdidos al emplear PsychoPy. Los resultados de este análisis mostraron un efecto principal de SO, $F(1, 2988) = 122{,}89$, $p < 0{,}001$, e Intervalo, $F(5, 2988) = 74{,}94$, $p < 0{,}001$. Asimismo, la interacción SO x Intervalo fue también estadísticamente significativa, $F(5, 2988) = 51{,}35$, $p < 0{,}001$. La figura 3.7 muestra el número de marcos perdidos al emplear PyschoPy sobre ambos sistemas operativos. Tal y como puede apreciarse, el rendimiento de PsychoPy es peor conforme va disminuyendo el intervalo de animación, pero este efecto es menos acuciado cuando PsychoPy se ejecuta sobre un núcleo Linux de tiempo real.

3.1.2.4 **Conclusiones**

A pesar de la aparente precisión y exactitud de los resultados obtenidos por los diferentes programas, conviene comentar algunos detalles al respecto.

En las pruebas realizadas sobre DMDX se ha obtenido un comportamiento excelente y resulta una opción muy recomendable para implementar experimentos que requieran una elevada precisión y exactitud en la presentación de contenido visual. Al compararlo con las otras alternativas evaluadas (i. e., E-Prime o PsychoPy), su principal inconveniente reside en la complejidad a la hora de diseñar un experimento. Sin embargo, a diferencia de ellas, DMDX define los tiempos de presentación en *ticks* y no en ms, una decisión de diseño que se ajusta más a lo que sucede a bajo nivel con el hardware de visualización y que explicita sus limitaciones, evitando que el investigador infiera que es posible configurar tiempos de presentación que no sean múltiplos de la duración de un *tick*. Relacionado con esto, es importante recordar que la presentación de elementos visuales con duraciones muy breves en DMDX debe realizarse obligatoriamente mediante la agregación

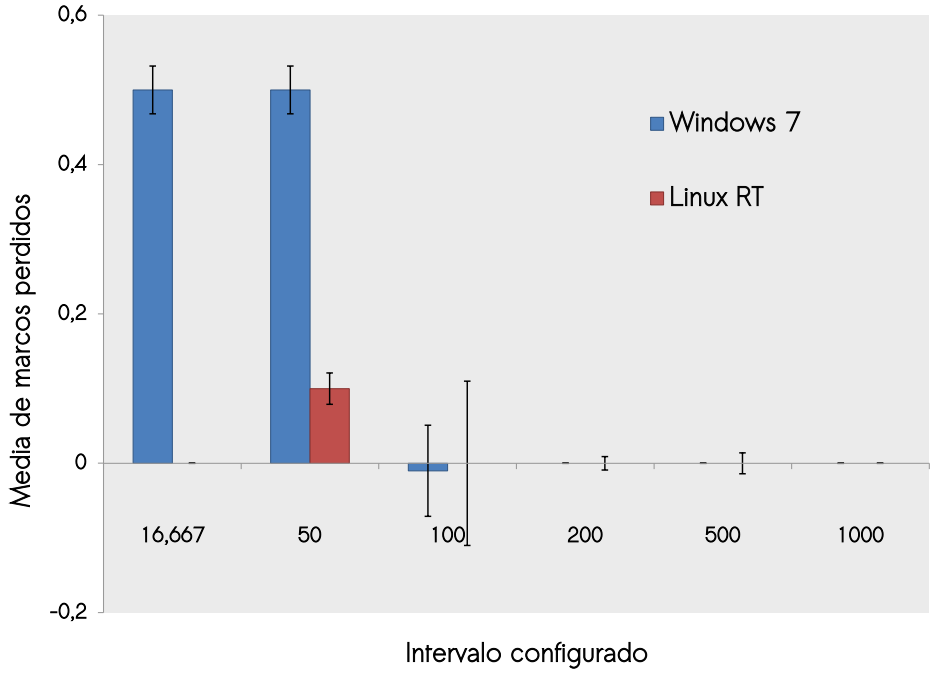

Figura 3.7: Marcos perdidos por cada condición empleando Psychopy sobre Windows 7 Professional 32-bit edition y Ubuntu 10.04 LTS con un núcleo Linux 2.6.33-29-realtime.

de marcos dentro de un mismo ítem, ya que de otra forma su duración se vería distorsionada por los retardos entre ítems.

En lo referente a E-Prime, cabe destacar su elevada exactitud y precisión temporal. Sin embargo, la posibilidad de configurar una duración arbitraria en ms para cada estímulo puede inducir al error. Por lo tanto, es recomendable que los investigadores que utilicen E-Prime comprueben sus experimentos con todas las opciones de registro activadas para detectar los retardos en la presentación de los contenidos audiovisuales, ya que pueden provocar un incremento considerable en la duración del contenido precedente, así como tratar de que las duraciones de cada presentación sean múltiplos de la duración de un *tick*.

PsychoPy comparte con E-Prime la facilidad de configurar experimentos y la posibilidad de definir tiempos arbitrarios para la duración de cada presentación, por

lo que pueden hacerse las mismas recomendaciones al respecto. Además de esto, conviene tener en cuenta el marco en negro que se intercala en cada transición a la hora de definir la duración de los intervalos, tal y como hemos descrito en la sección anterior. Asimismo, PsychoPy es capaz de ir más allá de lo que ofrece su interfaz gráfica de usuario a través de una API de desarrollo que puede ser empleada por programadores para ajustar al máximo el comportamiento de pruebas o experimentos, consiguiendo un mayor control sobre los mecanismos de temporización empleados. En general, PsychoPy es una alternativa con un buen desempeño y con las ventajas de ser software libre y multiplataforma. Gracias a esto, hemos podido evaluar la precisión y exactitud de PsychoPy tanto en un sistema operativo de propósito general como Microsoft Windows 7, como en un sistema operativo de tiempo real como Ubuntu con un núcleo Linux 2.6.33-29-realtime. Como se puede apreciar en los resultados mostrados en la sección anterior, el uso de un sistema operativo de tiempo real resultó beneficioso para las animaciones con intervalos de corta duración (i. e., 16,667, 50 y 100 ms), pero no supuso una ventaja para las animaciones con intervalos más largos.

En definitiva, consideramos que el software evaluado es apropiado para la implementación y despliegue de experimentos con altos requisitos de precisión y exactitud en la presentación de contenidos visuales, siempre y cuando sean tomadas en cuenta las consideraciones apuntadas en esta sección.

3.1.3 Tecnologías online clásicas

3.1.3.1 Descripción de las tecnologías analizadas

Las primeras versiones del estándar HTML no incorporaban las funcionalidades necesarias para desarrollar aplicaciones web y se limitaban a describir la sintaxis de documentos de hipertexto (Berners-Lee y Cailliau, 1990). Una vez que a mediados de la década de los 90 del siglo XX la Web se postuló como el servicio de Internet de referencia, usuarios y proveedores de servicios web encontraron serias limitaciones en este enfoque. A raíz de esto, diversos proveedores de software complementaron los agentes de usuario web con tecnologías en principio ajenas a la Web, tales como visualizadores de animaciones, reproductores multimedia o máquinas virtuales capaces de ejecutar programas completos desde el agente de

usuario. Dentro de esta generación de tecnologías podríamos enmarcar las animaciones hechas en GIF89a, las aplicaciones Flash, las *applets* en Java, o incluso las aplicaciones en Silverlight, que si bien son posteriores, tienen una filosofía muy similar a sus homólogas en Java o Flash. Todas ellas tienen en común que, a pesar de no estar definidas dentro del estándar HTML, han sido implementadas en la práctica totalidad de agentes de usuario mayoritarios, lo que ha provocado que un número amplio de aplicaciones web se hayan apoyado en ellas.

GIF89a es un formato gráfico estándar (*Graphics Interchange Format*) no diseñado para la Web, pero soportado por la gran mayoría de los agentes de usuario web debido a que su algoritmo de compresión LZW (*Lempel Ziv Welch*) ofrece la posibilidad de crear imágenes de tamaño muy reducido. Al igual que otros formatos gráficos como MNG (*Multiple-image Network Graphics*, ver Randers-Pehrson, 2001), GIF89a es capaz de definir animaciones. La principal ventaja de emplear una animación en este formato es que su visualización no corre a cuenta del entorno de ejecución en JavaScript proporcionado por el agente de usuario web, sino que es realizado de forma independiente. Sin embargo, esta ventaja se convierte en un inconveniente para los casos en los que se pretende controlar el estado de la animación desde un programa, ya que una animación en GIF89a comienza en el momento en el que se visualiza la imagen y no ofrece ningún mecanismo de comunicación para modificar su velocidad o contenido una vez iniciada. Además de esto, el estándar GIF89a solamente permite definir los intervalos temporales de la animación en centésimas de segundo en lugar de ms (CompuServe, 1990). Más grave aún resulta el hecho de que algunos agentes de usuario web hagan caso omiso de los intervalos configurados si éstos son demasiado exigentes (v. gr., en Microsoft Internet Explorer, por debajo de los 50 ms).

Java es tanto un lenguaje de programación como la plataforma de desarrollo y ejecución de las aplicaciones creadas con el mismo. El lenguaje de programación Java fue creado por James Gosling entre 1994 y 1995 (Gosling y McGilton, 1995). Su sintaxis es muy similar a la de C y C++, pero lo más novedoso es su enfoque "*write once, run anywhere*" (WORA) a través del que se pretende que la compilación de código Java a su código compilado propio (Java *bytecode*) pueda ejecutarse sobre cualquier plataforma soportada gracias a la existencia de máquinas virtuales

Java (*Java Virtual Machines*, JVM) compatibles. Su éxito en el ámbito del desarrollo de software está fuera de toda duda y está considerado uno de los lenguajes de programación más populares del mundo (Tiobe, 2012). En lo que respecta a la web, Java se ha empleado tanto para crear aplicaciones de servidor (a través de *servlets* o JSP, *Java Server Pages*) como para desplegar aplicaciones de cliente (a través de *applets*). La inclusión del código Java de un *applet* dentro de un documento se realizaba a través del elemento `applet` en versiones previas a HTML 4.01. A partir de esa revisión del estándar, el uso del elemento `object` es la forma recomendada de incluir un *applet* en un documento HTML. A pesar de su éxito en todo tipo de plataformas, Java está perdiendo paulatinamente soporte por parte de los desarrolladores de agentes de usuario web, por lo que cada vez menos sitios web confían en Java para extender la funcionalidad de HTML.

Adobe Flash es una tecnología de desarrollo de aplicaciones de Internet enriquecidas (*Rich Internet Applications*, RIA) gracias a sus capacidades para manipular diferentes tipos de contenido multimedia tales como texto, gráficos vectoriales y de mapa de bits, audio o vídeos, además de interactuar con dispositivos como el micrófono o la cámara de un equipo conectado a la Red. Estas aplicaciones pueden crearse por personas sin conocimientos de programación a través de herramientas de autor como Adobe Flash Professional o bien por programadores mediante el lenguaje ActionScript, uno de los dialectos de ECMAScript (recordemos que JavaScript es otro de los dialectos de ECMAScript, por lo que son muy similares). Una vez compiladas, las aplicaciones Flash se reproducen dentro de una máquina virtual conocida como Adobe Flash Player, instalable de forma gratuita como *plugin* en los agentes de usuario web mayoritarios de las principales plataformas de software. Esta arquitectura tiene indudables ventajas, como la independencia de las capas de software subyacentes, pero también ocasiona algunos problemas. El principal problema de Flash es que no está realmente integrado dentro de la aplicación web, sino embebido como un objeto externo. A más bajo nivel esto se traduce en grandes ineficiencias de rendimiento y consumo de energía debido al uso de complementos en los agentes de usuario en lugar de componentes nativos del mismo. Además de esto, en 2011 Adobe anunció que finalizaría su desarrollo de Flash para plataformas móviles y televisión, centrando sus esfuerzos en el estándar HTML5

(Winokur, 2011), por lo podría decirse que estamos ante una tecnología con un buen presente pero con un futuro poco prometedor.

Silverlight es, al igual que Flash, una tecnología de desarrollo de aplicaciones de Internet enriquecidas. La propuesta de Microsoft es muy similar a la de Adobe y, por lo tanto, también se apoya en una máquina virtual para ejecutar sus aplicaciones. Esta máquina virtual está disponible como complemento para los principales agentes de usuario web sobre Microsoft Windows y Apple Mac OS X. El proyecto Moonlight dentro de la plataforma Mono pretende cubrir esta funcionalidad para GNU/Linux, FreeBSD y otros sistemas operativos libres. La principal ventaja de Silverlight frente al resto de tecnologías mencionadas reside en su integración con el resto de tecnologías de desarrollo basadas en Microsoft .Net, lo que le proporciona una gran versatilidad a la hora de ser desplegada. Sin embargo, la falta de soporte más allá de Microsoft Windows y Apple Mac OS X, unida al gran índice de penetración en el mercado de una tecnología competidora como Adobe Flash, ha relegado su uso a entornos donde se emplean exclusivamente tecnologías Microsoft.

La consideración de estas tecnologías como clásicas responde a un criterio prospectivo. Si bien es cierto que algunas de estas tecnologías como Silverlight no son precisamente antiguas, sí lo son los planteamientos en los que se basan. El futuro de la Web orbita en torno al nuevo conjunto de estándares alrededor de HTML5 y todo apunta a que el lenguaje de programación de la Web será JavaScript y no Java, Flash (ActionScript) o Silverlight (Microsoft .Net). Como puede verse en la figura 3.8, solamente el 0,2 % de los sitios web emplea Java, el 0,3 % usa Silverlight y el 23,2 % Flash, frente al 92 % que se apoya en JavaScript para dotar de mayor funcionalidad a la Web (W3Techs, 2012).

Teniendo todo esto en cuenta, resulta conveniente realizar un análisis de estas tecnologías que permita comparar su precisión y exactitud con las tecnologías offline previamente estudiadas.

3.1.3.2 **Procedimiento**

El procedimiento seguido para el análisis de estas tecnologías es similar al utilizado en el análisis de las tecnologías offline ya explicado. Se prepararon transiciones no graduales de negro a blanco repetidas en las que se varió la duración

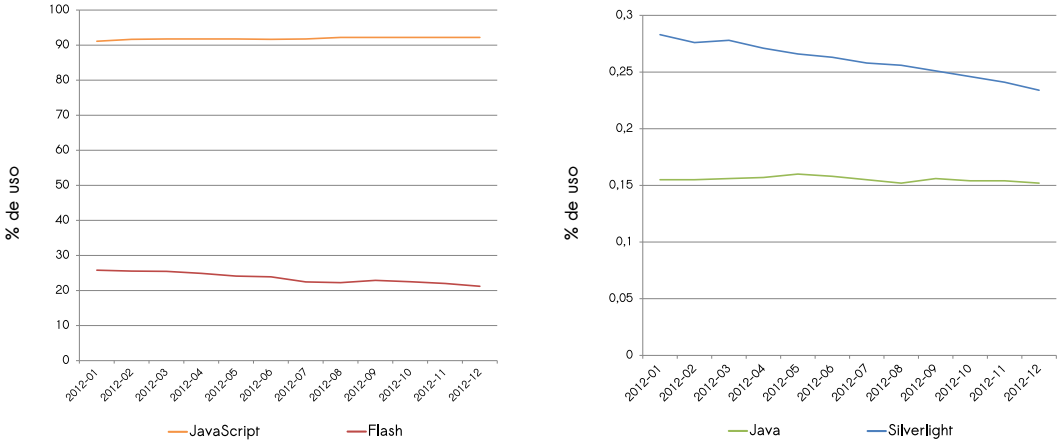

Figura 3.8: Evolución del uso de diferentes tecnologías web de cliente en Internet (W3Techs, 2012).

de cada marco, probándose las duraciones de 500, 100, 50 y 16,667 ms (i. e., 30, 6, 3 y 1 *ticks* a 60 Hz, respectivamente) en las tecnologías GIF89a, Flash, Java y Silverlight sobre los agentes de usuario Google Chrome 17, Mozilla Firefox 10 e Internet Explorer 9. Todas ellas fueron probadas sobre Microsoft Windows 7 SP1. Para cada uno de los intervalos se registraron 5 series independientes de 60 s de duración cada una en animaciones de 200 x 200 píxeles. De cada una de estas series se descartaron los 5 primeros y últimos segundos para evitar errores propios de la inicialización o la finalización, y se analizaron las primeras 100 muestras de cada serie, por lo que finalmente se obtuvieron 500 muestras por cada duración en cada combinación de tecnología web y agente de usuario.

Las razones de estas ligeras modificaciones frente al procedimiento de la sección anterior son las siguientes: 1) el número de intervalos evaluados ha sido reducido (500, 100, 50 y 16,667 ms frente a 1000, 500, 200, 100, 50 y 16,667 ms) por considerar que los intervalos excluidos no aportan hallazgos relevantes, bien por ser extremadamente largos (en el caso de 1000 ms), bien por ser muy próximos a otros intervalos ya evaluados (en el caso de 200 ms); 2) las animaciones se han configurado con un tamaño de 200 x 200 píxeles por tratarse de tecnologías embebidas en documentos web (no a pantalla completa) y ser una práctica habitual en este tipo de estudios (Schmidt, 2001); 3) los test se han limitado a un solo sistema operativo (Microsoft Windows 7 SP1) porque no todas estas tecnologías tienen

soporte oficial en sistemas libres como GNU/Linux; y 4) la selección de muestras (i. e., las 100 primeras muestras a partir de los 5 primeros segundos) tiene como objeto poder realizar comparaciones entre ellas mediante análisis estadísticos que requieran un número similar de observaciones, así como evitar posibles errores de inicialización.

Las pruebas correspondientes al formato GIF89a se han llevado a cabo mediante ficheros de 200 x 200 píxeles de tamaño a 72 ppp (puntos por pulgada) de resolución con una animación compuesta por dos marcos, uno negro y otro blanco, que se muestran durante un intervalo dado. Dado que a través de esta tecnología no es posible establecer los intervalos con una granularidad superior a la centésima de segundo, estos intervalos fueron configurados con los valores de 50, 10, 5 y 1 centésima de segundo, en cada una de las pruebas preparadas.

Para poner a prueba Adobe Flash se han utilizado animaciones en bucle a 60 marcos por segundo (FPS). Dado que la tasa de refresco es constante, se ha variado el número de marcos en negro o en blanco para hacer coincidir los cambios en la animación con los intervalos requeridos. Así, a la animación de 16,667 ms de intervalo, solamente le corresponden un marco negro y otro blanco, mientras que la animación de 500 ms de intervalo tiene 30 marcos negros seguidos de 30 marcos blancos. Las razones de haber elegido esta configuración para las pruebas de Adobe Flash se explican en detalle en la sección siguiente (*vide infra*).

En lo relativo a Java, las pruebas consisten en un *applet* que recibe un único parámetro en el que se define el intervalo de la animación en ms. Por lo tanto, se han preparado 4 documentos HTML en los que se ha incluido un *applet* con un tamaño de 200 x 200 píxeles y el valor apropiado para el intervalo de animación (500, 100, 50 y 16,667 ms). Este *applet* emplea el método `scheduleAtFixedRate` de la interfaz `ScheduledExecutorService`, que recibe como argumentos un hilo (interfaz `Runnable`), un retardo inicial, un periodo y la unidad de medida de estos dos últimos valores temporales. La configuración empleada ha sido la siguiente: 1) el hilo llama al método `repaint` que cambia de color el fondo del *applet*; 2) el retardo inicial se fija a 0; 3) el periodo se toma de los parámetros de invocación del *applet*; y 4) la unidad de tiempo se fija a ms (`TimeUnit.MILLISECONDS`). De esta forma, se produce un cambio de color de fondo cada periodo (cada llamada se lanza en el momento $retardo_{inicial} + N \times periodo$).

Las animaciones de las pruebas relativas a Silverlight fueron definidas declarativamente mediante XAML (*eXtensible Application Markup Language*), Gracias a la orientación hacia el contenido multimedia de este dialecto de XML, es posible definir animaciones como las requeridas en estas pruebas de forma declarativa, sin tener que preocuparse de la implementación a bajo nivel en .Net necesaria para llevarlas a cabo. De esta manera, se delega en el entorno de ejecución la tarea de implementar los mecanismos de temporización adecuados para cumplir con los requisitos de la animación. Así, en el documento XAML preparado para realizar la animación, un elemento `Rectangle` es animado una vez se ha cargado a través del elemento `ColorAnimationUsingKeyFrames` que tiene una duración de un segundo (`Duration="0:0:1.000"`) y repetirá por siempre (`RepeatBehavior="Forever"`) la inclusión de un marco en blanco de 500 ms de duración (`KeyTime="0:0:0.500"`), conformando la animación de 500 ms de intervalo requerida.

3.1.3.3 **Análisis previo para Flash**

Como se ha mencionado previamente, antes de proceder a la comparación de las distintas tecnologías web clásicas, se han evaluado las opciones de configuración y programación que ofrece Adobe Flash con el objetivo de seleccionar la mejor combinación. Los factores que se han puesto a prueba son dos: el mecanismo de temporización y la tasa de marcos por segundo (FPS) de la animación en Flash.

Los mecanismos de temporización evaluados han sido los siguientes: 1) *loop*, delegando el control de la animación al parámetro `loop` del reproductor de películas Flash; 2) *no-loop*, reiniciando la animación a través de un salto al final del último marco (a través de la llamada `gotoAndPlay(1)`); 3) *setInterval*, mediante un temporizador periódico en ActionScript (a través de la llamada `setInterval`); 4) *polling*, comprobando activamente si el momento de cambiar de fondo ha llegado; y 5) *timer*, controlando la animación mediante un temporizador (objeto de clase `Timer`) y un gestor para el evento `TimerEvent.TIMER`.

Las tasas de marcos por segundo evaluadas han sido las siguientes: 1) 20 FPS, una tasa habitual en *banners* y aplicaciones sencillas en Flash (un cambio cada 50 ms); 2) 60 FPS, la tasa de refresco del monitor empleado (un cambio cada 16,667 ms); y 3) 100 FPS, por encima de la tasa de refresco del monitor empleado

(un cambio cada 10 ms). Todas estas pruebas se han llevado a cabo sobre Google Chrome 17 y Microsoft Windows 7 SP1 por tratarse del navegador más empleado sobre el sistema operativo más empleado. Además, el único intervalo evaluado ha sido el de 50 ms, por tratarse del intervalo a partir del cual más diferencias en rendimiento se han dado entre las pruebas realizadas previamente.

Los resultados de las pruebas relativas al rendimiento de Adobe Flash con animaciones de 50 ms de intervalo sobre Google Chrome 17 y Microsoft Windows 7 SP1 se muestran en la figura 3.9 en forma de marcos perdidos por cada una de las 5 series de test realizadas.

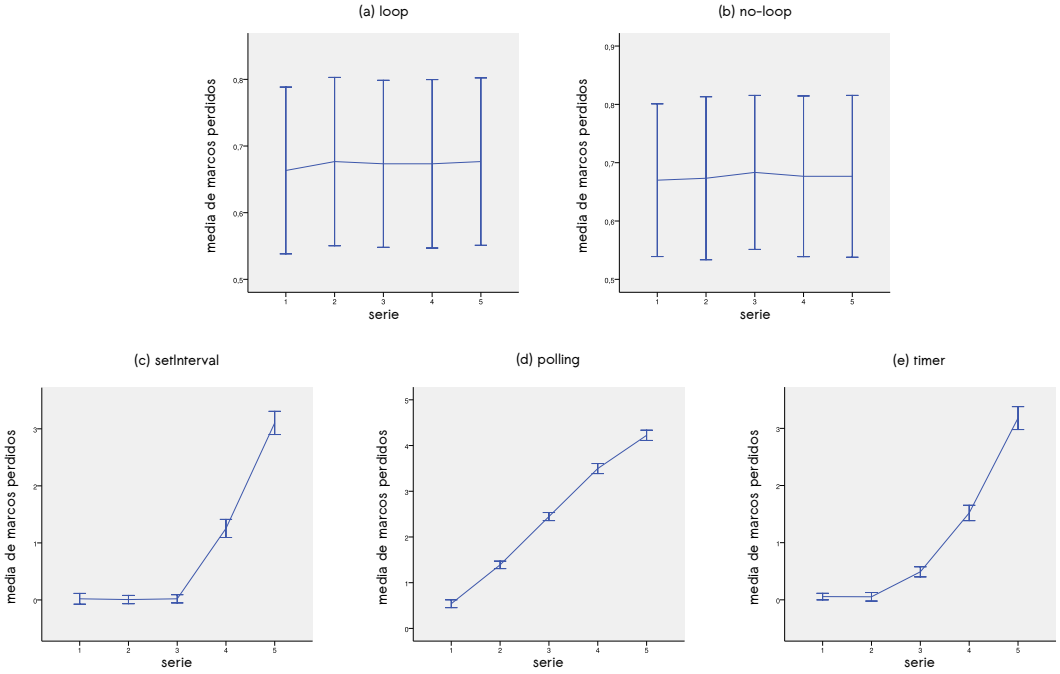

Figura 3.9: Número de marcos perdidos por serie para las animaciones en Flash con cada mecanismo de temporización: (a) *loop*, (b) *no-loop*, (c) *setInterval*, (d) *polling*, y (e) *timer*.

A simple vista puede observarse como los dos primeros mecanismos de temporización (*loop* y *no-loop*) permanecen estables a lo largo de las 5 series, mientras que el resto de mecanismos de temporización (*setInterval*, *polling* y *timer*) sufre una degradación en su rendimiento a medida que avanzan las series de pruebas. No obstante, conviene analizar estadísticamente todos los datos recogidos con vistas

a determinar qué factor o factores influyen en el número de marcos perdidos de esta tecnología. Así, se realizó un análisis de la varianza de los siguientes factores: Temporizador (*loop*, *no-loop*, *setInterval*, *polling*, *timer*) x FPS (20, 60, 100 FPS) x Serie (1 a 5) x Color (blanco, negro) en el que todos los resultados fueron significativos debido al alto número de muestras recogidas ($N = 7500$). Por este motivo, decidimos analizar el tamaño del efecto. En este sentido, la interacción más relevante es la de Temporizador x FPS x Serie, $F(32, 7499) = 118{,}232$, $p < 0{,}001$, $\eta_p^2 = 0{,}340$, lo que concuerda con el análisis preliminar: el rendimiento se ve afectado cuando el número de marcos por segundo es inferior (20 FPS) o superior (100 FPS) a la tasa de refresco del monitor (el efecto principal de FPS es significativo, $F(2, 7499) = 4201{,}554$, $p < 0{,}001$, $\eta_p^2 = 0{,}533$), y la degradación temporal que sufren los mecanismos de temporización más complejos (*setInterval*, *polling* y *timer*) provoca que la elección del mecanismo de temporización tenga un impacto significativo en el rendimiento final (el efecto principal de Temporizador es significativo, $F(4, 7499) = 2751{,}154$, $p < 0{,}001$, $\eta_p^2 = 0{,}600$). Aún sin tener en cuenta el efecto de degradación temporal a lo largo de las distintas series (el efecto principal de Serie también es significativo, $F(4, 7499) = 3390{,}185$, $p < 0{,}001$, $\eta_p^2 = 0{,}649$), los mecanismos de temporización evaluados pueden ordenarse de mejor a peor rendimiento de la siguiente manera: 1) *loop* (M: 0,67, DT: 1,087), 2) *no-loop* (M: 0,68, DT: 1,175), 3) *setInterval* (M: 0,88, DT: 1,658), 4) *timer* (M: 1,06, DT: 1,591), y 5) *polling* (M: 2,42, DT: 1,58). Por consiguiente, se empleará el mecanismo de temporización *loop* a 60 FPS en el resto de pruebas realizadas mediante Adobe Flash.

3.1.3.4 **Resultados**

Una vez descrito el procedimiento de cada una de las pruebas realizadas, analizamos los resultados de las mismas. El número de marcos perdidos en las pruebas realizadas mediante GIF89a ha sido el esperado. Esta tecnología apenas sufre retardos en la presentación de contenido visual cuando el intervalo de la animación se sitúa por encima de los 100 ms en los tres agentes de usuario evaluados (Google Chrome 17, Mozilla Firefox 10, Internet Explorer 9). A partir de ese umbral, el rendimiento cae alarmantemente, aunque de forma desigual, en los tres agentes de usuario. En el caso de Internet Explorer 9, cualquier intervalo por debajo de 100

ms es reinterpretado como un intervalo de 100 ms, por lo que las pruebas con intervalos de 50 ms arrojan retrasos de 50 ms, mientras que las pruebas de 16,667 ms sufren retrasos típicos en torno a los 85 ms. En el caso de Mozilla Firefox 10, las animaciones en GIF89a se muestran correctamente salvo para el caso de 16,667 ms de intervalo, donde se detectan retrasos similares a los de Internet Explorer, en torno a los 85 ms. Finalmente, en el caso de Google Chrome 17, los valores por debajo de los 100 ms son muy similares a los obtenidos en las pruebas con Mozilla Firefox 10, si bien la media de marcos perdidos es menor en todas las pruebas (ver tabla 3.3).

Intervalo (ms) / **Agente de usuario**	**500**		**100**		**50**		**10**	
	M	DT	M	DT	M	DT	M	DT
Google Chrome 17	0,06	0,342	0,01	0,891	0,01	1,065	5,51	1,024
Mozilla Firefox 10	0,24	1,379	0,29	1,055	0,17	0,956	5,79	1,176
Internet Explorer 9	0,06	0,266	0,02	0,922	3,02	0,923	5,51	1,072

Tabla 3.3: Estadísticos descriptivos del número de marcos perdidos para las animaciones con GIF89a.

Otro dato que muestran los resultados obtenidos es que esta tecnología es estable a lo largo del tiempo y no se aprecian variaciones en el número de marcos perdidos en función del número de serie evaluada (ver figura 3.10).

En lo que respecta a Adobe Flash, una vez descartadas en la sección anterior las configuraciones que peor rendimiento obtuvieron, los resultados son muy favorables, con un reducido número de marcos perdidos (ver tabla 3.4) y sin un efecto apreciable del número de serie sobre los valores obtenidos (el efecto principal de Serie no es significativo en un ANOVA posterior, *vide infra*). Cabe destacar de estos resultados que por encima de los 100 ms de intervalo no solamente el número de marcos perdidos medios es inferior a 1, sino que la DT se encuentra en valores inferiores a 0,25, lo que da una muestra de la exactitud y precisión temporal de esta tecnología.

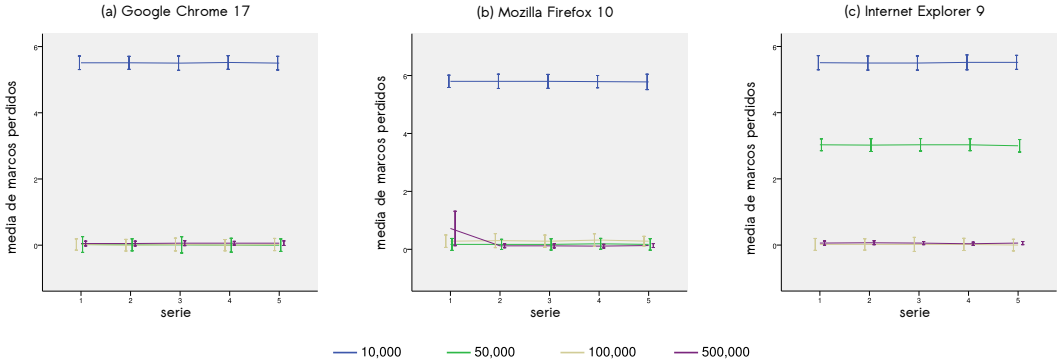

Figura 3.10: Número de marcos perdidos por serie para las animaciones en GIF89a : (a) Google Chrome 17, (b) Mozilla Firefox 10, (c) Internet Explorer 9.

Intervalo (ms)	**500**		**100**		**50**		**16,667**	
Agente de usuario								
	M	DT	M	DT	M	DT	M	DT
Google Chrome 17	0,06	0,234	0,01	0,109	0,01	0,425	0,04	0,215
Mozilla Firefox 10	0,06	0,246	0,01	0,134	0	0,35	0,05	0,24
Internet Explorer 9	0,06	0,238	0,01	0,134	0,01	0,816	0,51	0,653

Tabla 3.4: Estadísticos descriptivos del número de marcos perdidos para las animaciones con Flash.

Los resultados de las pruebas de Java son sensiblemente inferiores a los obtenidos mediante Adobe Flash, en especial en lo que respecta a los agentes de usuario Google Chrome 17 e Internet Explorer 9 (media de marcos perdidos por encima de 0,8 en todas las configuraciones, ver tabla 3.5). A pesar de estos valores poco favorables, la ausencia de patrones en el número de marcos perdidos en función del número de serie evaluada habla en favor de la estabilidad temporal de Java.

El rendimiento en Silverlight es muy similar al obtenido en las pruebas de Adobe Flash, a excepción de los resultados relativos al intervalo de 16,667 ms, donde se aprecia un peor rendimiento en todos los agentes de usuario (DT por encima de 1 en todos los casos, ver tabla 3.6).

Intervalo (ms) / **Agente de usuario**	**500**		**100**		**50**		**16,667**	
	M	DT	M	DT	M	DT	M	DT
Google Chrome 17	0,94	1,735	0,85	1,426	1,71	2,823	1	0,774
Mozilla Firefox 10	0,12	0,325	0,36	1,845	0,09	0,514	0,04	0,249
Internet Explorer 9	0,95	2,057	0,96	1,695	2,18	3,182	0,9	0,443

Tabla 3.5: Estadísticos descriptivos del número de marcos perdidos para las animaciones con Java.

Al igual que en el resto de tecnologías evaluadas en esta sección, no se encuentran patrones en el número de marcos perdidos en función del número de serie evaluada.

Intervalo (ms) / **Agente de usuario**	**500**		**100**		**50**		**16,667**	
	M	DT	M	DT	M	DT	M	DT
Google Chrome 17	0,06	0,94	0,01	0,713	0,02	0,806	0,26	1,154
Mozilla Firefox 10	0,06	0,893	0,01	0,741	0,01	0,797	0,33	1,533
Internet Explorer 9	0,06	0,935	0,02	0,758	0,01	0,792	0,28	1,027

Tabla 3.6: Estadísticos descriptivos del número de marcos perdidos para las animaciones con Silverlight.

Con el objeto de obtener una visión global de estos resultados y tratar de identificar los factores más influyentes en el número de marcos perdidos al emplear estas tecnologías, se ha realizado un ANOVA 4 (Tecnología: GIF89a, Flash, Java, Silverlight) x 3 (Agente de usuario: Google Chrome 17, Mozilla Firefox 10, Internet Explorer 9) x 4 (Intervalo: 500, 100, 50, 16,667) x 5 (Serie: 1 a 5) x 2 (Color: blanco, negro) de medidas independientes. Como cabía esperar, el efecto principal

de Serie no es significativo, lo que indica la ausencia de una degradación temporal a lo largo de las series al emplear estas tecnologías. Debido al gran número de observaciones, fueron muchos los efectos que alcanzaron el grado de significación, a pesar de que su tamaño fue poco considerable. Analizadas las interacciones según el tamaño del efecto, la más relevante de todas las encontradas en este análisis es la de Tecnología x Agente de usuario x Intervalo, $F(16, 23999) = 82{,}384$, $p < 0{,}001$, $\eta_p^2 = 0{,}053$, lo que podría sugerir que todas estas tecnologías se comportan de manera diferente en función del intervalo configurado y el agente de usuario sobre el que se prueben, aunque el reducido tamaño del efecto quizá no permita aseverar tal cosa. Como podría esperarse, de todos los factores analizados, el efecto principal de intervalo es el que más influencia tiene ($F(4, 23999) = 6434{,}541$, $p < 0{,}001$, $\eta_p^2 = 0{,}523$).

3.1.3.5 **Conclusiones**

En general, la exactitud y precisión temporal en la presentación de contenido visual de las tecnologías web clásicas es adecuada (media de marcos perdidos inferiores a 1) por encima de los 100 ms de intervalo. Sin embargo, hay resultados y condicionantes que desaconsejan ciertos usos.

En el caso de GIF89a, a pesar de los buenos resultados en Google Chrome 17 y Mozilla Firefox 10 para intervalos iguales o superiores a 50 ms, o 100 ms para el caso de Internet Explorer 9, su gran desventaja reside en la imposibilidad de controlar o sincronizar el estado de la animación una vez iniciada. Esta carencia hace que esta tecnología no sea adecuada para la gran mayoría de aplicaciones web que requieren una alta exactitud y precisión en la presentación de sus contenidos multimedia.

Al analizar Adobe Flash, los resultados obtenidos están en consonancia con el hecho de que ha sido la tecnología de desarrollo de aplicaciones web interactivas por excelencia durante los últimos años. La media del número de marcos perdidos es inferior a 0,1 en todos los casos salvo en Internet Explorer 9 con intervalos de 16,667 ms, y la DT es inferior a 1 en todos los casos. Conviene recordar que estos valores favorables corresponden a los resultados obtenidos empleando el mecanismo de temporización *loop* en las animaciones, ya que los mecanismos de tempo-

rización implementados en ActionScript (*setInterval*, *polling* y *timer*) ofrecieron peores resultados en las pruebas previas.

En lo que respecta a Java, el escaso rendimiento obtenido en Google Chrome 17 e Internet Explorer 9 (en torno a 1 marco perdido de media en todas las condiciones) contrasta con los buenos resultados obtenidos en Mozilla Firefox 10 (por debajo de 0,4 marcos perdidos de media en todas las condiciones). Este hecho resulta sorprendente si se tiene en cuenta que Java es una tecnología de máquina virtual con escasa interacción con el documento HTML en el que es embebida.

Finalmente, a pesar de su menor tasa de implantación y del muy reducido número de aplicaciones web implementadas en Silverlight, su rendimiento es muy similar al obtenido en Flash salvo en el intervalo más corto (16,667 ms), lo que parece razonable para una tecnología de cliente que ha sido concebida a la imagen y semejanza de la plataforma de Adobe.

En definitiva, podemos concluir que si bien la exactitud y precisión temporal de estas tecnologías es comparable con las obtenidas mediante tecnologías offline, su uso en la Web está en claro declive, por lo que se hace necesario analizar las tecnologías propuestas por los nuevos estándares web en torno a HTML5.

3.1.4 Tecnologías online modernas

3.1.4.1 Descripción de las tecnologías analizadas

Las limitaciones de las versiones de HTML precedentes a HTML5 (Ragget, 1997) han forzado a los desarrolladores web a embeber objetos ajenos a este estándar a través de un amplio abanico de tecnologías propietarias (Java, Flash, Silverlight) para lograr cubrir sus necesidades. Este hecho ha provocado un entorno de ejecución extremadamente fragmentado dentro de los agentes de usuario web, en el que muchas de las funcionalidades tienen que implementarse varias veces en cada tecnología complementaria para garantizar que funcionarán en todo el rango de posibles combinaciones de agentes de usuario y complementos. La necesidad de un estándar que desde HTML pueda ofrecer toda la funcionalidad que se viene demandando por parte de los desarrolladores de aplicaciones web es cada vez más acuciante. Por esta razón, el WHATWG y el W3C han unido sus esfuerzos para

definir HTML5, una nueva versión de HTML en la que el comportamiento, y no solamente el contenido, es también un elemento fundamental (Hickson, 2012a).

En lo que respecta a la presentación exacta y precisa de contenido audiovisual, los estándares alrededor de HTML5 proporcionan diversas API que para la generación de animaciones. Algunas de ellas son parte de la propia especificación de HTML5 (v. gr., Canvas 2D API), y otras están definidas en otras especificaciones (v. gr., WebGL). A pesar de esta clara distinción, la práctica totalidad de desarrolladores de agentes de usuario web (Microsoft, Apple, Google, Mozilla, Opera) emplean HTML5 como un término genérico para referirse a todas las tecnologías web modernas que han surgido en torno a esta especificación, tales como las transiciones y animaciones CSS, JavaScript versión 1.8 o WebGL, entre otras.

Crear animaciones en HTML5 no se limita a añadir contenido multimedia a un documento HTML a través del elemento `video`, sino que implica la manipulación del árbol DOM (*Document Object Model*, ver Le Hégaret, Whitmer, y Wood, 2009) a lo largo del tiempo, la gestión de eventos, la interacción humana, la sincronización e incluso el procesamiento de datos en tiempo real.

La mayoría de casos exigen la programación en JavaScript, aunque también existen animaciones que pueden definirse de manera exclusivamente declarativa. Tanto el uso de animaciones procedimentales como declarativas aporta ventajas e inconvenientes. Las animaciones declarativas son habitualmente más fáciles de crear. Los desarrolladores se limitan a definir el comportamiento deseado (v. gr., cambiar la anchura de una caja de 100 píxeles a 800 píxeles en 5 s), sin tener que especificar al agente de usuario web cómo ha de lograrlo.

Las animaciones procedimentales, por contra, son más flexibles y están menos limitadas por la funcionalidad definida por los estándares (Dahlström y cols., 2011; Fraser y cols., 2012; Jackson, Hyatt, Marrin, y Baron, 2012; Jackson, Hyatt, Marrin, Galineau, y Baron, 2012a). Sin embargo, este tipo de animaciones requiere la implementación precisa de los cambios que han de producirse en forma de procedimiento (v. gr., crear un bucle en el que se incrementa progresivamente la anchura de una caja, añadiendo un valor constante a la propiedad CSS correspondiente a lo largo del tiempo). Por esta razón, las animaciones procedimentales dependen habitualmente de los mecanismos de temporización de JavaScript, por lo que pueden

interferir o ser interferidas por otras interacciones que se estén procesando en la cola de eventos. Así, en los casos en los que sea posible, es recomendable definir una animación de forma declarativa, tanto por la simplicidad de su definición como por las ventajas en cuanto a rendimiento que pueden aportar (v. gr., aceleración gráfica por GPU, liberación de la cola de eventos).

Entre la familia de estándares en torno a HTML5, existen dos especificaciones para definir animaciones declarativas implementadas de forma nativa dentro del entorno de ejecución de los agentes de usuario modernos: SVG Animations (Dahlström y cols., 2011) y CSS Animations (Jackson, Hyatt, Marrin, Galineau, y Baron, 2012a). Las animaciones declarativas en SVG se apoyan en el estándar SMIL (Schmitz y Cohen, 2001), ya que SVG es un lenguaje huésped en terminología SMIL.

Esto significa que SVG soporta los elementos de animación definidos en la especificación SMIL (`animate`, `set`, `animateMotion`, `animateColor`), pero además incluye otras extensiones compatibles (`animateTransform`, `path`, `mpath`, `keyPoints`, `rotate`) para este propósito. El listado 3.3 muestra un ejemplo de una animación declarativa de la expansión de un rectángulo negro de 200 x 200 píxeles a una anchura de 800 píxeles durante 5 s (comenzando en 00:00:01s y finalizando en 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>
```

Listado 3.3: Animación declarativa empleando SVG y SMIL.

Además de los beneficios ya comentados, las animaciones en SVG pueden visualizarse de forma independiente al documento HTML que las contiene. Sin embargo, este tipo de animaciones no está exento de inconvenientes. El principal problema al que se enfrentan es la falta de soporte nativo de alguna de sus funcionalidades en los principales agentes de usuario web (v. gr., la definición de animaciones

repetitivas con SMIL en Google Chrome 17).

Las animaciones declarativas mediante CSS se definen a través de varias especificaciones. La API CSS Transforms (Fraser y cols., 2012) permite realizar transformaciones bidimensionales o tridimensionales a elementos a través de diferentes funciones de transformación (v. gr., `scaleX`, `skewY`, `rotate3d`) y propiedades relacionadas (v. gr., `perspective`, `transform-origin`, etc.). La API CSS Transitions (Jackson, Hyatt, Marrin, y Baron, 2012) se emplea para cambiar propiedades CSS de un valor a otro en un periodo de tiempo (i. e., a través de una función de transición temporal como `ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, o `cubic-bezier`). Finalmente, la API CSS Animations (Jackson, Hyatt, Marrin, Galineau, y Baron, 2012a) permite declarar un conjunto de marcos clave (*keyframes*) con diferentes transiciones entre ellos (i. e., diferentes valores de propiedades, duración, función de transición temporal, número de iteraciones, etc.). El listado 3.4 muestra un ejemplo de una animación declarativa usando CSS Animations en la que se modifica la anchura de un elemento `div` de 200 píxeles a 800 píxeles linealmente durante 5 s, repetida indefinidamente.

```
@keyframes change {
  0% { width: 200px; }
  100% { width: 800px; }
}

div#box {
  animation-name: change;
  animation-duration: 5s;
  animation-iteration-count: infinite;
  animation-timing-function: linear;
}
```

Listado 3.4: Animación declarativa empleando CSS Animations.

El uso de animaciones procedimentales en lugar de animaciones declarativas está motivado bien por la falta de soporte en el entorno de ejecución del agente de usuario empleado, bien por la complejidad de la animación en sí, que dificulta o evita su definición declarativa. En estos casos es habitual el uso de temporizadores

de JavaScript para modificar las propiedades de los elementos implicados en la animación a lo largo del tiempo (ver listado 3.5).

```
var interval = 1000 / 60; // 60 FPS

function start() {
  // Definir estado inicial
  setTimeout(animate, interval);
}

function animate() {
  // Cambio de propiedades
  setTimeout(animate, interval);
}
```

Listado 3.5: Esquema de animación procedimental empleando temporizadores de JavaScript.

Sin embargo, la exactitud y precisión de los temporizadores de JavaScript no depende exclusivamente de ellos, sino del estado de la cola de eventos. Por esta razón, algunos desarrolladores web emplean temporizadores de JavaScript con retardos de 0 ms para poder actualizar la animación tantas veces como sea posible. Esta mala práctica se ve compensada por la limitación del retardo mínimo fijada por el estándar (4 ms). Asimismo, el hecho de que todo el código JavaScript relativo a un documento HTML se ejecute en un único hilo provoca que este tipo de animaciones bloqueen el tratamiento de eventos asíncronos (v. gr., la entrada del usuario, una actualización del contenido mediante AJAX) al colapsar la cola de eventos (Resig, 2008). Por esta razón, la API Web Workers (Hickson, 2012c) puede resultar de ayuda en los casos en los que sea posible delegar en un hilo de ejecución secundario el cálculo de valores que no tengan que ver con la interfaz de usuario (v. gr., el cómputo de coordenadas en una animación compleja dentro de un motor de simulaciones físicas). Otra de las API relacionadas con los Web Workers para el paso de mensajes entre diferentes documentos HTML (la función `postMessage` de la API Cross-Document Messaging) ha sido empleada para recrear temporizadores con un retardo de 0 ms (Baron, 2010). Para ello, Baron (2010) define una clausura (*closure*) en JavaScript que añade el méto-

do `setZeroTimeout` al objeto `window`. Este método realiza un intercambio de mensajes a través de `postMessage` con el propio documento HTML, evitando la limitación temporal en el retardo mínimo de los temporizadores de JavaScript. La implementación nativa de este tipo de funcionalidad fue propuesta por Mann y Weber (2011) a través de el método `setImmediate`.

Teniendo en cuenta todas estas limitaciones y problemas, se ha desarrollado una nueva API para facilitar la creación de animaciones procedimentales en las que el agente de usuario sea quien controle la frecuencia de actualización de la animación, haciéndola coincidir con la tasa de refresco del monitor (Robinson y McCormack, 2012). De esta forma, el desarrollador delega en el agente de usuario la elección del mejor momento para actualizar su animación. Esto redunda en un mejor rendimiento y un menor consumo de recursos (sobre todo en los casos en los que la animación no está siendo visualizada en pantalla, puesto que el agente de usuario reduce su tasa de actualización al mínimo). Por estas razones, es muy recomendable el uso de esta API, si bien al tratarse de un desarrollo reciente deberán emplearse implementaciones suplementarias (llamadas coloquialmente *polyfills* o *shims*) que proporcionen una funcionalidad similar cuando la API no se encuentre disponible. En el listado 3.6 puede verse un ejemplo de su uso a través del método `requestAnimationFrame`.

En cuanto al contenido, las animaciones procedimentales no solamente se limitan a la modificación de propiedades CSS o elementos SVG, sino que también son capaces de interactuar con varias API de dibujado relacionadas con HTML5 como el contexto 2D de `canvas` (Hickson, 2012b) o el contexto 3D de WebGL (Marrin, 2011). Estas dos especificaciones proporcionan un lienzo virtual sobre el que dibujar formas, texto, imágenes, aplicar transformaciones (escalado, rotación, traslación), y cambiar valores de color y sombreado. En el caso de WebGL, estas funcionalidades pueden extenderse a través de una API de desarrollo en JavaScript derivada de OpenGL ES 2.0 (Munshi y Leech, 2010) que proporciona funcionalidades de dibujado en 3D en modo inmediato empleando recursos propios de OpenGL (i. e., texturas, *buffers*, *framebuffers*, *renderbuffers*, *shaders*, etc.) como objetos dentro del árbol DOM.

```
var interval = 1000 / 60, // 60 FPS
    last;

function start() {
  // Definir estado inicial
  last = Date.now();
  requestAnimationFrame(animate);
}

function animate(time) {
  if (time >= last + interval) {
    // Cambio de propiedades
    last = time;
  }
  requestAnimationFrame(animate);
}
```

Listado 3.6: Esquema de animación procedimental empleando la API para el control temporal de animaciones procedimentales.

3.1.4.2 Procedimiento

El análisis de la exactitud y precisión en la presentación de contenidos visuales de las tecnologías web previamente mencionadas se ha llevado a cabo de manera similar a los análisis previos realizados en este capítulo. Las pruebas correspondientes a animaciones declarativas se han definido empleando CSS Animations y SVG con SMIL, mientras que las animaciones procedimentales se han preparado utilizando SVG, Canvas y WebGL a través de JavaScript. El mecanismo de temporización elegido ha sido el proporcionado por la API para el control temporal de animaciones procedimentales (`requestAnimationFrame`). Esta decisión está motivada por dos razones: 1) esta API es la que más ventajas aporta en cuanto a funcionalidad y optimización de los recursos de entre todas las disponibles, y 2) en un análisis previo de los mecanismos de temporización, esta API ofreció un rendimiento similar al resto de temporizadores evaluados (*vide infra*).

Estas tecnologías han sido puestas a prueba en los agentes de usuario Google Chrome 17 y Mozilla Firefox 10, sobre los sistemas operativos Microsoft Windows 7 SP 1, Ubuntu Linux 10.04 LTS y Apple Mac OS X 10.7. Estos agentes de usuario han sido seleccionados por estar disponibles en los tres sistemas operativos

evaluados y por ser compatibles con todas las tecnologías mencionadas de manera nativa.

Para cada una de las combinaciones de tecnología web, agente de usuario y sistema operativo anteriormente mencionadas se prepararon varias animaciones de 200 x 200 píxeles de tamaño con transiciones no graduales de negro a blanco repetidas en las que se varió la duración de cada pantalla, probándose las duraciones de 500, 100, 50 y 16,667 ms (30, 6, 3 y 1 *ticks* a 60 Hz, respectivamente). Para cada una de estas duraciones se registraron 5 series independientes de 60 s de duración cada una. De cada una de estas series se descartaron los 5 primeros y últimos segundos para evitar errores propios de la inicialización o finalización, y se analizaron las primeras 100 muestras de cada serie, por lo que finalmente se obtuvieron 500 muestras por cada duración en cada combinación de software y sistema operativo. Las causas que motivan este procedimiento son coincidentes con las presentadas en la sección anterior.

```
@keyframes change {
  0% { background-color: #ffffff; }
  50% { background-color: #000000; }
}

div#box {
  animation-name: change;
  animation-duration: 1.0s;
  animation-iteration-count: infinite;
  animation-timing-function: steps(1);
  position: absolute;
}
```

Listado 3.7: Definición mediante una animación CSS de una transición no gradual de negro a blanco repetida indefinidamente.

Las animaciones de las pruebas relativas a CSS fueron definidas empleando una animación CSS con dos marcos clave. El primero, al 0 % de progreso de la animación, fija el color de fondo del elemento `div` a blanco, y el segundo, al 50 % de progreso de la animación, lo fija a negro. La duración total de la animación se ha configurado con el doble del valor del intervalo, de tal forma que el 50 % de la animación corresponda al valor del intervalo evaluado (v. gr., para 100 ms de

intervalo se configura una duración de 200 ms con el objeto de que el 50 % del total de la animación corresponda con una duración de 100 ms por marco). La función de transición temporal empleada es `step`, dado que se requiere una transición no gradual de negro a blanco. El listado 3.7 muestra un fragmento de la definición de una prueba de CSS para 500 ms de intervalo.

```
<?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="white">
    <set id="toblack" begin="0; towhite.begin+0.500s"
      attributeName="fill" to="black"/>
    <set id="towhite" begin="toblack.begin+0.500s"
      attributeName="fill" to="white"/>
  </rect>
</svg>
```

Listado 3.8: Definición mediante SVG con SMIL de una transición no gradual de negro a blanco repetida indefinidamente.

En el caso de las pruebas de SVG con SMIL, ha sido necesario preparar dos versiones diferentes para cada agente de usuario, puesto que Google Chrome 17 no soporta las referencias cruzadas al definir el comienzo de un elemento de animación (ver listado 3.8). Por esta razón, se han desenrollado los bucles de animación paras las pruebas de SVG con SMIL en este agente de usuario. Estos ficheros han sido creados a través de *shell scripts*, y tienen el inconveniente añadido de que no soportan la repetición indefinida de la animación (ver listado 3.9).

En cuanto a las pruebas relativas a las animaciones procedimentales, todas ellas emplean una estructura similar basada en el uso de la API para el control temporal de animaciones procedimentales (ver listado 3.6), actualizando el fondo del elemento correspondiente con los métodos apropiados en cada caso. Así, los test que ponen a prueba la exactitud y precisión de la API Canvas 2D al presentar contenido visual emplean la propiedad `fillStyle` y el método `fillRect` para actualizar el color de fondo de la animación. En el caso de WebGL, son los métodos `clearcolor` y `clear` del contexto 3D los que actualizan el color de fondo. Finalmente, en SVG es el método `setAttributeNS` el encargado de hacer lo propio.

```svg
<?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">
    <set begin="0.500s" attributeName="fill" to="white"/>
    <set begin="1.000s" attributeName="fill" to="black"/>
    <set begin="1.500s" attributeName="fill" to="white"/>
    [...]
    <set begin="299.000s" attributeName="fill" to="black"/>
    <set begin="299.500s" attributeName="fill" to="white"/>
    <set begin="300.000s" attributeName="fill" to="black"/>
  </rect>
</svg>
```

Listado 3.9: Definición mediante SVG con SMIL de una transición no gradual de negro a blanco repetida un número finito de veces a través del desenrollamiento de bucle.

3.1.4.3 Análisis previo de temporizadores

Antes de comenzar con el análisis de todas las tecnologías online modernas, conviene detenerse a estudiar qué mecanismo de temporización es el idóneo para las animaciones procedimentales que lo requieren. Así, se han analizado los temporizadores estándar de JavaScript (`setTimeout`), los temporizadores inmediatos basados en el paso de mensajes (`postMessage`) y los temporizadores proporcionados por la API para el control temporal de animaciones procedimentales (`requestAnimationFrame`) para las tecnologías SVG, Canvas 2D y WebGL.

Al tratarse de un estudio exploratorio, las pruebas se han limitado al agente de usuario más empleado (Google Chrome) sobre el sistema operativo más empleado (Microsoft Windows). El procedimiento es similar a los ya presentados previamente, en los que se prepararon varias animaciones de 200 x 200 píxeles de tamaño con transiciones no graduales de negro a blanco repetidas variando la duración de cada marco. Se analizan los intervalos de 500, 100, 50 y 16,667 ms y se registran 5 series independientes de 60 s de duración cada una. Tras descartar los 5 primeros segundos, se analizan las primeras 100 muestras de cada serie, por lo que finalmente se obtienen 500 muestras por cada duración en cada combinación de tecnología web y mecanismo de temporización.

Intervalo (ms) / Tecnología		500		100		50		16,667	
		M	DT	M	DT	M	DT	M	DT
SVG	`setTimeout`	0,95	0,218	0,57	1,026	0,75	0,433	0,09	0,299
	`postMessage`	0,67	0,625	0,59	1,054	1,18	1,85	2,98	6,105
	`requestAnimationFrame`	0,8	0,397	0,85	0,548	0,85	0,567	0,93	0,486
Canvas	`setTimeout`	0,96	0,488	0,57	1,073	0,75	0,431	0,09	0,318
	`postMessage`	0,77	0,954	0,62	1,042	0,86	1,057	1,54	1,756
	`requestAnimationFrame`	0,8	0,399	0,85	0,547	0,85	0,557	0,93	0,504
WebGL	`setTimeout`	0,06	0,238	0,71	0,255	0,01	0,109	0,03	0,181
	`postMessage`	0,16	0,453	0,2	0,746	0,16	0,638	0,1	0,518
	`requestAnimationFrame`	0,78	0,413	0,73	0,443	0,85	0,359	0,93	0,262

Tabla 3.7: Estadísticos descriptivos del número de marcos perdidos para distintas combinaciones de tecnologías web sobre Google Chrome 17 en Windows 7.

Los resultados de las pruebas pueden verse en la tabla 3.7, donde se muestran la media y DT de los marcos perdidos en cada prueba. A partir de estos valores puede colegirse que los temporizadores estándar de JavaScript son los más indicados para este tipo de animaciones. Sin embargo, como ya hemos mencionado anteriormente, estos temporizadores presentan limitaciones importantes relacionadas con la sobrecarga de la cola de eventos y el bajo aprovechamiento de los recursos del sistema, por lo que los resultados obtenidos podrían verse empeorados sensiblemente en casos en los que varias animaciones concurrentes compitieran por los mismos recursos. Teniendo estos aspectos en cuenta y considerando que los resultado de `requestAnimationFrame` frente al resto de temporizadores se encuentran en valores intermedios (salvo en el caso de WebGL), creemos conveniente realizar el resto de pruebas empleando el temporizador proporcionado por la API para el control temporal de animaciones procedimentales, por ser este el recomendado por el estándar para este tipo de casos.

Con el objeto de identificar los factores más influyentes en el número de marcos perdidos al emplear estas tecnologías, se ha realizado un ANOVA 3 (Tecno-

logía: SVG, Canvas, WebGL) x 3 (Temporizador: `setTimeout`, `postMessage`, `requestAnimationFrame`) x 4 (Intervalo: 500, 100, 50, 16,667) x 5 (Serie: 1 a 5) x 2 (Color: blanco, negro) de medidas independientes. Sus resultados indican que el efecto principal de Serie no es significativo, lo que sugiere que se trata de tecnologías estables en el tiempo. De forma similar a otros análisis previos, el gran número de observaciones propició que fueran muchos los efectos que alcanzaron el grado de significación. Como podría intuirse a priori, de todos los factores analizados, el efecto principal de Intervalo es el que más influencia tiene ($F(3, 21999) = 445{,}637$, $p < 0{,}001$, $\eta_p^2 = 0{,}058$), si bien el tamaño del efecto es muy inferior a anteriores análisis de otras tecnologías web. La interacción más relevante de todas las encontradas en este análisis según el tamaño de su efecto es la de Tecnología x Intervalo, $F(9, 21999) = 222{,}931$, $p < 0{,}001$, $\eta_p^2 = 0{,}085$, lo que podría sugerir que modificar el intervalo tiene diferentes efectos en función de la tecnología empleada. Sin embargo, el tamaño del efecto de esta interacción es tan pequeño que no permite aseverar esta conclusión.

3.1.4.4 **Resultados**

Después de realizar las pruebas descritas, analizamos sus resultados estableciendo el número de marcos perdidos en cada una de ellas y calculando sus estadísticos descriptivos.

En el caso de CSS, se observa una marcada caída del rendimiento por debajo de 50 ms, con resultados especialmente pobres en Mozilla Firefox 10 sobre Windows 7 (M: 4,89, DT: 5,041). Por encima de ese valor, el rendimiento es adecuado, comparable con los resultados obtenidos en las pruebas de tecnologías online clásicas (ver tabla 3.8).

Los resultados obtenidos en las pruebas realizadas con SVG y SMIL siguen un patrón similar al encontrado en los resultados de las pruebas con CSS Animations, si bien la degradación a partir de los 50 ms de intervalo es mucho más pronunciada (ver tabla 3.9), llegando a extremos como el de Google Chrome 17 sobre Windows 7 (M: 24,69, DT: 44,167).

En general, el rendimiento de las dos tecnologías de definición de animaciones declarativas es igual o mejor que el obtenido en las mismas pruebas mediante tecnologías web clásicas, salvo para el caso de 16,667 ms de intervalo en el que

Intervalo (en ms)		**500**		**100**		**50**		**16,667**	
SO	**Agente de usuario**								
		M	DT	M	DT	M	DT	M	DT
Windows 7	Google Chrome 17	0,06	0,649	0,01	0,522	0	0,606	2,62	0,79
	Mozilla Firefox 10	0,06	0,603	0,01	1,043	0,01	1,107	4,89	5,041
GNU/Linux	Google Chrome 17	0	0,961	0	0,537	0	0,459	2,17	0,588
	Mozilla Firefox 10	0	0,708	0	0,655	0	0,789	0,32	0,691
Mac OS X	Google Chrome 17	-0,06	0,85	0	0,667	0	0,461	2,42	0,71
	Mozilla Firefox 10	-0,06	0,707	-0,01	0,668	0	0,537	0,14	0,362

Tabla 3.8: Estadísticos descriptivos del número de marcos perdidos para las animaciones con CSS Animations.

Intervalo (en ms)		**500**		**100**		**50**		**16,667**	
SO	**Agente de usuario**								
		M	DT	M	DT	M	DT	M	DT
Windows 7	Google Chrome 17	0,06	0,281	0,01	0,893	0,01	1,041	24,69	44,167
	Mozilla Firefox 10	0,07	0,575	0,01	1,092	0	1,082	4,73	4,055
GNU/Linux	Google Chrome 17	0	0,11	0	0,564	0	0,753	11,11	3,983
	Mozilla Firefox 10	0	0,729	0	0,65	0	0,806	0,34	0,719
Mac OS X	Google Chrome 17	-0,06	0,254	-0,01	0,257	0	0,393	14,47	4,937
	Mozilla Firefox 10	-0,06	0,705	-0,01	0,679	0	0,554	0,13	0,369

Tabla 3.9: Estadísticos descriptivos del número de marcos perdidos para las animaciones con SVG y SMIL.

la caída de rendimiento es tal que resulta inaceptable. Estos valores permanecen estables, salvo en el caso de SVG y SMIL para 16,667 ms que sufre un incremento en el número de marcos perdidos, como puede verse en la figura 3.11.

En los resultados de las pruebas de animaciones procedimentales con la API Canvas 2D se observan valores muy similares entre todas las condiciones, con me-

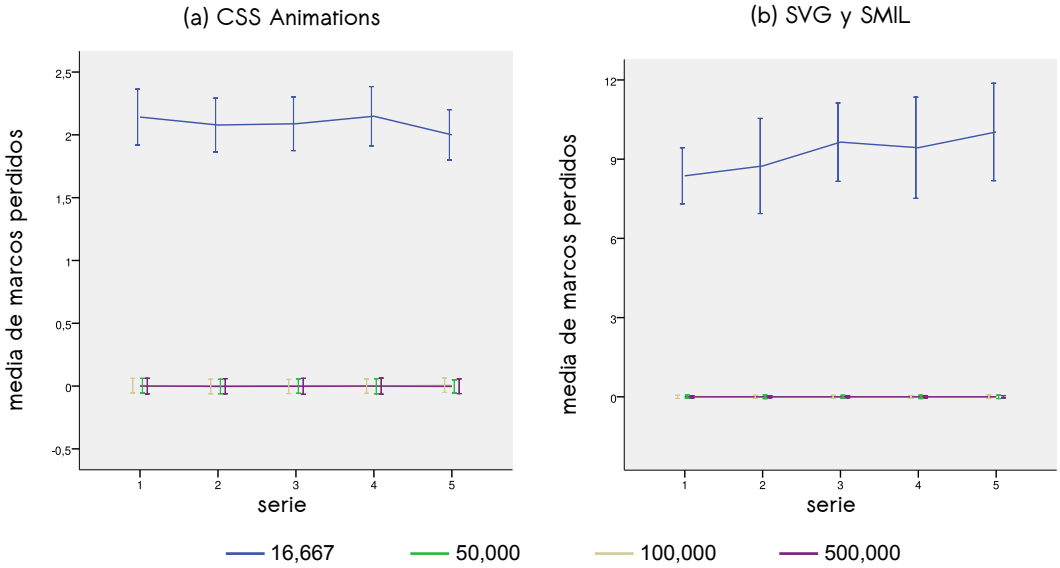

Figura 3.11: Número de marcos perdidos por serie para las animaciones: (a) con CSS Animations, (b) con SVG y SMIL.

dias de marcos perdidos entre 0,11 y 0,94, y DT entre 0,265 y 1,055 (ver tabla 3.10). Algo parecido ocurre con los resultados de las animaciones realizadas con SVG y JavaScript, con medias entre 0,07 y 1,50, y DT entre 0,252 y 0,890 (ver tabla 3.11), y los resultados de las animaciones realizadas con WebGL, con medias entre 0,09 y 0,98, y DT entre 0,262 y 1,028 (ver tabla 3.12). En todas ellas puede observarse como determinadas combinaciones de agente de usuario y sistema operativo obtienen sistemáticamente muy buenos resultados (v. gr., Mozilla Firefox 10 sobre Mac OS X es la combinación que mejores resultados obtiene en la mayoría de pruebas), pero lo contrario es difícil de identificar, ya que los peores resultados se reparten entre agentes de usuario y navegadores de forma desigual en cada prueba.

A modo de resumen en la figura 3.12 se presentan los valores de la media de marcos perdidos para las diferentes tecnologías web modernas evaluadas (CSS Animations, SVG con SMIL, Canvas 2D, SVG con JavaScript y WebGL) y los diferentes intervalos (500, 100, 50 y 16,667 ms) en los agentes de usuario analizados. En ellas se puede apreciar cómo las tecnologías de animaciones declarativas (CSS Animations y SVG con SMIL) presentan mejores resultados que las tecnologías de animaciones procedimentales (Canvas 2D, SVG con JavaScript y WebGL) salvo en el caso de 16,667 ms de intervalo, donde su rendimiento cae abruptamente.

Intervalo (en ms)		**500**		**100**		**50**		**16,667**	
SO	**Agente de usuario**								
		M	DT	M	DT	M	DT	M	DT
Windows 7	Google Chrome 17	0,8	0,399	0,85	0,547	0,85	0,557	0,93	0,504
	Mozilla Firefox 10	0,66	0,7	0,69	1,055	0,85	0,579	0,94	0,468
GNU/Linux	Google Chrome 17	0,11	0,708	0,8	0,467	0,88	0,442	0,82	0,726
	Mozilla Firefox 10	0,76	0,556	0,18	0,446	0,09	0,589	0,34	0,685
Mac OS X	Google Chrome 17	0,54	0,507	0,48	0,516	0,35	0,493	0,56	0,497
	Mozilla Firefox 10	0,45	0,498	0,47	0,499	0,22	0,412	0,06	0,265

Tabla 3.10: Estadísticos descriptivos del número de marcos perdidos para las animaciones con Canvas 2D.

Intervalo (en ms)		**500**		**100**		**50**		**16,667**	
SO	**Agente de usuario**								
		M	DT	M	DT	M	DT	M	DT
Windows 7	Google Chrome 17	0,8	0,397	0,85	0,548	0,85	0,567	0,93	0,485
	Mozilla Firefox 10	1,3	0,89	1,5	0,5	0,76	0,466	0,88	0,33
GNU/Linux	Google Chrome 17	0,11	0,677	0,8	0,508	0,88	0,401	0,81	0,738
	Mozilla Firefox 10	0,77	0,647	0,18	0,446	0,09	0,564	0,35	0,753
Mac OS X	Google Chrome 17	0,56	0,508	0,5	0,512	0,4	0,498	0,53	0,503
	Mozilla Firefox 10	0,54	0,499	0,52	0,5	0,26	0,437	0,07	0,252

Tabla 3.11: Estadísticos descriptivos del número de marcos perdidos para las animaciones con SVG y JavaScript.

Con el objeto de conocer el grado de influencia en el número de marcos perdidos de los distintos factores modificados en estas pruebas, se ha realizado un ANOVA 5 (Tecnología: CSS, SMIL, SVG, Canvas, WebGL) x 2 (Agente de usuario: Google Chrome 17, Mozilla Firefox 10) x 3 (Sistema Operativo: Windows 7, GNU/Linux, Mac OS X) x 4 (Intervalo: 500, 100, 50, 16,667) x 5 (Serie: 1 a

Intervalo (en ms)		500		100		50		16,667	
SO	**Agente de usuario**								
		M	DT	M	DT	M	DT	M	DT
Windows 7	Google Chrome 17	0,78	0,413	0,73	0,443	0,85	0,359	0,93	0,262
	Mozilla Firefox 10	0,98	0,942	0,69	1,028	0,91	0,659	0,95	0,514
GNU/Linux	Google Chrome 17	0,3	0,462	0,84	0,369	0,9	0,295	0,81	0,396
	Mozilla Firefox 10	0,73	0,536	0,17	0,486	0,09	0,353	0,4	0,725
Mac OS X	Google Chrome 17	0,58	0,507	0,47	0,523	0,44	0,497	0,55	0,498
	Mozilla Firefox 10	0,46	0,608	0,3	0,471	0,12	0,34	0,09	0,32

Tabla 3.12: Estadísticos descriptivos del número de marcos perdidos para las animaciones con WebGL.

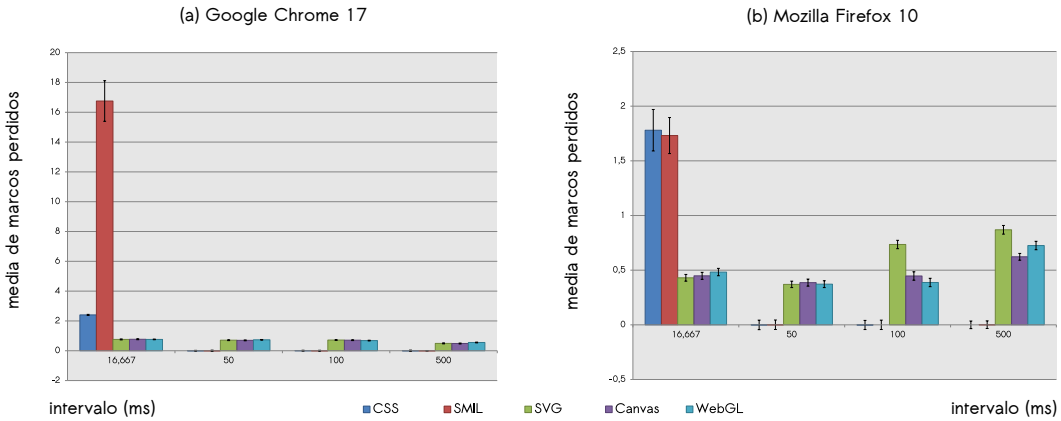

Figura 3.12: Resumen de los resultados obtenidos en las pruebas con tecnologías web modernas: (a) sobre Google Chrome 17, (b) sobre Mozilla Firefox 10.

5) x 2 (Color: blanco, negro) de medidas independientes. Los resultados de este análisis son muy similares a los anteriores: 1) el gran número de observaciones propicia que sean muchos los efectos que alcancen el grado de significación; 2) una vez más, el efecto principal de Serie no es significativo, de lo que se infiere que se trata de tecnologías que no sufren una degradación temporal; 3) de todos los factores analizados, el efecto principal de Intervalo es el que más influencia tiene ($F(3, 59999) = 1127{,}896$, $p < 0{,}001$, $\eta_p^2 = 0{,}054$) si los analizamos en función

del tamaño del efecto, y la interacción más relevante de todas las encontradas en este análisis según este mismo criterio es la de Tecnología x Agente de usuario x Sistema Operativo x Intervalo, $F(24, 59999) = 24{,}502$, $p < 0{,}001$, $\eta_p^2 = 0{,}010$, lo que concuerda con las explicaciones previas relativas a las variaciones entre tecnologías al probar distintos intervalos en diferentes agentes de usuario sobre distintos sistemas operativos. Sin embargo, ambos tamaños del efecto son tan pequeños que extraer conclusiones acerca de la influencia de cada factor a partir de estos resultados sería demasiado aventurado.

3.1.4.5 **Conclusiones**

A la vista de los resultados obtenidos se puede afirmar que de entre todas las tecnologías evaluadas, las animaciones declarativas con CSS son la opción más recomendable cuando los intervalos de animación se sitúan por encima de los 50 ms ya que el número de marcos perdidos es bajo (media entre 0,00 y -0,06 y DT entre 0,459 y 1,107 marcos perdidos), es una tecnología estable en el tiempo (sin efecto principal de Serie) e independiente del rendimiento de JavaScript, por lo que ni satura su cola de eventos ni se ve afectada por una eventual saturación de la misma por otras causas.

	M	**DT**
Flash	0,00–0,51	0,109–0,816
Java	0,04–2,18	0,249–3,182
Silverlight	0,01–0,33	0,713–1,533
WebGL	0,09–0,98	0,262–1,028
SVG	0,07–1,50	0,252–0,890
Canvas	0,06–0,94	0,265–1,055

Tabla 3.13: Resumen de los resultados obtenidos en las pruebas realizadas en los diferentes agentes de usuario sobre distintos sistemas operativos.

Algo similar podría decirse de la combinación de SVG y SMIL si su implementación en Google Chrome no fuera tan limitada en lo que concierne a rendimiento

y funcionalidad, como la ausencia de soporte para bucles a través de referencias temporales cruzadas.

El rendimiento de las tecnologías procedimentales (Canvas, SVG, WebGL con `requestAnimationFrame`) es similar al de tecnologías web previas (media de marcos perdidos entre 0,06 y 1,50 con DT entre 0,252 y 1,055, frente a medias entre 0,00 y 2,18 con DT entre 0,109 y 3182, ver tabla 3.13 para mayor detalle), con la ventaja de tratarse de tecnologías en proceso de estandarización y con un futuro prometedor, por lo que su uso es preferible frente a las tecnologías comentadas en la sección anterior (Java, Flash, Silverlight).

3.2 Análisis del contenido auditivo

Una vez analizado el grado de exactitud y precisión alcanzado por las diferentes tecnologías offline y online en la presentación de contenido visual, procedemos a hacer lo propio con un conjunto de tecnologías de contenido auditivo, focalizando la atención de este análisis en los nuevos estándares web relacionados con la reproducción y generación de contenido auditivo (HTML5 Audio Element, Audio Data API, Web Audio API) y comparándolos con el rendimiento obtenido al emplear software especializado en la presentación de contenido audiovisual de forma exacta y precisa.

3.2.1 Metodología y equipamiento

La metodología seguida en este análisis guarda un claro paralelismo con la seguida en el estudio relativo al contenido visual, ya que también se ha tomado un especial cuidado en minimizar las fuentes externas de retardos o latencias con el fin de obtener la medida más próxima al rendimiento real de cada tecnología evaluada. Otra de las similitudes entre estos análisis reside en la intención de que el equipamiento empleado para evaluar cada tecnología sea un agente externo a esta, un observador independiente que se limite a capturar el contenido generado de la manera más fidedigna posible, sin ningún otro tipo de interacción (ver figura 3.13). Por ello, la única conexión entre el equipo empleado para ejecutar cada una de las pruebas (Apple Macbook Pro A1211, con un procesador Intel Core 2 Duo T7600 y

una tarjeta de sonido SigmaTel 9220 A1 High Definiton Audio) y el equipo destinado a capturar el contenido generado (Lenovo con un procesador Core2 Duo E7500 a 2,93GHz y tarjeta de sonido Intel Corporation N10/ICH 7 Family High Definition Audio Controller) es a través de un cable macho-macho *mini-jack* de 3,5 mm entre la salida de audio del primero y la entrada de línea del segundo. Como software de grabación se empleó una distribución de GNU/Linux especializada en la grabación de audio con baja latencia (AVLinux 5.0.3 con un núcleo Linux 3.0.16 parcheado para tener soporte de tiempo real e IRQ Threading). Tal y como afirman Y. Wang y cols. (2012), a través de esta configuración se obtienen latencias próximas a 1,68 ms, un valor comparable a las consolas digitales de audio profesionales.

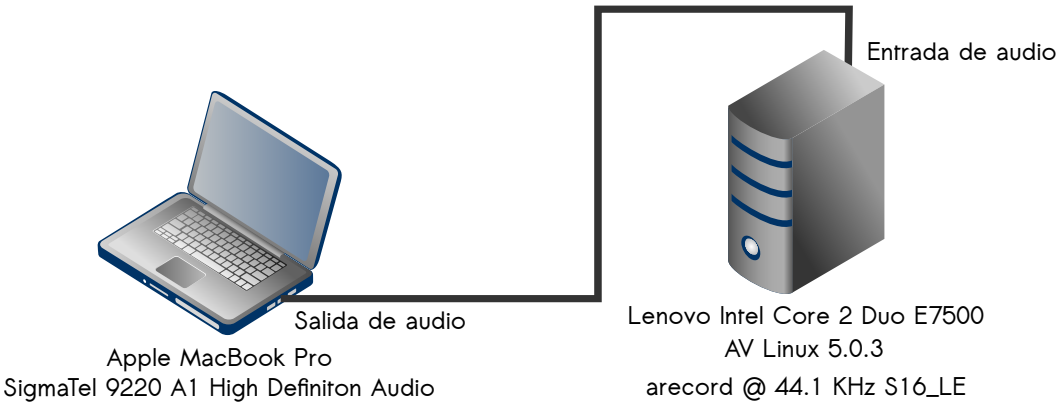

Figura 3.13: Elementos de prueba y medición en el análisis del contenido auditivo.

Como ya hemos avanzado previamente, las tecnologías que se han puesto a prueba han sido las siguientes: 1) DMDX, tecnología offline basada en DirectX de Microsoft que asegura cumplir con los altos requisitos temporales en la reproducción de contenido multimedia; 2) el elemento `audio` estándar dentro de la especificación de HTML5 (Hickson, 2012a); 3) la Audio Data API, propuesta por la Mozilla Foundation (Humphrey, Brook, MacDonald, Marxer, y Cliffe, 2012); y 4) la Web Audio API, propuesta por Google dentro del Audio Working Group del W3C (Rogers, 2012).

De esta forma, quedan excluidas del análisis tecnologías web que precisan complementos propietarios incrustados dentro del documento HTML (a través del elemento `object`, principalmente) o utilizan elementos HTML ya obsoletos. Este es el caso del audio embebido a través del elemento `embed`, o de las tecnologías

basadas en máquinas virtuales como Java, Flash o Silverlight. Sin embargo, estas eran hasta hace bien poco las únicas maneras de incluir audio dentro de una aplicación web. El elemento `audio` del estándar HTML5 pretende cubrir esta carencia al permitir incluir referencias a ficheros de audio dentro de un documento HTML estándar. La gestión de ese elemento de audio se hace a través de los métodos `load`, `play` y `pause` para la carga, reproducción y pausa respectivamente. Asimismo, existen un conjunto de propiedades (`currentSrc`, `currentTime`, `duration`) y de eventos (`loadedmetadata`, `timeupdate`, `pause`, `play` y `ended`) para tener conocimiento de su estado desde JavaScript. Dado que no todos los agentes de usuario web soportan todos los códecs de audio, el elemento audio soporta la inclusión de múltiples subelementos `source`.

A pesar de la sustancial mejora que supone poder gestionar elementos de audio de forma nativa, los desarrolladores de aplicaciones web demandan la existencia de una API que permita no solo reproducir audio sino también manipularlo en tiempo real. Como respuesta a esta necesidad han surgido varias iniciativas en paralelo, entre las que destacan la Audio Data API (Humphrey y cols., 2012) y la Web Audio API (Rogers, 2012).

La Web Audio API fue propuesta por la Mozilla Foundation e implementada en Mozilla Firefox 4 y superiores. Su principal característica es que aporta una extensión de la funcionalidad de los elementos `audio` y `video` propios del estándar HTML5 en forma de nuevas propiedades y eventos. Así, `loadedmetadata`, el evento propio del estándar HTML5, es complementado por esta API añadiendo los siguientes metadatos adicionales: 1) `mozChannels`, que indica el número de canales de audio, 2) `mozSampleRate`, que indica la frecuencia de muestreo del audio; y 3) `mozFrameBufferLength`, que indica el tamaño del *buffer* de reproducción de audio. El prefijo `moz`, propio de Mozilla, se incluye en el nombre de estas propiedades dado que la implementación de esta API todavía no ha sido aprobada como estándar. Se prevé que cuando este hecho se dé, el prefijo sea eliminado. Además de estos metadatos, esta API aporta un nuevo evento, `MozAudioAvailable`, que se genera cada vez que un fragmento de audio está disponible para ser reproducido (no tiene por qué coincidir con el ritmo de reproducción del audio). Este evento proporciona un vector con los datos de la muestra de audio decodificados (vector de valores de tipo `float`) y el tiempo en

segundos correspondiente a esa muestra medido desde el principio del elemento de audio.

Además de la posibilidad de conocer más en detalle el audio que está siendo reproducido y de modificar en tiempo real otros elementos del documento HTML gracias a estos nuevos metadatos y eventos, la Audio Data API proporciona varios métodos para la escritura de audio desde JavaScript. Estas funciones son: 1) `mozSetup`, para la configuración del número de canales y frecuencia de muestreo; 2) `mozWriteAudio`, para escribir los valores de audio que se desea reproducir (vector de valores de tipo `float`); y 3) `mozCurrentSampleOffset`, para obtener la posición actual del flujo de reproducción de audio medida en muestras.

Por otro lado, la Web Audio API, propuesta por Google (implementada en Google Chrome a partir de la versión 10) y adoptada por el Audio Working Group del W3C, tiene un enfoque muy diferente. En lugar de ampliar o completar el elemento audio de HTML5, emplea el concepto de contexto de audio, similar al que utilizan algunas API de dibujado como Canvas o WebGL. Dentro de este contexto de audio, se crean diferentes nodos de audio (`AudioNodes`) para cada necesidad. Así, existen los siguientes tipos de nodos de audio: 1) `AudioSourceNode`, para crear nodos de entrada de audio; 2) `AudioDestinationNode`, para crear nodos de salida de audio; 3) `DelayNode`, para crear nodos de retardos ajustables; 4) `AudioGainNode`, para crear nodos de ganancia; 5) `AudioPannerNode`, para crear nodos de audio en 3D; 6) `ConvolverNode`, para crear nodos de efectos de espacio, reverberaciones, etc.; 7) `RealtimeAnalyserNode`, para crear nodos de análisis en tiempo real de audio; 8) `DynamicsCompressorNode`, para crear nodos de efectos de compresión y expansión de audio; 9) `BiquadFilterNode`, para crear nodos de filtrado; y 10) `WaveShaperNode`, para crear nodos de efectos no-lineales y distorsiones. Gracias a estos nodos pueden configurarse múltiples entradas, salidas y filtros interconectables de formas complejas que permiten la generación y procesado de audio desde JavaScript.

No obstante, a diferencia de lo que sucede con la Audio Data API, esta API no delega en el desarrollador la implementación de los efectos típicos, filtros o el procesado de audio, sino que permite el acceso de alto nivel (desde JavaScript) a implementaciones eficientes de estas funcionalidades en lenguajes de bajo nivel (C y ensamblador).

En el listado 3.10 se muestra cómo a partir de un contexto de audio (el método `webkitAudioContext` tiene el prefijo `webkit` por tratarse de una implementación todavía no aprobada como estándar pero será eliminado cuando esto sea así, al igual que ocurría con el prefijo `moz` de Mozilla) se define un buffer de audio que es rellenado con una señal sinusoidal. Una vez hecho esto, se asigna a un nodo de tipo `BufferSourceNode` y se conecta esta entrada a la salida del contexto de audio, además de fijar su reproducción inmediata (retardo de 0 ms en el método `noteOn`).

```
var context = new webkitAudioContext(),
    buffer =  context.createBuffer(1, BUFFER_SIZE, SAMPLE_RATE),
    bufferData = buffer.getChannelData(0),
    samples = (duration / 1000) * SAMPLE_RATE;

for (i = 0; i < samples; i++) {
  bufferData[i] = Math.sin(pitch * PI_2 * i / SAMPLE_RATE);
}

var source = context.createBufferSource();
source.buffer = buffer;
source.connect(context.destination);
source.noteOn(0);
```

Listado 3.10: Escritura de audio desde JavaScript a través de la Web Audio API.

Teniendo en cuenta que ambas API pretenden cubrir la misma necesidad, parece razonable pensar que solamente una de ellas será la implementación de referencia en el futuro. Dado el apoyo del W3C a la Web Audio API y la ventaja que supone el procesado de audio mediante código nativo frente a su implementación en JavaScript, esta API cuenta con ventaja en esta contienda. Sin embargo, como ya hemos podido comprobar anteriormente, tampoco existe una sola manera de hacer animaciones en HTML5, por lo que podría darse un caso similar con las API de audio, en el que convivan ambas implementaciones.

3.2.2 Procedimiento

El procedimiento seguido para poner a prueba la exactitud y precisión en la presentación de contenido auditivo por parte de las tecnologías evaluadas es el

siguiente: 1) la tecnología evaluada se emplea para generar un tono de 1 KHz de 200 ms de duración cada 1000 ms, en una secuencia en la que a cada 200 ms de tono le suceden 800 ms de silencio; 2) para cada tecnología se capturan 5 series independientes de 64 s de duración cada una; y 3) de cada captura se descartan los 2 primeros segundos y se toman las 100 muestras a partir de ese momento. De esta forma se obtienen 500 mediciones para cada tecnología.

Las tecnologías evaluadas son las siguientes: 1) DMDX 4.0.4.8 sobre Windows 7 SP1; 2) el elemento `audio` del estándar HTML5 con diferentes temporizadores JavaScript; 3) la Audio Data API para la generación de audio con diferentes temporizadores JavaScript; y 4) la Web Audio API con su propio temporizador (noteOn), tanto mediante la generación de audio desde JavaScript como mediante su descarga por AJAX. Todas las pruebas relativas a tecnologías web fueron realizadas tanto en Google Chrome 17 como en Mozilla Firefox 10 sobre los sistemas operativos Windows 7 Professional 32-bit edition Service Pack 1, Ubuntu Linux 10.04 LTS "Lucid Lynx" 32-bit y Mac OS X 10.7.3 "Lion".

La configuración de las pruebas para DMDX es muy similar a la empleada en los test para evaluar la exactitud y precisión de la presentación de contenidos visuales, con la salvedad de que en este caso se ha añadido una etiqueta `wav` al comiendo de cada ensayo con una referencia a un fichero WAV con un tono de 1 KHz de 200 ms de duración (formato RIFF *little-endian*, PCM de 16 bit mono a 44.100 Hz).

En lo que respecta a la preparación de las pruebas del elemento `audio` del estándar HTML5, se ha configurado varios *scripts* similares para las pruebas de la presentación de contenido visual mediante diferentes temporizadores de JavaScript, añadiendo el elemento `audio` a través de JavaScript durante la inicialización de la prueba.

En las pruebas relativas a la Audio Data API se ha sustituido el elemento `audio` por el código necesario para la generación mediante JavaScript de una señal auditiva igual a la cargada desde fichero WAV anteriormente descrito. En el listado 3.11 puede verse el código correspondiente a esta generación del tono de prueba.

Finalmente, las pruebas de la Web Audio API se han configurado igualmente empleando las pruebas anteriores de presentación de contenido visual con los diferentes mecanismos de temporización como base. El listado 3.12 muestra el código

```
var SAMPLE_RATE = 44100, PI_2 = Math.PI * 2, duration = 200,
    pitch = 1000, audio, buffer;

try {
  audio = new Audio();
  audio.mozSetup(1, SAMPLE_RATE);
  buffer = new Float32Array(SAMPLE_RATE);
  for (var i = 0, samples = duration * SAMPLE_RATE / 1000; i < samples; i++) {
    buffer[i] = Math.sin(pitch * PI_2 * i / SAMPLE_RATE);
  }
} catch (error) {
  window.alert('Audio Data API error: ' + error);
}
```

Listado 3.11: Generación del tono de prueba mediante la Audio Data API.

utilizado en estas pruebas, en el que se puede apreciar la creación del contexto de audio, la generación del contenido auditivo a través de un `BufferSourceNode` y su temporización mediante `noteOn`. El código relativo a las pruebas de la Web Audio API que emplean la carga del contenido auditivo mediante una solicitud AJAX es muy similar a este, donde se decodifica el audio cargado asíncronamente a través del método `decodeAudioData` del contexto de audio.

3.2.3 Resultados

Tras llevar a cabo las pruebas descritas, analizamos los retardos sufridos en la presentación del contenido auditivo y calculamos sus estadísticos descriptivos.

Al igual que sucediera con el contenido visual, los resultados de DMDX al presentar contenido auditivo son muy satisfactorios, con una media de -1,445 ms de retardo (i. e., una antelación a lo establecido de 1,445 ms de media) y una DT de 3,214 ms, fluctuando todas las mediciones de las pruebas de DMDX entre -10,975 ms de valor mínimo y 0,091 ms de valor máximo.

En cuanto a los resultados obtenidos por las pruebas con el elemento audio del estándar HTML5, resulta muy llamativo el retardo de entre 2 y 4,6 s que se da en Mozilla Firefox 10 sobre GNU/Linux (ver tabla 3.14). Exceptuando este deficiente comportamiento, el resto de resultados están dentro de lo esperado, con

```
var SAMPLE_RATE = 44100, BUFFER_SIZE = 16384, PI_2 = Math.PI * 2,
    interval = 1000, duration = 200, pitch = 1000, count = 500, ctx;

function start() {
  var buffer, bufferData, samples, i;
  ctx = new AudioContext();
  buffer = ctx.createBuffer(1, BUFFER_SIZE, SAMPLE_RATE);
  bufferData = buffer.getChannelData(0);
  samples = (duration / 1000) * SAMPLE_RATE;
  for (i = 0; i < samples; i++) {
    bufferData[i] = Math.sin(pitch * PI_2 * i / SAMPLE_RATE);
  }
  for (i = 0; i < count; i++) {
    play(buffer, ctx.currentTime + 1.0 * i);
  }
}

function play(data, time) {
  var source = ctx.createBufferSource(0);
  source.buffer = data;
  source.connect(ctx.destination);
  source.noteOn(time);
}
```

Listado 3.12: Configuración de la generación y temporización de contenido auditivo mediante la Web Audio API.

medias entre -0,4 y 6,31 ms y DT entre 0,58 y 35,6 ms. Los mejores resultados se obtuvieron con Mozilla Firefox 10 en Mac OS X mediante los temporizadores estándar de JavaScript (M: 0,002 ms, DT: 0,590 ms), mientras que los peores (sin contar los obtenidos por Mozilla Firefox 10 sobre GNU/Linux) corresponden a Mozilla Firefox 10 en Windows 7 mediante `postMessage` (M: 15,550 ms, DT: 112,708 ms).

Los resultados relativos a las pruebas realizadas en Mozilla Firefox 10 con la Audio Data API muestran bajos valores medios de error (medias entre 0,074 y 6,310 ms), pero una notable falta de precisión para las pruebas en las que se emplea `postMessage` como temporizador (DT entre 24,376 y 60,932 ms, frente a valores entre 2,158 y 7,078 ms para los otros dos mecanismos de temporización). Como puede apreciarse en la tabla 3.15, las mejores mediciones se obtuvieron empleando los temporizadores estándar de JavaScript sobre Windows 7 (M: 0,743 ms,

Temporizador	Agente de usuario	SO	N	Min	Max	M	DT
`setTimeout`	Chrome 17	Windows 7	500	-10,952	9,252	-0,42522	1,28526
		GNU/Linux	500	-15,397	22,925	0,06934	14,560525
		Mac OS X	500	-2,88	12,063	0,29311	4,796259
	Firefox 10	Windows 7	500	-13,197	38,163	2,54608	10,21789
		GNU/Linux	500	0,045	2006,417	998,96839	999,927741
		Mac OS X	500	-1,338	2,744	0,00163	0,589604
`requestAnimationFrame`	Chrome 17	Windows 7	500	-1,361	9,274	3,59125	4,484105
		GNU/Linux	500	-15,465	23,424	0,43256	14,977614
		Mac OS X	500	-2,88	23,651	4,14431	8,086377
	Firefox 10	Windows 7	500	-8,209	45,442	6,31338	14,356105
		GNU/Linux	500	0,045	2043,22	1014,73796	15,696201
		Mac OS X	500	-2,177	18,186	5,27605	7,221332
`postMessage`	Chrome 17	Windows 7	500	-10,975	9,025	-0,36503	1,728711
		GNU/Linux	500	-15,488	22,109	-0,41029	14,450069
		Mac OS X	500	-2,88	46,848	-0,33338	4,287987
	Firefox 10	Windows 7	500	-444,286	1104,762	15,55029	112,707901
		GNU/Linux	500	0,045	4615,238	1025,78263	1042,651729
		Mac OS X	500	-5,85	532,177	2,71211	35,612974

Tabla 3.14: Estadísticos descriptivos de los errores temporales en ms en la presentación de contenido auditivo mediante el elemento audio de HTML5.

DT: 2,158 ms), y las peores se obtuvieron al emplear `postMessage` sobre Windows 7 (M: 5,950 ms, DT: 60,932 ms).

La figura 3.14 muestra una comparativa entre los resultados obtenidos en las pruebas ejecutadas sobre Mozilla Firefox tanto para la Audio Data API como para el uso del elemento `audio` del estándar HTML5. En ellas se puede apreciar como la nueva API supera en todos los casos el comportamiento del elemento estándar, salvo en el caso de los temporizadores estándar de JavaScript sobre Mac OS X.

Tras realizar un ANOVA 2 (Agente de usuario: Google Chrome 17, Mozilla Firefox 10) x 3 (SO: Windows 7, GNU/Linux, Mac OS X) x 3 (Temporizador: `setTimeout`, `requestAnimationFrame`, `postMessage`) x 5 (Serie: 1 a 5) x 2 (Sonido: silencio, tono) de medidas independientes, no se encuentran diferencias significativas en los resultados entre los mecanismos de temporización que emplean la cola de eventos de JavaScript al analizarlos en su conjunto. Cuando se eliminan del análisis los resultados anómalos encontrados en Mozilla Firefox 10

Temporizador	SO	N	Min	Max	M	DT
`setTimeout`	Windows 7	500	-0,998	9,002	0,07433	2,158112
	GNU/Linux	500	-0,34	17,46	6,30916	6,72188
	Mac OS X	500	-2,472	9,478	0,22367	3,580299
`requestAnimationFrame`	Windows 7	500	-0,998	9,025	1,07338	3,39774
	GNU/Linux	500	-1,247	18,345	4,92118	5,605473
	Mac OS X	500	-2,472	21,088	2,79905	7,077752
`postMessage`	Windows 7	500	-10,975	798,299	5,94902	60,932485
	GNU/Linux	500	-2,653	947,959	5,05161	55,311594
	Mac OS X	500	-2,472	450,249	1,59224	24,375849

Tabla 3.15: Estadísticos descriptivos de los errores temporales en ms en la presentación de contenido auditivo mediante la Audio Data API en Mozilla Firefox 10.

sobre GNU/Linux al usar el elemento `audio` del estándar HTML5, estas diferencias son parcialmente significativas ($F(2, 11999) = 5{,}080$, $p = 0{,}06$, $\eta_p^2 = 0{,}001$).

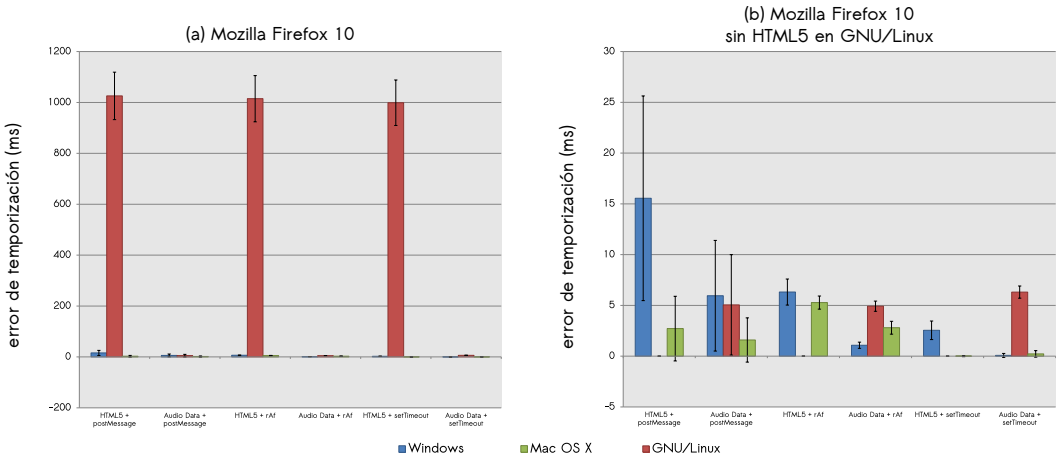

Figura 3.14: Comparativa entre los resultados obtenidos por la Audio Data API y el elemento estándar de HTML5: (a) Mozilla Firefox 10, (b) Mozilla Firefox 10 sin los datos de HTML5 sobre GNU/Linux.

Por último, los resultados de las pruebas realizadas mediante la Web Audio API son los más consistentes de entre todos los analizados. No solamente su exactitud es

elevada ya que los valores medios se encuentran todos en torno a los -0,4 ms, sino que su precisión también lo es ya que las DT obtenidas se encuentran en torno a los 0,151 y 0,726 ms (ver tabla 3.16). Las diferencias entre las distintas condiciones evaluadas (sistema operativo y mecanismo generador del contenido auditivo) no son significativas por lo que nos encontramos ante una tecnología que se comporta adecuadamente independientemente del sistema subyacente.

Audio	SO	N	Min	Max	M	DT
AJAX	Windows 7	500	-0,998	5,057	-0,43306	0,581366
	GNU/Linux	500	-0,975	0,091	-0,44517	0,517362
	Mac OS X	500	-0,907	0,023	-0,44472	0,150813
`BufferSourceNode`	Windows 7	500	-1,02	0,113	-0,44526	0,542153
	GNU/Linux	500	-0,907	15,397	-0,41315	0,726393
	Mac OS X	500	-0,93	0,045	-0,44522	0,204533

Tabla 3.16: Estadísticos descriptivos de los errores temporales en la presentación de contenido auditivo mediante la Web Audio API en Google Chrome 17.

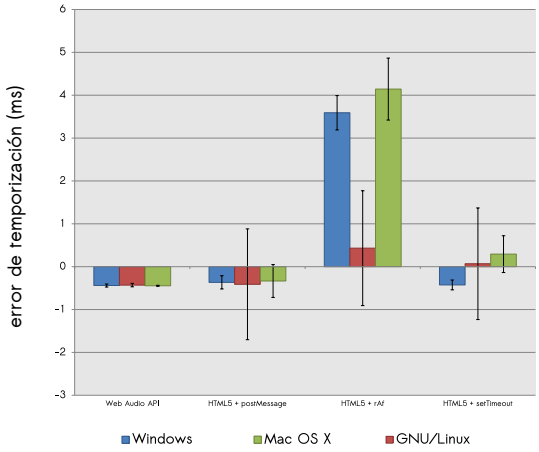

Figura 3.15: Comparativa entre los resultados obtenidos por la Web Audio API y el elemento estándar de HTML5 en Google Chrome 17.

Si comparamos los resultados de Google Chrome 17 al emplear el elemento

estándar de HTML5 con los obtenidos por este mismo agente de usuario al emplear la Web Audio API podemos realizar dos observaciones: 1) los resultados obtenidos mediante la Web Audio API son más precisos en todas las condiciones, y 2) el uso de `requestAnimationFrame` como mecanismo de temporización de presentaciones de audio no es recomendable, ya que esta API ha sido diseñada para presentaciones de contenido visual, dependientes de la tasa de refresco del monitor (ver figura 3.15).

3.2.4 Conclusiones

A tenor de los resultados presentados previamente se puede afirmar que los esfuerzos realizados por el Audio Working Group del W3C en la definición de la Web Audio API y por parte de Google en su implementación en Google Chrome han conseguido que sea posible presentar contenido auditivo en aplicaciones con altos requisitos de exactitud y precisión temporal. Tanto es así, que los resultados obtenidos por esta tecnología web se encuentran a la par de los obtenidos por tecnologías offline especializadas como DMDX.

Asimismo, como ya se ha comentado con anterioridad, resulta preocupante el pésimo rendimiento obtenido al emplear el elemento audio de HTML5 sobre Mozilla Firefox 10 sobre GNU/Linux (retardos entre 2 y 4,6 s de media), que lo inhabilita como alternativa técnicamente viable.

En relación a los mecanismos de temporización, el uso de un sistema de sincronización de audio independiente de la cola de eventos de JavaScript como el empleado en la Web Audio API es la opción más recomendable, no solo por los buenos resultados obtenidos sino por su inmunidad frente a eventuales sobrecargas en la gestión de eventos de JavaScript. De entre los mecanismos de temporización evaluados que sí hacen uso de esta cola de eventos, el menos adecuado es `requestAnimationFrame`, dado que la referencia temporal para esta API es la tasa de refresco visual por lo que no debería emplearse para audio no sincronizado con ella.

3.3 Conclusiones generales

A lo largo de este capítulo hemos puesto a prueba un amplio abanico de tecnologías para la presentación de contenidos audiovisuales. Al analizar su exactitud y precisión presentando contenidos visuales, hemos podido constatar que el software especializado en esta labor (E-Prime, DMDX, PsychoPy) cumple con los elevados requisitos temporales impuestos. Sin embargo, los investigadores que empleen estos programas deben tener en cuenta las particularidades y ajustes necesarios en cada uno de ellos para maximizar el ajuste a esos requisitos. Así, es conveniente recordar detalles como que el tiempo de preparación de cada presentación en E-Prime no es despreciable y sus desarrolladores aconsejan sustraer 10 ms al tiempo de presentación, o que las animaciones con intervalos inferiores a 50 ms en DMDX deben situarse dentro del mismo ensayo en el fichero de configuración, por citar dos ejemplos.

En el caso de las tecnologías web basadas en máquinas virtuales evaluadas (Java, Flash y Silverlight) los resultados son también satisfactorios (media de marcos perdidos inferior a 1 con DT en torno a 0,5 para Flash y Silverlight, valores ligeramente superiores para Java). No obstante, a pesar de su exactitud y precisión temporal, comparable con las obtenidas mediante tecnologías offline, la pérdida de cuota de mercado en la Web de estas tecnologías en favor las tecnologías propuestas por los nuevos estándares web en torno a la especificación HTML5 es inexorable.

De entre las nuevas tecnologías web analizadas, las animaciones declarativas con CSS han resultado ser las más adecuadas para aquellos casos en los que el intervalo de animación de 50 ms o mayor (medias de marcos perdidos entre 0,00 y -0,06 y DT entre 0,459 y 1,107), ya que además de ser una tecnología estable en el tiempo, no precisa de la cola de eventos de JavaScript para su funcionamiento y es independiente del grado de saturación de la misma. La combinación de SVG y SMIL es otra opción para la generación de animaciones declarativas, pero su implementación dista de ser completa en los agentes de usuario evaluados. En el caso de las tecnologías web de animaciones procedimentales analizadas (Canvas, SVG y WebGL con `requestAnimationFrame`) su rendimiento es comparable al de tecnologías web previas, por lo que su uso está recomendado frente a éstas dado que se trata de tecnologías estándar y en constante mejora.

Al combinar estas nuevas tecnologías web de presentación de contenido visual con las nuevas API de reproducción y generación de contenido auditivo hemos podido constatar que existen alternativas (v. gr., Web Audio API en Google Chrome 17) que igualan e incluso superan en exactitud y precisión a las tecnologías offline evaluadas (DMDX sobre Windows 7 SP 1).

Si se toman estos resultados en conjunto es posible afirmar que el grado de desarrollo actual de las tecnologías web estándar permite la implementación de aplicaciones web con elevados requisitos temporales en la presentación de contenido audiovisual. A lo largo del próximo capítulo evaluaremos si estas mismas tecnologías web estándar son capaces de registrar la interacción del usuario de forma exacta y precisa.

"If the user can't use it, it doesn't work"

Susan Dray

CAPÍTULO

4

Análisis de tecnologías para el registro preciso de la interacción del usuario

A lo largo de este capítulo analizamos el grado de exactitud y precisión del registro de la interacción del usuario alcanzable mediante tecnologías web. Para ello, repasamos las distintas aproximaciones realizadas hasta la fecha para la generación de la interacción de usuario, desde el uso de sujetos experimentales hasta el empleo de software especialmente diseñado con este propósito. Una vez hecho esto, definimos un procedimiento de medición para obtener unos resultados que analizamos estadísticamente. De estos resultados se extraen las conclusiones que cierran el capítulo.

El objetivo de este análisis es análogo al del capítulo anterior: disponer de suficiente base empírica como para obtener conclusiones sobre la plausibilidad del uso de tecnologías web estándar para el registro preciso de la interacción del usuario. De la misma forma, el procedimiento seguido pretende minimizar los errores de medición no atribuibles a la propia tecnología medida.

Si en el análisis de la exactitud y precisión en la presentación de contenidos audiovisuales los temporizadores (*timers*) han sido los elementos más importantes,

en el análisis del registro de la interacción del usuario cobrarán especial importancia las funciones de tiempo (*timing functions*) empleadas. La diferencia entre ambos ya ha sido explicada con anterioridad, pero puede resumirse recurriendo a un símil sencillo: los temporizadores funcionan como alarmas que se disparan cuando llega el momento configurado, mientras que las funciones de tiempo funcionan como cronómetros que se limitan a constatar el paso del tiempo. Ambos son fundamentales en el desarrollo de aplicaciones web con altos requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario.

4.1 Generación de la interacción de usuario

Antes de analizar la exactitud y precisión de las tecnologías offline y online en el registro de la interacción del usuario conviene seleccionar de entre las alternativas disponibles un procedimiento para la generación de esa interacción de usuario. A pesar de que todas ellas están sujetas a limitaciones, también aportan ciertas ventajas que las diferencian del resto y las ajustan a determinados escenarios. Por esta razón, repasaremos los diferentes enfoques llevados a cabo hasta la fecha y seleccionaremos el que mejor se adapte a las necesidades de nuestro análisis.

4.1.1 Uso de participantes voluntarios

Si se trata de obtener un patrón de interacción de usuario lo más ajustado posible al patrón real, el enfoque más evidente es recurrir a la colaboración de participantes voluntarios. Este es un procedimiento muy habitual en estudios de ciencias del comportamiento. Así, Reimers y Stewart (2007) emplearon para analizar el grado de exactitud y precisión temporal de la tecnología Flash a estudiantes de grado de la Universidad de Warwick que colaboraron a cambio de créditos. Keller y cols. (2009) también emplearon participantes voluntarios para poner a prueba su paquete WebExp destinado a la creación de experimentos online y desarrollado en Java.

Además de la colaboración de personas voluntarias próximas al ámbito local del experimentador, la propia Web puede verse como un enorme laboratorio de experimentación (Nosek, 2005), como ya hemos comentado con anterioridad.

La principal ventaja de este enfoque es su alta validez externa, es decir, la gran capacidad de generalización de las conclusiones del estudio, ya que se replica fielmente la situación natural. Sin embargo, su mayor inconveniente reside en la baja fiabilidad de la respuesta de los participantes, dado que los efectos de la práctica o las dificultades para encontrar grupos de personas comparables pueden limitar las posibilidades de replicar íntegramente un estudio.

Por estas razones consideramos este enfoque como complementario o validador de otros enfoques de más bajo nivel. En nuestro estudio contamos con la colaboración de participantes voluntarios una vez que la exactitud y precisión de las tecnologías web empleadas han sido comprobadas a través de procedimientos menos susceptibles de sufrir la influencia de variables extrañas (a través de los fotosensores del BBTK, con una resolución temporal varios órdenes de magnitud superior a la requerida en este tipo de estudios).

4.1.2 Uso de hardware especializado

Motivados por las razones esgrimidas en la sección anterior, algunos investigadores han preferido emplear hardware especializado para simular la interacción del usuario y gozar por tanto de un mayor control sobre la misma. Tal es el caso de aquellos que emplean las funcionalidades de generación de respuestas del BBTK (Plant y cols., 2004) para enviar a través del puerto paralelo una señal (Schneider y cols., 2002) o bien activar un pulsador situado en un dispositivo de entrada conectado al ordenador que lleva a cabo el experimento (Plant y Turner, 2009). Además de estas propuestas, otros investigadores han conectado la entrada de fotosensores a actuadores (Häusler, Sommer, y Chroust, 2007) o solenoides que generan pulsaciones sobre un teclado (Neath y cols., 2011) para disponer de un participante automatizado capaz de interactuar de forma consistente y repetible con la aplicación o tecnología estudiada.

La principal ventaja de este enfoque frente al anterior es su capacidad de ser automatizado, lo que proporciona al experimentador la independencia necesaria para replicar tantas veces como sea necesario un estudio –modificando los valores que considere oportunos– y evaluar su impacto en el resultado final. Lamentablemente este enfoque tampoco está exento de problemas. En los casos en los que

se emplea una señal desde un generador de señales externo, no se está replicando exactamente la interacción del usuario, por lo que podría darse el caso de que una fuente de errores afectara al dispositivo de entrada que los usuarios emplean pero no tuviera efecto sobre el sistema empleado para enviar y recibir la señal desde el generador externo. Lo contrario también podría darse (i. e., fuentes de error que afecten al envío y recepción de señales pero no tengan efecto sobre los dispositivos de entrada de usuario). De esta forma se obtendrían medidas no comparables con el enfoque anterior. Esto no sucede en el caso en el que los generadores de señales se conectan a un pulsador de un dispositivo de entrada, pero no siempre es sencillo emplear esta solución (v. gr., en la interacción con una pantalla táctil) y es muy dependiente de las características del dispositivo concreto empleado en cuanto a errores sistemáticos y aleatorios. Por último, el uso de solenoides para actuar sobre dispositivos de entrada de usuario puede aplicarse a muchos tipos de situaciones, pero al igual que en el caso anterior, no evitan que los errores temporales de esos dispositivos de entrada afecten a los resultados obtenidos.

4.1.3 Uso de hardware común

Dado que el hardware especializado resulta costoso en tiempo y dinero, algunos investigadores han evaluado la posibilidad de utilizar hardware de propósito general disponible en la mayoría de equipos para la generación de la interacción del usuario. Tal es el caso de Eichstaedt (2001), entre otros, que evaluó la precisión temporal de *applets* Java mediante el uso de la tasa de repetición tipomática en un teclado convencional. Esta repetición tipomática es capaz de generar automáticamente un gran número de pulsaciones de teclas por segundo. Como ya hemos explicado en el capítulo 2, para activarla, basta con mantener una tecla durante un tiempo superior al retardo tipomático. Una vez superado este tiempo, se generará una pulsación de tecla automáticamente en función de la tasa de repetición tipomática hasta que se deje de pulsar esa tecla o se pulse otra.

A pesar de que a simple vista pudiera parecer un método sencillo y eficaz para la generación de interacción de usuario constante y configurable sin la necesidad de equipamiento sofisticado ni configuraciones complejas, este enfoque no es recomendable por varias razones.

La primera de ellas tiene que ver con la falta de consistencia en la implementación de esta funcionalidad en los diferentes tipos de teclado. Así, en el caso de los teclados AT-PS/2, este comportamiento está definido en el protocolo de comunicación (Chapweske, 2003b), pero en los dispositivos USB es el sistema operativo quien lo implementa empleando la tasa de envío de paquetes y el número de paquetes recibidos para inferir la duración de la pulsación prolongada de la tecla. Por lo tanto, no en todos los casos esta repetición tipomática está generada por un dispositivo externo, sino que en muchos de ellos es necesaria la colaboración de propio sistema estudiado.

La segunda de las razones tiene que ver con la escasa calidad del equipamiento hardware que suele emplearse para la fabricación de los teclados de propósito general. Al contrario de otros dispositivos de entrada como *joysticks* o pulsadores, los teclados son dispositivos de los que se espera una tasa de respuesta baja (por debajo de las 500 ppm, pulsaciones por minuto) y, por tanto, sus fabricantes no ponen un especial cuidado a la hora de seleccionar sus componentes.

Finalmente, la popularización de teclados inalámbricos incorpora a este escenario una fuente extra de posibles errores ya que el medio electromagnético está más expuesto a interferencias que una conexión mediante cable. Por todas estas razones consideramos que el uso de la tasa de repetición tipomática de los teclados no debería emplearse para evaluar la exactitud y precisión de una aplicación puesto que no es posible asegurar la precisión ni la exactitud de la señal generada de esta forma.

4.1.4 Uso de software especializado

Teniendo en cuenta los potenciales problemas de los enfoques basados en hardware comentados, varios desarrolladores han creado sistemas software de generación de la interacción del usuario de forma automatizada. A los sistemas sencillos de inyección de pulsaciones de teclas o eventos de ratón como uinput o Keypresser se unen completas suites de automatización de la entrada de usuario pensadas para el entorno Web como Selenium (Badle y cols., 2012).

El hecho de que la generación de la interacción del usuario no dependa del hardware hace que este enfoque evite todos los errores y latencias propias del hard-

ware. Sin embargo, su principal problema es que el propio sistema de generación de la interacción de usuario puede afectar al rendimiento de la aplicación evaluada, y viceversa (una eventual sobrecarga en la aplicación evaluada y otro componente software del sistema podría afectar a la generación de la interacción de usuario). Por este motivo resulta fundamental que este tipo de sistemas se sitúen en un nivel de la arquitectura del software que les evite verse afectados por otros componentes software (v. gr., a nivel de núcleo) y que los mecanismos de temporización empleados sean a su vez lo más inmunes posible a la sobrecarga de las aplicaciones de usuario.

4.2 Procedimiento

De entre todas las opciones planteadas en la sección anterior para la generación de la interacción de usuario que permita analizar la exactitud y precisión de su registro por parte de distintas tecnologías offline y online se ha decidido emplear un controlador de teclado propio que emplee temporizadores de alta resolución y genere eventos de teclado configurables mediante parámetros. Las razones que motivan esta decisión son las siguientes: 1) ningún dispositivo de entrada convencional (teclado, ratón o pantalla táctil) es capaz de ofrecer retardos inferiores a 1 ms (Bhalla y Bhalla, 2010; Crosbie, 1990; Damian, 2010; Plant y cols., 2003; Plant y Turner, 2009; Segalowitz y Graves, 1990) por lo que suponen una fuente de errores de medición considerable; 2) el uso de puertos de comunicación como el puerto paralelo exige la adaptación de la aplicación offline u online a este mecanismo de entrada, lo que en muchos casos puede resultar inviable; 3) el hecho de que este controlador de teclado sea ejecutado a nivel de núcleo protege su funcionamiento de eventuales sobrecargas a nivel de usuario; 4) los temporizadores de alta resolución empleados (*High Resolution Timers*) garantizan una exactitud y precisión por debajo de 1 ms, 5) la función de tiempo empleada para el registro de las marcas de tiempo (`ktime`) tiene un crecimiento estrictamente monotónico y una resolución, precisión y exactitud por debajo de 1 ms, y 6) a todos los efectos, las aplicaciones offline y online obtendrán los eventos de teclado generados automáticamente como si fueran eventos generados por un usuario.

La elección de GNU/Linux como plataforma de desarrollo del controlador de teclado está motivada por la gran cantidad de documentación disponible para el desarrollo de módulos para el núcleo Linux (Bovet y Cesati, 2005), la sencillez de su desarrollo y las amplias posibilidades para su puesta a punto y testeo a través de distintos tipos de máquinas virtuales, depuradores del núcleo y sistemas de registro.

El procedimiento seguido ha sido el siguiente: 1) carga del controlador de teclado Metronome LKM (descrito más adelante) para la generación precisa y exacta de pulsaciones de teclas automáticamente con los parámetros correspondientes (en su configuración por defecto, el controlador permanece inactivo nada más ser cargado); 2) ejecución de la tecnología offline u online puesta a prueba; 3) activación del controlador a través de una petición al sistema (mediante la tecla PetSis o SysReq); 4) captura de datos a través de la tecnología estudiada y del propio controlador de teclado; y 5) análisis de los resultados obtenidos. Este procedimiento se ha seguido empleando diferentes cadencias de pulsación de teclas (1000, 500, 100, 50, 10, 5 y 1 ms) hasta tomar 10.000 muestras para cada tecnología analizada.

Al tratarse Metronome LKM de un controlador de teclado para GNU/Linux, se desarrolló una aplicación offline de registro de teclado en el sistema X-Window (Logkeys X-Window, descrita más adelante) para obtener una medición con una precisión por debajo de 1 ms de la cadencia de eventos de teclado registrada por ella. De esta forma, es posible comparar estos valores con los obtenidos por una sencilla aplicación web de registro del teclado a través de las marcas de tiempo de eventos DOM (Pixley, 2000) desde JavaScript ejecutada tanto en Google Chrome 17 como en Mozilla Firefox 10. Esta aplicación se limita a asignar al evento `keypress` una función que toma el valor de la marca temporal del evento DOM (propiedad `timeStamp`) y le resta el valor de la marca temporal del evento anterior, obteniendo el periodo de tiempo transcurrido entre ambos.

4.2.1 Metronome LKM

Metronome LKM es un controlador para el núcleo Linux 2.6.21 o posterior que emplea temporizadores de alta resolución para la generación exacta y precisa de eventos de teclado. Este controlador es configurable por parámetros en el momento de su carga o bien mediante el sistema de ficheros virtual `sys` y gestionable

mediante una petición al sistema a través de la tecla PetSis (SysReq). Además de generar los eventos de teclado requeridos, registra cada uno de ellos mediante funciones de tiempo con resolución de ns (`ktime`) en el fichero de registro del núcleo.

El código de Metronome LKM se encuentra recogido en el apéndice A. Como puede observarse, existen 4 parámetros que pueden modificarse en tiempo de ejecución en este controlador: 1) `metronome_delay`, que indica la cadencia en la generación de la pulsación de teclas, inicialmente tiene un valor de 1×10^9 ns (1000 ms); 2) `metronome_key`, que indica cuál será el evento de teclado que se generará cada vez que se cumpla el plazo fijado en el temporizador de Metronome LKM, inicialmente tiene un valor que se corresponde con la barra espaciadora (`KEY_SPACE`); 3) `metronome_sysrq`, que indica cuál será la combinación de teclas junto a PetSis para disparar la petición al sistema que habilite o deshabilite la generación de eventos de teclado, inicialmente tiene el valor «a», por lo que habrá que pulsar PetSis+A para disparar esta petición al sistema; y 4) `metronome_status`, que indica el estado activo o inactivo de la generación de eventos de teclado, inicialmente tiene el valor 0, que corresponde con el estado inactivo. De esta forma, una vez cargado el controlador podrá activarse mediante la combinación de teclas definida anteriormente o bien a través del comando:

```
echo 1 > /sys/module/metronome/parameters/metronome_status
```

Lo mismo sucede para desactivarlo, pulsando de nuevo la combinación de teclas configurada o a través del comando:

```
echo 0 > /sys/module/metronome/parameters/metronome_status
```

Una vez cargado, Metronome LKM ejecuta su función `metronome_init` que realiza lo siguiente: 1) reservar los recursos del sistema necesarios para registrar un nuevo dispositivo a través de `input_allocate_device`; 2a) si esta reserva del dispositivo de entrada se ha realizado con éxito, Metronome LKM registra finalmente este dispositivo (a través de `input_register_device`) después de configurarlo adecuadamente; 2b) si la reserva de recursos para el dispositivo de

entrada no tuvo éxito, se informa del error a través de un mensaje del núcleo y se sale con error; 3a) en el caso de que el registro del dispositivo de entrada tenga éxito, Metronome LKM registra la petición al sistema que activa o desactiva la generación de eventos de teclado y configura el temporizador de alta resolución monotónico (a través de la función `hrtimer_init`) teniendo en cuenta el parámetro `metronome_delay`; 3b) si el registro del dispositivo no tuvo éxito, se liberan los recursos solicitados y se informa del error a través de un mensaje del núcleo; y 4) una vez reservados todos los recursos y registrados todos los componentes de Metronome LKM, será la función `metronome_hrt_callback` la encargada de comprobar que no ha habido un retraso en la gestión del temporizador (a través de la función `hrtimer_forward`), generar los eventos de teclado necesarios (mediante las funciones `input_report_key` e `input_sync`), registrar el momento en el que fueron enviados (mediante la función `ktime_get`) y reconfigurar el temporizador de alta resolución (retornando el valor `HRTIMER_RESTART`).

Si en algún momento de su ejecución el usuario empleara la combinación de teclas configurada para la modificación del estado de Metronome LKM, se ejecutaría la función `sysrq_handle_metronome` encargada de este propósito.

4.2.2 Logkeys X-Window

Logkeys X-Window es una aplicación para el entorno gráfico X-Window desarrollada *ex profeso* para obtener un registro de la cadencia de eventos de teclado. Para ello se sirve de las siguientes funciones: 1) `XOpenDisplay`, `DefaultScreen` y `RootWindow`, para acceder a la ventana raíz del entorno gráfico X-Window; 2) `XGrabKeyboard`, para capturar los eventos de teclado; 3) `gettimeofday`, para obtener marcas temporales (resolución de μs); 4) `XPending` y `XNextEvent`, para capturar los eventos que llegan a la ventana raíz de X-Window; y 5) si alguno de esos eventos es de tipo `KeyPress`, se emplea `XLookupKeysym` para obtener el código de la tecla pulsada. Al finalizar su registro, Logkeys X-Window cierra el fichero de registro y libera tanto el teclado como la ventana raíz de X-Window con las funciones `XUngrabKeyboard` y `XCloseDisplay` respectivamente.

4.3 Resultados

Los resultados de las pruebas llevadas a cabo mediante Metronome LKM y Logkeys X-Window se muestran en la tabla 4.1, donde todos los valores se encuentran reflejados en ms. Como se puede observar, la media de los valores obtenidos mediante Metronome LKM se sitúa muy por debajo de 1 ms, en torno a 1 ns, con una DT de 4,499 μs en el peor de los casos. Los valores medios obtenidos por Logkeys X-Window también se sitúan por debajo de 1 ms (entre 4,599 y 0,1777 μs), pero las DT obtenidas son sensiblemente mayores que las obtenidas a nivel de núcleo (entre 9,127 μs y 0,112 ms). Estos valores sirven de referencia para las pruebas realizadas con tecnologías web estándar sobre Google Chrome 17 y Mozilla Firefox 10.

Registro	**Intervalo**	**M**	**DT**	**N**
Núcleo	1	-0,00000007	0,004660847	10000
	5	-0,00000995	0,00449937	10000
	10	0,00000003	0,000192422	10000
	50	-0,00000093	0,00103286	10000
	100	-0,00000555	0,000930028	10000
	500	-0,00001239	0,001294279	10000
	1000	-0,00000129	0,001002863	10000
Logkeys	1	0,0000012	0,058875759	10000
X-Window	5	0,0041082	0,112822616	10000
	10	-0,0008382	0,004214332	10000
	50	0,0005468	0,025032313	10000
	100	0,0045985	0,094976506	10000
	500	-0,0034937	0,009127118	10000
	1000	-0,0001777	0,046649194	10000

Tabla 4.1: Estadísticos descriptivos de los errores temporales en el registro de eventos de teclado mediante Logkeys X-Window (en ms).

En lo que respecta a Google Chrome, los resultados de las pruebas realizadas

se muestran en la tabla 4.2, donde igualmente todos los valores corresponden a ms. Lo primero que llama la atención es que, si bien los valores medios obtenidos por Metronome LKM (núcleo) se sitúan también muy próximos a 1 ns, su DT es mayor en todos los casos al compararse con los valores de Metronome LKM en las pruebas realizadas mediante Logkeys X-Window. Teniendo en cuenta que este hecho se repite en los resultados de Metronome LKM en las pruebas sobre Mozilla Firefox 10, es posible que esta pérdida de precisión temporal en la generación de eventos de teclado pueda verse afectada por la mayor carga para el sistema que supone la ejecución de la aplicación web sobre un agente de usuario frente a la sencillez de Logkeys X-Window. De cualquier manera, todos los valores se sitúan por debajo de 1 ms, más allá de la resolución disponible en las marcas de tiempo de los eventos DOM actuales (propiedad `timeStamp` del tipo `DOMTimeStamp`, que corresponde a un `unsigned long long` para almacenar un valor en ms). Las medias y DT de las pruebas empleando las marcas temporales de los eventos DOM en Google Chrome también se sitúan por debajo de 1 ms en todos los casos, lo que es congruente con la consideración de este mecanismo de registro de la interacción de usuario como apto para el tipo de aplicaciones web con altos requisitos temporales.

Por último, la tabla 4.3 muestra los resultados de las pruebas realizadas en Mozilla Firefox 10 (en ms). Como se puede observar, los resultados de Metronome LKM son muy similares, lo que sugiere que la carga del sistema al ejecutar la aplicación web basada en eventos DOM es semejante tanto en Google Chrome 17 como en Mozilla Firefox 10. En cuanto a los valores recogidos por la aplicación web, por encima de los 10 ms de intervalo no se aprecia ningún error de medición debido a la escasa resolución (de 1 ms) de las marcas temporales de los eventos DOM. Por debajo de los 10 ms de intervalo sí se encuentran algunos errores de medición aislados que sitúan las medias por debajo de 1 μs y las DT por debajo de 1 ms. Este hecho refrenda lo apuntado por los resultados en las pruebas sobre Google Chrome 17 y confirma la viabilidad del uso de las marcas temporales de los eventos DOM como mecanismo preciso para el registro de la interacción del usuario en aplicaciones web basadas en tecnologías estándar.

Registro	Intervalo	M	DT	N
Núcleo	1	-0,00139985	0,0058273	10000
	5	0,00000803	0,030747087	10000
	10	-0,00000674	0,014677891	10000
	50	0,00000695	0,015569002	10000
	100	0,00000047	0,013899963	10000
	500	-0,00000002	0,038588317	10000
	1000	0,00000572	0,01257442	10000
Google	1	0	0,034642748	10000
Chrome 17	5	-0,0007	0,041227175	10000
	10	0,1705	0,409690779	10000
	50	0,4025	0,644309913	10000
	100	-0,0001	0,156532553	10000
	500	0,0118	0,22195875	10000
	1000	0	0,345560329	10000

Tabla 4.2: Estadísticos descriptivos de los errores temporales en el registro de eventos de teclado mediante eventos DOM en Google Chrome 17 (en ms).

4.4 Conclusiones

Teniendo en cuenta los resultados mostrados previamente, podemos concluir que existe un mecanismo de registro de la interacción del usuario basado en tecnologías web estándar con una exactitud y precisión por debajo de 1 ms, lo que lo hace apto para el desarrollo de aplicaciones web con altos requisitos temporales.

Sin embargo, estos resultados también muestran una limitación actual en este mecanismo: su escasa resolución (ms) puede enmascarar ligeras variaciones que se estén produciendo a más bajo nivel, por lo que sería recomendable ampliar esta resolución hasta 1 μs o incluso más allá. Por esta misma razón, algunos desarrolladores de proyectos de medición temporal (v. gr., jsPerf, Benchmark.js) han optado por emplear soluciones alternativas como incluir un *applet* Java que exponga la función de tiempo `nanoTime`, con resolución de ns, a JavaScript o valerse de ex-

Registro	Intervalo	M	DT	N
Núcleo	1	-0,00030139	0,005213274	10000
	5	-0,00000209	0,011312149	10000
	10	0,00000035	0,024555596	10000
	50	0,00000528	0,028443645	10000
	100	0,00000495	0,014243673	10000
	500	-0,00000114	0,046871384	10000
	1000	0,00000114	0,018916281	10000
Mozilla	1	-0,0005	0,084264228	10000
Firefox 10	5	0,0001	0,01	10000
	10	0,0005	0,110909784	10000
	50	0	0	10000
	100	0	0	10000
	500	0	0	10000
	1000	0	0	10000

Tabla 4.3: Estadísticos descriptivos de los errores temporales en el registro de eventos de teclado mediante eventos DOM en Mozilla Firefox 10 (en ms).

tensiones propietarias como `chrome.Interval`, que devuelve el número de μs transcurridos desde el 1 de enero de 1970.

Afortunadamente, este tipo de soluciones alternativas no son necesarias gracias a la API High Resolution Time (Mann, 2012), que proporciona una función de tiempo monotónicamente creciente con una resolución por debajo de 1 ms (emplea el tipo `DOMHighResTimeStamp`, que representa un valor en ms con una precisión de al menos 1 μs) y no sujeta a problemas derivados de ajustes o cambios en el reloj del sistema (ver listado 4.1).

Por lo tanto, sería muy recomendable que los eventos DOM registraran sus marcas temporales empleando el tipo `DOMHighResTimeStamp` de cara a poder comparar esos valores con los obtenidos por API modernas que utilizan ese tipo de marcas temporales, como la High Resolution Time API (Mann, 2012) o la API pa-

```
1 partial interface Performance {
2   DOMHighResTimeStamp now();
3 };
```

Listado 4.1: Definición del método `now` en la API High Resolution Time.

ra el control temporal de animaciones procedimentales (Robinson y McCormack, 2012), entre otras. Por esta razón, realizamos una propuesta de mejora en este sentido durante el proceso de aceptación de propuestas del Working Group de DOM Events del W3C para la versión 3 de los eventos DOM. La evolución de la propuesta realizada y su implementación se encuentran detallados en el apéndice B.

4.5 Conclusiones generales

A lo largo del presente capítulo y el anterior hemos obtenido, entre otras, las siguientes conclusiones: 1) con intervalos de animación iguales o superiores a 50 ms, las animaciones declarativas en CSS (disponibles a partir de Google Chrome 4, Mozilla Firefox 5 e Internet Explorer 10, entre otros agentes de usuario) son la tecnología web estándar más recomendable para la presentación exacta y precisa de contenido audiovisual; 2) la Web Audio API (disponible en Google Chrome 10 y versiones superiores) permite la generación y temporización de contenido auditivo con una precisión y exactitud por debajo de 1 ms; y 3) el registro de la interacción del usuario mediante las marcas de tiempo de eventos DOM tiene una precisión y exactitud por debajo de 1 ms tanto en Google Chrome 17 como en Mozilla Firefox 10.

Considerando estos resultados en su conjunto, se puede concluir que es posible definir un conjunto de tecnologías web estándar compatible que permita la presentación exacta y precisa de contenido audiovisual y el registro de la interacción del usuario. Por ello, a lo largo del siguiente capítulo se explicarán los detalles de implementación de una aplicación web con altos requisitos temporales desarrollada a partir de este conjunto de tecnologías estándar y su puesta a prueba en varios estudios realizados con la colaboración de participantes voluntarios.

"The fundamental principle of science, the definition almost, is this: the sole test of the validity of any idea is experiment"

Richard P. Feynman

CAPÍTULO

5

Desarrollo de una aplicación experimental con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario

PREVIO al estudio de los detalles del desarrollo de una aplicación experimental con altos requisitos temporales, resulta adecuado repasar cuáles son las tecnologías fundamentales para cumplir con esos requisitos.

Como hemos podido comprobar en capítulos anteriores, los sistemas operativos actuales que se ejecutan sobre arquitecturas IBM-PC disponen de funciones de tiempo y mecanismos de temporización de alta exactitud, precisión y resolución.

Las funciones de tiempo más recomendables para el desarrollo de aplicaciones offline con altos requisitos temporales son aquellas que se valen de los temporizadores hardware de alta precisión, tales como `QueryPerformanceFrequency`

y `QueryPerformanceCounter` en sistemas operativos Microsoft, o la función `gettimeofday` en sistemas operativos compatibles con UNIX (GNU/Linux o Mac OS X, entre otros). Para el desarrollo de aplicaciones web basadas en estándares, conviene recordar que la resolución de las funciones de tiempo propias de JavaScript se limita a 1 ms, aunque las nuevas API relacionadas con la gestión de contenido multimedia ofrecen resoluciones mayores, en torno a 1 μs. Tal es el caso de la API para el control temporal de animaciones procedimentales (`requestAnimationFrame`), que utiliza el tipo `DOMHighResTimeStamp` para trabajar con marcas de tiempo de alta resolución; Web Audio API, cuyo contexto de audio incorpora un contador monotónicamente creciente y de alta resolución accesible desde su propiedad `currentTime`; o CSS Animations, cuyos eventos incorporan la propiedad `elapsedTime` que devuelve el tiempo que la animación ha estado funcionando en segundos con una resolución decimal inferior a 1 ms.

En lo que respecta a mecanismos de temporización, los sistemas operativos de Microsoft ofrecen una amplia oferta (temporizadores de la API Win32, multimedia, con tiempo de espera o de colas), pero de entre todos ellos los temporizadores multimedia son los que permiten una mayor precisión en su funcionamiento. En sistemas compatibles con UNIX las opciones también son múltiples (temporizadores de intervalo, temporizadores POSIX o el uso de hilos POSIX), pero la opción más recomendable es emplear aquellos temporizadores que se valgan de los dispositivos hardware de alta resolución disponibles para maximizar su exactitud y precisión. Tal es el caso de los temporizadores de alta resolución de GNU/Linux, por ejemplo. Dentro de las opciones disponibles en el entorno de ejecución estándar de un agente de usuario web, la exactitud y precisión de los temporizadores de JavaScript está a merced de la ocupación de la cola de eventos (Resig, 2008), por lo que no son apropiados. En los casos en los que la sincronización deseada esté limitada al contenido visual, la API para el control temporal de animaciones procedimentales `requestAnimationFrame` es la solución idónea. Lamentablemente, esta API se limita a ese caso concreto y su uso no se recomienda como mecanismo temporizador de propósito general. La API de intercambio de mensajes entre documentos (`postMessage`) puede emplearse para hacer *polling* y simular temporizadores

con un retardo nulo, pero éstos tampoco resultan aconsejables por el excesivo consumo de capacidad de cómputo y energía que suponen. De entre las nuevas API de generación y reproducción de audio, la Web Audio API dispone de un sistema de sincronización propio (a través del método `noteOn`) que ha resultado ser de alta exactitud y precisión, por lo que su uso resulta muy recomendable.

Finalmente, es conveniente recordar que estas aplicaciones experimentales con altos requisitos temporales se ejecutan en su mayoría sobre sistemas operativos que no son *de tiempo real*, por lo que pueden darse errores ocasionales que eviten el cumplimiento de dichos requisitos. Errores que podrán subsanarse con el suficiente tamaño muestral en los experimentos, pero que convendrá detectar mediante el registro de todas las marcas temporales involucradas para descartarlos en un análisis posterior.

5.1 El paradigma experimental del reloj de Libet

Con el propósito de corroborar mediante la experimentación con personas todas las conclusiones previas sobre el desarrollo de aplicaciones con altos requisitos temporales en la Web, decidimos implementar un paradigma experimental que precisara tanto la presentación exacta y precisa de contenido visual y auditivo, como el registro de la interacción del usuario. Tal es el caso del paradigma experimental del reloj de Libet (Libet y cols., 1983), basado en un procedimiento previo de Wundt (1902).

A mediados de los años 60 del siglo XX, Libet investigó las relaciones temporales entre estímulos externos y la percepción de los mismos (Libet, 1965, 1966, 1978). Uno de sus hallazgos más relevantes tiene que ver con la demora en la experiencia consciente. A través de sus experimentos, Libet, Wright, Feinstein, y Pearl (1979) hallaron una diferencia en torno a los 500 ms entre el momento en el que una persona es estimulada eléctricamente en su mano y el momento en el que esta persona informa de forma consciente de este estímulo. Esta diferencia, aunque notable, podría achacarse a los retardos ocasionados en la transmisión del estímulo hasta el cerebro. Sin embargo, mediante electroencefalografía (EGG) detectaron que la actividad cerebral comenzó casi inmediatamente al inicio del estímulo. Tomando esta diferencia como referencia, Libet y cols. (1983) investigaron la relación

entre el momento en el que una persona actúa (pulsa un botón) y el momento en el que informa acerca de su voluntad de actuar. Estos autores encontraron en su estudio sobre la voluntad consciente que la actividad cerebral se inicia al menos 400 ms antes del momento de la acción, mientras que la voluntad consciente se produce solamente 200–150 ms antes de la acción. A pesar de que algunas interpretaciones un tanto laxas de estos resultados han sugerido la ausencia del libre albedrío humano, el propio Libet ha explicado más recientemente que la voluntad consciente dispone de un intervalo en el que puede ejercer un «veto consciente» sobre la acción hasta 50 ms antes de que se produzca. Es decir, percibimos un estímulo y decidimos actuar (a este momento Libet y cols. lo denominan T_{RP}, de *readiness potential*), aproximadamente 200–150 ms antes de la acción somos conscientes de nuestra voluntad de actuar (a este momento Libet y cols. lo denominan T_W, de *conscious will*) y tenemos un margen para ejercer un veto sobre la acción hasta 50 ms antes de que se produzca, momento a partir del cual se transmite la orden a los músculos asociados a la acción (Libet, 2004).

Además de sus implicaciones acerca de la voluntad consciente, los experimentos de Libet y cols. también plantearon otra cuestión interesante acerca de la percepción temporal de los eventos relacionados con una acción (T_{RP}, T_W, $T_{acción}$) y su interacción con la causalidad. En este sentido, los resultados de Haggard, Clark, Kalogeras, y cols. (2002) sugieren que el hecho de que consideremos que nuestras acciones causen un evento provoca que lo percibamos como anterior en el tiempo. Según estos autores se da, por tanto, un efecto de «unión temporal» (*temporal binding*) o «unión intencional» (*intentional binding*) entre la acción intencionada y sus efectos percibidos. Otros autores como Banks y Isham (2009) van más allá al afirmar que el momento en el que decidimos actuar se infiere a partir del momento de la acción (ver figura 5.1). A pesar de la gran importancia dada a la voluntad de acción por estos y posteriores estudios (Engbert, Wohlschläger, y Haggard, 2008), recientemente Buehner (2012) ha sugerido que estos estudios confunden intención con causalidad, ya que sus resultados avalan que el efecto de «unión temporal» tiene su raíz en la predicción de causalidad más que en la intención de actuar.

Como se puede observar, la investigación acerca de la «unión temporal» es un tema candente en la actualidad, por lo que contar con una implementación pública de referencia del procedimiento experimental del reloj de Libet aportaría claros

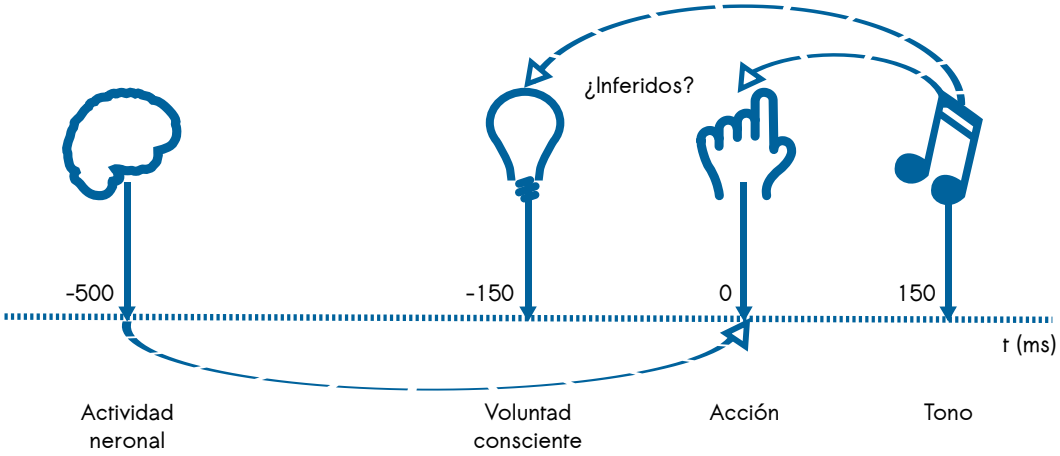

Figura 5.1: Relación entre los diferentes momentos relevantes dentro del paradigma experimental del reloj de Libet.

beneficios metodológicos, tales como la facilidad para llevar a cabo nuevos experimentos o replicar hallazgos previos.

En su versión del procedimiento, Libet y cols. emplearon el registro mediante electromiografía (EMG) para registrar el momento de la acción del participante y un osciloscopio para mostrar los estímulos visuales requeridos. Más adelante, Haggard y cols. llevaron a cabo este mismo procedimiento con la ayuda de un ordenador. Como cabe esperar, las implementaciones del procedimiento del reloj de Libet realizadas en este capítulo seguirán un enfoque similar, apoyándose en los dispositivos de entrada y salida habituales en un PC.

El funcionamiento de cada ensayo en un experimento que sigue el paradigma experimental del reloj de Libet es el siguiente: 1) el participante recibe por pantalla un mensaje indicando que en breves momentos comenzará el ensayo; 2) tras una pausa de duración variable (típicamente entre 1000 y 3000 ms), se mostrará una esfera de reloj en la pantalla sobre la que comienza a girar un punto rotatorio (ver figura 5.2); 3) durante la primera vuelta del punto rotatorio sobre la esfera de reloj el participante no deberá realizar ninguna acción (la primera vuelta se configura de esta forma para que el participante se habitúe a la velocidad del punto rotatorio); 4) durante la segunda vuelta del punto rotatorio, el participante deberá pulsar una tecla; 5) a raíz de esta acción, pueden darse tres consecuencias, a) suena un tono casi inmediatamente después de la pulsación de la tecla, b) suena un tono demo-

rado después de la pulsación de la tecla, o c) no suena ningún tono después de la pulsación de la tecla; 6) finalmente, al terminar su segunda vuelta el punto rotatorio se detiene y se muestra al participante la esfera de reloj vacía con el objeto de que indique sobre ella el momento en el que decidió pulsar la tecla (T_W). En lugar de esto, en algunos experimentos se solicitará al participante que informe sobre el momento en el que pulsó la tecla ($T_{acción}$). Una vez terminado el ensayo, se procede a presentar el siguiente ensayo, con un funcionamiento similar.

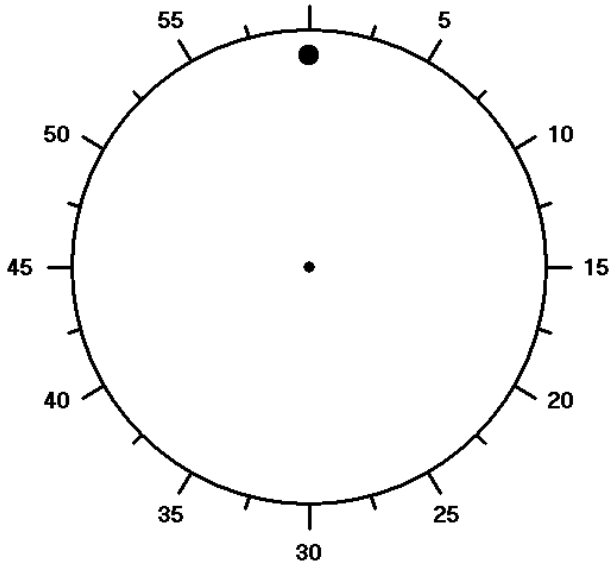

Figura 5.2: Esfera del reloj de Libet y su punto rotatorio.

5.2 Metodología

Partiendo de los objetivos mencionados en la sección anterior, se ha acometido el desarrollo de dos aplicaciones que implementan el paradigma del reloj de Libet: 1) Labclock, una aplicación offline Win32 nativa, y 2) Labclock Web, una aplicación online basada en los nuevos estándares web. Para el desarrollo de ambas aplicaciones se han tenido en cuenta los resultados obtenidos en capítulos anteriores. De esta forma, Labclock se apoya en los temporizadores multimedia y `QueryPerformanceCounter` para la temporización y registro temporal

respectivamente, mientras que Labclock Web emplea animaciones declarativas en CSS para controlar el movimiento del punto rotatorio del reloj, la Web Audio API para la generación del tono demorado y las marcas de tiempo de los eventos DOM para el registro de la interacción del usuario. Los detalles de su implementación y los motivos por los que son necesarias ambas versiones se explican más adelante.

Una vez desarrolladas estas aplicaciones ha sido preciso evaluar su grado de cumplimento de sus elevados requisitos temporales. Dada la naturaleza de la animación propia del paradigma experimental del reloj de Libet, no ha sido posible emplear el mismo procedimiento que en el análisis de la exactitud y precisión de la presentación del contenido audiovisual del capítulo 3, puesto que el punto rotatorio del reloj es circular y a su vez describe una trayectoria circular. Esto ocasiona patrones de detección difíciles de interpretar en el fotosensor del BBTK, con fluctuaciones rápidas de blanco a negro en los momentos en los que el punto rotatorio se aproxima al fotosensor. Por esta razón, se ha empleado otro equipamiento para la medición: una cámara de alta velocidad (modelo CASIO ZR-100) capaz de grabar una secuencia de vídeo a 1000 FPS. Una comunicación personal con Robert Plant, desarrollador del BBTK, confirmó que el enfoque seguido para evaluar estas aplicaciones ha sido adecuado, teniendo en cuenta las dificultades mencionadas anteriormente.

Dado que Labclock es una aplicación nativa de Windows, ha sido probada sobre Windows 7 Professional SP 1, mientras que Labclock Web ha sido probada en Google Chrome 17 sobre Windows 7 Professional SP 1, Ubuntu Linux 10.04 LTS y Mac OS X 10.7.3. Posteriormente se ha analizado el contenido de los vídeos grabados, extrayendo y etiquetando cada uno de los marcos que lo componen para comprobar que el punto rotatorio del reloj de Libet cumple con el periodo establecido en el fichero de configuración (típicamente de 2,56 s). Una vez etiquetados los fotogramas, se ha procedido a recrear el movimiento del punto rotatorio a través de una animación GIF89a por cada caso analizado. La elección de esta tecnología viene motivada por el hecho de que GIF89a no realiza optimizaciones entre marcos similares a las que emplean los *codecs* de vídeo, que ocasionalmente pueden provocar la pérdida o superposición de algún marco. La figura 5.3 muestra los fotogramas claves de estas animaciones para las pruebas de Labclock sobre Windows 7 y Labclock Web en Google Chrome 17 sobre Windows 7, GNU/Linux y Mac

OS X, junto con un código QR que enlaza a la animación GIF89a de la que se han extraído. Como puede observarse, el periodo de 2560 ms configurado en cada combinación de aplicación y plataforma se cumple adecuadamente en cada caso.

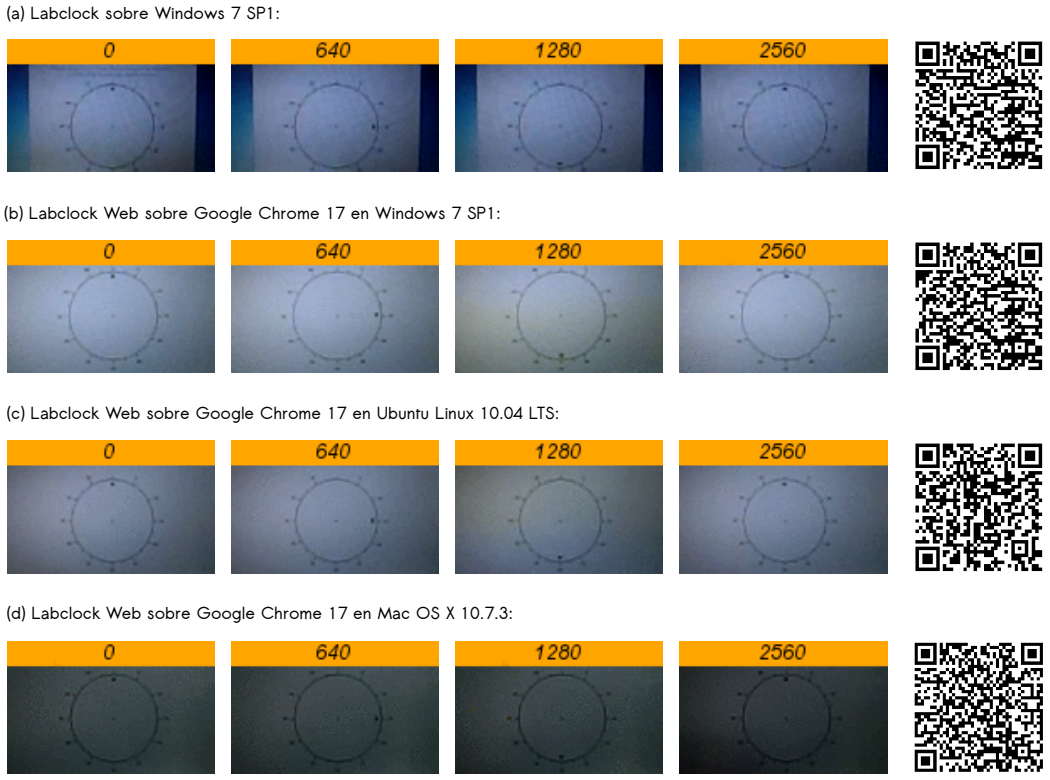

Figura 5.3: Marcos clave de las animaciones de alta velocidad para Labclock y Labclock Web: (a) Labclock sobre Windows 7, (b) Labclock Web en Google Chrome 17 sobre Windows 7, (c) Labclock Web en Google Chrome 17 sobre GNU/Linux, (d) Labclock Web en Google Chrome 17 sobre Mac OS X.

5.3 Labclock

Labclock es una aplicación Win32 desarrollada en Visual Basic 6.0 Professional. Las razones que motivan este entorno de desarrollo son las siguientes: 1) los experimentos relacionados con el paradigma experimental del reloj de Libet se realizaron en colaboración con Labpsico, el laboratorio de psicología experimental de la Universidad de Deusto; este grupo de investigadores está compuesto principal-

mente por psicólogos con conocimientos de programación en Visual Basic, por lo que desarrollando Labclock en este lenguaje, serán capaces de adaptarlo o completarlo en el futuro; 2) Visual Basic 6.0 emplea las llamadas a las API nativas de Windows, sin recurrir a código gestionado como sucede en la plataforma .Net de Microsoft; y 3) este entorno de desarrollo proporciona un adecuado equilibrio entre las facilidades para la programación de aplicaciones que brinda y la cantidad de recursos computacionales que precisa.

Dado que Labclock está diseñado para ser utilizado por investigadores sin conocimientos de programación, su configuración está alojada en ficheros auxiliares. En el apéndice C se detalla el formato de estos ficheros y se explica cómo realizar modificaciones en ellos para crear nuevos diseños experimentales.

El flujo de ejecución de Labclock es el siguiente: 1) inicialmente se carga el fichero de configuración general de Labclock (`config.ini`) y se accede a su contenido para conocer el nombre de los ficheros de definición del experimento; 2) tras la carga de los ficheros de definición del experimento, se muestra un diálogo para seleccionar el grupo o balanceo que se empleará; 3) una vez seleccionado el grupo o balanceo, se carga su configuración y se inicializan todas las estructuras de datos necesarias para llevar a cabo el experimento (`ArrayBegin`, que almacena las pantallas de información inicial; `ArrayPhases`, que almacena el contenido de las fases experimentales; y `ArrayEnd`, que almacena las pantallas de información final); 4) se entra en el estado `STATE_BEGIN` en el que se permite al usuario navegar entre las pantallas iniciales de instrucciones; 5) al llegar a la pantalla de demostración, se entra en el estado `STATE_DEMO`; 6) posteriormente se entra en el estado `STATE_CODE`, donde se muestra un formulario para introducir el código de paso facilitado por el experimentador (configurado en el fichero `config.ini`); 6) tras introducir correctamente el código de paso, se comienza con los ensayos de la primera fase; 7) en cada ensayo se pasa por los diferentes estados definidos (ver figura 5.4) para cada ensayo, en los que se indica un tono de alerta, se muestra la primera vuelta del reloj, la segunda vuelta y se permite que el usuario seleccione un lugar dentro de la esfera del reloj (ver figura 5.5) respectivamente; 8) al finalizar todos los ensayos de una fase, se entra en el estado `STATE_PHASE_END` en el que se muestra la pantalla de instrucciones que prosigue a la fase; 9) si aún quedan fases por realizar, se volverá al estado `STATE_TRIAL_READY` y se repetirán los

pasos 7 y 8; y 10) si no quedan fases por realizar, se pasa al estado `STATE_END` en el que se procede al almacenamiento de los resultados en el fichero definido por la propiedad `Results` del fichero de configuración `config.ini`. El formato de los resultados está descrito en la sección C.1.3.

A bajo nivel, Labclock emplea varias llamadas a diferentes API de Win32 para garantizar el cumplimiento de sus estrictos requisitos temporales. Tal es el caso de `QueryPerformanceFrequency` y `QueryPerformanceCounter` de Kernel32 para la obtención de marcas de tiempo de alta resolución adecuadamente, `SetThreadAffinityMask` y `GetCurrentThread` de Kernel32 para evitar que Labclock cambie de procesador en su ejecución con la intención de minimizar el riesgo de sufrir desplazamientos temporales, o `timeBeginPeriod`, `timeSetEvent`, `timeKillEvent` y `timeEndPeriod` de WinMM para el uso de los temporizadores multimedia. Estas últimas llamadas a la API WinMM se realizan a través de RS Timer 2.01.0.0002, un control ActiveX para la gestión de temporizadores de alta precisión (Sala, 2008).

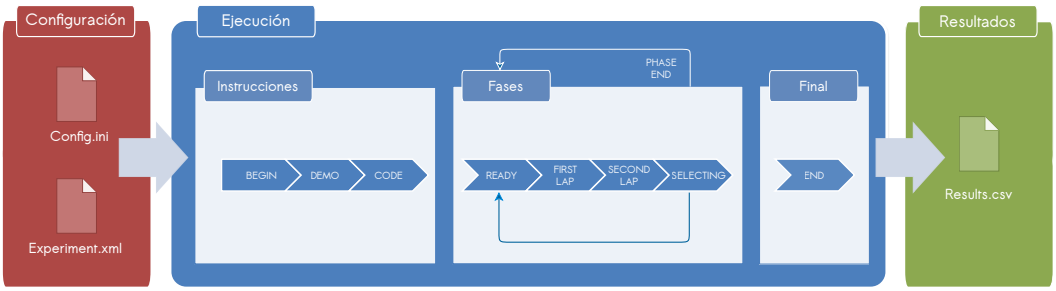

Figura 5.4: Flujo de ejecución de Labclock.

5.4 Labclock Web

Labclock Web implementa el paradigma experimental del reloj de Libet mediante una aplicación web que emplea los últimos estándares web en torno a HTML5 para garantizar el cumplimiento de los estrictos requisitos temporales exigidos: 1) CSS Animations para la presentación de contenidos visuales, 2) Web Audio API para la presentación de contenidos auditivos, y 3) marcas de tiempo de eventos DOM para el registro de la interacción del usuario.

La funcionalidad ofrecida por Labclock Web es muy similar a la que se puede conseguir con Labclock, si bien incorpora algunas mejoras reseñables como el almacenamiento de todas las pulsaciones por cada vuelta del reloj (de cara a detectar varias pulsaciones en una misma vuelta).

El diagrama de estados empleado en Labclock Web es equivalente al empleado en Labclock. Durante las pantallas iniciales de instrucciones el programa permanece en el estado `STATE_PRE` hasta que se pasa al estado `STATE_PASSWORD` al llegar al formulario en el que se solicita el código de paso. Posteriormente se inicia cada fase con el estado `STATE_PHASE_START` y cada ensayo supone la secuencia de estados `STATE_TRIAL_READY`, para hacer sonar el tono de atención y mostrar el aviso de que el ensayo va a empezar, `STATE_TRIAL_RUNNING` durante el funcionamiento del reloj y `STATE_TRIAL_SELECTING`, durante la selección por parte del usuario de una posición en el reloj (ver figura 5.5). Tras cada fase se pasa al estado `STATE_PHASE_END` y cuando ya no quedan más fases por realizar, se finaliza con el estado `STATE_POST` que muestra las pantallas finales y realiza el envío de los datos.

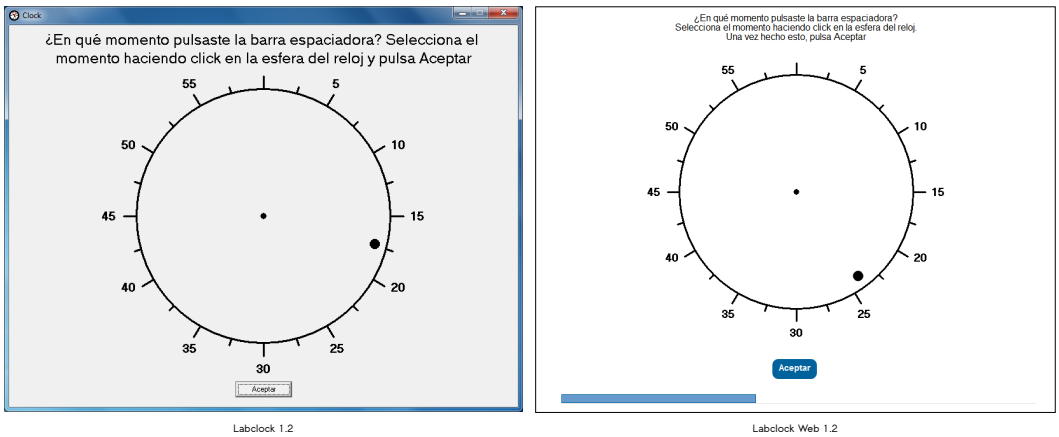

Figura 5.5: Solicitud al usuario de un juicio de acción tras la presentación de un ensayo en Labclock y Labclock Web.

Tanto el código de Labclock como sus ficheros de configuración están fuertemente basados en JSON (*JavaScript Object Notation*). Moderniz es la única biblioteca externa que se incluye en el código para facilitar la comprobación inicial de la idoneidad del entorno de ejecución. El resto de código JavaScript corresponde al

objeto `experiment`, donde reside la configuración del experimento, y el objeto `labclock`, donde reside el comportamiento de la aplicación. La configuración y el comportamiento están separados de tal forma que el objeto `experiment` no tiene métodos sino únicamente propiedades, mientras que en el objeto `labclock` los métodos prevalecen sobre las propiedades, que corresponden a valores temporales internos propios de la ejecución de la aplicación. De esta forma, los investigadores no tienen por qué conocer los detalles de implementación de la aplicación para definir sus experimentos, sino que simplemente deben crear o editar los ficheros de configuración y la aplicación web los interpretará para obtener los resultados deseados. En la sección C.2.1 se muestra un ejemplo de fichero de configuración resumido y se explica su estructura.

Como ya hemos comentado con anterioridad, al inicio de la ejecución de Labclock Web se realizan una serie de comprobaciones a través de la biblioteca Modernizr en las que se confirma que existe soporte para CSS Animations y para la Web Audio API en el entorno de ejecución del agente de usuario elegido. Además, dado que es importante que el reloj de Libet se muestre completo en la pantalla durante los ensayos, también se comprueba que la resolución es la adecuada para que esto sea así. Si no se cumpliera alguno de los prerrequisitos para ejecutar Labclock Web, se mostraría un error por pantalla indicando la causa del problema.

El movimiento del punto rotatorio del reloj de Libet se realiza a través de la animación CSS mostrada en el listado 5.1. Como se puede observar, la animación `spin` tiene dos marcos clave con sendas transformaciones de rotación asociadas. De esta forma, el agente de usuario rotará el elemento con identificativo `dot` desde los 0 a los 360 grados, dando un giro completo. El estado inicial de la animación es pausado (`animation-play-state`), puesto que será iniciada por el programa al comienzo de cada ensayo, y el número de iteraciones que se darán en la animación es de 2 (`animation-iteration-count`), ya que son necesarias dos vueltas de reloj en cada ensayo. El giro tendrá una velocidad angular constante porque la función de temporización (`animation-timing-function`) elegida es `linear`. La duración del ciclo (`animation-duration`) y la demora inicial (`animation-delay`) serán modificadas según la configuración de cada ensayo. Cabe mencionar que la definición mostrada en el listado 5.1 no incorpora las diferentes adecuaciones mediante prefijos que son necesarias temporalmente hasta que

la especificación CSS Animations sea considerada un estándar y esté implementada en todos los agentes de usuario web (`webkit` para Google Chrome y Apple Safari, `moz` para Mozilla Firefox, `ms` para Internet Explorer, `o` para Opera).

```
#marks #dot {
  animation-play-state: paused;
  animation-name: spin;
  animation-iteration-count: 2;
  animation-timing-function: linear;
  animation-duration: 2.560s;
  animation-delay: 0.5s;
}

@keyframes spin {
  from { transform: rotate(0deg); }
  to { transform: rotate(360deg); }
}
```

Listado 5.1: Animación en CSS del punto rotatorio en Labclock Web.

Una vez terminado el experimento, Labclock Web envía los resultados a través de una conexión HTTP Post realizada mediante AJAX. Adicionalmente y por seguridad, se almacenará una copia de respaldo en el propio agente de usuario desde donde se ha lanzado la aplicación, empleando la API de Almacenamiento Local (Local Storage API). En la sección C.2.2 se detalla el formato en el que se almacenan los resultados de Labclock Web.

5.5 Experimentación cognitiva empleando Labclock y Labclock Web

Como complemento a las pruebas automatizadas realizadas con equipamiento de precisión, la experimentación mediante la colaboración de personas voluntarias aporta la validez externa necesaria para que las aplicaciones desarrolladas no se limiten a ser un mero ejercicio teórico-práctico sino que sean capaces de demostrar su utilidad aplicada a la psicología cognitiva. Uno de los principales objetivos de la ingeniería y de los sistemas de información es precisamente ese, aportar la tecnología adecuada a otras ramas del saber. De esta manera, hemos llevado a cabo

6 experimentos mediante el uso de Labclock y Labclock Web en diferentes circunstancias con un doble objetivo: 1) aportar nueva evidencia empírica en torno al paradigma experimental del reloj de Libet que permita entender mejor el efecto de «unión temporal» en humanos, y 2) validar externamente las aplicaciones desarrolladas y, por ende, las tecnologías subyacentes.

El conjunto de experimentos llevados a cabo se muestra en la tabla 5.1 y se detalla más adelante.

Experimento	Software	Participantes	N	Lugar	Demora corta (ms)	Demora larga (ms)	Juicio
A	Labclock	1º psicología	49	Aula	1	500	Decisión
B	Labclock	1º psicología	56	Aula	1	500	Acción
C	Labclock Web	Voluntarios	52	Internet	1	500	Acción
D	Labclock Web	Voluntarios	57	Laboratorio	1	500	Acción
E	Labclock	1º psicología	41	Laboratorio	10	1000	Decisión
F	Labclock Web	1º psicología	52	Laboratorio	10	1000	Acción

Tabla 5.1: Listado de experimentos realizados empleando Labclock y Labclock Web.

5.5.1 Experimento A: «unión temporal» con demoras de 1 y 500 ms y juicio de decisión con tecnologías offline

Siguiendo el paradigma experimental del reloj de Libet explicado anteriormente (ver sección 5.1), en este experimento se emplearon demoras de 1 ms y 500 ms para los ensayos con demora corta y demora larga respectivamente. Además, a lo largo de todos los ensayos, se solicitó a las personas participantes que informaran acerca del momento en el que tomaron la decisión de responder (T_W). El diseño de este experimento corresponde a un diseño intrasujeto con dos tipos de ensayos: aquellos en los que el tono que sigue a la acción del sujeto suena con una demora corta (1 ms) y aquellos en los que la demora es de 500 ms. Así pues, a cada participante se le presentaron 40 ensayos de cada condición, de forma desordenada a partir de una secuencia concreta. Para controlar el posible efecto que pudiera tener

la secuencia precisa de ensayos se diseñaron dos secuencias pseudoaleatorias y se asignó a los participantes a una de ellas al azar. Después de cada ensayo se solicitó al participante que indicara el momento en el que tomó la decisión de pulsar la tecla. Finalmente, tras los 80 ensayos –40 de cada condición–, se presentaron 20 ensayos sin tono para obtener la tasa base de respuestas del participante en esta tarea.

Teniendo en cuenta los resultados de estudios previos y el diseño descrito, la hipótesis de partida fue que el T_W para los ensayos sin demora (1 ms) sería significativamente menor que para los ensayos demorados (500 ms) una vez ponderada la tasa base de cada participante en el experimento.

En este experimento tomaron parte 56 estudiantes de 1º de psicología de la Universidad de Deusto. Realizaron el experimento utilizando el software Labclock en una sala de ordenadores. Se eliminaron 7 participantes por tener más del 25 % de ensayos de alguna de las dos condiciones (demora corta o demora larga) sin responder. Para el resto de participantes ($N = 49$) se calculó la diferencia entre el momento en el que los participantes juzgaron haber decidido responder en cada una de las condiciones de demora y el momento en el que juzgaron haber decidido responder en la tasa base (ver figura 5.6). Estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(48) = 5{,}02$, $p < 0{,}001$, $d = 1{,}02$). Previo a este análisis, se realizó un ANOVA 2 (Secuencia: A vs. B) x 2 (Tipo de ensayo: demora corta vs. demora larga) y no se encontró un efecto principal de Secuencia ni interacciones de este factor con otros, por lo que se decidió analizar los datos de ambos grupos conjuntamente. Esta misma comprobación se realizó para el resto de experimentos de esta serie, con idénticos resultados. Por este motivo, el resto de resultados de los experimentos que se describirán a continuación corresponden a los análisis de los datos de ambos grupos conjuntamente.

Por lo tanto, estos resultados coinciden con la hipótesis de la existencia del efecto de «unión temporal» con demoras de 1 y 500 ms y juicios de decisión. En otras palabras, cuando el efecto de una respuesta se demora en el tiempo, también se desplaza la estimación que el participante hace de cuándo cree que realizó su respuesta. Este aporte tiene valor desde el punto de vista metodológico, puesto que demuestra que el efecto puede darse para demoras más largas que las habituales en

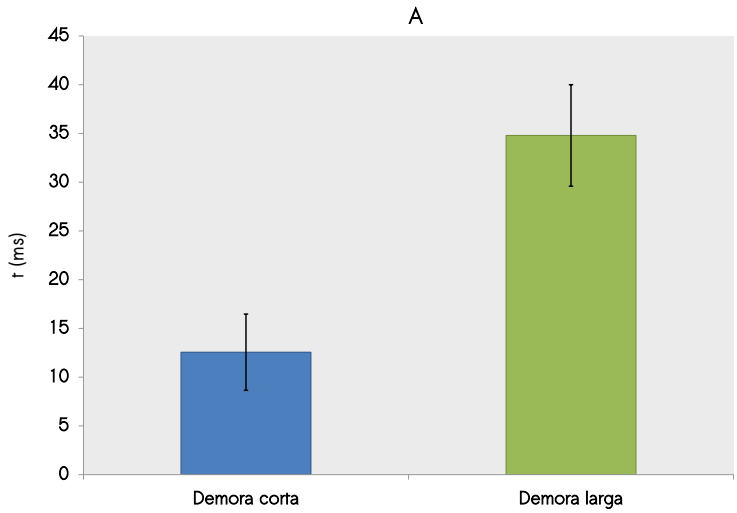

Figura 5.6: Experimento A: comparación entre las estimaciones en los juicios para los ensayos con demora corta (1 ms) frente a los ensayos con demora larga (500 ms).

la literatura empleando un PC como equipamiento (Banks y Isham, 2009; Moore, Lagnado, Deal, y Haggard, 2009). Asimismo, supone la validación externa de Labclock como aplicación para la presentación de contenido audiovisual y registro de la respuesta del usuario de forma exacta y precisa.

5.5.2 Experimento B: «unión temporal» con demoras de 1 y 500 ms y juicio de acción con tecnologías offline

El experimento B es muy similar al experimento A. El único cambio con respecto al experimento anterior tiene que ver con el tipo de juicio: en lugar de solicitar a los participantes que informen del momento en el que decidieron pulsar la tecla durante el funcionamiento del reloj (T_W), se les solicitó que informaran del momento en el que pulsaron la tecla. Se trata, por tanto, de un juicio acerca de su acción y no de su decisión. Este cambio se puso a prueba con el objetivo de descartar que los resultados obtenidos en el experimento A pudieran deberse a que los participantes estuvieran entendiendo mal su cometido y algunos de ellos estuvieran informando sobre el momento en el que decidieron pulsar la tecla mientras que otros lo hicieran sobre el momento en el que pulsaron la tecla. De esta forma, todos los participan-

tes del experimento B responderán acerca del momento en el que pulsaron la tecla puesto que estas instrucciones no inducen a la posible confusión entre ambos momentos. A excepción de este cambio, tanto el diseño del experimento B como su hipótesis de partida son los mismos que en el experimento A.

En este experimento tomaron parte 60 estudiantes de 1º de psicología de la Universidad de Deusto. Realizaron el experimento utilizando el software Labclock en una sala de ordenadores. Se eliminaron 4 participantes por tener más del 25 % de ensayos de alguna de las dos condiciones (demora corta o demora larga) sin responder. Para el resto de participantes ($N = 56$) se calculó la diferencia entre el momento en el que juzgaron haber respondido en cada una de las condiciones de demora y el momento en el que juzgaron haber respondido en la tasa base (ver figura 5.7). Posteriormente, estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(55) = 4{,}325$, $p < 0{,}001$, $d = 0{,}82$).

Estos resultados extienden los hallazgos encontrados en el experimento A de los juicios de decisión sobre la acción y estimación del T_W a los juicios de acción y estimación del momento en el que se actuó, por lo que pierde fuerza la hipótesis alternativa de que los resultados del experimento A pudieran deberse a la falta de comprensión de algunos de los participantes, mientras que se refuerza la evidencia de la existencia de una «unión temporal» en los experimentos realizados bajo el paradigma del reloj de Libet con demoras de 500 ms. En el plano tecnológico, que es el más relevante para los propósitos de la presente tesis, estos resultados confirman nuevamente la validez externa de Labclock como herramienta de investigación sobre el efecto de «unión temporal».

5.5.3 Experimento C: «unión temporal» con demoras de 1 y 500 ms y juicio de acción con tecnologías web en Internet

El objetivo del experimento C es el de dilucidar la posibilidad de llevar a cabo experimentos basados en el paradigma del reloj de Libet en Internet. Para ello replicamos el experimento B («unión temporal» con demoras de 1 y 500 ms y juicio de acción) empleando Labclock Web en lugar de Labclock y una muestra de parti-

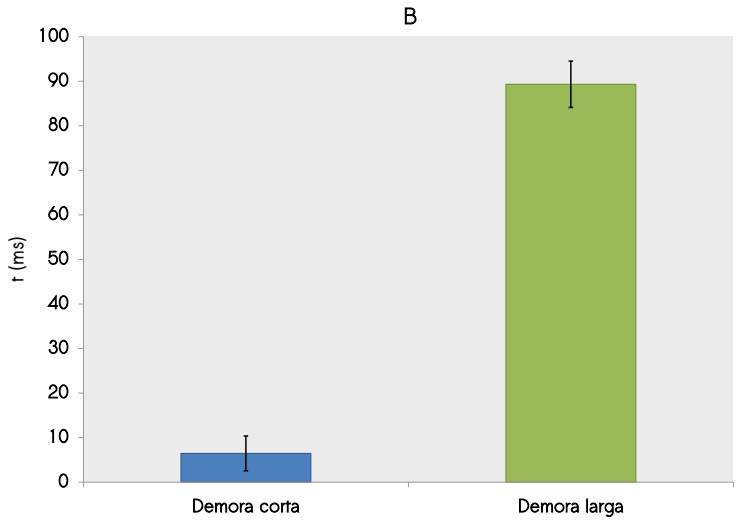

Figura 5.7: Experimento B: comparación entre las estimaciones en los juicios para los ensayos con demora corta (1 ms) frente a los ensayos con demora larga (500 ms).

cipantes proveniente de Internet en lugar de estudiantes voluntarios. El diseño, por tanto, es el mismo que el de los experimentos anteriores.

En este experimento tomaron parte 55 voluntarios de Internet que fueron reclutados de diversas maneras: mediante el uso de las redes sociales (Facebook y Twitter), a través de la página web de Labpsico[1], y gracias a la colaboración de las profesoras Itsaso Barberia (UOC) y Cristina Orgaz (UNED) que colgaron un enlace al experimento en los foros de sus asignaturas. Tres participantes fueron eliminados empleando el mismo criterio que en los experimentos A y B: no responder en más del 25 % de ensayos de alguna de las dos condiciones. Para el resto de participantes ($N = 52$) se calculó la diferencia entre el momento en el que juzgaron haber respondido en cada una de las condiciones de demora y el momento en el que juzgaron haber respondido en la tasa base (ver figura 5.8). De igual manera, estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(51) = 2{,}372$, $p = 0{,}021$, $d = 0{,}47$).

A tenor de estos resultados podemos concluir que es posible hallar el efecto de «unión temporal» en un experimento realizado en Internet. Las diferencias en el

[1] `http://www.labpsico.deusto.es`

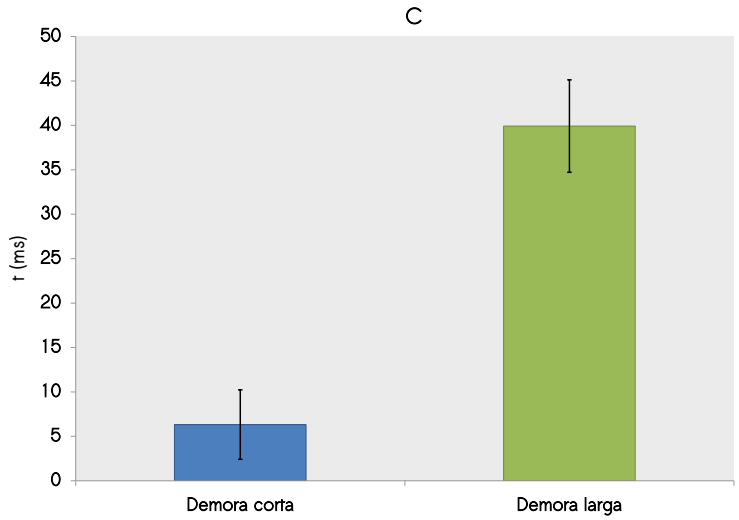

Figura 5.8: Experimento C: comparación entre las estimaciones en los juicios para los ensayos con demora corta (1 ms) frente a los ensayos con demora larga (500 ms).

tamaño del efecto obtenido en el experimento B y el obtenido en el experimento C pueden deberse a múltiples factores, entre los que se encuentran la distinta naturaleza de las muestras de participantes que colaboraron en cada experimento y el uso de una tecnología con limitaciones en su resolución temporal (Labclock Web frente a Labclock). No obstante, este tamaño del efecto se encuentra dentro de los valores habituales en este tipo de experimentos (ver Engbert y cols., 2008), por lo que se ha demostrado que la exactitud y precisión de Labclock Web es suficiente para encontrar diferencias estadísticamente significativas entre ambas condiciones y permitir la investigación sobre este efecto en Internet, algo inédito hasta la fecha.

5.5.4 Experimento D: «unión temporal» con demoras de 1 y 500 ms y juicio de acción con tecnologías web en el laboratorio

El experimento D es una réplica del experimento C –y por lo tanto, similar al experimento B en cuanto a diseño– con la única diferencia de la procedencia de los participantes y del lugar de realización del mismo. A diferencia del experimento anterior, este experimento se llevó a cabo en las cabinas de experimentación de Labpsico, con la participación de personas voluntarias que conocieron la existen-

cia del estudio a través de anuncios repartidos por la zona de Deusto (Bilbao) y decidieron participar debido al incentivo económico que se ofreció (5 euros por participante). Un total de 64 participantes colaboraron con el estudio, de los que 7 fueron descartados empleando el mismo criterio que en los experimentos anteriores de esta serie (dejar sin responder más del 25 % de ensayos de alguna de las dos condiciones). Para el resto de participantes ($N = 57$) se calculó la diferencia entre el momento en el que juzgaron haber respondido en cada una de las condiciones de demora y el momento en el que juzgaron haber respondido en la tasa base (ver figura 5.9). Una vez más, estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(56) = 2{,}459$, $p = 0{,}017$, $d = 0{,}46$).

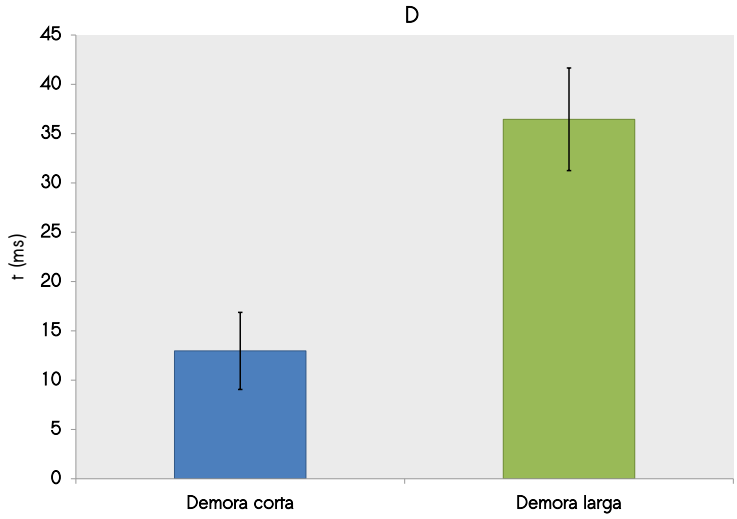

Figura 5.9: Experimento D: comparación entre las estimaciones en los juicios para los ensayos con demora corta (1 ms) frente a los ensayos con demora larga (500 ms).

Estos resultados suponen una confirmación de los datos obtenidos a través de Internet en el experimento C y un nuevo aporte a la evidencia experimental que apoya la validez externa de Labclock Web como aplicación para la realización de experimentos siguiendo el paradigma del reloj de Libet mediante tecnologías web. Como se puede observar, los resultados de este experimento se sitúan más próximos a los obtenidos en el experimento C (Labclock Web en Internet) que en el

experimento B (Labclock en aula de ordenadores), aunque los valores son similares a los obtenidos en el experimento A (Labclock en aula de ordenadores con juicios de decisión). Las razones de estas diferencias pueden deberse a las aplicaciones empleadas o al distinto rigor experimental esperable en un estudio con 60 participantes en un aula de ordenadores frente a la participación individual y bajo la supervisión de un experimentador en las instalaciones de Labpsico.

5.5.5 Experimento E: «unión temporal» con demoras de 10 y 1000 ms y juicio de decisión con tecnologías offline

Mediante los experimentos E y F pretendimos determinar si es posible obtener el efecto de «unión temporal» empleando demoras de 1000 ms, un valor sensiblemente superior a las demoras habituales en este tipo de experimentos (típicamente en torno a los 250 ms). Para evitar los problemas que suelen ocasionar los valores próximos a 0 en la experimentación psicológica, la duración de los ensayos con demora corta se incrementaron hasta los 10 ms. Así pues, en el experimento E se empleó el paradigma del reloj de Libet con demoras de 10 y 1000 ms consultando a los participantes sobre el momento en el que decidieron pulsar la tecla (juicio de decisión). Este diseño es idéntico al descrito en el experimento A, con la salvedad de que los tiempos de demora son más amplios (10 ms frente a 1 ms para la condición de demora corta y 1000 ms frente a 500 ms para la condición de demora larga).

En este experimento tomaron parte 44 estudiantes de 1º de psicología de la Universidad de Deusto. Realizaron el experimento utilizando el software Labclock en una sala de ordenadores. Se eliminaron 3 sujetos empleando el mismo criterio que en los experimentos anteriores (tener más del 25 % de ensayos de alguna de las dos condiciones sin responder). Para el resto de participantes ($N = 41$) se calculó la diferencia entre el momento en el que juzgaron haber decidido responder en cada una de las condiciones de demora y el momento en el que juzgaron haber decidido responder en la tasa base (ver figura 5.10). Posteriormente, estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(40) = 2{,}134$, $p = 0{,}039$, $d = 0{,}48$).

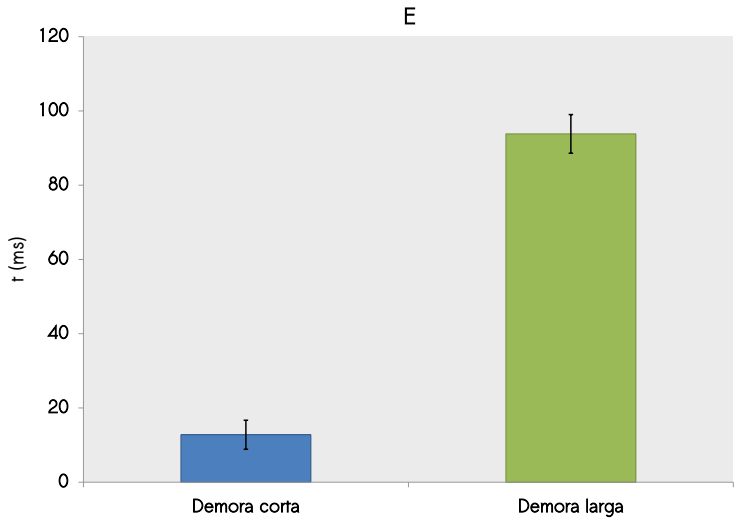

Figura 5.10: Experimento E: comparación entre las estimaciones en los juicios para los ensayos con demora corta (10 ms) frente a los ensayos con demora larga (1000 ms).

Como puede observarse, los resultados obtenidos son coherentes con la hipótesis de que el efecto de «unión temporal» también se da con demoras de 1000 ms en los juicios de decisión. Sin embargo, empleando el paradigma experimental del reloj de Libet en su configuración estándar (2560 ms de periodo de giro) es difícil ir más allá sin alterar la velocidad angular del punto rotatorio, ya que en ensayos con demoras superiores a 1280 ms –lo que equivale a la mitad de una vuelta del reloj– podrían surgir ambigüedades a la hora de atribuir la respuesta del participante a un error de apreciación por defecto o por exceso.

5.5.6 Experimento F: «unión temporal» con demoras de 10 y 1000 ms y juicio de acción con tecnologías web en el laboratorio

El experimento F cierra la serie de experimentos sobre «unión temporal» estudiando la existencia de este efecto con demoras de 10 y 1000 ms cuando los participantes son preguntados acerca del momento en el que actuaron (juicio de acción). Así pues, el experimento F puede entenderse tanto como una réplica del experimento B con tiempos de demora más largos, como una adaptación del ex-

perimento E cambiando los juicios de decisión por juicios de acción. Dado que en el total de los experimentos anteriores nos hemos decantado en más ocasiones por Labclock (experimentos A, B y E) que por Labclock Web (experimentos C y D), decidimos emplear Labclock Web para llevar a cabo este experimento.

56 estudiantes de 1º de psicología de la Universidad de Deusto colaboraron con su participación en este experimento en las instalaciones de Labpsico. Se descartaron 4 de ellos debido a que dejaron más del 25 % de ensayos de alguna de las dos condiciones sin responder (mismo criterio de eliminación de participantes que en el resto de experimentos de la serie). Para el resto de participantes ($N = 52$) se calculó la diferencia entre el momento en el que juzgaron haber respondido en cada una de las condiciones de demora y el momento en el que juzgaron haber respondido en la tasa base (ver figura 5.11). Posteriormente, estos dos valores fueron comparados mediante una prueba t de medidas independientes y se encontraron diferencias significativas entre ellos ($t(51) = 2{,}224$, $p = 0{,}031$, $d = 0{,}44$).

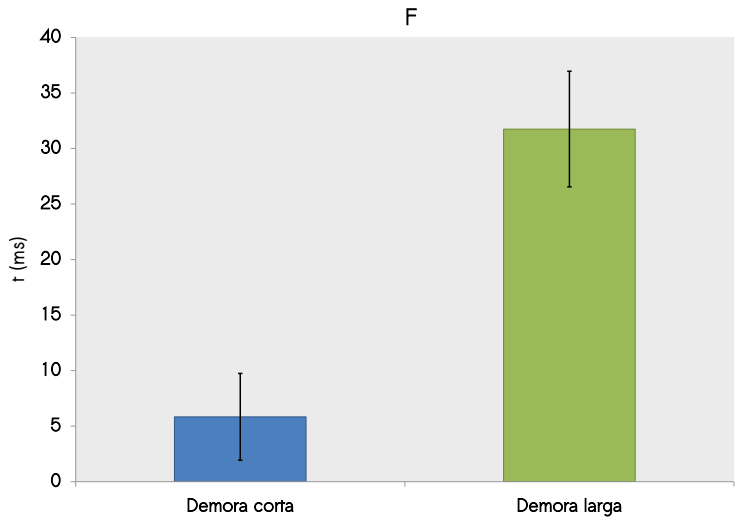

Figura 5.11: Experimento F: comparación entre las estimaciones en los juicios para los ensayos con demora corta (10 ms) frente a los ensayos con demora larga (1000 ms).

Así pues, estos resultados sugieren que el efecto de «unión temporal» se produce también en los juicios de acción para demoras de 1000 ms, extendiendo lo aportado por el experimento anterior. Además, el hecho de haber hallado el efecto

principal de este paradigma experimental supone un nuevo refrendo a la validez externa de Labclock Web como herramienta para llevar a cabo experimentos de este tipo.

En definitiva, el conjunto de experimentos mostrado en este capítulo constituye una considerable evidencia a favor de la exactitud y la precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario tanto en Labclock como en Labclock Web.

5.6 Conclusiones generales

Como conclusión general de todo el capítulo cabe mencionar que se ha logrado obtener el efecto principal de «unión temporal» en todos los estudios siguiendo el paradigma experimental del reloj de Libet llevado a cabo, tanto mediante tecnologías offline (Labclock) como mediante tecnologías web (Labclock Web). Es decir, se han encontrado diferencias significativas entre las condiciones de demora corta y de demora larga en todos ellos, y se da la circunstancia de que estas diferencias están motivadas porque los participantes no logran ser precisos en las dos condiciones: quien es preciso en la condición de demora larga, empeora sus juicios en la condición de demora corta o viceversa. Además, la variedad de diseños, tecnologías y muestras de participantes empleada a lo largo de esta serie de experimentos ha permitido conocer la influencia de estas variables en el tamaño del efecto o la variabilidad en las respuestas, por lo que deberán tenerse en consideración en futuros estudios.

Estos resultados tienen valor tanto desde el punto de vista de la psicología cognitiva como desde el punto de vista tecnológico, puesto que sirven para avanzar en ambos ámbitos al mismo tiempo. El aporte a la psicología cognitiva se resume en la mejora que se deriva de ampliar hasta 500 o 1000 ms la condición de demora larga en el paradigma experimental del reloj de Libet, por las ventajas metodológicas que ello supone para poder adecuar los experimentos de «unión temporal» a situaciones más naturales y fáciles de reproducir. En cuanto al aporte tecnológico, gracias a estos resultados queda demostrada la validez externa de Labclock y Labclock Web, y por ende, la viabilidad del uso de tecnologías web para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción de usuario.

"Just because the standard provides a cliff in front of you, you are not necessarily required to jump off it"

Norman Diamond

CAPÍTULO

6

Recomendaciones de desarrollo web para aplicaciones con estrictos requisitos temporales

En el presente capítulo se recogen varias recomendaciones de desarrollo de aplicaciones web con altos requisitos temporales. Estas recomendaciones están inspiradas tanto en la evidencia empírica estudiada en los capítulos precedentes como en las conclusiones obtenidas en sus estudios por parte de otros investigadores. Además de estas recomendaciones, se proporciona un breve recordatorio de la naturaleza del entorno de ejecución que proveen los agentes de usuario web y se cierra el capítulo con la explicación detallada de un conjunto de herramientas que pueden contribuir a incrementar la exactitud y precisión en las aplicaciones basadas en estándares web.

6.1 El entorno de ejecución de los agentes de usuario web

En 2004 el W3C publicó un diagrama en el que se relacionaban muchas de las tecnologías presentes en la Web en diferentes capas y agrupaciones: definición de documentos web, formatos de ficheros de datos, estándares de movilidad, servicios web, web semántica, seguridad, son algunas de ellas. Resulta sencillo señalar las incoherencias de ese diagrama después de todos los cambios que han sucedido en torno a la Web en los últimos años. Sin embargo, el diagrama todavía tiene su utilidad puesto que aporta una idea aproximada de la amalgama de especificaciones y tecnologías diferentes que cohabitan en el entorno de ejecución de un agente de usuario web (ver figura 6.1).

Figura 6.1: Diagrama de tecnologías de la Web por el W3C.

Más recientemente, Sivonen (2009) ha tratado de actualizar el diagrama del W3C mediante una propuesta que centra su atención en el agente de usuario web (ver figura 6.2). Esta propuesta no está exenta de críticas (Schepers, 2009), pero al igual que su antecesora, aporta cierta estructura y define algunas de las relaciones que se dan entre todas las tecnologías empleadas. En palabras de su propio autor,

las limitaciones que implican los diagramas en dos dimensiones hacen imposible reflejar convenientemente todas las interacciones entre las tecnologías presentes en los agentes de usuario web actuales.

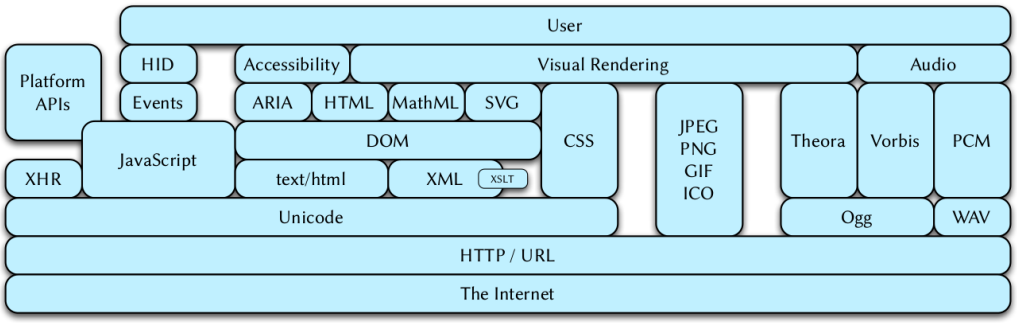

Figura 6.2: Diagrama de tecnologías de los agentes de usuario web por (Sivonen, 2009)

Así pues, un desarrollador web –conozca o no estos diagramas– sabe que no cuenta únicamente con la expresividad del estándar HTML para crear sus aplicaciones, sino que es capaz de apoyarse en otras tecnologías para lograr sus propósitos. Desgraciadamente, los rápidos avances en la Web han ocasionado disparidades notables en el grado de desarrollo de estas tecnologías en función del agente de usuario empleado. Por esta razón, es muy recomendable analizar previamente las funcionalidades disponibles dentro del entorno de ejecución de una aplicación web, y de esta forma conocer qué tecnologías pueden emplearse de forma nativa y cuáles tendrán que ser reemplazadas por alguna de las alternativas disponibles. Para esta labor, el enfoque recomendable es realizar pruebas individuales de cada tecnología (del modo en que bibliotecas como Modernizr las llevan a cabo), en lugar del clásico pero incorrecto enfoque consistente en detectar el modelo y versión del agente de usuario y presuponer sus funcionalidades.

Centrando nuestra atención en los mecanismos de temporización y funciones de tiempo disponibles en este entorno de ejecución, es importante repasar la relación entre la cola de eventos de JavaScript y el hilo de ejecución encargado de la actualización de la interfaz de usuario. El entorno en el que se ejecutan las aplicaciones web desarrolladas en JavaScript en un agente de usuario web difiere mucho

de otros entornos de ejecución de aplicaciones de usuario. En primer lugar, es fundamental tener en cuenta que solamente se dispone de un hilo de ejecución desde el que poder realizar modificaciones en la interfaz de usuario (conocido como UI Thread), por lo que cualquier rutina en JavaScript que se exceda en su duración provocará el bloqueo de la interfaz de usuario y, por tanto, una degradación de la experiencia de usuario. Gracias a los Web Workers (Hickson, 2012c) es posible ejecutar diferentes rutinas de JavaScript en paralelo dentro de una misma aplicación web. Sin embargo, los hilos de ejecución proporcionados por los Web Workers no tienen acceso al UI Thread, por lo que tienen que comunicarse con éste a través del paso de mensajes que informen del estado o resultado de sus operaciones. Asimismo, conviene recordar que el uso de los temporizadores de JavaScript no implica que la función de *callback* será ejecutada cuando se cumpla el periodo definido, sino que simplemente será encolada tantos ms en el futuro como el periodo del temporizador indique. Para una mejor comprensión de este hecho, en la figura 6.3 se muestra una secuencia en la se produce un problema en la temporización de las rutinas encoladas provocado por la limitación de disponer de un único hilo de ejecución capaz de modificar la interfaz de usuario (i. e., la excesiva duración de la función `B` provoca que se pierdan dos temporizaciones de la función `Int` y por lo tanto una ejecución de `Int` se descarta y la otra queda demorada).

```
setTimeout(A, tA)
setTimeout(B, tB)
setInterval(Int, ti)
```

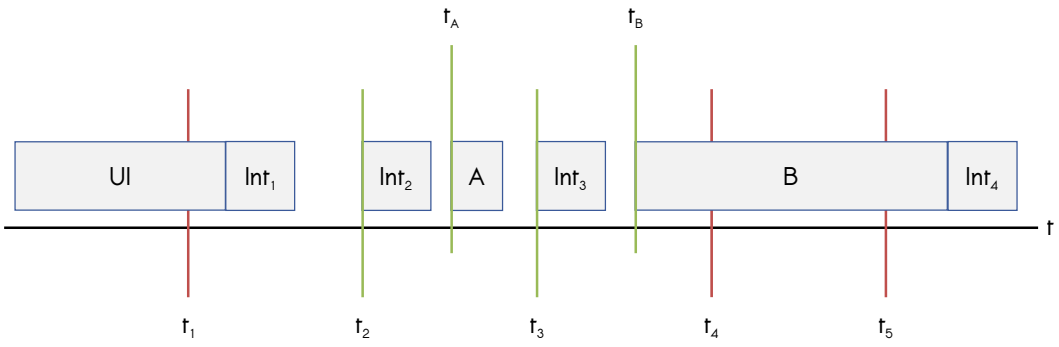

Figura 6.3: Uso de temporizadores en JavaScript y su relación con el hilo de actualización de la interfaz de usuario.

Debido a este tipo de problemas, el estándar de ECMAScript (Ecma, 2011) y la especificación de HTML5 (Hickson, 2012a) definen una limitación para el periodo de los temporizadores de 4 ms tanto para `setTimeout` como `setInterval`, por lo que cualquier periodo inferior a este valor pasado como argumento a estas funciones será redefinido a 4 ms. Este detalle es aprovechado por algunos desarrolladores web para encolar llamadas a funciones tan frecuentemente como sea posible mediante `setInterval(fn, 0)`, lo que origina una serie de llamadas encoladas cada 4 ms. A pesar de que esta propuesta está considerada una mala práctica, la limitación impuesta por el estándar e implementada en los agentes de usuario evita que el UI Thread se colapse. Sin embargo, la existencia de un gran número de aplicaciones web que se sirven de este truco para acaparar el flujo de ejecución ha impedido que algunos desarrolladores de agentes de usuario web pudieran rebajar ese valor sin afectar gravemente al rendimiento, al consumo de recursos y, por ende, al consumo de energía (Belshe, 2010). De hecho, ésta es la principal razón por la que los mecanismos de temporización tienen granularidades en torno a 1 ms aún en sistemas que funcionan sobre procesadores con frecuencias del orden de GHz, puesto que resulta necesario encontrar un equilibrio entre la sobrecarga que supone la reducción del periodo mínimo entre eventos temporales y la reducción de la latencia a la hora de atender esos eventos. Así, los desarrolladores de Google Chrome publicaron su versión 1.0 con soporte para temporizadores de 1 ms de periodo, pero pronto vieron que muchos de sus usuarios se quejaban del excesivo consumo de energía y de aplicaciones web que no funcionaban correctamente (Zakas, 2011). Esto les obligó a volver a la limitación de 4 ms en las siguientes versiones de Google Chrome, e incluso a descartar el uso de los Multimedia Timers en Windows en aquellas ocasiones en las que el agente de usuario detecte que está funcionando sin estar conectado a la red eléctrica, empleando entonces el reloj del sistema, con una granularidad de en torno a los 15 ms (Krishnan, 2008; Russinovich y cols., 2009). Algo similar ocurre en dispositivos móviles, tal y como describe Zakas (2010).

La tabla 6.1 muestra un resumen de los periodos mínimos configurables en temporizadores de JavaScript en función del agente de usuario y de las condiciones de ejecución de la aplicación web (Zakas, 2012). Como se puede observar, las diferencias en el periodo de los temporizadores entre una aplicación en primer plano

o en segundo plano son notables en la mayoría de los casos. Este detalle no debería pasarse por alto a la hora de desarrollar aplicaciones que puedan ser usadas simultáneamente con otras (v. gr., un videojuego de tipo *arcade* no debería preocuparse tanto de este hecho como debería hacerlo un visualizador de fractales que emplee temporizadores para no colapsar la interfaz mientras realiza sus cálculos).

	Red eléctrica	**Batería**	**Segundo plano**
Mozilla Firefox	4	4	1000
Google Chrome	4	15,625	1000
Apple Safari	4	4	4
Internet Explorer	4	15,625	1000
Opera	4	4	4
iOS	-	4	10
Kindle Fire	-	10	10
Android	-	10	10
Chrome for Android	-	4	1000

Tabla 6.1: Periodos mínimos (en ms) de los temporizadores en JavaScript en función del agente de usuario y sus condiciones de ejecución.

En lo referente a las funciones de tiempo, el estándar de ECMAScript indica que todas las fechas se almacenan en milisegundos desde el 1 de enero de 1970 (Ecma, 2011). Además de esta limitación en la resolución de la función de tiempo estándar en JavaScript, puede ocurrir que su valor se actualice en función del reloj del sistema, que en algunos sistemas operativos y agentes de usuario web ocurre cada 10–15 ms. Esto provoca que sus valores se agrupen en conjuntos separados por intervalos de esa duración (v. gr., después de tomar 10.000 marcas de tiempo seguidas, las 3000 primeras tienen un valor t, las 3500 segundas un valor $t + 15$ ms y las 3500 últimas un valor $t + 30$ ms).

Este es, por tanto, el peculiar entorno de ejecución disponible para las aplicaciones basadas en estándares web. No obstante, como se detalla en las siguientes secciones, el desarrollo de nuevas API orientadas a la mejora del rendimiento de las

aplicaciones web ha permitido proporcionar otros mecanismos de temporización y medición temporal a los desarrolladores, que podrán beneficiarse de ellos en los entornos que los incorporen.

6.2 Recomendaciones de desarrollo web

6.2.1 Velocidad frente a exactitud y precisión

La abundante literatura relacionada con la optimización del rendimiento en aplicaciones web en JavaScript (Crockford, 2008; Souders, 2008, 2009; Zakas, 2010) centra su atención en reducir al máximo los tiempos que la aplicación requiere para su carga, puesta en funcionamiento, actualización y comunicación a través de la red. Este enfoque tiene su razón de ser y se fundamenta en dos argumentos: 1) las aplicaciones web compiten contra aplicaciones de escritorio u otras aplicaciones web por la atención del usuario, por lo que minimizar el tiempo de carga y el tiempo de respuesta a la interacción del usuario es una buena estrategia para lograr mantener esa atención por parte del usuario, y 2) el tiempo de carga y rendimiento de las aplicaciones web es analizado por los principales motores de búsqueda para priorizar aquellas con menores tiempos de respuesta.

Podría decirse entonces que en la Web existe una elevada presión por responder en el menor tiempo posible, por las aplicaciones *en tiempo real*. Sin embargo, pocas aplicaciones web tienen en la actualidad requisitos temporales propios de sistemas *de tiempo real*. Como ya se ha explicado en el capítulo 2, un sistema *de tiempo real* es aquel en el que para considerar que su funcionamiento es correcto no basta con que sus resultados sean los esperados, sino que tienen que obtenerse dentro del intervalo de tiempo especificado. Así, en el caso de la aplicación Labclock Web detallada en el capítulo anterior, no se pretende que el reloj gire lo más rápido posible, sino que su giro tarde tanto como se haya indicado en el fichero de configuración. Si no se cumpliera esa condición, a pesar de que el punto rotatorio complete su giro en un tiempo superior (o incluso inferior) al estipulado, se entiende que su funcionamiento no es correcto.

Este tipo de aplicaciones web *de tiempo real* no ha sido estudiado con tanta profusión como las aplicaciones *en tiempo real* y uno de los principales aportes

de esta tesis es precisamente ahondar en ellas, explicando los criterios que conducen a lograr una mayor exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario en una aplicación web.

6.2.2 Recomendaciones genéricas

A pesar de que el desarrollo de aplicaciones web *de tiempo real* debe enfocarse a la consecución de la exactitud y la precisión requeridas para lograr cumplir con los requisitos temporales impuestos, las recomendaciones encaminadas a mejorar el rendimiento de las aplicaciones web también son útiles en las aplicaciones web *de tiempo real* en cuanto que reducen el tiempo de ejecución de sus rutinas, aumentando así la disponibilidad del hilo principal de ejecución.

Dentro de las recomendaciones de mejora del rendimiento en aplicaciones web pueden distinguirse las destinadas a minimizar el consumo de ancho de banda y optimizar la carga de la aplicación web, las que pretenden minimizar y optimizar el acceso al árbol DOM, y las optimizaciones de sintaxis del código en JavaScript.

6.2.2.1 Mejora del uso de la red de comunicaciones

En relación a las recomendaciones orientadas a un mejor uso de la red de comunicaciones, Souders (2008) las resume de la siguiente manera: 1) realizar menos peticiones HTTP, principalmente mediante la agregación de varios recursos del mismo tipo (v. gr., hojas de estilo, *scripts*, iconos); 2) emplear una Red de Envío de Contenido (*Content Delivery Network*, CDN), especializada en el envío rápido de recursos web (v. gr., Akamai, Amazon CloudFront CDN); 3) añadir una cabecera HTTP que indique la caducidad del contenido, de cara a facilitar su almacenamiento en memorias caché intermedias; 4) comprimir el contenido siempre que sea posible, habilitando GZIP (GNU ZIP) a nivel de servidor para el tráfico HTTP; 5) situar las hojas de estilo al comienzo del documento HTML, para facilitar su dibujado; 6) situar los *scripts* al final del documento HTML, para evitar que su descarga bloquee el dibujado de la página; 7) evitar las expresiones CSS en las hojas de estilo, para que se puedan procesar más rápidamente; 8) almacenar las hojas de estilo y los *scripts* en ficheros externos al documento HTML, para facilitar su almacenamiento en memorias caché; 9) reducir el número de consultas DNS, almacenando

los recursos necesarios en un número limitado de dominios diferentes; 10) reducir y comprimir los *scripts* en JavaScript, para facilitar su descarga; 11) evitar las redirecciones, que pueden ocasionar nuevas consultas DNS o aumentando la latencia en la obtención del recurso deseado; 12) eliminar *scripts* duplicados; 13) configurar correctamente las ETags (*Entity Tags*), para mejorar la gestión de la caché y evitar peticiones HTTP innecesarias; y 14) facilitar que el contenido dinámico por AJAX sea susceptible de ser almacenado en una memoria caché, principalmente indicando una fecha de expiración lejana.

Otros autores (Ward, 2012; Zakas, 2010) completan esta lista con otras recomendaciones a este respecto que comparten el objetivo principal de las citadas (i. e., minimizar el consumo de ancho de banda y la latencia de red a través de la compresión, la carga diferida y el uso extensivo de memorias caché), tales como la consulta DNS anticipada (*DNS prefetch*), la carga asíncrona de *scripts* mediante la propiedad `async` añadida al elemento `script` en HTML5, el uso del manifiesto de HTML5, o la inclusión de imágenes en formato Base64 dentro de hojas de estilos o documentos HTML (mediante el esquema Data URI definido en el RFC 2397).

6.2.2.2 **Mejoras del acceso al árbol DOM**

Las recomendaciones orientadas a minimizar el acceso al árbol DOM tratan de reducir la ocurrencia de dos sucesos que degradan el rendimiento de una aplicación web: el redibujado y el reflujo. El redibujado se produce cuando hay un cambio visual que no exige un cambio en la posición de los elementos del árbol DOM en la pantalla (v. gr., el cambio de color de una palabra), mientras que el reflujo exige el cálculo de la nueva posición de los elementos en la pantalla (v. gr., el cambio de tamaño de una caja, que provoca que otras cajas tengan que ser desplazadas).

Teniendo esto en cuenta, cada modificación en elementos o propiedades del árbol DOM que impliquen un repintado o reflujo debería hacerse de una vez, preparando un fragmento de documento (bien mediante `createDocumentFragment`, bien clonando un fragmento existente, o bien podando una rama del árbol DOM) que se incluirá en el árbol DOM una vez esté completo, o preparando una cadena con la definición HTML de los cambios requeridos que se aplicarán mediante

la propiedad `innerHTML` del elemento padre. En ningún caso, por tanto, se deberían modificar propiedades o elementos del árbol DOM dentro de un bucle, a no ser que se hayan invisibilizado primeramente y esas modificaciones no provoquen redibujados ni reflujos.

Por último, no solamente los cambios en la visualización del árbol DOM están penalizados, sino que el mero acceso a sus propiedades también es más lento que el acceso a propiedades de objetos en JavaScript. Por esta razón, es muy conveniente no emplear el árbol DOM para almacenar información en propiedades o elementos. Resulta más eficiente el uso de estructuras de memoria ajenas al mismo. Lo mismo sucede en sentido opuesto, por lo que es recomendable almacenar en variables los valores que vayan a ser empleados desde JavaScript (v. gr., anchura, altura o posición de un elemento en pantalla) siempre que sea posible, reduciendo el número de accesos a las propiedades en el árbol DOM.

6.2.2.3 **Mejora del código JavaScript**

Finalmente, las recomendaciones relativas a la optimización del código JavaScript tienen implicaciones a varios niveles.

A más bajo nivel conviene recordar que finalmente el código JavaScript de nuestra aplicación web será interpretado o compilado por un motor de JavaScript, por lo que conocer su diseño puede aportar mejoras de rendimiento considerables. Por citar un ejemplo de este tipo de mejoras, el motor V8 de Google Chrome optimiza el acceso a valores numéricos que puedan representarse como un entero con signo de 31 bits, por lo que teniendo esto en cuenta puede adaptarse el código JavaScript para minimizar los casos en los que los valores numéricos sobrepasen esa capacidad y de esta manera aprovecharse de esa optimización. Sin embargo, estas optimizaciones a bajo nivel son tan dependientes de la plataforma de ejecución que pueden catalogarse como micro-optimizaciones, poco recomendables. En el otro extremo se encuentran los patrones de diseño específicos para JavaScript (Osmani, 2012; Stefanov, 2010), e incluso las colecciones de antipatrones o diseños a evitar en este lenguaje (Chuan, 2012).

A medio camino entre las optimizaciones a bajo nivel y los patrones de diseño se encuentran los detalles de la sintaxis de JavaScript que pueden suponer una

mejora en su rendimiento (Crockford, 2008). Estas recomendaciones pueden agruparse de la siguiente forma: 1) mejoras en el control de flujo (v. gr., evitar bucles `for-in`, evitar el uso de `try-catch-finally` dentro de un bucle, uso de `||` o el operador ternario para evitar alternativas con `if`, uso de métodos nativos frente a funciones); 2) gestión adecuada del ámbito de las variables (v. gr., evitar el uso de `with`, minimizar el uso de variables globales, crear variables locales para valores fuera del ámbito); 3) uso de las estructuras de memoria y tipos apropiados (v. gr., inicializar vectores y objetos con `[]` y `{}`, usar coerciones, emplear *tries*); y 4) gestión adecuada de eventos (v. gr., delegación de eventos, evitar el reiterado uso de gestores de eventos).

Sin embargo, la rápida mejora de los motores de JavaScript disponibles en los agentes de usuario ha convertido a algunas otras recomendaciones (v. gr., almacenar en variables locales los valores de variables de otros ámbitos o el uso de métodos nativos) en obsoletas (Simon, Simpson, Dalton, y Joel, 2012).

6.2.3 Recomendaciones específicas para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario

Cuando el objetivo principal es cumplir con los requisitos de exactitud y precisión establecidos es necesario supeditar las recomendaciones explicadas en la sección anterior a las que se explican a continuación. Antes de empezar y a pesar de los datos aportados en capítulos anteriores, conviene no olvidar que cada caso concreto puede tener condicionantes (v. gr., necesidad de integración con otras tecnologías en un entorno heterogéneo) que obliguen a obviar estas recomendaciones y adoptar otro tipo de medidas. En este sentido, el conjunto de ejemplos de escenarios apropiados y escenarios poco indicados para el uso de las principales tecnologías web en torno al estándar HTML5 aportado por Garaizar, Vadillo, y López-de-Ipiña (2012) es un buen punto de partida para tomar una decisión al respecto.

Para los casos en los que la decisión sobre la tecnología no está condicionada por ningún factor externo, la regla de oro que debería primar al decantarse por una de ellas es la de tratar de mantener la cola de eventos desocupada durante el mayor

tiempo posible. De esta manera, se primará el uso de aquellas tecnologías que no requieran de eventos para su funcionamiento exacto y preciso sobre las que sí dependan de eventos para lograr su cometido. Por esta razón, se recomienda el uso de animaciones declarativas (mediante CSS Animations o SVG con SMIL) siempre que sea posible. En los casos en los que esto no suceda, ya sea por la imposibilidad de definir el comportamiento deseado de manera declarativa como por la falta de soporte de este tipo de tecnologías, se optará por algunas de las tecnologías de animación procedimental, tratando de evitar que el número de temporizadores necesarios sea alto, bien empleando la API para el control temporal de animaciones procedimentales (`requestAnimationFrame`), bien agrupando los gestores de cada animación en una única función *callback* que se encargue de actualizar el estado de todas las animaciones presentes en la aplicación web en un momento dado.

Como ya se ha mencionado con anterioridad, a pesar de que a través del uso de `requestAnimationFrame` pueda obtenerse un número inferior de FPS que mediante otros métodos (v. gr., *polling* mediante `postMessage`), su ajuste con la tasa de refresco del dispositivo de visualización hace que el número de FPS sea el idóneo y el gasto de recursos no se dispare, mejorando la eficiencia energética de la aplicación.

A más bajo nivel, el uso de transformaciones CSS que fuercen la activación de la aceleración 3D en los casos en los que esté disponible (empleando una traslación nula en el eje Z: `transform: translateZ(0)`) permitirán delegar en la GPU el dibujado de la animación y liberar a la CPU para que pueda encargarse de los gestores de eventos que queden pendientes de ejecución en la cola de eventos.

Asimismo, se debería poner especial cuidado en que el primer argumento de un temporizador estándar de JavaScript (`setTimeout` o `setInterval`) sea una función y no una cadena de texto con código JavaScript. De esta forma, los mecanismos de optimización disponibles en la mayoría de motores JavaScript (v. gr., el compilador Just-In-Time) podrán mejorar el rendimiento de estos gestores de eventos. Para aquellos casos en los que se requiera un intervalo de temporización de 0 ms, se recomienda el uso de la función `setImmediate` de la API Efficient Script Yielding (Mann y Weber, 2011). Lamentablemente, no todos los agentes de usuario implementan esta API, por lo que puede ser necesario emularla a través del paso de mensajes mediante `postMessage` (Baron, 2010) si se quiere burlar la

limitación de 4 ms de intervalo de mínimo de temporización fijada por el estándar, como ya se ha explicado con anterioridad.

En aquellos casos en los que la cola de eventos pueda verse saturada debido a la necesidad de realizar cálculos complejos, se recomienda el uso de Web Workers (Hickson, 2012c), capaces de ejecutarse en otro hilo de ejecución distinto al que tiene acceso a la modificación del interfaz.

Además de los Web Workers, existe la posibilidad de dividir la tarea compleja en un conjunto de subtareas que requieran pequeños periodos de tiempo (v. gr., cada una de las iteraciones en un bucle para el procesado de un vector) y emplear un temporizador para que sean encoladas y procesadas cuando la cola de eventos esté libre. En este sentido es muy apropiado el uso de funciones generadoras o iteradores a través de la palabra reservada `yield` (disponible a partir de JavaScript 1.7). Cuando el intérprete de JavaScript encuentra esta palabra reservada, en lugar de evaluar la función, crea una clausura que contiene el estado de la tarea en cada iteración y permite retomarlo en cada llamada a través del método `next`. En el listado 6.1 se muestra un ejemplo del uso de `yield` en el que la función potencias devuelve un vector con las 10 primeras potencias sucesivas cada vez que es llamada (i. e., inicialmente los 10 primeros cuadrados, seguidamente los 10 primeros cubos, y así sucesivamente). Obsérvese cómo la palabra reservada `yield` no tiene por qué ir en último lugar de la función, sino que puede emplearse en mitad de un bucle e ir seguida por otras sentencias. Cada llamada a este generador mediante el temporizador definido devolverá un nuevo vector con las 10 primeras potencias de grado enésimo. De esta forma, los cálculos implicados en la obtención de cada uno de estos vectores se van encolando sucesivamente y no bloquean el hilo de actualización de la interfaz gráfica.

En cuanto a la exactitud y precisión de la reproducción de contenido auditivo, lo recomendable es emplear la Web Audio API siempre que sea posible. Cuando esto no sea así, la alternativa es el uso del elemento audio de HTML5, bien de forma nativa o bien mediante la Audio Data API, que proporciona propiedades, métodos y eventos adicionales para su gestión. Teniendo en cuenta los resultados mostrados en el capítulo 3, en el caso de que la reproducción de contenido auditivo mediante el elemento audio tenga una latencia inaceptable (v. gr., en Mozilla Firefox 10 sobre GNU/Linux), debería considerarse el uso de alguna tecnología web alternativa que

```
function potencias() {
  var n = 1,
      t = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ];
  while (true) {
    var p = t.slice(0);
    for (var i = 0; i < 10; i++) {
      p[i] = Math.pow(t[i], n);
    }
    yield p;
    n++;
  }
}

var generador = potencias();

setInterval(function () {
    console.log(generador.next());
  }, 10);
```

Listado 6.1: Ejemplo de temporizador que emplea un generador en JavaScript creado a través de la palabra reservada `yield`.

proporcione un equivalente funcional (*polyfill*) de la Web Audio API, de tal forma que el programador web emplee las llamadas de la API y éstas sean ejecutadas bien de forma nativa en los casos en los que la API esté soportada, bien a través de la tecnología web alternativa en los casos en los que la API no forme parte del entorno de ejecución nativo que provee el agente de usuario.

En lo referente al registro de la interacción de usuario, a lo largo del capítulo 4 hemos podido constatar la exactitud y precisión de las marcas temporales de los eventos DOM empleados con este cometido. Este enfoque requiere que la cola de eventos se encuentre libre para que el gestor de evento asociado sea atendido en cuanto se produce el evento. A pesar de esto, el estándar indica que la marca temporal que acompaña al evento se define en el momento de su creación (Pixley, 2000), por lo que es preferible emplear esta marca temporal frente a realizar una llamada a una función de tiempo al principio de la función que gestiona el evento. Además, algunas de las nuevas API proporcionan marcas temporales de alta resolución a través de las propiedades de los eventos (v. gr., la propiedad `elapsedTime` en CSS Animations).

El Web Performance Working Group del W3C está trabajando en nuevas especificaciones que emplean funciones de tiempo de alta resolución tales como: 1) la API High Resolution Time (Mann, 2012), que proporciona marcas temporales bajo demanda a través del método `now`; 2) Performance Timeline (Mann y Wang, 2012), que define la interfaz de definición de datos de mediciones de rendimiento que se emplean en otras API; 3) la API Navigation Timing (Z. Wang, 2012), que permite obtener medidas acerca del tiempo de acceso a la red; 4) la API Resource Timing (Mann, Wang, y Quach, 2012a), que ofrece los tiempos de acceso a los recursos del documento; y 5) la API User Timing (Mann, Wang, y Quach, 2012b), que permite realizar mediciones de alta resolución de los usuarios en sus aplicaciones. Adicionalmente, este grupo de trabajo está definiendo otras API relacionadas con una mejor gestión temporal de los recursos como la API para el control temporal de animaciones procedimentales (Robinson y McCormack, 2012), la API Efficient Script Yielding (Mann y Weber, 2011) que proporciona el método `setImmediate` ya comentado, la API Page Visibility (Mann y Jain, 2012) que permite conocer qué parte de una aplicación web está visible en cada momento y adaptar sus exigencias temporales convenientemente, o el formato HTTP Archive (HAR) que define un formato de intercambio de mediciones de rendimiento entre aplicaciones web (Odvarko y Jain, 2012). A pesar del carácter preliminar de muchas de estas propuestas, conviene seguir de cerca su desarrollo para anticipar la exactitud y precisión temporal de las funcionalidades que serán implementadas en los agentes de usuario web en un futuro próximo.

6.3 Herramientas

Además de conocer qué tecnologías son las recomendadas para poder considerar a la Web como una plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario, es preciso contar con un conjunto de herramientas que permitan evaluar el entorno de ejecución del agente de usuario concreto en el que se despliega la aplicación web, estudiar el rendimiento de los diferentes componentes de la aplicación y favorecer un desarrollo dirigido por pruebas o comportamientos.

6.3.1 Análisis de la exactitud y precisión temporal con TickTackJS y AllanJS

Independientemente de los resultados mostrados en capítulos anteriores o de las recomendaciones explicadas a lo largo de este capítulo, el entorno de ejecución concreto en el que una aplicación web se ejecuta puede variar debido a factores externos (v. gr., programas en segundo plano, capacidad de la batería, configuraciones del perfil de usuario). Por esta razón, una aplicación web con altos requisitos temporales debería analizar la exactitud, precisión y resolución de las funciones de tiempo y mecanismos de temporización que vaya a utilizar para evitar desagradables sorpresas. Con este objetivo en mente, hemos desarrollado dos bibliotecas en JavaScript, TickTackJS y AllanJS, cuya funcionalidad explicamos a continuación.

TickTackJS (Garaizar, 2012) es una biblioteca en JavaScript cuyo principal propósito es servir de ayuda en el análisis de la exactitud, precisión y resolución de funciones de tiempo y mecanismos de temporización. Para su diseño se han tenido en cuenta las recomendaciones de Lilja (2000) sobre el uso de métricas para evaluar el comportamiento de soluciones informáticas, sobre todo en lo que respecta a la elección de los estadísticos apropiados para este caso.

El funcionamiento de TickTackJS es muy sencillo. Una vez creado un test (con el método `TickTack.Test`), pueden emplearse dos métodos para el análisis temporal: `benchmark` y `log`. El método `benchmark` recibe dos argumentos, el primero es la función de tiempo que será analizada y el segundo es el número de muestras consecutivas que se desea obtener de la función. Así, si se realiza la llamada `benchmark(Date.now, 1000)`, el test TickTackJS invocará 1000 veces a la función `Date.now` y almacenará los resultados obtenidos en la propiedad `dataset` del test). El método `log` está pensado para facilitar el análisis de temporizadores. Por ello, habitualmente se emplea dentro de la función *callback* asociada a un temporizador. Este método puede recibir un argumento que corresponde a la marca temporal que se desea almacenar o bien ser invocado sin argumentos para que él mismo obtenga la marca temporal correspondiente empleando la función de tiempo estándar en JavaScript.

El listado 6.2 muestra un ejemplo en el que se obtienen muestras tanto de una función de tiempo (`Date.now`) como de un temporizador (`setInterval`) con

TickTackJS. En el caso de la función de tiempo se emplea el método `benchmark` para obtener un millón de muestras, mientras que para el temporizador se utiliza el método `log` llamado a través de `bind` para poder definir el test como objeto `this` en la llamada. El temporizador periódico `intervalTimer` estará almacenando muestras hasta que otro temporizador aperiódico, `timeoutTimer`, fijado a 10.000 ms finalice su ejecución (mediante la función `clearInterval`).

```
1  var testDate = new TickTack.Test('Test Date'),
2      testInterval = new TickTack.Test('Test setInterval'),
3      intervalTimer = setInterval(testInterval.log.bind(testInterval), 50),
4      timeoutTimer = setTimeout(function () {
5        clearInterval(intervalTimer);
6      }, 10000);
7
8  testDate.benchmark(Date.now, 1000000);
```

Listado 6.2: Obtención de muestras de una función de tiempo y un temporizador empleando TickTackJS.

Una vez obtenidas las muestras necesarias puede procederse a su análisis a través del método `analyze`, que realiza un análisis estadístico descriptivo (valor máximo y mínimo, media, DT, error típico de la media, mediana, primer cuartil y tercer cuartil) de los datos almacenados en el `dataset` y de las diferencias entre muestras consecutivas, así como un histograma de su distribución.

Si el valor mínimo de esas diferencias fuera inferior a cero, estaríamos ante una función de tiempo que no es monotónicamente creciente y potencial fuente de muchos problemas en la medición de tiempos. Esto puede comprobarse a través del método `getMonotonicity`.

Dado que para cada una de las recogidas de muestras se almacena una marca de tiempo inicial y otra final, es posible inferir el coste de cada llamada a la función de tiempo dividiendo el periodo de tiempo empleado entre el número de muestras recogidas. El método `getCost` se encarga de calcular este valor.

Otro método de análisis es `getGranularity`, que permite obtener la granularidad de una función de tiempo o diferencia mínima entre dos muestras consecutivas no iguales dentro del conjunto de marcas temporales. Sin embargo, como

explican Kuperberg y cols. (2009), este valor puede no corresponderse con la granularidad real de la función de tiempo, sino con su cota mínima dentro del umbral de ± 1 medida de resolución (en el caso de marcas temporales estándar de JavaScript, ± 1 ms), debido a los errores que se producen al redondear el valor del temporizador interno ofrecido por la función de tiempo. Por esta razón, tal y como sugieren estos investigadores, lo recomendable es establecer agrupaciones (*clusters*) de valores y calcular la separación entre ellos. TickTackJS proporciona para ello el método `getGranularityClusters`, que a su vez se apoya en el método `getClusters`. El método `getClusters` halla las agrupaciones existentes en el conjunto de muestras obtenidas en función de un umbral, que inicialmente tiene el valor de 1 medida de resolución, aunque puede ser configurado. Una vez obtenidas las agrupaciones, `getGranularityClusters` evalúa la granularidad de la función de tiempo de forma similar al algoritmo TimerMeter de Kuperberg y cols. (2009). Para ello, emplea los valores máximos y mínimos de cada una de ellas para hallar el punto central del mismo (sin tener en cuenta la distribución de muestras dentro de la agrupación, puesto que lo que se pretende contrarrestar son los procedimientos de redondeo subyacentes) y con ellas poder calcular las distancias entre agrupaciones consecutivas. Finalmente se devuelve un análisis (máximo, mínimo, media, DT, mediana, etc.) de todas las distancias entre agrupaciones calculadas.

Con el objeto de facilitar el análisis de los datos recogidos, TicktackJS ofrece asimismo dos métodos (`getDataPlot` y `getHistogramPlot`) para exportar los valores de las muestras o diferencias entre ellas y de los respectivos histogramas al formato JSON adecuado para que la biblioteca gráfica Flot genere las correspondientes gráficas. Las figuras que se muestran en lo que resta de capítulo han sido generadas de esta manera.

A continuación, se muestra el análisis mediante TickTackJS de la exactitud y precisión de las marcas temporales estándares de JavaScript en Internet Explorer 8 sobre Microsoft Windows XP, con el objetivo de comprobar empíricamente que sufren limitaciones debidas al uso del reloj general del sistema (*clock wall*), que se actualiza únicamente cada 15–16 ms. Para ello, se han obtenido 10.000 muestras consecutivas y se han procesado mediante el método `analyze` de TickTackJS. La figura 6.4 muestra el conjunto de muestras en un gráfico en el que se puede observar a simple vista cómo el valor de las marcas temporales se actualiza cada 15–16

ms y se mantiene constante el resto del tiempo. Después de procesar el conjunto de datos mediante el método `getGranularityClusters` se corrobora que existen 15 agrupaciones cuyos puntos centrales están separados por 15 o 16 ms (M: 15,571 ms, DT: 0,495), lo que concuerda con estudios previos sobre esta cuestión (Krishnan, 2008; Russinovich y cols., 2009).

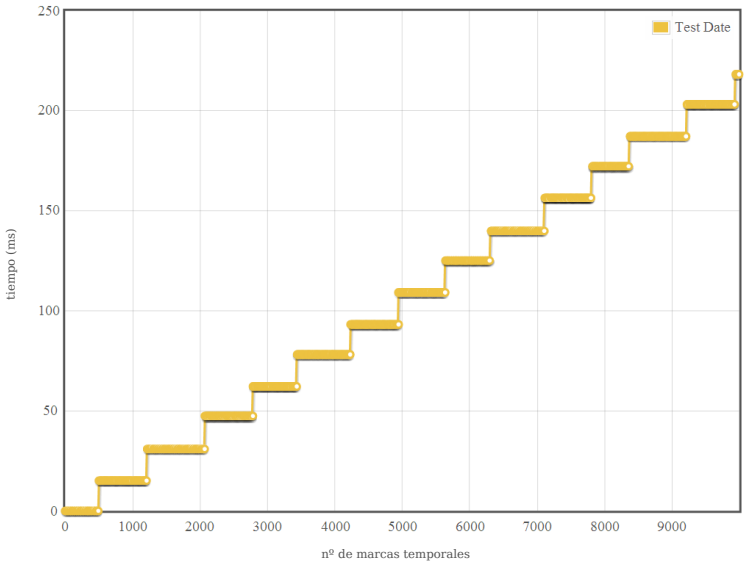

Figura 6.4: Conjunto de 10.000 marcas temporales estándares de Javascript obtenidas mediante TickTackJS en Internet Explorer 8 sobre Windows XP.

De la misma forma, gracias a TickTackJS se pueden descartar alternativas tecnológicas que inicialmente parecen provechosas. El siguiente análisis es un ejemplo de cómo la resolución de una función de tiempo no es el único factor para garantizar su idoneidad. Para que una función de tiempo sea adecuada tiene que cumplir unas características, entre las que destacan las siguientes: 1) monotonicidad, ha de ser monotónicamente creciente para evitar mediciones de tiempo negativas; 2) granularidad, cuanto menor sea la granularidad, menores serán los intervalos de tiempo que podrán medirse; y 3) coste de llamada, cuanto menor sea el coste de llamada a la función de tiempo, menos impacto tendrá este en el intervalo medido.

Teniendo en cuenta los resultados del análisis anterior, la captura de las marcas temporales en este análisis se ha realizado sobre GNU/Linux, ya que en el eventual caso de que alguna de las funciones de tiempo analizadas empleara el reloj del sis-

tema, éste proporciona una exactitud, precisión y resolución por debajo de 1 ms. El análisis comienza con la obtención de un millón de marcas de tiempo mediante tres mecanismos diferentes: 1) la función `Date.now`, 2) la propiedad `currentTime` del contexto de audio de la Web Audio API, y 3) el uso de la función de tiempo de alta resolución `chrome.Interval`. Todos estos mecanismos han sido probados en Google Chrome 17 sobre GNU/Linux. El resultado del análisis de estos datos se muestra en la tabla 6.2, mientras que la figura 6.5 muestra el gráfico que forman cada una de esas marcas temporales a lo largo del tiempo.

Función de tiempo	**Monotónica**	**Granularidad**		**Coste de llamada**
		M	**DT**	
Date.now	Sí	1,070166	0,962001	0,001998
AudioContext.currentTime	Sí	4,74074	10,43108	0,000554667
chrome.Interval	Sí	0,00186	0,032585	0,001860842

Tabla 6.2: Monotonicidad, granularidad y coste de llamada para muestras de un millón de marcas de tiempo obtenidas mediante diferentes mecanismos en Google Chrome 17 sobre GNU/Linux.

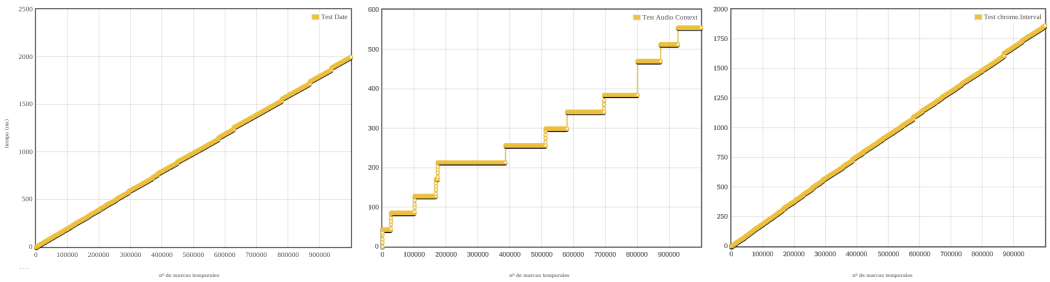

Figura 6.5: Conjunto de 1.000.000 marcas temporales con la función Date.now, con la propiedad currentTime del contexto de audio de la Web Audio API y con la función de tiempo de alta resolución chrome.Interval obtenidas mediante TickTackJS en Google Chrome 17 sobre GNU/Linux.

A tenor de los resultados mostrados, puede decirse que a pesar de que la pro-

piedad `currentTime` del contexto de audio de la Web Audio API es monotónicamente creciente y su coste de llamada está por debajo de 1 μs, su granularidad es inaceptable ya que se encuentra muy por encima de 1 ms. Por lo tanto podemos concluir que la función de tiempo aportada por la extensión de análisis de rendimiento de Google Chrome es la más adecuada para medir intervalos de tiempo, seguida de la función estándar de JavaScript –cuya granularidad impide la medición de intervalos por debajo de 1 ms– y finalmente la propiedad `currentTime` del contexto de audio de la Web Audio API, cuyo uso debería descartarse por ofrecer peores resultados que la función de tiempo estándar.

Dado que en la figura 6.5 no pueden apreciarse las diferencias entre `Date.now` y `chrome.Interval`, la figura 6.6 aporta una visión pormenorizada para 1000 muestras de cada una de ellas, donde sí es posible observar a simple vista cómo la granularidad de `chrome.Interval` es notablemente menor que la de `Date.now`.

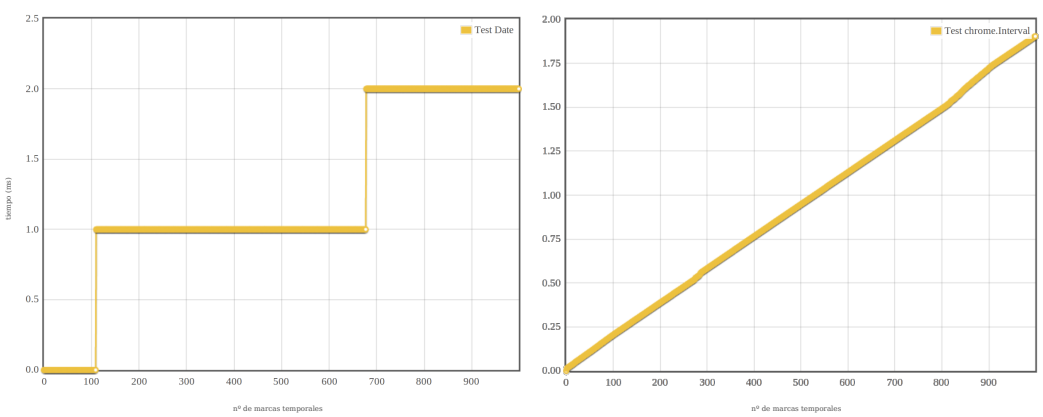

Figura 6.6: Conjunto de 1000 marcas temporales con la función Date.now y la función de tiempo de alta resolución chrome.Interval obtenidas mediante TickTackJS en Google Chrome 17 sobre GNU/Linux.

De igual manera, podemos emplear TickTackJS para comprobar mecanismos de temporización que presumiblemente pudieran suponer una mejora frente a los temporizadores nativos de JavaScript. Así, teniendo en cuenta los resultados del capítulo 3, alguien podría pensar que puesto que las animaciones CSS generan un evento al comenzar, en cada iteración y al finalizar, pueden servir de reemplazo de los temporizadores estándar de JavaScript (`setTimeout` y `setInterval`), incluso con funcionalidades añadidas como la posibilidad de configurar el número

exacto de iteraciones, o diferentes funciones *callback* para la primera o la última iteración. Para comprobar si esto es cierto o no, preparamos dos pruebas con TickTackJS. En la primera configuramos un temporizador periódico (`setInterval`) con 10 ms de intervalo y un temporizador aperiódico (`setTimeout`) con un millón de ms de intervalo para detener el temporizador periódico anterior. En la función *callback* del temporizador aperiódico se realiza una llamada al método `log` de TickTackJS para guardar una marca de tiempo del momento en el que fue ejecutada. La segunda prueba preparada es idéntica, salvo por el mecanismo de temporización, que en este caso se llevará a cabo mediante la biblioteca CSS3Clocks, desarrollada *ex profeso*, capaz de proporcionar temporizadores periódicos y aperiódicos a través de animaciones CSS.

Tras los 1000 s de cada prueba, los resultados obtenidos son los mostrados en las figura 6.7. Como puede observarse, la práctica totalidad de las marcas temporales obtenidas mediante los temporizadores estándar entra dentro del umbral de ± 1 medida de resolución (1 ms), aunque se dan algunos errores aislados (M: 10,195, DT: 0,407), con máximos de hasta 23 ms de error, que provocan que únicamente se obtengan 98.085 en lugar de las 100.000 esperadas. Sin embargo, en el caso de las marcas temporales obtenidas mediante los temporizadores de CSS3Clocks se observa un patrón claro en torno a los 16 ms (M: 16,169, DT: 0,538), que podría estar relacionado con la dependencia de este mecanismo de temporización con la tasa de refresco del monitor, controlada por el motor de dibujado del agente de usuario web. Debido a esta diferencia, únicamente se obtuvieron 62.839 marcas temporales en lugar de las 100.000 esperadas.

Como hemos visto, TickTackJS puede emplearse para comprobar en tiempo real la calidad de funciones de tiempo y temporizadores en aplicaciones web que se ejecuten en entornos en los que haya diferentes opciones y elegir la más adecuada en cada caso. Sin embargo, hay un factor que no se está teniendo en cuenta en estos análisis y es de gran importancia de cara a obtener buenas mediciones de tiempo, la estabilidad.

Los errores sistemáticos que afectan a la exactitud de un temporizador son fácilmente corregibles. Si se conoce que un temporizador siempre ofrece una medida que es, por ejemplo, 30 ms mayor a la real, bastaría con sustraer esa cantidad para compensar el error. De la misma manera, si un temporizador no está ajustado en su

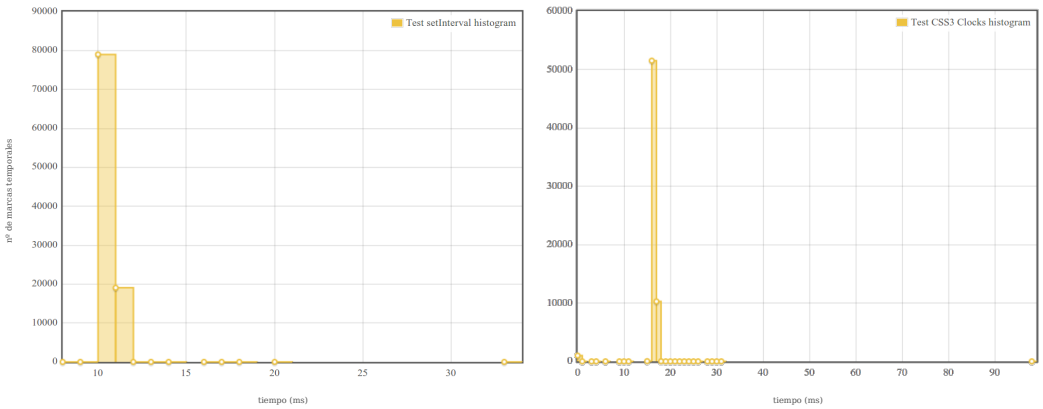

Figura 6.7: Histogramas de las marcas temporales obtenidas mediante el uso de setInterval y el uso de animaciones CSS en Google Chrome 17 sobre GNU/Linux.

frecuencia, actualizará su valor a un ritmo menor o mayor que lo esperado, pero si este desfase es constante también podrá corregirse empleando el factor adecuado. Así pues, lo que distingue la calidad entre temporizadores no es la exactitud de sus mediciones o de su frecuencia, sino la consistencia de estos valores a lo largo del tiempo, o lo que es lo mismo, su estabilidad.

Hasta comienzos de la década de los 60 del siglo XX el análisis de la estabilidad de un temporizador se realizaba empleando estadísticos tradicionales como la varianza o la desviación típica. Sin embargo, a mediados de esa década, Allan (1966) propuso una nueva varianza para muestras no estacionarias, que variaran con el tiempo: la varianza de dos muestras o varianza de Allan.

La varianza y desviación típica clásicas presuponen la independencia del tiempo de la muestra, por lo que cuanto mayor sea el tamaño de la muestra, más convergerán a valores concretos. Para datos que no son estacionarios, cada nueva medida que se añade a la muestra provoca divergencias en la varianza y desviación típica clásicas. La varianza de Allan, en cambio, ofrece una medida de la dispersión de una muestra dependiente del tiempo que sí converge a un valor concreto conforme se van añadiendo más mediciones a la misma tomadas a intervalos regulares.

En la actualidad es más común hablar de la desviación de Allan que de la varianza de Allan, aunque al igual que ocurre con la varianza y desviación típica clásicas, la primera no es más que la raíz cuadrada de la segunda. Para su cálculo se emplean las primeras diferencias $y_{i+1} - y_i$ de los datos de la frecuencia fraccional

o las segundas diferencias $x_{i+2} - 2x_{i+1} + x_i$ de los datos de fase. Así, el estimador original propuesto por Allan para el cálculo de la desviación de dos muestras (ADEV, *Allan Deviation*) en el dominio del tiempo es:

$$\sigma_y^2(\tau, N) = \frac{1}{2\tau^2(N-2)} \sum_{i=0}^{N-3} (x_{i+2} - 2x_{i+1} + x_i)^2 \tag{6.1}$$

Sin embargo, este estimador no ofrece altos niveles de confianza para valores altos del tiempo promedio. Por este motivo, a lo largo del tiempo se han ido desarrollando otros estimadores que ofrecen mayores niveles de confianza (v. gr., varianza de Allan con solapamiento, varianza de Allan modificada, varianza de tiempo, varianza de Hadamard, varianza total, varianza total modificada, Thêo1, ThêoH). Así, la versión con solapamiento del estimador de la varianza de Allan supone una mejora tal frente al estimador original que en la actualidad se emplea este estimador al referirse a la desviación de Allan en la literatura. En el dominio del tiempo, la desviación de Allan con solapamiento se estima como:

$$\sigma_y^2(n\tau_0, N) = \frac{1}{2n^2\tau_0^2(N-2n)} \sum_{i=0}^{N-2n-1} (x_{i+2n} - 2x_{i+n} + x_i)^2 \tag{6.2}$$

En la tabla 6.3 se muestran las principales características de los diferentes estimadores propuestos para la varianza de Allan (Riley y Howe, 2008).

Conceptualmente la varianza de Allan puede entenderse como una predicción, basada en el cálculo del promedio de diferentes intervalos en el pasado, de cómo de amplia será la variación estimada de las mediciones tomadas en el futuro (Van Baak, 2009). Por lo tanto, si para un valor de tiempo promedio τ = 100 s, la desviación de Allan es de $3,24 \times 10^{-}2$, significa que, basado en todos los valores recogidos en el pasado con una distancia de 100 s, la frecuencia del temporizador se mantendrá constante en un rango de $3,24 \times 10^{-}2$ dentro de 100 s. Como para una misma muestra de datos se pueden elegir diferentes tiempos promedio (τ), es posible representar en un gráfico la evolución de la estabilidad (σ) en función de los tiempos promedio, en lo que se conoce como un gráfico sigma-tau. Estos gráficos se presentan con escala logarítmica en ambos ejes. En la figura 6.8 se muestra un ejemplo de la evolución de la estabilidad en función de diferentes tiempos promedio empleando distintos estimadores de la desviación de Allan, en el que se puede observar la estabilidad esperada del temporizador.

Varianza	Características
Estándar	No convergente para medidas no estacionarias.
Allan	Clásica, no ofrece altos niveles de confianza para valores altos del tiempo promedio.
Allan con solapamiento	De propósito general. La más usada.
Allan modificada	Empleada para distinguir entre ruidos W y F PM.
Tiempo	Basada en la varianza Allan modificada, para el dominio del tiempo.
Hadamard	No se ve afectada por la deriva de frecuencia y soporta ruido divergente.
Hadamard con solapamiento	Mayor nivel de confianza que Hadamard.
Total	Mayor nivel de confianza para valores altos de tiempo promedio que Allan.
Total modificada	Mayor nivel de confianza para valores altos de tiempo promedio que Allan modificada.
Tiempo total	Mayor nivel de confianza para valores altos de tiempo promedio que Tiempo.
Hadamard total	Mayor nivel de confianza para valores altos de tiempo promedio que Hadamard.
Thêo1	Proporciona información de toda la muestra.
ThêoH	Híbrido entre Allan y ThêoBR (Thêo1 bias-removed).

Tabla 6.3: Tipos de varianza y sus características (Riley y Howe, 2008).

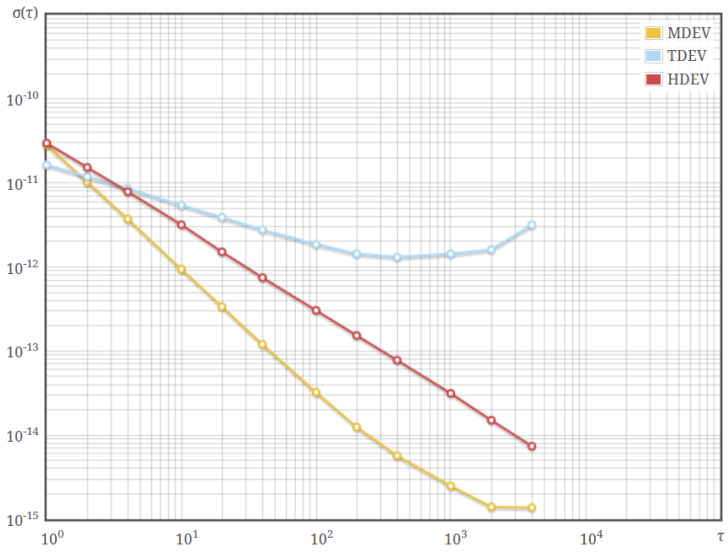

Figura 6.8: Gráfico sigma-tau de la desviación de Allan modificada, la desviación de tiempo y la desviación Hadamard para una muestra de 10.000 mediciones.

A pesar de que la estabilidad de los temporizadores empleados no es un factor relevante en la mayoría de aplicaciones web actuales debido a la escasa duración de

su ejecución continuada, se espera que en el futuro las aplicaciones web puedan cubrir todo tipo de usos, incluidos aquellos que requieren su ejecución durante largos periodos de tiempo de forma continua (v. gr., una aplicación de control domótico doméstico funcionando ininterrumpidamente). Por este motivo, hemos decidido implementar una biblioteca para el cálculo de estimadores de la desviación de Allan desde el entorno de ejecución del agente de usuario web, AllanJS.

AllanJS (Garaizar, 2012) pretende ofrecer lo necesario para realizar el análisis de la estabilidad temporal desde JavaScript, permitiendo: 1) la carga de la muestra tanto desde un vector como mediante la descarga mediante AJAX de un recurso externo, 2) la conversión de la muestra del dominio del tiempo al de la frecuencia y viceversa, 3) el cálculo de los estadísticos básicos de la muestra, 4) el cálculo de un gran número de estimadores de la desviación de Allan (desviación de Allan original, desviación de Allan con solapamiento, desviación de Allan modificada, desviación de tiempo, desviación de Hadamard, desviación de Hadamard con solapamiento, desviación total, desviación total modificada, desviación de tiempo total y desviación de Hadamard total), y 5) la creación de gráficos de fase o frecuencia, así como gráficos sigma-tau con la evolución de los diferentes estimadores en función del tiempo promedio.

Dado que el cálculo de algunos de los estimadores de la desviación de Allan requiere muchas iteraciones (principalmente los estimadores totales), AllanJS trata de almacenar en una memoria caché todo valor que se solicite para evitar realizar dos veces el mismo cálculo o inferir valores en función de otros previamente calculados (v. gr., el cálculo de la desviación de tiempo total en función de la desviación de Allan total modificada).

Aún así, para muestras de gran tamaño el tiempo de cómputo necesario puede exceder el valor establecido por el motor de JavaScript del agente de usuario para la duración máxima de un programa y provocar la aparición de un diálogo en el que se pregunte al usuario si desea interrumpir la ejecución del mismo. En los principales agentes de usuario web es posible modificar su configuración para evitar este comportamiento. Así, en Mozilla Firefox es necesario acceder a la configuración (`about:config` en la barra de direcciones) y cambiar el valor de la clave `dom.max_script_run_time`, mientras que en Google Chrome es necesario lanzar su ejecución desde un terminal con la opción

`--disable-hang-monitor`. En el caso de Internet Explorer, es necesario modificar una clave del registro de Windows[1]) para poder superar el límite de 5 millones de instrucciones por programa, que es el valor por defecto.

Otra solución, más recomendable como ya hemos comentado anteriormente, es la de crear un generador mediante la palabra reservada `yield`, bien para iterar el cálculo de los estimadores, bien para iterar entre los diferentes valores de tiempo promedio (τ), e invocarlo sucesivamente dentro de la función callback de un temporizador periódico con un intervalo que ofrezca la posibilidad de liberar la cola de eventos en ocasiones.

6.3.2 Desarrollo dirigido por pruebas o comportamientos

El desarrollo dirigido por comportamientos (*Behavior-Driven Development*, ver North (2006)) es una evolución del desarrollo dirigido por pruebas (*Test-Driven Development*, ver Beck (2003)) en la que el principal cambio es conceptual, ya que el procedimiento llevado a cabo es muy similar. Ambas metodologías de desarrollo son muy recomendables para el desarrollo web, dadas las peculiares características de este entorno: lenguaje interpretado con tipado débil, escasos mensajes de error (en ocasiones, erróneos), gran heterogeneidad de entornos de ejecución (agentes de usuario, versiones, sistemas operativos, complementos), limitadas herramientas de depuración, etc. Aun así, el desarrollo basado en el ensayo-error y la depuración mediante mensajes por consola es una práctica habitual en la programación web.

En un desarrollo dirigido por pruebas el punto de partida es la definición de un test unitario capaz de comprobar la adecuación a los requisitos de un fragmento de código. A pesar de que a primera vista pudiera parecer que el procedimiento es opuesto a la lógica (comprobar antes de hacer), la interpretación que se hace desde el desarrollo dirigido por comportamientos es la de plasmar los requisitos del sistema en especificaciones formales. De esta forma el desarrollo de una aplicación se realiza mediante un proceso cíclico que consta de los siguientes pasos: 1) plasmar en un test uno de los requisitos de la aplicación; 2) comprobar que el test falle, ya que no está implementado todavía ese requisito (si el test no fallase, habría un error en su definición); 3) programar lo mínimo para que la aplicación supere el test

[1] `HCKU\Software\Microsoft\InternetExplorer\Styles\MaxScriptStatements`

correctamente; 4) refactorizar el código de la aplicación para eliminar el código redundante o mejorar su estructura (aplicando patrones de diseño); y 5) comprobar que los cambios no han hecho que la aplicación no supere los test anteriores. Con cada iteración de este procedimiento la aplicación irá cumpliendo con cada uno de los requisitos planteados en el análisis.

Los beneficios de esta metodología de desarrollo son claros. En contra de la habitual tendencia a «programar primero y comprobar después» que prevalece dentro de muchos equipos de desarrollo web, el hecho de que haya que plasmar los requisitos en test obliga a tener clara la funcionalidad de la aplicación. Si resulta difícil definir la funcionalidad esperada de un componente, quizá sea que no se tienen claros sus requisitos. Diseñar un buen test exige tener en cuenta algunos casos potencialmente problemáticos, por lo que más adelante se tendrá mayor cuidado al codificar la funcionalidad en la aplicación para que pueda pasar el test. Además, disponer de un adecuado número de test ayuda a aislar y localizar un posible error de implementación tras la refactorización del código de la aplicación, por lo que se reducen las reticencias a este tipo de medidas que redundan en una mejor arquitectura del software.

A pesar de esto, el desarrollo dirigido por test o comportamientos no está exento de inconvenientes. La dificultad inicial para definir un buen test, o la sensación de que se está perdiendo un tiempo muy valioso en algo que realmente no aporta una mayor funcionalidad a la aplicación son dos ejemplos del tipo de críticas que recibe. Sin embargo, dadas las dificultades mencionadas anteriormente para la depuración de aplicaciones web (heterogeneidad de plataformas, escasos mensajes de error, características del lenguaje, etc.), el uso de una metodología que minimice el número de errores en el código termina provocando una reducción del tiempo de desarrollo necesario, por lo que está especialmente recomendado.

Entre las herramientas que facilitan el desarrollo dirigido por test destaca la familia de bibliotecas xUnit (v. gr., QUnit, jqUnit, JsUnit), mientras que para el desarrollo guiado por comportamientos en JavaScript es Jasmine la biblioteca de referencia, frente a una amplia oferta de bibliotecas derivadas de otros lenguajes (v. gr., JSpec derivado de RSpec en Ruby).

El uso de estas bilbiotecas puede suponer una gran ventaja para el desarrollo de aplicaciones web con altos requisitos temporales gracias a funcionalidades específi-

cas como el simulador del reloj de JavaScript de Jasmine, que permite realizar comprobaciones simulando el paso del tiempo y, por lo tanto, permitiendo tanto un ahorro considerable de tiempo a la hora de comprobar la aplicación como la posibilidad de simular condiciones de ejecución difíciles de reproducir en la vida real. Este simulador temporal se instala mediante la función `jasmine.Clock.useMock` dentro de una especificación y a partir de ese momento puede simularse el paso del tiempo mediante la función `jasmine.Clock.tick`, que recibe como argumento el número de ms que se quiere avanzar en el tiempo. Si hubiera eventos que deberían dispararse entre el momento actual y el número de ms simulados, los gestores de eventos asociados se dispararían en el orden en el que lo harían en una prueba en tiempo real. En el listado 6.3 se muestra un ejemplo del uso de este simulador temporal dentro de una especificación en Jasmine para la biblioteca TickTack JS explicada anteriormente.

6.3.3 Análisis del rendimiento

Tan importante como el uso de mecanismos de temporización apropiados en una aplicación web con altos requisitos temporales es asegurarse de que el rendimiento de sus gestores de eventos es el adecuado. Por esta razón, el desarrollo dirigido por test o comportamientos y la evaluación del rendimiento son las dos caras de la misma moneda en este tipo de aplicaciones.

Tal y como se ha comentado con anterioridad, una de las métricas más empleadas para el análisis del rendimiento es el tiempo de ejecución, dado que se trata de una métrica lineal, confiable, repetible, consistente, fácil de medir e independiente (Lilja, 2000)).

La manera más sencilla de medir del tiempo de ejecución es de forma manual, tomando una marca de tiempo antes y después de la ejecución del fragmento de código evaluado. Sin embargo , si la granularidad de la función de tiempo empleada para obtener las marcas temporales es inferior al tiempo de ejecución del código evaluado, es posible estimar su tiempo de ejecución ejecutando el código evaluado en varias ocasiones. Además, es necesario analizar de antemano si la ejecución sucesiva del código evaluado puede tener un impacto en su rendimiento, bien sea éste

```
describe("TickTackJS", function() {
  var test;

  beforeEach(function() {
    test = new TickTack.Test('my test');
    jasmine.Clock.useMock();
  });

  it("should log timestamps using a timer", function() {
    setInterval(function() {
      test.log();
    }, 100);

    jasmine.Clock.tick(1);
    expect(test.getDataset().length).toEqual(0);

    jasmine.Clock.tick(100);
    expect(test.getDataset().length).toEqual(1);

    jasmine.Clock.tick(50);
    expect(test.getDataset().length).toEqual(1);

    jasmine.Clock.tick(50);
    expect(test.getDataset().length).toEqual(2);
  });
});
```

Listado 6.3: Uso del simulador temporal en una especificación en Jasmine para TickTackJS.

positivo (v. gr., compilación JIT) o negativo (v. gr., consumo de la memoria disponible para el hilo de ejecución), para que esta estimación se aproxime al tiempo de ejecución real.

Otra precaución importante que hay que tener en cuenta es que ciertas herramientas de desarrollo web añadidas como extensiones o complementos a los agentes de usuario (v. gr., Firebug para Mozilla Firefox) pueden provocar el incorrecto funcionamiento de algunas de las optimizaciones del motor de JavaScript, alterando los resultados obtenidos. Por este motivo, el uso de suites especializadas en el análisis del rendimiento de código JavaScript como Benchmark.js o su contraparte social jsPerf resulta muy conveniente.

Benchmark.js es una biblioteca de evaluación del rendimiento diseñada para ser

usada en todas las plataformas en las que funciona JavaScript (Adobe AIR, Google Chrome, Mozilla Firefox, Internet Explorer, Opera, Apple Safari, Node.js, Narwhal, RingoJS y Rhino). Dada la escasa resolución de las marcas de tiempo estándar en JavaScript (1 ms), Benchmark.js ofrece la posibilidad de emplear otras funciones de tiempo con mayor resolución. Así, en agentes de usuario con soporte para *applets* Java, incluye una *applet* para exponer el método `System.nanoTime` a JavaScript (ver listado 6.4), en Google Chrome ofrece el uso de `chrome.Interval`, y en Node.js permite aprovechar la funcionalidad del módulo `microtime` para obtener marcas de tiempo con mayor resolución.

```
1 import java.applet.Applet;
2 public class nano extends Applet {
3   public long nanoTime() {
4     return System.nanoTime();
5   }
6 }
```

Listado 6.4: Applet en Java para exponer la función de tiempo System.nanoTime a JavaScript (tomado del fichero nano.java de Benchmark.js)

Otro de los puntos fuertes de Benchmark.js frente al análisis de rendimiento manual es la gran expresividad de su sintaxis para definir pruebas, la gestión de eventos propios relacionados con las baterías de pruebas (v. gr., `start`, `cycle`, `complete`, `abort`, `reset`, `error`, etc.), o la posibilidad de especificar un gran número de detalles como son los tiempos máximos y mínimos de la duración de una prueba, pausas entre cada ciclo de pruebas o la ejecución de pruebas asíncronamente, entre otras. Asimismo, Benchmark.js no se limita a indicar el tiempo de ejecución de cada prueba al ofrecer sus resultados, sino que realiza un análisis estadístico básico (media, varianza, DT, margen de error, error típico de la media) para una mejor comprensión de los mismos.

Podría decirse que jsPerf es la versión social de Benchmark.js. A través de jsPerf es posible definir pruebas de rendimiento de JavaScript desde un entorno web que no exige la instalación de ningún programa adicional. Estas pruebas de rendimiento pueden ser compartidas entre usuarios y probadas en diferentes plataformas

para obtener una visión más global de lo que implica cada prueba dentro de todo el ecosistema de entornos de ejecución de JavaScript. Para ello, jsPerf se apoya en Browserscope, una base de conocimiento acerca del rendimiento de JavaScript generada a partir de los resultados de diversas pruebas realizadas de forma distribuida y accesible desde una API pública.

Cuando el análisis del rendimiento no puede desglosarse en pruebas individualizadas sino que su objeto de análisis es la interacción de los diferentes componentes que conforman la aplicación web ejecutándose en su conjunto, pueden emplearse herramientas como stats.js o xStats.js.

6.4 Conclusiones

A lo largo de este capítulo hemos podido observar cómo es posible alcanzar altos niveles de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario si se pone un especial cuidado en los detalles y las tecnologías empleadas en la web. De igual manera, esta conclusión es corroborada por lo explicado en los tres capítulos anteriores, en los que, primeramente con equipamiento especializado y posteriormente mediante los estudios con personas, se ha aportado una importante cantidad de evidencia experimental a su favor.

El conocimiento de las funcionalidades propias del entorno de ejecución de un agente de usuario web y sus peculiaridades en relación a sus hilos de ejecución, la gestión de eventos o las características de temporizadores y funciones de tiempo, permiten orientar el diseño de aplicaciones con altos requisitos temporales, en las que no solamente es conveniente seguir las recomendaciones de desarrollo web genéricas (optimización de la carga de la aplicación, minimización del acceso al árbol DOM, uso de la sintaxis apropiada para maximizar el rendimiento), sino que también es fundamental elegir aquellas tecnologías que proporcionen una mayor exactitud y precisión temporal.

Asimismo, es fundamental el uso de las herramientas apropiadas para poder medir y comprobar que las recomendaciones anteriores suponen una mejora efectiva en el cumplimiento de los requisitos temporales establecidos: 1) aquellas que permitan el análisis de la exactitud, precisión y estabilidad de los mecanismos

de temporización disponibles (TickTackJS y AllanJS), 2) aquellas que permitan la comprobación sistemática mediante test o especificaciones de comportamiento, y 3) aquellas que permitan la evaluación sistemática del rendimiento del código de la aplicación.

"The purpose of computing is insight, not numbers"

Richard Hamming

CAPÍTULO

7

Conclusiones

A lo largo de esta tesis doctoral, se han descrito y analizado las principales tecnologías involucradas en la comprobación de la adecuación de la Web como entorno de ejecución para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario. En el presente capítulo se recogen las conclusiones de este estudio, resumiendo el proceso llevado a cabo, explicando los resultados de investigación obtenidos, detallando las aplicaciones de la investigación realizada e indicando sus limitaciones. Finalmente, dado que este estudio no concluye aquí, se señalan varias líneas de trabajo futuras.

7.1 Resumen del proceso de investigación

El proceso de investigación llevado a cabo puede desglosarse en un conjunto de fases con una estrecha relación de dependencia entre ellas. Cada una de estas fases tiene unos requisitos de partida y produce unos resultados que, a su vez, pueden constituir total o parcialmente los requisitos de las siguientes fases. A continuación, repasaremos el trabajo de investigación realizado en cada una de estas fases, explicando sus interdependencias.

Fase 1. ***Establecimiento de la hipótesis de este trabajo de investigación***

Requisitos previos:

- *Orientación y soporte por parte de investigadores experimentados.*

Resultados:

- *Definición de la hipótesis de este trabajo de investigación.*

Gracias a la colaboración con el laboratorio de psicología experimental de la Universidad de Deusto (Labpsico), conocimos la existencia de las aplicaciones con altos requisitos temporales en la presentación de contenidos audiovisuales y el registro de la interacción del usuario. Sin embargo, a pesar de la amplia experiencia previa de este laboratorio en la investigación en Internet, corroboramos las dificultades encontradas a la hora de implementar en la Web este tipo de aplicaciones. A pesar de ello, constatamos que muchas de las argumentaciones contrarias al desarrollo de aplicaciones con altos requisitos temporales en la Web se apoyaban en resultados ya obsoletos y no tenían en cuenta los desarrollos llevados a cabo en los últimos años para convertir a la Web en un entorno de ejecución de todo tipo de aplicaciones.

Considerando que los últimos avances en infraestructura y estándares han convertido a la Web en la plataforma de ejecución por excelencia, nos planteamos la siguiente cuestión: ¿es la Web una plataforma de ejecución adecuada para asegurar que la presentación de contenidos audiovisuales y el registro de la interacción del usuario se realicen de forma exacta y precisa?

Fase 2. ***Revisión de la literatura para determinar el estado de la cuestión***

Requisitos previos:

- *Definición de la hipótesis de este trabajo de investigación.*

Resultados:

- *Comprobación de que la hipótesis no ha sido validada o refutada con anterioridad.*

- *Definición de los límites previos conocidos de exactitud y precisión en las diferentes entidades implicadas (usuario, hardware y software).*

- *Conocimiento sobre los procedimientos y equipamiento necesarios para estudiar la exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario.*

El estudio del estado de la cuestión tuvo en cuenta a todas las entidades involucradas en el uso de una aplicación con altos requisitos temporales en la presentación de contenidos audiovisuales y el registro de la interacción del usuario. Así, se revisaron estudios previos acerca del comportamiento de los usuarios en la Web, de la exactitud y precisión del hardware empleado (teclados, ratones, pantallas táctiles, monitores, audio, temporizadores) y de los mecanismos de temporización disponibles en los principales sistemas operativos, poniendo especial interés en aquellos que tienen un impacto en el comportamiento de los mecanismos de temporización de las distintas tecnologías de desarrollo web.

Además de definir los límites conocidos de exactitud y validez de las entidades mencionadas, también pudimos conocer qué procedimientos y equipamiento se han utilizado para estudiar estas características y para cuáles de las tecnologías implicadas los datos disponibles son insuficientes o inexistentes (v. gr., CSS, SMIL, SVG, Canvas, WebGL).

Fase 3. ***Análisis de tecnologías***

Esta fase se descompone en el análisis de tecnologías para la presentación exacta y precisa de contenidos audiovisuales y para el registro preciso de la interacción del usuario.

1. ***Análisis de tecnologías offline y online para la presentación exacta y precisa de contenidos audiovisuales***

   Requisitos previos:

   - *Conocimiento sobre los procedimientos y equipamiento necesarios para estudiar la exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario.*

Resultados:

- *Definición de los límites de exactitud y precisión en la presentación de contenidos audiovisuales de las tecnologías analizadas.*
- *Determinación de la precedencia en cuanto a exactitud y precisión en la presentación de contenidos audiovisuales de las tecnologías analizadas.*

Para el análisis de la exactitud y precisión del software offline especializado en la presentación de contenidos audiovisuales se empleó el BBTK (Plant y cols., 2004) y se prepararon varias animaciones con transiciones no graduales de negro a blanco repetidas en las que se varió la duración de cada pantalla, probándose las duraciones de 1000, 500, 200, 100, 50 y 16,667 ms en E-Prime 2.0.8.90 y DMDX 4.0.4.8 sobre Microsoft Windows 7 Professional 32-bit edition con Service Pack 1, y PsychoPy 1.64.00 tanto en la misma versión de Windows como sobre Ubuntu Linux 10.04 LTS "Lucid Lynx" 32-bit con el núcleo Linux 2.6.33-29-realtime. Los resultados obtenidos confirman que el software offline evaluado es apropiado para la implementación y despliegue de experimentos con altos requerimientos de precisión y exactitud en la presentación de contenidos visuales (DT por debajo de 1 marco perdido en todas las condiciones analizadas).

En el análisis de la exactitud y precisión de la presentación de contenidos audiovisuales mediante tecnologías web también se empleó un procedimiento muy similar. En este caso se prepararon animaciones con duraciones de 500, 100, 50 y 16,667 ms en tecnologías web clásicas (GIF89a, Flash, Java, Silverlight) y tecnologías web modernas (CSS3, SMIL, Canvas, SVG, WebGL). En el caso de las tecnologías web clásicas los resultados son también satisfactorios (media de marcos perdidos inferior a 1 con DT en torno a 0,5 para Flash y Silverlight, y valores ligeramente superiores para Java), mientras que los resultados de las nuevas tecnologías web analizadas sugieren el uso de tecnologías que soporten animaciones declarativas (CSS o SVG con SMIL) para aquellos casos en los que el intervalo de animación sea mayor o igual a 50 ms (medias de marcos perdidos entre -0,06 y 0,00 y DT entre 0,459 y 1,107), ya que además de ser una tecnología estable en el tiempo, no se vale

de la cola de eventos de JavaScript y es independiente del grado de saturación de la misma.

En cuanto al contenido auditivo, al combinar las nuevas tecnologías web de presentación de estímulos visuales con las nuevas API de reproducción y generación de contenido auditivo se comprobó la existencia de alternativas (Web Audio API en Google Chrome 17) que igualan e incluso superan en exactitud y precisión a las tecnologías offline evaluadas (DMDX sobre Windows 7 Professional SP 1).

2. ***Análisis de tecnologías offline y online para el registro preciso de la interacción del usuario***

   Requisitos previos:

   - *Conocimiento sobre los procedimientos y equipamiento necesarios para estudiar la exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario.*

   Resultados:

   - *Definición de los límites de exactitud y precisión en el registro de la interacción del usuario de todas las tecnologías analizadas.*
   - *Determinación de la precedencia en cuanto a exactitud y precisión en el registro de la interacción del usuario de todas las tecnologías analizadas.*
   - *Herramienta de generación de la entrada de usuario exacta y precisa para GNU/Linux (Metronome LKM).*

   Para el análisis de la exactitud y precisión de tecnologías offline y online en el registro de la interacción del usuario, se desarrolló Metronome LKM, un controlador con temporizadores de alta precisión para la generación de eventos de teclado, y Logkeys X-Window, una aplicación para obtener un registro de la cadencia de eventos de teclado.

   En las diferentes pruebas se generaron eventos de teclado con cadencias de 1000, 500, 100, 50, 10, 5 y 1 ms a través de Metronome LKM y fueron

registradas tanto por Logkeys X-Window como por una sencilla aplicación web que emplea las marcas temporales de los eventos DOM estándar. Los resultados mostraron una exactitud y precisión por debajo de tanto para las tecnologías offline como para las online.

Fase 4. ***Desarrollo de aplicaciones experimentales***

Esta fase se descompone en el desarrollo de una aplicación experimental offline con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario y el desarrollo de una aplicación de las mismas características mediante tecnologías web.

1. ***Desarrollo de una aplicación experimental offline con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario***

   Requisitos previos:

   - *Conocimiento sobre la precedencia en cuanto a exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario de todas las tecnologías offline analizadas.*

   Resultados:

   - *Aplicación experimental offline con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario (Labclock).*

   Para el desarrollo de esta aplicación se tuvieron en cuenta los resultados obtenidos durante las fases previas. Por esta razón, Labclock emplea los temporizadores multimedia, así como las llamadas a la API Kernel32 de Win32 `QueryPerformanceFrequency` y `QueryPerformanceCounter` para garantizar el cumplimiento de sus altos requisitos temporales.

   Dado que el paradigma experimental que esta aplicación implementa (el reloj de Libet) emplea una animación con un punto rotatorio, se encontraron

problemas para evaluar la exactitud y precisión de la presentación de la misma mediante el BBTK, por lo que se comprobó a través de la grabación de un vídeo con una cámara de alta velocidad a 1000 marcos por segundo.

2. ***Desarrollo de una aplicación experimental online con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario***

   Requisitos previos:

   - *Conocimiento sobre la precedencia en cuanto a exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario de todas las tecnologías online analizadas.*

   Resultados:

   - *Aplicación experimental online con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario (Labclock Web).*

   Para el desarrollo de la versión web de la aplicación también se tuvieron en cuenta los resultados obtenidos en fases previas, por lo que se emplearon los últimos estándares web en torno a HTML5 para garantizar el cumplimiento de los elevados requisitos temporales exigidos: 1) CSS Animations para la presentación de contenidos visuales, 2) Web Audio API para la presentación de contenidos auditivos, y 3) marcas de tiempo de eventos DOM para el registro de la interacción del usuario.

   La comprobación de la exactitud y precisión de la presentación de la animación del paradigma experimental implementado también se realizó a través de la grabación de un vídeo con una cámara de alta velocidad.

Fase 5. ***Experimentación cognitiva empleando una aplicación offline y online con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario***

Requisitos previos:

- *Aplicaciones experimentales offline y online con presentación exacta y precisa de contenidos audiovisuales y registro de la interacción del usuario (Labclock y Labclock Web).*

Resultados:

- *Comprobación de la validez externa de las aplicaciones offline y online desarrolladas.*
- *Obtención del efecto de «unión temporal» con demoras de 500 y 1000 ms para juicios de decisión y de acción.*

Una vez desarrolladas las aplicaciones offline y online se comprobó su utilidad y validez externa mediante una serie de 6 experimentos en la que participaron más de 300 sujetos experimentales. En todos ellos se obtuvo el efecto principal de «unión temporal» siguiendo el paradigma experimental del reloj de Libet, tanto mediante tecnologías offline (Labclock) como mediante tecnologías web (Labclock Web). Estos resultados tienen valor tanto desde el punto de vista de la psicología cognitiva como desde el punto de vista tecnológico y metodológico, ya que gracias a ellos queda demostrada la viabilidad del uso de tecnologías web para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción de usuario.

Fase 6. ***Definición de unas recomendaciones de desarrollo para el uso de la Web como plataforma de ejecución para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario***

Requisitos previos:

- *Conocimiento sobre la precedencia en cuanto a exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario de todas las tecnologías online analizadas.*
- *Comprobación de la validez externa de la aplicación online desarrollada.*

Resultados:

- *Recomendaciones de desarrollo para aplicaciones web con altos requisitos temporales.*

- *Herramientas de análisis de temporizadores en el entorno de ejecución de los agentes de usuario web (TickTackJS y AllanJS).*

Tras la fase de experimentación, se procedió a recopilar todas las recomendaciones para el desarrollo de aplicaciones web con altos requisitos temporales obtenidas a partir del análisis de las tecnologías, el desarrollo de las aplicaciones y su puesta a prueba mediante la experimentación, así como otras recomendaciones de desarrollo web más generales que puedan tener su influencia en este ámbito.

Dada la heterogeneidad de entornos de ejecución para aplicaciones web, consideramos que estas recomendaciones deberían complementarse con un conjunto de herramientas que permitan evaluar los mecanismos de temporización disponibles en cada situación. Por esta razón, desarrollamos TickTackJS y AllanJS, además de sugerir el uso de otras muchas herramientas ya disponibles (v. gr., Benchmark.js, Jasmine) para facilitar el desarrollo de aplicaciones web con altos requisitos temporales.

Fase 7. ***Validación de la hipótesis de este trabajo de investigación***

Requisitos previos:

- *Conocimiento sobre la precedencia en cuanto a exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario de todas las tecnologías online analizadas.*
- *Comprobación de la validez externa de la aplicación online desarrollada.*

Resultados:

- *Validación de la hipótesis.*

Finalmente, la última fase de este proceso de investigación está destinada a la validación de la hipótesis inicial. Teniendo en cuenta la gran cantidad de resultados congruentes con la hipótesis presentada, tanto en forma de estudios previos, el análisis de tecnologías de presentación de contenidos y registro de respuesta, la comprobación del desarrollo de aplicaciones propias y la evidencia experimental obtenida mediante la colaboración de un gran número de participantes, estimamos razonable considerar a la Web como una plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario.

7.2 Resultados de la investigación

Como consecuencia del proceso de investigación descrito anteriormente, ha sido posible dar respuesta a las cuestiones de investigación planteadas al comienzo del mismo.

Cuestión 1. *¿Cuál es el grado de exactitud y precisión en la presentación de contenidos audiovisuales empleando software especializado?*

Según los resultados presentados en el capítulo 3 de esta tesis, la media de marcos perdidos para E-Prime 2.0.8.90 y DMDX 4.0.4.8 sobre Microsoft Windows 7 Professional 32-bit edition con Service Pack 1, y PsychoPy 1.64.00 sobre la misma versión de Windows y sobre Ubuntu Linux 10.04 LTS "Lucid Lynx" 32-bit con el núcleo Linux 2.6.33-29-realtime, se sitúa entre 0,0 y 0,50 en todas las condiciones evaluadas (animaciones con intervalos de 1000, 500, 200, 100, 50 y 16,667 ms) y su DT se sitúa entre 0,0 y 0,88 marcos perdidos.

Cuestión 2. *¿Cuál es el grado de precisión y exactitud en la presentación de contenidos audiovisuales empleando tecnologías web propietarias sobre agentes de usuario modernos?*

Tal y como se explica en el capítulo 3, la exactitud y precisión temporal en la presentación de contenido visual de las tecnologías web clásicas es adecuada (media de marcos perdidos por inferiores a 1) por encima de los 100 ms de intervalo. En el caso de Adobe Flash, la media del número de marcos perdidos es inferior a 0,1 en todos los casos salvo en las pruebas realizadas sobre Internet Explorer 9 con intervalos de 16,667 ms, y la DT es inferior a 1 en todos los casos. En lo que respecta a Java, el escaso rendimiento obtenido en Google Chrome 17 e Internet Explorer 9 (en torno a 1 marco perdido de media en todas las condiciones) contrasta con los buenos resultados obtenidos en Mozilla Firefox 10 (por debajo de 0,4 marcos perdidos de media en todas las condiciones). Finalmente, el rendimiento de Silverlight es muy similar al obtenido en Flash salvo en el intervalo más corto (16,667 ms).

Cuestión 3. *¿Cuál es el grado de precisión y exactitud en la presentación de contenidos audiovisuales empleando tecnologías web estándar sobre agentes de usuario modernos?*

Los resultados obtenidos en el capítulo 3 para este tipo de tecnologías pueden resumirse de la siguiente forma: 1) las animaciones declarativas con CSS han resultado ser la opción más recomendable cuando los intervalos de animación se sitúan por encima de los 50 ms (media entre -0,06 y 0,00 marcos perdidos y DT entre 0,459 y 1,107), 2) los resultados de SMIL con SVG son similares a CSS, si bien su grado de implementación en los distintos agentes de usuario es menor, 3) el rendimiento de las tecnologías procedimentales (Canvas, SVG, WebGL con `requestAnimationFrame`) es similar al de tecnologías web previas (media de marcos perdidos entre 0,06 y 1,50 con DT entre 0,252 y 1,055, frente a medias entre 0,00 y 2,18 con desviación típica entre 0,109 y 3,182, ver tabla 3.13 para mayor detalle).

Cuestión 4. *¿En qué medida estos tres conjuntos de tecnologías presentan diferencias en cuanto a la exactitud y precisión en la presentación de contenidos audiovisuales?*

Teniendo en cuenta la gran cantidad de muestras tomadas en cada una de las pruebas de las tecnologías evaluadas, todas las diferencias entre ellas resultan significativas. Sin embargo, puede establecerse una ordenación en cuanto a la exactitud y precisión entre las tecnologías de los conjuntos planteados (aplicaciones offline especializadas, tecnologías web propietarias y las tecnologías web estándar) en la que, para intervalos superiores a 50 ms, DMDX sería el más preciso, seguido de E-Prime, Flash, PsychoPy, SMIL, CSS, Silverlight, Canvas, WebGL, SVG, GIF89a y Java (ver tabla 7.1).

Cuestión 5. *¿Cuál es el grado de precisión y exactitud que puede lograrse en el registro de la interacción mediante tecnologías web estándar?*

Los resultados de las pruebas presentadas en el capítulo 4 indican que tanto en Mozilla Firefox 10 como en Google Chrome 17 el uso de las marcas temporales de los eventos DOM estándar para el registro de la interacción del usuario mediante teclado presentan una gran exactitud (media entre 0,00 y 0,40 ms) y precisión (DT entre 0,00 y 0,64 ms).

Tecnología	M	DT
DMDX	0	0,026
E-Prime	0	0,026
Flash	0,03	0,364
PsychoPy	0,1	0,489
SMIL	0	0,695
CSS	0	0,72
Silverlight	0,03	0,823
Canvas	0,56	0,632
WebGL	0,57	0,617
SVG	0,65	0,657
GIF89a	0,43	1,307
Java	0,9	2,049

Tabla 7.1: Clasificación de las distintas tecnologías offline y online según el número de marcos perdidos en las pruebas realizadas con intervalos superiores a 50 ms.

Cuestión 6. *A tenor de los resultados obtenidos anteriormente, ¿existen aplicaciones con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario que puedan implementarse empleando tecnologías web estándar?*

Tal y como se explica al concluir el capítulo 4, si se tiene en cuenta que: 1) con intervalos de animación iguales o superiores a 50 ms, las animaciones declarativas en CSS son la tecnología web estándar más recomendable para la presentación exacta y precisa de contenido audiovisual, 2) la Web Audio API permite la generación y temporización de contenido auditivo con una exactitud y precisión por debajo de 1 ms, y 3) el registro de la interacción del usuario mediante las marcas de tiempo de eventos DOM tiene una exactitud y precisión por debajo de 1 ms, se puede concluir que es posible definir un conjunto de tecnologías web estándar compatible que permita el desarrollo de aplicaciones con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro

de la interacción del usuario.

Cuestión 7. *Si la cuestión anterior resulta ser cierta, ¿es una aplicación de este tipo implementada mediante tecnologías web estándar capaz de obtener resultados equivalentes a una aplicación* offline *implementada a través de los mecanismos de temporización óptimos disponibles en el sistema?*

En el capítulo 5 se detalla tanto el desarrollo de Labclock Web, una aplicación web que emplea tecnologías estándar para implementar el paradigma experimental del reloj de Libet, así como de Labclock, una aplicación offline para Microsoft Windows que emplea temporizadores multimedia y funciones de tiempo de alta precisión (`QueryPerformanceCounter`). Asimismo, se muestra cómo tanto en los tres experimentos realizados con Labclock Web como en los tres experimentos llevados a cabo con Labclock se encontró el efecto principal de «unión temporal» propio de este paradigma experimental.

Cuestión 8. *Finalmente, en el caso de que exista evidencia experimental que corrobore la cuestión anterior, ¿sería posible definir unas recomendaciones de desarrollo para la implementación de una aplicación con elevados requisitos de exactitud y precisión en la presentación de contenidos audiovisuales y el registro de la interacción del usuario empleando tecnologías web estándar?*

Tal y como se recoge a lo largo del capítulo 6, el conjunto de resultados previamente obtenidos, unidos a las recomendaciones generales para la optimización de aplicaciones web, el estudio del entorno de ejecución de JavaScript en los agentes de usuario web y el uso de las herramientas adecuadas permite la definición de unas recomendaciones de desarrollo para la implementación de este tipo de aplicaciones mediante tecnologías web estándar.

Dicho lo cual, estamos en condiciones de afirmar que el conjunto de los resultados obtenidos es coherente con la hipótesis de partida:

Hipótesis. *«La Web es una plataforma de ejecución adecuada para la presentación exacta y precisa de contenidos audiovisuales y el registro de la interacción del usuario.»*

7.3 Aplicaciones y limitaciones de la investigación

Considerando que parte de los resultados obtenidos en esta investigación son netamente aplicados (ver capítulo 5), resulta evidente afirmar que la viabilidad de las aplicaciones web con una presentación de contenidos audiovisuales y registro de la interacción del usuario exactos y precisos supone un avance metodológico considerable para la experimentación cognitiva a través de la Web. El hecho de que no haya otras implementaciones del paradigma experimental de Libet con tecnologías web hasta la fecha es una muestra de ello. Asimismo, sería posible implementar otros paradigmas experimentales, que actualmente no pueden llevarse a cabo en la Web, basándose en las recomendaciones de desarrollo y la metodología de análisis aquí descritas.

De forma similar, los resultados parciales mostrados en los capítulos previos pueden tener su aplicación en otros ámbitos no estrictamente relacionados con la experimentación cognitiva, tales como el desarrollo de laboratorios remotos, la creación de simulaciones o videojuegos basados en tecnologías web o la retransmisión de contenidos multimedia sincronizados. A pesar de que muchas de las tecnologías aquí descritas se considerarán obsoletas en pocos años, los resultados obtenidos siempre podrán emplearse a modo de cota mínima de las capacidades de las tecnologías web para presentar contenidos y registrar la interacción de forma exacta y precisa.

La principal limitación de este trabajo de investigación reside en la escasa atención prestada a las plataformas web móviles. Varias razones explican esta decisión: 1) la gran fragmentación existente en el ecosistema de plataformas móviles, tanto en lo que se refiere al hardware (grandes diferencias en capacidad de cómputo, visualización, audio, interacción con el usuario, etc.) como al software (múltiples sistemas operativos incompatibles entre sí y repartidos en distintas versiones con diferencias significativas entre ellas, ver figura 7.1); 2) la limitada oferta de agentes de usuario web con las funcionalidades estudiadas para estas plataformas (ver figura 7.2); 3) el bajo grado de adopción de los nuevos estándares web en los agentes de usuario móviles; y 4) las limitaciones espaciales y temporales de los dispositivos de visualización e interacción, que pueden suponer la imposibilidad de llevar a cabo

un experimento en ellos, o lo que es peor, un importante sesgo en las respuestas de ese participante (Reimers y Stewart, 2008).

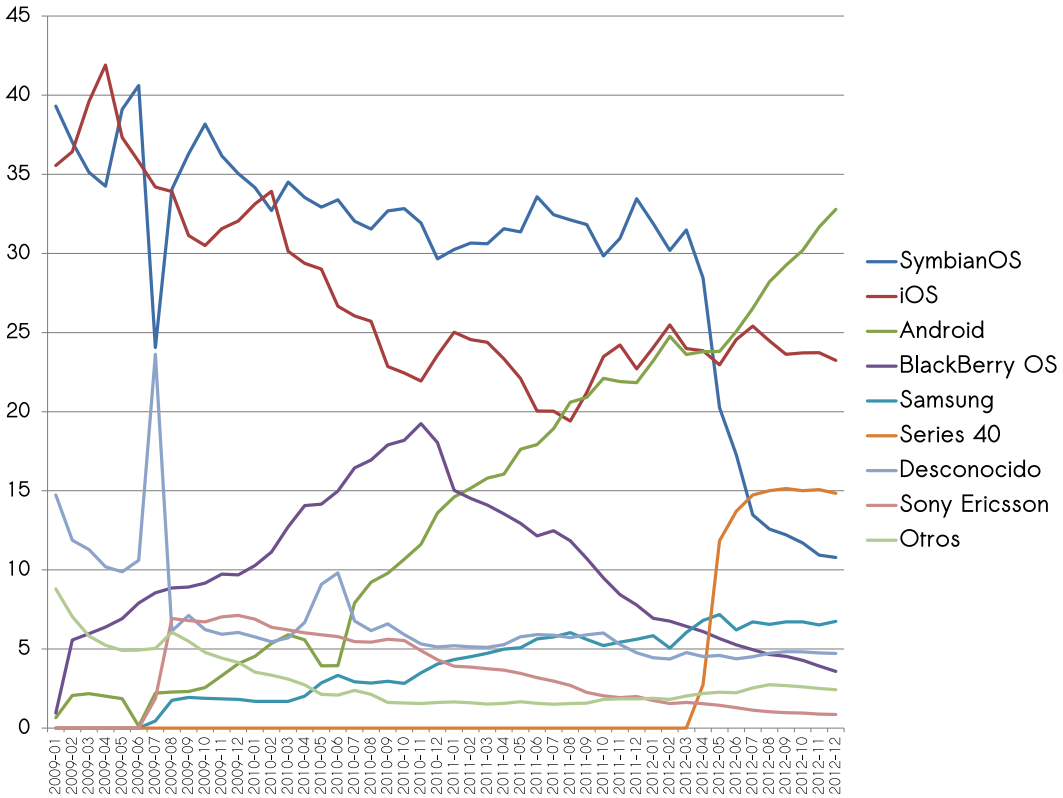

Figura 7.1: Evolución del porcentaje en cuota de mercado de los principales sistemas operativos móviles entre 2008 y 2012 (fuente: StatCounter).

Sin embargo, la Web será eminentemente móvil en pocos años, por lo que esta limitación debe entenderse también como una línea futura de trabajo, preferiblemente una vez se haya llegado a un grado de desarrollo del acceso a la Web en movilidad tal que su rendimiento y capacidades sea comparable al de los agentes de usuario web en ordenadores personales.

7.4 Líneas futuras de trabajo

Tal y como se ha apuntado en la sección anterior, existen varias líneas futuras de trabajo relacionadas con la movilidad (G. Miller, 2012).

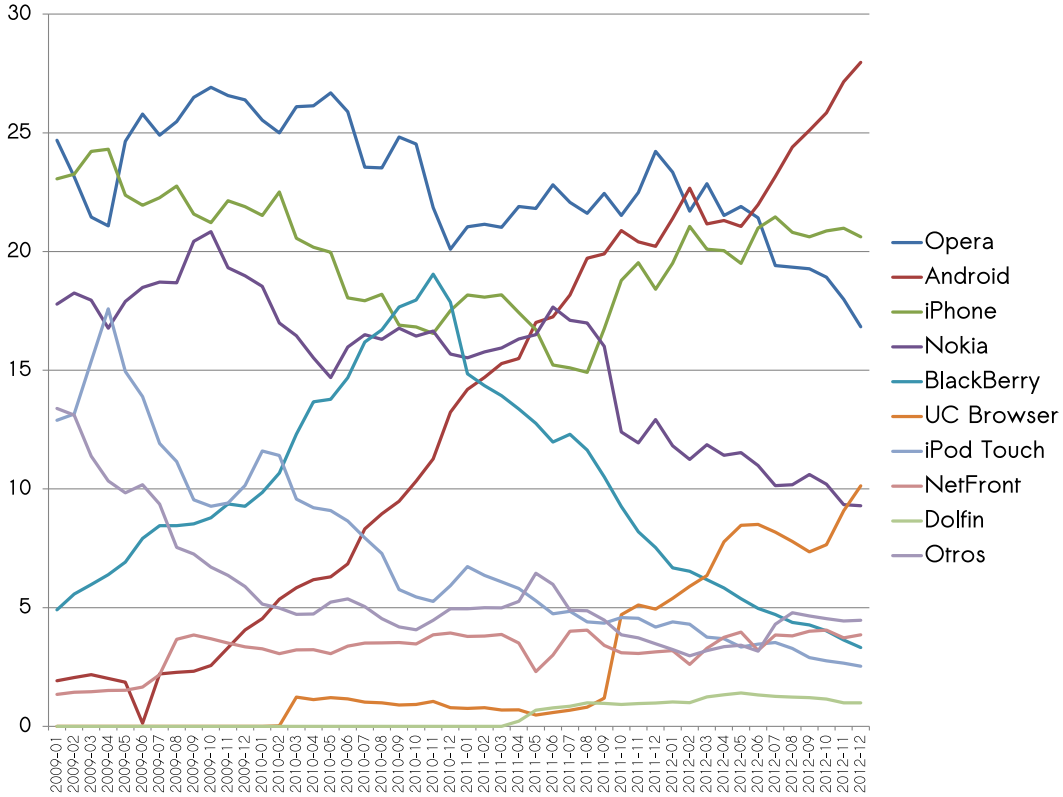

Figura 7.2: Evolución del porcentaje en cuota de mercado de los principales agentes de usuario web móviles entre 2010 y 2012 (fuente: StatCounter).

La primera de ellas tiene que ver con el análisis de la gestión de la concurrencia de los principales sistemas operativos móviles (Apple iOS, Google Android, Microsoft Windows Phone) en lo que respecta a las tecnologías aquí estudiadas. ¿Resultan adecuadas las recomendaciones de desarrollo aquí indicadas para el desarrollo de aplicaciones web en movilidad con altos requisitos temporales? ¿En qué medida el uso de tecnologías que maximicen el uso de la GPU (v. gr., WebGL, animaciones CSS en 3D) puede suponer una mejora en la exactitud y precisión de las aplicaciones web móviles? ¿Supone el uso intensivo de varios subsistemas de dibujado (v. gr., animaciones CSS combinadas con animaciones procedimentales en JavaScript) concurrentemente una mejora en movilidad?

La segunda de ellas tiene que ver con las condiciones cambiantes del entorno de un usuario en movilidad (principalmente la temperatura y el consumo de energía) y su efecto en la exactitud y precisión de los mecanismos de temporización.

La tercera está relacionada con la mejora del registro de la interacción del usuario aprovechando la gran densidad de dispositivos de captura de datos presentes en un dispositivo móvil. Así, podría estudiarse la viabilidad de la captura de la interacción del usuario con la pantalla mediante el dispositivo de grabación de audio para su posterior procesado a través de las distintas API de tratamiento de audio disponibles. De esta manera podría incrementarse la exactitud y precisión de esta medida en varios órdenes de magnitud teniendo en cuenta que incluso en una grabación de muy baja calidad se recogen del orden de 8000 muestras por segundo y lo habitual es capturar 22.050 o 44.100 muestras por segundo, lo que permitiría precisiones muy por debajo de 1 ms.

Otra línea de trabajo futuro ineludible está relacionada con el aprovechamiento de la miríada de nuevas API alrededor que están siendo definidas y desarrolladas en los últimos años del estándar HTML5. Gracias a ellas se están cubriendo todas las funcionalidades que están presentes en el entorno de ejecución nativo y no lo estaban en el entorno de ejecución web, por lo que la concepción de la Web como plataforma de ejecución de aplicaciones universal resulta cada vez más posible con cada una de ellas. A pesar de ello, resulta inevitable la competición entre aquellas API que ofrezcan funcionalidades equivalentes o solapadas, por lo que no todas ellas estarán disponibles a largo plazo. Por este motivo y considerando la apertura de los procesos de definición de estas propuestas, resulta conveniente participar en ellos para argumentar a favor de aquellos mecanismos de temporización más adecuados (tal y como se comentó al finalizar el capítulo 4).

Por último, como propuesta a largo plazo se plantea la línea futura de trabajo hacia la *Real-Time Web*, la Web *de tiempo real*. Si bien son muchos los esfuerzos puestos en la Web *en tiempo real*, donde priman la inmediatez y la velocidad, no son tantos los que centran su atención en el cumplimiento estricto de los requisitos temporales establecidos. Si se desea presentar a la Web como la plataforma sobre la cual desplegar todo tipo de aplicaciones, es necesario trabajar en esta línea, que permitiría la adopción de tecnologías web en ámbitos aún no explorados como los sistemas de control de la navegación (v. gr., terrestre, marítima, aeroespacial) o de monitorización (v. gr., salud, seguridad, vigilancia).

7.5 Consideraciones finales

Finalmente, no queda sino embarcar hacia nuevas Ítacas.

Ιθάκη	**Ítaca**
Σα βγείς στον πηγαιμό για την Ιθάκη, *να εύχεσαι νάναι μακρύς ο δρόμος,* *γεμάτος περιπέτειες, γεμάτος γνώσεις.*	Cuando emprendas tu viaje hacia Ítaca debes rogar que el viaje sea largo, lleno de peripecias, lleno de experiencias.
Τους Λαιστρυγόνας και τους Κύκλωπας, *τον θυμωμένο Ποσειδώνα μη φοβάσαι,* *τέτοια στον δρόμο σου ποτέ σου δεν θα βρείς,* *αν μέν' η σκέψις σου υψηλή, αν εκλεκτή* *συγκίνησις το πνεύμα και το σώμα σου αγγίζει.* *Τους Λαιστρυγόνας και τους Κύκλωπας,* *τον άγριο Ποσειδώνα δεν θα συναντήσεις,* *αν δεν τους κουβανείς μες στην ψυχή σου,* *αν η ψυχή σου δεν τους στήνει εμπρός σου.*	No has de temer ni a los lestrigones ni a los cíclopes, ni la cólera del airado Poseidón. Nunca tales monstruos hallarás en tu ruta si tu pensamiento es elevado, si una exquisita emoción penetra en tu alma y en tu cuerpo. Los lestrigones y los cíclopes y el feroz Poseidón no podrán encontrarte si tú no los llevas ya dentro, en tu alma, si tu alma no los conjura ante ti.
Να εύχεσαι νάναι μακρύς ο δρόμος. *Πολλά τα καλοκαιρινά πρωϊά να είναι* *που με τι ευχαρίστησι, με τι χαρά* *θα μπαίνεις σε λιμένας πρωτοειδωμένους·* *να σταματήσεις σ' εμπορεία Φοινικικά,* *και τες καλές πραγμάτειες ν' αποκτήσεις,* *σεντέφια και κοράλλια, κεχριμπάρια κ' έβενους,* *και ηδονικά μυρωδικά κάθε λογής,* *όσο μπορείς πιο άφθονα ηδονικά μυρωδικά·* *σε πόλεις Αιγυπτιακές πολλές να πας,* *να μάθεις και να μάθεις απ' τους σπουδασμένους.*	Debes rogar que el viaje sea largo, que sean muchos los días de verano; que te vean arribar con gozo, alegremente, a puertos que tú antes ignorabas. Que puedas detenerte en los mercados de Fenicia, y comprar unas bellas mercancías: madreperlas, coral, ébano, y ámbar, y perfumes placenteros de mil clases. Acude a muchas ciudades del Egipto para aprender, y aprender de quienes saben.
Πάντα στον νου σου νάχεις την Ιθάκη. *Το φθάσιμον εκεί είν' ο προορισμός σου.* *Αλλά μη βιάζεις το ταξίδι διόλου.* *Καλλίτερα χρόνια πολλά να διαρκέσει·* *και γέρος πια ν' αράξεις στο νησί,* *πλούσιος με όσα κέρδισες στον δρόμο,* *μη προσδοκώντας πλούτη να σε δώσει η Ιθάκη.*	Conserva siempre en tu alma la idea de Ítaca: llegar allí, he aquí tu destino. Mas no hagas con prisas tu camino; mejor será que dure muchos años, y que llegues, ya viejo, a la pequeña isla, rico de cuanto habrás ganado en el camino.
Η Ιθάκη σ' έδωσε το ωραίο ταξίδι. *Χωρίς αυτήν δεν θάβγαινες στον δρόμο.* *Αλλο δεν έχει να σε δώσει πια.* *Κι αν πτωχική την βρεις, η Ιθάκη δεν σε γέλασε.* *Ετσι σοφός που έγινες, με τόση πείρα,* *ήδη θα το κατάλαβες η Ιθάκες τι σημαίνουν.*	No has de esperar que Ítaca te enriquezca: Ítaca te ha concedido ya un hermoso viaje. Sin ella, jamás habrías partido; Y si la encuentras pobre, Ítaca no te ha engañado. Y siendo ya tan viejo, con tanta experiencia, sin duda sabrás ya qué significan las Ítacas.
Κωνσταντίνος Π. Καβάφης	Konstantinos P. Kavafis.

"Talk is cheap. Show me the code"

Linus Torvalds

APÉNDICE

A

Metronome LKM

METRONOME LKM es un controlador para el núcleo Linux 2.6.21 o posterior que emplea temporizadores de alta resolución para la generación exacta y precisa de eventos de teclado. Este controlador es configurable por parámetros en el momento de su carga o bien mediante el sistema de ficheros virtual `sys` y gestionable mediante una petición al sistema a través de la tecla PetSis (SysReq). Además de generar los eventos de teclado requeridos, registra cada uno de ellos mediante funciones de tiempo con resolución de ns (`ktime`) en el fichero de registro del núcleo.

El código fuente de Metronome LKM está publicado bajo la Licencia Pública General (GPL) versión 3 (FSF, 2007).

A. METRONOME LKM

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/input.h>
#include <linux/hrtimer.h>
#include <linux/ktime.h>
#include <linux/sysrq.h>
#include <linux/tty.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Pablo Garaizar");

#define METRO_DELAY    1E9L
#define METRO_KEY      KEY_SPACE
#define METRO_SYSRQ    0x61
#define METRO_STATUS   0

static long metronome_delay = METRO_DELAY;
static unsigned int metronome_key = METRO_KEY;
static int metronome_sysrq = METRO_SYSRQ;
static int metronome_status = METRO_STATUS;

module_param(metronome_delay, long, S_IRUGO | S_IWUSR);
MODULE_PARM_DESC(metronome_delay,
  "delay of the high-resolution timer in ns (default = 1E+9, 1000 ms)");
module_param(metronome_key, uint, S_IRUGO | S_IWUSR);
MODULE_PARM_DESC(metronome_key, "key (default = KEY_SPACE)");
module_param(metronome_sysrq, int, S_IRUGO | S_IWUSR);
MODULE_PARM_DESC(metronome_sysrq, "SysRq key (default = 'a')");
module_param(metronome_status, int, S_IRUGO | S_IWUSR);
MODULE_PARM_DESC(metronome_status, "status (default = 0, off)");

static struct hrtimer hrt;
static struct input_dev *dev;

static void sysrq_handle_metronome(int key, struct tty_struct *tty)
{
  metronome_status = (metronome_status) ? 0 : 1;
}

static struct sysrq_key_op sysrq_metronome_op = {
  .handler      = (void *)sysrq_handle_metronome,
  .help_msg     = "metronome(A)ctivation",
  .action_msg   = "Changing metronome state",
  .enable_mask  = SYSRQ_ENABLE_KEYBOARD,
};

enum hrtimer_restart metronome_hrt_callback(struct hrtimer *timer)
{
  ktime_t now, t;
  unsigned long missed;
```

```
51
52    now = ktime_get();
53    t = ktime_set(0, metronome_delay);
54    missed = hrtimer_forward(&hrt, now, t);
55
56    if (missed > 1)
57    {
58      printk(KERN_INFO "metronome: Missed ticks %ld.\n", missed - 1);
59    }
60
61    if (metronome_status)
62    {
63      input_report_key(dev, metronome_key, 1);
64      input_sync(dev);
65      input_report_key(dev, metronome_key, 0);
66      input_sync(dev);
67      printk(KERN_INFO "metronome: Key event (%lluns).\n", ktime_to_ns(now));
68    }
69
70    return HRTIMER_RESTART;
71  }
72
73  int __init metronome_init(void)
74  {
75    ktime_t t;
76    int ret = 0;
77
78    printk(KERN_NOTICE "metronome: Loaded.\n");
79
80    printk(KERN_INFO "metronome: Registering device...\n");
81    dev = input_allocate_device();
82
83    if (dev)
84    {
85      dev->name = "Generic device";
86      dev->evbit[0] = BIT (EV_KEY) | BIT (EV_REP);
87      set_bit(metronome_key, dev->keybit);
88      ret = input_register_device(dev);
89      if (ret)
90      {
91        printk(KERN_ERR "metronome: Failed to register device.\n");
92        input_free_device(dev);
93        ret = -1;
94      }
95      else
96      {
97        printk(KERN_INFO "metronome: Registering SysRq key (%c)...\n",
98          (char)metronome_sysrq);
99        register_sysrq_key(metronome_sysrq, &sysrq_metronome_op);
100       t = ktime_set(0, metronome_delay);
```

```
      printk(KERN_INFO
        "metronome: Starting high-resolution timer (%lluns)...\n",
        ktime_to_ns(t));
      hrtimer_init(&hrt, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
      hrt.function = &metronome_hrt_callback;
      hrtimer_start(&hrt, t, HRTIMER_MODE_REL);
    }
  }
  else
  {
    printk(KERN_ERR "metronome: Failed to register device.\n");
    ret = -1;
  }

  return ret;
}

static void __exit metronome_exit(void)
{
  printk(KERN_INFO "metronome: Unregistering SysRq key (%c)...\n",
    (char)metronome_sysrq);
  unregister_sysrq_key(metronome_sysrq, &sysrq_metronome_op);
  printk(KERN_INFO "metronome: Stopping high-resolution timer...\n");
  hrtimer_cancel(&hrt);
  printk(KERN_INFO "metronome: Unregistering device...\n");
  input_unregister_device(dev);
  printk(KERN_NOTICE "metronome: Unloaded.\n");
}

module_init(metronome_init);
module_exit(metronome_exit);
```

Listado A.1: Código fuente de Metronome LKM.

"Writing specs is like flossing: everybody agrees that it's a good thing, but nobody does it"

Joel Spolsky

APÉNDICE

B

Propuesta de mejora para marcas temporales en eventos DOM

La resolución es un factor importante en cualquier sistema de registro de marcas temporales. Como ya explicamos en el capítulo 1, la falta de resolución temporal puede provocar que eventos que han sucedidos en momentos distintos sean registrados como coincidentes en el tiempo. A pesar de la aparente sencillez de la solución de este problema a alto nivel –empleemos mejores mecanismos de registro temporal que ofrezcan una mayor resolución–, el escenario se complica cuando el cambio propuesto puede afectar a una amplia muestra de aplicaciones ya desplegadas.

Tal es el caso de las marcas temporales en eventos DOM. Ampliar su resolución implica decidir entre el progreso y la compatibilidad con las aplicaciones existentes. Por esta razón, el Working Group de DOM Events del W3C decidió abrir el proceso de propuestas de mejora para la versión 3 de los eventos DOM. Aprovechando la oportunidad brindada, decidimos colaborar en el proceso sugiriendo la mejora de las marcas temporales de los eventos DOM, dado que los mecanismos subyacentes que las generan (agentes de usuario web, principalmente) son capaces

de proveer una mayor resolución y otras API del W3C están adoptando de forma gradual el tipo de datos `DOMHighResTimeStamp` para sus marcas temporales. En este apéndice se presentan las acciones llevadas a cabo en este proceso.

B.1 Propuesta inicial

La propuesta inicial se realizó en el grupo de trabajo Web Perfomance del W3C, dentro de su proceso de definición de la API High Resolution Time (Mann, 2012). En esta propuesta[1] solicitamos extender el tipo de datos `DOMHighResTimeStamp` no solamente a la salida del método `now` propuesto en la especificación, sino también a la propiedad timeStamp de los eventos DOM, ya que estos eventos adolecen de los mismos problemas que intenta resolver esta API. Anne van Kesteren, empleado de Opera y miembro del mencionado grupo de trabajo, nos sugirió que tratáramos este asunto en el grupo de trabajo sobre DOM del W3C[2].

Dado que en aquel momento se estaba cerrando el proceso público de solicitud de propuestas para la versión 3 de DOM, decidimos que era interesante aprovechar la oportunidad para realizar nuestra sugerencia[3]. En la misma, mencionamos nuestras dudas acerca de la idoneidad de modificar el tipo de datos de la propiedad existente (`timeStamp`) o, por el contrario, incluir una nueva propiedad adicional de tipo `DOMHighResTimeStamp`.

Travis Leithead, empleado de Microsoft y miembro del grupo de trabajo del W3C sobre DOM, respondió que tras hablar con Jatinder Mann, editor de la especificación (Mann, 2012), había conocido los detalles de las funcionalidades que esta nueva API proporciona y había llegado a la conclusión de que cambiar el tipo de datos a la propiedad `timeStamp` podría tener muy malas consecuencias en términos de retrocompatibilidad. Además, Leithead sugirió que el método `now` de la API mencionada cubría gran parte de los escenarios planteados[4].

Como respuesta a Leithead, argumentamos que la propiedad `timeStamp` se define en el momento de creación del evento DOM y no cuando este es atendido, por lo que el uso de `Date` o `now` como primera llamada de la función que

[1] `http://lists.w3.org/Archives/Public/public-web-perf/2012Mar/0033.htm`
[2] `http://lists.w3.org/Archives/Public/public-web-perf/2012Mar/0034.html`
[3] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0069.html`
[4] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0075.html`

atiende el evento podría ser problemático en casos en los que la cola de eventos esté saturada o sean necesarias interacciones con API que empleen el nuevo tipo de datos `DOMHighResTimeStamp`, puesto que no solamente difieren en su resolución, sino también en su momento de referencia (*UNIX-epoch* frente al momento de carga de la página, respectivamente)[1].

Al mismo tiempo, James Robinson, empleado de Google y miembro del grupo de trabajo, sugirió que elaboráramos una lista de casos de uso en los que la propuesta fuera adecuada, de cara a valorarla para la versión 4 de los eventos DOM[2].

B.2 Casos de uso

A raíz de la sugerencia de Robinson, procedimos a redactar una lista de casos de uso en los que la propuesta de ampliación de la resolución de las marcas temporales de los eventos DOM fuera deseable o necesaria [3]. Esta lista fue completada con las aportaciones y comentarios de Jonas Sicking[4] y Glenn Maynard[5], y comprende los siguientes escenarios: 1) sincronización de múltiples fuentes de contenido multimedia y animaciones; 2) registro exacto y preciso de la interacción del usuario; 3) pruebas de rendimiento precisas que requieran el uso de eventos; 4) generación de *feedback* al usuario acorde al momento de su interacción; y 5) generación de animaciones adaptativas a los retrasos producidos por otros eventos.

B.3 Implementación

En octubre de 2012, Robert Flack, desarrollador de Google Chrome, hizo pública la implementación de esta funcionalidad llevada a cabo en Webkit[6]. El desarrollo de esta funcionalidad y las pruebas de uso pueden seguirse en el sistema de gestión de cambios de Webkit[7], donde se menciona nuestra contribución a su proposición y definición.

[1] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0078.html`
[2] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0077.html`
[3] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0092.html`
[4] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0095.html`
[5] `http://lists.w3.org/Archives/Public/www-dom/2012AprJun/0096.html`
[6] `http://lists.w3.org/Archives/Public/www-dom/2012OctDec/0008.html`
[7] `https://bugs.webkit.org/show_bug.cgi?id=94987`

"If the implementation is hard to explain, it's a bad idea"

Tim Peters

APÉNDICE

C

Configuración de experimentos con Labclock y Labclock Web

LABCLOCK y Labclock Web son dos aplicaciones para la realización de experimentos basados en el reloj de Libet. Uno de sus objetivos iniciales fue el de ser empleadas por investigadores sin conocimientos de programación. Por esta razón, su funcionamiento se define a través de ficheros de configuración fácilmente definibles y separados del código de estas aplicaciones. En el presente apéndice se explica cómo crear o modificar estos ficheros de configuración para sacar el mayor provecho a Labclock y Labclock Web.

C.1 Labclock

La configuración de Labclock está alojada en dos tipos de ficheros auxiliares: 1) el fichero de configuración general y 2) los ficheros de experimentos. Los siguientes apartados detallan su formato.

C.1.1 Fichero de configuración general

El fichero de configuración general se denomina `config.ini` y se encuentra en el mismo directorio que el programa `clock.exe`. Este fichero almacena valores globales del experimento tales como el nombre del experimento, el código de paso para poder comenzar el experimento, las rutas a los ficheros de audio y de definición de las fases del experimento, traducciones para los mensajes y botones de la interfaz, y detalles relativos a los mecanismos de temporización (granularidad, opciones de depuración). En el listado C.1 se muestra un ejemplo de este fichero (algunas líneas –marcadas con `[...]`– han sido recortadas por razones de espacio).

```
[Experiment]
Name="Binding"
PassCode="99"

[Files]
GetReadySound="250-440Hz_44100Hz_16bit_1000ms.wav"
FeedbackSound="1kHz_44100Hz_16bit_200ms.wav"
ExperimentFilename="experiment"
ExperimentExtension="xml"
Results="results.csv"

[Messages]
SelectGroup="Select group"
Demo = "Clock in demo mode. Press 'OK' once you undestand how it works."
Code = "Please enter the code provided by the research group to start [...]"
WrongCode = "Wrong code"
TrialReady = "GET READY. Remember that you have to press the SPACEBAR [...]"
TrialSelecting = "In which moment did you decide to press the [...]"
TrialNotSelected = "Please select the moment when you decided to [...]"
TrialSelected = "Get ready for the next trial"
CommandOK = "OK"
CommandPrevious = "< Previous"
CommandNext = "Next >"
CommandFinish = "Finish"

[Timer]
Type = "RSTimer"
Interval = "1"
Debug = "False"
```

Listado C.1: Fichero de configuración general de Labclock.

C.1.2 Ficheros de configuración de experimentos

Los ficheros de configuración de experimentos son el segundo tipo de ficheros de configuración de Labclock. A través de ellos se define el propio experimento en sí, con sus instrucciones iniciales, fases experimentales y explicaciones finales. En el listado C.2 se muestra la definición de la estructura de un fichero de configuración de Labclock en XML.

```
1  <!ELEMENT experiment (title, begin, phases, end)>
2  <!ELEMENT begin (screen*)>
3  <!ELEMENT phases (phase*)>
4  <!ELEMENT end (screen*)>
5  <!ELEMENT screen (title, text)>
6  <!ELEMENT title (#PCDATA)>
7  <!ELEMENT text (#PCDATA)>
8  <!ELEMENT phase (scramble?, progress?, trials, screen)>
9  <!ELEMENT trials (trial*)>
10 <!ELEMENT trial (cycle, tone?)>
11 <!ELEMENT cycle (#PCDATA)>
12 <!ELEMENT tone (#PCDATA)>
13 <!ELEMENT scramble (#PCDATA)>
14 <!ELEMENT progress (#PCDATA)>
```

Listado C.2: Fichero DTD con la definición del formato de ficheros de configuración de Labclock.

Como se puede observar, un experimento está compuesto por un elemento `begin` que contiene un conjunto de elementos `screen` que definen las pantallas de instrucciones iniciales (mediante un elemento `title` y un elemento `text` cada una). Seguidamente, el elemento `phases` engloba un conjunto de elementos `phase` donde se definen cada una de las fases del experimento. Cada una de estas fases tiene un conjunto de ensayos (`trials`) y una pantalla final (`screen`) por si fuera necesario aportar nuevas instrucciones al final de cada fase. Además pueden definirse dos elementos opcionales, `scramble` y `progress`, para activar la aleatorización del orden de ensayos y la barra de progreso, respectivamente. Los ensayos se definen mediante elementos `trial` que contienen la duración de un ciclo (elemento `cycle`) y el elemento opcional `tone` que indica el número de ms

que debe demorarse el tono tras la acción del usuario (si no se define, se tratará de un ensayo sin tono).

Dado que un mismo experimento puede requerir distintas configuraciones por grupo o el contrabalanceo de determinadas variables, se podrán definir tantos ficheros de configuración como sea necesario, siempre que se emplee el prefijo y la extensión configuradas en el fichero `config.ini` (mediante los valores de `ExperimentFilename` y `ExperimentExtension`).

C.1.3 Ficheros de resultados

Labclock almacena sus resultados en un fichero. La ruta a este fichero está definida el fichero de configuración general (`Results` dentro de `config.ini`). Su contenido es el siguiente:

1. Información general acerca del experimento: nombre del experimento, fecha de realización y nombre del ordenador desde donde se lanzó el experimento.

2. Por cada fase se almacena su número, y de cada uno de sus ensayos, las siguientes variables:

   a) Tiempo de respuesta: ms desde que empezó la vuelta hasta que el participante responde (se almacena el valor `99999` si no ha habido respuesta).

   b) Tiempo en el que el participante informa que tomó la decisión de pulsar o pulsó la tecla, en ms (se almacena el valor `99999` si no ha habido respuesta).

   c) Tiempo de retardo en la reproducción del tono, en ms.

Además de estos valores, fundamentales para el tipo de experimentos realizados bajo el paradigma experimental del reloj de Libet, se incluyen otros valores adicionales para cada ensayo con el fin de detectar posibles problemas en el procedimiento o en la ejecución del programa:

1. `InitialRandomTime`: tiempo de espera inicial en ms previo a que el reloj comience a girar (aleatorio entre 1000 y 3000 ms).

2. `Cycle`: tiempo en ms que debe tardarse en completar una vuelta.
3. `ResponseLap1`: variable booleana que recoge si el sujeto ha pulsado la tecla durante la primera vuelta (de ser así, podría considerarse que el ensayo es erróneo, a elección del experimentador).
4. `ResponseTimeLap1`: tiempo en ms el que se ha pulsado la tecla durante la primera vuelta (normalmente no se usa, ya que la pulsación debería darse durante la segunda vuelta).
5. `TimeLap1`: tiempo real en ms que se ha tardado en completar la primera vuelta (aproximadamente el mismo valor que `Cycle` si no se ha producido ninguna demora en el sistema).
6. `ResponseLap2`: variable booleana que recoge si el sujeto ha pulsado la tecla durante la segunda vuelta (valor verdadero en la mayoría de ensayos).
7. `ResponseTimeLap2`: tiempo en ms el que se ha pulsado la tecla durante la segunda vuelta.
8. `TimeLap2`: tiempo real en ms que se ha tardado en completar la segunda vuelta (aproximadamente el mismo valor que `Cycle` si no se ha producido ninguna demora en el sistema).
9. `Tone`: variable booleana que recoge si ha sonado un tono o no (valor verdadero en los ensayos con tono).
10. `ToneTime`: tiempo en ms el que debe sonar el tono.
11. `ToneRealTime`: tiempo real en ms que se ha tardado en hacer sonar el tono (aproximadamente el mismo valor que `ToneTime` si no se ha producido ninguna demora en el sistema).
12. `Guess`: variable booleana que recoge si el sujeto ha emitido un juicio o no (valor verdadero en aquellos ensayos en los que se dispone de `GuessTime`).
13. `GuessTime`: tiempo en ms en el que el participante informa que tomó la decisión de pulsar o pulsó la tecla.

C.2 Labclock Web

Labclock Web carece de un fichero de configuración general. Toda su configuración se almacena en ficheros de configuración de experimentos.

C.2.1 Ficheros de configuración de experimentos

En el listado C.3 se muestra un ejemplo de fichero de configuración resumido (los puntos suspensivos indican omisiones, realizadas para facilitar la comprensión del formato).

Como se puede observar, cada grupo deberá definirse como una propiedad diferente del objeto `experiment`. En este caso la propiedad `A` representa al grupo A en el experimento.

Posteriormente hay una serie de propiedades generales como: 1) `code`, para indicar el nombre o código del experimento; 2) `password`, para definir el código de paso que permitirá comenzar el experimento; 3) `randomDelayMin` y `randomDelayMax`, para definir la demora aleatoria que precede cada ensayo (pueden fijarse ambas a un mismo valor para que la demora sea constante); 4) `postResultsURL`, para indicar la URL a la que se enviarán los resultados al concluir el experimento; 5) `responseKey`, para indicar qué tecla se considerará la tecla de respuesta; 6) `sounds`, un objeto que almacena en sus propiedades las referencias a los sonidos que se emplearán en el experimento para avisar de que comienza un ensayo (`getReady`, mediante una ruta a un fichero de audio) y para el tono (`feedback`, mediante una frecuencia y duración, puesto que el tono se genera procedimentalmente mediante la Web Audio API); 7) `messages`, para localizar el experimento al idioma que sea necesario; 8) `preScreens`, un vector de objetos que representan las pantallas iniciales con su título y contenido; 9) `passwordScreen`, para mostrar el formulario de solicitud del código de paso; 10) `phases`, un vector de objetos que representan cada una de las fases; y 11) `postScreens`, un vector de objetos que representan las pantallas de información finales. Por cada fase puede definirse un texto para su descripción, si se desea mostrar una barra de progreso (muy útil para fases con muchos ensayos) y si se desea desordenar pseudoaleatoriamente los ensayos mediante el algoritmo Fisher-Yates. Seguidamente se define un vector de ensayos, cada uno de los cuales contiene la

duración de un periodo de vuelta del reloj (propiedad `cycle`) y –opcionalmente– la demora en ms del tono (propiedad `tone`). Si no se define esta demora, el ensayo no tendrá *feedback*.

Por último, la definición de la fase concluye con un objeto de tipo `screen` con su título y contenido, que será mostrado una vez terminada ésta.

```
experiment.A = {
  code: 'Binding A',
  password: '99',
  randomDelayMin: 1000,
  randomDelayMax: 3000,
  postResultsURL: 'datasent.asp',
  responseKey: ' ',
  sounds: {
    getReady: { file: 'media/250-440Hz_44100Hz_16bit_1000ms.wav' },
    feedback: { duration: 200, pitch: 1000, }
  },
  messages: {
    commandOK: 'Aceptar',
    ...
  },
  preScreens: [
    { title: '...', content: '...' },
    ...
  ],
  passwordScreen: { title: '...', content: '...' },
  phases: [
    {
      description: 'Binding',
      progress: true,
      scramble: true,
      trials: [
        { cycle: 2560, tone: 1 },
        ...
        { cycle: 2560, tone: 500 },
        ...
      ],
      screen: { title: '...', content: '...' }
    },
    {
      description: 'Post base rate',
      progress: true,
      scramble: false,
      trials: [
        { cycle: 2560 },
        ...
```

```
        ],
        screen: { title: '...', content: '...' }
    ],
    postScreens: [
        screen: { title: '...', content: '...' }
    ]
};
```

Listado C.3: Fichero de configuración de un experimento en Labclock Web.

C.2.2 Ficheros de resultados

Como ya se ha explicado en el capítulo 5, Labclock Web envía los resultados a través de una conexión HTTP Post realizada mediante AJAX. Además, por seguridad, realiza una copia de respaldo empleando la API de Almacenamiento Local (Local Storage API).

De cada ensayo se almacenan los siguientes valores:

1. Tiempo de respuesta: ms desde que empezó el ensayo hasta que el participante responde, puede contener varios valores separados por comas si ha habido más de una respuesta, o tomar el valor `undefined` si no ha habido respuesta.

2. Tiempo en el que el participante informa que tomó la decisión de pulsar o pulsó, en ms; puede tomar el valor `undefined` si no ha habido respuesta.

3. Tiempo de retardo en el tono, en ms (aproximadamente el mismo valor que la propiedad `tone` del objeto que define el ensayo si no se ha producido ninguna demora en el sistema).

Adicionalmente, se almacenan los siguientes datos extra de cada ensayo para detectar posibles problemas en el procedimiento o en el programa:

1. `InitialRandomTime`: tiempo de espera inicial en ms previo a que el reloj comience a girar (configurable mediante los valores de `randomDelayMin` y `randomDelayMax` en el fichero de configuración).

2. `Cycle`: tiempo en ms que debe tardarse en completar una vuelta.
3. `CycleTime`: tiempo real en ms en el que se han completado las dos vueltas del reloj.
4. `Tone`: tiempo en ms en el que debe sonar el tono.
5. `ToneTime`: tiempo real en ms que se ha tardado en hacer sonar el tono (aproximadamente el mismo valor que `Tone` si no se ha producido ninguna demora en el sistema).
6. `KeyPressTrialTimes`: lista de tiempos en ms de las pulsaciones de la barra espaciadora durante un ensayo; si no hay ninguna pulsación, contendrá `undefined`, y si hay pulsaciones, contendrá una lista separada por comas con los tiempos de cada pulsación en ms desde el principio del ensayo.
7. `StartTrialTime`: tiempo en ms en el que da comienzo el ensayo.
8. `EndTrialTime`: tiempo en ms en el que finaliza el ensayo (la resta entre este tiempo y `StartTrialTime` da el tiempo que ha durado el ensayo).
9. `StartTrialAudioTime`: tiempo en s desde el comienzo del primer ensayo, tomado por parte del sistema de audio (propiedad `currentTime` del contexto de audio de la Web Audio API).

Cualquiera de estas variables que no contenga un valor, se almacenará con el valor `undefined`.

Bibliografía

Adams, C. (2010). *HTML5 versus Flash: Animation benchmarking.* Descargado el 10/12/2012, de http://www.themaninblue.com/writing/perspective/2010/03/22/ 66

Allan, D. (1966). Statistics of atomic frequency standards. *Proceedings of the IEEE*, *54*(2), 221–230. 195

Aschersleben, G. (2002). Temporal control of movements in sensorimotor synchronization. *Brain and cognition*, *48*(1), 66–79. 36

Bach, M., Meigen, T., y Strasburger, H. (1997). Raster-scan cathode-ray tubes for vision research-limits of resolution in space, time and intensity, and some solutions. *Spatial Vision*, *10*(4), 403–414. 32

Badle, S., Bakken, J., Barantsev, A., Beans, E., Berrada, D., Bevan, J., ... Wagner-Hall, D. (2012). *Selenium - Web Browser Automation.* Descargado el 10/12/2012, de http://seleniumhq.org 139

Banks, W., y Isham, E. (2009). We infer rather than perceive the moment we decided to act. *Psychological Science*, *20*(1), 17–21. 152, 164

Baron, D. (2010). *setTimeout with a shorter delay.* Descargado el 10/12/2012, de http://dbaron.org/log/20100309-faster-timeouts 108, 184

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional. 199

Beilner, H. (1988). *Measuring with slow clocks*. International Computer Science Institute. 53

Belshe, M. (2010). *Chrome: Cranking up the clock.* Descargado el 10/12/2012, de http://www.belshe.com/2010/06/04/chrome-cranking-up-the-clock/ 66, 177

Beringer, D. (1989). Touch panel sampling strategies and keypad performance comparisons. En *Proceedings of the human factors and ergonomics society annual meeting* (Vol. 33, pp. 71–75). 28

Beringer, D., y Peterson, J. (1985). Underlying behavioral parameters of the operation of touch-input devices: Biases, models, and feedback. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *27*(4), 445–458. 28

Beringer, J. (1992). Timing accuracy of mouse response registration on the IBM microcomputer family. *Behavior Research Methods*, *24*(3), 486–490. 25

Berners-Lee, T. (2006). *Linked data - design issues.* Descargado el 10/12/2012, de `http://www.w3.org/DesignIssues/LinkedData.html` 4, 5

Berners-Lee, T., y Cailliau, R. (1990). WorldWideWeb: Proposal for a hypertext project. *European Particle Physics Laboratory (CERN)*. 3, 20, 91

Bhalla, M., y Bhalla, A. (2010). Comparative study of various touchscreen technologies. *International Journal of Computer Applications IJCA*, *6*(8), 12–18. 28, 140

Birnbaum, M. (2004). Human research and data collection via the Internet. *Annu. Rev. Psychol.*, *55*, 803–832. 19

Birnbaum, M., y Mellers, B. (1989). Mediated models for the analysis of confounded variables and self-selected samples. *Journal of Educational and Behavioral Statistics*, *14*(2), 146–158. 19

Block, R., y Zakay, D. (1997). Prospective and retrospective duration judgments: A meta-analytic review. *Psychonomic Bulletin & Review*, *4*(2), 184–197. 17

Bovet, D., y Cesati, M. (2005). *Understanding the linux kernel*. O'Reilly Media, Incorporated. 48, 141

Brainard, D. (1997). The psychophysics toolbox. *Spatial vision*, *10*(4), 433–436. 54, 58, 68

Brandt, E., y Dannenberg, R. (1998). Low-latency music software using off-the-shelf operating systems. En (pp. 137–140). International Computer Music Association, Ann Arbor, MI, 1998. ICMA. 37

Buehner, M. (2012). Understanding the past, predicting the future causation, not intentional action, is the root of temporal binding. *Psychological Science*. 152

Buhrmester, M., Kwang, T., y Gosling, S. (2011). Amazon's Mechanical Turk a

new source of inexpensive, yet high-quality, data? *Perspectives on Psychological Science*, *6*(1), 3–5. 5

Carbonell, J., Elkind, J., y Nickerson, R. (1968). On the psychological importance of time in a time sharing system. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *10*(2), 135–142. 13, 18

Castellano, J. (1992). *Handbook of display technology*. Academic Pr. 29

Chappell, D. (1996). *Understanding ActiveX and OLE: a guide for developers and managers*. Microsoft Press. 64

Chapweske, A. (2003a). *The PS/2 mouse interface.* Descargado el 10/12/2012, de `http://www.computer-engineering.org/ps2mouse/` 26

Chapweske, A. (2003b). *The PS/2 mouse/keyboard protocol.* Descargado el 10/12/2012, de `http://http://www.computer-engineering.org/ps2keyboard/` 22, 139

Chuan, S. (2012). *JavaScript patterns collection.* Descargado el 10/12/2012, de `http://shichuan.github.com/javascript-patterns/` 182

Chung, S., y Pereira, A. (2005). Timed Petri net representation of SMIL. *Multimedia, IEEE*, *12*(1), 64–72. 63

Cohen, J., MacWhinney, B., Flatt, M., y Provost, J. (1993). PsyScope: An interactive graphic system for designing and controlling experiments in the psychology laboratory using Macintosh computers. *Behavior Research Methods*, *25*(2), 257–271. 54, 58

CompuServe. (1990). *Graphics Interchange Format programming reference.* Descargado el 10/12/2012, de `http://www.w3.org/Graphics/GIF/spec-gif89a.txt` 63, 92

Cowan, W. (1995). Displays for vision research. *Handbook of optics*, *1*, 27–21. 29

Cowan, W., y Rowell, N. (1986). On the gun independence and phosphor constancy of colour video monitors. *Color Res. Appl., suppl.*, *11*, S34–S38. 32

Crockford, D. (2008). *JavaScript: the good parts*. O'Reilly Media, Incorporated. 179, 183

Crosbie, J. (1990). The Microsoft mouse as a multipurpose response device for the IBM PC/XT/AT. *Behavior Research Methods*, *22*(3), 305–316. 25, 140

Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, ... Watt, J. (2011). *Scalable Vector graphics (SVG) 1.1 (Second Edition), W3C Recommendation.* Descargado el 10/12/2012, de `http://www.w3.org/`

TR/SVG11/ 105, 106

Damian, M. (2010). Does variability in human performance outweigh imprecision in response devices such as computer keyboards? *Behavior research methods*, *42*(1), 205–211. 24, 70, 140

Danzig, P., y Melvin, S. (1990). High resolution timing with low resolution clocks and microsecond resolution timer for sun workstations. *ACM SIGOPS Operating Systems Review*, *24*(1), 23–26. 53

DiNucci, D. (1999). Fragmented future. *Print*, *53*(4), 32. 3

Downs, J., Holbrook, M., Sheng, S., y Cranor, L. (2010). Are your participants gaming the system?: screening mechanical turk workers. En *Proceedings of the 28th international conference on human factors in computing systems* (pp. 2399–2402). 5

Dufau, S., Duñabeitia, J., Moret-Tatay, C., McGonigal, A., Peeters, D., Alario, F., ... others (2011). Smart phone, smart science: How the use of smartphones can revolutionize research in cognitive science. *PloS one*, *6*(9), e24974. 29

Earls, J. (2009). *Real Time Clock.* Descargado el 10/12/2012, de `http://wiki.osdev.org/RTC` 39

Ecma. (2011). *Standard ECMA-262. ECMAScript language specification. edition 5.1 (June 2011)* (Inf. Téc.). Ecma International. 65, 67, 177, 178

Eichstaedt, J. (2001). An inaccurate-timing filter for reaction time measurement by java applets implementing internet-based experiments. *Behavior Research Methods*, *33*(2), 179–186. 66, 138

Elze, T. (2010). Achieving precise display timing in visual neuroscience experiments. *Journal of neuroscience methods*, *191*(2), 171–179. 34

Elze, T., y Tanner, T. (2009). Liquid crystal display response time estimation for medical applications. *Medical physics*, *36*, 4984. 34

Elze, T., Tanner, T., Lochmann, T., y Becker, M. (2007). Lcd monitors in vision science. *Journal of Vision*, *7*(15), 61–61. 15, 35

Engbert, K., Wohlschläger, A., y Haggard, P. (2008). Who is causing what? the sense of agency is relational and efferent-triggered. *Cognition*, *107*(2), 693–704. 152, 167

Erl, T. (2004). *Service-oriented architecture: a field guide to integrating XML and web services*. Prentice Hall PTR. 2

Eysenbach, G. (2011). Can tweets predict citations? metrics of social impact

based on Twitter and correlation with traditional metrics of scientific impact. *Journal of Medical Internet Research*, *13*(4). 4

Fazio, R. H., Sanbonmatsu, D. M., Powell, M. C., y Kardes, F. R. (1986). On the automatic activation of attitudes. *Journal of personality and social psychology*, *50*, 229–238. 68

Fielding, R. (2000). *Architectural styles and the design of network-based software architectures*. Tesis Doctoral no publicada, University of California. 2

Finney, S. (2001). Real-time data collection in linux: A case study. *Behavior Research Methods*, *33*(2), 167–173. 53

Fitts, P. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, *47*(6), 381. 28

Forlines, C., Wigdor, D., Shen, C., y Balakrishnan, R. (2007). Direct-touch vs. mouse input for tabletop displays. En *Proceedings of the sigchi conference on human factors in computing systems* (pp. 647–656). 28

Forster, J. (2007). *DMDX updates page, FAQ*. Descargado el 10/12/2012, de `http://www.u.arizona.edu/~jforster/dmdx.htm` 24

Forster, K., y Forster, J. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods*, *35*(1), 116–124. 7, 54, 55, 56, 61, 70, 78

Forum, U. I. (2001). Device class definition for Human Interface Devices (HID), firmware specification. *Version*, *1*, 27. 22

Fraser, S., Jackson, D., Hyatt, D., Marrin, C., O'Connor, E., Schulze, D., y Gregor, A. (2012). *CSS Transforms, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/css3-transforms/` 105, 107

FSF. (2007). *GNU General Public License, version 3.* Descargado el 10/12/2012, de `http://www.gnu.org/licenses/gpl.html` 225

Gal, A., Eich, B., Shaver, M., Anderson, D., Mandelin, D., Haghighat, M., ... others (2009). Trace-based just-in-time type specialization for dynamic languages. En *Acm sigplan notices* (Vol. 44, pp. 465–478). 65

Galletta, D., Henry, R., McCoy, S., y Polak, P. (2004). Web site delays: How tolerant are users? *Journal of the Association for Information Systems*, *5*(1), 1–28. 18, 19

Garaizar, P. (2012). *Allanjs: Frequency stability analysis library for Java-*

Script. Descargado el 10/12/2012, de `http://txipi.github.com/AllanJS/` 188, 198

Garaizar, P., Vadillo, M., y López-de-Ipiña, D. (2012). Benefits and pitfalls of using HTML5 APIs for online experiments and simulations. *Journal of Online Engineering (iJOE)*, *8*(1), 20–25. 183

Geller, A., Schleifer, I., Sederberg, P., Jacobs, J., y Kahana, M. (2007). PyEPL: A cross-platform experiment-programming library. *Behavior research methods*, *39*(4), 950–958. 54, 58

Gofen, A., y Mackeben, M. (1997). An introduction to accurate display timing for PCs under "Windows". *Spatial vision*, *10*(4), 361–368. 32

Gorn, G., Chattopadhyay, A., Sengupta, J., y Tripathi, S. (2004). Waiting for the web: how screen color affects time perception. *Journal of Marketing Research*, 215–225. 14, 18

Gosling, J., y McGilton, H. (1995). *The java language environment* (Vol. 2550). Sun Microsystems Computer Company. 92

Gould, J., Greene, S., Boies, S., Meluson, A., y Rasamny, M. (1990). Using a touchscreen for simple tasks. *Interacting with Computers*, *2*(1), 59–74. 28

Graves, R., y Bradley, R. (1987). Millisecond interval timer and auditory reaction time programs for the IBM PC. *Behavior Research Methods*, *19*(1), 30–35. 23

Greenwald, A., McGhee, D., y Schwartz, J. (1998). Measuring individual differences in implicit cognition: the implicit association test. *Journal of personality and social psychology*, *74*(6), 1464. 68

Haggard, P., Clark, S., Kalogeras, J., y cols. (2002). Voluntary action and conscious awareness. *Nature neuroscience*, *5*(4), 382–385. 152, 153

Hall, A., Cunningham, J., Roache, R., y Cox, J. (1988). Factors affecting performance using touch-entry systems: Tactual recognition fields and system accuracy. *Journal of applied psychology*, *73*(4), 711. 28

Hand, E., y cols. (2010). Citizen science: People power. *Nature*, *466*(7307), 685. 5

Häusler, J., Sommer, M., y Chroust, S. (2007). Optimizing technical precision of measurement in computerized psychological assessment on windows platforms. *Psychology Science*, *49*(2), 116–131. 137

Hewlett-Packard, Intel, Microsoft, y Phoenix Technologies. (2004). *Advanced*

configuration and power interface specification (Vol. 3; Inf. Téc.). 40

Hickson, I. (2012a). *HTML5. a vocabulary and associated APIs for HTML and XHTML.* Descargado el 10/12/2012, de `http://dev.w3.org/html5/spec/` 2, 105, 122, 177

Hickson, I. (2012b). *HTML canvas 2d context, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/2dcontext/` 109

Hickson, I. (2012c). *WHATWG HTML living standard, 9 Web Workers.* Descargado el 10/12/2012, de `http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html` 108, 176, 185

Hinchcliffe, D. (2008). What is WOA? it's the future of Service-Oriented Architecture (SOA). *Dion Hinchcliffe's Blog–Musings and Ruminations on Building Great Systems*. 2

Hoschka, P. (1998). CSCW research at GMD-FIT: From basic groupware to the social web. *ACM SIGGROUP Bulletin*, *19*(2), 5–9. 3

Hoxmeier, J., y DiCesare, C. (2000). System response time and user satisfaction: An experimental study of browser-based applications. En *Proceedings of the association of information systems americas conference* (pp. 140–145). 19

Humphrey, D., Brook, C., MacDonald, D. Y., A. and, Marxer, R., y Cliffe, C. (2012). *Audio Data API.* Descargado el 10/12/2012, de `https://wiki.mozilla.org/Audio_Data_API` 122, 123

Intel. (1998). *Using the RDTSC instruction for performance monitoring* (Inf. Téc.). Descargado el 01/01/2000, de `http://developer.intel.com/drg/pentiumII/appnotes/RDTSCPM1.HTM` 41

Jackson, D., Hyatt, D., Marrin, C., y Baron, D. (2012). *CSS Transitions, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/css3-transitions/` 105, 107

Jackson, D., Hyatt, D., Marrin, C., Galineau, S., y Baron, D. (2012a). *CSS Animations, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/css3-animations/` 105, 106, 107

Jackson, D., Hyatt, D., Marrin, C., Galineau, S., y Baron, D. (2012b). *CSS animations. W3C Working Draft, 3 April 2012.* Descargado el 10/12/2012, de `http://www.w3.org/TR/2012/WD-css3-animations-20120403/` 64

Jansen, B., Zhang, M., Sobel, K., y Chowdury, A. (2009). Twitter power: Tweets

as electronic word of mouth. *Journal of the American society for information science and technology*, *60*(11), 2169–2188. 4

Johnson, E. (1965). Touch display¿ a novel input/output device for computers. *Electronics Letters*, *1*(8), 219–220. 27

Johnson, E. (1967). Touch displays: A programmed man-machine interface. *Ergonomics*, *10*(2), 271–277. 27

Jones, T. (2010). *Kernel APIs, part 3: Timers and lists in the 2.6 kernel. efficient processing with work-deferral APIs.* Descargado el 10/12/2012, de `http://www.ibm.com/developerworks/linux/library/l-timers-list/` 50

Keller, F., Gunasekharan, S., Mayo, N., y Corley, M. (2009). Timing accuracy of web experiments: A case study using the webexp software package. *Behavior Research Methods*, *41*(1), 1–12. 66, 79, 136

Khatib, F., Cooper, S., Tyka, M., Xu, K., Makedon, I., Popović, Z., ... Players, F. (2011). Algorithm discovery by protein folding game players. *Proceedings of the National Academy of Sciences*, *108*(47), 18949–18953. 5

King, P., Schmitz, P., y Thompson, S. (2004). Behavioral reactivity and real time programming in XML: functional programming meets SMIL animation. En *Proceedings of the 2004 acm symposium on document engineering* (pp. 57–66). 63

Krantz, J. (2001). Stimulus delivery on the Web: What can be presented when calibration isn't possible. *Dimensions of Internet science*, 113–130. 19, 33

Krantz, J., Ballard, J., y Scher, J. (1997). Comparing the results of laboratory and World-Wide Web samples on the determinants of female attractiveness. *Behavior Research Methods*, *29*(2), 264–269. 33

Krishnan, K. (2008). *Cpu power utilization on intel® architectures.* Descargado el 10/12/2012, de `http://software.intel.com/en-us/articles/cpu-power-utilization-on-intel-architectures/` 51, 177, 191

Kuperberg, M., Krogmann, M., y Reussner, R. (2009). Timermeter: Quantifying properties of software timers for system analysis. En *Quantitative evaluation of systems, 2009. qest'09. sixth international conference on the* (pp. 85–94). 52, 190

Lago, N., y Kon, F. (2004). The quest for low latency. En *Proceedings of the*

international computer music conference (pp. 33–36). 37

Lagroix, H., Yanko, M., y Spalek, T. (2012). LCDs are better: Psychophysical and photometric estimates of the temporal characteristics of CRT and LCD monitors. *Attention, Perception, & Psychophysics*, 1–9. 34

Lambert, J., y Power, J. (2008). Platform independent timing of java virtual machine bytecode instructions. *Electronic Notes in Theoretical Computer Science*, *220*(3), 97–113. 53

Lee, A. (2009). *Win32 timer queues are not suitable for high-performance timing.* Descargado el 10/12/2012, de `http://www.virtualdub.org/blog/pivot/entry.php?id=272` 46

Le Hégaret, P., Whitmer, R., y Wood, L. (2009). *Document Object Model (DOM).* Descargado el 10/12/2012, de `http://www.w3.org/DOM/` 105

Liang, H., y Badano, A. (2007). Temporal response of medical liquid crystal displays. *Medical physics*, *34*, 639. 33

Libet, B. (1965). Cortical activation in conscious and unconscious experience. *Perspectives in biology and medicine*, *9*(1), 77. 151

Libet, B. (1966). Brain stimulation and the threshold of conscious experience. *Brain and conscious experience*, 165–181. 151

Libet, B. (1978). Neuronal vs. subjective timing for a conscious sensory experience. *Cerebral correlates of conscious experience*, 69–82. 151

Libet, B. (2004). *Mind time: The temporal factor in consciousness*. Harvard university press. 152

Libet, B., Gleason, C., Wright, E., y Pearl, D. (1983). Time of conscious intention to act in relation to onset of cerebral activity (readiness-potential) the unconscious initiation of a freely voluntary act. *Brain*, *106*(3), 623–642. 7, 68, 151, 152, 153

Libet, B., Wright, E., Feinstein, B., y Pearl, D. (1979). Subjective referral of the timing for a conscious sensory experience. *Brain*, *102*(1), 193–224. 151

Lilja, D. (2000). *Measuring computer performance: a practitioner's guide*. Cambridge University Press. 188, 201

Lincoln, C., y Lane, D. (1980). Reaction time measurement errors resulting from the use of CRT displays. *Behavior Research Methods*, *12*(1), 55–57. 31

Lockhart, J. (1967). Ambient temperature and time estimation. *Journal of Experimental Psychology; Journal of Experimental Psychology*, *73*(2), 286. 13,

18

MacInnes, W., y Taylor, T. (2001). Millisecond timing on PCs and Macs. *Behavior Research Methods*, *33*(2), 174–178. 7, 58

Mangan, M., y Reips, U. (2007). Sleep, sex, and the Web: Surveying the difficult-to-reach clinical population suffering from sexsomnia. *Behavior Research Methods*, *39*(2), 233–236. 5

Mann, J. (2012). *High Resolution Time, W3C Proposed Recommendation.* Descargado el 10/12/2012, de `http://www.w3.org/TR/hr-time/` 147, 187, 230

Mann, J., y Jain, A. (2012). *Page Visibility, W3C Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/PageVisibility/Overview.html` 187

Mann, J., y Wang, Z. (2012). *Performance Timeline, W3C Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/PerformanceTimeline/Overview.html` 187

Mann, J., Wang, Z., y Quach, A. (2012a). *Resource Timing, W3C Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/ResourceTiming/Overview.html` 187

Mann, J., Wang, Z., y Quach, A. (2012b). *User Timing, W3C Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/UserTiming/Overview.html` 187

Mann, J., y Weber, J. (2011). *Efficient Script Yielding, W3C Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/setImmediate/Overview.html` 109, 184, 187

Mapou, R. (1982). Tachistoscopic timing on the TRS-80. *Behavior Research Methods*, *14*(6), 534–538. 7

Marrin, C. (2011). *WebGL Specification, version 1.0.* Descargado el 10/12/2012, de `https://www.khronos.org/registry/webgl/specs/1.0/` 109

Mathôt, S., Schreij, D., y Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior research methods*, 1–11. 54, 58, 59

McGraw, K., Tew, M., y Williams, J. (2000). The integrity of web-delivered experiments: Can you trust the data? *Psychological Science*, *11*(6), 502–506. 66

Mckinney, C., MacCormac, E., y Welsh-Bohmer, K. (1999). Hardware and software for tachistoscopy: How to make accurate measurements on any PC utilizing the Microsoft Windows operating system. *Behavior Research Methods*, *31*(1), 129–136. 7

Miller, G. (2012). The smartphone psychology manifesto. *Perspectives on Psychological Science*, *7*(3), 221–237. 221

Miller, R. (1968). Response time in man-computer conversational transactions. En *Proceedings of the december 9-11, 1968, fall joint computer conference, part i* (pp. 267–277). 18

Mollon, J., y Baker, M. (1995). The use of CRT displays in research on colour vision. *Documenta Ophthalmologica Proceedings Series*, *57*, 423–444. 32

Molnar, I. (2005). *kernel/timer.c design.* Descargado el 10/12/2012, de `http://lwn.net/Articles/156329/` 48

Moore, J., Lagnado, D., Deal, D., y Haggard, P. (2009). Feelings of control: Contingency determines experience of action. *Cognition*, *110*(2), 279–283. 164

Moore, J., y Obhi, S. (2012). Intentional binding and the sense of agency: a review. *Consciousness and cognition*. 7

Mueller, S. (2010). *The PEBL manual.* Descargado el 10/12/2012, de `http://pebl.sourceforge.net/` 54, 58

Munshi, A., y Leech, J. (2010). *OpenGL® ES Common Profile Specification Version 2.0.25.* Descargado el 10/12/2012, de `http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf` 109

Nah, F. (2004). A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, *23*(3), 153–163. 18

Neath, I., Earle, A., Hallett, D., y Surprenant, A. (2011). Response time accuracy in Apple Macintosh computers. *Behavior research methods*, *43*(2), 353–362. 24, 67, 137

Neely, J. (1977). Semantic priming and retrieval from lexical memory: Roles of inhibitionless spreading activation and limited-capacity attention. *Journal of Experimental Psychology: General*, *106*(3), 226. 68

Nielsen, J. (1994). Response times: the three important limits. *Usability Engineering*. 18

North, D. (2006). Introducing bdd. *Better Software, March*. 199

Nosek, B. (2005). Moderators of the relationship between implicit and explicit evaluation. *Journal of Experimental Psychology: General*, *134*(4), 565. 4, 136

Nosek, B., y Banaji, M. (2001). The go/no-go association task. *Social cognition*, *19*(6), 625–666. 68

Odvarko, J., y Jain, A. (2012). *HTTP Archive (HAR) format, Editor's Draft.* Descargado el 10/12/2012, de `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/HAR/Overview.html` 187

Oeschger, I. (2002). *API reference: Netscape Gecko plugins.* Netscape Communications. 64

Orduña, P., García-Zubia, J., Irurzun, J., López-de-Ipiña, D., y Rodríguez-Gil, L. (2011). Enabling mobile access to remote laboratories. En *Global engineering education conference (educon), 2011 ieee* (pp. 312–318). 29

OReilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & strategies*(1), 17. 3

Orr, N., y Hopkin, V. (1968). *The role of the touch display in air traffic control* (Inf. Téc.). DTIC Document. 27

Osmani, A. (2012). *Learning javascript design patterns*. O'Reilly Media. 182

Osterman, L. (2005). *APIs you never heard of - the Timer APIs.* Descargado el 10/12/2012, de `http://blogs.msdn.com/b/larryosterman/archive/2005/09/08/462477.aspx` 44, 46

Pang, B., y Lee, L. (2008). *Opinion mining and sentiment analysis*. Now Pub. 4

Paolacci, G., Chandler, J., y Ipeirotis, P. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision Making*, *5*(5), 411–419. 5

Pedersen, T., Pakhomov, S., Patwardhan, S., y Chute, C. (2007). Measures of semantic similarity and relatedness in the biomedical domain. *Journal of biomedical informatics*, *40*(3), 288–299. 5

Peirce, J. (2007). PsychoPy—psychophysics software in Python. *Journal of neuroscience methods*, *162*(1), 8–13. 54, 58, 78

Pierce, J. (1999). Hearing in time and space. En *Music, cognition, and computerized sound* (pp. 89–103). 36

Pixley, T. (2000). *Document Object Model events.* Descargado el 10/12/2012, de `http://www.w3.org/TR/DOM-Level-2-Events/events.html` 141, 186

Plant, R., Hammond, N., y Turner, G. (2004). Self-validating presentation and response timing in cognitive paradigms: How and why? *Behavior Research Methods*, *36*(2), 291–303. 14, 58, 74, 75, 77, 137, 210

Plant, R., Hammond, N., y Whitehouse, T. (2002). Toward an experimental timing standards lab: Benchmarking precision in the real world. *Behavior Research Methods*, *34*(2), 218–226. 58

Plant, R., Hammond, N., y Whitehouse, T. (2003). How choice of mouse may affect response timing in psychological studies. *Behavior Research Methods*, *35*(2), 276–284. 26, 140

Plant, R., y Turner, G. (2009). Millisecond precision psychological research in a world of commodity computers: New hardware, new problems? *Behavior research methods*, *41*(3), 598–614. 15, 24, 26, 33, 37, 137, 140

Ragget, D. (1997). *HTML 3.2 reference specification, W3C recommendation.* Descargado el 10/12/2012, de `http://www.w3.org/TR/REC-html32` 104

Ramsay, J., Barbesi, A., y Preece, J. (1998). A psychological investigation of long retrieval times on the World Wide Web. *Interacting with computers*, *10*(1), 77–86. 19

Randers-Pehrson, G. (2001). *MNG (Multiple-image Network Graphics) format version 1.0.* Descargado el 10/12/2012, de `http://www.libpng.org/pub/mng/spec/` 92

Rasch, R. (1979). Synchronization in performed ensemble music. *Acustica*, *43*, 121–131. 36

Ratanaworabhan, P., Livshits, B., y Zorn, B. (2010). JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. En *Proceedings of the 2010 usenix conference on web application development* (pp. 3–3). 65

Ratcliff, R., y McKoon, G. (1978). Priming in item recognition: Evidence for the propositional structure of sentences. *Journal of Verbal Learning and Verbal Behavior*, *17*(4), 403–417. 68

Reimers, S., y Stewart, N. (2007). Adobe flash as a medium for online experimentation: A test of reaction time measurement capabilities. *Behavior Research*

Methods, *39*(3), 365–370. 67, 79, 136

Reimers, S., y Stewart, N. (2008). Using adobe flash lite on mobile phones for psychological research: Reaction time measurement reliability and interdevice variability. *Behavior research methods*, *40*(4), 1170–1176. 67, 221

Reips, U. (2000). The Web experiment method: Advantages, disadvantages, and solutions. *Psychological experiments on the Internet*, 89–117. 19

Reips, U. (2002). Standards for Internet-based experimenting. *Experimental Psychology (formerly Zeitschrift für Experimentelle Psychologie)*, *49*(4), 243–256. 19

Reips, U., y Garaizar, P. (2011). Mining twitter: A source for psychological wisdom of the crowds. *Behavior Research Methods*, *43*(3), 635–642. 4

Resig, J. (2008). *How JavaScript timers work.* Descargado el 10/12/2012, de `http://ejohn.org/blog/how-javascript-timers-work/` 108, 150

Riemersma, T. (1994). *Periodic interrupts with the Real Time Clock.* Descargado el 10/12/2012, de `http://www.compuphase.com/int70.txt` 39

Riley, W., y Howe, D. (2008). *Handbook of frequency stability analysis*. US Department of Commerce, National Institute of Standards and Technology. 196, 197

Robinson, J., y McCormack, C. (2012). *Timing control for script-based animations, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/animation-timing/` 109, 148, 187

Rogers, C. (2012). *Web Audio API, W3C Working Draft.* Descargado el 10/12/2012, de `http://www.w3.org/TR/webaudio/` 122, 123

Russinovich, M., Solomon, D., y Ionescu, A. (2009). *Windows® internals*. Microsoft Press. 51, 177, 191

Sala, R. (2008). *RS Timer 2.01.0.0002.* Descargado el 10/12/2012, de `http://www.easycode.cat/Spanish/ActiveX.htm` 158

Schepers, D. (2009). *Webmandering. reinventing fire.* Descargado el 10/12/2012, de `http://schepers.cc/webmandering` 174

Schiff, W., y Thayer, S. (1968). Cognitive and affective factors in temporal experience: Anticipated or experiencedpleasant and unpleasant sensory events. *Perceptual and Motor Skills*, *26*(3), 799–808. 13, 18

Schmidt, W. (2001). Presentation accuracy of Web animation methods. *Behavior*

Research Methods, *33*(2), 187–200. 63, 65, 66, 67, 74, 79, 95

Schmitz, P., y Cohen, A. (2001). *SMIL Animation, W3C Recommendation.* Descargado el 10/12/2012, de `http://www.w3.org/TR/2001/REC-smil-animation-20010904/` 106

Schneider, W. (1988). Micro experimental laboratory: An integrated system for ibm pc compatibles. *Behavior Research Methods*, *20*(2), 206–217. 54

Schneider, W., Eschman, A., y Zuccolotto, A. (2002). *E-prime: User's guide*. Psychology Software Incorporated. 54, 55, 59, 70, 78, 80, 81, 137

Sears, A. (1991). Improving touchscreen keyboards: Design issues and a comparison with other devices. *Interacting with computers*, *3*(3), 253–269. 28

Sears, A., y Shneiderman, B. (1991). High precision touchscreens: design strategies and comparisons with a mouse. *International Journal of Man-Machine Studies*, *34*(4), 593–613. 28

Segalowitz, S. (1987). IBM PC tachistoscope: Text stimuli. *Behavior Research Methods*, *19*(4), 383–388. 7

Segalowitz, S., y Graves, R. (1990). Suitability of the IBM XT, AT, and PS/2 keyboard, mouse, and game port as response devices in reaction time paradigms. *Behavior Research Methods*, *22*(3), 283–289. 23, 25, 140

Selvidge, P. (1999). How long is too long to wait for a website to load. *Usability news*, *1*(2). 19

Shimizu, H. (2002). Measuring keyboard response delays by comparing keyboard and joystick inputs. *Behavior Research Methods*, *34*(2), 250–256. 23

Shneiderman, B. (1980). *Software psychology: Human factors in computer and information systems (winthrop computer systems series).* Winthrop Publishers. 13, 18

Shneiderman, B. (1984). Response time and display rate in human performance with computers. *ACM Computing Surveys (CSUR)*, *16*(3), 265–285. 18

Shneiderman, B. (2008). Copernican challenges face those who suggest that collaboration, not computation are the driving energy for socio-technical systems that characterize Web 2.0. *Science*, *319*, 1349–1350. 4

Simon, L., Simpson, K., Dalton, J.-D., y Joel, C. (2012). JavaScript performance mythbusters™ (via jsperf). En *Sxsw 2012, austin, tx, usa.* 183

Sivonen, H. (2009). *Browser technology stack.* Descargado el 10/12/2012, de `http://hsivonen.iki.fi/web-stack/` 174, 175

Snow, R., O'Connor, B., Jurafsky, D., y Ng, A. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. En *Proceedings of the conference on empirical methods in natural language processing* (pp. 254–263). 5

Sondaar, M. (2007). *Advanced Programmable Interrupt Controller.* Descargado el 10/12/2012, de `http://wiki.osdev.org/APIC` 40

Souders, S. (2008). High-performance web sites. *Communications of the ACM*, *51*(12), 36–41. 179, 180

Souders, S. (2009). *Even faster web sites*. O'Reilly. 179

Sprouse, J. (2011). A validation of Amazon Mechanical Turk for the collection of acceptability judgments in linguistic theory. *Behavior Research Methods*, *43*(1), 155–167. 5

Stahl, C. (2006). Software for generating psychological experiments. *Experimental Psychology*, *53*(3), 218. 7, 54, 57

Stefanov, S. (2010). *Javascript patterns*. O'Reilly Media, Incorporated. 182

Stevens, M., Lammertyn, J., Verbruggen, F., y Vandierendonck, A. (2006). Tscope: AC library for programming cognitive experiments on the MS Windows platform. *Behavior Research Methods*, *38*(2), 280–286. 54, 58

Stewart, N. (2006). Millisecond accuracy video display using OpenGL under Linux. *Behavior research methods*, *38*(1), 142–145. 7, 67

Stoet, G. (2010). PsyToolkit: A software package for programming psychological experiments using Linux. *Behavior research methods*, *42*(4), 1096–1104. 54, 58

Straw, A. (2008). Vision egg: an open-source library for realtime visual stimulus generation. *Frontiers in neuroinformatics*, *2*. 54, 58

Sutter, E., y Tran, D. (1992). The field topography of ERG components in man—I. the photopic luminance response. *Vision research*, *32*(3), 433–446. 32

Teo, H., Oh, L., Liu, C., y Wei, K. (2003). An empirical study of the effects of interactivity on web user attitude. *International Journal of Human-Computer Studies*, *58*(3), 281–305. 18

Thom, B., y Nelson, M. (2004). An in-depth analysis of real-time MIDI performance. En *Proc. 2004 intl. computer music conf.(icmc-04).* 37

Tiobe. (2012). *Most popular programming languages.* Descargado el 10/12/2012, de `http://www.tiobe.com/index.php/content/`

paperinfo/tpci/index.html 58, 93

Treisman, M. (1963). Temporal discrimination and the indifference interval: Implications for a model of the "internal clock". *Psychological Monographs: General and Applied*, *77*(13), 1. 13

Ulrich, R., y Giray, M. (1989). Time resolution of clocks: Effects on reaction time measurement—good news for bad clocks. *British Journal of Mathematical and Statistical Psychology*, *42*, 1–12. 25, 70

Van Baak, T. (2009). *A short lesson in Allan Deviation (tides, part 2).* Descargado el 10/12/2012, de http://www.leapsecond.com/hsn2006/pendulum-tides-ch2.pdf 196

Vernberg, D., Snyder, C., y Schuh, M. (2005). Preliminary validation of a hope scale for a rare health condition using web-based methodology. *Cognition & Emotion*, *19*(4), 601–610. 5

W3Techs. (2012). *Usage of client-side programming languages for websites.* Descargado el 10/12/2012, de http://w3techs.com/technologies/overview/client_side_language/all 94, 95

Walling, B. (2012). *Programmable Interval Timer.* Descargado el 10/12/2012, de http://wiki.osdev.org/PIT 38

Wang, P., y Nikolić, D. (2011). An LCD monitor with sufficiently precise timing for research in vision. *Frontiers in human neuroscience*, *5*. 34

Wang, Y., Stables, R., y Reiss, J. (2012). Audio latency measurement for desktop operating systems with onboard soundcards. En *128th convention of the audio engineering society* (Vol. 8081). Audio Engineering Society. 35, 37, 122

Wang, Z. (2012). *Navigation Timing, W3C Proposed Recommendation.* Descargado el 10/12/2012, de http://www.w3.org/TR/navigation-timing/ 187

Ward, B. (2012). Pragmatic performance with third-party JavaScript. En *Open web camp, july 14th, 2012, san jose, ca, usa.* 181

Wilson, M. (2003). *Win32 performance measurement options.* Descargado el 10/12/2012, de http://www.drdobbs.com/windows/win32-performance-measurement-options/184416651 52

Winokur, D. (2011). *Flash to focus on PC browsing and mobile apps; adobe to more aggressively contribute to HTML5.* Descarga-

do el 10/12/2012, de `http://blogs.adobe.com/conversations/2011/11/flash-focus.html` 94

Wundt, W. (1902). *Philosophische studien* (Vol. 19). W. Engelmann. 151

Yee, B., Sehr, D., Dardyk, G., Chen, J., Muth, R., Ormandy, T., ... Fullagar, N. (2009). Native client: A sandbox for portable, untrusted x86 native code. En *Security and privacy, 2009 30th ieee symposium on* (pp. 79–93). 64

Zakas, N. (2010). *High performance JavaScript*. O'Reilly Media / Yahoo Press. 177, 179, 181

Zakas, N. (2011). *Timer resolution in browsers.* Descargado el 10/12/2012, de `http://www.nczonline.net/blog/2011/12/14/timer-resolution-in-browsers/` 177

Zakas, N. (2012). Timers, power consumption, and performance. En *Velocity 2012, web performance and operations conference, june 25-27, 2012, santa clara, ca, usa.* 177

Zhang, X., Lu, X., Zhang, Y., y Lian, P. (2009). Research of accuracy and stability on computer timing control in behavioral and cognitive neuroscience. En *Intelligent ubiquitous computing and education, 2009 international symposium on* (pp. 295–299). 46, 52