SOPORTE DE INTERACCIONES MULTIMODALES PARA APLICACIONES WEB

DIEGO I. PAEZ

Tesina de Grado

Facultad de Informática
Universidad Nacional de La Plata

Diego I. Paez: *Soporte de Interacciones Multimodales para Aplicaciones Web*, Tesina de Grado, © Diciembre 2015

DIRIGIDA POR:
Dr. Rossi, Gustavo
Mg. Rodriguez, Andrés

LUGAR:
La Plata, Buenos Aires

Dedicado a mis padres por su apoyo incondicional.

Y a todas aquellas personas,
con las que he tenido la suerte de cruzar caminos.

RESUMEN

En este trabajo de tesis se propone una herramienta que hace posible aprovechar mas de una modalidad dentro del contexto de una aplicación web, de forma que se aumente el conjunto de interacciones, soportando así nuevas interacciones multimodales.

Por interacciones multimodales, se hace referencia a todas aquellas interacciones, tanto para con el mundo físico real, como para el virtual a través de los diferentes *modos*; cada uno de estos modos esta "asociado" a los sentidos del ser humano, de acuerdo a Bourguet [3]. Se busca, de esta forma, eliminar o disminuir barreras en la comunicación hombre-máquina/aplicación, e. g. eliminando o dando una alternativa al uso del teclado, eligiendo favorecer a nuevas capacidades de interacción naturales, i. e. asociados a los sentidos.

Cuando se permite operar con mas de uno de estos *modos* en simultáneo, estamos frente a interacciones multimodales, una de las aplicaciones clásicas que abrió el camino multimodal, es el trabajo de Bolt [2] con *put-that-there*, donde se estudiaba la interacción en una aplicación, de la voz y gestos para indicar comandos.

Las aplicaciones web son crecientemente populares, extendiéndose continuamente a nuevos dominios y plataformas. Un número increíblemente grande de usuarios entran en contacto directos con aplicaciones web diariamente utilizando dispositivos que ofrecen mecanismos de interacción diferentes, los cuales en muchos casos, pueden ser utilizados de manera simultánea.

Teniendo en cuenta este contexto, se ha desarrollado una plataforma, denominada *Plusultra*, junto a un protocolo de comunicación y un cliente modular, *Gyes*, que no solo permite consumir eventos generados por las modalidades si no también definir nuevos drivers para ampliar el catálogo de modalidades soportadas. Todas estas herramientas son *open source* lo que significa que no solo pueden ser mejoradas por la comunidad si no también extendidas en alcance, por ejemplo, como se mencionó a través del desarrollo de nuevos drivers. Así es como se aumentan las capacidades interactivas de las aplicaciones web.

El contenido de este documento está organizado de la siguiente manera; el primer capítulo introduce al lector en el *estado del arte* actual del soporte a interacciones multimodales. En el segundo y tercer capítulo se analiza en dos partes las estrategias elegidas para desarrollar la plataforma *Plusultra* y el módulo cliente *Gyes*. Luego, en el capítulo cuatro, se define que se entiende por aplicación web y como se extienden sus capacidades de interacción usando la plataforma. En el capítulo cinco, se muestra a todo el conjunto de tecnologías

desarrolladas en acción, usando como ejemplo a la aplicación *Shapes*, desarrollada específicamente para este trabajo. Finalmente, el lector encontrará en el capítulo seis, las conclusiones y las posibilidades de extensión, detalladas como *trabajo a futuro*. A su vez se agregan dos apéndices con información en detalle sobre las modalidades elegidas en el trabajo, estas son háptica y gestual.

AGRADECIMIENTOS

A todo el equipo que hace a la Facultad de Informática, UNLP.

En cada esfuerzo, en cada paso, es donde radica el avance hacia un lugar mejor.

ÍNDICE GENERAL

Parte I

INTRODUCCIÓN

Ahora es común estar en contacto con algún dispositivo con alta capacidad de computo, mas aun, estos dispositivos pueden ser móviles (i. e.smartphones, tablets) y contar con una gran variedad de sensores y actuadores. Estos artefactos tienen la capacidad de aumentar las formas de interacción entre el usuario y el software. Por ejemplo, podría accederse a los motores de vibración y utilizarlo para brindar feedback que estimule el sentido del tacto, junto a una señal auditiva, afectando de esta forma, a mas de un sentido.

Otros dispositivos pueden agregar periféricos con hardware y software especifico, como el detector de gestos Leap Motion o incluso algún diseño a medida montado sobre placas similares a Arduino o Rapsberry Pi, sólo por mencionar algunas posibilidades. Quedan abiertos nuevos modos de interacción con el software que incluso pueden ser empleados de manera simultanea, dando lugar a interfaces multimodales.

A continuación se describe el estado del arte actual en aplicaciones multimodales. Específicamente, se distinguirá qué clase de soporte se brinda para el desarrollo de este tipo de aplicaciones, si esta orientado a generar aplicaciones de propósito general o particular y si intenta cubrir todo el espectro de modalidades o se concentra solo en un conjunto limitado de ellas.

1 ESTADO DEL ARTE

A continuación se introducirán las distintas soluciones que conforman el estado del arte actual en cuanto a técnicas que permitan desarrollar aplicaciones con soporte de interacciones multimodales.

Por interacciones multimodales, se hace referencia a todas aquellas interacciones, tanto para con el mundo físico real como para el virtual a través de los diferentes *modos*; cada uno de estos modos esta "asociado" a los sentidos del ser humano, de acuerdo a Bourguet [3]. Se busca, de esta forma, eliminar o disminuir barreras en la comunicación hombre-máquina/aplicación, e. g.eliminando o dando una alternativa al uso del clásico teclado/ratón, eligiendo favorecer a nuevas capacidades de interacción naturales. Cuando se permite operar con mas de uno de estos *modos* en simultaneo, estamos frente a interacciones multimodales.

En este capítulo se analizarán diferentes soluciones, la forma en que se brindan estas soluciones varía según cada trabajo, algunas optan por utilizar un formato tipo framework, es decir como una herramienta de software que puede ser usada para crear otras aplicaciones; otras por una plataforma brindando una solución mas general ó incluso como un servicio web. Para identificar mejor las diferentes soluciones se ha decidido presentar las mismas organizadas de acuerdo al formato (i. e., plataforma, framework, etc.) o al contexto (i. e., educativo, uso general, entre otros). Luego se analizan los problemas comunes que pueden padecer estos trabajos y se compara con las estrategias seleccionadas en la solución propuesta. Al finalizar el capítulo se encuentra un cuadro que resume las distintas publicaciones mencionadas.

1.1 TRABAJOS RELACIONADOS

1.1.1 *Clasificación según el Formato*

En el trabajo de Lo et al. [26], se propone un framework de uso general para añadir nuevas modalidades al ámbito de prácticamente cualquier aplicación. Para lograr esto, el framework propuesto, permite controlar una interfaz de acceso a nuevos dispositivos al sistema, estos nuevos dispositivos son los que añaden las nuevas modalidades. Las pruebas que han realizado incluyen hardware como el Wiimote e iPhone. Las modalidades que han sido integradas al framework son: gestual (utilizando las manos), háptica y reconocimiento de voz. A través de una arquitectura de tres capas, las señales producidas por

los dispositivos son transformadas en comandos que actúan sobre las aplicaciones.

En un principio este framework esta orientado a disminuir las barreras en la inserción de nuevas modalidades en un sistema previamente desarrollado.

La solución que se propone en éste trabajo, encuentra similitudes con el acercamiento de Lo et al. [26], aunque el trabajo aquí propuesto se enfoca directamente en el desarrollo integral de aplicaciones web. Además, si bien existen similitudes arquitectónicas entre las soluciones, como puede ser la separación en capas, dando independencia al ingeniero de modalidad para trabajar con el lenguaje necesario y aun así poder conectarse al framework; en este trabajo se han tomado decisiones novedosas que no se han visto en otros trabajos similares, como la separación y distribución de los componentes de fisión y fusión. En la parte dos se introducen todos los detalles arquitectónicos propios de la plataforma propuesta.

En el caso de Cutugno et al. [7], introducen también un framework que permite añadir interacciones multimodales en dispositivos móviles. La importancia de tener en cuenta estos aparatos se basa en su popularidad y en sus capacidades, muchos cuentan con pantallas sensibles al tacto, cámaras, micrófonos, parlantes y un conjunto de sensores. Es decir, mecanismos que permiten abrir nuevos canales de comunicación con el usuario a través de una herramienta portátil. Otra característica importante, es que en el trabajo de Cutugno et al. [7], siguen los lineamientos de la W3C [37] [39] para producir un framework compatible a estas disposiciones. Las aplicaciones que utilicen este framework podrán recibir notificaciones generadas por un reconocedor de modalidad, es decir existe una vía de comunicación multimodal (entrada). Además, señalan que los desarrolladores pueden indicar a cual evento multimodal suscribirse a través de la generación de un documento XML especifico. Con el uso de este archivo el servidor puede conocer a qué componente de la aplicación cliente alertar cuando ocurre determinado evento multimodal, entre otras funciones; para efectivizar esta comunicación los autores proponen un conjunto de etiquetas basado en SMUIML [10]. Otra característica de este trabajo es el uso de NFAs (*Nondeterministic Finite Automaton*) para generar los distintos escenarios posibles de interacción.

De esta manera el framework genera una aplicación que en el *Interaction Manager* funciona como una maquina de estados.

El framework es validado a través del desarrollo de una aplicación *Android* que hace uso de GMaps para resolver consultas sobre el transporte público que pueden ser emitidas mediante el uso de gestos táctiles y voz.

El uso de NFA's y de un metalenguaje para representar componentes, son características propias de este trabajo que no se encuentran en la solución propuesta. Se considero como añadir excesiva comple-

jidad, introducir una máquina de estados en el núcleo del *Interaction Manager* de forma tan temprana, cuando lo que se estaba desarrollando era un sistema novedoso, aunque no se descarta una estrategia así para futuras versiones mas estables. Respecto a la decisión de no brindar una representación basada en un metalenguaje, la razón se basa en que esta plataforma no se apoya en un servidor, corre íntegramente en clientes, distribuida; utilizando el servicio de mensajería provisto, el cual se encarga de mantener y distinguir a los diferentes clientes.

A diferencia de los trabajos anteriores, Lo, Tang, Ngai, Chan, Leong, and Chan [27] consideran, no solo la construcción de un framework, sino también los aspectos de infraestructura necesarios para interconectar los distintos componentes de la solución, de esta manera se introduce una plataforma para el desarrollo de aplicaciones que soportan interacciones multimodales, la misma es denominada *i*Chameleon*. Proponen un ciclo de desarrollo con una clara división de tareas entre los diferentes responsables de una aplicación multimodal, usando un framework que sigue el patrón MVC.

De esta forma, los ingenieros de dispositivos son los responsables de integrar nuevos artefactos a la plataforma, según Lo et al. esta etapa corresponde al desarrollo de "vistas"; luego los diseñadores de modalidades se encargan de crear la modalidad a utilizar, la misma puede estar compuesta por uno o mas componentes encargados de analizar los datos de la modalidad. Estos componentes son los "modelos" dentro del framework. Luego de que la modalidad ha sido diseñada es momento de implementar el nivel de fusión de la misma. Los desarrolladores son los encargados de esta tarea que se corresponde al componente "controlador" del framework. Finalmente, los diseñadores de interacción son los encargados de generar la aplicación; para conseguir esto *i*Chameleon* brinda una interfaz de alto nivel, lograda gracias a la separación de responsabilidades antes mencionada, de esta forma no es necesario tener conocimientos de programación para llevar a cabo esta etapa (diseño de interacción).

En *i*Chameleon* se destaca claramente la separación de responsabilidades de un equipo de desarrollo multimodal, esto lo consiguen a través del desarrollo y uso de un framework modular. Este aspecto es similar a lo que se propone en el presente trabajo y mas aun, se busco mantener esta separación de tareas y roles.

El ciclo de desarrollo profesional en aplicaciones multimodales es multidisciplinario. El mismo puede estar compuesto por: desarrolladores, ingenieros de modalidad, psicólogos e ingenieros de interacción; entre los actores destacados.

1.1.2 *Clasificación según el Dominio*

Gabbanini et al. [17] introducen un framework donde el foco esta puesto en los entornos inteligentes. En dichos entornos, la presencia de múltiples dispositivos que interactúan de alguna manera con el usuario es algo común, por lo tanto contar con una estrategia para aprovechar estos recursos de forma organizada es importante según

establecen los autores. Una de las características de este trabajo son los principios que establecen para el diseño de interacciones multimodales, se fijan tres, son: (1) no establecer relaciones de dependencia entre modalidades, (2) respetar y explotar las características de cada modalidad y (3) modelar criteriosamente las actividades que podrá desarrollar el usuario. Dentro del framework manipulan una máquina de estados, la misma permite modelar las transiciones de una modalidad. Cada estado representa una forma controlada de respuesta a un evento. Un evento puede ser generado por el uso de una modalidad. Al igual que en el caso de *i*Chameleon* los autores mencionan la necesidad de no perder de vista el rasgo integrador en cuanto a las distintos tipos de profesionales (desde programadores e ingenieros hasta psicólogos y desarrolladores de interacción), que son necesarios para el desarrollo de una correcta aplicación multimodal.

Así, Gabbanini et al. [17] desarrollan una aplicación multimodal particular que hace uso del canal de voz y auditivo.

Un punto de vista diferente que pone las características de una aplicación multimodal como una necesidad es expuesto por da Silva & da Rocha [8]. El contexto de las aplicaciones analizadas es el ámbito educativo. Particularmente aplicaciones web como *moodle*, *sakai* o la brasileña *teleduc* fueron tenidas en cuenta en el trabajo. Los autores plantean como hipótesis que añadir interacciones multimodales a estas aplicaciones mejorará no sólo la usabilidad sino también la accesibilidad de las mismas.

Para probarlo construyen un ambiente web educativo, extendiendo *Ae e-Learning environment*, para que tenga soporte a interacciones multimodales; a través del uso de este framework los autores esperan obtener mediciones concretas que validen su teoría. Utilizan las modalidades gestuales y hápticas (táctil) como canales de entrada al sistema que hacen juego con determinados componentes del entorno educativo como son el *Weblog* y el *Whiteboard*.

Finalmente, cabe destacar el enfoque transversal que hacen las técnicas multi-dispositivo, en particular el desarrollo de interfaces de usuario distribuidas. Si bien esta temática no tiene como objetivo el soporte a múltiples modalidades, el efecto de expandir el contenido a otros dispositivos tales como tablets o smartphones hace posible que dichos contenidos puedan ser accedidos a través de otras modalidades, como pueden ser gestos táctiles. Por lo tanto es interesante tener en cuenta el desafío técnico que se realiza en esta área. En particular se analiza el trabajo de Frosini et al. [16], donde exponen tanto un framework como aspectos detallados de la arquitectura de la solución.

Uno de los puntos destacados es el estilo distribuido de la solución, el framework se divide en dos partes, una de las cuales reside en los dispositivos clientes (*client side*); la otra puede estar en un servidor (*engine side*). De esta forma, los distintos dispositivos, que pueden contar con capacidades diferentes de interacción, pueden procesar

no solo la lógica de la aplicación sino también señales del framework (*engine side*). Así pueden distinguirse diferentes responsabilidades, los dispositivos *client side*, pueden decidir como actuar y qué eventos o notificaciones "escuchar" del *engine side*; también se destaca la necesidad de un sistema de mensajería entre los distintos componentes del sistema como un aspecto importante. Estas características arquitectónicas encuentran similitudes con las de la solución aquí presentada, en aspectos tales como, distribuir parte de la lógica del framework en los clientes o utilizar un sistema de mensajería y señalización para comunicarse con los clientes.

1.2 ASPECTOS DE DISEÑO

Continuando con la clasificación propuesta, se analizarán primero los desafíos que deben superar las soluciones desde el punto de vista del formato. Luego se considerarán los trabajos de acuerdo a la forma de instanciar y utilizar el framework en la práctica.

Desde el punto de vista del formato, un requisito importante es la modularidad del sistema. Esto converge de diferentes fuentes, como Dumas et al. [12] o el grupo de trabajo de la W3C respecto a arquitecturas multimodales [39], donde se distinguen al menos cuatro módulos que permiten una clara separación de responsabilidades, estos son:

- Componentes de modalidad de entrada y salida (*Input & Output modalities*),también conocidos como reconocedores y sintetizadores de modalidad, a veces pueden ser un único componente.
- Comité de integración (*Integration Committee*), el encargado de mantener la lógica de coordinación de las diferentes modalidades. Aquí se encuentran dos módulos importantes que serán especificados mas adelante; son los componentes de fusión y de fisión.
- Administrador de dialogo (*Dialog Manager*), a veces considerado parte del comité de integración, este componente es el encargado de manejar la comunicación entre aplicación y los componentes de fusión y fisión.
- Aplicación (o también *Runtime Framework*), el sector de nuestro sistema donde comenzaremos a instanciar las diferentes partes que componen el dominio del problema de la aplicación particular.

Una clara separación de los componentes posibilita el desarrollo en paralelo de los mismos así como beneficia la escalabilidad del sistema en general. Mas información sobre la implementación de la plataforma se verá en la segunda parte (capítulos: 2 y 3).

Tanto en *MIF*[26] como en *i*Chameleon*[27] se presentan soluciones modulares, por ejemplo en el gráfico 1, se muestran los diferentes módulos de MFI en acción.

Particularmente, *i*Chameleon* presenta una arquitectura que se adapta casi directamente a la referenciada anteriormente. En *MIF*, los autores eligen otro acercamiento, en un framework de tres capas donde ponen énfasis en la integración de nuevas modalidades a aplicaciones existentes.

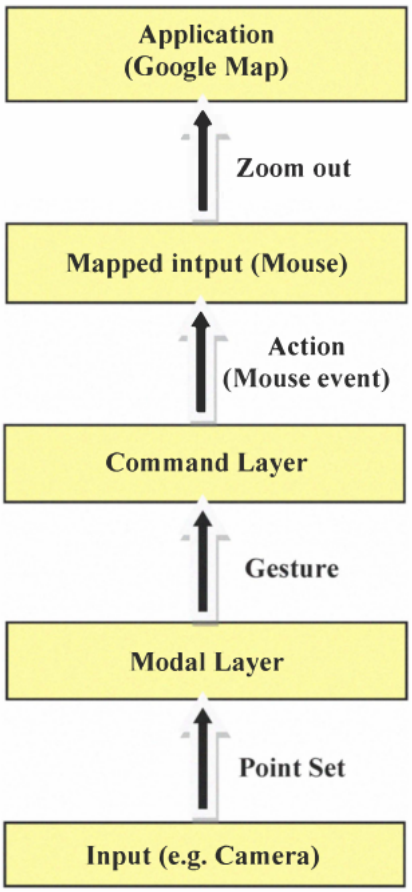

Figura 1: Módulos de MFI en acción.

Otro factor de diseño importante es la capacidad de añadir nuevas modalidades. Es decir, el soporte que se brinda al desarrollador, desde el framework o plataforma, para agregar una nueva modalidad al sistema. En muchos casos, solo se considera un subconjunto de tecnologías particulares, en lugar de un subconjunto de modalidades en general. Por ejemplo, no es lo mismo un framework que permite el desarrollo de aplicaciones que hacen uso del Novint Falcon[1] exclusivamente, que otro que posibilita el desarrollo de aplica-

1 Falcon de la compañía Novint, es un dispositivo que permite generar feedback háptico mediante la utilización de un sistema de servo-motores especialmente diseñado.

ciones que hagan uso de la modalidad háptica en general(e. g., *MIF*). En este punto ambos trabajos presentan buenas soluciones. En particular, en *i*Chameleon* se distingue una etapa del ciclo de desarrollo, la de integración de driver de dispositivo (*Device Driver Integration*), donde los ingenieros de dispositivos pueden desarrollar los módulos necesarios para conectarlos al sistema, los mismos incluso pueden ser re-utilizados, lo que avala técnicas como DRY y minimiza el grado de error.
Siguiendo con la división planteada, se revisan los aspectos de diseño en trabajos donde la atención está puesta en frameworks que brindan soluciones a problemas específicos i. e., entornos educativos, ambientes inteligentes, dispositivos móviles, etc. Entonces los siguientes aspectos serán de interés: re-usabilidad de componentes multimodales, extensibilidad del framework, orientado a ciclo de desarrollo y facilidad de uso. Estos, entre otros aspectos, son tratados por Dumas et al. [12].

Aquí se destaca el enfoque de Cutugno et al. [7], en dicho trabajo descentralizan el framework consiguiendo distribuir la lógica entre servidor y clientes i. e., estos últimos pueden procesar el mecanismo de fisión. Esta separación es lograda a través del uso de un archivo XML de configuración. Por medio de este archivo el desarrollador puede:

- Especificar los eventos que pueden generar un cambio de estado.
- Indicar los comandos que el usuario puede disparar del lado cliente.
- Indicar cómo el usuario puede activar estos comandos a través de las modalidades.

De esta forma, este archivo representa una abstracción, que le permite al desarrollador indicar cuales comandos podrán ser lanzados y a través de que acciones multimodales, todo esto a través de claras etiquetas XML. Así el sistema brinda facilidad de uso y una posibilidad de re-usabilidad de componentes multimodales, la cual depende de la configuración XML del framework y la posibilidad de compartirla entre otras instancias del mismo. Si bien el trabajo de da Silva and da Rocha [8] es un trabajo en progreso, tiene gran potencial para demostrar extensibilidad del framework multimodal y facilidad de uso, ya que estas plataformas de educación online son utilizadas en diferentes ámbitos por un gran conjunto de usuarios finales y no solo desarrolladores experimentados. Finalmente, es notable el trabajo de Lo, Tang, Ngai, Chan, Leong, and Chan [27] ya que además de considerar a la plataforma necesaria junto al framework, los autores

Para mas información sobre el producto pueden visitar el sitio de Novint. Para conocer mas sobre la modalidad háptica pueden dirigirse al apéndice B.

incluyen diversos aspectos, como técnicas que posibilitan re-utilizar interacciones multimodales entre distintas aplicaciones. También presentan un caso de desarrollo completo de una aplicación, haciendo énfasis en la clara separación de responsabilidades que brinda su herramienta, la cual es conseguida gracias al alto grado de modularización del framework; estos distintos componentes pueden asociarse claramente a los distintos roles involucrados en el desarrollo integral de una aplicación multimodal. De esta manera, es posible dividir exitosamente el trabajo entre los distintos profesionales que intervienen en el desarrollo, estos son: ingenieros de dispositivo, diseñadores de modalidad, programadores y diseñadores de interacción.

1.2.1 *Sobre la Plataforma Propuesta*

Teniendo en cuenta estos factores de diseño, la estrategia presentada en este trabajo presenta una arquitectura de componentes, similar a la propuesta por la W3C, la cual tendrá como objetivos definir claramente responsabilidades y facilitar tanto la extension de la plataforma como así también aspectos de escalabilidad de la misma. En cuanto al soporte para añadir nuevas modalidades, se aprovecha la separación de responsabilidades antes mencionada, introduciendo una etapa similar a la de integración de dispositivo de *i*Chameleon* con el agregado de un protocolo de comunicación que permita facilitar la inserción de nuevo hardware. También se tendrá en cuenta, lograr una plataforma fácil de usar por los desarrolladores de aplicaciones web. La posibilidad de conectar nuevas modalidades estará dada por los módulos desarrollados por los ingenieros de modalidades; como el desarrollo de los mismos no esta atado a un lenguaje en particular, la posibilidad de compartirlos para re-utilizarlos no será tan directa, aunque si se gana en flexibilidad. Por otra parte, se considerará brindar una clara separación en componentes, de los cuales se puedan etiquetar claramente las responsabilidades. Con respecto a la capacidad de extender la plataforma, en principio se plantea como para ser extendida "a los extremos", es decir que no solo se contempla el crecimiento a través de la conexión de nuevas modalidades sino también desde el lado cliente, por medio de lo que pueden ser diferentes módulos que agreguen capacidades "extra" en la interacción con la aplicación y con las nuevas señales que esta recibe (producidas por las modalidades).

1.3 COMPARACIÓN DEL ESCENARIO ACTUAL

La tabla 1 compara algunas de las soluciones tratadas en este capítulo. Se considera: si se soporta un canal de entrada y salida para la modalidad, se aclara si es de propósito general o no y cual es el grado de soporte al desarrollo de aplicaciones web en particular.

TRABAJOS - PROPIEDAD	Modalidad: Entrada	Modalidad: Salida	Ámbito de la solución	Orientado a ciclo de desarrollo MMI	Soporte al desarrollo de aplicaciones web
Introduction to a Framework for Multi-modal and Tangible Interaction	Si	Limitada	General	No [a]	Medio
i*Chameleon	Si	Si	General	Si	Medio
An innovative framework to support multimodal interaction with Smart environments.	Si	Limitada	Smart Environments	Parcial [b]	Bajo
e-Learning Environment with Multimodal Interaction	Si	No	Plataformas Educativas, accesibilidad	No [a]	Medio
Multimodal Framework for Mobile Interaction	Si	No	Dispositivos Móviles	Parcial [b]	Medio
A Framework for the Development of Distributed Interactive Applications	Posible	Posible	UI Multi-dispositivo	No [a]	Bajo

Cuadro 1: Escenario actual en soporte multimodal para aplicaciones.

a Los casos negativos o de ausencia de alguna propiedad, se deben a no estar explícitamente indicados en los trabajos, todas las soluciones analizadas presentan buenas practicas de desarrollo de software.

b Arq. en componentes, clara separación de los mismos pero no hay aclaración en el trabajo sobre este aspecto.

1.4 RESUMEN DEL CAPÍTULO

Se han analizado seis trabajos que representan la vanguardia en cuanto a tecnologías para el desarrollo de aplicaciones multimodales. Los mismos fueron clasificados de acuerdo al formato y a la especificidad de la solución. Entre los trabajos revisados se encuentra también el análisis de un framework para el desarrollo de aplicaciones multidispositivo con interfaces de usuario migratorias; este trabajo se considera como transversal a los otros y puntualmente son de interés las soluciones arquitectónicas que brinda a un problema similar.

También se destaca la plataforma propuesta por Lo, Tang, Ngai, Chan, Leong, and Chan [27] como la solución mas completa y de referencia para este trabajo. Luego se analizan algunos aspectos de diseño que deben sortear tecnologías de este estilo. Especificamente, se analiza: modularidad de la plataforma en relación a arquitecturas conceptuales como la de la W3C [39], facilidad para añadir nuevas modalidades, posibilidad de re-utilizar modalidades, extensibilidad, facilidad de uso para los desarrolladores y soporte para un ciclo de desarrollo. También se exponen las estrategias seleccionadas por la solución aquí propuesta.

Finalmente se presenta un cuadro que aglomera algunas de las características mas importantes de los trabajos analizados.

Parte II

ASPECTOS TÉCNICOS

A continuación se exponen las características del medio al cual se agregará el soporte para las diferentes modalidades. Luego se esgrimirán los aspectos particulares que esta arquitectura propone y su relación con otras alternativas clásicas . En este trabajo, el "medio" es la Web como plataforma. Sobre la misma se construye la solución aquí propuesta; es entonces importante conocer las condiciones que este medio impone para comprender mejor la plataforma.

Al proponer una solución novedosa en determinados aspectos, fue necesario primero buscar algunos puntos de comparación en el desarrollo de plataformas multimodales. A continuación se analizan y comparan la arquitectura de *Plusultra* con los trabajos de Dumas y la W3C.

2 INFRAESTRUCTURA

En los trabajos analizados en el capitulo 1, las soluciones propuestas han necesitado algún conjunto mínimo de capas de abstracción, que les permita brindar el soporte para la conexión o el agregado de alguna modalidad especifica en un sistema particular.

El trabajo de Dumas et al. [12] presenta una arquitectura genérica para el desarrollo de aplicaciones multimodales, como se puede ver en la figura 4. Esta será una referencia importante en esta parte del documento.

En el presente capitulo se introducirán las decisiones de bajo nivel de la solución propuesta para la conexión de nuevas modalidades al contexto de una aplicación web, las mismas constituirán la plataforma denominada *Plusultra*.

Primero se describen las características generales de las aplicaciones web (clientes de la plataforma), para luego introducir las decisiones arquitectónicas. En el siguiente capitulo se completan los aspectos de desarrollo, añadiendo el modulo cliente (que corre en las aplicaciones web).

2.1 CARACTERÍSTICAS DE LAS APLICACIONES WEB

El concepto de *aplicación web* ha evolucionado desde sus inicios, con algunos pasos destacados como el surgimiento de AJAX en los 90s, que introdujo una nueva capa en el cliente mediante un conjunto de tecnologías con el fin de fragmentar la comunicación con el servidor, mejorando la experiencia de usuario; según indica Garrett et al. [18]. Desde ese momento, las aplicaciones web fueron delegando, consistentemente, lógica de negocio hacia el cliente.

En los últimos años, han surgido conceptos desde el ámbito académico como RIA (Rich Internet Applications) [15], que busca dar un marco conceptual a dicho movimiento. Por el lado de la industria, el desplazamiento hacia el cliente también es claro, librerías como Backbone.js [22] (desde el 2010), como una de las mas populares e inspiradoras para otros frameworks como Ember.js [1] y Angular.js [19], han llevado el paradigma MVC fuera del servidor.

Por otra parte, el proyecto Node.js [23], a través de V8, permite la utilización de JavaScript del lado del servidor. Esta característica ha generado versatilidad en el software desarrollado, la misma puede observarse con proyectos como *Browserify* [5] que posibilita la generación de módulos híbridos, capaces de correr en ambos extremos (cliente y servidor). Teniendo en cuenta esto, sumado a la ya

conocida sólida presencia de JavaScript, como lenguaje estándar en el entorno del navegador, ha impulsado el ecosistema de desarrollo en JavaScript, posicionando a npm, como uno de los manejadores de paquetes mas populares hoy en día [35].

2.1.1 *Aspectos Particulares de las Aplicaciones Web Actuales*

Las aplicaciones web han evolucionado tomando formas similares a las de las aplicaciones de escritorio. Es decir, aplicaciones tipo single-page (SPA), con múltiples componentes de interfaz de usuario que permitan la interacción con datos in-situ sin necesidad de refrescar o navegar a diferentes secciones. Por *single page applications* (SPA) o *single page interface* (SPI)[36] se hace referencia a aquellas aplicaciones web que cuentan con las siguientes características: minimizan la carga inicial de componentes a un request (*single page load*), modularizan el contenido JavaScript (abstrayendo las llamadas AJAX con el servidor), tienen capacidad de manejar ruteo y la historia del navegador, soporte de templates *client-side*, comunicación bidireccional (*real-time*) con el servidor y capacidad de usar almacenamiento local en el cliente.

El concepto de RIA (*Rich Internet Applications*) define estos nuevos tipos de aplicaciones. Es decir, el cambio que hubo desde compartir contenido estático hasta el dinamismo actual. Con el fin de expandir las características definidas a través del termino *RIA* y buscando describir mas estrictamente el escenario donde serán incluidas la capacidad de usar múltiples modalidades, se define el siguiente listado de características que hacen a una aplicación web actual:

- **Aplicaciones inherentemente distribuidas**; de acuerdo al paradigma de comunicaciones cliente-servidor, aún vigente en la web, las aplicaciones se conectan al servidor para obtener algo y realizar alguna tarea, el servidor luego puede centralizar resultados. En la web, ocurre algo similar, las aplicaciones web, a través de los agentes de usuario, descargan toda la aplicación, incluso algún conjunto de datos básico; esto le permite al usuario interactuar con la aplicación en su completitud e incluso de forma ”offline” si se adecuan algunos parámetros.

- **Aplicaciones inherentemente multiplataforma**; luego del surgimiento de la *web 2.0*, se consolida la web como plataforma (*Web as a plataform*), este concepto se apoya en la multiplicidad de servicios que la web ofrece o que una aplicación desarrollada para este ecosistema puede consumir, de acuerdo a Group [20]. Estos servicios, son ofrecidos a través de vínculos nativos, por el navegador del usuario. Las aplicaciones son codificadas en lenguajes estándares, que corren dentro del ambiente provisto por el navegador.

- **Clientes "pesados" con características de aplicaciones de escritorio**; la lógica de aplicación ya no se encuentra totalmente en el servidor. El servidor provee funciones de centralización (i. e.autenticación) o recursos a través de *APIs* (i. e.REST); pero la manipulación y el tratado de los mismos ocurre del lado del cliente.
- **APIs estándar para consumo de características nativas de hardware**; a través de la W3C, se busca expandir la web en múltiples direcciones. En [20] se listan varios de estos tópicos. Por ejemplo, se destaca, la API de Vibración [41] o la especificación para el acceso a eventos táctil [40]; las mismas definen como utilizar diferentes capacidades de interacción.
- **Adaptabilidad**; mediante estrategias como *responsive-design* es posible no solo adaptar contenido de acuerdo a la pantalla del dispositivo, si no también, a las características disponibles de cada uno. Se desarrolla buscando una mejora progresiva. Entre otras ventajas, esto permite mantener una única base de código en lugar de diseñar diferentes aplicaciones, tanto para escritorio como para dispositivos móviles.
- **Bidireccional en el inicio de la conversación**; *Web Sockets* es otra especificación de la W3C [38], con implementaciones concretas utilizadas por la industria, como Socket.io o WS, entre otras; permiten la comunicación bidireccional entre cliente y servidor y el "inicio de la conversación" por parte de las aplicaciones web. Esto posibilita un escenario con características de *tiempo-real*.

Socket.io y WS, son proyectos open source, desarrollados en JavaScript, que implementan el protocolo WebSocket. Si bien difieren en como lo implementan, ambas librerías permiten crear clientes y servidores de WebSockets.

2.2 ARQUITECTURA PROPUESTA

Teniendo en cuenta el escenario planteado, insertar de forma genérica una nueva modalidad implica algunos desafíos técnicos. A continuación se describe la estrategia elegida y su fundamentación. La misma se compone, a grandes rasgos, de tres partes; dos *APIs*, una para el navegador como una dependencia mas de la aplicación web y la otra para conectar la nueva modalidad al entorno web; el componente restante es una pasarela de mensajes. Es el componente que sera descrito en este capitulo, los otros dos serán tratados mas adelante.

2.2.1 *Pasarela de Mensajes: Introducción*

En la figura 2 se puede observar de manera rápida la arquitectura del sistema. En esta sección se pondrá el foco sobre el *Message Gateway*

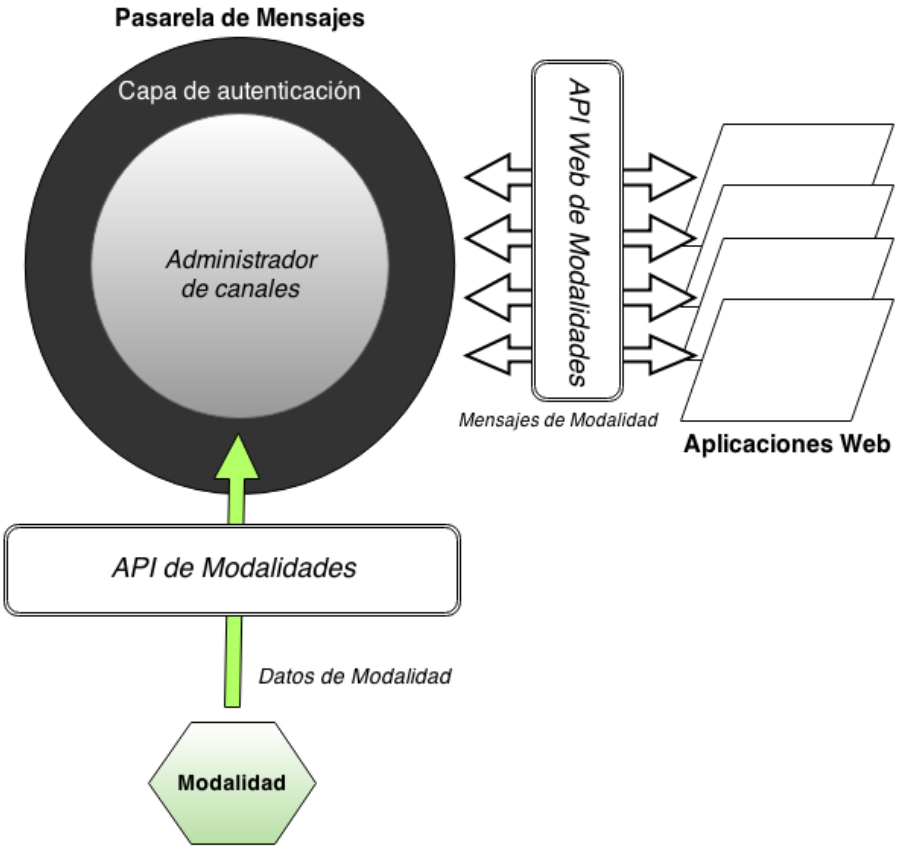

Figura 2: Componentes de arquitectura para una aplicación web

o *Pasarela de Mensajes*; de ahora en mas se usará PM para referirse a la misma con el fin de abreviar. En la construcción de la PM se tuvo en cuenta un modelo de capas, con el *Channels Manager* o *Administrador de canales* como núcleo y único componente requerido para su funcionamiento. El resto de las capas agrega funcionalidades, de manera muy similar a la de un *middleware*. Es decir, interceptan los mensajes, los utilizan como entrada para ampliar el contexto, validar el pedido o cualquier otra tarea que pueda hacerse con los datos del mensaje; luego de realizada dicha tarea, le pasan el mensaje al siguiente nivel de profundidad hasta llegar al núcleo.

El núcleo, a través de una base de datos en memoria, mantiene los canales activos de comunicación. Un canal equivale a una aplicación. Los canales pueden ser multiplexados, es decir, usando un canal, podemos fragmentarlo y generar sub-canales en caso de ser necesario. El núcleo, implementa un patrón similar al de *Publishers/Subscribers*, entonces, de forma simple, cuando recibe un mensaje, chequea el canal por el cual debe retransmitirlo hacia todos los clientes suscritos al mismo. Como se ha mencionado antes, es valido visualizar una cardinalidad 1 a 1 entre canales y aplicaciones.

Por otra parte, una aplicación puede tener *N* clientes. Cada cliente conectado a la aplicación, es automáticamente suscrito a su corres-

pondiente canal, a través de la API provista al navegador (usando el módulo *Gyes* que veremos en el próximo capitulo). De manera similar, la modalidad, a través de la interfaz provista para las mismas se conectará al canal de aplicación que le corresponda. Ya sea una persona (desarrollador) o un equipo de trabajo multimodal, deberán tener acceso al token de acceso por aplicación. Este token podría ser accedido mediante alguna aplicación de registro a la plataforma como servicio o podría ser simplemente generado y distribuido internamente en el grupo de trabajo, si se toma la decisión de hostear de forma autónoma una instancia de la plataforma.

De cualquier manera, se genera un puente entre modalidad y aplicaciones cliente, lo que equivale a decir que es posible intercambiar mensajes bidireccionalmente entre aplicaciones "cliente" y modalidades conectadas a la aplicación. Por lo que no solo los clientes se ven afectados por la señal generada por la modalidad, si no que ellos también pueden modificar, por ejemplo, la configuración de la modalidad en tiempo-real. La cual es una característica con la que muy pocos sistemas multimodales cuentan.

Otra capa importante pensando en la plataforma como PaaS (*Platform as a Service*), y que esta provista en el sistema de forma inicial, es la de autenticación (*Authentication Layer*). La forma de autenticar clientes es algo simple, pero efectiva para el caso. Se basa en una estrategia tipo "algo que posea" el cliente. En este caso, al registrar una aplicación web que usará esta plataforma, la misma recibirá un *token*. Tanto el desarrollador de modalidad (si es que lo hay), como el desarrollador de la aplicación web, comparten este token. El sistema registra a la aplicación con un token y un canal. Cabe destacar que en este aspecto puntual, por aplicación se refiere tanto a las modalidades usadas como a los clientes web (son todos "parte de").

PaaS o en español, plataforma como servicio, hace referencia a una plataforma en la nube que brinda una solución que puede ser aplicada por el consumidor como una capa mas a su sistema, abstrayéndolo de diferentes cuestiones relacionadas al hardware y brindándole capacidades de escalabilidad.

Al llegar un mensaje nuevo, tanto por parte de un cliente como de una modalidad, esta capa lo intercepta, toma el token que viene con el mensaje, y lo valida. Si es positiva, el mensaje continua a la siguiente capa, y se cachea el token para mejorar la performance en futuros mensajes. Si no es valido, el mensaje es desechado y no continua. Este proceso se describe aquí de manera rápida, porque el foco, por el momento no esta puesto en la seguridad del sistema, pero se cree que es un mecanismo que junto con otras previsiones puede garantizar autenticidad sin producir demasiado *overhead* en la comunicación.

2.2.2 *Pasarela de Mensajes: Diseño*

La pasarela de mensajes *Plusultra*, esta implementada usando un esquema de interacción distribuida tipo *publish/subscriber*, ver figura 3. Se decidió utilizar un esquema de comunicación basado en eventos para favorecer a los dispositivos de modalidad, los principales productores del sistema. Los dispositivos de modalidad pueden ser cla-

sificados como reconocedores, sintetizadores o ambos. De cualquier forma, estos se comunicarán a través de eventos, e. g.generando uno luego de reconocer un gesto (patrón) o recibiendo un evento que señalice la ocurrencia de una fisión, dando lugar a una posible acción de sintetizado.

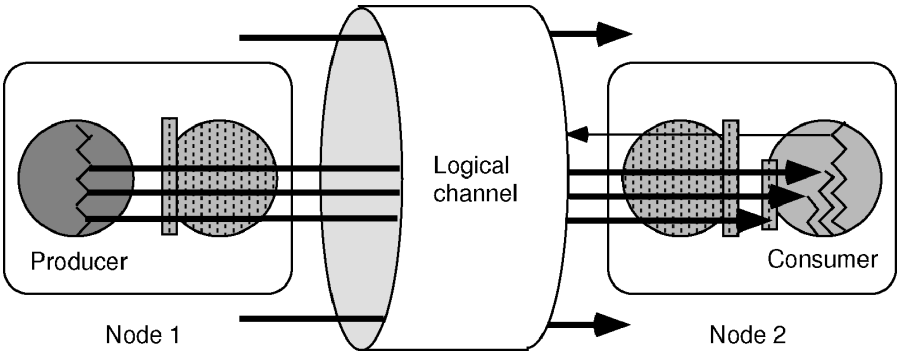

Figura 3: El paradigma de interacción Publishers/Subscribers, de acuerdo a Kermarrec [24].

Los eventos pueden ser mensajes en texto plano o con algún formato, como JSON en este caso. Los mismos encapsulan un protocolo de comunicación que se definirá junto al modulo cliente *Gyes*, mas adelante.

Redis, es un sistema open source, que permite almacenamiento y cacheo de estructuras clave-valor. *Es también conocido como un servidor para estructuras de datos. Para mas información visitar: redis.io*

La información sobre canales y aplicaciones activas es almacenada utilizando algún sistema de base de datos en memoria, en la implementación actual se utiliza **Redis** (configurado especialmente para no persistir datos), esto permite un acceso rápido a la información de una aplicación y ayuda a perfilar a la plataforma como un sistema de intercambio de mensajes volátil, es decir, esto no es una base de datos ni una cola de mensajes.

Cada instancia de la pasarela funciona como un modulo *productor/consumidor* distribuido. Esto permite ver al sistema en general como un motor de notificación de eventos distribuido en un conjunto de procesos y servidores, compartiendo diferentes bases de datos en memoria. Esta visión se corresponde con la del sistema funcionando como un PaaS, aunque también puede correr *standalone*, es decir, como una unidad de trabajo particular. Esta forma de ver al sistema corresponde con una arquitectura *publishers/subscribers* centralizada, muy similar a una cola de mensajes.

Las principales ventajas de implementar a *Plusultra* como un sistema de notificación *publish/subscriber* radican en tres cualidades, de acuerdo con Kermarrec [24]:

- **Desacoplamiento en tiempo**; las partes que interactúan, dispositivos de modalidad y aplicaciones, no necesitan estar conectadas al mismo tiempo para comunicarse. Los dispositivos de

modalidad son independientes al número de clientes conectados y viceversa. La importancia radica en el momento, esto es, si se produce una señal entonces puedo captarla y hacer algo; "si ocurrió una señal en el pasado no me interesa".

- **Desacoplamiento en espacio**; este desacople se produce entre aplicaciones y dispositivos de modalidad e implica que no es necesario que estos entes se conozcan entre ellos para funcionar. Las dispositivos envían señales, en forma de eventos, hacia la plataforma y continúan su trabajo. Por otra parte, los clientes, es decir las aplicaciones web, cuando detectan una nueva fusión (que pudo haberse producido) por una señal de una modalidad, pueden ejecutar un *callback* asociado en ese momento.
- **Desacoplamiento en sincronización**; No ocurre ningún tipo de bloqueo entre dispositivos de modalidad y clientes web. Cuando una señal es reconocida y un evento es disparado, el driver del dispositivo nunca esperará por una respuesta para "continuar". De forma similar, cuando algo ocurre y se debe señalizar a una aplicación, este evento particular generará la ejecución de un *callback* en el cliente; es decir, éste nunca se detuvo a esperar por la ocurrencia de dicho evento.

La plataforma fue diseñada teniendo en cuentas estas características. Mas aun, un sistema desarrollado teniendo en cuenta estos items tendrá facilidades al momento de escalar [24].

Los desafíos o posibles inconvenientes a los que puede verse afectado un sistema de notificaciones de este tipo se encuentran en el hardware y plataforma sobre la que corran, así como también de las posibles limitaciones en la topología y protocolos de comunicación usados. Por ejemplo, el sistema puede verse beneficiado si se utiliza un protocolo de comunicación probabilístico en lugar de uno diseñado para *WAN* como RMTP, que genera *overhead* debido a la gran cantidad de mensajes de confirmación que ocurrirían [24]. La extensión y consolidación de esta *Plusultra* como PaaS queda fuera del alcance de este trabajo y sería posible analizarla solo con mas tiempo y en condiciones diferentes.

2.2.3 *Pasarela de Mensajes: Implementación*

Para implementar la pasarela se ha decidido utilizar **Node.js**. Esta tecnología, comprende a un entorno de desarrollo multi-plataforma (gracias al motor V8 en el que corre), orientado a I/O y con una arquitectura basada en eventos, lo que lo convierte en una herramienta favorable para desarrollar servicios o aplicaciones orientadas a *networking*, como es el caso de la plataforma aquí propuesta; donde el foco esta puesto en las comunicaciones y no en el procesamiento. Otra ventajas de usar Node.js, es que se mantiene un mismo lenguaje

para el código del proyecto: JavaScript. Haciendo mas fácil pasar de una tarea a otra. Por ejemplo cambiar de contexto, al desarrollar un modulo cliente y luego volver a la plataforma.

2.3 COMPARACIÓN CON LA ARQUITECTURA CLÁSICA

Se considera como *arquitectura clásica* para un sistema con interacciones multimodales a la definida por Dumas et al. [12]. En dicho trabajo, los autores abstraen las características genéricas que un sistema multimodal debería poseer, definen componentes arquitectónicos fundamentales.

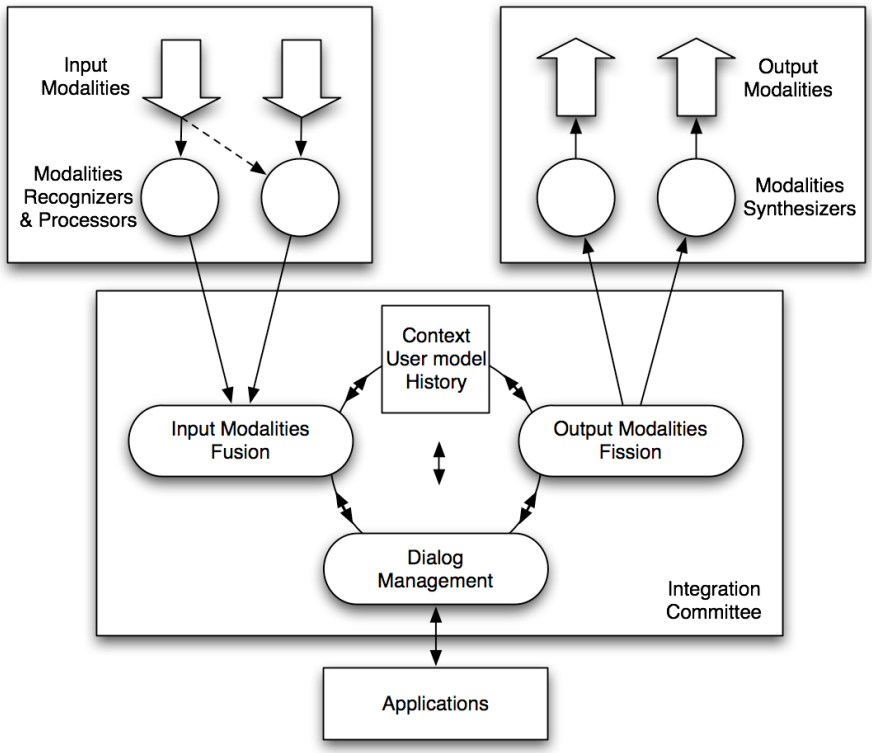

Figura 4: La arquitectura de un sistema multimodal clásico por Dumas et al. [12].

En la figura 4 se pueden observar los principales componentes, el *comité de integración* con sus sub-componentes y las modalidades con los correspondientes, *reconocedores*, *pre-procesadores* y *sintetizadores* entre otros posibles filtros.

Este es un esquema claro en cuanto a la división de responsabilidades. La arquitectura propuesta en este capítulo se concentra en la forma de interconectar las modalidades, la implementación realizada del **comité de integración** será analizada en el próximo capítulo. De acuerdo a las características antes mencionadas, la arquitectura debe

ser capaz de insertarse en un ambiente distribuido. Es decir, debe ser capaz de conectar de una forma clara y efectiva modalidades (entrada y salida) con múltiples clientes de una aplicación web, en tiempo real. Para eso, como ya se ha mencionado, se utiliza un sistema de interconexión basado en eventos. Entones, re-interpretando la arquitectura clásica, el diseño de la solución propuesta se ve en la figura 5.

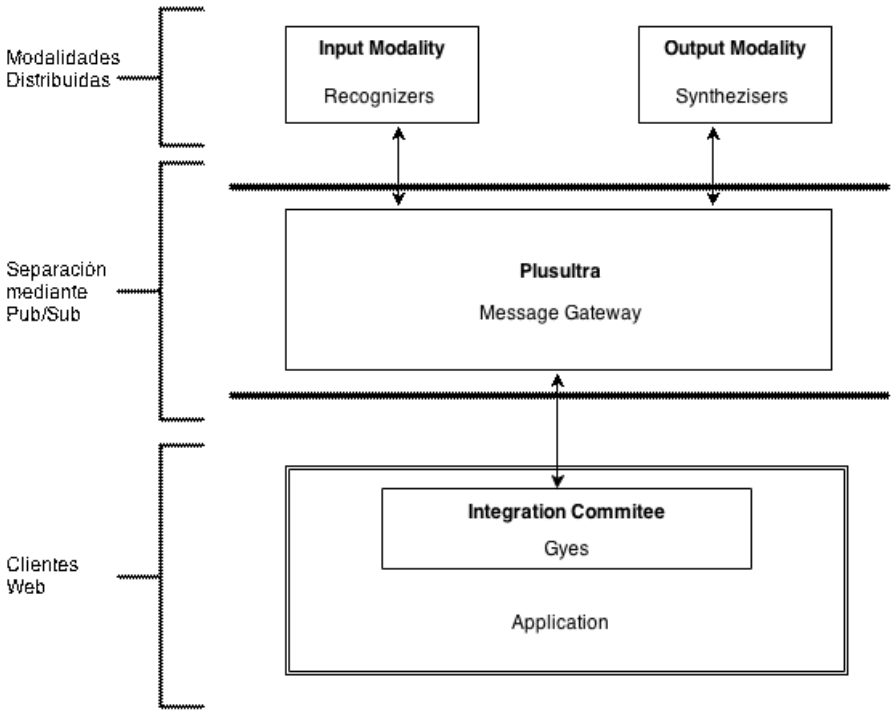

Figura 5: La arquitectura propuesta en este trabajo.

La *pasarela de mensajes*, se convierte en un aspecto central, que permite mantener la separación de responsabilidades original de la aplicación y a la vez adaptarse al contexto de una aplicación web.

Ese diseño brinda otra ventaja, importante si se piensa a la *plataforma como servicio*, se trata de la capacidad para escalar horizontalmente, como se ha mencionado en la sección 2.2.2. En el siguiente gráfico 6 se muestra un hipotético escenario escalable. Esta es solo una configuración posible para conseguir escalabilidad, utiliza un balanceador de carga como una dependencia externa (i. e.nginx); otra alternativa podría desechar la necesidad de agregar otro componente mediante el desarrollo de una nueva capa de "middleware" que auto-balancee la carga y distribuya a otro cliente, de los posibles instanciados por el usuario de la plataforma. Hay mas alternativas para explorar y queda abierto (escapando a los objetivos del presente trabajo), en el campo de PaaS o de servicios sobre Internet en general, elegir y probar los mejores caminos.

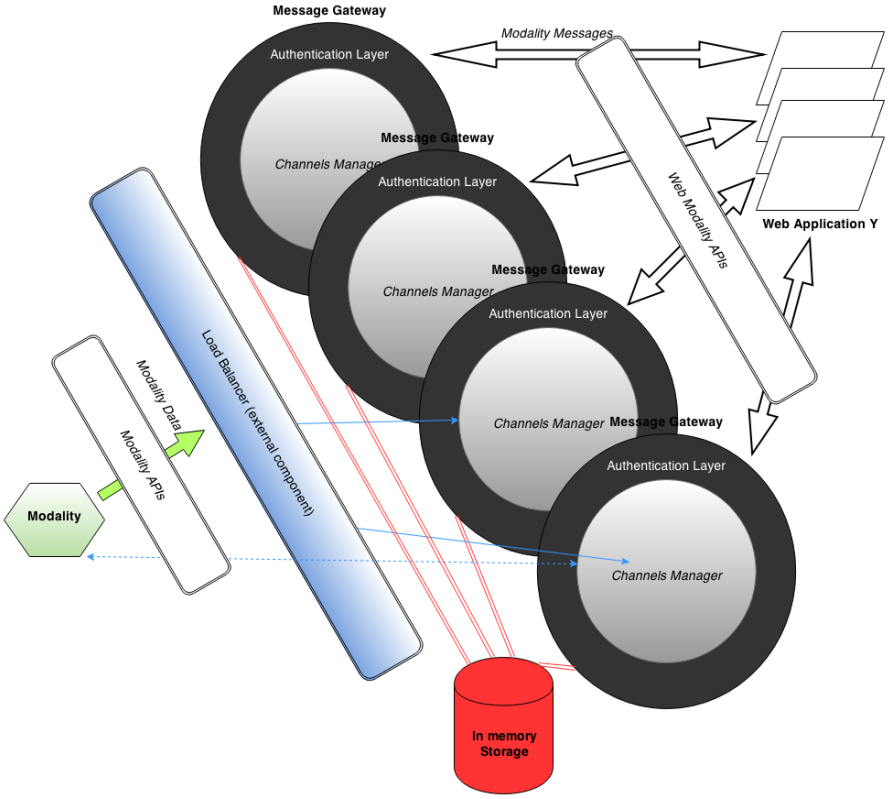

Figura 6: Componentes de arquitectura en un escenario escalable, múltiples PMs.

2.4 COMPARACIÓN CON LA ARQUITECTURA PROPUESTA POR LA W3C

En W3C [39], se propone una arquitectura similar a la denominada "clásica". Dentro del *Runtime Framework*, donde se definen muchos aspectos técnicos y que esta delegado a quien implemente la arquitectura se define un *Interaction Manager*(administrador de interacción) o IM, que debe recibir todos los eventos generados por los diferentes *Modality Components*(componentes de modalidad) o MC; allí se define la interacción con el componente. Algo similar a los componentes de fisión y fusión de la arquitectura clásica, solo que aquí se encuentran distribuidos. El modulo IM es similar a la pasarela de mensajes de la estrategia propuesta. La diferencia radica, en parte, en la fuerte orientación a eventos y a trabajar como máquina de estados por parte del IM. En cambio el componente PM, en un principio, esta orientado a trabajar transmitiendo datos en "crudo" por parte de las modalidades y no eventos. Aunque esto puede modificarse, añadiendo algún tipo de pre-procesador de los datos generados por la modalidad. Esto depende del ingeniero de modalidad.

Las modalidades se definen usando algún tipo de lenguaje de marcado (i. e.CCXML, SCXML, HTML, entre otros). Otra diferencia importante entre el sistema propuesto por la W3C y el del presente trabajo, es la API que conecta el IM con los diferentes MC. Es la sección menos flexible del documento W3C [39]. En ella se definen cuestiones tales como activación y detención de la modalidad, acciones previas a recibir un mensaje, inicio y fin de la conversación. Esta API le da fuerza al modelo de máquina de estados.

En este trabajo no se considera un modelo de maquina de estado porque en un principio se busca favorecer al máximo, o lo que es lo mismo, minimizar cualquier tipo de obstactulo en la transmisión de datos "crudos". Por otra parte, la estrategia aquí presentada, mueve los componentes de fisión y fusión a los clientes y le entrega dicha responsabilidad al desarrollador de la aplicación para que fusione con libertad el contexto de la aplicación que esta desarrollando con las modalidades disponibles. Se cree que de esta manera se puede explotar el uso de las modalidades.

En el gráfico 7 se muestra la arquitectura propuesta por la W3C antes mencionada.

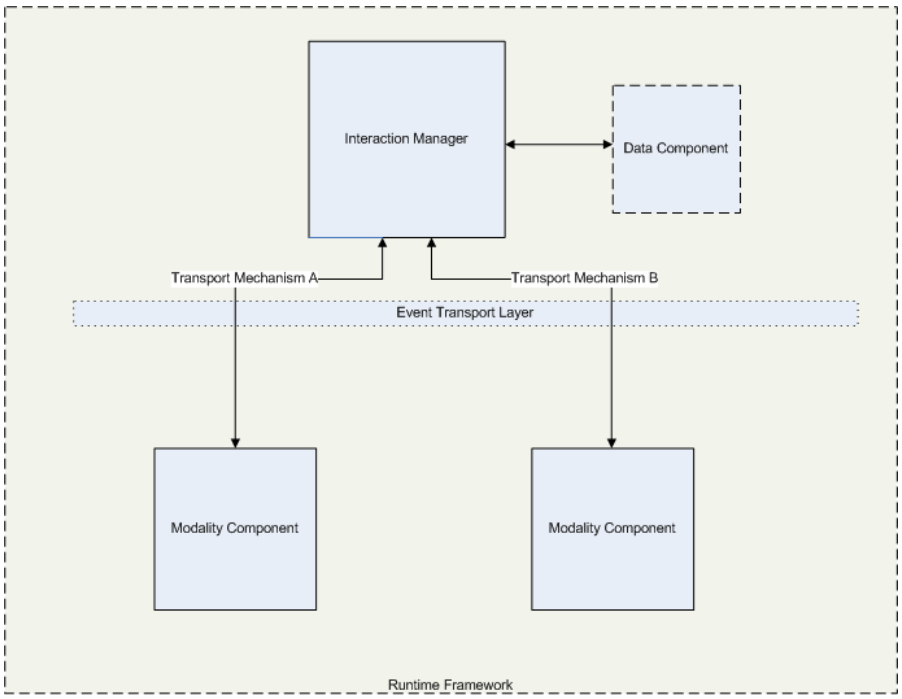

Figura 7: Arquitectura de la W3C en tiempo de ejecución

2.5 RESUMEN DEL CAPÍTULO

En este capítulo se ha mostrado la estrategia elegida en la creación de la arquitectura necesaria para poder expandir las capacidades de las aplicaciones web actuales con el uso de nuevas modalidades. Se han detallado características particulares al ambiente de las aplicaciones web y que han influido en las decisiones tomadas para desarrollar dicha arquitectura. A su vez, se ha comparado el trabajo realizado con otras soluciones conocidas, como son el modelo presentado por Dumas et al. [12] y el trabajo del grupo de MMI W3C [39]

Mas adelante, se completara la arquitectura, definiendo la implementación elegida para el "núcleo" de la misma, como son los componentes de *fusión* y *fisión*; con el fin de mantener la claridad se ha decidido analizarlos por separado.

3 ENLACE CON LAS MODALIDADES

3.1 ARQUITECTURA PROPUESTA

A grandes rasgos, los puntos de entrada a la plataforma *Plusultra* introducida en el capítulo anterior, estarán ubicados en dos lugares diferentes:

- **Clientes Web**; aquí también reside la lógica de la aplicación. En conjunto crean una experiencia multimodal en la web. Dentro de los clientes web, podemos encontrar capacidades extra que pueden ser usadas a su vez como modalidades (i. e. un cliente corriendo en un smartphone puede tener acceso a sensores, que a su vez, pueden ser usados como reconocedores). Mas adelante en este capítulo, se analizará esta situación.
- **Dispositivos de modalidad**; se refiere a todos aquellos dispositivos especialmente dedicados a reconocer o sintetizar una modalidad, e. g. Falcon de Novint o Leap Motion, por mencionar algunos.

Para conectar estos puntos de entrada, se construyo un modulo de acceso que puede ser consumido por igual tanto del lado del cliente como del servidor o *standalone* (dispositivos de modalidad). Ademas se ha desarrollado una primera versión de un protocolo de comunicación que acompaña dicha interfaz de comunicación. Este protocolo abre el camino a otros desarrolladores para que puedan diseñar soluciones utilizando diferentes lenguajes. En el mismo se describe los mensajes utilizados para un enlace efectivo con la plataforma *Plusultra*. De esta manera, siguiendo a este protocolo versionado, la API puede portarse a otros lenguajes de programación extendiendo así el alcance y uso de la plataforma.

3.1.1 *Modulo Híbrido Cliente/Servidor*

Prácticamente todo el desarrollo de este trabajo se ha llevado a cabo utilizando el lenguaje JavaScript. Dado que se busca expandir la capacidad actual de comunicación de las aplicaciones web, es lógico basar la mayor parte de la base de código en dicho lenguaje. Usando la plataforma Node.js, es posible correr

JavaScript del lado del servidor también, haciéndolo ubicuo en múltiples escenarios. Teniendo en cuenta estas capacidades, se decidió desarrollar un modulo que pueda aprovechar estos aspectos y ser de utilidad en diversas circunstancias o casos de uso.

De esta forma se introduce el módulo *Gyes*. A través de este componente es posible conectarse al sistema, transmitir datos de modalidades, crear nuevas modalidades, crear interpretaciones, capturar salidas del sistema de fisión; por nombrar los casos mas importantes.

3.1.2 *Vista General*

El modulo *Gyes* permite realizar diferentes tareas, las cuales tienen como finalidad interactuar con la plataforma *Plusultra* y por consiguiente con el resto de los actores de la aplicación, es decir, con el contexto entero de la aplicación web. El siguiente gráfico 8 simplifica una de las principales actividades, utilizar una modalidad como reconocedora de algún tipo de señal para luego propagarla en la aplicación, mediante la plataforma.

3.2 DESCRIPCIÓN DE LA INTERFAZ DEL MODULO

En la figura 9 se observa de manera directa los principales componentes del modulo de acceso al sistema. Luego extenderemos esta visión a la interacción que ocurre con la plataforma. A continuación se describirán las interfaces de los componentes junto a una breve introducción a la primer version del protocolo de comunicación entre *Plusultra* y *Gyes*.

3.2.1 *Componente de Conexión*

Es el componente principal. Desde aquí no solo accedemos a la plataforma si no también al resto de los componentes.

La función principal es proveer una forma de conectar, tanto para modalidades como clientes a *Plusultra*.

Constructor:

```
1 Gyes( appKey, uri, opts )
```

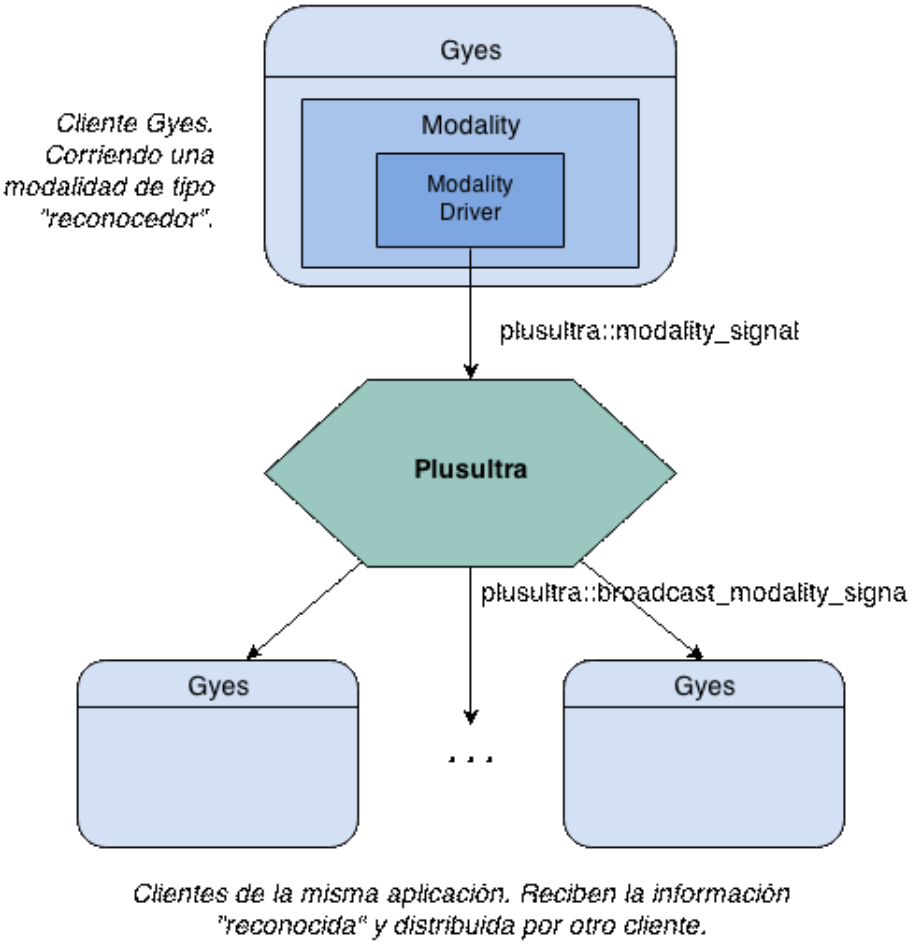

Figura 8: Modulo Gyes en acción: transmitiendo señales

Crea una nueva instancia del módulo *Gyes*. Automáticamente inicia una conexión con la plataforma *Plusultra*.

Principales métodos:

```
gyes::authenticate( appKey )
```

Una vez creada la instancia, necesitamos autenticar nuestra aplicación. Para eso el desarrollador web debe contar con una llave, previamente generada, por ejemplo usando algún servicio donde registre la aplicación y la cantidad de instancias a consumir de la plataforma.

Plusultra es una plataforma que ha sido diseñada para ser fácilmente escalable a un modelo de PAAS (*Platform as a Service*), de esta forma, múltiples instancias del módulo de plataforma pueden servir a diferentes aplicaciones. Por

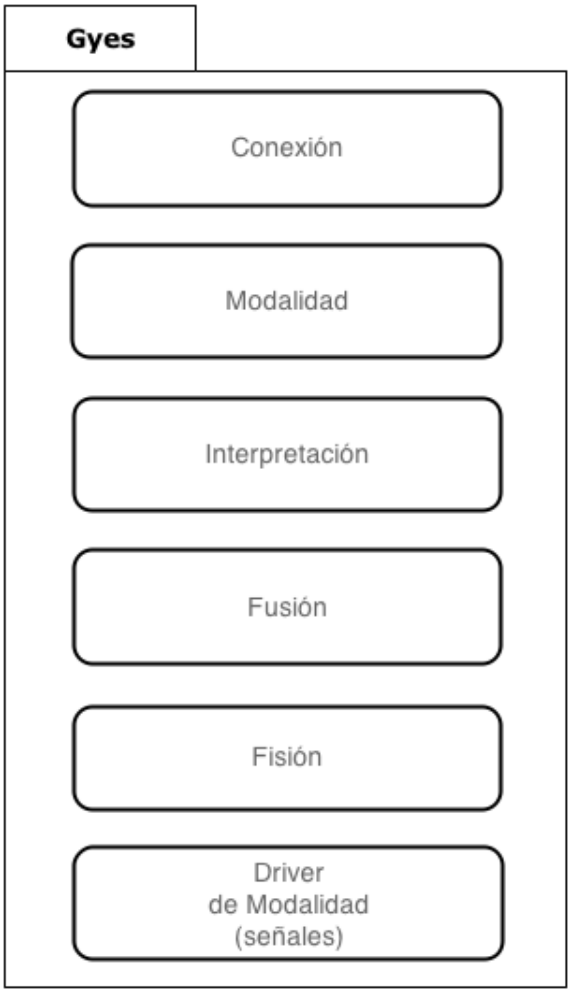

Figura 9: Principales componentes del modulo Gyes

eso es necesario que cada aplicación posea una llave que la distinga.

```
gyes::addModality( aModality )
```

Permite agregar una nueva modalidad al sistema. Esto es útil para indicar que determinado cliente cuenta con una modalidad particular la cual, al estar "agregada" a la plataforma, puede compartir la información que genera.

3.2.2 *Componente de Modalidad*

Este módulo permite crear y agregar nuevas modalidades al sistema. Dada la naturaleza variada de las mismas, se decidió separar comportamiento de representación/identificación. El comportamiento esta definido por los drivers, aquí especificamos una representación. Dentro de una aplicación, pueden

conectarse mas de una modalidad y por ahora, una modalidad puede usar un driver.

Constructor:

```
Gyes::Modality( name, type, opts )
```

Permite crear un nuevo objeto que representa a una modalidad, la cual podrá ser agregada al sistema. Recibe un nombre (*name*), tipo (*type*), de acuerdo a si es de entrada/salida o ambos y luego opciones (*opts*) particulares para compatibilidad a futuro.

Principales Métodos:

```
modality::use( modalityDriver )
```

Conecta una instancia de modalidad con una instancia de driver.

3.2.3 *Componente de Fusión Distribuida*

Permite controlar el motor de fusión de manera programable. Este componente captura *señales* que pueden ser generadas por modalidades o por eventos de la aplicación.

Constructor:

```
Gyes::Fusion( opts )
```

Genera una nueva instancia del motor de fusión.

Principales métodos:

```
fusion::fuse( anInterpretation )
```

Detecta un determinado conjunto de señales, definidos por una interpretación. Cuando dicha interpretación ocurre, puede distribuir esta información entre todos los clientes y comenzar el proceso de activar al sistema de fisión.

La captura de señales se produce manteniendo una ventana de tiempo, donde la misma se actualiza (extiende) cada vez que ocurre un evento que pertenezca a una interpretación. Esto ocurre de forma independiente y distribuida

entre todos de la aplicación. Es decir, cada uno cuenta con un motor de fusión. Para que esto funcione, las señales deben ser distribuidas, usando la plataforma, en tiempo real.

3.2.4 *Componente de Fisión Distribuida*

Brinda acceso al sistema de fisión. El desarrollador web, encuentra en este componente un "punto de salida" de la plataforma, donde puede conectar la lógica de la aplicación web mientras consume información generada por las distintas señales que recorren el sistema.

Constructor:

```
Gyes::Fission( opts )
```

Genera una nueva instancia del motor de fisión.

Principales Métodos:

```
fission::on( interpretationName, callbackFn )
```

El sistema de fisión permite al desarrollador conectar la ocurrencia, asincrónica, de una interpretación con código que afecte a la lógica de negocios de la aplicación.

3.2.5 *Componente de Interpretación*

En los componentes anteriores se menciono el concepto de *interpretación*, aquí se muestra su interfaz y lo que representa.

Este componente permite agrupar un conjunto determinado de señales en un único objeto. Una interpretación puede representar la conjunción de diferentes modalidades en un único evento. Las interpretaciones son usadas por el motor de fusión para detectar la ocurrencia de dichas señales e interpretarlas como un único evento alertando al resto de los componentes del sistema sobre dicha ocurrencia.

Constructor:

```
Gyes::Interpretation( eventsList )
```

Genera una nueva instancia de interpretación. Recibe una lista de señales la cual servirá para organizar al sistema de fusión

en el escaneo y detección de ocurrencias de interpretaciones.

Principales Métodos:

```
interpretation::getName()
```

Regresa una identificación única para la interpretación. Útil para "escuchar" por una determinada ocurrencia.

```
interpretation::canSynthetize( modalityID,modalityDriverID,param )
```

Permite activar desde la lógica de la aplicación alguna capacidad de sintetizado provista por algún driver de modalidad. Recibe un identificador de modalidad (*modalityID*), un identificador de driver de modalidad (*modalityDriverID*) y parámetros (*params*) opcionales para la función de sintetizado.

3.2.6 *Componente de Driver de Modalidad*

Brinda una interfaz para el desarrollo de diferentes comportamientos sobre una modalidad. Esto permite que la plataforma sea agnóstica a una determinada modalidad particular. A través del uso de señales, el desarrollador de driver de modalidad, podrá escribir la lógica necesaria para su dispositivo de modalidad, ya sea un *reconocedor*, *sintetizador* o de ambos tipos; y conectar esa lógica de forma uniforme a la plataforma.

Constructor:

```
Gyes::ModalityDriver()
```

Genera una instancia que contiene las diferentes señales para conectarse con el cliente y así con la plataforma.

Principales Métodos:

```
modalityDriver::on( signal, callbackFn )
```

Permite al desarrollador de driver de modalidad, conectar de forma clara una señal de sintetizado (*synthetized*) o actualización (*updated*) con funcionalidad propia del driver.

```
modalityDriver::fire( signal, data )
```

Permite disparar una señal de reconocimiento (*recognized*), acompañado de datos producidos por la modalidad. Esta información sera distribuida mediante la plataforma.

3.2.7 *Descripción del Protocolo*

Se definió un protocolo de comunicación entre la plataforma y los múltiples clientes *Gyes*. El mismo permite expresar los conceptos definidos durante el trabajo. Estos son: modalidades, señales e interpretaciones. Se utilizo la notación JSON para definirlo.

A través del uso del protocolo podrían crearse clientes en otros lenguajes, que serian capaces de interactuar de igual forma con la plataforma que el cliente Gyes aquí propuesto (desarrollado en JavaScript).

A continuación se detallan los eventos que conforman el protocolo en la versión 1.

ID	PARÁMETROS:TIPO
plusultra::authenticate	appKey:string
plusultra::welcome	- - -
plusultra::new_modality	modality:object
plusultra::broadcast_new_modality	modality:object
plusultra::modality_signal	signal:object
plusultra::broadcast_modality_signal	signal:object
plusultra::interpretation	interpretation:object
plusultra::broadcast_interpretation	interpretation:object

3.3 CUESTIONES IMPORTANTES

Tener a los módulos de fusión y fisión *distribuidos*, es un acercamiento que plantea desafíos.

En primer lugar, cualquier aplicación distribuida gana mayor poder de computo de forma mucha mas "económica" ante una variante centralizada. Luego, la decisión de hacer esta solución *distribuida*, esta relacionada directamente con la arquitectura de bajo nivel del ambiente, es decir, la web. Por lo tanto, el acercamiento *per se* no fue forzado, esto dejo lugar a poder ver claramente como podían desarrollarse los aspectos propios del problema, i. e.los diferentes componentes que hacen a una aplicación multimodal. Para ello, se partió de la definición clásica de Dumas et al. [12].

Allí se identificaron los principales componentes a exportar dentro de esta solución. En particular se descartaron, o no se implementaron directamente, el sistema de administración de dialogo (*Dialog Management*) y el sistema de manejo de contexto/historia del usuario (*Context User Model History*). La decisión se basa en que estos componentes pueden ser mantenidos indirectamente en la lógica de la aplicación web, de acuerdo a la necesidad del desarrollador; además la ausencia de estos componentes de forma directa, facilita la transición hacia un modelo distribuido ya que presentan características de aplicación centralizada, i. e. no hace necesidad de un servicio externo manteniendo esta información.

De todas formas, se mantiene abierto el análisis, específicamente al componente de administración de dialogo, si se detecta alguna manera concreta en la que pueda mejorar la calidad de uso de la herramienta en general.

Si bien, un sistema multimodal con componentes de fusión y fisión distribuidos, puede generar demasiados mensajes y estos pueden afectar su performance general, la solución propuesta esta diseñada para ser escalable, añadiendo mas instancias de *Plusultra* que a su vez puedan manejar mas volumen de mensajes en el tiempo.

Uno de los aspectos que inicialmente se agrego como distribuido y luego de algunas pruebas se determino que sea mantenido localmente, fue el componente de modalidad. En un principio, al agregar una modalidad a la plataforma, mediante el cliente *Gyes*, la misma era transmitida a todos los demás clientes web. Esto generaba un problema al momento de disparar una interpretación, ya que la misma podía tener asociada una capacidad de sintetizado y si se intentaba activar dicha capacidad en la modalidad recibida por la plataforma, podría ocurrir un error o en un caso silencioso, un gasto de mensajería en vano. La clave de la plataforma se encuentra en los mensajes que distribuye, estos son: señales de modalidad e interpretaciones (conjunto de señales agrupadas bajo algún valor lógico). Cualquier otro mensaje a ser distribuido puede añadir valor agregado con un costo que debe ser tenido en cuenta porque añade mas mensajería a una plataforma en tiempo real, lo cual puede repercutir directamente en su performance.

3.4 RESUMEN DEL CAPÍTULO

En el presente capítulo se termino de introducir a los principales actores de la arquitectura de la solución propuesta. En particular, se han descrito aquí a aquellos relacionados a los

puntos de entrada/salida del sistema. Estos componentes, en su mayoría, tienen algún tipo de relación con la lógica de la aplicación web donde sean usados; o con la modalidad que se desee integrar al sistema.

Particularmente, se ha incluido el módulo multiuso *Gyes*, el cual representa una solución tanto para el browser como *standalone*. También se agrega la primer version del protocolo de comunicación entre el cliente y la plataforma, con el fin de dejar abierta una puerta para el desarrollo de otros clientes en diferentes lenguajes. Finalmente se destacan algunos puntos característicos de esta solución distribuida. Mas investigación en este punto puede ocurrir en el futuro, con el desarrollo de mas y variadas aplicaciones que consuman esta plataforma.

Parte III

DESARROLLO DE UNA APLICACIÓN MULTIMODAL

En esta parte se considera como funcionan algunos conceptos introducidos dentro del ámbito de una aplicación web. Para esto se analiza la calificación de aplicación web que existe tanto en la academia así como en la industria.

Se mostrará también el desarrollo de una aplicación web multimodal usando la plataforma para concretar una validación por implementación.

4

EXTENSIÓN DE UNA APLICACIÓN RIA

El concepto que engloba a una aplicación web como las que se referencian en este trabajo, es el de *Rich Internet Application* (RIA para resumir). Estas aplicaciones poseen un conjunto de características que las distinguen de los "sitios web" o "paginas web" y de las aplicaciones de escritorio.

La industria ha evolucionado en los últimos años hacia aplicaciones que pueden catalogarse como RIAs; aplicaciones web con una gran cantidad de lógica de negocio del lado del cliente y con alta atención en la experiencia del usuario. Algunos ejemplos son, Facebook, Gmail, Twitter, dentro del aplicaciones sociales; Rdio, Beats, como aplicaciones multimedia; BBC y NYTimes entre aplicaciones informativas. Estas compañías han impulsado diferentes técnicas y herramientas para el desarrollo de aplicaciones web modernas.

Esta información será usada en este capítulo como contexto para entender la forma de encastrar la plataforma propuesta con las aplicaciones web ricas.

4.1 APLICACIONES RIA

Las denominadas *aplicaciones web ricas* o RIAs se ubican, en una posible linea de tiempo histórica, como paso siguiente a los sitios web o paginas web pertenecientes a la *web 1.0*. Son parte del cambio que implicó a la denominada *web 2.0* y fueron evolucionando buscando algunos objetivos determinados que esta tendencia demandaba y que las distinguía de las aplicaciones web pertenecientes a la era "1.0", según Farrell and Nezlek [13].

En este marco histórico, el desarrollo busca centrarse en el usuario a través de la creación de experiencias e interacción mas ricas. La arquitectura de las aplicaciones *web 1.0* estaba demasiado limitada por el paradigma cliente-servidor, donde el cliente era un simple consumidor de cada pagina de hipertexto que el servidor le generaba, esto presenta un sincronismo duro que afectaba fuertemente las capacidades de interacción.

En Duhl [9], se definen los siguientes problemas tradicionales de las aplicaciones web pertenecientes a la época "1.0":

- Procesos complejos.

- Dificultad en en el acceso y tratamiento de datos (ausencia de capacidades exploratorias).
- Ausencia de capacidades de configuración y previsualización de objetos.
- Bajo feedback (altamente fragmentado).

De acuerdo a Fraternali et al. [15], las aplicaciones RIA representan una solución a estas limitaciones y/o problemas, ya que permiten construir aplicaciones con las siguientes características:

- Posibilidad de mantener datos en ambos extremos (cliente, servidor).
- Esquema de lógica de negocios distribuida entre cliente servidor, con la posibilidad de balancear la carga completamente servidor, completamente cliente o mixto.
- Igualdad en la capacidad de iniciar la conversación, tanto por parte del cliente como del servidor.

Estas capacidades permiten enriquecer la experiencia de uso de la aplicación web, haciéndola similar o incluso mejor que la contra-parte de escritorio. Particularmente, se destaca las mejoras que generan las RIA sobre el ultimo punto, el feedback. Estas aplicaciones se basan en el asincronismo en la carga y actualización de componentes, permitiendo así "independizarlos" en la información que representan mientras siguen siendo partes de un aplicativo superior. Por ejemplo, esto puede notarse en cualquier aplicación web que permita elegir qué elementos cargar de una lista sin refrescar toda la página y a la vez permitiendo interactuar con este, recientemente agregado, componente. Este estilo de aplicación es conocido como *Single Page Application* (SPA) o aplicación de una sola pagina.

Finalmente, es menester tener en cuenta que las características previamente listadas han mejorado notablemente en los últimos años. Un ejemplo de esto es el nuevo estándar HTML5, que no representa una tecnología particular si no mas bien un conjunto de tecnologías (CSS, HTML y JS). Estas herramientas en conjunto dan muestra de progresos importantes en la presentación, estructura y lógica de la aplicación web que se va a ejecutar del lado del cliente.

4.2 CONEXIÓN CON RIA

La definición de una aplicación rica (RIA) no esta ligada a un lenguaje en particular, por lo tanto pueden existir diferentes

implementaciones de las mismas en el mercado, sin embargo, de acuerdo al trabajo de Bozzon et al. [4] es posible clasificarlas en cuatro tipos:

- *Scripting-based* o basadas en scripting; estas aplicaciones son de las mas populares y están compuestas en su mayoría de JavaScript y técnicas como AJAX.
- *Plugin-based* o basadas en plugines; este tipo de aplicación se apoya en alguna plataforma y ambiente particular, e. g. Flash, Flex.
- *Browser-based* o basadas en algún browser; aplicaciones de este estilo corren dentro del ambiente de algún browser especifico, usualmente como extensiones del mismo y aprovechan a su vez APIs que éste define.
- *Web-based desktop technologies*; son aplicaciones que pueden ser descargadas en primer medida desde la web, pero tienen requerimientos extras particulares (i. e. dependencias) y corren fuera del browser.

La solución propuesta en este trabajo de tesis, apunta a operar con aplicaciones RIA del tipo *scripting-based*, ya que se han convertido en las mas populares y con mas soport *open-source*, aunque hay que destacar que la plataforma es agnóstica a la caracterización de la aplicación, esto implica que no posee requerimientos fuertes sobre donde será usada. Habiendo mencionado esto, es probable que sea necesario el desarrollo o uso de algún *wrapper* o adaptador especifico para poder correr en un ambiente determinado como puede ser el de una solución estilo *plugin-based*, aunque este desarrollo extra debería ser mínimo y acotado.

4.2.1 *Conexión con RIA y MDD*

El desarrollo dirigido por modelos o *model-driven development* (resumido como MDD), es una técnica bastante popular para el desarrollo de RIAs. Uno de los objetivos en el uso de MDD es minimizar y controlar la complejidad inherente que posee una aplicación web rica.

Muchas metodologías se han propuesto para modelar RIAs [43] [34], la mayoría extiende estrategias de modelado para aplicaciones de la era "1.0". Esto ha generado demasiada fragmentación que dejo en evidencia, al menos, las dificultades para modelar aplicaciones de este estilo.

WebML[29] es una de estas técnicas, que recientemente ha sido reformateada en el nuevo estándar del OMG [28], como IFML [30] o *Interaction Flow Modeling Language*.

Si bien la plataforma *Plusultra*, como ya se ha mencionado, es agnóstica al tipo de RIA desarrollada, ésta añade capacidades de interacción nuevas al conjunto soportado por las aplicaciones web. Estas nuevas capacidades de interacción pueden ser modeladas como *eventos* dentro del sistema. Es decir, estas formas de interacción pueden modelarse, dentro de una metodología de desarrollo basado en modelos, usando a su vez IFML para contemplar la interacción; como nuevos *eventos* añadidos al contexto de la aplicación.

4.2.2 *Conexión con RIA y MDD: Ejemplo*

Si bien no es el objetivo principal del presente trabajo explayarse sobre una metodología de desarrollo particular, parece importante destacar al menos con un pequeño y concreto ejemplo la posibilidad de utilizar un estándar para el modelado de la aplicación desde el *front-end*.

Para ello, se ha considerado la metodología IFML para modelar un posible flujo de interacción de la aplicación ejemplo, *Shapes*. En la figura 10, se muestra el modelado de acciones relacionadas a una fisión. En este momento, el sistema ha detectado que se ha producido una fisión , la cual es replicada a través de *Gyes* y *Plusultra* y se han ejecutado las acciones correspondientes asociadas a los eventos, en este caso, de interpretación y fisión.

4.3 APLICACIONES WEB EN LA INDUSTRIA

La industria ha sido participe en la mencionada "revolución 2.0". Negocios como *amazon* crecen considerablemente en esta época y generan necesidades nuevas, lo que motiva a la competencia a imitarlas o superarlas. Con el objetivo de mejorar cualitativamente las aplicaciones desarrolladas y las nuevas por crear, la industria comenzó a generar sus propias herramientas, muchas de ellas *open source*, lo que permitió que otros las usaran, mejorando de esta manera la calidad general del ecosistema.

En este contexto es donde surgen librerías troncales como Backbone.js [22], en el 2010, la cual esta desarrollada íntegramente en JavaScript y ofrece directamente al desarrollador abs-

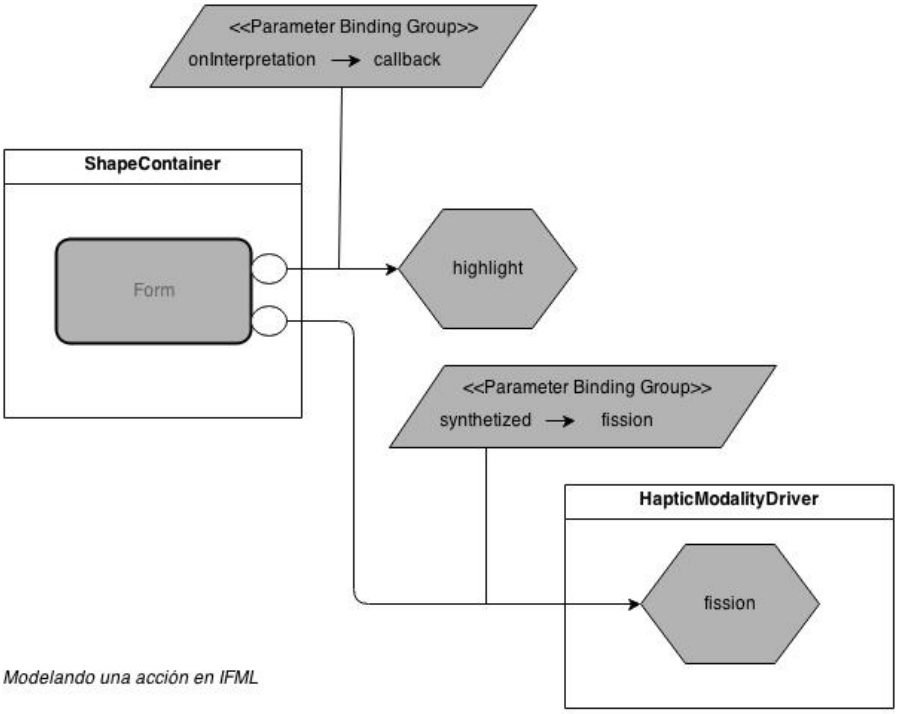

Figura 10: Modelado de acciones en Shapes App.

tracciones para modelos, vistas y controladores desde el lado del cliente, asumiendo del lado del servidor, una API REST. Una librería de este estilo, con un conjunto de dependencias mínimo colaboro directamente con el desarrollo de aplicaciones web ricas, con gran parte de lógica del lado del cliente, capaces de manejar y disparar *in-situ* eventos generados por la misma aplicación o por algún *third-party*. De esta forma se produce un traslado masivo del patrón MVC relacionado tradicionalmente al *backend*, hacia el *frontend*.

Unos años mas tarde, aparecen nuevas herramientas mas robustas; del tipo *frameworks*, nuevamente desarrolladas en JavaScript. El objetivo de estos frameworks es el mismo, ayudar al desarrollador a generar, mantener y permitir escalar una aplicación "del lado del cliente", usualmente siguiendo el estilo de backbone.js, consumiendo datos desde algún servicio que se encuentre corriendo del lado del servidor.

Entre estos frameworks, se pueden destacar Angular.js de [19] y Ember.js [1]. Nuevamente, estas herramientas son *open source* y tienen gran soporte por parte de sus respectivas comunidades, lo que permite retro-alimentar el ecosistema de aplicaciones ricas, haciéndolas perdurar en el tiempo.

Dentro de la industria no hay que dejar de lado a los desarrolladores de navegadores. Grandes empresas como Google, Apple y la fundación Mozilla contribuyen a la mejora de este

escenario a través del desarrollo de navegadores que implementen los estándares de la W3C. Esto posibilita el acceso a nuevas capacidades, como por ejemplo la API para WebRTC, la tecnología que permite acceder a recursos tales como la cámara o el micrófono y compartirlos por internet. Hay que tener en cuenta la relación existente entre la implementación de los estándares y estas grandes compañías, ya que usualmente desarrolladores pertenecientes a las mismas participan en los comités donde se definen estas nuevas APIs.

4.4 CONEXIÓN CON LA INDUSTRIA

Similar al caso anterior, la relación entre las aplicaciones web modernas que se pueden desarrollar utilizando las herramientas que brinda la industria y la plataforma propuesta no implica una dependencia entre ambas.

La plataforma puede ser consumida desde el contexto de una aplicación web a través del uso de una librería provista, *Gyes*. Dicha librería, encapsula una API de acceso a la plataforma y es vista desde la aplicación como una dependencia mas.

La definición actual JavaScript (ES2015) no soporta un concepto de modulo nativo, para suplir esta carencia la industria ha desarrollado varias técnicas que involucran wrappers *que permiten modularizar y cargar bajo demanda. Estas técnicas son CommonJS (CJS), AMD y UMD. CommonJS ha sido implementado en Node.js, AMD fue una de las primeras especificaciones para componentizar desde el lado del cliente. CJS y AMD no son compatibles entre si; es por esto que nace UMD, un patrón que envuelve tanto código CJS y/o AMD y lo hace interoperable.*

Gyes puede esta escrita usando el formato de módulo CommonJS [6] compatible con Node.js. Esto permitió modularizar las distintas dependencias internas de la aplicación. Ya que este formato no es soportado por los navegadores, se utiliza un proceso de construcción que genera un archivo único que representa la dependencia antes mencionada y que es compatible con los navegadores, gracias al uso del patrón UMD[1]. A su vez, esta dependencia puede encapsularse dentro del formato AMD para ser gestionada mediante otras herramientas como requirejs [14], por ejemplo.

En un futuro seria interesante portar *Gyes* al sistema de módulos estándar propuesto para JavaScript, pero el mismo hasta el momento no se encuentra soportado por los navegadores corrientes.

4.5 RESUMEN DEL CAPÍTULO

Se ha especificado el contexto de aplicación web en el cual la plataforma puede insertarse.

Se ha puesto énfasis en analizar este contexto desde dos aristas, el ámbito académico y la industria. De esta forma es mas claro ver como se puede utilizar la herramienta desarrollada y

1 Para mas información ver: https://github.com/umdjs/umd

como encaja en el escenario actual y en un entorno de producción.

En el siguiente capítulo se mostrará de forma integral el uso de la plataforma por medio del desarrollo de una aplicación web que servirá también como modo de *validación por implementación* para la plataforma.

5

DESARROLLO DE UNA APLICACIÓN WEB MULTIMODAL

Los desarrollos de software en la web han avanzado desde sitios estáticos fuertemente orientados al hipertexto, en sus primeros años, a las *aplicaciones* web actuales, similares en robustez y capacidad a las homólogas de escritorio pero con nuevas capacidades típicas del medio, como pueden ser aspectos de programación distribuida o tiempo real. Este avance no ha sido excluyente si no que se han ido añadiendo nuevas capas de funcionalidades.

La aplicación web introducida en este capítulo tiene el objeto de actuar como herramienta de demostración y validación de la plataforma desarrollada, *Plusultra*. Aunque también añade, a través de un conjunto de tecnologías, una nueva capa de funcionalidad multimodal al dominio de las aplicaciones web.

5.1 INTRODUCCIÓN

Dentro del ámbito de una aplicación web, numerosos eventos pueden ocurrir, algunos generados por el usuario otros por la lógica de la aplicación (tanto cliente como servidor). Estos eventos generan en la mayoría de los casos que se dispare una acción y son usualmente acompañados por una animación a modo de feedback visual para darle fuerza al foco de la acción, ayudando a que el usuario mantenga en todo momento una "imagen" clara del estado del sistema, i. e. que el usuario sepa qué ha ocurrido y que está ocurriendo. Pero esta no es la única forma de indicarle y/o darle mas feedback al usuario, existen diferentes vías de comunicación que pueden ser utilizadas con el simple e importante objetivo de brindar mas información y contexto E. g. un estado de error es comúnmente transmitido usando el color rojo; es posible agregar otro canal y estimular otro sensor, como puede ser la piel a través de una corta y clara vibración (similar a la que estamos recibimos de los smartphones), logrando así enriquecer la comunicación, cargando la autopista de información usuario-computador.

A través del uso de la plataforma *Plusultra* añadiremos un conjunto de nuevas modalidades al sistema, que generaran nuevos y definidos eventos para que puedan ser capturados dentro del ambiente de una aplicación web. Los usuarios de la misma

se beneficiaran de estas nuevas capacidades de interacción, enriqueciendo así a una simple aplicación desarrollada a modo de probar las ventajas que provee el uso de la plataforma.

5.1.1 *Objetivo*

Desarrollar una aplicación web rica, utilizando un conjunto de dependencias acotado con el fin de hacer énfasis en el uso de la plataforma *Plusultra* y en el agregado de nuevas modalidades dentro de la aplicación.

La aplicación, si bien es de alcance limitado por ser una demostración, tiene el objeto de funcionar dentro del área de entretenimiento. Representa al clásico juego de niños de "formas", donde se busca estimular el reconocimiento de patrones. Al hacer encajar las piezas en su lugar, se consigue una retroalimentación que puede verse incluso acompañado de estímulos externos (por parte de los padres, por ejemplo), premiando el éxito, es decir, esta acción de encastrar una pieza en su lugar; todo esto derivando en el conocimiento (y/o refuerzo) de nuevas formas para el niño.

Aquí se traslada este juego a una versión digital del mismo. A través del uso de la plataforma *Plusultra* y un conjunto acotado de modalidades, se extenderán las capacidades de interacción con el fin de generar mas estímulos en el usuario y así incrementar el *feedback* total de la aplicación.

5.1.2 *Requerimientos*

Mas allá de los requerimientos iniciales para desarrollar cualquier aplicación, la creación de una aplicación multimodal implica conocer cuáles son y cómo operan los dispositivos de modalidad que vayan a ser usados.

Para el desarrollo puntual de esta aplicación fueron creados dos *drivers* de modalidad para conectar dos nuevas modalidades, una háptica y otra aero-gestual. Estos controladores ofrecen un conjunto de eventos y en algunos casos abren un camino bidireccional entre modalidad y aplicación, es decir, permiten la modificación de parámetros de modalidad en tiempo real.

Mas adelante se muestran los principales detalles para la creación de un driver de modalidad genérico.

5.1.3 *Módulos Principales*

A continuación se describen los principales módulos que componen a la aplicación *shapes*:

- `main.js`; representa el contenedor principal de la aplicación, unifica las principales dependencias y dispara el inicio de la aplicación web junto con el panel visor de información de desarrollo.
- `visor.js`; contiene la lógica necesaria para brindar información de estado útil para depurar la aplicación durante el desarrollo.
- `shapes.js`; coordina los principales eventos de la aplicación, estos pueden ser eventos internos disparados por la aplicación o externos, despachados por alguna de las modalidades. Estos últimos, cuando ocurren son distribuidos a lo largo de todos los clientes conectados a la plataforma.
- `interactive.js`; se encarga de administrar de manera uniforme los eventos de drag & drop que ocurran en la aplicación.
- `gyes.js`; es el módulo cliente oficial para conectarse con *Plusultra*. Luego de obtener la llave de acceso a la plataforma, el desarrollador puede comenzar a utilizar un determinado número de instancias de la plataforma. El módulo brinda los principales eventos a los cuales la aplicación puede conectarse y actuar.
- `HapticMD`; representa al controlador de modalidad háptico. El mismo esta desarrollado para aprovechar las capacidades de los dispositivos móviles modernos en el ámbito de una aplicación web rica. Provee eventos para definir interacción de entrada a través de gestos táctiles y de salida, a través de feedback vibro-táctiles.
- `AirPointerMD`; es el controlador que administra la modalidad aero-gestual, i. e.gestos aéreos, utilizando las manos. Este módulo utiliza como dispositivo de hardware al artefacto Leap, lo que permite conseguir información variada sobre manos y dedos sin necesidad de hardware extra *en el* usuario. Este controlador brinda eventos de entrada sobre la posición de alguno de los dedos, el cual será usado como puntero para conseguir una manipulación básica directa sobre los elementos de la aplicación.

En la siguiente figura 11, se pueden observar a simple vista los principales módulos de la aplicación *shapes*.

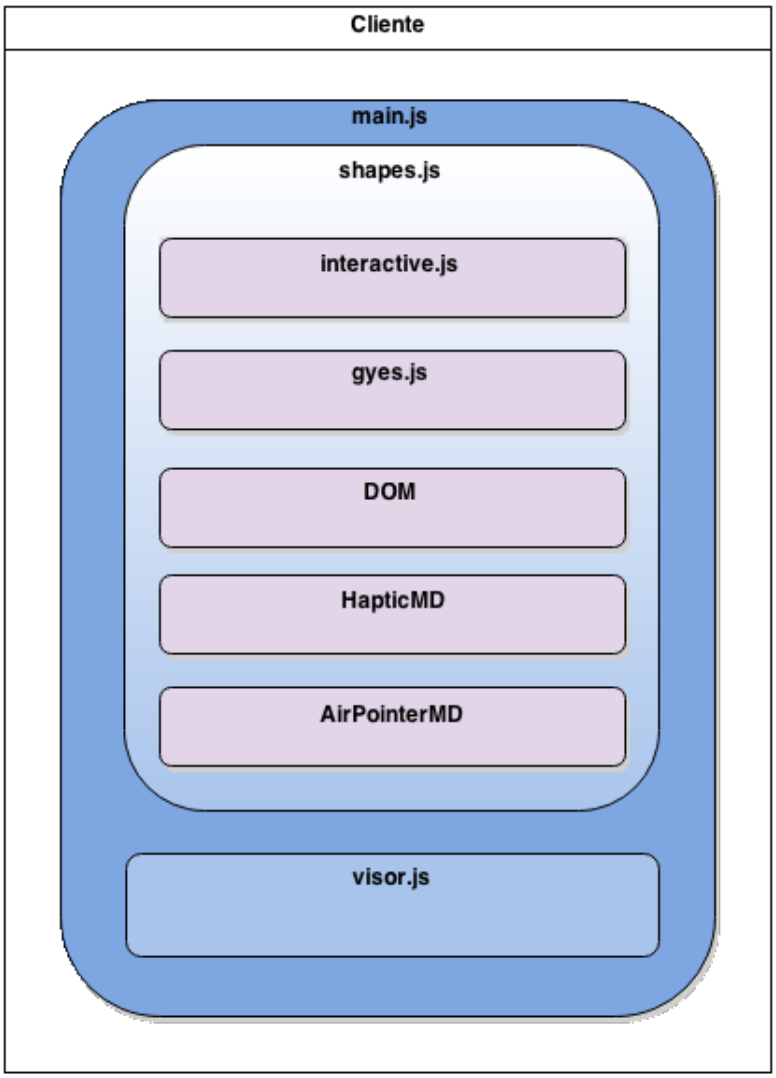

Figura 11: Módulos de la aplicación web Shapes

Los módulos desarrollados para esta aplicación se realizaron usando el patrón *module*, como puede verse en Osmani [31]. A través del uso del mismo se consigue una división de responsabilidades y conocimiento, distinguiendo partes publicas (API) de privadas (implementación). Además permite aplicar una simple pero solida capacidad de manejo de dependencias internas.

5.1.4 *Conectando a la Plataforma*

Con el fin de extender el conjunto de modalidades de una aplicación web, se propone la utilización de *Plusultra*. La misma provee un servicio, en tiempo-real, de comunicación con dispositivos de modalidad. En el caso concreto de la aplicación `Shapes`, se han utilizado dos dispositivos de modalidad, el Leap Motion y los controladores táctiles de los dispositivos móviles. Agregar estas nuevas modalidades aumenta la capacidad de interacción con la aplicación desarrollada. Estas nuevas capacidades pueden ser aprovechadas por el desarrollador de la aplicación web mediante el consumo de los eventos que las mismas generan y que son transmitidos mediante la plataforma.

En la figura 12 puede verse la relación entre los componentes arquitectonicos de la aplicación desarrollada:

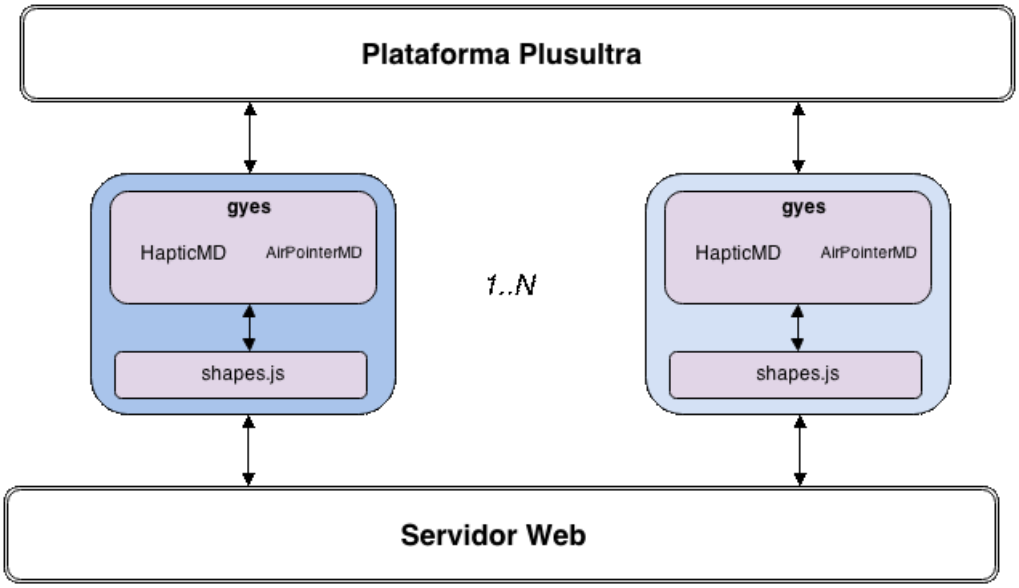

Figura 12: Vista de la arquitectura de la aplicación Shapes

Para conectarse con la plataforma desde el contexto de una aplicación web el desarrollador debe incluir como una dependencia JavaScript, a su entorno de desarrollo, al módulo `gyes`. Este módulo brinda una interfaz de acceso a la plataforma.

Para conectarse es necesario contar con una llave de acceso y ejecutar el llamado a conexión usando el método `authenticate`. Este sistema de autenticación es simple y busca emular un escenario "real"; debería ser mejorado de forma acorde antes de ser llevado a un entorno de producción.

Luego de la autenticación a la plataforma, el desarrollador puede comenzar a utilizar los drivers de modalidad que considere necesarios. Para esta aplicación se utilizan, un driver aerogestual, denominado *AirPointerMD* y otro, *HapticMD*, usado como controlador háptico. A través del módulo `gyes` el desarrollador instancia las modalidades dentro de la aplicación web, por ejemplo:

```
// Construye una nueva modalidad háptica
var hapticMod = new gyes.Modality( 'webHaptic', 'both', {} );
// Configura algunas opciones del driver como por ej, qu\'{e} gestos
    hápticos capturar
var driverOptions = {
  'hapticEvents': [ 'touch, hold' ],
  'element': doc.getElementById('shapes')
};
// Construye el driver de modalidad
var hapticDriver = new HapticMD( driverOptions );
// Conecta la modalidad con el driver
hapticMod.use( hapticDriver );
// Inserta la modalidad en la plataforma.
_client.addModality( app_key, hapticMod );
```

Luego de instanciar y conectar las modalidades deben definirse las interpretaciones necesarias, i. e. una estructura que da valor semántico a un conjunto de eventos (posiblemente generados por las modalidades) que ocurren en un intervalo de tiempo acotado. Entonces, para que una interpretación ocurra deben generarse determinados eventos en un instante de tiempo, la captura de estos eventos produce una fusión y es tratada por el motor de fusión distribuida del módulo `gyes`. Un ejemplo donde se muestra como instanciar el motor de fusión es el siguiente:

```
// El motor de fusión recibe interpretaciones como entrada.
_gestureInterpretation = new gyes.Interpretation( ['fingerover', '
    hold'] );
// Algunas intepretaciones pueden producir sintetizado de eventos.
_gestureInterpretation.canSynthetize( hapticMod.name, hapticDriver.
    getID(), 2000 );

// Construcción de un motor de fusión.
_fusion = new gyes.Fusion( {'verbose':true} );
// Agregando interpretaciones al motor de fusión.
_fusion.fuse( _gestureInterpretation );
```

Finalmente, luego de crear las interpretaciones necesarias junto al motor de fusión distribuida, lo único que queda por hacer es instanciar el sistema de fisión.

A través de este sistema, el desarrollador puede controlar programaticamente qué hacer cuando es detectada alguna interpretación.

El sistema de fisión, permite conectar una función para que sea llamada de manera asincrónica luego de la ocurrencia de una interpretación, como puede verse a continuación:

```
1    // Construye el sistema de fisión
     var _fission = new gyes.Fission();

     // Conecta una interpretación con una función anónima.
     _fission.on( _gestureInterpretation.getName(), function(data){
6      console.log( 'A new interpretation happened: ', data );
       gestName.textContent = '';
       gestElem.classList.remove( 'highlight' );
       gestDataText.classList.remove( 'invisible' );
       gestDataText.textContent = 'FISION';
11     gestData.classList.add( 'highlight' );
       setTimeout(function(){
         gestDataText.classList.add( 'invisible' );
         gestData.classList.remove( 'highlight' );
         gestDataText.textContent = '';
16     }, 2000);
     });
```

De esta forma se han mostrado los principales puntos de conexión necesarios para utilizar la plataforma *Plusultra* en su primera versión.

5.2 CONSTRUCCIÓN DE UN DRIVER DE MODALIDAD

Para el desarrollo de una aplicación web multimodal es necesario contar con drivers o controladores que den soporte a los dispositivos de modalidad que vayan a ser usados. La plataforma *Plusultra* es agnóstica respecto al dispositivo de modalidad usado, brindando a través del módulo *gyes* una interfaz para conectar un nuevo dispositivo al sistema. Un posible trabajo a futuro y aporte a la plataforma sería la creación de un catálogo online de todos los drivers de modalidad disponibles.

El desarrollo de un nuevo driver de modalidad es una tarea relativamente simple, aunque esto puede variar entre dispositivos. El módulo *gyes* provee una interfaz para la creación de drivers en JavaScript, aunque es posible desarrollar un driver en otro lenguaje, simplemente siguiendo el protocolo (ver subsección protocolo 3.2.7), provisto por *Plusultra*. A continuación se muestran los principales métodos a implementar para crear un driver de modalidad, la conexión del mismo a la plataforma se definió anteriormente.

```
// Incluir el canal de comunicación de gyes.
var ModalityDriverChannel = require( '../gyes/build/gyes' ).
    ModalityDriver;
3
// Declaramos el constructor del driver de modalidad. Ej:
```

```
module.exports = HapticModalityDriver;

// El constructor generado hereda los eventos a implementar del canal de
    drivers de modalidad.
inherits( HapticModalityDriver, ModalityDriverChannel );

// Si el dispositivo de modalidad puede sintetizar datos, entonces es
    posible conectar el evento de sintesis con alguna función interna.
    Ejemplo del driver háptico:
this.on( 'synthetized', this.fission.bind(this) );

// Si solo necesitamos disparar eventos de "entrada":
this.fire( 'recognized', {'gesture':data.type} );
```

Por lo tanto existen dos eventos importantes a tener en cuenta en la realización de un driver de modalidad; el evento para disparar algo "sensado" por el dispositivo, etiquetado como *recognized* y el evento para recibir datos de parte de la aplicación y sintetizarlos, denominado *synthetized*. De esta forma es posible crear un nuevo driver para conectar un nuevo dispositivo al sistema y usarlo en el ámbito de una aplicación web. Es importante destacar que un componente creado de esta manera puede ser compartido y re-utilizado, la única dependencia es contar con el hardware indicado, es decir aquel para el cual fue desarrollado dicho driver.

Esto permite pensar en la posibilidad de contar con un catalogo de drivers de la plataforma.

5.3 COMPONENTES EN ACCIÓN

La aplicación Shapes es una prueba de concepto que busca poner en evidencia una interacción multimodal en el navegador. Sumado a esto, la plataforma posibilita la colaboración en tiempo real entre pares. Por lo tanto, es posible contar con mas de un usuario al mismo tiempo y generar entre los usuarios la interacción multimodal. Más aun y debido a la naturaleza distribuida del entorno, la plataforma propaga la señal generada en un evento multimodal de fisión que podrá ser percibido por todos los clientes (no solo aquellos involucrados en la generación del evento). Esta señal de fisión será recibida en los clientes, los cuales representan un conjunto de hardware heterogéneo, implicando distintas capacidades de afectar al usuario; en el ejemplo aquí presentado, si el usuario esta utilizando la aplicación desde un dispositivo móvil podrá recibir una vibración particular como resultado de una acción multimodal.

En el siguiente gráfico 13 se sintetiza el flujo de comunicación habitual de una aplicación web multimodal.

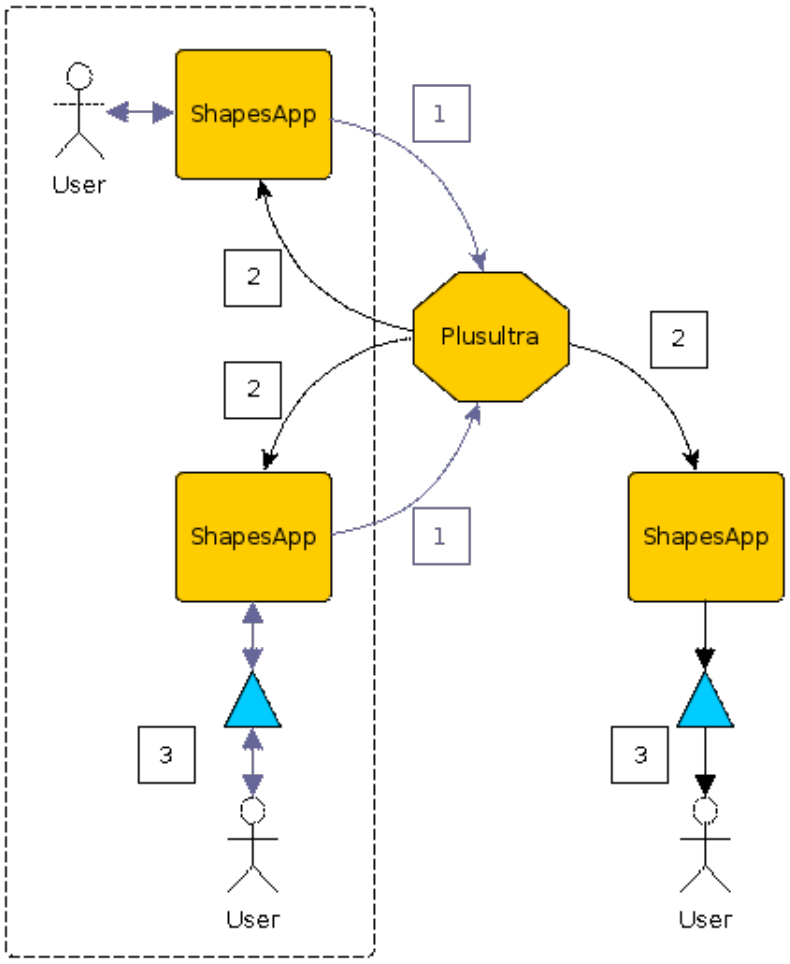

Figura 13: Flujo de comunicación.

Referencias:
-Triangulo: representa a un dispositivo de modalidad.

Flujo:

- *1)* Parte de un acción multimodal. Distintos clientes están disparando las partes que conforman a una interacción soportada en la aplicación, esto es por ejemplo, depositar una pieza en el lugar correspondiente sumado a seleccionar y mantener la selección de la pieza sobre el contenedor. Esto ocurre localmente y será distribuido por la plataforma.

- *2)* Se realiza la distribución de eventos generados en 1. Esto puede disparar en algunos clientes la interacción multimodal, mas específicamente, una fusión y luego una fisión. Estos eventos son tratados de forma independiente y son propagados por la plataforma.

- *3)* Si el proceso de fusión genera una fisión, la misma es propagada, esto puede desencadenar alguna acción en la modalidad conectada al cliente.

5.4 RESUMEN DEL CAPÍTULO

El principal objetivo de la plataforma *Plusultra* es agrandar de manera uniforme el conjunto de modalidades que soportan actualmente las aplicaciones web. Este conjunto es muy limitado, concentrado en las principales formas de comunicación con el ordenador, como lo son el mouse, el teclado y el feedback visual. En muy pocos casos una aplicación web va mas allá de eso y la principal razón puede estar ubicada en las dificultades propias de la tarea junto a la falta de información al respecto. Yendo un poco mas allá, imaginar el soporte a mas de una modalidad al mismo tiempo, parece difícil de visualizar y por ende de llevar a cabo.

Superar estas cuestiones fueron las premisas constantes durante el desarrollo de la plataforma y a través de la construcción de la aplicación ejemplo `Shapes` se mostró como es posible crear una aplicación web multimodal con un conjunto acotado y mínimo de dependencias externas. Si bien existen cuestiones a mejorar, como la simplificación de la API del módulo `gyes` o las mejoras en el sistema de autenticación con la plataforma, la solución aquí propuesta ha funcionado como un recurso exitoso y concreto al problema de desarrollar una aplicación web multimodal.

Parte IV

CONCLUSIONES Y TRABAJO FUTURO

El desarrollo de aplicaciones multimodales parece ser el camino a seguir por una parte de la industria. Esto se refleja en el avance en los dispositivos móviles y en diversos proyectos de hardware abierto los cuales "oxigenan" las vías de comunicación aun mas.

Profundizar y mantener en órbita estos conceptos se vuelve entonces una estrategia adecuada.

6

CONCLUSIONES & TRABAJO FUTURO

Conceptos como *Wearable Computing*, *Sensors Fusion* o incluso *Pervasive Computing* han vuelto a ganar impulso recientemente. Utilizando tecnología como placas arduino o similares, implementar pruebas de concepto o prototipos de proyectos de las áreas antes mencionadas, se ha vuelvo mas simple, ya no es necesario contar con conocimientos avanzados de electrónica o saber programar un PIC; estas ventajas reducen las barreras para continuar experimentando en dichas áreas.

Es en este contexto de computación ubicua o "fuertemente" móvil es donde es interesante pensar en usar conceptos de desarrollo de aplicaciones multimodales, aprovechando el posible enriquecimiento en las formas de comunicación que estos nuevos "caminos" ofrecen para aumentar la experiencia del usuario dentro de la aplicación o incluso para introducir nuevos usuarios gracias a los beneficios conseguidos por la expansión en el conjunto de interacción del sistema.

Sharon Oviatt escribió en 1999 un trabajo donde ataca 10 mitos que suelen generarse alrededor de las aplicaciones multimodales [32]. Este trabajo resulta aun hoy en dia muy relevante y es una lectura recomendada para todo aquel que se vea frente a la necesidad de comenzar a desarrollar una aplicación multimodal.

De acuerdo a Kortum [25], los sistemas multimodales dan libertad a los usuarios de elegir como comunicarse con la aplicación, robustez para disminuir errores al tener mas de una modalidad para indicar una acción, e. g. el comando «*apuntar aquí*» en una aplicación de información geoespacial, puede ganar precisión utilizando un disparador mediante voz ("*apuntar aquí*"), sumado a un evento sobre una superficie táctil; especialmente si lo comparamos con una aplicación similar que solo cuente con una única modalidad (háptica o por voz) de interacción. Estos sistemas incluso pueden introducir ventajas de rendimiento para determinaras tareas (e. g. visual-espacial), aunque esto depende de la combinación de modalidades. Por ejemplo, en el caso de sistemas que combine gestos (mediante un lápiz óptico) con ordenes vía voz, resultados han mostrado que se obtiene una mejora del 10 % en los tiempos de terminación de la tarea, se reducen los errores críticos en un 36 % y una reducción del 50 % en problemas de fluidez espontanea, junto a mejoras en las construcción lingüísticas, según Oviatt and Kuhn [33].

6.1 CONCLUSIONES SOBRE EL DESARROLLO

Dentro del marco del presente trabajo de grado, se ha desarrollado una plataforma que permite aumentar el conjunto de

canales de interacción de una aplicación web, a través del soporte de interacciones multimodales. Estas nuevas capacidades de interacción pueden ser consumidas en conjunto o por separado, el desarrollador de la aplicación web es quien tiene el control sobre esto. Esta plataforma, en conjunto con el cliente y el protocolo desarrollados, han sido validadas mediante el desarrollo de una aplicación web que no solo permite su uso mediante los canales clásicos como el teclado y mouse, si no que también es posible interactuar usando el canal háptico y gestual-aéreo. Incluso en el canal háptico se explora la retroalimentación, mediante el uso de motores de vibración. Es decir, el usuario puede *sentir* nuestra aplicación. Todo esto sin perder las características clave de la web, como colaboración, fácil acceso y distribución.

Esta aplicación ha sido desarrollada con los fines de validar la plataforma propuesta y ayudar a mostrar las capacidades de los sistemas multimodales en la web. Por lo tanto constituye una simple demostración del alcance de la herramienta desarrollada. Con mas trabajo y recursos es perfectamente posible generar una experiencia mas rica que incluya, tal vez, a mas dispositivos y canales.

Teniendo esto en cuenta, los avances aquí propuestos resultan satisfactorios y cumplen con el punto establecido en un principio, extender las capacidades de interacción de las aplicaciones web. La experiencia en general ha resultado satisfactoria y de gran aprendizaje, por ejemplo, en el desarrollo de una aplicación multimodal, uno de los componentes clave es el motor de fusión, en este trabajo se ha desarrollado un nuevo tipo de moto, si bien es bastante simple, cuenta con una característica completamente novedosa, esta distribuido entre los clientes. A través de mas pruebas seria posible saber que tan buena o mala puede ser este nuevo rasgo, en la aplicación a modo de demostración desarrollada, este motor ha sido funcional permitiendo una detección rápida de eventos mientras disminuía la complejidad en el desarrollo del mismo. En la próxima sección se detallan algunas de estas cuestiones, que pueden extenderse como posibles trabajos a futuro.

6.2 TRABAJO FUTURO

El desarrollo de un sistema nuevo en un escenario donde antes no existía nada similar de acuerdo a lo visto en el capítulo 1 supone un desafió. Mas allá de la dificultad *per se* de llevar a cabo el desarrollo, existen numerosas decisiones que deben ser tomadas con el objetivo de llegar a destino, algunas pueden ser

supuestas con antelación y otras simplemente, no pueden serlo, mas aun cuando todo el equipo de trabajo esta constituido en una sola persona. Muchas de estas decisiones se encuentran detalladas en esta sección y pueden ser vistas como posibles mejoras a realizar, otras resultaron en nuevos problemas que no podían ser tratados en su momento porque hubiesen desviado la atención y el objetivo principal del trabajo, construir la plataforma y aumentar las capacidades de interacción de las aplicaciones web; a las cuales se les ha brindado una solución que puede no ser la mas apropiada o efectiva, pero que ha servido de manera empírica para cumplir el objetivo propuesto.

6.2.1 *Performance de los Motores de Fusión y Fisión Distribuidos*

Seria interesante conocer la performance de los motores de fusión y fisión desarrollados. Esto permitiría detectar problemas desconocidos o cuellos de botella de la plataforma. Además de brindar una forma de comparación con otras herramientas similares del mercado. Una posible técnica de medición de rendimiento podría ser la propuesta por Dumas et al. [11]. Durante el desarrollo del trabajo se priorizo cumplir con el objetivo de crear la plataforma, para luego de eso, considerar una etapa de medición de performance.

6.2.2 *Mejoras Especificas al Desarrollo del Motor de Fusión*

Durante la creación del motor de fusión, uno de los momentos clave, fue el desarrollo de la "ventana" de captura de eventos. La misma es configurable, en cuanto al tiempo que se encuentra *abierta* y fue construida usando algunas primitivas del lenguaje JavaScript. Construir un sistema de relojes propio, para controlar la apertura de la ventana seria ser una mejora notable al mantenimiento del sistema. También seria importante enriquecer el motor de fusión añadiendo soporte de prioridades entre modalidades, configurable por el desarrollador a su vez como el manejo de orden y no solo simple ocurrencia, de eventos.

6.2.3 *Mejoras de Usabilidad en la API*

Actualmente, la primer versión de la API del cliente es la que ha sido descripta en el capítulo 3, y vista en uso en el capítulo 5. La misma puede ser mejorada, ya sea añadiendo nuevos métodos que hagan mas simple la tarea del desarrollador o re-

moviendo o unificando operaciones existentes, con el mismo objetivo. Una manera de detectar estos puntos críticos es involucrando mas desarrolladores que consuman y utilicen la plataforma, teniendo en cuenta el feedback de los mismos en una nueva versión de la API.

6.2.4 *Consolidar a plusultra como PAAS*

La plataforma desarrollada tiene intenciones de funcionar como una plataforma como servicio (*PAAS*). Es decir, podría convertirse en un producto comercial que brinde la capacidad de desarrollo web multimodal ocultando cuestiones de hardware externas al problema que se desea resolver. Para conseguir esto, algunas cuestiones fueron tenidas en cuenta durante el desarrollo de este trabajo, pero en otras se decidió no llevarlas a cabo, e. g.capacidades de autenticación o un servicio web para indicar el desarrollo de una nueva aplicación multimodal.

6.2.5 *Crear Catalogo de Drivers de Modalidad*

Un aporte significativo para esta plataforma sería la creación de un catalogo de drivers de modalidad, ofrecido como un servicio web. De esta manera, cualquier persona que desee crear una nueva aplicación multimodal podría revisar primero el catalogo para saber si el hardware con el que cuenta tiene soporte. También serviría para impulsar y unificar el desarrollo de los mismos.

6.2.6 *Crear Aplicaciones Web usando la Plataforma*

Para detectar posibles problemas existentes que han permanecido en silencio o mejorar la usabilidad de la plataforma, desde el punto de vista de un posible equipo multimodal, es importante y beneficioso contar con un ecosistema saludable de aplicaciones y drivers. El equipo de desarrollo detrás de una nueva aplicación multimodal que use la plataforma puede generar una considerable cantidad de feedback que puede ser muy útil.

6.3 ENLACES A LOS REPOSITORIOS

A continuación se encuentra un listado de enlaces a los repositorios de los componentes de la plataforma:

- Plataforma plusultra. (github.com/dpaez/plusultra)

- Módulo Interfaz gyes. (github.com/dpaez/gyes)
- Driver de Modalidad háptico. (github.com/dpaez/HapticModalityDriver)
- Driver de Modalidad gestual-aéreo. (github.com/dpaez/AirPointerDriver)
- Aplicación Desarrollada usando la Plataforma. (github.com/dpaez/shapes_app)

Parte V

APÉNDICE

Hasta ahora hemos visto una serie de componentes, que agrupados, son capaces de dar soporte a interacciones multimodales en aplicaciones web. Pero nada se ha especificado sobre alguna modalidad concreta. Aquí se introducirá un primer acercamiento a dos modalidades, para las cuales se han desarrollado sus correspondientes drivers. Se trata de las modalidades háptica y aéreo-gestual.

A

MODALIDAD HÁPTICA

La modalidad háptica es la que permite la interacción con el órgano mas grande del cuerpo humano, la piel. Particularmente, el sentido del tacto. Este sentido, junto a la vista, permiten, por ejemplo, el rápido reconocimiento de escenarios, percibiendo profundidad y distancias estimadas con los elementos del ambiente así como también una referencia centrada en el usuario, i. e.desde mi lugar donde están los elementos. Una de las principales características del sentido del tacto, es que brinda una autopista directa al sistema nervioso, a través de su propio canal. Esto representa una ventaja con respecto a otros sentidos, como por ejemplo, vista y oído, ya que no requiere un procesamiento (reconocimiento de patrones) previo. Por otro lado, esto indica que la información obtenida por este canal es de mas bajo nivel o mas "cruda" y seguramente serán necesarias tareas de filtrado o adaptación de la entrada.

A.1 SENTIDOS INVOLUCRADOS

El principal sentido afectado por esta modalidad es el del tacto. A través de este sentido, se pueden identificar dos interfaces, una cutánea y otra cinestésica. La primera se relaciona directamente con la piel y sus diferentes terminales sensitivas, la segunda, en cambio, con músculos y tendones y representa información sobre la ubicación espacial de todo nuestro cuerpo en el ambiente. Ambos canales proporcionan diferente *feedback* al usuario y deberán ser capturados/sintetizados usando diferente hardware, i. e.display táctil vs. display cinestésico. Ambos pueden trabajar en conjunto y a diferencia de otros sentidos, el tacto, permite una comunicación bidireccional entre usuario y sistema, de esta forma, se nota claramente que mediante el uso de este canal es posible aumentar claramente la experiencia de usuario, ya que se aumenta el *feedback*. Una posible forma de distinguir estas interfaces, es en cuanto al caudal de información que intentemos enviar por el canal, i. e.si solo necesitamos transmitir una alerta para corregir una acción vs. si deseamos transmitir información espacial al usuario; la primera puede ser solucionada usando la interfaz táctil y la segunda la cinestésica.

Existen diferentes formas de estimular el sentido del tacto, los receptores hápticos del cuerpo humano son de tres tipos:

mecano-receptores (actúan de acuerdo a la fuerza o presión), termo-receptores (estimulados por la temperatura que reciben) y nociceptores (estimulados por el dolor). Los sistemas mecano-receptores son los mas populares, y se pueden clasificar en cuatro diferentes sensaciones: presión, tacto, vibración y cosquilleo. La distribución de estos sensores no es uniforme en todo el cuerpo, por lo que aplicar una mecano-estimulación impactara de manera diferente de acuerdo al lugar de acción. Es una cuestión a tener en cuenta antes de desarrollar cualquier tipo de aplicación multimodal que planee utilizar este sentido. En el trabajo de Hale and Stanney [21], se clasifican y recomiendan distintas interfaces hápticas, siendo una guía importante al momento de desarrollar una interfaz que consuma esta modalidad.

A.2 INTERFACES POPULARES

Algunas interfaces populares para esta modalidad son el dispositivo Falcon de Novint o PHANTOM, ambos de características similares, operan sobre la interfaz táctil brindando un punto de conocimiento al usuario, que mediante el uso del dispositivo puede enviar información aplicando presión y recibir un estimulo, mediante un sistema de motores de vibración. Dispositivos de este estilo, permiten reconocer texturas o ambientes a través de dicha estimulación viéndose limitados a un único punto de contacto, es decir no transmiten información masivamente a todo el cuerpo.

Figura 14: Dispositivo Falcon de la compañía Novint

Es interesante destacar también la ocurrencia de estimuladores hápticos en dispositivos de uso masivo como son los teléfonos móviles y también en los sistemas de entretenimiento, específicamente presentes en los joysticks desde hace bastante tiempo. La interfaz con la que estos dispositivos interactúan es táctil. La interacción se produce través de un sistema de estímulos dirigidos al usuario en forma de vibraciones que podrá sentir en su piel, la intención común es brindar algún tipo de alerta sobre alguna acción a realizar. Esta acción puede brindar soporte a otras intenciones disparadas con el fin de indicar la ocurrencia de un evento al usuario (i. e. como soporte a un alerta visual o auditivo que indica la terminación de una tarea asíncrona).

La presencia de esta modalidad en dispositivos masivos fue una de las características que motivo a que sea una de las primeras modalidades en ser soportadas por la plataforma mediante el desarrollo de un driver especifico. Existen mas implementaciones de interfaces hápticas que las que se mencionan aquí, para mas información se recomienda leer el capítulo 2 (*Haptic Interfaces*) de Kortum [25].

A.3 TECNOLOGÍA UTILIZADA

En este trabajo se incluye un driver de modalidad que trabaja con displays táctiles y feedback vibro-táctil, por lo que se hará foco en la forma de trabajo de los mecano-receptores. Para aprovechar la ocurrencia masiva de dispositivos vibro-táctiles e interfaces sensibles al tacto, se decidió que el driver utilice el hardware incluido en los móviles, accediendo a los mismos a través de las interfaces estándar HTML5 correspondientes, e. g.API de vibración.

B

MODALIDAD AERO-GESTUAL

Esta modalidad debe ser catalogada inicialmente, dentro del grupo superior de las interfaces gestuales.

Por interfaces gestuales se hace referencia a aquellas que utilizan movimientos del cuerpo y rostro como señales, que pueden acompañar a la comunicación verbal. En el contexto HCI, el uso de dichas interfaces implica, a través de cámaras o sensores diversos, capturar estos gestos y hacerlos útiles dentro de alguna aplicación.

En el uso cotidiano que le damos a los gestos, estos suelen complementar la comunicación verbal y son producidos mediante el uso del cuerpo y el rostro. Por esta razón es que esta modalidad tiende a ser usada fácilmente como complemento de otras, i. e. interfaz de voz. Una de las ventajas que trae consigo esta interfaz es la universalidad de los gestos en muchos casos, cuando esto ocurre es porque contamos con gestos autodefinidos que rompen las barreras del idioma.

Usualmente encontramos interfaces gestuales en los *pads* de las computadoras portátiles. Un buen ejemplo es el pad de las *ultrabooks*, ya que es común encontrar soporte de numerosos gestos que podemos realizar usando uno o mas dedos. Al momento de desarrollar cualquier interfaz, si consideramos incluir gestos en la misma, la premisa no debe ser simplemente incluirlos, si no, que tengan un objetivo propio y claro dentro del sistema.

Uno de los propósitos mas comunes cuando se utilizan interfaces gestuales es el de brindar una forma de trabajo mas "natural" al usuario, eliminando cualquier necesidad de hardware extra para interactuar con el sistema, incluso permitiéndole enviar ordenes usando gestos populares, evitando la necesidad de aprender un lenguaje nuevo.

B.1 SENTIDOS INVOLUCRADOS

Cuando se habla de gestos, el principal sentido involucrado es el de la visión. Como en muchos otros casos, a través de la vista, recibimos un patrón que cerebro puede detectar muchas veces gracias a nuestro bagaje cultural, aunque en otros casos puede darse que el individuo aprenda un conjunto de gestos nuevos para interactuar con el sistema.

Esta modalidad es de una sola vía, es decir, es un canal de entrada al sistema que no podemos retro-estimular directamente (es posible estimular a otros sentidos como respuesta a un gesto, pero esto es otra cuestión). Por este motivo y como se menciono anteriormente, esta modalidad suele trabajar en conjunto con otras modalidades en sistemas multimodales, muchas

veces funcionando como una herramienta para eliminar ambigüedad introducida por otras interfaces.

Los gestos pueden ser clasificados de tres maneras:

- Semántica; gestos auto-contenidos, comúnmente usados en comunicación no-verbal.
- Funcional; relacionados al contexto de una aplicación, suelen indicar directivas o comandos, e. g.manipulación directa de un elemento.
- Descriptiva; se concentra en *como* el gesto es realizado.

Finalmente, es necesario tener en cuenta la distancia requerida para realizar e interpretar correctamente el gesto, ya que la misma impacta contra el *vocabulario* de gestos a utilizar. Por ejemplo, no es la misma la interacción que podemos tener con una persona frente a frente que si estamos a varios metros de distancia, el conjunto de gestos que usamos cambia. Este es un factor a tener en cuenta cuando se diseña una interfaz gestual.

Particularmente en este trabajo, se hace referencia a interfaces aéreo-gestuales, es decir aquellas que no necesitan de una interfaz externa, como puede ser un dispositivo táctil, para ser usadas; tampoco es necesario que el usuario se acerque a alguna pantalla (aunque esto depende del vocabulario gestual a utilizar). Estas interfaces tienden a usar muy poco o ningún hardware extra, mas allá del dispositivo de escaneo, esto les permite funcionar mas "naturalmente", i. e.sin necesidad de nada extra en el usuario, posibilitando incluso la captura de gestos conscientes o inconscientes en el usuario.

B.2 INTERFACES POPULARES

Algunas interfaces utilizan técnicas de visión por computadora para detectar los gestos relevantes del sistema. El trabajo de Wang and Popović [42] es un excelente ejemplo de una técnica efectiva para *trackear*, en este caso, las manos y usarlas como un sistema de gestos funcional, realizando tareas que incluyan manipulación directa con algún elemento de nuestro sistema.

Otra interfaz que permite capturar información gestual sin necesidad de hardware extra *en el* usuario es el dispositivo Leap Motion. El mismo utiliza sensores infrarrojos para detectar las manos y triangular posición en tiempo real. Finalmente, otros dispositivos, populares en sistemas de entretenimiento como son Kinect, de Microsoft y Wii U de Nintendo, permiten detectar gestos a nivel cuerpo y a una distancia de un par de metros,

generando experiencias de inmersión dentro de los juegos, *trackeando* información gestual del esqueleto humano.

B.3 TECNOLOGÍA UTILIZADA

Para este trabajo, se utilizo el dispositivo **Leap Motion** como herramienta para capturar y acceder a la modalidad aéreo-gestual. Para esto se desarrollo un driver que captura información de uno de los dedos de las manos y la entrega al sistema como si fuera puntero, permitiendo al usuario interactuar de manera funcional con los objetos de la aplicación.

En la figura 15 se puede observar la representación un gesto que puede ser capturado por algún reconocedor, como el mencionado Leap Motion o incluso una cámara web. Este gesto se convierte en una entrada para el sistema que podemos aprovechar, utilizando la plataforma *plusultra*, en el contexto de una aplicación web.

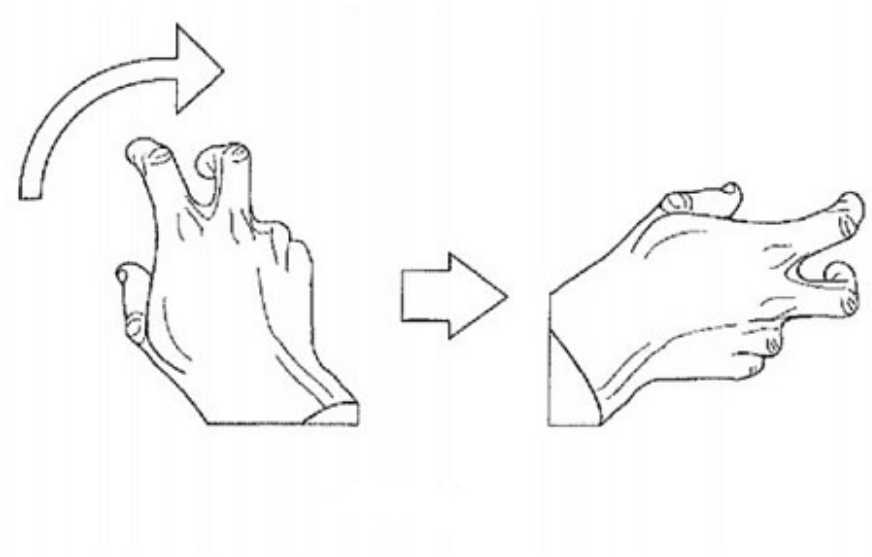

Figura 15: Representación de un gesto aereo.

ÍNDICE DE FIGURAS

BIBLIOGRAFÍA

[1] Tilde Inc. . Ember.js, 2011. URL http://emberjs.com/. Accedido: 2014-08-03.

[2] Richard A Bolt. *"Put-that-there": Voice and gesture at the graphics interface*, volume 14. ACM, 1980.

[3] ML Bourguet. Designing and Prototyping Multimodal Commands. *INTERACT*, (c):717–720, 2003. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Designing+and+Prototyping+Multimodal+Commands#0.

[4] Alessandro Bozzon, Sara Comai, Piero Fraternali, and Giovanni Toffetti Carughi. Conceptual modeling and code generation for rich internet applications. *Proceedings of the 6th international conference on Web engineering - ICWE '06*, page 353, 2006. doi: 10.1145/1145581.1145649. URL http://portal.acm.org/citation.cfm?doid=1145581.1145649.

[5] Browserling. browserify, 2010. URL http://browserify.org/. Accedido: 2014-08-05.

[6] CommonJS. Modules, 2013. URL http://wiki.commonjs.org/wiki/Modules. Accedido: 2015-08-09.

[7] Francesco Cutugno, Vincenza Anna Leano, Roberto Rinaldi, and Gianluca Mignini. Multimodal framework for mobile interaction. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 197–203. ACM, 2012.

[8] André da Silva and Heloísa da Rocha. e-learning environment with multimodal interaction: A proposal to improve the usability, accessibility and learnability of e-learning environments. In *ACHI 2013, The Sixth International Conference on Advances in Computer-Human Interactions*, pages 483–487, 2013.

[9] J Duhl. White paper: Rich internet applications. *Avail-able at http://wwwmacromedia. comn/ . . .*, 2003. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:WHITE+PAPER:+Rich+Internet+Applications#0.

[10] B Dumas, D Lalanne, and R Ingold. Prototyping multimodal interfaces with smuiml modeling language. In *CHI 2008 Workshop on User Interface Description Languages for Next Generation User Interfaces, CHI*, pages 63–66, 2008.

[11] Bruno Dumas, Rolf Ingold, and Denis Lalanne. Benchmarking fusion engines of multimodal interactive systems. In *Proceedings of the 2009 international conference on Multimodal interfaces*, pages 169–176. ACM, 2009.

[12] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. Multimodal interfaces: A survey of principles, models and frameworks. In *Human Machine Interaction*, pages 3–26. Springer, 2009.

[13] Jason Farrell and George S. Nezlek. Rich internet applications the next stage of application development. *2007 29th International Conference on Information Technology Interfaces*, pages 413–418, June 2007. ISSN 1330-1012. doi: 10.1109/ITI.2007.4283806. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4283806.

[14] The Dojo Foundation. Requirejs module loader, 2011. URL http://requirejs.org/docs/1.0/docs/whyamd.html#amd. Accedido: 2015-08-09.

[15] Piero Fraternali, Gustavo Rossi, and Fernando Sánchez-Figueroa. Rich internet applications. *IEEE Internet Computing*, 14(3):9–12, May 2010. ISSN 1089-7801. doi: 10.1109/MIC.2010.76. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5481362.

[16] Luca Frosini, Marco Manca, and Fabio Paternò. A framework for the development of distributed interactive applications. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems*, pages 249–254. ACM, 2013.

[17] Francesco Gabbanini, Laura Burzagli, and Pier Luigi Emiliani. An innovative framework to support multimodal interaction with smart environments. *Expert Systems with Applications*, 39(3):2239–2246, 2012.

[18] Jesse James Garrett et al. Ajax: A new approach to web applications. 2005.

[19] Google. Angularjs, 2010. URL https://angularjs.org/. Accedido: 2014-08-03.

[20] The Web Hypertext Application Technology Working Group. The web platform: Browser technologies, 2011. URL http://platform.html5.org/. Accedido: 2014-08-05.

[21] Kelly S Hale and Kay M Stanney. Deriving haptic design guidelines from human physiological, psychophysical, and neurological foundations. *Computer Graphics and Applications, IEEE*, 24(2):33–39, 2004.

[22] DocumentCloud Jeremy Ashkenas. Backbone.js, 2010. URL http://backbonejs.org/. Accedido: 2014-08-03.

[23] Inc. Joyent. Node.js, 2014. URL http://nodejs.org/. Accedido: 2014-08-05.

[24] Anne-marie Kermarrec. The Many Faces of Publish/Subscribe. 35(2):114–131, 2003.

[25] Philip Kortum. *HCI beyond the GUI: design for haptic, speech, olfactory, and other nontraditional interfaces*. Morgan Kaufmann, 2008.

[26] Kenneth WK Lo, Will WW Tang, Grace Ngai, Stephen CF Chan, and Jason TP Tse. Introduction to a framework for multi-modal and tangible interaction. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 3001–3007. IEEE, 2010.

[27] Kenneth WK Lo, Will WW Tang, Grace Ngai, Alvin TS Chan, Hong Va Leong, and Stephen CF Chan. i* chameleon: a platform for developing multimodal application with comprehensive development cycle. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1103–1108. ACM, 2013.

[28] OMG. Object management group, 1997. URL http://www.omg.org/. Accedido: 2014-08-03.

[29] OMG. Webml: The web modeling language, 2003. URL http://www.webml.org/webml/page1.do. Accedido: 2014-08-03.

[30] OMG. Ifml: The interaction flow modeling language, 2014. URL http://www.ifml.org/. Accedido: 2014-08-03.

[31] Addy Osmani. Learning javascript design patterns, 2014. URL http://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript. Accedido: 2014-08-17.

[32] Sharon Oviatt. Ten myths of multimodal interaction. *Communications of the ACM*, 42(11):74–81, 1999.

[33] Sharon L Oviatt and Karen Kuhn. Referential features and linguistic indirection in multimodal language. In *ICSLP*, 1998.

[34] Juan Carlos Preciado, Marino Linaje, Fernando Sanchez, and Sara Comai. Necessity of methodologies to model rich internet applications. In *Web Site Evolution, 2005.(WSE 2005). Seventh IEEE International Symposium on*, pages 7–13. IEEE, 2005.

[35] Charlie Robbins. npm: innovation through modularity, 2013. URL http://blog.nodejitsu.com/npm-innovation-through-modularity/. Accedido: 2014-08-05.

[36] Jose Maria Arranz Santamaria. The single page interface manifesto. URL http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php. Accedido: 2014-09-05.

[37] W3C. Multimodal interaction framework, 2003. URL http://www.w3.org/TR/mmi-framework/. Accedido: 2013-07-06.

[38] W3C. The websocket api, 2012. URL http://www.w3.org/TR/websockets/. Accedido: 2014-09-05.

[39] W3C. Multimodal architecture and interfaces, 2012. URL http://www.w3.org/TR/mmi-arch/. Accedido: 2013-07-06.

[40] W3C. Touch events, 2013. URL http://www.w3.org/TR/touch-events/. Accedido: 2014-09-05.

[41] W3C. Vibration api, 2014. URL http://www.w3.org/TR/vibration/. Accedido: 2014-09-05.

[42] Robert Y Wang and Jovan Popović. Real-time hand-tracking with a color glove. In *ACM Transactions on Graphics (TOG)*, volume 28, page 63. ACM, 2009.

[43] Jevon M Wright and Jens B Dietrich. Requirements for rich internet application design methodologies. In *Web Information Systems Engineering-WISE 2008*, pages 106–119. Springer, 2008.

COLOFÓN

Documento original compuesto en LaTeX utilizando el modulo tipográfico `classicthesis` desarrollado por André Miede.

Impreso en la ciudad de La Plata, Buenos Aires, Argentina en Diciembre de 2015.