Un enfoque para soporte de tareas mediante adaptación en el cliente Web

Sergio Firmenich

Facultad de Informática

Universidad Nacional de La Plata

Director: Dra. Silvia Gordillo

Co-Director: Dr. Gustavo Rossi

Co-Director: Dr. Marco Winckler

Tesis realizada para obtener el grado de

Doctor en Ciencias Informáticas

...

Agradecimientos

Quiero agradecer a mi familia por el apoyo brindado en este tiempo.

También quiero agradecer muy especialmente a mis directores Gustavo Rossi, Marco Winckler y Silvia Gordillo que han sido mi guía constante.

Finalmente, a todas las personas que me han contribuido y me han acompañado durante el desarrollo de esta tesis.

Índice general

Índice de figuras

Capítulo 1

Introducción

Este capítulo introduce el contenido y establece los fundamentos de esta tesis. La subsección 1.1 explica la motivación de este trabajo a la vez que plantea la problemática. En la subsección 1.2 se establecen concretamente los objetivos, mientras que en la 1.3 se define la contribución. Finalmente, la subsección 1.4 resume y explica la estructura de esta tesis.

1.1. Motivación

Desde sus inicios hasta la fecha, las aplicaciones que conforman lo que normalmente llamamos la Web (*World Wide Web*) han evolucionado considerablemente. Al comienzo, dichas aplicaciones eran simples repositorios de información basados en hipertexto e hipermedios (imágenes, vídeos, etc.), enlazados unos con otros. Ante esta estructura los usuarios tenían prácticamente el único rol de consumir la información provista por estas aplicaciones o sitios Web. Con el transcurso del tiempo y el avance de las tecnologías las aplicaciones Web se han vuelto cada vez mas complejas y, en este crecimiento en la complejidad, el rol de los usuarios ha mutado también a un rol mas complejo. Actualmente, un usuario normal no solo consume información sino que además, en un rol mas activo, participa en la generación de lo que hoy en día vemos publicado en muchas de las aplicaciones llamadas Web 2.0: redes sociales, wikis, etc. De todas maneras, no es que la Web sea solo una plataforma donde los usuarios consumen y generan contenido; por

el contrario, dejó de serlo para ser también una plataforma de servicios. Esto implica que los usuarios consumen un conjunto de servicios provistos en la Web para poder realizar diversas actividades al margen, claro está, de aquellas otras actividades relacionadas con la información *per se*. Desde hace ya varios años, y con una tendencia creciente, los usuarios consumen todo tipo de servicios de la Web: compran de pasajes de transporte, compran artículos en sitios de comercio electrónico, reservan hoteles, alquilan autos, se registran en conferencias y congresos, administran sus cuentas bancarias e incluso utilizan aplicaciones para desenvolverse en sus ocupaciones (los estudiantes tienen sistemas donde administran los cursos a los que asisten, los contadores utilizan sistemas de instituciones gubernamentales para realizar sus labores, etc.).

Cada una por separada, las tareas mencionadas pueden parecer simples. Sin embargo, el análisis no debe terminar allí. En primer lugar, existe un factor de reincidencia o repetición: reservar un habitación en un hotel, puede ser una tarea que determinados usuarios realizan frecuentemente. Tal vez hacer la reserva, implique utilizar mas de una aplicación Web (para comparar precios, ubicaciones, condiciones, etc.) lo cual significa, por su parte, ingresar la misma información reiteradamente (lugar, fecha de ingreso, fecha de salida, etc.).

Además existe un factor de relación entre las tareas. Por ejemplo, reservar un habitación en un hotel probablemente sea una actividad que un usuario realiza en conjunto con la compra de pasajes de transporte. La tarea por si misma de comprar, digamos pasajes aéreos, conlleva las mismas vicisitudes que la tarea de reservar una habitación de hotel, pero ambas en conjunto son aun una tarea mas abstracta, mas compleja: es planificar un viaje. Comúnmente planificar un viaje también puede involucrar búsqueda de información turística como horarios de museos, puntos de interés, etc.

El punto a destacar aquí es que una actividad determinada, por ejemplo programar un viaje, requiere que se utilicen varias aplicaciones Web que no estando integradas, obligan al usuario a manejar un flujo de información de manera *ad-hoc*. Supongamos que un investigador quiere asistir a una conferencia, entonces, posiblemente utilice mas de una aplicación Web para organizar su viaje:

- El sitio Web de la conferencia para buscar información acerca de la misma: lugar, fechas, etc.

- El sitio Web para registrarse en la conferencia, no necesariamente el mismo que el anterior.
- Un sitio para comprar pasajes hasta el lugar de la conferencia.
- Un sitio para reservar alojamiento en algún hotel.
- Un sitio para reservar un vehículo.

Este tipo de escenario es común para muchos usuarios, y si bien requiere el uso de múltiples aplicaciones Web, además implica un flujo de información entre dichas aplicaciones que por límites legales y técnicos no pueden manejar.

Si bien podemos suponer que no todos los usuarios realizan este tipo de tareas, sabemos que no es una casualidad que exista una diversidad de usuarios que si las realizan.

Aunque podemos suponer que no todos los usuarios realizan este tipo de tareas; podemos suponerlo porque existe una diversidad de usuarios que no es casual. En este punto, entra en juego un hecho que no puede quedar fuera de discusión: mientras las aplicaciones Web crecían en su complejidad, la Web se expandía y alcanzaba cada vez a mas usuarios.

Esto es relevante ya que en combinación tenemos: i) complejas tareas que se realizan luego de utilizar varias aplicaciones Web y manejando dificultosamente flujos de información por parte del usuario, ii) una muy diversa masa de usuarios que pueden realizar estas tareas en la Web.

Dicha combinación conlleva una problemática a la hora de construir una aplicación Web que además de ser compleja (en cuanto a los servicios que provee) e integrable (con otras aplicaciones), a la vez se adapte a las necesidades, preferencias o capacidades de cada potencial usuario. Estas problemáticas no son nuevas, y se han realizado varios esfuerzos desde el campo de la investigación para resolverlas.

Por un lado, tal vez el área mas antigua y madura sea la de adaptabilidad de las aplicaciones Web Brusilovsky et al. [2007a]. Estos trabajos reúnen toda una serie de mecanismos y propuestas para dar soporte de adaptación a las aplicaciones Web considerando tanto a las preferencias y capacidades cómo a las actividades

del usuario. La mayoría de estos trabajos están pensados para ser aplicados durante el proceso de diseño de las aplicaciones, es decir, cuando se es parte del equipo de desarrollo. El principal problema con este tipo de propuestas es que el usuario nunca es considerado como usuario de otras aplicaciones, es decir, que los mecanismos de adaptación funcionan a lo sumo considerando la actividad del usuario dentro de los limites de la aplicación.

Sin embargo este no es el único problema, ya que los usuarios están restringidos en cuanto a qué se adapta de las aplicaciones: note que estas son decisiones de los desarrolladores. En este sentido, y con el avance tanto de tecnologías del lado del cliente cómo de las habilidades de los usuarios, se han ido construyendo distintas herramientas para adaptar el contenido y la funcionalidad que las aplicaciones Web prevén originalmente. De la misma manera, las adaptaciones suelen estar enmarcadas en función de las preferencias de los usuarios, generalmente considerando cómo el usuario prefiere usar una aplicación Web en particular. Este enfoque, llamado *Web augmentation* Bouvin [1999], permite modificar a distintos niveles las interfaces gráficas de las aplicaciones Web, agregando, modificando o eliminando tanto funcionalidad como contenido.

Por otro lado, a la hora de la integración de aplicaciones Web, esta claro qué, como se mencionó previamente, existen límites legales y técnicos que impiden que esto sea factible. Legalmente, la información que los usuarios introducen en las aplicaciones Web no puede ser compartida con quién el usuario no lo ha autorizado. Técnicamente, no es posible integrar cualquier par de aplicaciones Web que el usuario desee utilizar. Desde una aplicación determinada, solo puede saberse lo que el usuario realiza dentro de los límites de esa aplicación. No obstante, así como los mecanismos de *Web augmentation* permiten adaptar las aplicaciones Web de otras maneras a las soportadas por la mismas, existen propuestas desde el campo de la Ingeniería Web para integrar aplicaciones que originalmente no estaban integradas. En este contexto han surgido una serie de herramientas que permiten integrar contenidos e informaciones de diferentes sitios Web para así obtener una nueva aplicación que surge de haber consumido y combinado diferentes servicios Web de varias otras aplicaciones. Estas herramientas llamadas *mash-ups* Yu et al. [2008] son pensadas a partir de la necesidad de integrar aplicaciones Web que originalmente no estaban integradas. Es decir, que estas herramientas para construir

mash-ups no están dirigidas a los desarrolladores de las aplicaciones Web sino que estan pensadas para ser utilizadas por terceros, por usuarios finales. El principal problema con las aplicaciones *mash-ups* es que requieren que las aplicaciones Web que integra tengan una API (***A****pplication* ***P****rogramming* ***I****nterface*) definida y pública, es decir, un conjunto de servicios conocidos los cuales pueden ser consumidos por terceros. Sumado a esto, los *mash-ups* suelen sacar a las aplicaciones Web de su contexto natural: son aplicaciones diferentes a las aplicaciones Web que integra y hace que los usuarios no accedan a las aplicaciones originales.

Detrás de estas dos líneas de investigación, *Mash-ups* y *Web augmentation*, subyace la filosofía de poner en manos del usuario final un poder que extralimita lo que originalmente se le permite hacer desde las aplicaciones. Aunque no cualquier usuario tiene la habilidad de desarrollar aplicaciones *mash-ups* ni tampoco artefactos de *Web augmentation*, es verdad que muchos sí, y que los que pueden hacerlo también pueden compartir lo que han hecho. Un claro ejemplo de ello son las comunidades de *client-side scripting*. Aunque el concepto de *client-side scripting* será abordado y definido en los capítulos siguientes, basta por ahora aclarar que es la actividad de desarrollar *scripts* que corran en el cliente en pos de mejorar en algún aspecto la experiencia del usuario al navegar la Web. El mas importante de los repositorios posee miles de scripts (desarrollados por usuarios), pero aun mas importante, muchos de estos scripts han sido descargados e instalados por otro usuarios (no desarrolladores) millones de veces. En este trabajo se considera sumamente importante la idea del desarrollo por parte de usuarios finales (*End-User Development*) Paternò et al. [2003], no solo porque tiene su éxito comprobado, sino porque además el objetivo final es mejorar la experiencia del usuario al realizar sus tareas. Tareas que ellos saben cómo realizan usualmente, con qué aplicaciones Web, con qué información, etc.; es decir, nadie mejor que la propia masa de usuarios para desarrollar los artefactos que soporten cómo ellos quieren soportar sus tareas.

Sin embargo, mientras propuestas de *Web augmentation* permiten adaptar un sitio Web y propuestas como *mash-ups* permiten integrar varias aplicaciones Web para generar una aplicación nueva, existe una distancia considerable entre ambos enfoques. En esta tesis, se considera un punto medio en el qué a través de tareas que llamaremos de augmentación, se tratará de extender lo percibido por el usua-

rio durante su uso de la Web para llevar a cabo una tarea determinada. De esta manera, las adaptaciones que se realizan en las aplicaciones Web y las acciones que el usuario puede realizar están sujetas a su tarea actual. Esto se logra mediante la combinación de lo que llamaremos tareas primitivas (aquellas realizadas manualmente por el usuario) con tareas de adaptación (o *augmentación*) basadas en artefactos de software con objetivos concretos (objetivos de adaptación, integración, etc.) que pueden ser automatizadas.

Normalmente, el usuario cuando navega por la Web realiza un conjunto de tareas caracterizadas como *primitivas* Byrne et al. [1999] Heath [2010] (Completar un campo de un formulario, Cargar una URL en el Browser Web, Navegar un link, etc., son algunos ejemplos). En el contexto de este trabajo, se pone en juego un nuevo tipo de tareas, tareas de augmentación, que servirán para facilitar e integrar las tareas de usuario. Una tarea será una tarea de augmentación cuando al no ser considerada una tarea primitiva permita: i) augmentar de algún modo lo que el usuario percibiría normalmente de la Web; ó ii) permitir al usuario realizar acciones no nativas en el contexto del uso de la Web. Teniendo esto en consideración, mediante la combinación de tareas primitivas y de tareas de augmentación se puede facilitar la ejecución de una tarea mas abstracta: planear un viaje, comprar productos, etc. En esta tesis además de realizar un análisis respecto de estos tipos de tarea, se especifica una manera de integración que bajo un formalismo a partir del cual estas combinaciones puedan ser objeto de estudio, se puede intentar mejorar esa combinación en pos de encontrar defectos en dicha composición y por lo tanto poder refinar cómo el usuario realiza la tarea.

1.2. Objetivos

Como consecuencia del complejo uso que los usuarios tienen sobre la Web surge una nueva problemática que no ha sido aún resuelta y que las técnicas de adaptabilidad clásicas (aquellas que se aplican durante el desarrollo de la aplicación) son incapaces de resolver: la navegación inter-aplicación (e incluso a veces intra- aplicación) genera perdidas de contexto en la tarea y/o interés de los usuarios que implican un empeoramiento de sus experiencias. Si mirásemos el historial de navegación cuando un usuario ha realizado una tarea en la Web, posiblemen-

te encontraremos que más de una aplicación ha sido utilizada. Lo cual implica que la adaptabilidad no solo es necesaria porque diferentes usuarios pueden tener diferentes necesidades dentro de una aplicación determinada, sino también que es necesaria porque diferentes usuarios utilizan diferentes subconjuntos de aplicaciones Web para realizar tareas similares y que los cambios de la aplicación en uso suelen romper de una u otra manera el proceso cognitivo que los mismos van llevando a cabo.

Basado en esto, se plantea como objetivo general de este trabajo *diseñar e implementar una solución a partir de la cual se extienda el actual enfoque de adaptabilidad e integración de aplicaciones Web con nuevos mecanismos que permitan adaptar las mismas en pos de mejorar la experiencia de los usuarios cuando realizan tareas intra e inter-aplicación.*

Este objetivo general, se descompone en los siguientes objetivos específicos:

- Identificar y definir qué tipo de información referida al usuario y su actividad esta disponible en el lado del cliente considerando su uso para soportar sus tareas.
- Identificar y tipificar aquellas tareas de augmentación pueden realizarse en el cliente.
- Analizar y diseñar mecanismos que permitan a los usuarios realizar adaptaciones bajo demanda en pos de satisfacer requerimientos volátiles referidos a la tarea actual de los mismos.
- Analizar y diseñar mecanismos que den soporte al usuario cuando realiza tareas reiteradamente, siempre siguiendo un mismo escenario de uso.
- Analizar y diseñar los artefactos de software que son necesarios para llevar a cabo estas adaptaciones.
- Mantener la filosofía subyacente en propuestas como *Web augmentation* y *mash-ups* respecto al desarrollo por parte de usuarios finales
- Permitir a los usuarios compartir los artefactos desarrollados.

- Proveer una estructura de software robusta pero a su vez lo suficientemente flexible para orquestar los artefactos desarrollados por usuarios, es decir tareas de augmentación, con tareas primitivas en pos de soportar sus tareas.
- Evaluar los resultados conseguidos para poder medir el impacto de los nuevos mecanismos de adaptación diseñados y orientando dichas evaluaciones a relevar en que medida es mejorada la experiencia de usuario al realizar sus tareas.

1.3. Contribución

Los siguientes puntos resumen la contribución concreta de este trabajo:

- Un enfoque que combina la idea de integración existente en *mash-ups* con los objetivos provenientes del área de *Web augmentation*. Esta combinación es en pos de soportar las tareas de usuarios.
- Un conjunto de herramientas conceptuales que sustentan los mecanismos de integración propuestos.
- Un *Framework* que provee un conjunto de herramientas tecnológicas a partir de las cuales los usuarios pueden desarrollar principalmente:
  - Artefactos de adaptación basados en técnicas de *Web augmentation*
  - Composiciones de tareas primitivas y de estos artefactos para el soporte de tareas de usuario
- Un DSL (*Domain Specific Language*) Fowler and Parsons [2010] con el cual poder especificar composiciones de dichos artefactos. De esta manera, se proveen escenarios de uso de la Web que tienen un objetivo concreto y que están basados en una secuencia de: i) actividades que usualmente el usuario realiza en el web (visitar un sitio Web, llenar un formulario, etc.); ii) actividades realizadas por los artefactos de adaptación que ayudan de alguna manera al usuario.

- Un conjunto de herramientas que soportan a los usuarios no-desarrolladores en el uso del *framework* y sus extensiones.

1.4. Estructura del documento

Dejando de lado este capítulo introductorio donde se ha explicado la motivación de este trabajo y descripto los objetivos y la contribución, esta tesis esta estructurada en 8 capítulos principales:

- Capítulo 2: introduce el contexto teórico necesario que incluye aspectos genéricos de hipermedia y sistemas basados en Web. También abarca los mecanismos clásicos contemplados en aplicaciones Web e Hipermedia. En este mismo capítulo se describe la adaptación del lado del cliente y las diferentes alternativas para realizar dichas adaptaciones. A su vez, se plantea la potencialidad y aplicabilidad de las adaptaciones del lado del cliente.

- Capítulo 3: la gama de posibilidades que existen bajo el manto de adaptación del lado del cliente es muy amplia. En este sentido, el capítulo 3 presenta los fundamentos para una arquitectura flexible, analizando las siguientes dimensiones claves: tipos de adaptación soportadas, los mecanismos de ejecución de dichas adaptaciones, qué información es posible consumir para realizar las adaptaciones, hasta llegar a cómo debería ser posible desarrollar nuevos artefactos de adaptación.

- Capítulo 4: el motor principal de esta propuesta de adaptación del lado del cliente es presentado en este capítulo. Aquí se describe el *framework* con todos los artefactos involucrados en el proceso de adaptación, la arquitectura, la extensibilidad y, además, las formas de interacción desde el punto de vista del usuario final.

- Capítulo 5: en el capítulo 5 se introduce y desarrolla la propuesta de composición de tareas para la mejora de la experiencia de usuario utilizando adaptación del lado del cliente. Este capítulo incluye principalmente una descripción de los tipos de tareas contempladas, el proceso de composición, un lenguaje especifico de dominio para conseguir dicha composición, para

finalmente presentar un conjunto de herramientas para crear y utilizar estas composiciones.

- Capítulo 6: en este capítulo se plantean diversos ejemplos que son analizados en profundidad para relevar el efecto concreto del uso de las herramientas conseguidas aplicando las ideas presentadas en esta tesis.

- Capítulo 7: una evaluación basada en el uso de las herramientas desarrolladas es presentada en capítulo 7. La evaluación es planteada desde distintos puntos de vista considerando los distintos tipos de artefactos que se han ideado en el marco de esta tesis.

- Capítulo 8: un completo análisis del estado del arte es realizado en el capítulo 8. En dicho capítulo se hace un repaso de los trabajos relacionados en dos categorías de acuerdo al tipo de contribución (*mash-ups* y *web augmentation*). También se incluye una breve discusión sobre el modelado de tareas.

- Capítulo 9: se presentan las conclusiones del trabajo, que incluye tanto un repaso entre los objetivos planteados y lo conseguido como también algunas consideraciones finales acerca de la contribución. Además, en esta sección también se plantean los trabajos de investigación que están planeados para el futuro en pos de seguir profundizando las contribuciones aquí presentadas y también otras lineas de investigación relacionadas.

Se incluyen tres anexos para poder mostrar aspectos que aunque importante, no fueron desarrollados durante el texto escrito en los capítulos presentados.

- Anexo A: Como desarrollar un *augmenter*.

- Anexo B: Escenarios completos para la evaluación.

- Anexo C: Publicaciones científicas realizadas durante el desarrollo de esta tesis.

Capítulo 2

Adaptabilidad en aplicaciones Web

Adaptar tiene más de una acepción, y tal vez la más simple y directa de ellas sea la más acorde para describir la adaptabilidad (desde el enfoque clásico y del modo mas simple) de sistemas informáticos: *Acomodar, ajustar algo a otra cosa.* Claramente, hablamos de *acomodar o ajustar* un sistema informático *a* el usuario (en una o mas de las muchas dimensiones referidas al usuario: características, preferencias, necesidades, capacidades, etc.). En el campo de la informática, el concepto de *sistema adaptable* no es para nada nuevo. Desde mediados de los años ochenta la temática de la adaptabilidad de los sistemas empieza a ser parte de la discusión. En la década del noventa empieza a establecerse una de las primeras clasificaciones respecto a que maneras existen de adaptar un sistema, terminando de definir así dos términos relevantes hasta el día de hoy: *adaptability* (adaptabilidad) y *adaptivity* ("adaptividad") Oppermann [1994].

Basta decir, por ahora, que la diferencia primordial entre un enfoque y otro se basa no tanto en qué es lo que se adapta del sistema sino a partir de qué y de quién se realiza la adaptación. Un sistema se dice *adaptable* cuando provee al usuario con herramientas que hacen posible que él mismo *cambie* (adapte) las características del sistema. En cambio, un sistema soporta "*adaptividad*" (es decir que el sistema es "*adaptativo*") si el sistema adapta por si mismo sus características en correspondencia con lo que detecta como necesidad del usuario.

Lo importante, mas allá del enfoque utilizado, es comprender que *adaptar* un sistema en pos de *personalizarlo* a un usuario es sumamente importante y que contribuye a mejorar la calidad en uso del software en general y de las aplicaciones Web en particular. Claro, calidad en uso que implica usabilidad y por ende una buena experiencia de usuario. Imaginemos un sistema desarrollado exclusivamente para un usuario, evidentemente, dicho sistema podría estar desarrollado a su medida siguiendo cada uno de sus requerimientos, preferencias, etc. En estas condiciones una buena experiencia de usuario podría ser fácilmente garantizada. Ahora imaginemos una aplicación Web que, esencialmente, puede ser utilizada por una gran masa de usuarios. Usuarios de muy diversas características: preferencias personales, capacidades personales, con distintos dispositivos desde los cuales acceden, diferentes contextos, de tarea, incluso hasta étnicas y de costumbres sociales. En estas circunstancias asegurar una buena experiencia en el uso de la aplicación para cualquier potencial usuario no es tan fácil, el sistema debería poder adaptarse para que cada usuario lo utilice de la manera que mejor experiencia le brinde.

En el campo de la Ingeniería Web se han realizado varios esfuerzos por hacer, en este sentido, lo mas flexible posible las aplicaciones. Pero podemos plantear el siguiente interrogante: *¿puede diseñarse e implementarse una aplicación Web de manera tal que se pueda adaptar de manera total a cualquier usuario?* La respuesta es no, como veremos en la subsección 2.2, y es por este motivo que dicho tópico sigue siendo un área de investigación en desarrollo.

Sin embargo, dicha restricción ha hecho que surjan otras alternativas de adaptación que se basan en modificar los documentos originales que las aplicaciones Web sirven. Hay situaciones que no pueden evitarse, entre ellas que los usuarios deseen mejorar su propia experiencia incluso mas allá de lo que las aplicaciones que utilizan les permiten. Un claro ejemplo son los sistemas de *transcodings* Bharadvaj et al. [1998], ampliamente utilizados para adaptar los contenidos de las aplicaciones Web para por ejemplo, hacerlos acorde a dispositivos de pantallas pequeñas. En esta línea, la adaptación del lado del cliente viene a sugerir que adaptar una aplicación no tiene que ver con la acepción introducida al inicio de este capítulo, sino con otra de sus acepciones: *Hacer que un objeto o mecanismo desempeñe funciones distintas de aquellas para las que fue construido.* Aunque

desde el punto de vista del *para qué* esta definición no es exacta, si lo es desde la visión de que ese objeto (una aplicación Web) sea distinto a lo que originalmente era, es decir, que alguien externo al equipo de desarrollo en el que se concibió la aplicación pueda aplicar modificaciones en pos de transformar la aplicación en algo que originalmente no era, ya sea agregando, modificando, eliminando contenido o bien agregando, eliminando o modificando funcionalidad.

En este capítulo se presenta el trasfondo teoríco-conceptual (relacionado con la adaptación de aplicaciones Web) necesario para entender la contribución de este trabajo. En la subsección 2.1 se presentan algunas definiciones básicas necesarias para contextualizar el problema en el ámbito Web. Como se dijo, en la subsección 2.2 se analizan y definen los métodos clásicos con los que las aplicaciones Web pueden soportan algún tipo de adaptación. Finalmente, en la subsección 2.3 se introduce el concepto de adaptación del lado del cliente (*Client-Side Adaptation*), las posibilidades que este enfoque brinda y también sus limitaciones.

2.1. Hipermedia y Sistemas basados en Web

Esta subsección esta enfocada en introducir los sistemas hipermedia en general para dar pie a las secciones subsiguientes que tratarán específicamente sobre adaptabilidad en aplicaciones Web.

Hipermedia es el nombre que se le da a un conjunto de documentos que están relacionados, es decir que están enlazados mediante *hiperenlaces*. Hipermedia es un concepto mas global de lo que se conoce como hipertexto, cuya diferencia radica en los tipos de documentos que entran en juego. Mientras que hipertexto hace referencia, claramente, a documentos de texto; la hipermedia no solo incluye documentos de texto sino que además incluye también archivos de sonido, gráfico y video.

La idea de sistemas hipertexto no es nueva, desde la segunda mitad del siglo XX (incluso algunos años antes) la idea de una plataforma basada en *hipertexto* ha sido forjada.

El primer prototipo de sistema de hipertexto fue descripto por Vannevar Bush en 1945 Bush [1945]. En este artículo Bush “plantea” un sistema llamado Memex basado en cómo los individuos deberían manejar sus textos y a partir del cual

ellos podrían hacer consultas sobre los mismos y además permitiría enlazar de alguna manera diversos documentos para luego poder consultarlos mas fácilmente a través de estas relaciones. Esta estructura ramificada planteada por Vannevar Bush, en la que el acceso a un documento puede darse de diversas maneras, ha sido incluso comparada con ideas provenientes de la literatura. Existen escritos sobre la comparación entre textos literarios y científicos, y algunos describen una similitud entre el trabajo de Bush y el escrito de Jorge Luis Borges *"El jardín de los senderos que se bifurcan"* Borges [1944], claro, desde el punto de vista de una estructura ramificada para la organización de información y para representar la experiencia de los individuos Wardrip-Fruin and Montfort [2003].

Al margen de la concepción original de dicha estructura, luego de años de diseño de hipotéticas plataformas basadas en hipertexto, se consiguieron los primeros sistemas concretos de este tipo y varios años después ya eran sistemas conocidos que, generalmente, eran distribuidos en CD-roms. Estos sistemas usualmente se definieron como un conjunto de documentos interrelacionados los cuales podían ser accedidos por los lectores. El lector (es decir, el usuario) mediante el uso de algún visor puede acceder a los documentos los cuales son presentados de tal manera que el lector pueda distinguir cuales son los hiperenlaces y además, claro está, pueda viajar o navegar a través de estos *links* (hiperenlaces), para visualizar otro nodo.

Ante este esquema de red de nodos enlazados con nodos, las aplicaciones pueden derivar en una estructura muy compleja desde el punto de vista de la navegación, es decir, desde el punto de vista de cómo el usuario navega por los hiperenlaces para pasar de un nodo a otro. Esta estructura podría ser tan compleja a punto tal de que el usuario puede *perderse* al seguir una secuencia de links determinada, lo cual significa que el usuario no sabe en que nodo de esta red esta actualmente visualizando, ni que secuencia de hiperenlaces navegar para llegar a otro nodo. A fines de la década de los ochenta se caracterizó este problema llamándolo *Lost in Hyperspace* Edwards and Hardman [1988] y tal vez fue una importante motivación para considerar la adaptación en sistemas hipertexto e hipermedia. Inicialmente, la adaptación de los nodos de las aplicaciones simplemente consideraba *qué links* mostrar para cierto usuario. Adaptando un nodo de una aplicación para restringir o direccionar mejor la navegación del usuario en

cuestión, tendría un impacto directo en la posibilidad de *perderse en el hiperespacio*.

Esto, y haciendo una comparación con la analogía mencionada acerca de la estructura ramificada descripta en el cuento *El jardín de los senderos que se bifurcan*, sería como restringir los *senderos* que alguien puede escoger, en pos de restringir los posibles desenlaces: *marcarle un rumbo* determinado al suceso de las cosas.

Sin embargo, y volviendo a hacer en foco en la adaptabilidad, éste fue solo el inicio de la adaptación de sistemas hipertexto. Debido a la masividad que fue tomando el uso de aplicaciones hipertexto e hipermedia se produjo, como se dijo en la introducción, un incremento en la diversidad de usuarios.

Este nuevo panorama incentivó aun más el interés por mejorar los mecanismos para adaptar las aplicaciones. Según Peter Brusilovsky, uno de los pioneros en el área de sistemas hipermedia adaptativos, existió un punto de inflexión en 1996 respecto a la magnitud de la investigación en este tópico, desde ya y como se dijo, impulsado entre otras cosas por el rápido incremento en el uso de la *Web* Brusilovsky [2001]. En el mismo trabajo, Brusilovsky remarca el impacto que tuvo en esta motivación un *special issue* de la revista científica UMUAI (*User Modeling and User-Adapted Interaction*): *UMUAI on Adaptive Hypermedia*. Además, en este trabajo, el autor habla acerca de dos ejes fundamentales en el contexto de hipermedia adaptativa:

- *¿Adaptar a qué?*: la decisión de adaptación en sistemas adaptativos estaba basada, al menos hasta 1996, en varias características de sus usuarios abstraídas en modelos de usuarios. Luego de este punto de inflexión señalado por Brusilovsky los sistemas empezaron a contemplar otros aspectos referidos al usuario, mas allá de las preferencias o sus propias características registradas en un modelo determinado.

  Después de 1996, empezaron a surgir sistemas hipermedia adaptativos que tenían en consideración la interacción del usuario con el sistema. Otros sistemas empezaron a considerar variables de contexto o entorno, los primeros de este tipo (y máxime con el auge de la Web) empezaron a tener en cuenta, por ejemplo, la plataforma del usuario (*hardware*, *software*, ancho de banda

de su conexión) o bien la ubicación del mismo.

- *¿Qué puede ser adaptado?*: en 1996 ya existía una taxonomía con dos tipos de objetivos claros en el contexto de sistemas hipermedia. Dicha clasificación distinguía principalmente la adaptación a nivel de presentación (*Adaptive Presentation*) y la adaptación a nivel de navegación (*Adaptive Navigation Support*).

Aunque estos dos interrogantes siguen siendo válidos (y es por esto que aun queda camino por recorrer en el campo de la adaptabilidad de aplicaciones Web) esta taxonomía evolucionó para luego ser una clasificación de sistemas hipermedia adaptativos; clasificación basada no solo en el modo en el cual se obtiene la información necesaria para realizar la adaptación, sino también en las causas o eventos que disparan las adaptaciones. En este sentido, puede decirse que los siguientes son, en la literatura actual, los posibles tipos de sistemas que soportan adaptación Garrigós [2008]:

- Sistema de Hipermedia Adaptable: un sistema de este tipo es aquel que permite que el usuario ingrese, mediante el uso de herramientas de configuración que el sistema provee, determinada información acerca de sus preferencias y/o características. En este sentido, la información usada para ejecutar la adaptación es conseguida de una manera explícita (es decir, solicitándosela al usuario) y la personalización es causada por una acción explícita del usuario. Básicamente, estos sistemas son aquellos que son adaptables desde el punto de vista de *adaptability* Oppermann [1994] mencionado al inicio de este capítulo.

- Sistema de Hipermedia Adaptativo: un sistema es adaptativo (en correspondencia con la definición de *adaptivity* Oppermann [1994]) cuando las decisiones de adaptación se toman automáticamente por parte del sistema ante cierta actividad del usuario. En contraposición con los sistemas adaptables, en este tipo de sistemas no sólo sucede que las adaptaciones se realizan sin intervención del usuario sino que la información para realizar las mismas suelen ser adquiridas implícitamente. Esto es posible gracias a

complejos modelos de usuarios que son actualizados dinámicamente ante la interacción del usuario.

Una variación de los sistemas hipermedia adaptativos encontrada en la literatura son sistemas llamados *proactivos*. Estos sistemas, que aunque son autónomos desde el punto de vista de aplicar las adaptaciones automáticamente, las mismas se realizan no ante el monitoreo de la actividad del usuario, sino por cuestiones de contexto.

En esta sección se introdujo de un modo muy conciso el concepto de hipermedia e hipertexto así como también el inicio de la investigación acerca de la adaptabilidad de las aplicaciones hipermedia, llegando así a una clasificación de tipos de sistemas hipermedia adaptativos.

En la siguiente sección se profundizan estas definiciones haciendo alusión a los mecanismos mas conocidos que son utilizados en el contexto de aplicaciones Web.

2.2. Mecanismos clásicos de adaptación en aplicaciones Web

Como se ha dicho hasta aquí, la adaptabilidad puede aportar, bien conducida, a la mejora de la calidad en uso de las aplicaciones Web. Esto se debe a que cuando se adapta una aplicación a determinadas condiciones del usuario o bien a condiciones en las que éste accede, su experiencia en el uso de la aplicación puede verse mejorada, ya que se considera de alguna manera su contexto.

Para remontarnos a uno de los primeros usos de adaptación (adaptación de contenido) en la Web en pos de mejorar la experiencia del usuario mediante la adecuación del contenido, tomaremos un típico ejemplo de adaptación basado en el ancho de banda de la conexión que el usuario esta utilizando. De esta manera, por ejemplo, en lugar de enviarse contenido de video se enviaban gráficos para así no hacer esperar al usuario cuando este accedía utilizando una conexión cuyo ancho de banda no era suficiente para proporcionarle un buen *feedback* desde la aplicación.

Este simple ejemplo de *qué* adaptar (en este caso adaptar el contenido que la aplicación brinda al usuario) ilustra el *por qué* adaptar: mejorar la experiencia de usuario. Adaptar una aplicación es en el fondo una manera de mejorar la calidad en uso de las aplicaciones Web.

En las subsecciones siguientes se abarcan dos partes fundamentales (a su vez relacionadas) en el camino de la adaptación: qué información se contempla acerca de los usuarios (modelo de usuario) y que tipos propuestas de adaptaciones existen en aplicaciones hipermedia adaptativas clásicas (donde clásico se refiere al hecho de que los mecanismos de adaptación son definidos y provistos por los desarrolladores de la aplicación). Estas áreas son verdaderamente amplias y siendo que no es intención de este trabajo hacer una descripción exhaustiva solo se tomarán propuestas que representativas del área.

2.2.1. Modelos de usuario

El modelado de usuarios es una de las características principales de sistemas hipermedia, máxime en aquellos que soportan adaptación. Existen muchos trabajos de investigación que proponen diversos modelos de usuarios, orientados también a distintos tipos de aplicaciones. En esta tesis el problema de modelar características del usuario no es relevante ya que la personalización esta comandada por el usuario y los artefactos de adaptación que éste desee utilizar; es por ello que no se hará un análisis exhaustivo de la temática sino que se brindará la información necesaria para comprender el poder y la intención detrás de los mismos.

En primer lugar debe quedar claro qué es un modelo de usuario. Un modelo de usuario es una representación de información sobre un usuario individual que es esencial para un sistema adaptativo, ya que con dicha información puede programarse el efecto de la adaptación: para distintos usuarios, la aplicación se comportaría de manera diferente Brusilovsky and Millán [2007].

Estos modelos abstraen información especifica y relevante para la aplicación. Un uso bien conocido es la búsqueda personalizada en la Web Micarelli et al. [2007] cuyo objetivo principal es responder a la búsquedas de los usuarios con aquella información de mayor relevancia para ellos.

Los modelos de usuarios mas complejos mantienen la información actualizada de distintas maneras: solicitando el ingreso de información explícitamente o incluso recogiendo información mediante la observación de la actividad del usuario.

Es decir que sin un modelo de usuario con información valedera, difícilmente puede adaptarse el sistema según se crea conveniente para el usuario. Esto significa que en enfoques clásicos, el modelado de usuario y los mecanismos de adaptación son caras de la misma moneda, como señala Brusilovsky and Millán [2007].

Esta relación de las caras la misma moneda implica que la cantidad, complejidad y naturaleza de la información representada en el modelo de usuario dependerá de las adaptaciones que se requieren realizar en el sistema. Cuanta mas información, mayores decisiones de adaptación podrán ser tomadas.

Según Brusilovsky and Millán [2007], existen tipos principales de información y que son importantes no solo para ser consideradas en un modelo de usuario, sino además para definir qué tipo de modelo de usuario utilizar:

- Conocimiento: este tipo de información implica conocer el conocimiento que el usuario tiene sobre el dominio representado por la aplicación. Esta característica de las mas tenidas en cuenta por aplicaciones hipermedia adaptativas. Es una característica variable, ya que el conocimiento del usuario sobre un tema en particular puede incrementar o no. Es decir que los sistemas que consideran esta información deben reconocer los cambios en el conocimiento del usuario. En general, la manera mas simple de modelarlo es con un modelo escalar, que estima el nivel del conocimiento del usuario en el dominio utilizando una escala (por ejemplo: buena, normal, pobre, ninguna).

- Intereses: los intereses de los usuarios constituyen la parte mas importante de un perfil de usuario, sobre todo en aquellos sistemas de búsqueda y recuperación de información, como así también en sistemas de recomendación.

- Objetivos y tareas: esta información representa el propósito inmediato en el trabajo del usuario dentro de un sistema. Este puede definirse al responder la pregunta ¿Qué es lo que el usuario quiere conseguir/realizar?. Esta

información es una de las mas volátiles, es decir, que puede cambiar varias veces incluso dentro de la misma sesión de uso de la aplicación.

- *Background*: se refiere al conjunto de características relacionadas a la experiencia previa del usuario aunque fuera del dominio central de la aplicación. Puede incluir información relativa a la profesión del usuario, a sus responsabilidades, experiencias de trabajo en áreas relacionadas, etc. Este tipo de información suele utilizarse para adaptación de contenido. Aunque tiene similitudes con el tipo de información descripta como *Conocimiento*, en este tipo no se requiere información detallada, sino mas basadas en estereotipos.

- Características individuales: es el conjunto de características que definen a un usuario. Algunos ejemplos son rasgos de personalidad (introvertido/extrovertido), factores relacionado a lo cognitivo o estilos de aprendizaje. En general, son características muy estables y prácticamente no cambian.

- Contexto: el contexto del usuario puede implicar la ubicación del usuario, el entorno físico, el contexto social, plataforma desde la cual accede a la aplicación, etc. Aunque algunos aspectos no pueden ser considerados como información *acerca del usuario*, esta información puede ser muy útil para realizar adaptaciones.

Una vez detectada qué información se tendrá en cuenta, dos aspectos mas del proceso de modelado de usuarios son requeridos:

- Cómo estructurar o representar la información de los usuarios.

- Cómo construir y mantener esa información.

En esta tesis no se consideran de relevancia estos aspectos, el usuario interesado puede encontrar información en Brusilovsky et al. [2007b], específicamente en los capítulos 1-3, 10-11 y 17.

Lo importante aquí es notar que sea cual fuere la información a contemplar, la estructura del modelo de usuario, los mecanismos de actualización de la información contenida en los mismos, etc., estas son todas decisiones de los desarrolladores de las aplicaciones que implican dejar fuera otros aspectos, ya que eso es un modelo: una abstracción de los aspectos de la realidad que son de interés para un

objetivo concreto. En este sentido, cualquier sistema adaptativo será adaptable bajo ciertas circunstancias y con mecanismos de adaptación predefinidos, como los explicados en la siguiente subsección.

2.2.2. Métodos para la adaptación en sistemas hipermedia

Existen numerosos trabajos de adaptación de aplicaciones hipermedia. Aquí, nos centraremos en algunos que muestren el potencial que existe en términos de adaptabilidad, considerando especialmente aquellos que contemplen la tarea o navegación del usuario. La intención detrás de las descripciones brindadas es mostrar que a pesar de la diversidad de enfoques existen limitaciones cuando éstos son implementados del lado del servidor de la aplicación.

En primer lugar, un trabajo sumamente relevante en el contexto de esta tesis es *Adaptive Navigation Support* Brusilovsky [2007]. En este trabajo de describen un conjunto de tecnologías que, originadas en el contexto de aplicaciones hipermedia, han sido aplicadas luego en aplicaciones Web. La intención es adaptar la presentación en pos de que la navegación de la aplicación, es decir, los links que el usuario percibe, sean alterados con algún criterio en particular. Claramente, para hacerlo se basa la decisión de la adaptación según la información registrada en perfiles o modelos de usuarios. Se describen varios métodos que pueden ser aplicados para guiar al usuario en la navegación:

- *Direct guidance*: para sugerir cuál es el mejor *link* para ser navegado por el usuario.
- *Link ordering*: para priorizar todos los links de una página Web de acuerdo al modelo de usuario: la idea es dejar en primer lugar el link de mayor relevancia.
- *Link hiding*: la intención detrás de este método es restringir el espacio de navegación mediante el ocultamiento o eliminación de links que dirigen a páginas irrelevantes.
- *Link annotation*: un método para augmentar los links con alguna forma de anotación que permita al usuario conocer mas acerca del estado de nodos que hay *detrás* de esos links.

- *Link generation*: este método permite crear nuevos links en los sitios Web dinámicamente, algunos de los cuales pueden convertirse en links permanentes mientras que otros pueden estar basados en el contexto actual del usuario.

Otros trabajos no están dirigidos tan específicamente a la navegación *per se*, sino que adaptan de manera mas genérica el contenido que las aplicaciones Web muestran. Como se ha mencionado, un caso clásico es el de los buscadores, que basado en información sobre intereses de los usuarios pueden tomar decisiones sobre los resultados a mostrar cuando se realizan búsquedas Micarelli et al. [2007].

También existen enfoques, como los descriptos por Bunt et al. [2007], en donde se describen técnicas para adaptar la presentación de los contenidos. Estas técnicas están orientadas a seleccionar y estructurar el contenido mas relevante para el usuario actual en el contexto de la interacción actual.

También son relevantes en el contexto de aplicaciones Web que soportan adaptación, lo que se conoce como sistemas de recomendación. Estos sistemas remiendan un *item* a un usuario en base a la descripción de tal *item* y de las características asociadas al usuario en su perfil (por supuesto, soportado por un modelo de usuario). Los sistemas de recomendación son ampliamente utilizados, y en un gran abanico de dominios.

Note que en esta subsección no se ha intentado hacer un análisis profundo de los trabajos realizados en el área de adaptabilidad, sino que, por el contrario, se ha intentado hacer un repaso por algunos trabajos emblemáticos en la temática que muestren el potencial y dejen entrever los limites explicados a continuación.

2.2.3. Límites

Las principales limitaciones que aquí se presentarán respecto a los mecanismos descriptos en las subsecciones se deben principalmente a aspectos técnicos y legales que imponen restricciones ya sea a los mecanismos utilizados o la información utilizada por dichos mecanismos de adaptación.

Sin embargo, no deja de ser importante un componente *político* acerca de la *no-integración* en las aplicaciones, por que claro, es también un problema de intereses. Intereses que no son de los usuarios, sino que son de la propietarios

de las aplicaciones web. Casualmente, no siempre el interés de estos es el interés de los usuarios propiamente dicho, y en ese sentido algunas decisiones van en contra la experiencia del usuario. Por ejemplo, ¿Por qué Facebook esta integrada por ejemplo, en la mayoría de los portales de noticias? ¿Por qué también lo esta Twitter?. Claramente, son sitios web que no compiten entre si: un sitio de noticias no compite con Facebook, por lo contrario, sacan rédito el uno del otro.

Mientras que Facebook saca provecho porque el debate de una noticia en realidad sucede en Facebook, el sitio Web se beneficia no solo en que no tiene ninguna lógica compleja para soportar esta característica de foro, sino ademas, que tiene a disposición una comunidad ya activa de usuarios.

Twitter y Facebook son aplicaciones que en algún punto son competencia. Los dos compiten, mas allá de la diferencia *per se* de cada red social, por un nicho de usuarios. A estas aplicaciones no les conviene la integración con la otra. Supongamos una nueva red social llamada Related-In destinada a socializar los contactos laborales. Los responsables de Linked-in, por su parte, difícilmente tomen la decisión de integrar los perfiles de sus usuarios con los respectivos perfiles de Related-In. Entonces la falta de integración de distintas aplicaciones Web muchas veces no se debe a razones técnicas, sino que a veces no sucede por motivos políticos. Esta decisión política, estratégica de una empresa, va en detrimento de la integración y por ende, puede repercutir negativamente en la experiencia de los usuarios.

Retomando los aspectos técnicos y legales, y aunque esto fue mencionado en el capítulo introductorio de esta presentación, aquí haremos un análisis con mayor profundidad.

Vamos a contextualizar el problema de dichas técnicas en el uso actual de la Web. Ubicarlo en este contexto es fundamental para entender el problema planteado en esta tesis ya que hay dos cuestiones complejas: la cantidad y diversidad de usuarios cada vez mas demandantes; y la cantidad y diversidad de aplicaciones Web que son utilizadas alrededor del mundo con diferentes objetivos.

Desde el punto de vista de la masiva y diversa población de usuarios en la Web, resulta técnicamente imposible *personalizar* una aplicación a las preferencias, características, contexto, etc., de cada potencial usuario. En este sentido son los desarrolladores quienes definen, diseñan e implementan los mecanismo de

adaptación de sus aplicaciones. De aquí surge directamente una limitación: las adaptaciones contempladas son finitas y no necesariamente satisfacen los requerimientos de todos los usuarios.

Peor aún, muchas aplicaciones incluso no permiten siquiera un mínimo grado de personalización y ante tal panorama los usuarios han buscado alternativas para personalizar su experiencia en la Web. Aunque parezca irreal no lo es, no es casual la proliferación actual de todo tipo de plug-ins para Browser Webs orientados a adaptar de una u otra manera los sitios Web que los usuarios visitan.

Desde el punto de vista de la diversidad de aplicaciones Web, que ya fue mencionado en el capítulo anterior, el problema radica en que un tarea determinada en el uso actual de la Web puede involucrar varias de dichas aplicaciones, ya que las tareas mas abstractas como *planificar un viaje* pueden implicar el uso de mas de una de ellas debido a la gran plataforma de servicios que hoy la Web contiene.

En este sentido, es muy dificultoso prever desde el punto de vista de los desarrolladores de una determinada aplicación (supongamos, una aplicación para la reserva y compra de pasajes aéreos), la integración con cualquier otra aplicación afín (supongamos, una aplicación para la reserva de habitaciones en hoteles). Dos usuarios probablemente utilizarían conjuntos de aplicaciones diferentes, cuando no disjuntos, en pos de llevar a cabo la misma tarea. La diversidad de aplicaciones Web con las cuales una subtarea puede ser realizada hace técnicamente inviable la integración de una de esas aplicaciones con cualquier otra que el usuario puede estar también navegando para completar otra subtarea relacionada. De hecho, las aplicaciones son *inconscientes* de que el usuario puede estar navegando otra aplicación. Hay excepciones, pero eso son: excepciones. Por ejemplo, la integración de cualquier sitio Web con redes sociales hoy en día es moneda corriente. Además existen alianzas estratégicas entre distintas empresas para facilitar la integración de sus productos. En cualquier caso, eso son: excepciones. La regla general es que una aplicación Web cualquiera no contempla al usuario como usuario de otras aplicaciones. En el uso actual de la Web significaría, bajo la hipótesis de este trabajo, desconocer la tarea del usuario. La tarea como algo mas abstracto que el uso de una única aplicación Web no es considerada.

En trabajos ya publicados, tal falta de consideración que termina significando una falta de integración de las aplicaciones Web, se ha explicado en términos de

navegación intra-aplicación versus navegación inter-aplicación Firmenich et al. [2010b]. Como ya se ha explicado, una navegación intra-aplicación tiene lugar cuando el usuario navega un hiperenlace desde un nodo A hacia un nodo B, siendo que ambos (nodos A y B) pertenecen a la misma aplicación. Por el contrario, una navegación inter-aplicación es realizada cuando en una navegación del usuario es entre nodos A y C, siendo que estos nodos pertenecen a distintas aplicaciones. En este punto de la exposición el detalle relevante (el resto será profundizado en siguientes subsecciones de este mismo capítulo) es que al realizar la navegación hacia el nodo A, la aplicación no puede determinar que lo hizo desde un nodo C de otra aplicación: no lo sabe, lo desconoce, y esto va en detrimento de las decisiones de adaptación que la aplicación dueña del nodo A puede tomar. En contrapartida, si la navegación hubiese sido desde el nodo B (perteneciente a la misma aplicación) entonces si pudo haber tomado acciones de adaptación al respecto.

Al final de este breve análisis nos queda la sensación de que la integración de mas de una aplicación controladas por diferentes grupos de desarrolladores no es trivial, y que sin capacidades de integración, no pueden realizarse medidas de adaptación basadas en el uso de dichas aplicaciones.

El modo en el cual se utiliza la Web hoy en día, es decir, el modo en el cual se consumen servicios de todo tipo y provenientes de un conjunto interminable de aplicaciones, y como se quiso ilustrar, el modo en el cual se utiliza un conjunto de aplicaciones para que la suma de ellas sea la que le permite a los usuarios realizar una determinada tarea, hace que la falta de adaptación de una aplicación que considere al resto de las aplicaciones repercuta en la experiencia del usuario. Y aunque parezca menor, cuando se va en decremento de la experiencia del usuario, se va en detrimento de la calidad en uso, digamos, calidad de uso no de una aplicación, sino calidad de uso de la Web.

Con mecanismos que funcionen en el cliente y que cumplan el rol de *integradores* se trata en este trabajo de adaptar, de personalizar, la experiencia en la Web o bien la *navegación* de la Web, mediante la adaptación de las aplicaciones que el usuario en cuestión utiliza.

2.3. Concern-Sensitive Navigation

En este punto se ha creado una subsección aparte para explicar el concepto de navegación sensible al *concern* (CSN: Concern-Sensitive Navigation) Nanard et al. [2008], como un mecanismo de adaptación atada al *concern* navegacional del usuario. En esta herramienta conceptual se ha basado gran parte de este trabajo.

Según Jr. and Rouvellou [2002] se define un *concern* como "materia de consideración en un sistema de software". Un *concern* puede reflejar aspectos funcionales de una aplicación como subir Videos en Youtube, buscar productos en Amazon, etc. Existen también *concerns* no funcionales, como usabilidad o accesibilidad, que aunque son críticos en aplicaciones Web no son objeto central de estudio en este trabajo.

Un *concern* puede ser genérico, cuando éstos aparecen en un amplio número de aplicaciones (por ejemplo: soportar log-in); o bien específico de dominio, cuando solo son aplicados en un conjunto particular de aplicaciones (por ejemplo, características de generación de caminos en aplicaciones como Google Maps). Los *concerns* pueden ser definidos abstractamente durante el desarrollo de la aplicación (por ejemplo, en una enciclopedia como Wikipedia, manejar categorías de los artículos) para luego ser instanciados por usuarios (interesados en categorías concretas).

Un *concern* navegacional es un *concern* de la aplicación que afecta a la navegación, es decir, que este repercute en la estructura navegacional de la aplicación, ya sea en los contenidos exhibidos, en las operaciones disponibles y links mostrados, etc.

Volviendo a la analogía con los escritos de Borges, en el cuento citado queda bien planteado que aunque diferentes senderos pueden llevarte a la misma intersección, al mismo punto, al llegar a éste cada uno es alguien distinto marcado por las experiencias de los senderos caminados previamente: "*...los senderos de ese laberinto convergen; por ejemplo, usted llega a esta casa, pero en uno de los pasados posibles usted es mi enemigo, en otro mi amigo...*" Borges [1944]. La analogía, claramente, viene dada por el hecho de que un usuario puede llegar a un sitio Web luego de haber seguido diferentes caminos navegacionales y al llegar al sitio Web el camino seguido tiene un peso propio, puede marcar un interés (un

concern) distinto al que hubiese tenido si seguía otro camino. El sitio Web podría, en consecuencia, ser otro también: podría estar enmarcado en ese momento actual del usuario.

2.3.1. Definición

Como se ha establecido en Nanard et al. [2008] y Firmenich et al. [2010b], una aplicación Web soporta CSN cuando los contenidos, hiperenlaces y la funcionalidad exhibida por las páginas de la aplicación no son fijas para diferentes caminos navegacionales, sino que, por el contrario, pueden alterarse cuando son accedidas en el contexto de diferentes *concerns* navegacionales.

Para hacer práctica y concreta la idea de CSN, se asume que los usuarios navegan a través de objetos navegacionales, es decir, los nodos hipermedia de la aplicación.

Entonces, supongamos un objeto Nj, instancia del objeto de navegación tipo N. Mientras en una navegación convencional el objeto Nj mostrará el mismo contenido e hiperenlaces sin importar cómo este objeto fue accedido, en CSN sus propiedades pueden ser levemente ajustadas de acuerdo con el *concern* actual del usuario, como se muestra en la Figura 2.1. La Figura 2.1 muestra a la izquierda la versión normal de Nj (con dos atributos), mientras a la derecha el mismo objeto como es accedido con un *concern* C presenta propiedades adicionales. Esta misma idea genérica es mostrada en la Figura 2.2 con un ejemplo concreto.

Figura 2.1: Nodo genérico

En pos de dar un punto de vista mas profundo a la noción de CSN, a continuación se explicará la diferencia entre la especificación de un objeto de navegación normal y de un objeto de navegación sensible al *concern*, es decir, de un objeto de navegación que soporta el hecho ser accedido bajo el concepto de CSN. Como

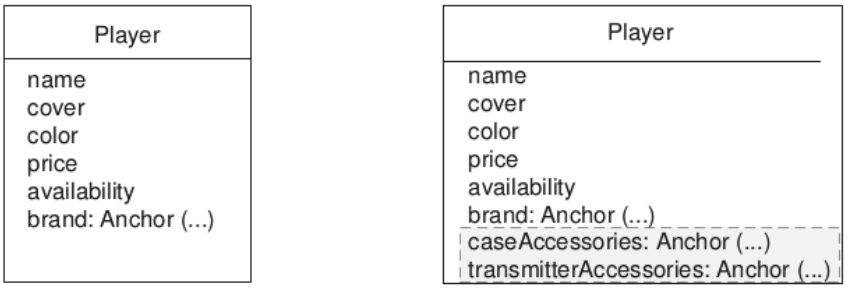

Figura 2.2: Nodo de un reproductor de audio

se mencionó, esto es posible mediante la separación de las propiedades intrínsecas al objeto de aquellas que son relevantes bajo el rol que el objeto juega bajo cierto *concern* en relación a otros objetos. El concepto de *rol* tiene variadas significados semánticos en la literatura, aquí se utilizan las definiciones de Kristensen and Østerbye [1996], además adoptadas por Riehle and Gross [1998].

En el ejemplo de la Figura 2.2, el *PlayerInIpodList* especifica aquellas propiedades que pertenecen a un reproductor cuando es accedido desde un listado, con otras palabras: cuando está siendo navegado con el *concern* del listado de productos. Similarmente, por cada posible *concern* en el cual el nodo correspondiente al reproductor puede ser accedido, podría utilizarse el concepto de CSN, entonces, debemos especificar el tipo de rol correspondiente. Por ejemplo, los reproductores muestran propiedades adicionales cuando son accedidos desde un listado de Dispositivos de Audio: un texto que describa por qué el producto está aconsejado, un set de hiperenlaces a productos de audio similares, etc. En este caso el tipo de rol *PlayerInAudibleDevice* definirá estas propiedades.

Resumiendo, un nodo de navegación de tipo N (por ejemplo *Player*) exhibe:

- Propiedades intrínsecas del tipo de objeto
- Propiedades que, dado un *concern* Ci (por ejemplo, Player*In*IpodList), corresponden a ser perceptibles cuando el objeto N es accedido en el *concern* Ci, es decir, N*in*Ci. Además, estas propiedades están expresadas utilizando notación de roles.

La Figura 2.3 muestra cómo con la notación de roles se modelan las propiedades del objeto *Player* relativas a lo dos *concerns*: *Accessories* y *AudibleReadyDevice*.

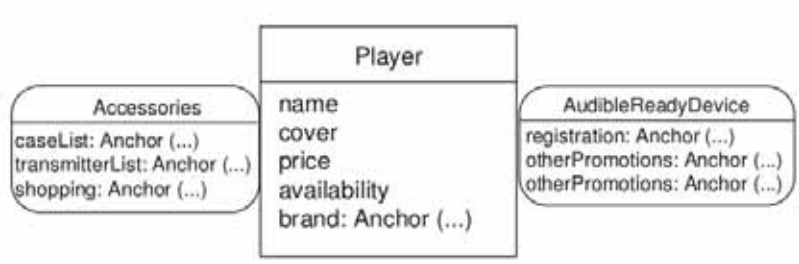

Figura 2.3: Distintos *concerns* del objeto *Player* con notación de roles

Alineando las propiedades de un objeto de navegación con el *concern* en el que fue accedido, puede mejorarse la estructura navegacional de la aplicación haciendo que los hiperenlaces y los contenidos estén mas focalizados al interés con el cual el usuario está navegando. La Figura 2.4 muestra un esquema navegacional simplificado de una aplicación en el que se muestra cómo los roles son utilizados para indicar el acceso al nodo bajo un *concern*, por ejemplo cuando se navega desde *IpodAccessories* entonces *PlayerInIpodList* es el rol utilizado. Debe notarse también que existe navegación *plana* desde el *Home* hacia *Player*.

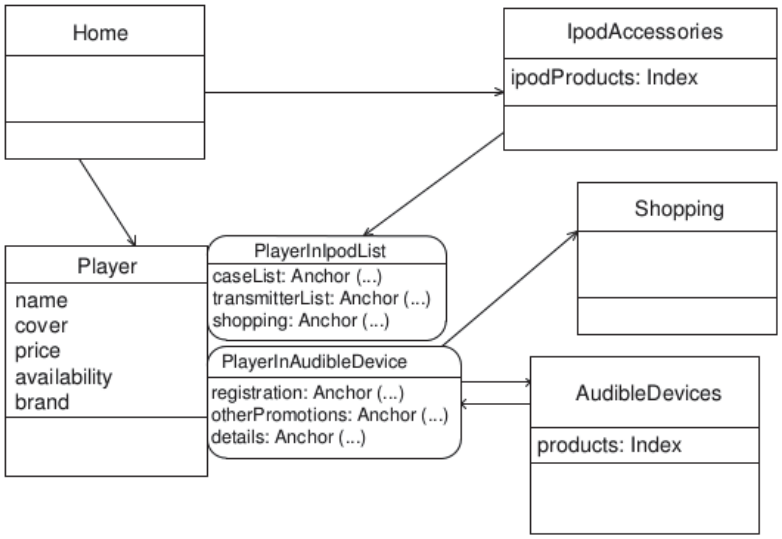

Figura 2.4: Esquema genérico de navegación plana vs. navegación Concern-Sensitive

2.3.2. Patrones de enriquecimiento y tipos de *concern*

Para usar la herramienta conceptual que nos brinda CSN correctamente y así maximizar las ventajas es necesario entender cómo un nodo de una aplicación puede ser enriquecido cuando es accedido en diferentes *concerns*. A continuación se muestra una lista de tipos de enriquecimientos modelados con roles (los tipos de enriquecimientos pueden ser combinados):

- Agregar o modificar contenido: un rol puede agregar atributos al objeto, o eventualmente puede cambiar el valor de un atributo específico. En el rol *PlayerInAudibleDevice*, el nodo es enriquecido con nuevo contenido que describe los términos de la oferta.

- Agregar o modificar *anchors* y *links*: un rol puede agregar *anchors* al objeto navegacional; eventualmente puede también redefinir hacia dónde se navega con el *link*. Como un ejemplo, los *links* salientes provistos por el rol *PlayerInAudibleDevice* introduce un nuevo camino de navegación hacia los nodos *AudibleDevice* y *Shopping*.

- Agregar o modificar operaciones: los roles pueden mejorar el conjunto de acciones o eventualmente modificar un comportamiento específico. Nuevas operaciones simplifica la experiencia del usuario mediante la agregación de servicios que corresponden al *concern* actual.

Un uso disciplinado de este enfoque y de los enriquecimientos basados en roles implicaría limitar los atributos, *links* y operaciones a aquellos que son válidos siempre, es decir, sin consideración de cómo el usuario navegó hasta el nodo en cuestión.

Aunque el tipo de enriquecimiento que se necesita utilizar dado un *concern* y un objeto particular claramente depende de aplicaciones específicas, podemos caracterizar los patrones de enriquecimientos mas usuales si se analizan los diferentes tipos de *concern* que surgen usualmente en las aplicaciones Web. De este modo, se pueden proveer guías básicas y buenas prácticas para un fructífero uso de CSN.

A pesar de la variedad de *concerns* que pueden surgir en el desarrollo de una aplicación Web, algunos de estos *concerns* tienen un impacto en la navegación

(lo cual significa que el usuario puede acceder estos objetos navegacionales en el contexto de un *concern* en particular), y por lo tanto estos pueden ser una fuente de información para mejoras sensibles al *concern* del usuario.

A continuación se resumen los mas importante y recurrentes *concerns* con los que nos podemos encontrar en aplicaciones Web; además, por cada uno se indican los enriquecimientos mas usuales respectivos. Se llama a esto, patrones de enriquecimiento ya que pueden ser expresados en forma genérica e instanciados para cada caso de uso particular.

- ***Concern basado en tarea***: La mayoría de las aplicaciones Web permiten al usuario realizar alguna tarea (explorar productos, manejar el carrito de compras, reservar, actualizar contenido, ingresar comentarios, etc.). Algunas de estas tareas pueden involucrar más de una página Web para ser completada, por ejemplo, reservar una habitación de un hotel. Se requiere, claramente, ayudarlo al usuario a avanzar en el desarrollo de esta tarea.

  Enriquecimiento: Cuando un *concern* es definido por una tarea o un proceso de negocio, y si operar con el nodo correspondiente puede traer conflictos con el proceso, es aconsejable o bien eliminar aquellas operaciones que tienen conflictos con el *concern* o agregar avisos específicos para alertar al usuario. Cuando la tarea es inter-aplicación (por ejemplo, involucra una navegación de Facebook a Flickr) sería prudente agregar en la aplicación final de la navegación una indicación de operaciones relacionadas con la aplicación inicial desde la que se inició la navegación (por ejemplo, agregar una foto de la página de Facebook en la de Flickr).

  En la Figura 2.5 se muestra un ejemplo de instanciación de este patrón para el proceso de *checkout*. El nodo *Product* es accesido en el *concern check-out* (supongamos que el usuario quiere confirmar alguna propiedad específica del producto). El rol *InCheckout* contiene un atributo con una indicación del *concern* y un link para volver al proceso y la redefinición de la operación *addToCart* (ya que el producto actualmente ya esta dentro del carrito de compras).

- ***Concern basado en tópicos***: sitios de pura información pueden introducir incluso *concerns* de granuralidad mas fina, por ejemplo, tópicos o temas

como en una enciclopedia. *Concerns* basados en tópicos están también presentes en el contexto de tareas, por ejemplo durante la búsqueda de libros en Amazon: el género del libro (técnico, viajes, etc.) o el área de la temática (Ingeniería de Software, Programación, etc.) pueden convertirse en un *concern*.

Enriquecimiento: Agregar información y links específicos hacia el tópico relacionado al *concern*. La Figura 2.4 ilustra un ejemplo de *concern* basado en tópico, en este caso, los tópicos son "Audible Devices" y "iPod Accessories" tienen relacionados diferentes roles que agregan diferentes informaciones.

En la Figura 2.6 se muestra otro ejemplo basado en una guía turística interactiva. El nodo *Cathedral* puede ser accedido en tres diferentes *concerns*, representados por los roles *History*, *Architecture* y *PlaceOfWorship*. Lo destacable en este ejemplo, es que cada uno de los roles especifica distinta información, como ser en el caso del rol *PlaceOfWorship* la información referida a las misas y descripta por la propiedad de ese rol llamada *massSchedules*.

- ***Concerns navegacional puro***: como los tours guiados o conjuntos de nodos, que son abstracciones en el diseño navegacional, y por lo tanto pueden ser considerados también *concerns* específicos. Un claro entendimiento de estos *concerns* permite mejorar la navegación a través de los nodos pertenecientes al mismo conjunto.

  Enriquecimiento: Cuando el *concern* puede ser representado como un conjunto como con los contextos navegacionales de OOHDM Rossi and Schwabe [2008], es deseable enriquecer el nodo actual con links hacia el índice del conjunto actual, además de a los nodos anterior y siguiente del mismo conjunto. Un caso usual se presenta en aplicaciones con navegación basada en *tags* como en Flickr, donde desde el nodo de una foto se proveen links hacia otras fotos cuyos *tags* sean parecidos.

  En la Figura 2.7 se muestra un ejemplo para una guía turística de pinturas. El nodo principal, *Painting* con el contenido básico es enriquecido con un rol *InGuidedTour* que incluye los links para navegar hacia otras pinturas

(la anterior y la siguiente en la guía) o bien hacia el índice.

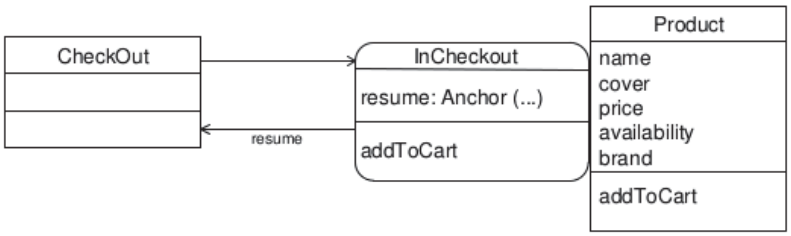

Figura 2.5: Aplicación del patrón *Task concern*

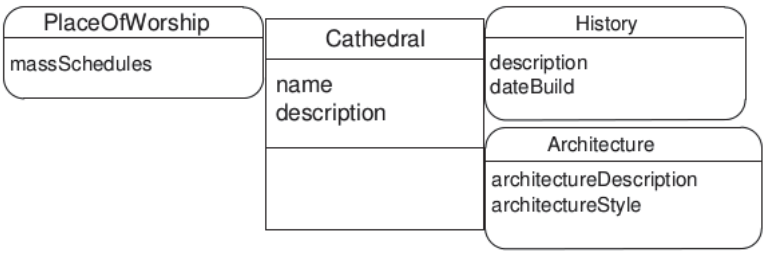

Figura 2.6: Aplicación del patrón *Topic concern*

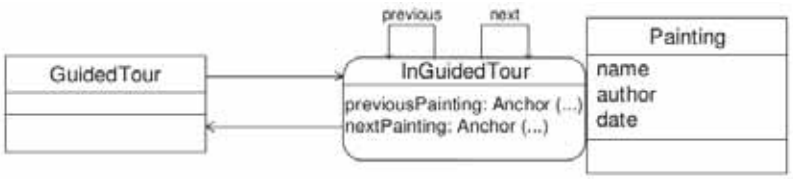

Figura 2.7: Aplicación del patrón *Pure Navigational concern*

2.3.3. Discusión

Como herramienta conceptual CSN puede ser aplicada durante el proceso de desarrollo de las aplicaciones Web, sin embargo, no es el foco de esta tesis. Para el lector interesado en el proceso de diseño involucrando CSN se recomiendan dos artículos: Nanard et al. [2008] y Firmenich et al. [2010b].

Existen actualmente buenos ejemplos de CSN en la Web. Por ejemplo en la Figura 2.8 el mismo producto publicado en Amazon es accedido habiendo

seguido distintos caminos de navegación. A la izquierda, el reproductor de MP3 contiene información acerca de ofertas especiales mientras que a la derecha, en la misma ubicación, se muestra información relacionada a accesorios compatible con el producto, en este ultimo caso porque el producto fue accedido de el listado "iPod y accesorios".

Figura 2.8: Ejemplo de CSN en Amazon.com

De todas maneras, no es fácil encontrar este tipo de ejemplos y además no se suele explotar totalmente lo que CSN como herramienta conceptual ofrece. No es difícil encontrar (fundamentalmente en sitios de turismo, museos, etc.) guías turísticas con navegación basada en conjuntos de hiperenlaces [1]. Este concepto de navegación basada en conjuntos de hiperenlaces es, claramente, lo que se conoce como contexto navegacional en OOHDM Rossi and Schwabe [2008].

En general, en estos enfoques los *contextos navegacionales* son manejados dentro de la misma aplicación. Aunque contemplar este tipo de navegación mejora la experiencia del usuario, no es cierto que la navegación del usuario este acotada a una sola aplicación. Actualmente el usuario navega entre aplicaciones diversas, como ya se ha mencionado, y aplicar el concepto de CSN cuando la navegación es inter-aplicación no es tan simple. En este sentido, mediante la aplicación de técnicas de adaptación en el cliente sí puede reconocerse cuando un usuario navega entre nodos de aplicaciones distintas.

[1] Tour guiado en Museo Roman Open Air Museum. http://www.villa-rustica.de/tour/toure.html

2.4. Adaptación del lado del cliente

Habiendo hecho un repaso sobre la historia de sistemas hipermedia y el inicio de la adaptación en los mismos, y habiendo presentado brevemente los mecanismos clásicos de adaptabilidad, en esta sección se define finalmente uno de las bases teóricas fundamentales de esta presentación: la adaptación del lado del cliente.

La presente sección comienza con básicas pero relevantes definiciones, mientras que el resto de la misma se focaliza en el potencial de este tipo de enfoque, a la vez que se define una taxonomía de adaptaciones.

2.4.1. Definiciones

En primer lugar, resulta indispensable definir qué es el *cliente*. Desde el punto de vista de la arquitectura cliente-servidor, el cliente es cualquier componente de software que solicita un contenido o servicio a un servidor y a partir de la solicitud el servidor provee algún tipo de respuesta. En el contexto de las aplicaciones Web sujeto a estudio el cliente será el Browser Web, es decir, el software que utiliza el usuario para acceder a las aplicaciones (realizar peticiones a los servidores de las aplicaciones Web). A su vez, la respuesta relevante de los servidores son los documentos HTML (*HyperText Markup Language*) que terminan siendo, luego de ser cargados y procesados por el Browser Web, lo que el usuario percibe de la aplicación, lo que ve y con lo que interactúa.

Utilizar el Browsers Web como software base para la adaptación no es la única configuración. Una alternativa para la modificación de contenido servido por servidores ajenos son los transcodings. Transcodings son sistemas que, generalmente mediando entre el cliente real y el servidor (funcionando como *proxy*), modifican el contenido antes de ser recibido por el cliente, de tal forma que desde el punto de vista del cliente el funcionamiento es un tanto mas transparente que con sistemas basados en el cliente en si mismo Bharadvaj et al. [1998].

Los sistemas de transcodings son utilizados con diversos objetivos. A pesar de ser muy utilizados sobre *streamings* de video, también existen enfoques orientados a modificar los HTMLs originales devueltos por las aplicaciones Web. Estos últimos mayormente orientados a adaptar el contenido para dispositivos de pantallas pequeñas o bien para mejorar la accesibilidad. Normalmente, los transcodings no

proveen aspectos de personalización, y aquellos enfoques que sí, caen en modelos de usuarios con idénticas limitaciones a las mencionadas previamente y referidas a los modelos de usuarios de aplicaciones Web. Además, los sistemas de transcodings de ninguna manera están orientados a integrar diversas aplicaciones Web. Por lo dicho, y aunque se volverá con algunas pequeñas consideraciones en pasajes específicos de esta presentación, los transcodings no son foco de estudio ni comparación, de modo tal, que se retoma a continuación el Browser Web como ámbito de adaptación de las aplicaciones utilizadas.

Aunque existen diversas tecnologías para modificar y/o manipular archivos HTML, lo común y ampliamente mas conocido es un lenguaje de *scripting* llamado JavaScript. Al cargar el HTML de un sitio en el Browser Web, este proporciona un DOM (***D**ocument **O**bjet **M**odel*) que básicamente es una API de acceso y manipulación de los elementos descriptos en el HTML. Desde un script JavaScript, es sumamente sencillo utilizar el DOM y además (desde el punto de vista de quién puede desarrollarlo) es un lenguaje de programación muy difundido, lo que lo hace una alternativa tentadora para cualquier esquema de adaptación en el cliente, tal es así que es la alternativa mas utilizada con este fin, incluso a partir del cual se han generado DSLs (***D**omain **S**pecific **L**anguage*) con algún grado mayor de abstracción como veremos en el capítulo sobre el estado del arte y trabajos relacionados.

2.4.2. Características de la adaptación del lado del cliente

2.4.2.1. End-User Programming

A este punto de la presentación ya deberían estar claras las diferencias entre un enfoque de adaptación *server-side* vs. *client-side*, utilizando el término *client-side* como el uso de técnicas aplicadas en el cliente para adaptar contenido de terceros, es decir, sin contemplar lo que sería la adaptación *client-side* propietaria (aquella basada en scripts autocontenidos en los documentos devueltos por la aplicación) que hace ya varios años existe, de hecho, es uno de los pilares de aplicaciones Web 2.0 y RIAs (***R**ich **I**nternet **A**pplication*).

Cuando éste es el ángulo desde el cual nos paramos para establecer la comparación surge inmediatamente la diferencia de quién es el responsable de definir

las adaptaciones. *Server-side*: los desarrolladores de las aplicaciones, *client-side*: cualquier usuario con capacidades o habilidades para desarrollar un artefacto de software que corra en el cliente. Poner tal poder en las manos de los usuarios no hubiese significado mucho en la década de los 90: pocos usuarios tenían capacidad de generar software por su cuenta. Sin embargo, y aunque la ciencia de la computación solo parece alcanzar a gente de la academia, la informática, el hardware, el software, en el estado mas puro y de la manera mas básica han ido alcanzando un nivel de masividad incomparable con el de décadas pasadas y nada indica que esta tendencia fuese a revertirse. Este aspecto social involucra, por supuesto, a toda una masa de aficionados que aunque sin necesariamente pertenecer a la academia saben como utilizar, configurar e incluso extender numerosas herramientas con múltiples niveles de complejidad. En el fondo nadie sabe que nivel de formación académica tiene la comunidad de usuarios desarrolladores de scripts para GreaseMonkey. Lo que se sabe es que existe una gran cantidad de ellos que han desarrollado una gran cantidad de scripts y que estos scripts han incorporado características tan importantes en las aplicaciones Web al punto de que los desarrolladores de las mismas han ido incluyéndolas como características propias de la aplicación. La importancia de estas nuevas características queda automáticamente definida a partir de la gran cantidad de usuarios que han instalado esos scripts en sus propias computadoras. Esto último sigue la línea anterior: los usuarios saben como configurar y/o utilizar herramientas no tan triviales y de aquí un corolario: los usuarios quieren modificar las aplicaciones agregándole o modificándole características no provistas originalmente.

No es una aseveración caprichosa: si uno busca artículos científicos (sin hacer un análisis cualitativo, sino absolutamente cuantitativo y además poco exhaustivo: solo buscando en Google Scholar) que consideren (es decir, que al menos mencionan) lo que se conoce como *end-user programming* el resultado puede sorprendernos. En la década de los ochenta, pueden encontrarse 105 resultados. En la década de los noventa el número de resultados crece a 678. En la década iniciada en el año 2000 podemos encontrar 2520 resultados. Y finalmente solo en los casi dos años que van de la década del 2010 encontramos 1090 resultados. No es una aseveración caprichosa, es una tendencia aceptada desde la academia y que, por supuesto, se ha profundizado a lo largo de los años porque ha encontrado su

contrapartida fuera del ámbito académico. Debe notarse que los números mencionados incluyen también artículos que no son referidos a la Web. Para dar una idea, si solo tenemos en cuenta artículos relacionados con la Web, la cantidad de artículos en este área de los últimos casi dos años se reduce de 1090 a 827[1].

Retomando la idea de cómo características desarrolladas por la comunidad de usuarios han sido incorporadas nativamente en las aplicaciones, podemos citar varios scripts que lo demuestran. Solo a modo de ejemplo, se describen 3 scripts que adaptaban Gmail y cuyas funcionalidades fueron incorporadas por la aplicación [1]:

- Gmail Delete Button[2]: Curiosamente, el botón "Eliminar", para eliminar los emails seleccionados de una bandeja, no estaba originalmente en Gmail. Esta funcionalidad era posible gracias a este script que adaptaba Gmail una vez cargada en el Browser Web. Este script fue desarrollado en el año 2005, y 26.000 veces había sido instalado por usuarios hasta que el equipo de Gmail agregó este botón en la aplicación.

- Gmail + Google Calendar[3]: Un script para integrar dos aplicaciones de la misma empresa, Gmail y Google Calendar. Hoy en día esta integración es proporcionada por las misma aplicación a través de plug-ins específicos. Este script fue desarrollado en el 2007 e instalado 8.000 veces.

- Gmail Chat Right Side[4]: Un simple script para reordenar el *layout* de Gmail que permitía ubicar la lista de contactos del chat embebido al lado derecho del listado de e-mails. Hoy en día, al igual que el anterior, esto puede activarse desde el mismo panel de configuración de Gmail. Script desarrollado en el 2006 e instalado 3.717 veces.

Los datos brindados, tanto acerca de la producción científica como de la producción de software ad-hoc, no hacen otra cosa que contribuir a la comprensión del fenómeno real que sucede para modificar, personalizar por parte de los usuarios (incluso a espaldas de los desarrolladores de las aplicaciones) las aplicaciones.

[1] Datos actualizados al 22 de Octubre del 2012
[2] http://userscripts.org/scripts/review/1345
[3] http://userscripts.org/scripts/show/9411
[4] http://userscripts.org/scripts/show/4665

En el caso de la Web, esto es un tanto mas factible desde que la gran mayoría de los sitios Web utilizan lenguajes estandarizados para los contenidos publicados (HTML, etc.), con lo cual, también pueden establecerse modos *comunes* para adaptarlos a la vez que se difunden estos modos. Claro que sistemas de escritorio, como Microsoft Office Word, pueden ser tanto o mas utilizados que cualquier aplicación Web. Seguramente también tengan aspectos que un usuario quiera modificar o adaptar pero el sistema no se lo permite. Esta es la clave: la masividad de usuarios no es suficiente, tiene que ser posible adaptar el software a bajo costo, por lo dicho la Web es un ámbito ideal.

Es tan así que sucede lo descripto: los usuarios desarrollan artefactos por su cuenta para modificar las aplicaciones Web y estos artefactos pueden alcanzar un éxito tan grande que las aplicaciones terminan incorporando estas nuevas características. Pero la clave es la siguiente: los usuarios tienen sus propios requerimientos, muchos de ellos son capaces de buscar soluciones *ad-hoc* para lograrlos y el ámbito de la Web se los permite.

La diferencia entonces radica en que si se le da al usuario el poder de *desarrollar* sus propios artefactos, los usuarios podrán satisfacer requerimientos que los desarrolladores tal vez nunca habrían podido imaginarse. Es el poder de dar lugar a la imaginación y a la necesidad de cada uno de los usuarios para mejorar una aplicación Web.

Desde el punto de vista técnico, como ya se ha mencionado y se discutirá en la siguiente subsección, existen algunas ventajas al utilizar el cliente como contexto de adaptación. Ventajas que radican principalmente en conocer el total de la actividad del usuario.

2.4.2.2. Potencial, aplicabilidad y factibilidad

Como en todo, hay varias ventajas y desventajas de un enfoque de adaptación en el cliente. En esta sección destacaremos aquellos puntos positivos, mientras que en la última subsección de este capítulo se abarcan las principales problemáticas del área.

Uno de los potenciales de la adaptación del lado del cliente es el hecho de poder contemplar toda la actividad del usuario y que los datos sobre esa actividad, sobre

esa interacción, pueden ser utilizados para adaptar las aplicaciones utilizadas. Veamos una pequeña lista descriptiva. Toda la interacción que el usuario realiza con cualquier aplicación, desde el uso del *mouse*, cada click del mouse, etc., puede ser utilizado para adaptar esa misma aplicación o, y he aquí la gran diferencia, cualquier otra aplicación. En el cliente, podemos registrar cada dato ingresado por el usuario en una aplicación, y ese dato puede ser utilizado en cualquier otra aplicación. En el cliente, se conoce todo el historial de navegación del usuario y no solo el sub-historial referido a una sola aplicación (esta es la visión recortada del historial de navegación del usuario que puede verse desde una aplicación). En el cliente, podemos extender los modos normales en los que el usuario maneja información relacionada con sus tareas. En el cliente, podemos saber cuáles son todas las aplicaciones en uso. En el cliente, podemos manipular lo que el usuario ve y percibe, y acomodarlo de la manera que el usuario prefiera.

En el cliente puede ponerse el poder en manos de los usuarios para que realicen modificaciones que de ninguna manera la aplicación originalmente contempla. Todo esto, bajo una filosofía de *crowdsoursing* Nebeling et al. [2012]: la misma masa de usuarios desarrolla, comparte, distribuye, utiliza y perfecciona los artefactos conseguidos.

En este trabajo no se descubre el la adaptación en el cliente de aplicaciones de terceros. En este trabajo se propone un enfoque que basado en adaptaciones realizadas en el cliente se soportan las tareas de los usuarios. Esto, en el trasfondo, significa que la adaptación del lado del cliente, aunque es una tendencia novedosa, no es nueva como tal. Desde ya hace varios años existen innumerables herramientas con una gran comunidad de usuario que demuestra no solo la factibilidad de un enfoque en el cliente sino ademas el interés de este tipo de herramientas en la masa de usuarios.

Una vez más, GreaseMonkey es el ejemplo emblemático: un motor de *client-side scripting* para el cual la masa de usuarios ha desarrollado no menos de noventa mil *scripts* para adaptar sitios Web (agregando y/o modificando contenido y funcionalidad), y de los cuales los mas utilizados por los usuarios han sido instalados mas de diez millones de veces. Una comunidad sumamente activa: en el preciso momento en el que se escribe este párrafo el código de un script acaba de ser actualizado hace un minuto, y no menos de una centena de ellos han

sido actualizados en las seis horas previas[1]. Esto marca un ritmo vertiginoso y un desarrollo incremental casi imposible de seguir desde una aplicación sin la masa de colaboradores que tiene una plataforma como ésta.

En la siguiente subsección se muestran ejemplos y se define una taxonomía (relevante para caracterizar adaptaciones y fijar un vocablo relevante en este trabajo).

2.4.3. Tipos de adaptaciones posibles

Lo que uno puede modificar de una aplicación de terceros son los documentos que esta nos brinda. Los mas relevantes aquí son los documentos que representan la *cara visible* de la aplicación: los documentos HTML y todos los utilizados por uno de ellos (imágenes, archivos de estilos - CSS -, etc.). En cualquier sentido en esos archivos los elementos presentados (es decir cada subitem de contenido) puede ser eliminado o modificado para agregar nuevos, todo esto realizando manipulaciones sobre el DOM.

Así es que en general puede eliminarse, agregarse o modificarse los elementos que componen los DOMs, lo cual significa en términos generales:

- Agregar, modificar o eliminar contenido.
- Agregar, modificar o eliminar funcionalidad.

Esta diferenciación es explicada y desarrollada a continuación, sin embargo es importante dejar en claro que si uno hace un leve relevamiento, encontrará que lo más común es agregar, augmentar tanto contenido como funcionalidad, seguido tal vez por la modificación de contenido.

Respecto del contenido

Contenido será tomado como la información hipermedia que el usuario percibe de una aplicación Web, lo cual incluye: texto, sonido, imagenes, videos, etc.

El ejemplo mas claro (y además los mas utilizados por los usuarios) de eliminación de contenido es la eliminación de publicidad. Existen numerosos scripts y plug-ins que se dedican a remover particularmente publicidad.

[1] http://userscripts.org/

Respecto a la modificación, generalmente esta asociada a la presentación del mismo: cómo el usuario ve el contenido. Así es que en realidad en lugar de eliminar o agregar elementos del DOM, suelen cambiarse sus propiedades, por ejemplo el estilo (color de fuente o de fondo, tamaño de texto, etc.). Es muy común también ver plug-ins de internacionalización para traducir los contenidos mostrados a otros idiomas.

Tal vez la adaptación de contenido mas buscada por los usuarios sea la agregación, la augmentación de la información original de una aplicación Web con otra información obtenida de otro nodo de la misma aplicación o bien de otras aplicaciones.

A modo de ejemplo, en la Figura 2.9 se muestra la adaptación realizada por un script de GreaseMonkey[1]. Básicamente, el script obtiene las imágenes resultantes de buscar en Bing[2] por el título del artículo de Wikipedia cargado y adapta el mismo insertando las imágenes mas relevantes según la búsqueda.

Figura 2.9: Ejemplo de modificación de contenido realizado mediante client-side scripting

Respecto de la funcionalidad

Aquí, la funcionalidad será contemplada como todo aquello con lo que el usuario puede interactuar: formularios, hiperenlaces, aspectos interactivos de aplicaciones RIA, es decir, todo lo que el usuario puede hacer con la aplicación para que la aplicación le brinde cierto feedback relacionado a dicha interacción, etc.

No es común ver artefactos de adaptación que eliminen funcionalidad en las aplicaciones utilizadas por los usuarios. Tal vez, lo mas frecuente es la desactivación, es decir la eliminación de hiperenlaces o bien la eliminación de funcio-

[1] Wikipedia (instalado 4.542 veces al 22/10/2012), http://userscripts.org/scripts/show/74601
[2] http://www.bing.com/?scope=images

nalidades de diferentes plug-ins de reproducción de contenidos multimedia. Para aquellas aplicaciones que utilizan *short-cuts* suelen verse también scripts que los eliminan para aquellos usuarios que no los prefieren.

La modificación de funcionalidad, tampoco común, suele verse generalmente en *scripts* que modifican las propiedades de los links o hiperenlaces para que el usuario rápidamente identifique y/o modifique ciertas características de los mismos. Por ejemplo un hiperenlace puede ser adaptado según varias condiciones: que abre una nueva pestaña/ventana, si el sitio Web (es decir, el sitio a cuál el usuario navegaría) es de la misma aplicación o no, o bien si el recurso es un recurso existente.

Ampliamente, lo mas buscado por los usuarios respecto a la funcionalidad es su augmetnación: agregar funcionalidad, mecanismos de interacción que originalmente la aplicación no provee. Hay scripts que agregan funcionalidad que realmente tienen una lógica compleja y que incluso requieren de persistencia de información para poder llevar a cabo sus objetivos. Este es el caso, por ejemplo, de un script de GreaseMonkey que aplica sobre Facebook y cuyo objetivo es darle a los usuarios la funcionalidad de identificar los "*unfriends*", es decir las personas en Facebook que habiendo aceptado la relación de amistad en la red social luego han sacado al usuario en cuestión de sus contactos[1]. Si miramos en la esquina superior derecha la Figura 2.10, que muestra la adaptación en cuestión, se verá una nueva opción "Unfriends"; si el usuario hace un click, se despliega la lista de estos ex-amigos. Claramente esta funcionalidad no es nativa de Facebook, al menos no a la fecha en la que se muestra este ejemplo. De todas maneras, no es menor la popularidad del mismo, que ha sido instalado por los usuarios alrededor de cuarenta y cinco millones de veces.

Modificaciones no perceptibles por el usuario

Aunque se alteren los documentos HTML originales, estos cambios no necesariamente repercuten de modo directo en lo que el usuario percibe. Existen herramientas que adaptan los DOM en pos de allanar el camino a otras herramientas. Este es el caso, por ejemplo, de herramientas y *scripts* de anotación que

[1] Unfriends (instalado 46.075.975 veces al 22/10/2012), http://userscripts.org/scripts/show/58852

Figura 2.10: Ejemplo de funcionalidad agregada mediante client-side scripting

agregan Microformatos a los sitios Web cargados en el Web Browser Firmenich et al. [2012]. De esta manera, otras herramientas que "buscan" Microformatos podrán realizar sus acciones con estas nuevas anotaciones.

2.4.3.1. Taxonomía de adaptaciones

Si bien la subsección anterior abarcó una descripción de los tipos de cambios que pueden realizarse sobre documentos DOM en el cliente, la descripción brindada nos habla del potencial en sí, pero no de cómo o en qué medida los cambios mejorarían la experiencia de los usuarios.

En esta subsección se presenta una taxonomía de adaptaciones posibles en el cliente basada en los tipos de cambios descriptos en la sección anterior. Taxonomía que no se ha definido en términos de qué cambios aplicar sobre el DOM, sino que a otro nivel de abstracción, define cuándo los cambios deberían ser aplicados y ademas cuales son los objetivos de los mismos, es decir, en que aspecto se intenta mejorar la experiencia del usuario.

Basadas en preferencias

Adaptaciones basadas preferencias son aquellas que serán ejecutadas sobre una aplicación cada vez que la aplicación es utilizada, es decir, que el usuario prefiere, dado un nodo A de una aplicación Web, utilizar A', que resulta de haber aplicado alguna adaptación. Aunque este hecho de ejecutar la adaptación *siempre* puede repetirse en otra categoría de la taxonomía las adaptaciones tienen objetivos

diferentes, los motivos por los cuales el usuario desea aplicarlas son realmente diferentes, como se explicará.

La Figura 2.11 muestra una adaptación de este tipo. Aquí, un *script* analiza las notas al pie de los artículos de Wikipedia y le agrega a las citas el evento MouseOver a partir del cual, cuando el usuario posiciona el puntero del mouse encime de una cita, le muestra la información de la cita correspondiente [1].

Figura 2.11: Esquema genérico de navegación plana vs. navegación Concern-Sensitive aplicado a Wikipedia

Debe notarse que el mencionado script de Wikipedia puede ejecutarse por que analiza el DOM particularmente de Wikipedia, lo cual es necesario porque no debe confundir cualquier hiperenlace con una nota al pie. Sin embargo, otros scripts realizan adaptaciones mas genéricas que se basan en adaptar, por ejemplo, todas las imágenes de cualquier sitio Web[2].

Accesibilidad

La forma inequívoca de aplicar las adaptaciones basadas en preferencias (es decir, cada vez que un sitio en particular sea utilizado) también puede utilizarse para atacar problemas concisos, por ejemplo, hacer mas accesible una aplicación. Desde el punto de vista de la ejecución de las adaptaciones, aquellas abocadas a la accesibilidad (se ejecutan siempre que una sitio Web objetivo es cargado en el Web Browser) podrían ser categorizadas también como adaptaciones estáticas o basadas en preferencias. Sin embargo la diferencia en el objetivo final de las adaptaciones permite diferenciar a las de accesibilidad como una categoría distinta

[1] WikipediaFootnotePopup (instalado 26.598 veces al 22/10/2012), http://userscripts.org/scripts/show/75187

[2] Mouseover Popup Image Viewer (instalado 89.320 veces al 22/10/2012), http://userscripts.org/scripts/show/109262

de adaptación, que claramente, es similar a la anterior desde el punto de vista del funcionamiento.

Adaptaciones orientadas a accesibilidad son sumamente comunes ya que a pesar del esfuerzo realizado desde el campo académico y de los estándares que varias organizaciones han definido, gran cantidad de aplicaciones Web no respetan tales recomendaciones a la hora de hacer de sus sitios Web, sitios accesibles. Los sistemas de transcodings, sin ir mas lejos, han ido lentamente siendo aplicados a la mejora de la accesibilidad de la Web Asakawa and Takagi [2008]. Además existen varios enfoques que funcionan en el cliente real. Por ejemplo *AccessMonkey* Bigham and Ladner [2007] es un motor de scripting similar a GreaseMonkey, pero con mecanismos que facilitan la generación de scripts para mejorar la accesibilidad brindando facilidades para realizar cambios de estilo y cierto grado de reorganización del contenido para que sean interpretado por los *screen readers* en un orden diferente al original.

Al margen de herramientas como AccessMonkey, de los sistemas de transcodings orientados a accesibilidad, e incluso de los estándares establecidos; existen otros aspectos sobre la accesibilidad en las aplicaciones que requieren adaptaciones mas profundas. Que una aplicación cumpla con estándares de accesibilidad, no garantiza que sea fácil de usar para un usuario, supongamos, no-vidente. Veamos un ejemplo. En la Figura 2.12 se muestra, a la izquierda, la versión original de GMail. Aquí puede observarse un listado de emails con tres operaciones que pueden realizarse una vez seleccionado uno o mas emails del listado. Aunque parece simple aplicar alguna de estas operaciones a algun elemento, para un usuario ciego no lo es tanto. Debe primero encontrar los emails, seleccionar aquellos a los que quiere aplicar una operación, luego volver navegando con el cursor hasta el menú con las operaciones y finalmente escoger y aplicar alguna Garrido et al..

A la derecha, en la misma Figura, se muestra una adaptación realizada sobre esta versión original de GMail, en la que cada una de las operaciones disponibles es distribuida a cada email en particular, así el usuario, luego de leer el remitente y el asunto puede elegir una operación directamente.

Figura 2.12: Adaptación de accesibilidad aplicada sobre Gmail.

Concern-Sensitive

En la subsección 2.3 se ha explicado el concepto de *Concern-Sensitive Navigation* como una manera de adaptar las aplicaciones Web en consecuencia del camino navegacional que el usuario ha seguido y que denota un interés en particular. Cabe destacar que aunque valiosa es esta herramienta conceptual, aplicarla en el diseño de las aplicaciones Web puede mejorar la experiencia del usuario dentro de los límites de la aplicación. No así cuando el usuario navega a un nodo de la aplicación desde otro el cual no pertenece a la misma. El concepto de CSN es aun mas rico cuando se contemplan navegaciones inter-aplicación, y esto es posible aplicando técnicas de adaptación en el cliente, desde que es en este ámbito donde se puede conocer todos y cada uno de los sitios Web visitados. En la Figura 2.13 se muestra un ejemplo de adaptación CSN basada en la navegación desde Delicious.com hacia Wikipedia.com. A la izquierda de la Figura, se muestra los resultados de búsqueda de *bookmarks* que un usuario realizo en Delicious. Uno de los resultados es un artículo de Wikipedia, que al navegarlo, el usuario accede al artículo correspondiente de Wikipedia que ha sido enriquecido con dos menús de navegación mostrados a la derecha. Uno de los menús (el superior) presenta otros links de Delicious que están relacionados al que el usuario previamente había navegado. El menú inferior, muestra links hacia los artículos de Wikipedia correspondientes a los *tags* del *bookmark* de Delicious.com.

Debe notarse que este tipo de adaptaciones no son técnicamente viables desde el servidor, ya que al realizar la petición del artículo a Wikipedia, la aplicación no puede determinar cómo el usuario llego a la URL correspondiente.

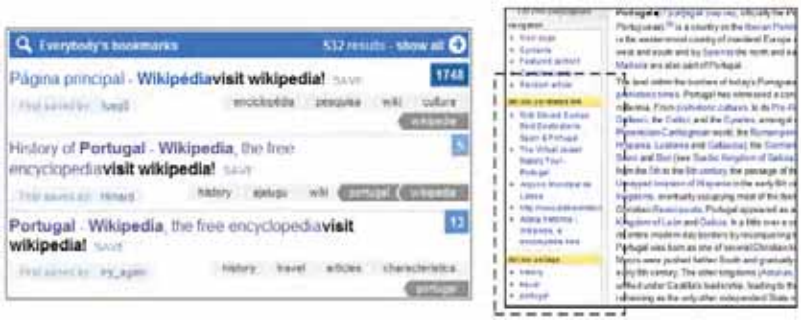

Figura 2.13: Adaptación CSN Inter-aplicación entre Delicious.com y Wikipedia.com

Sensible a la tarea del usuario

Existen adaptaciones, del estilo *task-aware*, que pueden realizarse en el cliente, que sin estar enfocadas en el camino navegacional en sí, sí están basadas en la interacción que el usuario tiene con diversas aplicaciones. En la Figura 2.14 se muestra una secuencia posible de uso a partir de la cual se dispara una adaptación sobre Wikipedia. A la izquierda, se muestra un usuario navegando por el artículo de Wikipedia referido a *Personalization*. Como segunda interacción del usuario, éste abre una pestaña para buscar en Google.com acerca de *adaptivity*. Finalmente, a la derecha de la Figura, se ve como al volver a focalizar el artículo de Wikipedia, éste fue adaptado agregando un link al artículo de *Adaptivity* de Wikipedia, siendo que éste fue el texto utilizado en la búsqueda en Google.

Figura 2.14: Adaptación basada en la tarea del usuario

Aunque parezcan adaptaciones parecidas a las conseguidas con CSN, debe notarse, que no hubo en esta interacción una navegación entre las aplicaciones sino que con solo *monitorear* la actividad del usuario un script puede realizar

adaptaciones que traten de facilitar la tarea actual del usuario.

2.4.4. Principales problemáticas

Tal vez la principal problemática del enfoque de adaptación en el cliente viene de su mayor fortaleza: adaptar una aplicación de terceros implica modificar los documentos que dicha aplicación provee; el problema en ello: para modificarlos es necesario *conocerlos* en pos de poder generar artefactos de adaptación específicos para esos documentos.

Claro, el problema radica en que cuando éstos cambian (algo que no puede controlarse ya que dichos documentos son ajenos) pasan a ser documentos desconocidos. Si esto sucede, los artefactos que estaban funcionando con una versión particular de un documento tal vez dejen de funcionar.

En este sentido, y yendo a términos concretos, esta dependencia del DOM (de los documentos) es una de las principales debilidades de cualquier enfoque de adaptación que en realidad no tiene que ver en el hecho en si mismo de que se realiza en el cliente, sino que se realiza sobre aplicaciones cuyos documentos no controlamos. Esta diferencia debe quedar clara, ya que la adaptación en el cliente puede no tener estos problemas cuando esta contemplada por los mismos desarrolladores de la aplicación.

En este trabajo se propondrán mecanismos de adaptación a varios niveles de dependencia. Es decir, que aunque es necesario manipular el DOM de manera muy específica para lograr complejas adaptaciones (especialmente aquellas orientadas a la integración de aplicaciones), también se han desarrollado mecanismos para generar adaptaciones independientes del DOM.

Aunque los desarrolladores de la aplicación pueden modificar la versión original de un DOM esperada por un artefacto, esta no es la única manera a partir de la cual un artefacto de adaptación intente manipular un DOM diferente al previsto. Si mas de un artefacto de adaptación está definido para ser aplicado sobre el mismo DOM, entonces la versión del DOM que resulta de la adaptación provocada por el primer artefacto puede alterar lo que el los subsiguientes artefactos esperan como DOM de entrada. Esta es, desde el punto de vista de la debilidad de los artefactos, la segunda problemática a contemplar: interferencias

entre diferentes artefactos.

Lamentablemente, la interferencia entre artefactos puede solucionarse dentro del alcance de una herramienta determinada (lo cual significa controlar los artefactos que se ejecutan dentro del contexto de esa herramienta, por ejemplo, dándoles un orden de ejecución determinado), pero en la actualidad donde existen incontables herramientas de este tipo, no puede asegurarse un correcto funcionamiento de todos los artefactos de cualquier herramienta si el usuario no lo gestiona correctamente. Debe notarse que el problema de interferencia entre artefactos de adaptación no es solo que un artefacto puede recibir un DOM de entrada que no corresponde con la versión del DOM esperada, sino que este artefacto puede echar a perder las adaptaciones realizadas por artefactos ejecutados previamente.

Escapa a esta presentación analizar aquellos aspectos de seguridad que vienen de la mano con la instalación de artefactos de adaptación diseñados e implementados por terceros. En general, las herramientas de adaptación del lado del cliente *agregan* comportamiento en los Browsers Web de manera tal que los artefactos que se introducen, y cuyos objetivos pueden ser malintencionados, podrían utilizar cualquier información proveniente de la interacción del usuario. La seguridad será analizada particularmente para el contexto de la herramienta desarrollada como soporte tecnológico de este trabajo en capítulos posteriores.

Capítulo 3

Fundamentos para una arquitectura flexible de adaptación en el cliente orientada al soporte de tareas de usuario

El alcance de la adaptación de aplicaciones Web en el cliente es realmente amplio, como puede apreciarse en la descripción del capítulo anterior. Es importante limitar el panorama a la hora de establecer los aspectos que serán considerados en este trabajo. En este sentido, y en términos generales, son cuatro las dimensiones fundamentales que deben ser tenidas en cuenta para definir un marco de trabajo de adaptación en el cliente:

- Tipos de adaptaciones: cuáles son los tipos de adaptaciones relevantes en el contexto de soporte de tareas de usuarios.
- Información involucrada en las adaptaciones: qué información es relevante para ejecutar las adaptaciones.
- Métodos de ejecución de adaptaciones: cómo las adaptaciones deben ser ejecutadas, bajo qué mecanismos.
- Desarrollo de artefactos de adaptación: cómo y en qué niveles pueden desarrollarse los artefactos de adaptación.

Cada una de estas dimensiones será analizada en una subsección especializada a continuación, enfatizando el punto de vista que a este trabajo respecta: soporte de tareas de usuario.

3.1. Tipos de adaptaciones a soportar

Hay aspectos de la adaptación de las aplicaciones Web del lado del cliente que aunque mejoran notablemente la experiencia del usuario en el uso de la aplicación, no están verdaderamente sujetas a la tarea que el usuario está realizando en un determinado momento. Por ejemplo, las adaptaciones por preferencia se ejecutarán siempre, sin importar la actividad del usuario. Idéntico es el caso para aquellas adaptaciones que atacan problemas de accesibilidad.

En este trabajo se busca soportar, mediante adaptación en el cliente, al usuario en su tarea actual; así es que serán de importancia en este contexto las adaptaciones que puedan ser aplicadas bajo determinadas circunstancias, sin necesariamente descartar al resto. Es decir que en función de la taxonomía especificada en el capítulo anterior, las *adaptaciones* relevantes son las que caen dentro de las categorías de *concern-sensitive navigation* y *task-aware*.

Soportar especialmente estos tipos de adaptaciones, como se dijo, no implica dejar directamente fuera del enfoque a las otras adaptaciones (de accesibilidad, basada en preferencias, etc.) sino que simplemente significa poner los esfuerzos en definir mecanismos de abstracción para soportar de mejor manera la creación de artefactos que soporten al usuario en su tarea o *concern* actual.

3.2. Información involucrada en las adaptaciones

En pos de soportar al usuario en su tarea actual deben contemplarse conjuntos de información que sean importantes o relevantes en el contexto de dicha tarea. La información no es solo un objetivo, es decir, *lo que* se adapta en función de la tarea del usuario, sino que también puede ser un medio, es decir *con lo que* se adapta. En cualquiera de los casos, es necesario obtener esa información ya

sea infiriendo qué información utilizar o bien permitiéndole al usuario que sea él quien especifique que información es de importancia.

Pero bien, ¿cuál es la información dentro del marco de adaptación en el cliente?: la información desde el punto de vista natural de uso del usuario es algún conjunto de datos que el usuario puede utilizar de alguna manera. Esto puede ser texto, una imagen, una tabla, o cualquier elemento que pueda ser presentado al usuario en una aplicación Web. Además existe un conjunto de información inherente al usuario y que posiblemente no esté presente dentro de los sitios Web; este es el caso, por ejemplo, de la información que el usuario ingresa en los formularios.

Veamos tres ejemplos sumamente simples en los cuales se jerarquizan distintos *tipos* de información:

- Texto plano: Un usuario visita una aplicación Web de una institución académica para buscar la dirección postal de la misma. Una vez obtenida, y mediante *copy and paste* el usuario abre GoogleMaps para buscar dicha dirección. Claramente, aquí el usuario utiliza información representada con texto plano y requiere *mover* esa información de una aplicación a otra.

- Información no presente en el sitio Web: Un usuario está organizando un viaje, y para ello visita dos sitios Web con distintos fines. Uno para la reserva y compra de pasajes y otro para la reserva de una habitación de hotel. Claramente, para realizar estas tareas el usuario requiere ingresar información personal mediante el uso de formularios Web. En este caso, la información no está originalmente presente en el sitio Web sino que es ingresada por el usuario.

- Elementos del DOM: Un usuario esta en búsqueda de una imagen para luego crear una publicación con la misma en una red social. En este caso la información proviene no del texto plano que los sitios proveen, sino de los elementos del DOM, en este caso las imágenes.

Además de la información ingresada por los usuarios y de la información contenida en las aplicaciones Web, en el cliente también existe la posibilidad de obtener información acerca de la actividad del usuario. Este tipo de información es claramente mas completa en el cliente que en el servidor. En cliente el historial

de navegación puede ser accedido en su totalidad, a la vez que puede conocerse lo que el usuario realiza en cualquier aplicación Web y no solo en una sola aplicación, como sucede en el servidor, donde solo se puede *ver* lo que el usuario hace dentro de los límites de la aplicación.

Para resumir, a continuación se especifica una lista de tipos de información relevantes en el contexto de soporte a la tarea del usuario mediante adaptación del lado del cliente:

- Historial de navegación: el historial de navegación muestra cuales son las aplicaciones que han sido utilizadas por el usuario. De alguna manera, y haciendo un análisis sobre las aplicaciones registradas en este historial, uno podría deducir cuál es la actividad del usuario. En el campo de la adaptabilidad, se ha utilizado el historial de navegación, pero de un modo mas restringido: intra-aplicación. Un ejemplo emblemático es Brusilovsky [2007], donde a partir de los hiperenlaces seguidos por el usuario se adaptan las páginas de la aplicación. En ese trabajo se puede contemplar el historial contenido dentro de los límites de la misma aplicación, es decir, que al no poder acceder al historial relativo a otras aplicaciones, las adaptaciones no pueden contemplar el total de la navegación del usuario. En el cliente, el historial de navegación completo puede ser utilizado.

- Aplicaciones en uso: aunque el historial de navegación nos muestra todas las aplicaciones utilizadas por el usuario en un tiempo determinado, también es relevante conocer cuáles son las aplicaciones actualmente en uso. Los Browser Web desde hace varios años posibilitan a los usuarios navegar múltiples aplicaciones simultáneamente Dubroy and Balakrishnan [2010]. Aunque a veces puede estar asociado a un comportamiento de multitarea (*multitasking*) Aula et al. [2005], otras veces la utilización simultánea de varias aplicaciones (ya sea en mediante el uso de *tabs* o bien de ventanas) están relacionadas a la misma tarea, por ejemplo, la comparación de resultados de búsqueda Weinreich et al. [2006]. Así es que, además de que algunas adaptaciones pueden estar basadas en el historial de navegación en sí mismo, otras adaptaciones pueden estar *pendientes* de las aplicaciones actualmente en uso.

- Actividad del usuario: además de saber cuáles aplicaciones están en uso y cuáles fueron utilizadas, un escalón puede subirse en pos de determinar con mayor granuralidad cual es la actividad del usuario. Imagine que el usuario esta realizando una búsqueda en Google. Esto implica que se sabe, según los puntos anteriores, que Google aparecerá en el historial de navegación y se sabe que Google esta siendo utilizada. Distinto es ello de saber que el usuario ha realizado una nueva búsqueda en Google, o bien que ha abierto un resultado de búsqueda de Google. Este tipo de información acerca de la actividad del usuario, es decir, de la interacción que éste realiza con las aplicaciones, puede ser útil a la hora de soportar sus tareas. Esta información además de las interacciones, digamos *nativas* (abrir un sitio Web, enviar un formularios, etc), también puede incluir los datos de la nueva interacción que se habilita por el enfoque de adaptación que se ejecuta en el cliente.

- Información colectada por el usuario: información colectada por el usuario es aquella información que el usuario detecta en las aplicaciones Web que usa y que requiere utilizarla de alguna manera en esa aplicación o en cualquier otra. Este es el tipo de información presentada en el primer ejemplo sobre la utilización de texto plano que es copiado de una aplicación y pegado en otra. En este trabajo se presenta una manera que sobrepasa el modo *ad-hoc* en el cual hoy en día los usuarios *mueven* información entre las aplicaciones. Esta información colectada también puede ser utilizada para disparar adaptaciones.

- Espacio de información personal: en función del ejemplo acerca de información relevante para una tarea en particular pero no presente en las aplicaciones Web (es decir, aquella ingresada por los usuarios) puede considerarse de gran importancia el manejo de un espacio de información personal Teevan et al. [2008] que haga fácil la compleción de formularios, e incluso que permite automatizar la misma por parte de los artefactos de adaptación. Cabe aclarar que técnicas como Araújo et al. [2010] pueden ser utilizadas por scripts en el cliente en pos de automáticamente completar formularios. Además, en Firmenich et al. [2012] se ha demostrado la conveniencia de utilizar un espacio de información personal junto que técnicas de augmentación

de formularios Web en el cliente.

- Modelos de abstracción del DOM: Los sitios web, y sobre todo aplicaciones con lógica compleja, generan páginas Web en las que se muestran objetos del modelo de negocio de la aplicación. Claramente, en el cliente estos *objetos* pueden recuperarse parcialmente y generar un modelo del lado del cliente con la información disponible en las páginas Web. Esta idea se ve reflejada en la Figura 3.1. En esta Figura podemos apreciar que de la página Web correspondiente a Facebook pueden recuperarse los objetos *UserPost* y *User* y ademas sus correspondientes propiedades. Note que la abstracción del concepto u objeto se realiza desde un elemento del DOM en particular, y las propiedades son subelementos del DOM del elemento principal.

  Luego de definir estos modelos, los artefactos de adaptación pueden ser definidos en términos de estos conceptos en lugar de manipular objetos específicos del DOM. En otras palabras, los artefactos pueden evitar referirse a elementos del DOM en particular y acceder al DOM mediante uno de los conceptos abstraídos, siendo que cada concepto o propiedad debe ser capaz de retornar su elemento del DOM correspondiente.

  Esta abstracción en modelos en el cliente tiene directa implicancia en la reutilización de las adaptaciones. Imagine que además de en Facebook, el mismo modelo (con los mismos conceptos y las mismas propiedades) es creado para otra red social. Esto implica que cada modelo (uno por cada aplicación) conoce elementos del DOM específicos (correspondientes a cada aplicación), pero los artefactos de adaptación funcionarán con ambas aplicaciones, ya que no dependen directamente de un elemento del DOM, sino que accede al DOM mediante representantes: conceptos y propiedades. Esto fue dado a conocer como modelos de familias de aplicaciones en Firmenich et al. [2010b], y la propuesta de estos modelos estuvo fuertemente inspirada por Díaz et al. [2008].

  La Figura 3.2 muestra de forma genérica esta idea. En la capa superior aparecen los plug-in o artefactos de adaptación que mediante el acceso a un modelo (capa del medio) termina accediendo a los elementos del DOM específicos de la aplicación. Acá puede notarse que cualquiera de los artefactos

(*Concrete plugin 1* o *Concrete plugin 2*) funcionarían con ambas aplicaciones que comparten el mismo modelo. Las aplicaciones se representan en la Figura en la capa inferior.

Habiendo aclarado que significa un modelo de abstracción del DOM, debe quedar claro que se consideran de importancia en un enfoque de adaptación del lado del cliente y por ello se consideran como parte de la información disponible por las adaptaciones.

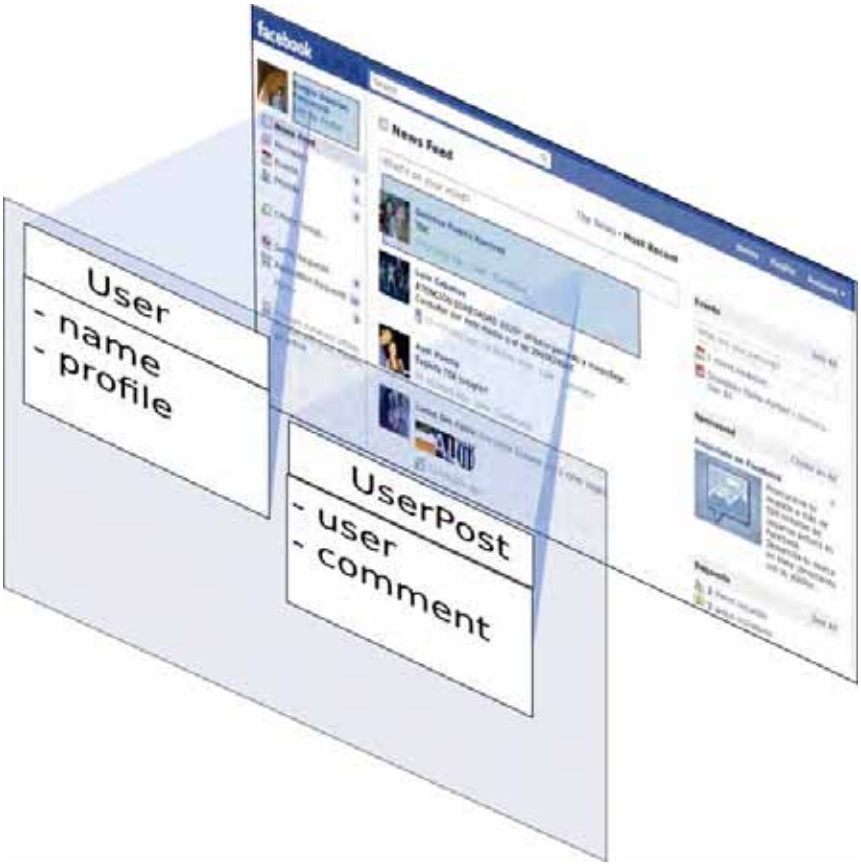

Figura 3.1: Abstracción de un elemento del DOM en un concepto

Determinar qué información puede ser útil para el usuario es un aspecto, pero desde el punto de vista de la ejecución de las adaptaciones como obtener tal información es igual de importante. La información relevante a la tarea del usuario puede ser obtenida de varias maneras, pero en el trasfondo hay tres tipos de métodos según quién es responsable de obtenerla:

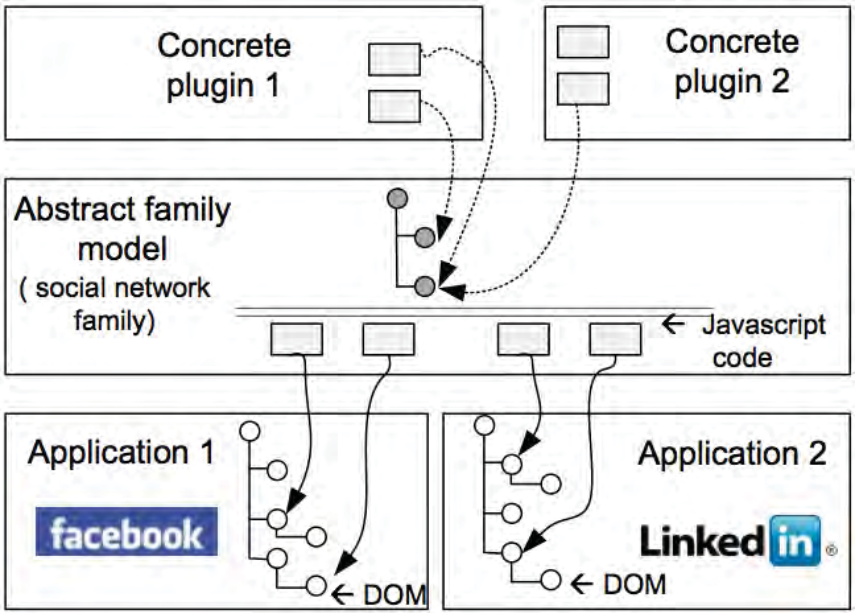

Figura 3.2: Esquema genérico de reutilización de artefactos basado en modelos de familias de aplicaciones

- Manual: El usuario explícitamente escoge a consciencia la información y colecta esta de algún modo. En esta tesis se ha considerado de suma importancia la posibilidad de que los usuarios puedan colectar información manualmente, y en este sentido se ha desarrollado una herramienta que inspirada en la metáfora de un *bolsillo*, permite guardar ítems de información y utilizarlos posteriormente en otro momento, incluso en otra aplicación Web.

  A modo de ejemplo, en la Figura 3.3 se muestra una de las herramientas principales desarrolladas en el marco de esta tesis, y que será explicada en profundidad en los capítulos siguientes.

  En dicha Figura se muestra la herramienta *Pocket*, una herramienta que le permite al usuario colectar información en los sitios Web (tanto texto plano como elementos específicos del DOM) y que le permite visualizar y utilizar (de distintas maneras, como será explicado) en otras aplicaciones Web dicha información. Debe notarse que la información es colectada, en este caso, de un modo totalmente manual. El usuario es quien escoge la información deseada y además quién colecta la misma utilizando los mecanismos provis-

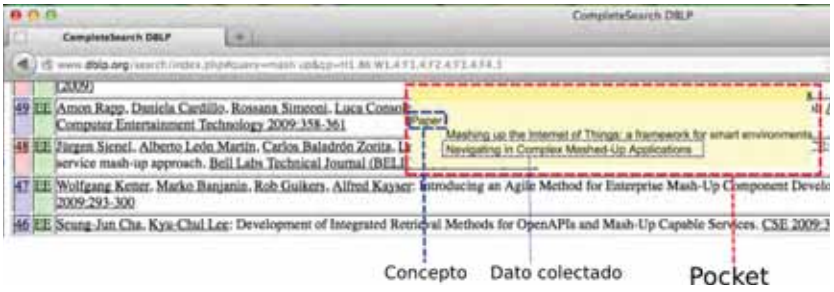

Figura 3.3: Herramienta *Pocket*

tos, como muestra la Figura 3.4. En esta Figura se muestra cómo habiendo seleccionado el título de un artículo, el usuario puede colectarlo en el *Pocket* utilizando el menú contextual.

Figura 3.4: Herramienta *DataCollector*

- Automático: Existen varias técnicas para colectan información automáticamente. Tal vez la mas clara sea la obtenida a partir de observar las entradas en formularios que los usuarios realizan. Por ejemplo, la mayoría de los Web Browsers *memorizan* los datos ingresados en los campos de los formularios para que, la próxima vez que campos similares (en el mismo o en otros formularios) sean completados, ofrecerle al usuario las entradas anteriores. Este tipo de técnica es *cross-site* e independiente del *concern*. Sin embargo, utilizando técnicas para dar semántica a elementos del DOM, por ejemplo, utilizando Microformats Khare and Çelik [2006] o bien cualquier otro mecanismo de abstracción como se ha utilizado en Firmenich et al. [2010b].

De esta manera un artefacto de adaptación puede *buscar* por su cuenta información y colectarla para uso posterior.

Enfoques híbridos entre lo automático y lo manual pueden establecerse cuando, por ejemplo, la detección de información se realiza automáticamente pero la decisión final de colectar la información la realiza manualmente el usuario.

Debe quedar claro, detrás de una herramienta que le permita al usuario colectar información debe haber una herramienta que le permita utilizarla de algún modo, cuando menos que le permita visualizarla. Esto esta sumamente relacionado en cómo las adaptaciones son ejecutadas, las posibilidades son analizadas en la siguientes subsección.

3.3. Ejecución de adaptaciones

La adaptación del lado del cliente no es mágica, las adaptaciones son realizadas gracias a que algún script fue ejecutado. La ejecución de scripts no es tema menor en el contexto de soportar la tarea del usuario. Básicamente hay dos formas en la que pueden ejecutarse los artefactos de adaptación. Por un lado, estos pueden ser ejecutados automáticamente (a grandes rasgos significa que una adaptación es ejecutada porque alguna condición dentro del contexto del Browser Web fue satisfecha) o manualmente (lo cual implica que el usuario explícitamente decide ejecutar la adaptación).

3.3.1. Adaptaciones automáticas

La ejecución automática de artefactos que adapten las aplicaciones Web visitadas por el usuario es la forma de ejecución mas común. Esto implica que las adaptaciones son realizadas a partir de la interacción del usuario con la Web, pero de una manera indirecta. El ejemplo mas común es cuando el usuario ingresa a un sitio Web y existe algún artefacto de adaptación instalado y cuyo objetivo (es decir a *qué ente* adapta) es dicho sitio Web, entonces el artefacto sería ejecutado. En este simple ejemplo, no existe una intervención directa del usuario, simplemente la adaptación fue ejecutada a partir de cierta actividad del usuario. En el contexto del cliente son muchos los *eventos* que pueden desencadenar la ejecución

de un artefacto. Simplemente se necesita que el artefacto, o bien la herramienta que coordina la ejecución de los mismos, registre su interés en ese evento.

Las adaptaciones que se realizan automáticamente son, en general, adaptaciones que siempre son necesarias o de preferencia para el usuario: adaptaciones orientadas a la accesibilidad, adaptaciones basadas en preferencias, etc. Lo cierto es que, en el marco de soportar al usuario en sus tareas, no siempre es necesario aplicar una adaptación cuando un sitio Web especifico es cargado sino que, por el contrario, depende de la actividad del usuario, de su *concern*, y en este sentido no siempre una adaptación tiene sentido.

3.3.2. Adaptaciones bajo demanda del usuario

Incluso una adaptación basada en el *concern* del usuario puede ser ejecutada automáticamente cuando parte del artefacto que realiza la adaptación se encarga también de *detectar* el *concern* actual del usuario. Como se explicó en la subsección anterior, solo hace falta escuchar el evento o el conjunto de eventos correspondiente. Sin embargo, que un artefacto de adaptación no detecte un *concern* no significa que el usuario en ese momento no este realizando ninguna tarea en particular. Claramente, no necesariamente existe un artefacto de soporte de tarea de usuario para cualquier tarea. Esto no significa que simples adaptaciones sobre los sitios que están siendo utilizados por los usuarios no sean de utilidad, como por ejemplo, resaltar las ocurrencias de la información colectada por el usuario existentes en el sitio actual. Puede notarse que este tipo de adaptación puede ser útil bajo cierta tarea, y aunque no sea ejecutada automáticamente por un artefacto que este pendiente de los *eventos*, puede ser realizada bajo demanda del usuario.

Así es que en el contexto de la tarea del usuario pueden ser útiles adaptaciones que *explícitamente* son ejecutadas por el usuario, y que por tal motivo, deben ser independientes de toda aplicación en particular.

3.3.3. Combinación de adaptaciones

Para un soporte total de la tarea de usuarios debería contemplar ambos modos de ejecución: automático y bajo demanda del usuario. Las adaptaciones automáti-

cas son realmente beneficiosas y esto se ha probado a lo largo de los años. Sin embargo el comportamiento de los usuarios en la Web es de cierta manera impredecible y por lo tanto no siempre se cumplen las condiciones de ejecución de las adaptaciones a la vez que, por lo dicho, no significa que el usuario no pueda ser beneficiado de adaptaciones que pueda ejecutar manualmente. Incluso mas, para ciertas tareas posiblemente el usuario no quiera delegar en una herramienta totalmente automáticamente por cuestiones de riesgo. En este sentido, y como se mostrará y probará mediante diversas evaluaciones, en este trabajo se considera la combinación de ambos modos de ejecución para cubrir de mejor manera el rango de situaciones en las que la experiencia del usuario puede ser mejorada.

3.4. Desarrollo de artefactos de adaptación

Un determinado enfoque de adaptación puede ser *cerrado* o *abierto*, donde *cerrado* o *abierto* significa, correspondientemente, que las adaptaciones son incluidas dentro del enfoque o bien que son *agregadas* por el usuario final. Existen numerosas herramientas *cerradas* de adaptación, que siempre aplican las mismas adaptaciones; un ejemplo claro de ello es un plug-in que una vez instalado en el Browser Web adapta los sitios Web visitados cambiando el texto plano cuyo formato corresponde al de un número telefónico por un botón para realizar un llamado vía Skype.

Sin embargo los enfoques de adaptación *abiertos* proveen, en términos generales, distintas funcionalidades base que ofrecen a *desarrolladores* facilidades para crear nuevos artefactos. Atrás de dichos enfoques como GreaseMonkey (que permite agregar scripts JavaScript en el cliente para que sean ejecutados al cargar los sitios Web), AccessMonkey (similar a GreaseMonkey, pero orientado a accesibilidad), Stylish (que con el mismo concepto subyacente, permite agregar estilos CSS en lugar de scripts JavaScript), etc., existe una masa de usuarios, como se dijo en el capítulo anterior, atendiendo necesidades por su cuenta: siendo los *desarrolladores*.

Desde este punto de vista los *usuarios desarrolladores* son un subconjunto de la masa de usuarios que, con diferentes conocimientos de programación Web, generan artefactos de adaptación. Uno de los desafíos mas grandes de las pro-

puestas de adaptación del lado del cliente es hacer crecer la cardinalidad de tal subconjunto, es decir, que abarque la mayor cantidad de usuarios. Claramente, cuantos menos mecanismos de abstracción el enfoque o la herramienta proveen, menos *usuarios desarrolladores* estarán habilitados para desarrollar artefactos bajo dicho enfoque o herramienta ya que más específicos y profundos deben ser sus conocimientos en el área.

Aunque cierto tipo de programación Web es difícil de abstraer a un punto tal que cualquier usuario pueda realizarla, un enfoque flexible podría contemplar un esquema mas colaborativo, donde distintos tipos de artefactos requieran distintos niveles de conocimiento y/o habilidades para así incluir a mas usuarios en el grupo de usuarios desarrolladores sin perder la flexibilidad y el potencial de desarrollar artefactos a bajo nivel.

3.5. Conclusiones

Habiendo realizado un análisis desde el punto de vista de soportar la tarea del usuario, que abarca información, métodos de ejecución, tipos de adaptación y desarrollo de adaptación resta concluir brevemente los objetivos bajo los cuales se han diseñado los distintos artefactos, conceptuales y técnicos, que hacen a la arquitectura y herramienta sugeridas para el soporte de tareas de usuario:

- Que soporte principalmente adaptaciones que consideren el *concern* o la tarea actual del usuario, para lo cual, distintos tipos de información referidas a tal deben ser contempladas.

- Permitir tanto la ejecución automática de adaptaciones para aquellos casos en los que el *concern* o tarea pueda ser detectado a la vez de soportar ejecución bajo demanda del usuario de adaptaciones para soportar al usuario cuando éste no dispone de un artefacto específico para su tarea actual.

- Conseguir un enfoque inclusivo de desarrollo de artefactos de adaptación, proveyendo a los usuarios distintos mecanismos de abstracción que permitan a usuarios con distintos conocimientos desarrollar artefactos de distintos niveles de abstracción.

En el siguiente capítulo se muestra la arquitectura conseguida aplicando los fundamentos descriptos en el presente capítulo.

Capítulo 4

CSA Framework: una solución concreta

En este capítulo se mostrará una estructura de software definida siguiendo los fundamentos descriptos hasta aquí y referidos al soporte a la tarea del usuario mediante adaptación de aplicaciones Web. Un listado de los componentes principales del *framework* es presentado, a la vez que se muestra la arquitectura. También se explicarán los puntos de extensión del *framework* y aspectos sobre la extensibilidad del mismo. Finalmente se concluye el capítulo con una descripción de cómo se soportan los fundamentos planteados en el capítulo anterior.

4.1. Artefactos involucrados y Arquitectura del lado del Cliente

En pos del buen entendimiento y valoración de los componentes del *framework* diseñado, primero se dará un esquema del mismo donde se intenta resaltar los distintos puntos de vista que existen en el enfoque.

4.1.1. Una visión estratificada

El *framework* obtenido es una herramienta que terminará siendo utilizada tanto por desarrolladores como por usuario finales. Esto es así dado que los desarro-

lladores son usuarios del *framework* en tanto y en cuánto desarrollan extensiones sobre el mismo; mientras que, por supuesto, los usuarios finales utilizan el *framework* como motor de ejecución de adaptaciones. Además, y como se ha remarcado en varios pasajes de esta presentación, será importante brindar diferentes herramientas para diferentes niveles de capacidades de desarrollo de software por parte de los usuarios.

Todo esto en conjunto implica que el *framework* obtenido es una combinación de herramientas, conceptos y mecanismos orientados a i) algunos de ellos a desarrolladores y ii) otros a los usuarios finales. En este sentido, se presenta a continuación una visión estratificada en función de la abstracción necesaria para el punto de vista de cada tipo de usuario del *framework* (usuarios finales y usuarios desarrolladores).

En la Figura 4.1 se muestran tres capas de abstracción que describen al *framework* desde tres puntos de vista diferentes de la funcionalidad. Estas *visiones* son de real importancia a la hora de comprender el rol de cada uno de los componentes que integran cada una de estas *franjas* de abstracción y la colaboración entre los mismos.

Siguiendo un enfoque piramidal Meusel et al. [1997] se definen en la cima de la pirámide los componentes mas abstractos, aquellos orientados al uso del *framework* de adaptación por parte de usuarios finales, claramente esto tiene que ver con la ejecución de las adaptaciones.

Siguiendo en forma descendente, en la capa intermedia figuran aquellos componentes relevantes desde el punto de vista de los desarrolladores de adaptaciones. Aquí aparece el uso de los puntos de extensión que requieren distintos niveles de conocimientos para ser utilizados por estos desarrolladores.

Finalmente, en el pie de la pirámide aparece la capa de soporte para ambas tareas: las de desarrollo y la de ejecución de las adaptaciones.

4.1.1.1. Componentes del Framework

Habiendo presentado un esquema genérico para proveer una visión general y a la vez estratificada de la funcionalidad del *framework*, se describirán en esta subsección los componentes de cada una de estas estratificaciones. Para esto, y

Figura 4.1: Un visión funcional general del framework

respetando el enfoque piramidal, se muestra en la Figura 4.2 la distribución de los componentes del *framework* y sus relaciones.

En la cima de la pirámide se representa el punto de vista de los usuario finales. Así es que aquí se incluyen, por ejemplo, las contrapartes visuales de las herramientas de recolección de datos que serán explicadas en secciones posteriores. Además figuran aquellos componentes que le permiten a los usuarios finales ejecutar adaptaciones manualmente o que les brindan algún tipo de respuesta respecto a sucesos de las mismas. Es decir que en esta capa se incluye todo aquello que el usuario final *percibe* en el uso del *framework*.

En el medio, los desarrolladores (ni mas ni menos que usuarios finales con conocimientos en el desarrollo de software) pueden extender de distintas maneras el *framework*: ya sea desarrollando nuevos *augmenters* (mediante la especialización de un nuevo artefacto que herede del punto de extensión *AbstractAugmenter*), o bien nuevos *escenarios* (mediante la instanciación de un *AbstractScenario*).

La capa inferior muestra un visión mas detallada del diseño del *framework*, donde además se ofrecen otros puntos de extensión como *AbstractComponent* y *AbstractDataCollector*. El primero *AbstractComponent* permite realizar desarrollar nuevos componentes del *framework* que realicen tareas especificas para soportar la ejecución de las instancias de *AbstractScenario*. Por ejemplo, uno de los componentes desarrollados e incluidos en el *framework* ofrece información relativa a geolocalización. *AbstractDataCollector* es un punto de extensión del

framework que ofrece la posibilidad de extender las herramientas brindadas al usuario para colectar información mientras navega la Web. Actualmente el *framework* incluye tres tipos de *DataCollector*. Estos *DataCollector* ya incluidos y el resto de los componentes mas relevantes son explicados con mayor especificidad en la siguiente lista:

- **Capa de soporte a la adaptación**

  - *ClientSideAdaptationManager*: es el componente central del *framework* cuyas funciones son las de coordinar al resto de los elementos y de servir como comunicador con el Browser Web.

  - *WebAppWrapper*: este componente se aboca a abstraer una aplicación Web en particular. Como se verá a continuación, una de las funcionalidades del framework es la definición de modelos que abstraigan al DOM de las aplicaciones (lo cual fue explicado en el capítulo anterior). Así es que cuando el usuario visita una aplicación Web determinada, al cargar la misma el correspondiente *WebAppWrapper* es instanciado para dejarlo a disposición de los artefactos de adaptación y que estos últimos puedan *consultar* acerca de los elementos del modelo definido para el sitio Web en particular. Es decir que un *WebAppWrapper* conocerá, cuando corresponda, a un conjunto de instancias de *Concept*.

  - *Concept*: es la clase que maneja las abstracciones del DOM. Estos conceptos ofrecen un acceso transparente a los elementos del DOM que corresponden a una instancia de este tipo. Un *Concept* no representa a un elemento DOM en particular, sino que abstrae todos los elementos del DOM que son instancias de *Concept*. Lo cierto es que por cada instancia de un *Concept*, se crea un *ConcreteInstance* que accede al elemento del DOM real.

  - *ConcreteInstance*: es la instancia concreta de un *Concept*, es decir, que una instancia concreta de un concepto es la que realmente posee la información necesaria para realizar el *mapping* entre un concepto abstracto al elemento del DOM especifico de una aplicación. Para comprender mejor esta idea es bueno un ejemplo. Suponga que se define el concepto ResultadoDeBusqueda. Un concepto que puede ser relevante para mas de una

aplicación Web: Google.com, Yahoo.com, etc. Para realizar el *mapping* de un concepto abstracto al elemento específico de cada DOM, se crean instancias de *ConcreteInstance* logrando que para el DOM, por ejemplo de Google.com, se obtengan N instancias de *ConcreteInstance* (una para cada instancia del concepto ResultadoDeBusqueda) y se definan en consecuencia N xPath distintos, uno para cada instancia de *ConcreteInstance*. Así es que una adaptación que requiera el *Concept* ResultadoDeBusqueda funcionará con cualquier aplicación que posea al menos un *ConcreteInstance* de tal concepto abstracto. Esto significa que un artefacto de adaptación puede interactuar con una instancia de *Concept*, utilizando un *ConcrteInstance*, sin saber con exactitud que elemento del DOM esta modificando o alterando. Esto tiene directa implicancia en la reutilización de las adaptaciones definidas, siendo que si diferentes aplicaciones Web son abstraídas con modelos equivalentes, los artefactos de adaptación seguirán funcionando ya que operan con el DOM mediante estos *proxys* Gamma et al. [2011]

- *History*: es el elemento que maneja el historial de navegación del Browser Web.

-*AbstractAugmenter* y *AbstractScenario*: Son puntos de extensión del *framework* que proveen funcionalidad básica correspondiente a cada tipo de artefacto. Los desarrolladores deben heredar o crear una instancia (según corresponda) de estas clases para generar artefactos concretos de adaptación. En la siguiente subsección se tratarán estos componentes particularmente, haciendo hincapié no solo en la definición de los mismos sino en la manera en que se desarrollan.

-*AbstractComponent*: es un punto de extensión del *framework* que permite extender los componentes utilizados para soportar a los artefactos de adaptación agregando nuevas capacidades: información de geolocalización, simple manipulación de imágenes, etc. El desarrollo de estos componentes es tal vez el de mayor desafío respecto a los conocimientos de programación requeridos.

-*AbstractDataCollector*: es un punto de extensión del *framework* que permite agregar nuevos tipos de *DataCollector* (los objetos del tipo *Data-*

Collector son definidos en la capa de ejecución de adaptaciones).

-*Pocket*: es el componente que almacena la información colectada mediante el uso de los *DataCollector*.

-*PocketElement*: el *Pocket* almacena objetos del tipo *PocketElement*. Básicamente hay dos tipos de elementos del *Pocket*. El objeto *PocketInstance* que representan cada ítem de información que ha sido agregado al *Pocket*. El objeto *PocketConcept*, que puede estar relacionado con mas de una *PocketInstance*, sirve para dar un significado o peso semántico a un *PocketInstance*.

-*PIMI*: es el componente que permite al usuario almacenar información de uso recurrente de forma mas definitiva (es decir, independiente del *concern* actual). La información es almacenada bajo una estructura semántica utilizando Microformatos Khare and Çelik [2006]. Hay dos diferencias principales con el objeto *Pocket*. En primer lugar, el *PIMI* tiene una estructura semántica mas fuerte. En segundo lugar la información persiste a lo largo de las sesiones de uso del Browser Web.

-*ConceptPersistenceManager*: es el elemento responsable de guardar y recuperar aquella información que requiere persistencia local. Por ejemplo, información del *PIMI*.

- **Capa de definición de adaptación**

-*DOMOperationLibrary*: una librería que opera con elementos del DOM. Básicamente tiene el objetivo de levantar el nivel de abstracción de las sentencias típicas de manipulación del DOM en JavaScript, tal como lo hacen otras librerías bien conocidas para el desarrollo Web (jQuery[1], Prototype[2], etc.).

-*EventManager*: como se ha dicho en capítulos anteriores, muchas de las adaptaciones requieren ser ejecutadas bajo ciertas condiciones. Cuando estas condiciones no están satisfechas, es necesario poder reaccionar cuando

[1]http://www.jquery.com
[2]http://prototypejs.org

sí lo estén. Para esto se incluye el objeto *EventManager*, componente responsable de manejar (agregar o eliminar) los *listeners* de los eventos en los cuales los artefactos registran interés.

-*NavigationHistoryManager*: es un *wrapper* que permite a los *escenarios* realizar consultas mas complejas sobre el historial de navegación, que son relevantes para la utilización de precondiciones (el concepto de precondición es explicado en el capítulo siguiente).

-*ConcreteAugmenter* y *ConcreteScenario*: son los artefactos desarrollados por aquellos usuarios con capacidades de programación/desarrollo. Cabe aclarar que en realidad no son parte del framework, por el contrario, aparecen en la Figura 4.2 para marcar en que posición de esta visión piramidal de la funcionalidad están ubicados. El *framework* provee por defecto varias instancias de *ConcreteAugmenter*, la lista completa de los mismos esta desarrollada en el capítulo siguiente.

-*ScenarioEditor*: es una herramienta visual para la definición de instancias de *AbstractScenario*.

- **Capa de ejecución de adaptación**

-*DataCollector*: son las herramientas que le permiten a los usuarios colectar información mientras navegan la Web para tener dicha información a disposición en cualquier aplicación. El la Figura del *framework* se incluyen, por razones de simplicidad, tres *DataCollector*: *UntypedInstanceCollector* (para colectar información en texto plano sin ningún significado semántico), *TextInstanceCollector* (para colectar información en texto plano con un peso semántico que denote el concepto: por ejemplo para el texto *París* podría definirse que es una instancia del concepto *Ciudad*), *DOMInstanceCollector* (para abstraer elementos del DOM que no son contemplados en el modelo que brinda *WebAppWrapper*). Sin embargo existen otros DataCollectors mas complejos: basados en expresiones regulares, para obtener datos de sitios Web remotos (que no están en uso), etc.

-*PocketManager*: es la herramienta que permite a los usuarios visualizar la información colectada en el *Pocket*. Cada ítem de información colectada

con cualquier *DataCollector* es almacenada en el *Pocket*, y el *PocketManager* es la interfaz gráfica del mismo: lo que usuario percibe y mediante lo que interactúa con el *Pocket*.

-*PIMIManager*: es la herramienta que permite a los usuarios visualizar y utilizar la información ingresada en el *PIMI*, a la vez que le permite eliminar o agregar información del mismo.

-*ModelManager*: permite a los usuarios crear e instanciar *Concept* y *ConcreteInstance* para el modelo de un sitio Web en particular. Note que esta herramienta es algo diferente a un *DataCollector*. Los modelos creados mediante la utilización de *ModelManager* son usualmente creados sin considerar una adaptación en particular sino que simplemente para describir los objetos del sitio Web, y así luego las adaptaciones pueden estar basadas en estos modelos. Además los modelos creados con *ModelManager* son premeditados, mientras que la información colectada mediante el uso de un *DataCollector* (por ejemplo, *DOMInstanceCollector*) es obtenida de una manera impredecible por parte del usuario.

-*AdaptationDispatcher*: permite a los usuarios ejecutar adaptaciones (cualquier *ConcreteAgumenter*) bajo demanda. Es útil para satisfacer requerimientos volátiles de adaptación, por ejemplo, copiar un valor del *Pocket* en un campo de un formulario, resaltar las instancias de un concepto del *Pocket* en el sitio actual, etc.

-*ScenarioManager* y UserArtefactManager: los usuarios finales son los responsables de descargar, instalar o desinstalar tanto *escenarios* como *augmenters*. Para ello disponen de estas herramientas para la administración de estos artefactos.

-*ScenarioPlayer*: si bien los *ConcreteAugmenters* pueden ser ejecutados bajo demanda del usuario utilizando el objeto *AdaptationDispatcher*, éste no contempla la ejecución de una instancia particular de *escenario*. Para ejecutar los *escenarios* instalados el usuario puede utilizar el *ScenarioPlayer*.

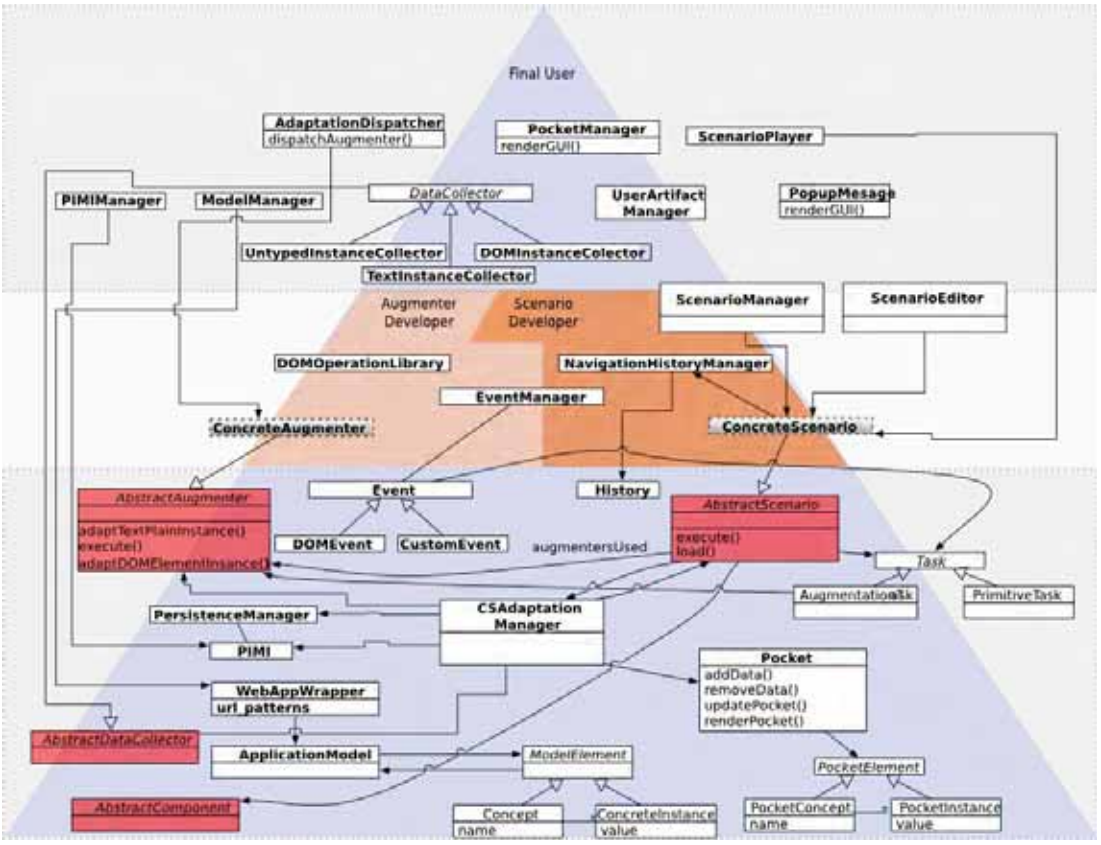

Figura 4.2: Estructura del Framework

4.2. Funcionamiento y creación de los principales artefactos de adaptación

La arquitectura especificada en la sección anterior describe varios puntos de extensión del *framework*. En la presente sección se tratará los aspectos referidos a cómo estos puntos de extensión son utilizados y creados, haciendo foco en los artefactos de adaptación y soporte de tarea de usuarios: *augmenters* y *escenarios*.

4.2.1. Augmenters

4.2.1.1. Definición y funcionamiento

La manera mas directa de extender el *framework* es desarrollar un nuevo *augmenter*. Un *augmenter* es un componente de adaptación desarrollado por usua-

rios con conocimientos en JavaScript. Los *augmenters* tienen dos contribuciones principales: i) proveen herramientas para satisfacer requerimientos volátiles de adaptación (ya que pueden ejecutarse bajo demanda de los usuario finales), ii) soportan el desarrollo de sofisticados *escenarios* que orquestan y combinan la ejecución de estos *augmenters* automáticamente.

Un *augmenter* puede ser ejecutado sin información extra, o puede recibir información (por ejemplo data colectada con un *DataCollector*) como parámetros. En este último caso la información es provista por el actor que ejecuta el *augmenter* (ya sea el usuario cuando es una ejecución manual o bien el *escenario* cuando se ejecuta de forma automática). Por ejemplo un *augmenter* que resalta los elementos de un sitio Web (*HighlightAugmenter*), debe recibir cuáles son los elementos a resaltar. La información recibida por el augmenter puede ser un simple valor (por ejemplo el texto *París*) o bien un conjunto de valores (por ejemplo todos los valores colectados en el *Pocket* bajo el *tag* semántico *Ciudad*). Además, como la información colectada también puede ser un elemento del DOM, los augmenters deben estar preparados (idealmente) para realizar la adaptación no solo sobre texto plano, sino también sobre elementos del DOM, ya sean *anchors*, *divs*, *inputs*, etc.

En la Figura 4.3 se muestra un ejemplo de *augmenter* (*WikiLinkConverter*) aplicado al sitio Web parisinfo.com. En este ejemplo, el usuario ha navegado al sitio actual habiendo colectado varias instancias de texto plano del concepto *PointOfInterest* en el *Pocket* (el que es renderizado por el *PocketManager* como la *box* flotante a la izquierda de la Figura).

Como se aprecia en la Figura, cuando el usuario realiza un click derecho sobre el concepto *PointOfInterest* presente en el *Pocket*, un menú con todos los *augmenters* disponibles es desplegado. Una de las opciones corresponde al *augmenter WikiLinkConverter*: *Convert to Wiki Link*. Cuando el usuario escoge dicho augmenter, el objeto *AdaptationDispatcher* ejecuta el artefacto correspondiente con todas las instancias de *PointOfInterest* como parámetros.

Finalmente, cuando el *augmenter* actúa, se adapta el DOM del sitio actual reemplazando texto plano por elementos *anchor* y así nuevos *links* están disponibles para ser navegados en pos de ir hacia los artículos de Wikipedia correspondientes a cada uno de los puntos de interés colectado. Como muestra la Figura 4.3, el

augmenter *WikiLinkConverter* es aplicado sobre cualquier instancia de *PointOfInterest* encontrada en el sitio Web. Note que la versión original del texto es la que aparece en la 4.4, que, como puede verse, no posee ningún hiperenlace.

Figura 4.3: WikiLinkConverter augmenter

Figura 4.4: WikiLinkConverter augmenter

4.2.1.2. Creación e instalación de un augmenter

Un *augmenter* es esencialmente un archivo JavaScript con una estructura bien definida. Esta estructura no es ni casual ni cualquiera, sino que viene definida por la necesidad que existe desde la arquitectura general del *framework* de poder ejecutar el *augmenter* automáticamente o bien posibilitarle al usuario que lo ejecute. Los *augmenters* están pensados para que se desarrolle lo mínimo posible; desde el punto de extensión correspondiente se brinda una buena parte del comportamiento necesario para ejecutarlos. De esta manera se logra que el esfuerzo en el desarrollo este realmente enfocado en la adaptación *per se*.

Es así que en el código de un *augmenter* se requiere:

- Metadata: información acerca del nombre del *augmenter*, el autor del mismo, una especificación de los argumentos que requiere el *augmenter* para ser ejecutado, etc.

- Herencia de *AbstractAugmenter*: es necesario que la clase que representa al augmenter herede de este punto de extensión del *framework* dado que de esta clase abstracta se hereda buena parte del comportamiento requerido para ejecutarlo.

- Definición de métodos específicos: se requiere la definición de métodos específicos para que el augmenter funcione correctamente, en sí, estos son los métodos que realmente realizan alguna manipulación en el DOM y son solo dos métodos: #adaptTextPlainInstance, #adaptDOMInstance y #isExecutableUnderDemand.

- Redefinición de comportamiento: parte del comportamiento de la ejecución del *augmenter* esta definida por defecto en diferentes métodos en *AbstractAugmenter*. Cuando un desarrollador desea alterar este comportamiento puede redefinir estos métodos y así dejar de utilizar los *hook methods* heredados, que son utilizados en la aplicación del patrón *Template Method* Gamma et al. [2011].

Note que de esta lista descriptiva puede verse que la programación de un nuevo *augmenter* requiere mínimos conocimientos acerca del *framework*; simplemente siguiendo una guía de los elementos descriptos un desarrollador JavaScript esta habilitado para crear uno nuevo. No se considera de importancia conceptual, debido al nivel técnico utilizado, dicha guía de desarrollo. Es por ello que la misma no se incluye en este pasaje de la presentación sino que es adjuntada como anexo, específicamente en el anexo A.

Desde el punto de vista del usuario final, aquel que no desarrolla los *augmenters* sino que los utiliza, es necesario instalar los *augmenters* de preferencia. Para ello, y teniendo el plug-in del Browser Web instalado, solo es necesario que abra el archivo del *augmenter* escogido con el Browser Web. El *framework* y en especial el componente *UserArtefactManager* ofrecerán al usuario la instalación del mismo.

4.2.2. *Escenarios*

Los *escenrios* son, básicamente, una secuencia determinada de tareas que habiendo sido especificada por un usuario desarrollador puede ser ejecutada por cualquier otro usuario. Como ya se ha mencionado, las tareas pueden ser de dos tipos: tareas primitivas (aquellas que usualmente se realizan en el Browser Web) y tareas de augmentación (referida a todas las nuevas tareas que incorpora al Browser Web la instalación del *framework*: uso de *DataCollector*, ejecución de augmenters, etc.).

La coordinación de estas tareas no es trivial y por tanto en lugar de dejar que los usuarios desarrolladores codifiquen en JavaScript la lógica necesaria para controlar la secuencia, se ha especificado un DSL (***D**omain-**S**pecific **L**anguage*) Fowler and Parsons [2010] para representar dicha composición de tareas. Así es que los *escenarios* se distribuyen como archivos XML con la estructura definida por el DSL especificado, habiendo además herramientas (como *ScenarioManager*, *ScenarioEditor*, *ScenarioPlayer*) que facilitan la creación e instalación y permiten la ejecución de los mismos.

En el marco del enfoque de esta tesis los *escenarios* son el mecanismo de automatización de las adaptaciones. Por el contrario de los *augmenters* (que son artefactos pasivos ya que no actúan por si solos sino que requieren de un actor que los ejecute), los *escenarios* están constantemente *observando* la actividad del usuario. Monitorear tal actividad les da a estos la posibilidad de saber cuándo una tarea (sea primitiva o de augmentación) fue ejecutada y le da el pie para intentar ejecutar la próxima. Como se verá, cada tarea individualmente puede ser marcada para que se ejecute automáticamente, así es que las adaptaciones pueden ser realizadas de un modo mas transparente desde el punto de vista del usuario final.

La composición de tareas, el DSL correspondiente para definir esta composición y la herramienta para ejecutarlos son parte central de la contribución de este trabajo y por tal motivo no se describen en la presente sección sino que se abarcan en el siguiente capítulo dedicado a ello.

4.2.3. Componentes

Brevemente en este apartado se define el punto de extensión llamado *Component* y su objetivo principal.

En términos generales un *Component* es una especie de librería que queda disponible para los artefactos responsables de las adaptaciones. En ciertos casos existen acciones que son necesarias para ejecutar una adaptación, pero que no es parte de la adaptación en si misma. Por ejemplo obtener la información geográfica del usuario no es una adaptación en sí, es semejante a un servicio. Es así que los componentes registrados en el *framework* brindan algún tipo de soporte a los *augmetners* o bien a los *escenarios*. Siendo que el desarrollo de nuevos componentes requiere de alto nivel de tecnicismo, no se abarcaran mas detalles acerca de cómo se desarrollan estos componentes.

Los nuevos componentes, al igual que los *augmenters*, son instalados cuando se abre el archivo de los mismos con el Browser Web, momento en el cual se le sugiere al usuario la instalación.

4.2.4. Repositorios de artefactos

A pesar de que el *framework* aquí presentado permite desarrollar varios tipos de artefactos por parte de usuarios, debe existir una manera directa y simple de que éstos compartan sus artefactos con aquellos usuarios finales que están interesados en el uso del *framework*. En este sentido, el *framework* esta preparado para funcionar con un repositorio remoto; un repositorio que sigue la filosofía de los repositorios actuales de scripting[1].

4.3. Conclusiones

Si se hace un repaso por los componentes definidos, la estructura del *framework* y la descripción de los componentes principales (*DataCollector*, *Pocket*, *augmenters* y *escenarios*, etc.), podemos decir que se satisfacen los fundamentos especificados en el capítulo anterior, ya que:

[1] Repositorios como http://userscripts.org

- Se pone a disposición de las adaptaciones diferentes tipos de información:

  - *ModelManager/WebAppWrapper*: Modelos abstraídos del DOM.

  - *PIMI*: Información de uso recurrente persistente y disponible.

  - *DataCollectors/Pocket*: Información volátil y colectada por el usuario.

  - *NavigationalHistory*: Información acerca de las aplicaciones Web utilizadas por el usuario.

- Se facilitan dos tipos de ejecución de las adaptaciones:

  - *AdaptationDispatcher/Augmenters*: pone a disposición del usuario la ejecución manual de los *augmenters*.

  - *ScenarioPlayer/Scenario/Augmenters*: posibilita la detección de la actividad del usuario seguida de la ejecución automática de los *augmenters*.

- Se consideran distintos niveles de desarrollo de artefactos:

  - *AbstractAugmenter*: manipulación del DOM con JavaScript.

  - *AbstractComponent/AbstractDataCollector*: librerías de bajo nivel y artefactos de recolección de datos.

  - *AbstractScenario/ScenarioEditor*: orquestación de tareas a alto nivel utilizando *ScenarioEditor* como herramienta visual para la especificación de dichos *escenarios*.

- Se brindan herramientas de administración y de colaboración:

  - *ScenarioManager*: permite descargar, instalar y eliminar *escenarios*.

  - *UserArtefactManager*: permite descargar, instalar y eliminar *augmenters*, *components*, *datacollectors*.

Al final vemos que se obtiene una arquitectura sumamente extensible donde cualquiera de los artefactos concretos puede ser desarrollado por cualquier usuario que posea las capacidades necesarias y luego compartido con el resto de los usuarios, logrando así un desarrollo basado en *crowdsourcing*.

Capítulo 5

Tareas de usuarios en la Web soportadas con Web Augmentation

5.1. Motivación e introducción al enfoque de *escenarios*

En el presente capítulo se abarcará con mayor profundidad el concepto de lo que se llama, en el contexto de esta tesis, *escenario*. La idea principal tras este concepto es crear complejos procesos mediante la composición de tareas individuales bien simples. Un proceso complejo podrá corresponder a una tarea abstracta mas amplia como ser la organización de un viaje. Para comprender que esta tarea abstracta es realmente la composición de tareas individuales veamos, a través de un ejemplo, una secuencia reducida de actividades, tareas o pasos que un usuario realizaría para organizar un viaje (reservar hotel, vuelos y conseguir información turística). El ejemplo puede verse en la tabla 5.1:

Cuadro 5.1: Lista de tareas de usuario y sus respectivas tareas específicas

Tarea abstracta de usuario	**Tarea**	**Datos utilizados**
Reservar hotel	Ingresar a un sitio de reserva de hospedaje	url

-	Ingresar destino	campo del formulario, valor ingresado
-	Ingresar fecha inicial	campo del formulario, fecha inicial
-	Ingresar fecha final	campo del formulario, fecha final
-	Enviar formulario de búsqueda	Botón del formulario
-	...	...
Pagar habitación	Seleccionar tarjeta de crédito	menú de tarjetas, opción elegida
-	Ingresar numero tarjeta	input de formulario, numero de tarjeta
-	Ingresar código de seguridad	input de formulario, código de seguridad
-	Ingresar fecha vencimiento	input de formulario, fecha
-	...	...
Comprar pasajes aéreos	Ingresar a un sitio de vuelos	url
-	Ingresar destino	campo del formulario, destino
-	Ingresar fecha ida	campo del formulario, fecha ida
-	Ingresar fecha vuelta	campo del formulario, fecha vuelta
-	...	...
Pagar tickets	Seleccionar tarjeta de crédito	menú de tarjetas, opción elegida
-	Ingresar numero tarjeta	input de formulario, numero de tarjeta
-	Ingresar código de seguridad	input de formulario, código de seguridad
-	Ingresar fecha vencimiento	input de formulario, fecha
-	...	...
Buscar puntos de interés	Proveer URL	url
-	...	...

Buscar puntos de interés seleccionados en GoogleMaps	Proveer URL	url
-	...	...
Buscar información extra de los puntos de interés	Proveer URL	url
-	...	...

Aunque es una tabla incompleta, ya que se han obviado subtareas específicas intermedias, puede realizarse un análisis de la misma y así hacer algunas conjeturas:

- Cada tarea de alto nivel de usuario (como *Pagar habitación*) incluye varias subtareas sumamente especificas: Proveer una URL para entrar a un sitio, llenar un campo de un formulario, seleccionar una opción en un menú, enviar un formulario, etc. Estas son las que llamaremos tareas primitivas: tareas que describen lo que los usuarios hacen manualmente dentro de un Web Browser Byrne et al. [1999] Heath [2010].

- Cierta información es ingresada reiteradamente: información relativa a la tarjeta de crédito, destino, fechas, etc.

- Parte de la tarea, por ejemplo aquella referida a la búsqueda de información turística, esta desatendida desde el punto de vista del soporte que se le da al usuario. Es decir, solo se utilizan tareas primitivas para llevar a cabo estas acciones. Esto implica que el usuario realiza estas tareas *ad-hoc*: busca y detecta sus puntos de interés, posiblemente copia y pega los mismos en GoogleMaps para obtener información geográfica, busca en páginas oficiales de museos, buscará información en Wikipedia, etc.

El eje aquí es apreciar cómo componiendo estas tareas primitivas con tareas de augmentación podremos mejorar la experiencia del usuario. Entonces ha lugar un análisis de qué rol pueden tener en este contexto los componentes descriptos en el capítulo anterior:

- Si el usuario colecta en el *Pocket* información de “destino”, “fecha desde”, “fecha hasta‘”, esa información luego estará disponible en cualquier sitio Web para ser utilizado, por ejemplo, para llenar formularios.
- Si el usuario posee un *PIM*, la información acerca de la tarjeta de crédito puede estar disponible para completar formularios.
- Si el usuario colecta las instancias de “PuntosDeInteres” en el *Pocket*, entonces podrá ejecutar augmenters automáticamente: agregar links a GoogleMaps, a wikipedia, resaltar estos puntos de interés en los sitios visitados, etc.

Debe notarse que la complejidad y la riqueza en la composición radica en cuáles son las tareas que pueden componerse. Aunque se ha definido un conjunto estable de las tareas primitivas contempladas, siguiendo el estudio realizado por Byrne et al. [1999], no implica que este conjunto no pueda extenderse en un futuro. Debe quedar claro que en el ejemplo de la Tabla 5.1, solo se utilizan tareas primitivas, y aunque podrían automatizarse ello implicaría solo la automatización del uso del Web browser, pero no la automatización de artefactos de adaptación. Es por esto que en este enfoque se consideran, ademas de tareas primitivas, las tareas de augmentación.

Respecto a las tareas de augmentación contempladas, estas no tienen límites siendo que en realidad depende de los *augmenters* instalados en el Browser Web del usuario. De todas maneras, todo el proceso de definición del enfoque es mas complejo y es tratado en la siguiente sección.

5.2. Visión general del proceso de definición de *escenarios*

Como se ha dicho, se plantea la idea de composición como la especificación de una secuencia de tareas individuales (primitivas y de augmentación) la cual es formalizada en un DSL (presentado en las siguientes secciones de este capítulo) y guardado como un archivo XML. Cabe la posibilidad de que grandes *escenarios* estén compuestos por decenas de tareas, tanto primitivas como de augmentación.

Cuando esto sucede, la coordinación entre cada subtarea no necesariamente es la mas eficiente, pero esto no puede saberse a no ser que se realice un análisis de la tarea abstracta. Desde que un archivo XML donde se encontraría eventualmente la composición es difícil de leer y analizar este aspecto para un ser humano. Por ello se decide la utilización de modelos de tareas que, como se ha demostrado, permite un análisis profundo de las tareas involucradas en un modelo determinado. Para ello, cada tarea (primitivas y de augmentación) debe tener asociada la documentación necesaria para realizar este análisis: un modelo de tarea donde se describa la interacción del usuario con el sistema, para lo cual se ha utilizado la notación HAMSTERS Martinie et al. [2011].

5.2.1. Ejemplo de documentación

Para hacer mas concreta esta idea sigamos un ejemplo donde se define un nuevo *DataCollector*: *PlainTextDataCollector*, que permite colectar datos sin ser conceptualizados semánticamente en el *Pocket*. Una vez se ha definido el objetivo del mismo, podemos desarrollar el código JavaScript correspondiente resultando el que se muestra en la Figura 5.1.

```
function PlainTextDataCollector(pocket){
    this.setName("PlainTextDataCollector");
    this.setPocket(pocket);
    this.setMenuItem(null);
    this.load();
};

PlainTextDataCollector.prototype = new AbstractDataCollector();

PlainTextDataCollector.prototype.onShowingPopUpShowMyItem = function(was_collector_used){
    return this.targetElements["text"] != "";
};

PlainTextDataCollector.prototype.load = function(){
    this.registerContextMenuItem();
};

PlainTextDataCollector.prototype.createInstance = function(){
    var concept = this.pocket.getConceptByName("untyped");
    var instance = new TextPlainInstance(this.targetElements["text"],concept);
    return instance;
};

PlainTextDataCollector.prototype.getLabelForMenuItem = function(){
    return "Put it into Pocket";
};

```

Figura 5.1: Código de PlainTextDataCollector

Para este *DataCollector* se define el modelo de tareas de la Figura 5.2 que

describe cada una de las subtareas involucradas. Estas tareas pueden ser llevadas a cabo por el usuario, con interacción entre el usuario y el sistema, o bien directamente por el sistema.

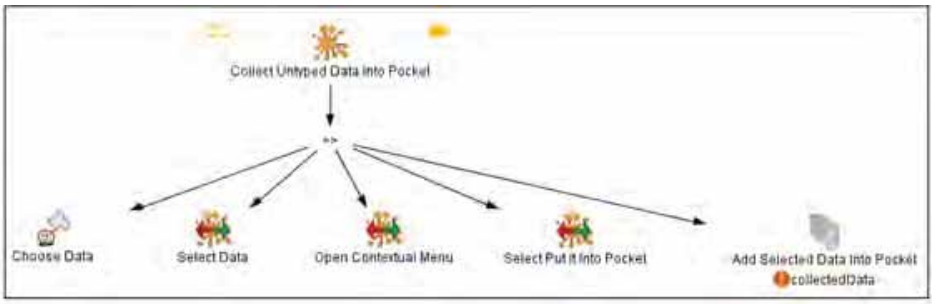

Figura 5.2: Modelo de tareas de PlainTextDataCollector

La Figura 5.2 muestra un modelo de tareas (con notación HAMSTERS) donde se muestra la secuencia de sucesos que tienen lugar cuando el usuario decide colectar texto plano en el *Pocket*. La primer tarea (*Choose Data*) corresponde a una tarea cognitiva, es decir, algo que el usuario decidió o escogió mentalmente: sin interacción con el sistema. Luego de haber escogido que porción de texto colectar, realiza tres tareas con interacción del sistema: selecciona el texto con el mouse, abre el menú contextual, selecciona la opción del PlainTextDataCollector. Finalmente una tarea realizada por el sistema almacena esa porción de texto dentro del *Pocket*.

Para realizar otro tipo análisis acerca de la interacción requerida por esta tarea, se provee también un diagrama de secuencia UML como muestra la Figura 5.3.

Aunque a primera vista parece tedioso tener que especificar cada uno de estos modelos para cada extensión del *framework* que un usuario desea desarrollar, la realidad es que según el *tipo* del mismo (*Augmenter*, *DataCollector*, *Component*, etc.) la descripción en términos de modelo de tareas y diagrama de secuencia es semejante, con lo cual estos modelos pueden reutilizarse.

5.2.2. Especificación de *Escenarios*

La posibilidad de especificar un nuevo *escenario* viene dada por la posibilidad que existe de combinar tareas primitivas o de augmentación a partir de los

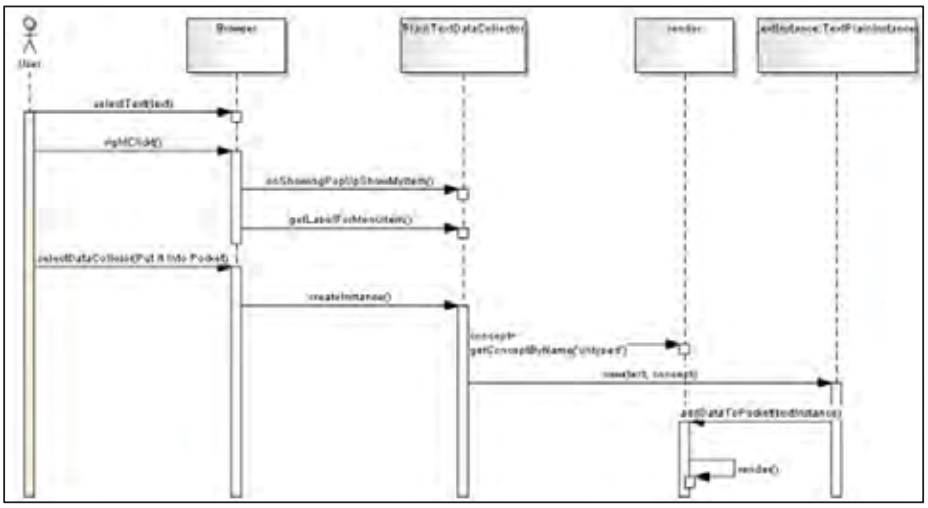

Figura 5.3: Diagrama de secuecnia de PlainTextDataCollector

modelos obtenidos siguiendo la documentación de ejemplo presentada en la subsección anterior. Siendo que la propuesta de *escenarios* es nueva y que a priori no existen tareas que puedan combinarse, es necesario empezar con la definición de un repositorio donde estén disponibles todas las tareas que pueden formar parte de la composición. Así es que el proceso completo involucra la definición de las tareas hasta la creación de nuevos *escenarios*.

Antes de detallar en profundidad cada fase del proceso, es preferible hacer un resumen del mismo: *Un* ***escenario*** *es la composición de una secuencia de tareas individuales (primitivas y de augmentación), la cual es formalizada por un DSL y almacenada en archivos XML para ser distribuidos. Una herramienta dedicada sirve para parsear los XML generados y reconstruir los escenarios en el Browser Web. Los escenarios creados de esta manera y teniendo la documentación descripta de cada tarea permitirá trasladarlo en modelos de tarea para soportar futuros análisis de la composición y llegado el caso realizar la optimización pertinente en la composición.*

El proceso completo de definición comprende las siguientes fases:

- Definición de tareas primitivas y tareas de augmentación: esta fase se enfoca en la inclusión de nuevas tareas en el *repositorio*; los pasos de **1** a **3** describen cómo las tareas primitivas son identificadas y luego modeladas

usando un modelo de tareas (notación HAMSTERS) para finalmente ser almacenadas en el repositorio de tareas. Las tareas de augmentación son definidas en los pasos **4**, **5** y **6**. La definición de cada tarea de augmentación, representada por artefactos que extienden el *framework*, incluye el código JavaScript, el modelo de tareas (también utilizando HAMSTERS) y el diagrama de secuencia como fue explicado en la subsección anterior. Una vez en el repositorio de tareas, las tareas de augmentación pueden ser integradas y utilizadas en una nueva composición. Cabe aclarar que aunque la creación de estas tareas requieren usuarios desarrolladores capaces de programar los artefactos, el trabajo una vez realizado beneficia al resto de los usuarios que acceden al repositorio.

- Composición: esta fase concierne a la definición de *escenarios* (por ejemplo: *planificación de viaje*), conseguida mediante la composición de las tareas disponibles en el repositorio. El enfoque es incremental, el usuario selecciona una unidad de composición (esto es cualquier tarea disponible en el repositorio) y crea un nuevo paso en la secuencia, paso **7**; este paso se repite hasta que el usuario considera finalizada la composición. Al finalizar la composición, se obtiene la información de la misma en el paso **8**. A partir de la información de la composición, en el paso **9**, el componente llamado *Factory* reusa la descripción de las tareas y el código correspondiente a las mismas.

- Análisis: el proceso de composición produce una secuencia de tarea que será un único *escenario*. De todas maneras, no se asegura que la secuencia original de tareas es óptima. Con tal motivo, se considera en el proceso global del enfoque una fase en la cual los modelos de tareas pueden ser exportados y analizados con la ayuda de la herramienta apropiada. La actividad de análisis esta soportada por la herramienta distribuida con la notación HAMSTERS Martinie et al. [2011]. Por supuesto que así como para el desarrollo de nuevos artefactos el usuario requiere de habilidades para tal fin, para el usuario requiere para esta fase aptitudes y experiencia en el modelado de tareas y procesos. De todas maneras, y aunque el análisis de los modelos de tareas es relevante para la mejora de los *escenarios*, no

se profundizará este aspecto en el resto de esta presentación.

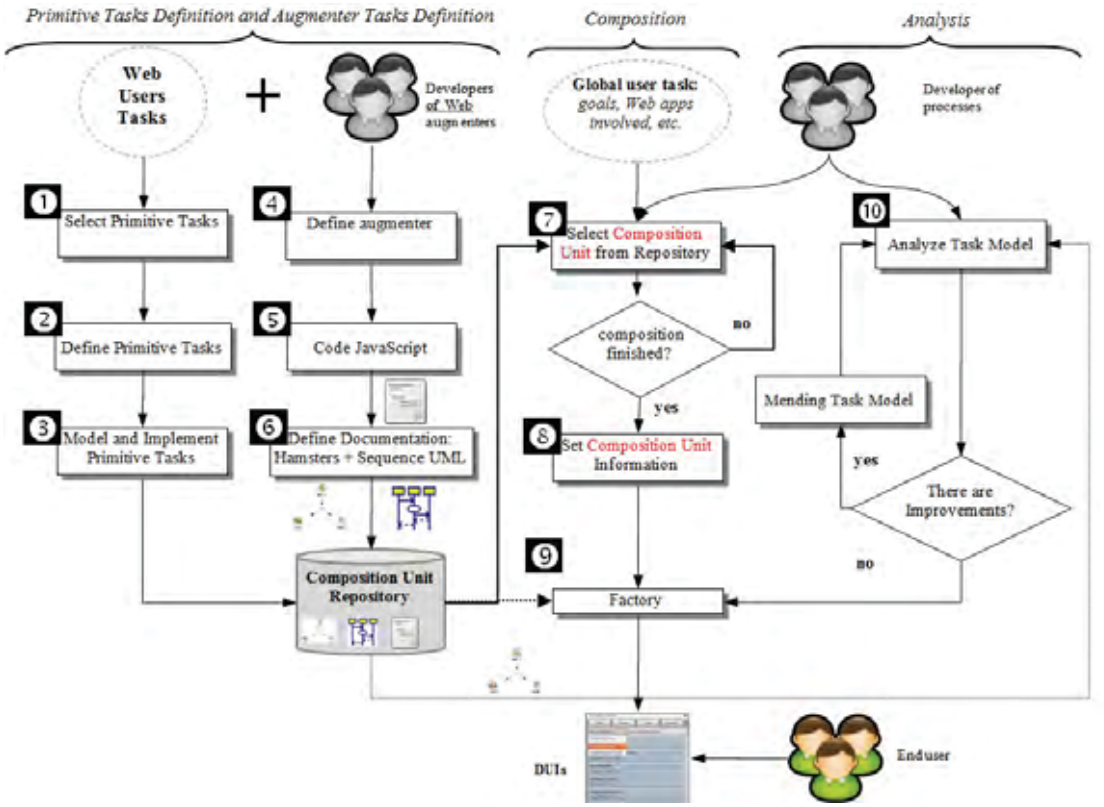

Figura 5.4: Proceso de definición de *escenarios*

Actualmente, el repositorio de tareas, es decir de unidades de composición, contempla aquellas tareas primitivas que han sido definidas por Byrne et al. [1999]), y además incluye un conjunto de tareas de augmentación que sirven de base inicial y que fueron desarrolladas para realizar los casos de estudio del Capítulo 6 y las evaluaciones del Capítulo 7, pero que puede crecer por el trabajo de los usuarios. El siguiente es el listado de las tareas de augmentación incluidas por defecto:

Cuadro 5.3: Lista de tareas de augmentación disponibles por defecto

Tarea	Categoría	Descripción	Ejecución
Untyped Data Collector	data collector	Agrega rápidamente información al *Pocket* sin peso semántico	usuario / sistema

Concept Instance Collector	data collector	Agrega información con peso semántico en el *Pocket*, por ejemplo, para el concepto "Ciudad " se puede colectar la instancia "Barcelona"	usuario / sistema
DOM Element Collector	data collector	Colecta un elemento específico del DOM en el *Pocket*	usuario / sistema
Remote Collector	data collector	Colecta información de una aplicación Web remota, es decir, que no esta en uso actualmente	sistema
Regular Expression Based Collector	data collector	Agrega en el *Pocket* información que corresponda a una expresión regular, por ejemplo, para colectar direcciones de correo electrónico	sistema
Microformat BasedData Collector	data collector	Agrega en el *Pocket* información descripta por Microformatos	usuario / sistema
CopyDataIntoInput	filling forms	Copia un valor en un campo de formulario. En ejecuciones bajo demanda del usuario, éste debe seleccionar el elemento del *Pocket* y luego el campo del formulario	usuario / sistema
Annotate Input With Microformat	filling forms	Anota los campos de los formularios con Microformatos	usuario / sistema
Microformat Form Filler	filling forms	Completa los campos según sus anotaciones de Microformato	usuario / sistema
Highlight	select data	Resalta elementos de información en los sitios Web. Puede ser ejecutado bajo demanda del usuario seleccionando un elemento del *Pocket*	usuario / sistema
Remove	select data	Elimina el elemento seleccionado	usuario / sistema
WikiLinkConvertion	navigation	Convierte elementos en links a los artículos de Wikipedia correspondientes. Por ejemplo, para el texto "Barcelona" crea un link al artículo de Wikipedia de Barcelona	user/system
IconifiedLink	navigation	Agrega links a GoogleMaps o Flickr	usuario / sistema

PopUpMessage	navigation	Muestra un mensaje popup al usuario donde puede sugerirse links para ser navegados	sistema
New Navigation Menu	navigation	Agrega un nuevo menú navegacional	sistema
Convert Pocket Into Menu	navigation	Transforma cada elemento del *Pocket* en un link relevante para la aplicación actual o tarea actual del usuario	sistema
Replace Image With Alt Text	accessibility	Reemplaza todas las imágenes por el texto del atributo *alt*	sistema
IncreaseTextSize	accessibility	Aumenta el tamaño del texto	sistema
Script Executer	*	Ejecuta un script JavaScript	sistema

5.3. DSL para la especificación de composiciones

En esta sección se abarca el DSL propuesto para especificar los *escenarios*. Básicamente, el DSL es un archivo XML que contiene la lista de tareas involucradas, las cuales pueden ser tareas primitivas o bien tareas de augmentación.

Los elementos del DSL son:

- Tareas primitivas: estas tareas se describen con el *tag*

  `<primitivetask> <primitivetask/>`

- Tareas de augmentación: estas tareas se describen con el *tag*

  `<augmentationtask> <augmentationtask/>`

- Tareas compuestas: son descriptas con el *tag*

  `<composedtask> <composedtask/>`

  y son utilizadas para agrupar un conjunto de subtareas en un bloque simple. De esta manera es todo el grupo el que puede ser marcado como: repetitivo, opcional o automático.

- Precondiciones: las precondiciones son especificadas para cualquier tarea y son utilizadas para condicionar la ejecución de la tarea en función de si se dispone de la información necesaria o no, para ellos debe utilizarse el tag

  `<precondition> <precondition/>`

  Hay dos grupos de precondiciones:

  - Precondiciones sobre la información colectada: con este tipo de precondiciones una tarea es ejecutada si y solo si la información colectada disponible satisface alguna condición. Por ejemplo, la precondición *PocketHasInstanceOf()* permite especificar que una tarea será ejecutada solo si el *Pocket* contiene un dato asociado con un tag semántico en particular. Por ejemplo, *PocketHasInstanceOf("PointOfInterest")* chequearía que una instancia de *PointOfInterest* está disponible en el *Pocket*. Otras precondiciones menos específicas están disponibles, como *PocketIsNotEmpty()*, para aquellas tareas que pueden ser ejecutadas con cualquier información disponible en el *Pocket*.

  - Precondiciones sobre las aplicaciones utilizadas: en algunas ocasiones, las tareas deberían ser ejecutadas en sitios Web específicos, sobre todo para aquellos casos en los que la tarea debería ser ejecutada automáticamente con información específica de un sitio Web en particular. Además son útiles para utilizarlas como condicionantes de detección de un *concern* en particular. Este grupo de precondiciones incluyen: *LastUsedWebApplicationIs(aURL)*, *WebApplicationInUse(aURL)*, *CurrentWebApplicationInUse(aURL)*, *WebApplicationUsed(aURL)* y *WebApplicationUsed(aURL, depthOfNavigation)*. La precondición *WebApplicationUsed(aURL)*, por ejemplo, chequea que una aplicación Web (especificada por el parámetro aURL) aparezca en el historial de navegación, sin importar en que posición. Un segundo parámetro puede definirse para esta precondición en pos de posibilitar al desarrollador del *escenario* que determina cuán pronto la aplicación en cuestión debió ser utilizada. Una versión corta es *LastUsedWebApplicationIs*, por su frecuente utilización. Esta sirve para chequear que una aplicación Web específica fue utilizada inmediatamente antes de la actual. *WebAppli-*

cationInUse sirve para chequear que actualmente una aplicación esta siendo utilizada, sin importar en que *tab* o pestaña del navegador este. Finalmente, la precondición *CurrentWebApplicationInUse* posibilita al desarrollador del *escenario* definir que una tarea se ejecute sí y solo sí actualmente una aplicación Web está en uso y además es la que actualmente esta visualizada.

- Postcondiciones: el tag

  `<postcondition> <postcondition/>`

  sirve para especificar el efecto de haber ejecutado una tarea. Las postcondiciones pueden ser *AffectCurrent()* (para especificar que la ejecución modificará el sitio Web actual, es decir, el sitio Web que el usuario esta visualizando), *AffectAny()* (para especificar que la ejecución no modificará ningún sitio), *AffectSubset(URLs)* (cuando la tarea esta especializada en un conjunto de sitios Web, incluso aquellas abiertas en *tabs* o pestañas no visualizadas) y *AffectAll()* (para especificar que la ejecución modificará cualquier aplicación Web en uso).

- Atributos: las tareas pueden requerir diferentes atributos para llevar a cabo sus objetivos. Por ejemplo, la tarea primitiva *Provide URL* requiere de un atributo llamado *URL*. Los atributos, con su nombre, su tipo, su valor y un valor de ejemplo son definidos para cada tarea en su especificación o documentación. La definición de atributos se realiza con el tag

  `<attribute> <attribute/>`

- Propiedades de repetición, opción y automatización: ambos tipos de tareas, primitivas y de augmentación, tienen tres propiedades intrínsecas. Una propiedad de repetición para especificar que la tarea puede ser ejecutada mas de una vez antes de seguir con la siguiente tarea definida en la secuencia, propiedad *repetitive*. La tarea puede ser marcada como opcional mediante la propiedad *optional* en pos de posibilitar la ejecución de la próxima tarea sin haber completado la actual. Y una propiedad de automatización, especificada con por *automatic*, para que el el *ScenarioPlayer* intente ejecutar

automáticamente dicha tarea. Cuando esta última propiedad esta habilitada, toda la información de los atributos requeridos por la tarea deben ser provistos con antelación. No es necesario especificar estas tres propiedades ya que en estos casos tendrán asociados valores por defecto.

En el código XML mostrado en el Listado 5.1 se muestra un ejemplo simple de un *escenario* descripto con el DSL especificado. Este *escenario* es llamado *Point of Interest at Google Maps* y la idea principal es colectar instancias del concepto *Point of Interest* en Wikipedia mientras se agregan nuevos links hacia GoogleMaps. La adaptación realizada es la mostrada en la Figura 5.5, donde a la izquierda se muestra el artículo de Wikipedia adaptado, mientras que a la derecha se muestra lo que ocurre al navegar uno de los nuevos links.

Es necesario realizar una explicación de lo visto en el XML. Por ejemplo, la primer tarea *Provide an URL* es definida como manual (no automática), no repetitiva y con un atributo llamado *URL*. Note que este atributo tiene un valor asociado, el cual es una expresión regular que represente a un conjunto de URLs: aquellas que corresponden a un artículo de Wikipedia. Este valor puede ser utilizado para considerar la tarea como finalizada, ya que no se espera que cualquier aplicación sea cargada, sino que por el contrario, tiene que ser algún artículo de Wikipedia; y hasta que uno no sea visitado, la tarea no será completada.

Muchas veces una tarea no automática (por ejemplo colectar algo en el *Pocket* con un *DataCollector*) es seguida por una tarea automática (por ejemplo ejecutar un augmenter con el nuevo dato colectado). El DSL permite esta clase de dependencia mediante la agrupación de tareas utilizando $<$ *composedtask*/ $>$. Utilizando este tipo de construcción una secuencia finita de tareas puede ser manejada en conjunto en pos de marcar como repetitivo, opcional o automático a todo el grupo. En el código XML de ejemplo mostrado en el Listado 5.1, se ha utilizado para componer *ConceptInstanceCollector* y *IconifiedLink*. De esta manera, cada vez que una instancia de *PointOfInterest* es colectada, la tarea de augmentación *IconifiedLink* es ejecutada ya que está marcada como automática.

En el ejemplo también se han definido tanto pre como postcondiciones. Por ejemplo en la tarea de augmentación *IconifiedLink* se ha especificado la postcondición *AffectSubset*, siendo el valor del atributo la expresión regular que corresponde a los artículos de Wikipedia. También se utiliza la precondición *PocketHasInstan-*

ceOf solo para asegurarse que una instancia de *PointOfInterest* esta disponible en el *Pocket*. Claramente, otras precondiciones pudieron haber sido utilizadas en este ejemplo, como para verificar que una aplicación Web esta siendo utilizada a través de la precondición *WebApplicationInUse*.

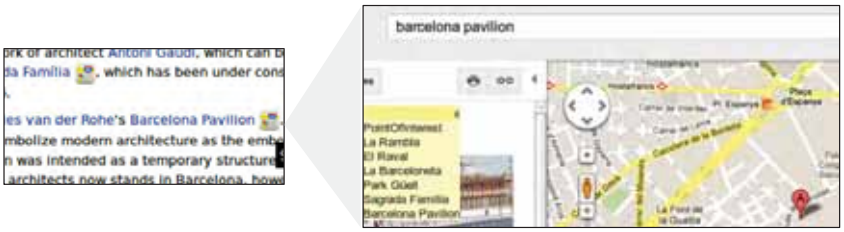

Figura 5.5: Resultado de adaptación para el *escenario* de ejemplo

Listing 5.1: *Escenario* para describir la composición de tareas usando el DSL

```
<?xml version="1.0" encoding="UTF-8" ?>
<scenario name="Point Of Interest at Google Maps">
  <tasks>
    <primitivetask id="Provide a URL" repetitive="False" automatic="False">
      <attributes>
        <attribute id="URL">
          <name>URL</name>
          <type>uri</type>
          <value>http://en.wikipedia.org/wiki/*</value>
        </attribute>
    </attributes>
    </primitivetask>
    <composedtask repetitive="True">
        <augmentationtask id="ConceptInstanceCollector">
          <attributes>
            <attribute id="concept">
              <name>PointOfInterest</name>
            </attribute>
        </attributes>
        </augmentationtask>
        <augmentationtask id="IconifiedLink" automatic="True">
          <attributes>
            <attribute id="targetData">
              <name>TargetData</name>
              <type>Concept</type>
              <value>PointOfConference</value>
            </attribute>
                <attribute id="targetWebSite">
                    <name>TargetWebSite</name>
```

```
                <type>uri</type>
                <value>http://maps.google.com</value>
            </attribute>
    </attributes>
        <postconditions>
            <postcondition>
                <type>AffectSubset</type>
                <attributes>
                    <name>URLs</name>
                    <type>uri</type>
                    <value>http://en.wikipedia.org/wiki/*</value>
                </attributes>
            </postcondition>
        </postconditions>
    </augmentationtask>
</composedtask>
<primitivetask id="Navigate a Link">
  <attributes>
    <attribute id="link">
      <name>Link</name>
      <type>xpath</type>
      <value>*</value>
    </attribute>
</attributes>
</primitivetask>
<augmentationtask id="CopyIntoInput" repetitive="True" automatic="False">
    <attributes>
        <attribute id="targetData">
            <name>TargetData</name>
            <type>InstanceOf</type>
            <value>PointOfConference</value>
        </attribute>
    </attributes>
</augmentationtask>
<augmentationtask id="CopyIntoInput" repetitive="True" automatic="False">
    <preconditions>
        <precondition>
            <type>PocketHasInstanceOf</type>
            <attributes>
                <name>targetData</name>
                <type>concept</type>
                <value>PointOfInterest</value>
            </attributes>
        </precondition>
    </preconditions>
  <attributes>
    <attribute id="targetData">
      <name>TargetData</name>
      <type>InstanceOf</type>
      <value>PointOfConference</value>
```

```
        </attribute>
    </attributes>
    </augmentationtask>
  </tasks>
</scenario>
```

Las condiciones asociadas al DSL pueden ser probadas a partir de la información provista por el Browser Web. Los otros elementos del lenguaje son fuertemente inspirados por las construcciones de la notación de modelo de tareas HAMSTERS Martinie et al. [2011]. El DSL especificado no es un equivalente a un modelo de tareas, por el contrario, tiene un solo propósito: permitir presentar las tareas en un cierto orden cuando se ejecuta el *escenario*. Para un análisis complejo de la composición de tareas, se recomienda trasladar el *escenario* a su correspondiente notación en HAMSTERS y subsecuentemente analizarlo con la herramienta apropiada. Sin embargo este aspecto de análisis aunque queda planteado, esta fuera del alcance de esta tesis.

5.4. Herramienta de soporte

En esta sección se muestran las herramientas desarrolladas para soportar el ciclo de vida completo de los *escenarios*: definición, reproducción y finalmente la exportación o publicación de los mismos. En esta sección sólo se ilustraran las herramientas con el objetivo central de dar a entender cada una de las funcionalidades brindadas por las mismas. El uso de las mismas en ejemplos concretos se aborda en el capítulo siguiente.

5.4.1. *ScenarioEditor*: Herramienta de creación

Para empezar, es conveniente entender cómo es que se define un nuevo *escenario*, lo cual puede lograrse con la herramienta de edición, *ScenarioEditor*, mostrada en la Figura 5.6. Esta herramienta esta implementada como un panel lateral del Browser Web, de modo que a la derecha de la pantalla el usuario puede observar normalmente los sitios Web mientras que simultáneamente edita el *escenario*. Como ya se sabe a esta altura de la presentación, el *escenario* es una secuencia de tareas y en el modo de edición esta secuencia se expresa del modo

natural: de arriba hacia abajo. Principalmente, la funcionalidad de la herramienta presentada incluye:

- Agregar nuevas tareas al *escenario*. Para ello, dos menús están a disposición del desarrollador del *escenario*: uno para escoger el tipo de tarea (primitiva o de augmentación) y el otro que, una vez seleccionada una opción del primer menú, permite seleccionar una tarea específica como se muestra en la Figura 5.7.

- Reorganizar y eliminar las tareas previamente seleccionadas. Para lo cual cada tarea posee (en este modo de edición) las opciones correspondientes: *Edit* (para editar las propiedades de la tarea), *Move Up* y *Move Down* (para ubicar la tarea en otra posición de la secuencia) y finalmente la opción *Remove* para eliminar la tarea del *escenario*.

- Editar las propiedades de cada tarea. Como puede apreciarse en la Figura 5.8, para cada tarea pueden especificarse los valores para sus propiedades. Asimismo desde la misma configuración de la tarea pueden agregarse precondiciones y postcondiciones cómo así también ingresar valores a los atributos especificados. El nombre de la tarea puede responder no a la identificación de la tarea específica (es decir a qué tarea primitiva o de augmentación corresponde) sino a una descripción de qué significa esa tarea en el contexto del *escenario*, lo cual es de gran importancia para tareas que deben ser ejecutadas manualmente.

Cuando el desarrollador del *escenario* desea terminar la especificación del mismo puede o bien reproducir el mismo para corroborar que realiza lo esperado (ver botón *Run*), guardarlo localmente (*Save*) o bien generar el equivalente en el DSL especificado y exportar el XML correspondiente (*Export*).

5.4.2. *ScenarioPlayer*: Herramienta de ejecución

Una herramienta similar a la de edición fue construida para la ejecución de los *escenario*: *ScenarioPlayer* Figura 5.9.

Figura 5.6: ScenarioEditor

En el caso de la Figura, puede apreciarse por poseer distintos colores de fondo, que las primeras dos tareas han sido ejecutadas mientras que aun no el resto de las tareas que conforman el *escenario*.

Por estar en ejecución, claramente, las tareas no pueden ser editadas, reubicadas ni eliminadas, con lo cual al crear la vista de cada tarea los botones correspondientes no son mostrados. Sin embargo, nuevas operaciones concernientes al estado de ejecución pueden ser realizadas:

- Conocer el estado de ejecución: en la parte inferior de la vista de cada tarea, puede leer el estado de ejecución en el que se encuentra. Mientras que para los primeras el estado es *Finished*, la tercera de ellas, "*Collect conference place*", posee el estado de *Executing*. El resto de las tareas no han empezado a ejecutarse (estado *Not started*) ya que aún se esta esperando que la tarea "*Collect conference place*" sea finalizada.

- Conocer la información de ejecución: al colocar el puntero del *mouse* sobre el

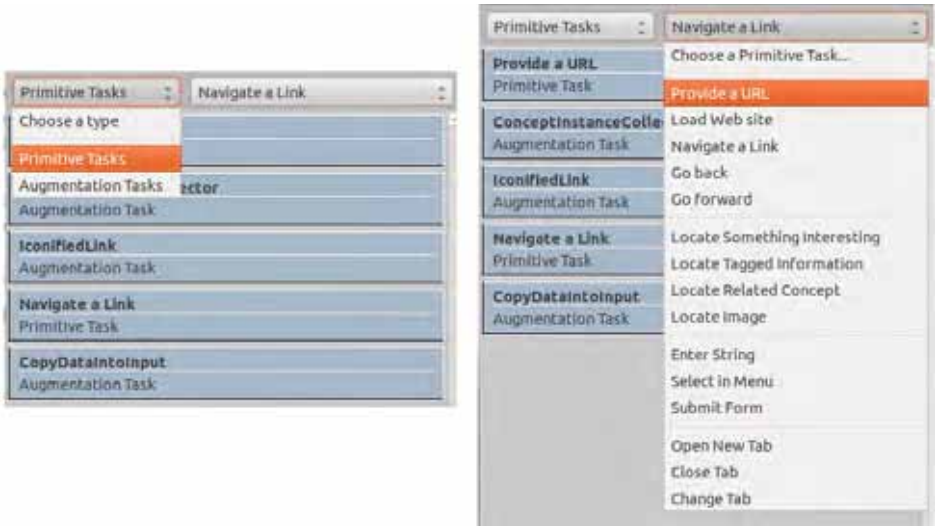

Figura 5.7: ScenarioEditor: Seleccionando una tarea específica para ser agregada

estado de las tareas finalizadas, un mensaje del estilo *tooltip* es mostrado con el valor que cada atributo tomó en la ejecución. En el ejemplo de la Figura, puede verse este mensaje para la tarea "*Collect conference city*". Note que en este caso la tarea (tarea de augmentación *TextInstanceDataCollector*) ha sido ejecutada con el concepto *ConferenceCity* y el valor *Capte Town*, lo cual está reflejado en el *Pocket* también visible en la Figura.

- Saltear tareas opcionales: Cuando alguna tarea es marcada como opcional, el objeto *ScenarioPlayer* ofrece al usuario que pueda pasar a la siguiente tarea de la secuencia. Este es el caso de la tarea que posee el estado *Executing* en la Figura, "*Collect conference place*", para la cual se muestra la operación *Skip*.

- Saltear tareas repetitivas: Cuando alguna tarea es marcada como repetitiva (pero no opcional), el objeto *ScenarioPlayer* ofrece al usuario que pueda pasar a la siguiente tarea de la secuencia luego de que la tarea haya sido ejecutada al menos una vez. En estos casos, el *ScenarioPlayer* también habilitará la operación *Skip*.

- Pausar y reiniciar: el usuario puede en cualquier momento pausar la ejecución del *escenario* e incluso re-iniciarla.

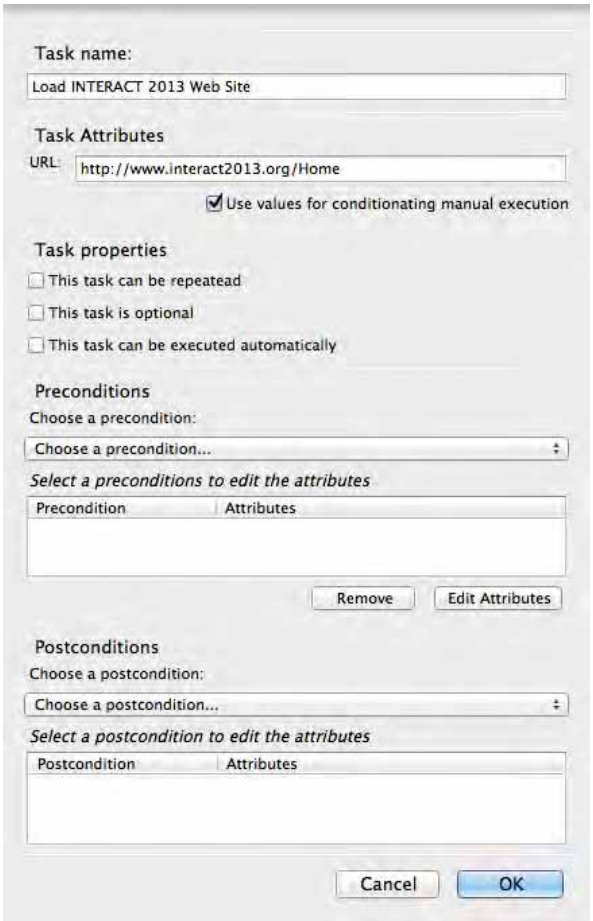

Figura 5.8: ScenarioEditor: Edición de las propiedades de una tarea

Cuando la ejecución ha finalizado, o bien si esta pausada, el usuario puede compartir el estado de la misma como muestra la Figura 5.11, lo cual se explica en la siguiente subsección.

5.4.3. *ScenarioManager*: Herramienta de administración

Al margen de la edición o la ejecución de un *escenario* debe existir un punto de entrada a partir del cual se dispare alguna de tales acciones. Este es el caso de *ScenarioManager*, cuya interfaz principal se muestra en la Figura 5.10.

La herramienta de administración es simplemente la forma que tiene el usuario para realizar acciones con los *escenarios* instalados. Claramente, también puede o bien crearse un nuevo *escenario* o bien importar un *escenario* desde el corres-

Figura 5.9: ScenarioPlayer: Ejecución de un *escenario*

pondiente archivo XML.

Cuando las tareas son realizadas por varios usuarios que forman parte de un grupo es común que haya una interacción entre sí para saber cómo, qué y en dónde realizaron las actividades en la Web. Por ejemplo, dos investigadores de la misma institución que viajen a la misma conferencia donde presentarán un artículo, posiblemente realicen la misma tarea solo variando ciertos datos utilizados. Para estos casos donde se quiera transmitir la experiencia en la ejecución de un *escenario* los usuarios pueden compartir (de forma privada, es decir, compartir a quienes ellos decidan) la ejecución de los *escenarios*. La Figura 5.11 muestra la ventana a partir de la cual el usuario puede compartir su ejecución, con toda la información concreta utilizados en la misma. Note que para ciertas tareas que requieran información que no se desea compartir, la herramienta permite especificarlo. En el ejemplo de la Figura 5.11 se han seleccionado las tareas de las que no se quiere compartir la información de su ejecución.

Finalmente, en una tercera Figura, Figura 5.12, se puede apreciar como se permite seleccionar la ejecución deseada: la del propio usuario o aquella que otros han compartido con él, para así poder ver/comparar/analizar la información de une ejecución en particular.

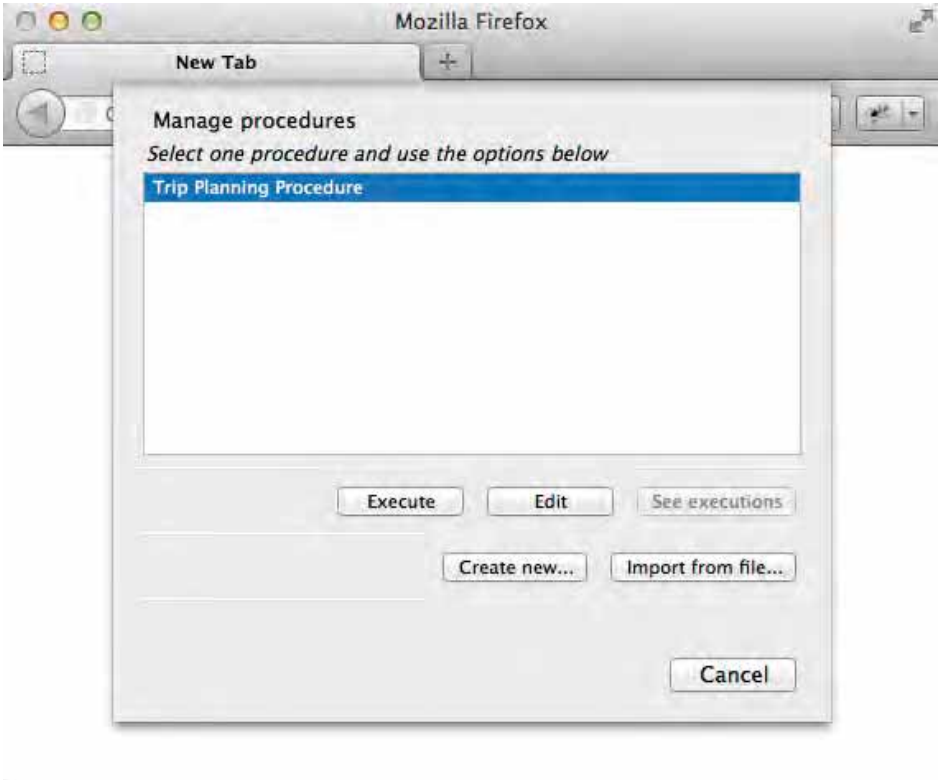

Figura 5.10: ScenarioManager

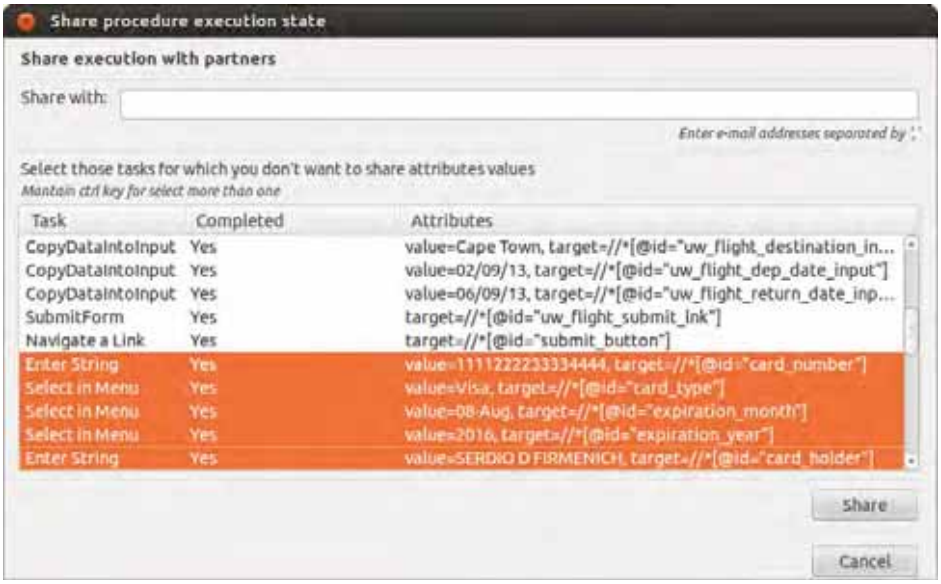

Figura 5.11: ScenarioManager: vista para compartir *escenarios* y estados de ejecución

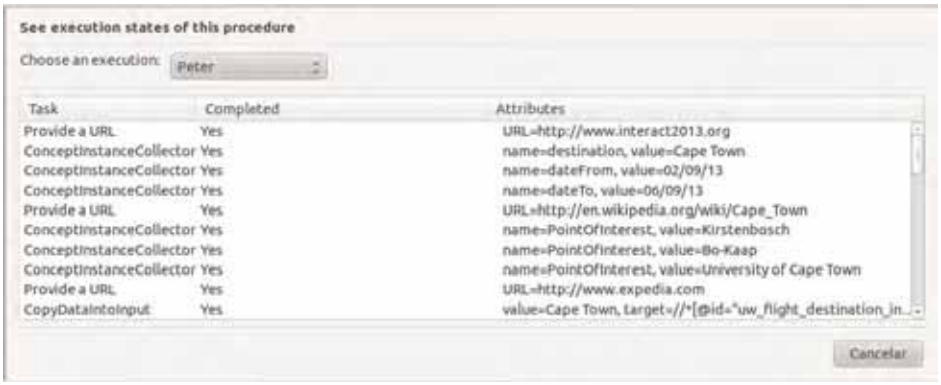

Figura 5.12: ScenarioManager: vista de ejecuciones de *escenario*

Capítulo 6

Casos de estudio

Utilizando como núcleo o *core* el framework CSN presentado inicialmente en Firmenich et al. [2011] se desarrollaron herramientas con objetivos específicos que estando integradas se distribuyen juntas (con la extensión para el Browser Web).

Las herramientas podrían dividirse, según los objetivos con que fueron definidas, de la siguiente manera:

- Reutilización de adaptaciones y modelos abstraídos del DOM.
- Utilización de *augmenters*, *DataCollectors* y *Pocket* para requerimientos volátiles.
- Utilización de *PIMI* para soportar la compleción de formularios.
- Ejecución de *escenarios* para automatización de adaptaciones.

En este capítulo se intenta mostrar el potencial de las herramientas mediante casos de estudios. Siendo que el conjunto total de las herramientas ofrece un amplio abanico de funcionalidades, y además considerando que bajo la linea general de soportar al usuario en sus tareas mediante adaptación en el cliente Web las herramientas fueron desarrollas con objetivos concretos y diferentes, se presentarán más de un caso de estudio cada uno enfocado en un subgrupo de las herramientas. De esta manera se mostrará en ejemplos focalizados el potencial de cada herramienta, sin ello significar que no existe un uso integrado de todas ellas.

Los casos de estudio planteados tienen el objetivo de mostrar:

- Caso 1: Reutilización de adaptaciones y modelos abstraídos del DOM.
- Caso 2, Navegación: Utilización de *augmenters*, *DataCollectors* y *Pocket* para requerimientos volátiles que se integran fácilmente en las tareas de los usuarios.
- Caso 3, PIMI: Utilización de *PIMI* para soportar la compleción de formularios en tareas bien identificadas, donde se utilizan *augmenters* específicos para adaptar una aplicación a un objetivo concreto.
- Caso 4, Escenarios: Automatización de adaptaciones y utilización de un sistema de colaboración para la concreción de tareas.

6.1. Reutilización de adaptaciones basada en modelos

En el caso de estudio presentado en esta sección se busca mostrar la posibilidad de reutilización de las adaptaciones ejecutadas en el cliente. Como ya se ha mencionado, esto se ha conseguido mediante la utilización de modelos que abstraen el DOM y a partir de los cuales las adaptaciones son desarrolladas en función de los conceptos definidos en dichos modelos Firmenich et al. [2010a], modelos fuertemente inspirados en el concepto de *Modding Interface* definidos por Díaz et al. [2008].

En el ejemplo de la Figura 6.1, se muestra la herramienta de definición de conceptos que abstraen elementos del DOM. En el caso del ejemplo se crea el concepto *SearchResult* para abstraer todos los resultados de búsqueda brindados por el buscador yahoo.com. Básicamente, el usuario selecciona el elemento del DOM que desea abstraer, lo cual es posible gracias a que se resaltan los objetos del DOM cuando el puntero del *mouse* se posiciona sobre estos. Luego de escoger el elemento (punto 1 en la Figura), una ventana de edición del concepto es mostrada (punto 2). Aquí el usuario ingresa el nombre del concepto que se esta creando (*SearchResult*) y selecciona las propiedades del mismo. Para escoger las propiedades se le facilita al usuario una lista con todos los elementos del DOM que están contenidos en el elemento del DOM que se seleccionó como concepto

(*los nodos hijos*). En este caso se ha seleccionado como propiedad el elemento *anchor* que corresponde al sitio resultante de la búsqueda, así es que en el punto 3 de la Figura, se muestra la configuración de la propiedad.

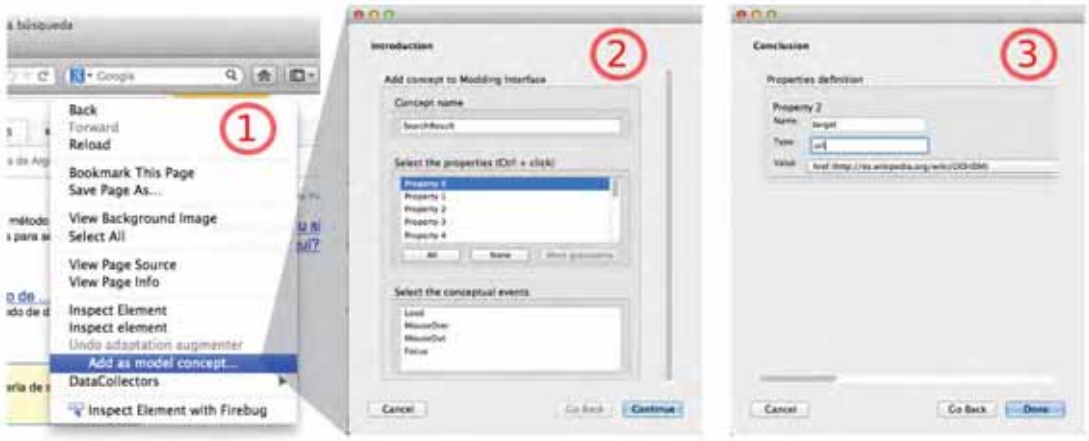

Figura 6.1: Creación de conceptos de modelo de abstracción del DOM

Finalmente, el nuevo concepto creado para el modelo de yahoo.com es mostrado por el *ModelViewer*, como puede apreciarse en la imagen superior izquierda de la Figura 6.2. En la imagen inferior izquierda puede verse el mismo concepto definido en el modelo correspondiente al buscador google.com. Note que aunque los conceptos son los mismos, entre un modelo y otro los valores utilizados para obtener el elemento del DOM varían (ver las expresiones *xPath* encerradas entre corchetes tanto al costado del nombre del concepto como de la propiedad).

Finalmente, la reutilización de la adaptación queda evidenciada por las imágenes superior derecha e inferior derecha de la Figura 6.2. Estas dos imágenes, muestran, correspondientemente, los sitios Web de yahoo.com y google.com luego de haber sido adaptados por el mismo artefacto: un script que resalta las instancias de *SearchResult* cuando éstos poseen en su propiedad *target* una URL que corresponde a un artículo de Wikipedia.

Es importante aclarar que en ambos modelos el concepto *SearchResult* se ha definido a partir del atributo *class* del elemento inicialmente seleccionado para crear el concepto Díaz et al. [2008]. Es importante ya que el valor del atributo *class* generalmente es compartido entre todos los elementos que semánticamente significan lo mismo (y que por ello tienen el mismo estilo CSS), es por ello que

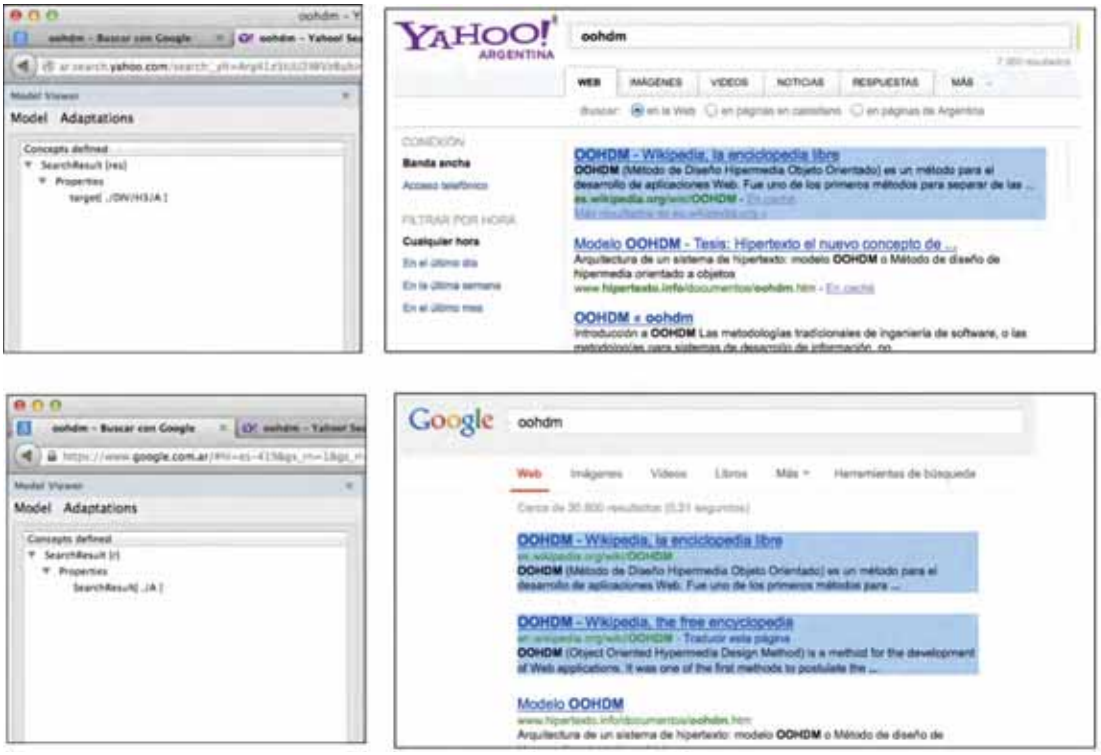

Figura 6.2: Reutilización de adaptación para resaltar resultados de búsqueda

aunque para definir el concepto *SearchResult* alcanzó con seleccionar uno de los elementos del DOM que representa un resultado de búsqueda, la realidad es que por cada uno de estos sitios las instancias de *SearchResult* son mas de una. Esto es gracias a los dos niveles que se logran con los objetos *Concept* y *ConcreteInstance* definidos en el Capítulo 4.

6.2. Navegando la Web utilizando *Augmenters*, *DataCollectors* y *Pocket*

Como se ha establecido en el Capítulo 3, que no haya un artefacto especializado para soportar la tarea actual del usuario no significa que éste no realice una tarea en particular. Para estos casos se brindan herramientas que activadas bajo demanda del usuario pueden satisfacer requerimientos volátiles de adaptación. En esta sección se muestran ejemplos de uso de *DataCollectors*, *Pocket* y *Augmenters*

mientras que una evaluación sobre las mismas es presentadas en el Capítulo 7.

En la Figura 6.3 se ve la interacción necesaria para agregar al *Pocket* nueva información. Aquí, pueden apreciarse distintas situaciones: i) ya existe información colectada en el *Pocket* bajo el *tag* semántico de *PointOfInterest* (note que esta información pudo haber sido colectada en el sitio Web actual o en cualquier otro visitado previamente); ii) el usuario decidió colectar una nueva instancia de *PointOfInterest*, y ha seleccionado el texto *Place de la Bastille*; iii) luego de seleccionar el *DataCollector* cuya etiqueta es *"Create instance into Pocket..."*, se muestra una ventana (ver a la derecha en la misma Figura) donde se le solicita al usuario en el nombre del concepto para el texto seleccionado.

Figura 6.3: Uso de *DataCollector* para colectar puntos de interés

Una vez colectadas varias instancias de *PointOfInterest*, el usuario decide navegar a Google Maps, sitio Web en el cual también se mostrará el *Pocket* con las instancias de *PointOfInterest* colectadas previamente, Figura 6.4. En la imagen derecha de esta Figura también vemos cómo el usuario ejecuta un *augmenter* desde el *Pocket*, en este caso: el *augmenter CopyIntoInput* (cuyo objetivo es copiar el valor del *Pocket* seleccionado el el campo donde el usuario haga un *click* con el mouse).

Finalmente, a la derecha de la Figura 6.4 se muestra que el usuario ha podido buscar en Google Maps el punto de interés seleccionado. Esta misma operación puede realizarla con cualquier otro punto de interés registrado en el *Pocket*.

Otro ejemplo de la ejecución manual de *augmenters* puede verse en la Figura 6.5, donde un usuario busca información acerca de los puntos de interés en un sitio Web de turismo. En este caso se ejecuta el *augmenter Highlight* usan-

Figura 6.4: Uso del *augmenter CopyIntoInput* para buscar puntos de interés en GoogleMaps

do como elemento del *Pocket* el concepto *PointOfInterest*. Cuando esto sucede, el *augmenter* es finalmente ejecutado mas de una vez, utilizando cada vez una instancia del concepto como parámetro. Así es que en una sola acción el usuario puede resaltar todos los puntos de interés colectados.

Figura 6.5: Uso del *augmenter Highlight* para encontrar todos los puntos de interés

6.3. Completando manualmente formularios con *PIMI*

En esta sección se mostrará el uso de la herramienta *PIMI* y de los *augmenters* de anotación de formularios. La idea central es mostrar cómo se facilita la tarea de usuario cuando, teniendo información en un espacio de información bien estructurado (esto es con estructura semántica) y además habiendo anotado los formularios con la misma estructura semántica, debe realizar tareas donde diversos datos deben ser ingresadas mediante formularios Firmenich et al. [2012]. Estas herramientas fueron desarrolladas, en sus primeras versiones por Gaits [2011].

En primer lugar resulta relevante entender cómo el usuario accede al *PIMI*. La Figura 6.6 (a) muestra la herramienta *PIMI*, en donde el usuario puede almacenar información estructurada semánticamente, en este caso mediante la utilización de Microformatos Khare and Çelik [2006]. *PIMI* es el espacio de información personal del usuario, que habilita a la reutilización de la información contenida para completar formularios en las aplicaciones Web utilizadas, como se muestra en la Figura 6.6 (b).

La información personal es organizada en forma de tarjetas que pueden ser amontonadas y manejadas como *posts-it* electrónicos conteniendo simples etiquetas (d). Cada registro puede ser expandido para así ver la información detallada, por ejemplo en (e) muestra todos los campos asociados a la dirección postal del trabajo del usuario. Además de los existentes, pueden agregarse nuevos registros en el *PIMI* haciendo uso de las opciones de Microformatos (f).

Figura 6.6: Uso *augmenters* de anotación de formularios

En la parte superior de la herramienta (c) se ubica un menú que le posibilita al usuario escoger entre la vista del espacio personal de información o bien, la vista de la herramienta de anotación que se explica a continuación. Debe quedar claro que las anotaciones son necesarias en pos de proveer a los usuarios con:

- Información contextual para ayudar a los usuarios a comprender que información se requiere para un campo en particular.

- Hacer interoperables los datos registrados en *PIMI* con los formularios de cualquier sitio Web.

La Figura 6.7 ilustra el proceso de anotación de un formulario disponible en el sitio Web Expedia.com. Aquí se muestra la herramienta de anotación en donde puede verse una lista de los Microformatos disponibles. Actualmente los Microformatos soportados son: *Hcalendar*, *Hcard*, *Hreview*, *Xoxo*, *Haddress*, *Hbank*, *Hcontact*, *Hidentity*, *Hlog*.

En pos de anotar un campo de un formulario, la única acción que el usuario debe llevar a cabo es seleccionar un tipo de Microformato del listado (por ejemplo la propiedad *value* del campo *email* embebido en el *Microformat Hcard*) y luego especificar, mediante *Drag&Drop*, el campo objetivo. Dicha acción puede verse a la izquierda de la Figura, donde el campo, que además de estar resaltado en verde por la herramienta, es señalado por la etiqueta *Target Input*. La anotación, además de un Microformato en particular, también puede incluir una descripción personalizada. Este proceso puede repetirse para cada campo del formulario y todas las anotaciones serán almacenadas en una base de datos dedicada.

Luego de haber realizado la anotación (y si la opción *"Show inputs detected semantic"* esta habilitada) se le mostrara la anotación al usuario cuando posicione el puntero del *mouse* sobre el campo anotado en forma de post-it virtual, como se aprecia a la derecha de la Figura 6.7 para el campo *"Email Address"*.

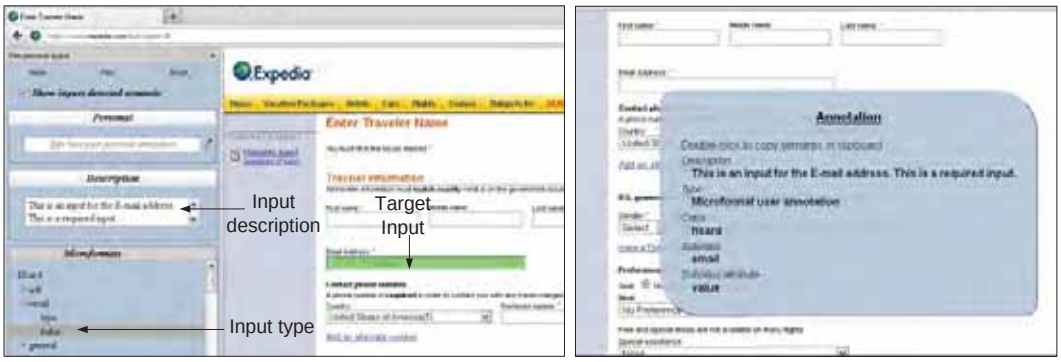

Figura 6.7: Uso *augmenters* de anotación de formularios

Una vez anotados los formularios, estos serán compatibles con la información estructurada en Microformatos almacenada en *PIMI*. Así es que luego puede haber lugar para las interacciones de i): completar formularios utilizando datos de *PIMI*; ii) crear registros en *PIMI* utilizando la información ingresada manualmente en formularios anotados. Estas dos interacciones pueden apreciarse de manera genérica en la Figura 6.8.

Figura 6.8: Uso de *PIMI* y formularios anotados

Por un lado, en la Figura 6.9, se muestra la interacción respectiva a la complección de los campos de un fomulario con información perteneciente a un registro de *PIMI*. Para ello el usuario mueve, mediante *Drag&Drop*, el registro deseado sobre el formulario que quiere completar. Cuando el usuario termina con la interacción *Drag&Drop* la herramienta se encarga de completar el formulario automáticamente, lo cual puede llevarse a cabo gracias a la compatibilidad semántica.

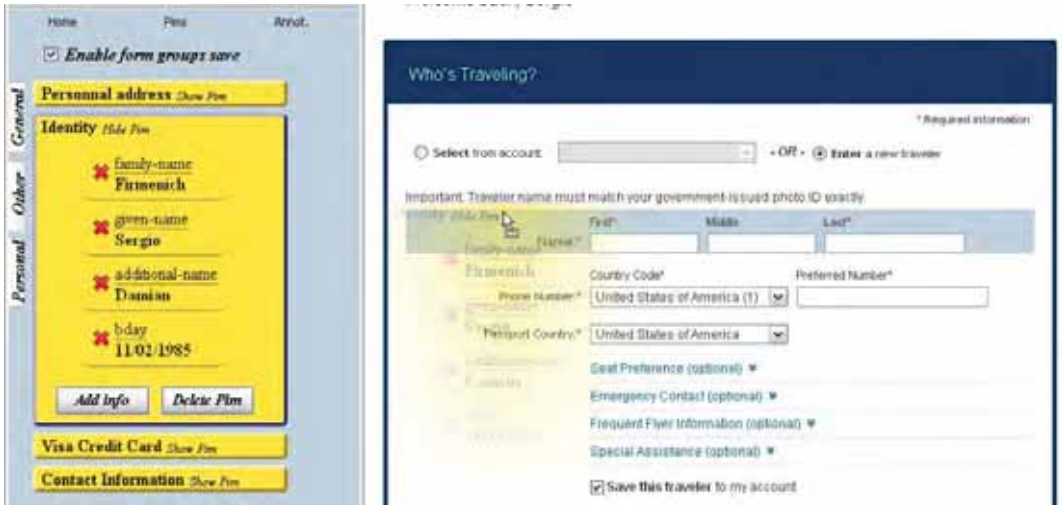

Figura 6.9: Uso de *PIMI* para completar formularios

Por otro lado, en la Figura 6.10, se muestra la interacción respectiva a la creación de un nuevo registro en *PIMI* con la información ingresada manualmente en un formulario anotado. En esta Figura, puede verse que la misma interacción *Drag&Drop* es utilizada en sentido contrario (desde un elemento del sitio Web hacia el espacio de la herramienta *PIMI*).

Figura 6.10: Uso de formularios anotados para guardar nuevos registros en *PIMI*

Esta última interacción es otra manera de completar el espacio de información personal, no completando los datos manualmente para un Microformato dado, sino utilizando la información de un formulario anotado.

Una evaluación referida al uso de estas herramientas es detallada en el Capítulo 7.

6.4. Ejecutando *escenarios*

En esta sección se presentan diversos ejemplos en donde haciendo uso de tareas de augmentación y combinando las mismas con tareas primitivas se soportan tareas de usuario y se muestra así el potencial del enfoque de *escenarios*.

En un primer ejemplo, mostrado en la Figura 6.11, vemos un *escenario* para soportar la tarea de comprar películas en amazon.com. En un primer paso, el usuario navega por imdb.com donde ingresa a varios sitios Web de distintas películas, como muestra el punto 1; en estos casos, el *escenario* colecta automáticamente los títulos de las películas visitadas en imdb.com. Cuando el usuario

decide comprar una película en particular siguiendo el link correspondiente hacia amazon.com, resaltado en el punto 2, se ofrece al usuario un mensaje *popup* (punto 3) para buscar los DVD en amazon.com correspondientes a las otras películas que el usuario había navegado en imdb.com.

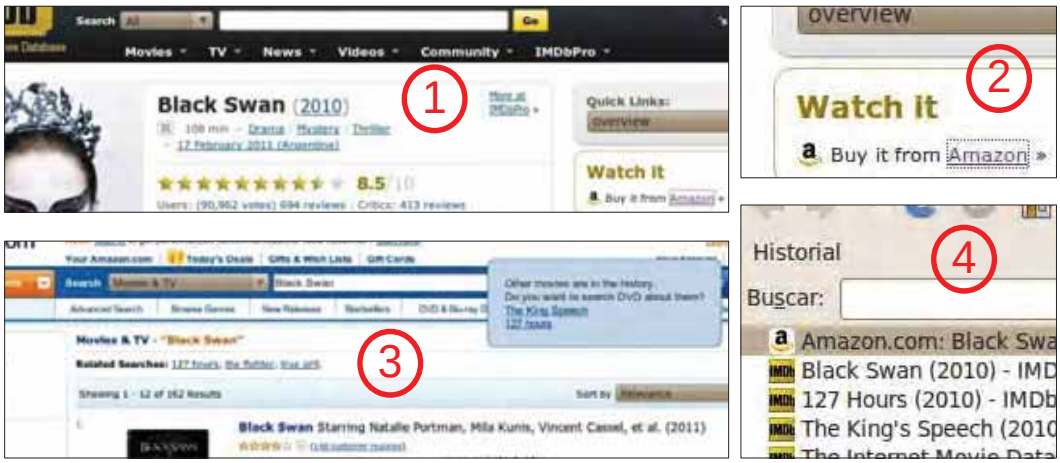

Figura 6.11: *Escenario* basado en IMDB.com y Amazon.com

En un segundo ejemplo de *escenario*, mostrado en las Figuras 6.12 y 6.13, se adapta a los artículos de Wikipedia automáticamente tomando en cuenta el historial de navegación.

Este *escenario* considera qué aplicación fue utilizada previamente: GoogleMaps o Flickr, y toma ese dato para agregar a cada instancia del concepto *PointOfInterest* un link, mostrado con el ícono de la aplicación correspondiente. Este simple ejemplo denota cómo pueden soportarse adaptaciones CSN, siendo que si ni GoogleMaps ni Flickr habían sido utilizadas, no se produce adaptación alguna.

En Figura 6.12, vemos que haber sido utilizada la aplicación Flickr ha sido mas recientemente que Google, entonces el link corresponde a la búsqueda de fotografías del punto de interés en Flickr.

En cambio, cuando la aplicación utilizada previamente fue Google Maps, el link, como se muestra en la Figura 6.13, sirve para navegar a GoogleMaps y allí buscar la ubicación del punto de interés correspondiente.

El mismo *escenario*, que contempla que el usuario pueda seguir agregando instancias de *PointOfInterest* en el *Pocket*, luego de que el usuario navega uno de

Figura 6.12: *Escenario* que agrega nuevos hiperenlaces a Flickr

Figura 6.13: *Escenario* que agrega nuevos hiperenlaces a GoogleMaps

los links agregados automáticamente, utiliza los valores en el *Pocket* para armar un menú navegacional en la aplicación destino, como muestra la Figura 6.14.

Figura 6.14: *Escenario* que convierte Pocket en Menu dedicado de Google-Maps

Finalmente, y en pos de mostrar un ejemplo donde se combinan acciones manuales y automáticas, se mostrará un escenario en el que habiendo colectado instancias de los conceptos *Destination*, *StartDate* y *EndDate*, se ejecuta un *escenario* para comprar pasajes en Expedia.com.

En primer lugar, como puede verse a la izquierda de la Figura 6.15, el usuario ha colectado la información y está en condiciones de ejecutar el *escenario*.

Figura 6.15: *Escenario* que convierte Pocket en menú dedicado de Google-Maps

Note que las primeras cuatro tareas en la secuencia están marcadas como automáticas, lo cual implica que, inmediatamente después de ejecutar el *escenario* (utilizando el botón *Run*), las cuatro tareas serán ejecutadas consecutivamente. En la Figura 6.16 se muestra el *escenario* ya en ejecución, donde las primeras cuatro tareas automáticas han sido ejecutadas, como así también la tarea primitiva (y manual) de completar el campo "Leaving From", el cual ha sido completado con el valor *Paris*. En esta Figura también puede apreciarse como la siguiente tarea (*Click Search Flights*) esta en estado de ejecución, ya que se espera que el usuario chequee los datos antes de realizar la búsqueda.

Note que esta tarea pudo haber sido automatizada, pero la combinación de tareas manuales y automáticas se considera esencial ya que el usuario, si bien es soportado por diferentes tareas de augmentación que pueden ser ejecutadas automáticamente, sigue teniendo el control. Así es que esta tarea en estado de

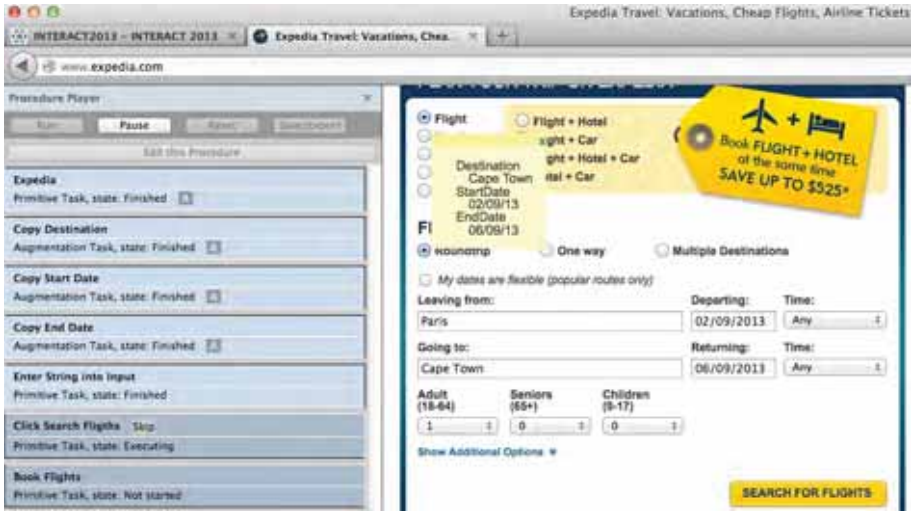

Figura 6.16: Ejecución parcial del *escenario*

ejecución, solo aguarda que el usuario haga *click* sobre el botón *SEARCH FOR FLIGHTS*.

Una vez realizada la búsqueda, el *escenario* queda en estado de ejecución de la tarea *Book Flights*, Figura 6.17. Esta tarea esta aguardando que la precondición *WebApplicationInUse* sea cumplida (ver en el contenedor de la tarea la inscripción *"Waiting for a Web application"*). Esto se debe a que el *escenario* no fue definido para soportar la selección de los vuelos resultantes de la búsqueda, por lo tanto, hace falta que el usuario realice dicha selección manualmente, y una vez realizada, llevará a la página que dará por válida la precondición.

Una vez la tarea *Book Flights* es ejecutada, podemos ver como sigue la ejecución del *escenario* en la Figure 6.18, en donde mediante una secuencia de tareas manuales el usuario completa los campos que solicitan información de la tarjeta de crédito con la que abonarán los pasajes aéreos.

Cabe destacar que luego de finalizada la ejecución del *escenario*, todas las informaciones con las que se ha ejecutado cada tarea quedan almacenadas. Con lo cual, puede editarse el *escenario* marcando para ejecución automática tareas que en la primer ejecución fueron manuales. Por ejemplo, las últimas tareas referidas al ingreso de los datos de la tarjeta de crédito podrían ejecutarse automáticamente en la próxima utilización del *escenario*, ya que se dispone de la información necesaria. Lo cual significa que este *escenario* es reproducible en cualquier otro

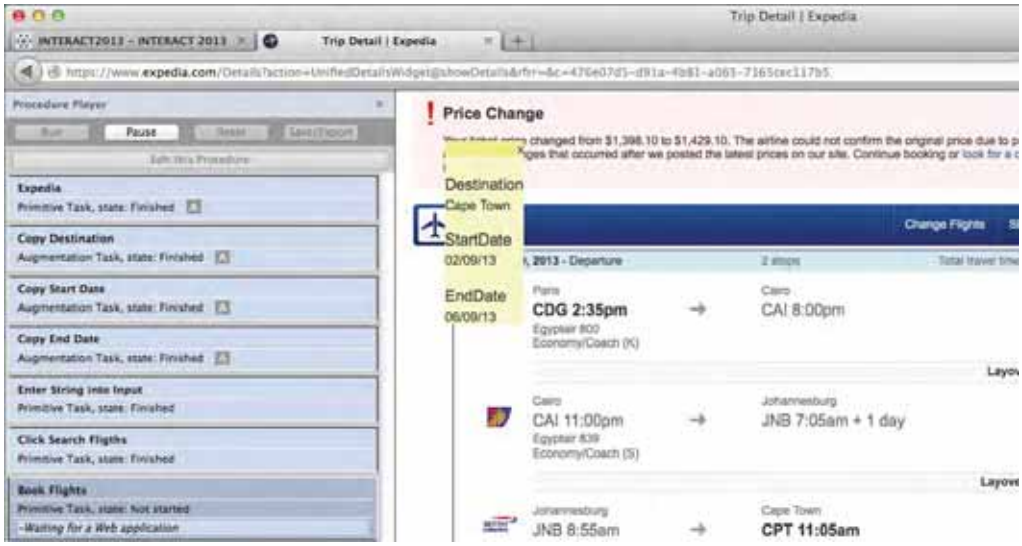

Figura 6.17: Ejecución parcial del *escenario*, aguardando por precondición

momento en el que se colecten en el *Pocket* otros valores para los conceptos *Destination*, *StartDate* y *EndDate*, sin importar en que sitios Web esta información es colectada. Además, la compleción de algunos formularios puede ser realizada automáticamente por augmenters que utilicen la información disponible en *PIMI*, siempre y cuando, claro, los formularios de los sitios Web involucrados posean anotaciones relacionadas.

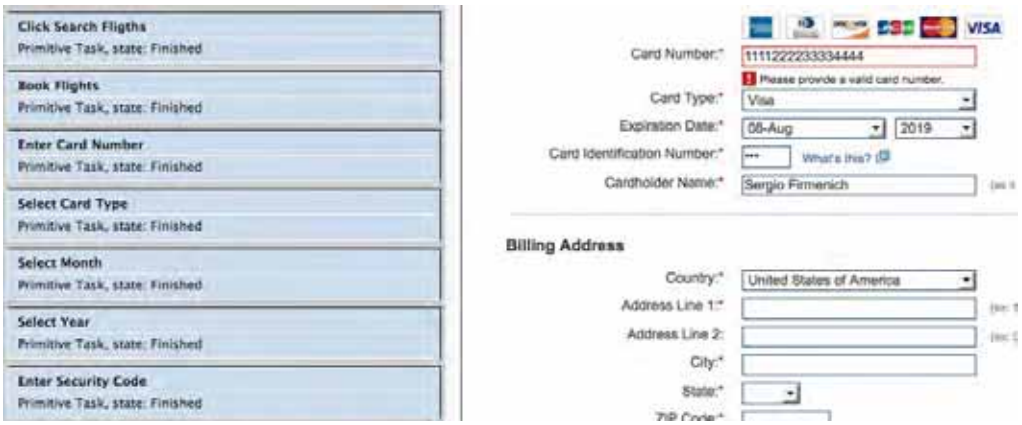

Figura 6.18: Ejecución parcial del *escenario*, completando datos de tarjeta de crédito

Capítulo 7

Evaluación

La factibilidad de los conceptos definidos en el contexto de esta tesis ha sido demostrada por herramientas concretas que fueron ilustradas en secciones previas.

Sin embargo, por sí mismas no son una prueba real de que finalmente favorecen al usuario en el desarrollo de sus tareas. Así es que cómo parte del desarrollo de esta tesis se han elaborado diferentes experimentos para evaluar tanto los conceptos como herramientas definidas.

Las evaluaciones fueron realizadas a partir de distintas hipótesis. En la subsección 7.1 se muestra la evaluación realizada bajo la hipótesis de que perciben de buena manera los artefactos de adaptación utilizados en el cliente y que estos *augmenters* su experiencia en la navegación.

En la subsección 7.2 otra evaluación es presentada para validar la hipótesis de que utilizando *augmenters* de anotación de formularios mejora la *performance* al realizar las tareas. Para este último estudio fue utilizado un modelo de simulación basado en GOMS-KML.

Las evaluaciones realizadas son detallas particularmente en cada una de las secciones de este capítulo.

7.1. Evaluación del uso de *augmenters*

En pos de investigar la usabilidad de los *augmenters*, el *Pocket* y los *DataCollectors*, se han realizado pruebas empíricas de usuarios con once participantes.

El caso de estudio abarca los siguientes *augmenters*: *Highlight* (para cambiar el color de información relevante), *WikiLinkConverter* (para convertir texto plano en links a Wikipedia), *CopyIntoInput* (para completar formularios con los datos previamente colectados en el *Pocket*). El *DataCollector* puesto a disposición de los usuarios para colectar información fue *TextInstanceDataCollector*.

La prueba de usuarios estuvo apuntada a investigar:

- Aspectos de usabilidad de las herramientas.
- Efectividad: si los usuarios pudieron completar las mismas utilizando las herramientas.
- Eficiencia: si los usuarios mejoran la *performance* en la ejecución de las tareas.
- Satisfacción: cómo los usuarios perciben a las herramientas.

7.1.1. Configuración, método y procedimiento

La prueba consistió, básicamente, en observar la *performance* de los usuarios mientras ellos desarrollaron sus tareas Rubin [1994].

El estudio fue realizado en una habitación controlada y equipada con Windows 7 y Firefox como Web Browser. Cabe aclarar que el *plug-in* correspondiente al *framework* y los *augmenters* fueron instalados previamente. No se utilizaron elementos de grabación pero cada usuario fue observado por un evaluador que tomó notas durante la totalidad de la prueba.

Una vez todos los participantes estuvieron reunidos, obtuvieron una charla introductoria acerca de los objetivos del estudio y tanto usuarios como evaluadores firmaron el acuerdo de privacidad. Los participantes, además, completaron un formulario que fue utilizado para obtener información demográfica y su experiencia en el uso de aplicaciones Web.

Finalmente se realizó una demostración (entre 2 y 5 minutos) de las herramientas a utilizar. Durante este período todas las preguntas relacionadas al framework y los componentes del mismo fueron respondidas por el líder del experimento. Desde este paso del experimento en adelante, no se permitió responder preguntas de los usuarios.

La configuración inicial fue sobre la exhibición de arte *De Stijl et Mondrian* [1]. La prueba empezó diciéndoles a los usuarios que consideren el siguiente escenario:

Asuman que están organizando un viaje a París para visitar una exhibición de arte llamada De Stijl et Mondrian*. Para completar sus objetivos, se les pide que realicen cinco tareas en diferentes sitios Web. Ustedes pueden usar la herramienta que crean mas útil.*

Una vez los usuarios estuvieron preparados, las tareas fueron presentadas una por una. Se les solicito a los usuarios en el comienzo de la tarea, que de ser posible, intenten pensar en voz alta durante la compleción de la tarea (lo que se conoce como procotolo de *thinking aloud*) Lewis [1982]. Si los participantes naturalmente pararon de pensar en voz alta, no se los fuerza a que continúen.

Después de que los usuarios terminaron cada tarea, se les solicitó que punteen la dificultad de la tarea considerando un rango de 1 a 5 (significando 1 *muy fácil* y 5 *muy difícil*).

Las tareas fueron:

1. Colectar datos requeridos para planificar el viaje incluyendo: fechas, palabras clave y direcciones.

2. Reservar hotel en París, en un lugar cercano a la exposición para el fin de semana del 18 de Febrero de 2011.

3. Seleccionar un hotel en el vecindario de *Les Marais*.

4. Guardar información sobre el hotel.

5. Crear una relación entre el sitio Web de la exhibición y el sitio Web Wikipedia.

Eficiencia y efectividad fueron respectivamente medidas en términos de tiempo en el que las tareas fueron completadas (eficiencia) y el número de tareas realizadas satisfactoriamente (efectividad). La satisfacción de usuario fue evaluada mediante un cuestionario incluyendo: qué les gusto de las herramientas utilizadas, qué no les gusto, si en algun momento tuvieron una experiencia dificultosa

[1] Disponible en: http://www.parisinfo.com/sorties/1193219/de-stijl-et-mondrian, último acceso: 16 de Diciembre de 2012

durante el uso de las herramientas, cuáles tareas fueron difíciles de llevar a cabo, si fueron conscientes en el uso de las herramientas, qué tipos de adaptaciones realizaron sobre los sitios Web, si les gustaría cambiar algo del estado actual de las herramientas, y si les gustaría que algo no cambie.

El estudio se concluyó al final con una entrevista y un cuestionario SUS (*System Usability Scale*) Brooke [1996]. El cuestionario SUS ha sido como un complemento de la observación siendo que es una herramienta ampliamente utilizada para realizar evaluaciones de usabilidad, con lo cual puede utilizarse para realizar comparaciones.

7.1.2. Participantes

Para este primer estudio de usabilidad de las herramientas mencionadas, se ha focalizado en usuarios con experiencia porque se asume que usuarios de este tipo son mas adecuados para formular necesidades específicas de adaptación de los sitios Web que los usuarios novatos. Incluso más, basado en la previa experiencia en navegación Web se podría obtener información de calidad acerca de si las herramientas pudieron ayudarlos o no en realizar las tareas, ya que los usuarios avanzados pueden comparar con la navegación Web tradicional.

Once participantes fueron reclutados (seis hombres y cinco mujeres, con edades entre 23 y 46 años de edad). Entre los participantes se encontraron: cuatro profesores, tres estudiantes de doctorado, dos estudiantes de grado y dos técnicos. Todo los participantes tenían experiencia como usuarios de la Web (mas de 5 años); utilizando la misma como parte de sus actividades diarias (con un promedio de 4.1 horas de navegación Web por dia, SD=2.4 horas). Por propósitos del experimento, el grupo seleccionado de usuarios fue homogéneo por lo que se han probado las herramientas con un limitado grupo de participantes, solo once, lo cual sigue estando acorde con las prácticas para experimentos de prueba con usuarios, como es descripto por Rubin [1994].

7.1.3. Resultados

Los once participantes usaron las herramientas presentadas durante el periodo de entrenamiento para realizar las tareas. Los usuarios completaron la prueba en,

aproximadamente, 37 minutos (SD=9 minutos) incluyendo la sesión del periodo de entrenamiento, la presentación a los usuarios del escenario de prueba y el tiempo requerido para completar los cuestionarios.

La Tabla 1 resume los descubrimientos en términos de:

- Grado de éxito considerando el número de usuarios que completaron satisfactoriamente la tarea utilizando un *augmenter*.
- Tiempo requerido para utilizar cada *augmenter* (promedio de tiempo y desvío estándar).
- Dificultad percibida, considerando un rango entre 1 y 5, siendo que el menor puntaje es el mejor.
- Comentarios de los usuarios.

Vale la pena mencionar que algunas tareas requerían el uso combinado de *augmenters*, aun así, la Tabla 7.1 solo reporta el tiempo gastado por los usuarios para usar un *augmenter* en la primer ocurrencia en la secuencia de las tareas. El desvío estándar puede indicar que algunos usuarios necesitaron mas tiempo para comprender el uso de las herramientas y además utilizarlas satisfactoriamente. Esta hipótesis puede estar relacionada con la dificultad percibida, la cual fue dada por todos los participantes sin importar el hecho de haber realizado la tarea satisfactoriamente o haber fallado en el intento. Otros estudios son requeridos para explicar este resultado, pero podemos esperar que el desvío estándar sea menor una vez los usuarios hayan utilizado las herramientas con mayor frecuencia. Cabe destacar que el promedio de tiempo solo toma en cuenta los usos satisfactorios de las herramientas.

La herramienta *DataCollector* en conjunto con el *Pocket* fueron las mas exitosas en el uso por parte de los participantes; ésta fue considerada muy útil y considerada como una buena alternativa a los *post-its*.

De todas maneras, el grado de éxito varió en el uso de cada *augmenter*. *CopyIntoInput* fue considerado muy fácil de utilizar por los participantes y fue utilizado con éxito por diez de ellos (90,9 %).

El *augmenter HighlightAugmenter* (con un grado de 72 % de uso satisfactorio, 8 participantes) fue considerado como fácil de utilizar, pero 5 usuarios se quejaron

porque este solo podía ser aplicado a las instancias exactas de las palabras previamente colectadas y además porque ellos no podían elegir el color incluso para resaltar con diferentes colores distintas piezas de información. Mientras que el uso de este *augmenter* requiere unas pocas acciones, algunos usuarios fallaron en usarlo satisfactoriamente, siendo que se pudieron haber confundido con el hecho de usar el *highlight* solo con datos disponibles en el *Pocket* en vez de ser aplicado directamente sobre el texto disponible en el sitio Web.

Los participantes estuvieron muy sorprendidos por el *augmenter* que permite convertir texto en links a Wikipedia, este es *WikiLinkConverter*; que aunque fue considerado extremadamente útil, el grado de éxito de este *augmenter* fue el mas bajo del estudio (con solo el 18 %) debido a dos cuestiones principales: el hecho de que los links a Wikipedia pueden ser solo creados desde elementos que han sido agregados al *Pocket* con un *tag* semántico y por la falta de *feedback* visual (por ejemplo, un ícono) indicando donde la acción de adaptación fue realizada. Solo dos usuarios tuvieron éxito en aplicar el *augmenter* (y navegar el nuevo link).

Cuadro 7.1: Resumen de resultados sobre el uso de *augmenters* por participante

Augmenter	**Exito**	**Tiempo** ($\overline{x}$)	**Dificultad percibida** 1 - 5	**Comentarios**
DataCollector	100 % (N=11)	12,3 s. (SD=3,7)	muy fácil - fácil ($\overline{x}$=1,6 - SD=0,7)	El uso de este *augmenter* requiere la selección de texto en un sitio Web, la selección del *augmenter* y el tiempo de ingresar un *tag* semántico (según el *DataCollector*). Los participantes usaron *tags* cortos (de 5 a 11 caracteres) para identificar datos pero el número de caracteres no fue analizado en el tiempo de completar la tarea.
Highlight Augmenter	72 % (N=8)	20,5 s. (SD=32,1)	fácil - neutral ($\overline{x}$=2,4 - SD=0,7)	El uso de este *augmenter* requiere la selección de datos previamente colectados en el *Pocket* y la selección de un *augmenter* del menú.
WikiLink Converter	18 % (N=2)	52,5 s. (SD=32,1)	neutral - difícil ($\overline{x}$=3,6 - SD=1,4)	El uso de este *augmenter* requiere la selección de texto en un sitio Web (que será convertido en un *anchor* para al *link*) y luego la selección del *augmenter* en el menú.
CopyIntoInput	90,9 % (N=10)	8,8 s. (SD=7,6)	fácil - neutral ($\overline{x}$=2,4 - SD=1,2)	El uso de este *augmenter* requiere la selección de información previamente colectada en el *Pocket* y además seleccionar (realizar un *click* con el mouse) en el campo del formulario que debe ser completado.

Obs.: Participantes en el estudio=11, N=número de participantes que usaron satisfactoriamente el augmenter, $\overline{x}$ = promedio, SD= desvío estándar.

Nueve participantes (81,8 %) mencionaron que la utilización de los *augmenters* mejoran su rendimiento al realizar las tareas; un usuario dijo que pudo haberlo hecho mas rápido sin utilizarlas y otro dijo no haber notado diferencia alguna. Esta percepción de los usuarios ha sido confirmada por el tiempo registrado durante la ejecución de las tareas utilizando las herramientas.

El estudio también reveló algunos problemas de usabilidad que motivaron desarrollos futuros de la herramienta. Por ejemplo, los usuarios solicitaban tener un indicador permitiéndoles distinguir dónde los *augmenters* han sido aplicados.

Intuitivamente, los usuarios trataron de colectar datos usando *Drag&Drop*, lo cual es un indicador para investigaciones acerca de nuevas interacciones con la herramienta.

Las sugerencias mas frecuentes para nuevos *augmenters* incluyeron: automatización en la compleción de formularios, creación de hiperenlaces a otros sitios que Wikipedia y automatización en el uso de *Highlight* en los sitios Web visitados.

Esta claro, que este tipo de automatizaciones no era parte de la evaluación, que estuvo mas orienta a probar que las herramientas servían para soportar los requerimientos volátiles de adaptación y soporte de tarea de los usuarios bajo mecanismos activados bajo su propia demanda.

7.2. Evaluación de PIMI y de *augmenters* de formularios

Otra de las herramientas que han sido objeto de estudio son aquellas orientadas a adaptar los formularios de las aplicaciones Web. Estas adaptaciones, realizadas por algunos del los *augmenters* descriptos en las secciones previas no modifican de manera directa lo que el usuario percibe de los sitios Web visitados, pero sí posibilita una nueva interacción con los elementos de información almacenados en el objeto *PIMI*.

En esta sección se presenta una evaluación cuantitativa de las tareas de usuario que involucran formularios Web de acuerdo con el modelo GOMS-Keystroke (KML) Card et al. [1983].

GOMS (***G****oals,* ***O****perators,* ***M****ethods,* ***S****election rules*) es un modelo formal usado para evaluar de modo riguroso cuán eficientemente una persona entrenada puede interactuar con un sistema de software dado. GOMS es un modelo humano de procesamiento de información que es construido sobre una detallada secuencia de operaciones que el usuario realiza con un sistema. La variante de este modelo llamada *GOMS-Keystroke (KLM)*, propone valores para acciones bien conocidas de los usuarios, de manera tal que es posible predecir qué hará un usuario experto en situaciones impredecibles. Por ejemplo, el promedio de tiempo para realizar la acción de tomar el *mouse* con la mano es de 0,40 segundos; mientras que hacer

un *click* sobre un campo de un formulario es de 0,20 segundos. Así una lista de tareas relativas a las acciones del usuario y su tiempo estimado es brindado por este modelo.

En este sentido, si se provee un escenario detallado de las acciones del usuario con y sin las herramientas (incluyendo todas las acciones), es posible usar GOMS-KML para predecir el rendimiento, es decir, la velocidad con la que algo es llevado a cabo con y sin las herramientas.

Debido a que no es pertinente toda la especificación de las tareas, en la Tabla 7.2 se presenta un resumen de los resultados obtenidos aplicando GOMS-KML sobre las tareas identificadas en el escenario *Planificar un viaje a Berlín*. En el Anexo B se encuentra el escenario en su totalidad. Solo a modo de entendimiento, el escenario responde al siguiente enunciado:

"Juan quiere ir de viaje a Berlin con su mujer por vacaciones, con tal motivo él esta comprando pasajes aéreos, reservando una habitación habitación de hotel y también reservando un automóvil en los siguientes sitios: expedia.com, booking.com y hertz.com. Al hacerlo, Juan completa tres formularios en dichos sitios Web con información sobre su viaje (esto es: fechas, ciudad de origen, destino) y también información las personas que viajarán (nombre, dirección, información de tarjeta de crédito, número de pasajero frecuente, licencia conducir, número de pasaporte, etc.). Juan sabe su información personal, pero no conoce la información del pasaporte de su mujer ni tampoco su número de pasajero frecuente."

La última linea en la tabla provee el tiempo estimado para el escenario completo. Se han comparado las tareas utilizando *PIMI* y sin utilizar *PIMI*. Como puede notarse, el tiempo estimado para algunas tareas en el escenario, como "Buscar Vuelos" y "Seleccionar Vuelos" son el mismo. Sin embargo, los usuario ahorrarán tiempo cuando reutilicen la información que esta disponible en el espacio de información personal ya que el mecanismo de *Drag&Drop* es más rápido que ingresar valor por valor en cada uno de los campos.

En la misma tabla, se puede apreciar también que nuevas tareas aparecen en el escenario donde se utiliza *PIMI*, ya que los usuarios eventualmente deben completar con información el mismo: información personal, datos de la tarjeta

de crédito, etc. Esto también incluye el tiempo de crear las anotaciones de los formularios de sitios Web; por ejemplo, crear las anotaciones (mediante las tareas de augmentación correspondientes) de los campos de la tarjeta de crédito lleva un tiempo estimado de 15 segundos. Mientras los usuarios gastan tiempo extra completando el espacio de información personal, luego ahorrarán tiempo ya que dicha información será reutilizada en futuras operaciones.

Cuadro 7.2: Rendimiento estimado de acuerdo al modelo GOMS-KML

Tarea	**Normalmente (sin PIM)**	**Con PIM**
Buscar vuelos en Expedia.com	6,3 seg	6,3 seg
Seleccionar vuelos en Expedia.com	32 seg	32 seg
Completar formulario con información de pasajero Expedia.com	26 seg	9,9 seg
Completar formulario con información de tarjeta de crédito Expedia.com	67 seg	11,9 seg
Agregar información de contacto personal en el espacio de información	-	27,5 seg
Agregar información de tarjeta de crédito en el espacio de información	-	41 seg
Anotar campos de formulario relacionados a la tarjeta de crédito	-	15 seg
Tarea total: planificación de viaje comprando vuelos (expedia.com), y reservando hospedaje (booking.com)	240,6 seg	143,6 seg

Debe notarse que, al margen de las estimaciones de tiempo, el enfoque y las herramientas utilizadas para esta evaluación ayudan a vencer limitaciones en los formularios existentes en los sitios Web como falta de ayuda contextual, etiquetas de campos no claras, etc.

7.3. Conclusiones

Los estudios realizados (secciones 7.1 y 7.2) muestran que el *framework* es utilizable por usuarios finales trabajando en tanto en tareas bajo requerimientos volátiles como tareas bien definidas en el caso del uso del PIMI.

Principalmente, el estudio de la sección 7.1 muestra que los usuarios aprecian este tipo de herramientas y, como se mostró en la sección 7.2, pueden ganar

tiempo en la realización de tareas repetitivas, .

Es importante notar que el *framework* posibilita la creación de un número aun mayor de herramientas y también explorar escenarios más complejos. Por ejemplo, el *framework* puede ser utilizado para construir herramientas de colaboración, como las mostradas en el capítulo 5. Aunque no se han realizado pruebas con usuarios al respecto sí se ha mostrado la factibilidad en la colaboración entre usuarios para realizar tareas en común.

La construcción de nuevas herramientas brinda la posibilidad de plantear nuevas hipótesis, basadas en, por ejemplo, cómo los usuarios finales perciben el uso colaborativo de adaptación basada en el cliente.

Otras evaluaciones sumamente relevantes apuntan a poder determinar con qué facilidad los desarrolladores pueden crear nuevos *augmenters*. Respecto a esto, se están llevando a cabo pruebas con 8 alumnos de UPS (*Université Paul Sabatier, Toulouse, Francia*), para poder analizar qué *augmenters* los alumnos desarrollan y que complicaciones encuentran. Hasta ahora, los resultados concretos brindados por las primeras fases del experimento muestran que han podido instalar y ejecutar correctamente el *framework* y las extensiones (*augmenters*) brindadas. Es importante destacar que este paso de la evaluación ha sido llevado a cabo sin asistencia ni entrenamiento.

Lo importante a destacar aquí, finalmente, es que varias evaluaciones podrían ser realizadas en el contexto de esta tesis, ya que el *framework* provisto brinda muchas posibilidades de extensión y que por ser éste un enfoque basado en *crowdsourcing* no lo solo es pertinente evaluar la utilidad de las extensiones desde el punto de vista de los usuarios finales sino también sería óptimo demostrar que hay una masa de usuarios que es capaz de desarrollar dichas extensiones. Sin embargo, las evaluaciones que sí se han realizado validan gran parte de la propuesta, mostrando además que es posible mejorar la experiencia del usuario mediante este tipo de técnicas.

Capítulo 8

Estado del arte

Durante el desarrollo de la presentación se han dado a conocer diversas razones por las cuales informaciones y servicios podrían ser integrados y/o adaptados para mejorar cómo el usuario accede a los mismos. Esta diferencia entre integración y adaptación ha hecho que distintas áreas de investigación se dediquen a temáticas similares pudiendo hoy en día diferenciar claramente entre propuestas de puramente de integración (*mash-ups* Yu et al. [2008]) y propuestas de adaptación (augmentación Web Bouvin [1999]).

En el enfoque de soporte a tareas de usuario presentado tienen lugar ambos aspectos (tanto integración como adaptación). Incluso así, los trabajos relacionados serán analizados por área para un mejor entendimiento. Note que aunque algunos enfoques de augmentación Web puedan incluir mecanismos de integración, la linea que separa a estos de un enfoque de *mash-up* se define, en general, por el hecho de modificar un sitio Web pero aun así permanecer en el mismo mientras que un enfoque de *mash-up* clásico creará una nueva aplicación especializada para una integración en particular.

Así es que en este capítulo se incluyen dos secciones principales, una para tratar enfoques de aplicaciones *mash-ups* y otra para tratar enfoques de augmentación Web.

Además una sección dedicada a la temática de modelado de tareas es incluida.

8.1. Mashups

Aunque parezca novedosa dado el incremento vertiginoso de propuestas de *mash-up*, la intención de *integrar* partes de distintos sitios Web en un nueva aplicación no es para nada nueva. Ya en el año 1998 Sugiura and Koseki [1998a] (ver también Sugiura and Koseki [1998b]) planteaba un enfoque de integración de contenidos orientado a la tarea del usuario. En este trabajo, los autores permitían, mediante el uso de herramientas visuales, que los usuarios escojan secciones de sitios Web relativas a una misma tarea (por ejemplo: leer las noticias del día) y así integrar todas esas secciones en una nueva aplicación.

Claro que la evolución de la Web y el incremento en los servicios provisto por las aplicaciones dio lugar a un nuevo tipo de integración que ademas de utilizar elementos de interfaz gráfica también contemplaba la utilización de datos obtenidos por el consumo de servicios. Todo esto para generar una nueva aplicación especializada.

La mayoría de los enfoques de *mash-up* facilitan al usuario sin capacidades de programación la construcción de estas nuevas aplicaciones mediante herramientas intuitivas como las presentadas por Wong and Hong [2007] y Daniel et al. [2009]. Una de las herramientas mas exitosas sea tal vez Yahoo! Pipes[1]; herramienta orientada no tanto a la integración de componentes de interfaz gráfica sino a la integración de datos. Existe una comunidad activa de usuarios de esta herramienta Jones and Churchill [2009] que una vez mas demuestra el interés de los mismos en personalizar su experiencia en la Web.

Uno de los principales problemas relacionados con *mash-ups* es que en general están basados en servicios Web. El hecho es que la mayoría de las aplicaciones Web no proveen este tipo de servicios para acceder a su funcionalidad o información, y es por esto que herramientas *mash-ups* que se basan en la integración de componentes de interfaz gráfica siguen siendo requeridos. Sin embargo, la falta de servicios Web puede ser solucionada como propone Han and Tokuda [2008]. En este trabajo, los autores proponen un enfoque de integración del estilo mash-up para integrar contenidos de aplicaciones Web de terceros. La idea principal es describir estos contenidos en el cliente para generar servicios Web virtuales (también

[1]Yahoo Pipes!, http://pipes.yahoo.com/, última visita 10 of Diciembre, 2012

soportados en el cliente) que pueden ser utilizados en composiciones posteriores.

Como se dijo con anterioridad, el objetivo principal de las herramientas de *mash-up* es integrar interfaces gráficas y/o datos en pos de generar una nueva aplicación. Claramente, esto elimina las versiones originales de las aplicaciones Web, lo cual puede no ser deseable por muchos usuarios, máxime cuando hoy en día la aplicaciones Web son sumamente dinámicas respecto a funcionalidad o contenidos: noticias, ofertas especiales, etc.

Note que las aplicaciones *mash-up* obtenidas con estos enfoques difícilmente consideran requerimientos volátiles de integración, por el contrario, suelen ser mas estáticas (en cuanto a su definición), lo que va en contraposición con algunos de los fundamentos planteados en el capítulo 3.

En este sentido, debe quedar claro que aunque existe una comparación lógica entre aplicaciones *mash-ups* y el enfoque de esta tesis, la realidad es que se basa mas en la intención de integrar diversos sitios Web, que en cómo esta integración se realiza. Sin ningún lugar a dudas la comparación mas exhaustiva debe realizarse con enfoques de augmentación Web como trata la siguiente sección.

8.2. Web Augmentation

En contraposición a los *mash-ups*, las herramientas de augmentación Web apuntan a modificar los sitios Web de preferencia de los usuarios, pero manteniendo las características intrínsecas de tales sitios. Usualmente, este tipo de herramientas son desarrolladas como extensiones de Browsers Web, aunque pueden existir otros enfoques basados en transcodings (aquellos sistemas basados en un *proxy*). Al igual que sucede con los *mash-ups* actuales, también existen precursores de las herramientas actuales de augmentación Web, precursores al menos en cuanto a las intenciones y objetivos de las herramientas. Ya en el año 1997 Barrett et al. [1997] proponían un sistema que, basado en un *proxy*, asistía al usuario en la navegación Web mediante la modificación del contenido solicitado a las aplicaciones antes de que este contenido sea enviado al cliente real. Este sistema, a pesar de no ser realmente un sistema que funcione del lado del cliente, fue uno de los primeros sistemas de adaptación *cerrados*. Cabe recordar que

en esta tesis se calificó como *cerrados* a aquellos sistemas de augmentación Web que, si bien adaptan los sitios visitados por el usuario, lo hacen con adaptaciones predeterminadas por los desarrolladores de las herramientas. Por el contrario, se calificó como *abiertos* a aquellos sistemas que proveyendo una estructura de software base permiten a los usuarios desarrollar y/o instalar nuevos artefactos de augmentación definidos en correspondencia a lo especificado por el software base provisto.

Habiendo considerado este punto, y siendo que existen infinidad de extensiones de Browser Web *cerradas* que ofrecen todo tipo de adaptaciones estáticas que no están relacionadas necesariamente con la tarea del usuario, a continuación se realiza la comparación con enfoques *abiertos*.

Usualmente, los enfoques de adaptación *abiertos* que permiten la modificación de aplicaciones de terceros se distribuyen como extensiones de los navegadores Web. Una vez mas: el caso emblemático es GreaseMonkey Pilgrim [2005], por su comunidad y también por ser uno de los primeros sistemas de este tipo. No por eso el único, ya que con el paso del tiempo y el incremento en los usuarios interesados, han surgido otras herramientas muy similares (a tal punto que mantienen compatibilidad con los *script* de GreaseMonkey) como Scriptish[1], que manteniendo el núcleo principal de GreaseMonkey agrega nuevas funciones a la API de éste.

Otras herramientas muy similares, también populares, se basan en la adaptación del contenido mediante la aplicación de nuevos estilos. Así como GreaseMonkey o Scriptish permiten *agregar* a un sitio Web código JavaScript nuevo, Stylish hace lo propio con hojas de estilo (CSS). Y así como existe un repositorio de scripts[2] también existe un repositorio de estilos[3]. En ambos casos se mantiene la misma idea: algunos usuarios son capaces de desarrollar artefactos que comparten en estos repositorios para dejarlos disponibles para otros usuarios.

Estas herramientas son ideales para satisfacer requerimientos de adaptación estáticos, como los descriptos en esta tesis como adaptaciones de preferencia. Cuando un sitio Web es cargado se ejecutarán siempre los scripts que estén asig-

[1] Scriptish, https://addons.mozilla.org/es/firefox/addon/scriptish/
[2] UserScripts, http://userscripts.org/
[3] UserStyles, http://userstyles.org/

nados a este sitio y así se modifica el DOM para que el usuario perciba las adaptaciones. Sin embargo no dan mecanismos específicos para soportar las tareas o *concerns* del usuario. Aunque permiten que los *scripts* almacenen información para uso posterior, la coordinación de varios *scripts* que sean parte de la misma tarea es sumamente compleja de conseguir, y no esta mal, dado que los usuarios de GreaseMonkey (como así también sus desarrolladores) buscan otro tipo de adaptación. Estas deficiencias no significan que no puedan desarrollarse algunas adaptaciones del estilo CSN, pero en todo caso, el peso cae sobre el desarrollador que sin ningún mecanismo de abstracción debe desarrollar toda la lógica necesaria. Además claro, de que difícilmente los scripts conseguidos puedan ser rápidamente adaptados y reutilizados.

GreaseMonkey, sin embargo, posibilita la ejecución simultánea de scripts, con lo cual nuevas capas de abstracción pueden ser *montadas* sobre estos motores de *weaving*. Un buen ejemplo es *Modding Interface*, definido por Díaz et al. [2008]. En este trabajo los autores plantean una modelo de abstracción sobre el DOM que brindado por los desarrolladores de las aplicaciones sirve de contrato para aquellos usuarios que quieran realizar adaptaciones basadas en los conceptos definidos en estos modelos. Es decir, los desarrolladores de la aplicación se comprometen a mantener la correspondencia necesaria entre los conceptos definidos en sus modelos y los DOM de sus aplicaciones. Si los usuarios desarrolladores de *scripts* se mantienen en esos límites, entonces sus artefactos de adaptación no corren riesgos de dejar de funcionar.

En este trabajo fue inspirado el modelo de abstracción del DOM presentado en esta tesis. En Firmenich et al. [2010b] se planteo que los usuarios sean los que no solo desarrollen los scripts sino que también definan los modelos utilizando las herramientas mostradas en el capítulo 6. Aunque el DOM original puede cambiar y así hacer que los *scripts* dejen de funcionar, la definición del modelo de abstracción del DOM puede redefinirse rápidamente con estas herramientas.

Los autores de *Modding Interface* definieron en trabajos posteriores lo que dieron a conocer como *Scripting Interface* Díaz et al. [2010] orientada a la robustez y reutilización de *scripts*.

Aunque estas *capas de abstracción montadas sobre* GreaseMonkey no fueron diseñadas con el objetivo de realizar adaptaciones que soporten a la tarea de

usuario de la manera que se ha mostrado en esta tesis, los resultados logrados dan un enfoque mas de ingeniería al desarrollo de *scripts* respecto a la manera anárquica con la que se desarrollan normalmente.

Haciendo foco en propuestas cuyos objetivos sean soportar las tareas de usuario, también podemos encontrar otras extensiones de navegadores Web. Por ejemplo Mozilla Ubiquity[1] (una extensión para el navegador Mozilla Firefox) intenta mejorar la experiencia del usuario mediante la augmentación no tanto de los sitios Web sino del navegador en si mismo. La idea detrás de esta extensión es permitirle al usuario agregar *operaciones* al navegador. Operaciones que el usuario puede ejecutar cuando sean pertinentes su tarea. Una operación o comando podría ser, por ejemplo, que habiendo seleccionado un texto en un sitio Web cualquiera el usuario pueda publicar el mismo en una red social solamente utilizando ese comando. Mozilla Ubiquity no soporta la automatización de este tipo de comandos, es decir que siempre son ejecutados bajo demanda del usuario, sin poder de combinación alguna.

Una herramienta similar es *Operator*[2]. Operator es otra extensión de Mozilla Firefox para ejecutar este tipo de operaciones salvo que su funcionamiento esta basado en Microformatos. Es decir, las operaciones disponibles para ser ejecutadas por el usuario son aquellas que tienen la información necesaria (estructurada con Microformatos) disponible en el sitio Web actual. Si se encuentran los Microformatos necesarios, Operator puede utilizarlos para consumir servicios Web de *Flickr*, *Google Calendar*, etc., en pos de facilitarle al usuario alguna tarea, por ejemplo, agregar un evento en *Google Calendar* basado en información estructurada con el Microformato correspondiente a una fecha. Aunque *Operator* contempla casos bien conocidos de uso en los cuales mas de una aplicación Web es involucrada en la tarea (note que la fecha con la que puede ser creado el evento en *Google Calendar* puede surgir de cualquier otro sitio Web) el hecho de que este basado totalmente en Microformatos hace algo rígido el enfoque.

Otras propuestas apuntan a soportar al usuario en tareas repetitivas que tienen lugar en el uso de aplicaciones Web. Algunas propuestas tienen su contribución principalmente en un DSL (*Domain Specific Language*), mientras que otras

[1] MozillaUbiquity, http://mozillalabs.com/ubiquity/, última visita 7/6/2011
[2] Operator, https://addons.mozilla.org/en-US/firefox/addon/operator/

además tienen un soporte total con herramientas para definir fácilmente nuevos *scripts*. Por ejemplo, CoScripter Bogart et al. [2008], Leshed et al. [2008] (otra extensión de Mozilla Firefox desarrollada por IBM) permite a los usuario *grabar* sus interacciones (páginas Web visitadas, eventos relacionados a elementos del DOM: *mouseover*, *click*, etc.) y luego repetir el proceso automáticamente. Aunque la mayoría de los escenarios grabados son rígidos, CoScripter brinda un grado de flexibilidad al contemplar ítems de información definidas en variables. Así en cada ejecución del script, si se cambian los valores asignados a dichas variables, podrán conseguirse distintos resultados; un caso simple sería el valor que se ingresa automáticamente en un campo de un formulario: en vez de realizar este paso del script siempre con el mismo valor, se llevará a cabo con el valor que tenga asignado una variable en particular. De esta manera, CoScripter mantiene un espacio de información de los *scripts*. Uno de los principales problemas con CoScripter es que si el proceso cambia ligeramente, los pasos grabados con anterioridad ya no pueden ser utilizados. Esto no es tema menor ya que las tareas de usuario no suelen ser tan rígidas como CoScripter requiere. La diferencia clave entre CoScripter y la contribución de esta tesis es la falta de augmentación Web por parte de CoScripter. CoScripter soporta procesos de negocio repetitivos, pero basados mas en el *uso* de las aplicaciones (y en un uso automatizado) más que en adaptar las mismas. Aunque la compleción automática de formularios puede llegar a tomarse como una adaptación (en definitiva no es mas que cambiar las propiedades de los objetos del DOM), esto no alcanza para decir que se está soportando al usuario mediante augmentación Web.

CoScripter no es el único que propone una "API de alto nivel" para soportar procesos automatizados en la Web. ChickenFoot Bolin et al. [2005] es otra herramienta para programar scripts por parte de usuario finales. ChickenFoot extiende JavaScript con la definición de nuevas funciones o comandos como "click()", "enter()", "find()", etc., que hacen mas fácil el desarrollo de *scripts* de ejecución automática. Aunque ChickenFoot es un lenguaje poderoso y realmente expresivo, no contempla cambios en el proceso que llevan a cabo los scripts y en cuanto a la reutilización, ChickenFoot reacciona aun peor, ya que son realmente dependientes del DOM.

Un enfoque similar a ChickenFoot es Koala, que con objetivos similares, propo-

ne un lenguaje natural para la programación de scripts. Por ejemplo, la expresión de ChickenFoot:

```
‘‘click(‘search button’)’’
```

es equivalente a la expresión

```
‘‘Click ‘search button‘’’
```

(donde ‘search button‘ corresponde al elemento del DOM del tipo *button* cuyo valor del atributo *name* es *search*). Los scripts definidos con Koala, al igual que con CoScripter, pueden ser programados por demostración. También bajo esta misma técnica de programación fue definida otra herramienta llamada Selenium[1] que aunque originalmente fue construida para realizar pruebas sobre interfaces gráficas, éste no se limita solo a esto, ya que también puede ser útil como CoScripter, Koala o ChickenFoot para ejecutar tareas repetitivas.

Aunque las herramientas descriptas son muy útiles en muchos casos, los usuarios pueden no sentirse confortables con la delegación de la totalidad de una tarea en una herramienta automática. Por ejemplo, no cualquier usuario estaría dispuesto a planificar un viaje con este tipo de herramientas ya que existen riesgos que seguramente no quiera correr, como ser la compra de un pasaje aéreo incorrecto o a un precio que no estaba dispuesto a pagar. Además, están claramente abocadas al uso de las aplicaciones, es decir que no proveen ningún tipo de augmentación en los sitios Web visitados.

Otro enfoque de augmentación Web llamado *Sticklet* definido por Díaz and Arellano [2012] (que también funciona sobre GreaseMonkey) facilita la programación de adaptaciones por parte de usuario finales. En lugar de escribir código JavaScript, utilizando *Sticklet* el usuario final puede desarrollar los scripts utilizando un editor dedicado con el que se obtiene un código de mayor abstracción que JavaScript (definido por un DSL propio) a costa de perder capacidad de expresión. Sticklet puede ser muy útil para crear integraciones del estilo *mash-up* pero sin dejar al ámbito natural de los sitios Web.

Todas las herramientas mencionadas son muy útiles y se comparte la filosofía detrás de estos enfoques: poner en mano de los usuarios el poder de mejorar su

[1] SELENIUM, http://jroller.com/selenium/

experiencia en la Web. Sin embargo, y como puede apreciarse en el desarrollo de los capítulos anteriores, en esta tesis se consideran las tareas de usuarios como una actividad realmente compleja que puede sufrir alteraciones en cualquier momento debido a lo impredecible que es el comportamiento de los usuarios. En esta tesis se ha propuesto un enfoque que contempla aristas que ninguno de los enfoques anteriores tiene en consideración: desde mecanismos para soportar requerimientos volátiles, hasta complejas composiciones en donde las unidades de composición pueden ser tanto tareas primitivas como de augmentación y considerando además distintos niveles de flexibilidad.

En las herramientas de automatización de procesos (como CoScripter) se consideran a los eventos que ocurren en el navegador Web como hechos decisivos cuando en realidad estos eventos, que pueden estar condicionados, tienen un peso semántico que no es tenido en cuenta y que sí es relevante en el contexto de lo que ese evento significa para la tarea o *concern* del usuario. En este sentido, se considera que estas herramientas son mas para un uso automatizado de la Web en lugar de estar orientadas a agregar o adaptar contenidos y/o funcionalidad de interés para la tarea del usuario.

Esta última afirmación acerca de las herramientas de automatización de procesos queda justificada desde que toda la esta secuencia automática de tareas puede ser descripta en términos de tareas primitivas, es decir, de las tareas que sabemos que los usuarios realizan en los Browser Web de manera normal.

8.2.1. Web Augmentation orientada a formularios en la Web

En esta subsección se abarcará un análisis de trabajos relacionados específicamente con el enfoque de manejo de formularios presentado en esta tesis, cuya herramienta central son *PIMI* y los *augmenters* anotación de formularios.

En primer lugar es importante aclarar que el estudio de técnicas de interacción para mejorar la experiencia del usuario en el uso de formularios Web no es un tema de investigación nuevo. Claramente, esto se debe a que la falta de estandarización en los formularios de entrada de datos tampoco lo es Girgensohn and Lee [1997].

La diversidad de estructuras y esquemas de organización que pueden encontrarse en los formularios Web es la mayor restricción que aun prohibe el desarrollo de una solución final para la automatización del completado de los formularios. La mayoría de las técnicas actuales se enfocan en la automatización de tareas que completan formularios como *Auto-Complete* Stocky et al. [2004] o bien *Auto-Filling*[1].

Algunos trabajos mas recientes (como los realizados por Araújo et al. [2010], Guo et al. [2012], Toda et al. [2010]) intentan atacar el problema mediante el uso de funciones de similaridad para predecir qué información personal es esperada para cada campo del formulario. Aun así, estas técnicas fallan en proveer al usuario con un control total de su propia información, lo cual puede tener un impacto en la confianza del usuario y su potencial adaptación como una solución final.

Otros autores (Bownik et al. [2009], Wang et al. [2009]) han investigado el uso de tecnologías de la Web Semántica. La principal contra con este tipo de técnicas es que uno debe tener una ontología que describa aplicaciones de terceros antes de poder realizar la integración de los datos. OpenId Recordon and Reed [2006] permite a los usuarios proveer identificación certificada y así compartir información con sitios Web de confianza. Luego, los registros personales pueden ser utilizados para completar formularios de estos sitios automáticamente. Uno de los inconvenientes es que requiere del acuerdo de los propietarios de tales sitios Web para poder operar.

En lugar de focalizarse en una ontología que describa una aplicación en particular, algunos esquemas de vinculación o integración de datos se basan en estándares abiertos de tipos de datos como Microformatos Khare and Çelik [2006] y Microdata[2] (un estándar en desarrollo de la W3C -*World Wide Web Consortium*-). La estructura subyacente de estos enfoques es muy similar, aunque los Microformatos tienen la ventaja de estar soportados por una comunidad ya existente y abierta, además de estar soportados por diversas herramientas.

En función de lo dicho, en esta tesis se ha tomado los Microformatos solo como prueba de concepto para la anotación de formularios Web, siendo que esta

[1] Autofill Forms - Mozilla Firefox add-on. http://autofillforms.mozdev.org/
[2] Hickson, I. HTML Microdata. http://www.w3.org/TR/microdata/

capa semántica puede variar.

Otro punto importante en este contexto es el de los PIMs (***P****ersonal* ***I****nformation* ***M****anagement* ***S****ystems*) Teevan et al. [2008]. Los estudios realizados con PIMs están, mayormente, centrados en grandes conjuntos de datos, como el contenido total del disco rígido de un usuario. Por ello, están concentrados en aspectos de búsqueda y recuperación de información. Sin embargo, nuevos trabajos de investigación han llevado el concepto de PIM sobre la Web Norrie [2008], Leone et al. [2010]. Por ejemplo Norrie [2008] propone una arquitectura completa basada en tecnologías Web 2.0 que habilita a los usuarios a administrar sus registros personales en la Web y sincronizarlos con otras aplicaciones, particularmente con redes sociales. No obstante, estos trabajos se centran principalmente en datos de texto plano y no tienen en cuenta la interacción que un usuario podría tener con los mismos en pos de completar formularios.

En Firmenich et al. [2012] se ha mostrado otro enfoque que motivado en el uso de PIMs en la Web, intenta soportar al usuario en tareas que requieren el uso intensivo de formularios.

8.3. Modelado de tareas

En la subsección 5.2.1 del capítulo 5, se ha mostrado una manera de documentar un *augmenter* considerando además del código funcional del mismo, un modelo de tareas que describa la secuencia de actividades que tienen lugar durante la ejecución del mismo. Aunque no es foco central de esta tesis, en esta subsección se justifica la elección de HAMSTERS como notación a utilizar.

El análisis de tareas es ampliamente reconocido como uno de los modos fundamentales para focalizar necesidades específicas de usuarios y proveer un entendimiento general de cómo los usuarios pueden interactuar con una interfaz gráfica en pos de completar un objetivo Diaper and Stanton [2004]. En el campo de HCI (*Human-Computer Interaction*) han sido propuestas varias notaciones para modelos de tareas en pos de crear modelos que consideren la mayor cantidad de elementos posibles de la actividad del usuario en sistemas interactivos Paternò et al. [1997].

La mayoría de las notaciones son estructuradas principalmente alrededor de dos conceptos: descomposición de tareas (frecuentemente representado como una jerarquía) y flujo de tareas (para mostrar el orden en el cual las tareas son ejecutadas) Limbourg et al. [2001]. Cuando están combinados adecuadamente, estos conceptos pueden proveer una representación completa de una gran cantidad de información en un solo modelo. Sin embargo, como fue discutido por Paternò and Zini [2004], cuando son aplicados en sistemas reales, se terminan obteniendo modelos muy grandes y difíciles de manejar, lo cual hace difícil las tareas de modelado y análisis.

Notaciones mas recientes como HAMSTERSMartinie et al. [2011] integra mecanismos de estructuración como modularidad en la representación de tareas, reutilización de elementos de subtareas, flujos de datos y características de comunicación entre las tareas.

Desde que en la definición de la documentación se propone un modelo de tarea por cada tarea específica, potencialmente podría construirse el modelo total del *escenario* mediante la composición de cada uno de esos modelos.

8.4. Conclusiones

En este repaso, área por área, vemos que hay una brecha entre la integración de aplicaciones Web, la augmentación de las mismas y la asistencia al usuario al realizar sus tareas. Los enfoque mencionados pueden ser muy *invasivos* para los usuario y es por ello que la contribución de esta tesis se busca un gran nivel de flexibilidad para que distintos tipos de soporte a la tarea del usuario sean soportados.

Manteniendo la filosofía de muchos de los trabajos mencionados, se mantiene una estructura donde los usuarios cooperan para desarrollar artefactos, lo cual ha sido puesto en valor por Arellano et al. [2010].

Al poder escoger los artefactos a utilizar, aquellos usuarios que deseen tareas totalmente automatizadas podrán tenerlas, pero también se brinda soporte o asistencia para tareas que requieren inequívocamente de la interacción del usuario. El enfoque puede ser utilizado tanto para tareas bien conocidas como también para

tareas de índole mas volátil, lo cual no es posible con muchas de las herramientas mencionadas.

Capítulo 9

Conclusiones

La programación por parte de usuarios finales está en auge, y como se ha visto la Web es el contexto de mayor protagonismo en este sentido. En esta tesis se presentó un enfoque completo para el soporte de tareas de usuario en la Web mediante adaptación en el cliente. El enfoque incluye aspectos teóricos y un conjunto de herramientas que habiendo sido evaluadas prueban la factibilidad y la mejora en la experiencia del usuarios al realizar sus tareas en la Web bajo el uso de dichas herramientas.

La contribuciones aquí presentadas han sido incluidas en distintas publicaciones científicas del área de Ingeniería Web. En el Anexo C el lector puede encontrar el listado de publicaciones donde además de todos los datos de la misma podrá encontrar un resumen de la contribución específica y su relación con esta tesis.

9.1. Contribución y cumplimiento de objetivos

La contribución específica de esta tesis es un enfoque, junto con un soporte tecnológico del mismo, para soportar las tareas de usuario mediante adaptación en el cliente Web. Tomando la navegación Web en toda su dimensión, esto es, no solo considerando navegación intra-aplicación sino también inter-aplicación, se consigue contemplar al usuario como usuario de varias aplicaciones; evitando así una de las limitaciones principales presentadas en los mecanismos clásicos de adaptabilidad.

Parte de la contribución se basa en la definición de mecanismos que soporten

al usuario de manera volátil, contemplando la utilización bajo demanda de los mismos, y que facilitan al usuario en su navegación en la Web.

Además bajo la ejecución de *escenarios* se contemplan adaptaciones automatizadas que permiten, entre otras cosas, integrar aplicaciones Web siguiendo los conceptos de *Concern-Sensitive Navigation*, logrando así una solución que media entre aplicaciones del estilo *mash-ups* (que sacan a las aplicaciones Web de su ámbito natural) y simples herramientas de augmentación Web (que ofrecen una adaptación mas estática).

El nivel de integración conseguido, orientado a la tarea actual del usuario, es sumamente relevante para ser aplicado sobre aplicaciones Web existentes. Es decir que, las soluciones conseguidas son relevantes para que los usuarios generen nuevos artefactos que satisfagan sus necesidades. Pero también puede ser una herramienta que abre la puerta a que instituciones de diversos tipos brinden artefactos (como *escenarios*) para ofrecer a sus usuarios un soporte al realizar acciones/tareas en sus sitios Web, como así también un manera de facilitar la integración de distintos sistemas pertenecientes a la misma institución, posibilitando la generación de artefactos que podrían ser calificados como *Shadow IT*. Como se ha dicho en el capítulo introductorio, las tareas que se realizan sobre aplicaciones Web no solo son aquellas realizadas en aplicaciones Web bien conocidas. Aunque los ejemplos mostrados en esta tesis lo son, por cuestiones didácticas, se ha dejado claro que las personas utilizan aplicaciones Web de todo tipo para realizar sus labores cotidianas, labores de diverso índole, y el enfoque claramente puede ser aplicado en cualquier caso.

En el capítulo introductorio se había planteado el siguiente como objetivo general:

"*diseñar e implementar una solución a partir de la cual se extienda el actual enfoque de adaptabilidad e integración de aplicaciones Web con nuevos mecanismos que permitan adaptar las mismas en pos de mejorar la experiencia de los usuarios cuando realizan tareas intra e inter-aplicación.*"

Luego, en los capítulos 2 al 8, no solo se fundamento el por qué una solución de este tipo tiene lugar en el momento actual de área de adaptabilidad en apli-

caciones Web, sino que también se definieron aspectos teóricos y técnicos de un enfoque concreto. Retomando los objetivos específicos planteados en el capítulo introductorio, a continuación se realiza un contraste que detalla cómo se han contemplado los mismos a lo largo de toda la presentación.

9.1.1. Compleción de objetivos específicos

Identificar y definir qué tipo de información referida al usuario y su actividad esta disponible en el lado del cliente considerando su uso para soportar sus tareas.
En el Capítulo 3, *Fundamentos para una arquitectura flexible de adaptación en el cliente orientada al soporte de tareas de usuario*, específicamente en la sección 3.1, se realizó un análisis de la información relevante en el contexto de la tarea del usuario resultando de tal análisis la consideración indispensable de: información referida a las aplicaciones Web utilizadas, información relativa a la interacción del usuario con las aplicaciones Web, información contenida en las aplicaciones Web (en cualquiera de sus formas: texto plano, elementos del DOM, etc.) y finalmente información contenida en un espacio de información personal.

Identificar y tipificar aquellas tareas de augmentación pueden realizarse en el cliente.
A lo largo de la tesis se ha referido como tarea de augmentación, de forma genérica, a toda aquella tarea que no siendo contemplada nativamente por el Browser Web es agregada utilizando el enfoque. Sin embargo, y analizando pasajes de los capítulos 3 y 4, vemos que se han contemplado en el enfoque tareas de augmentación de contenido Web (*augmenters*) y tareas de augmentación relacionadas con colectar información (*DataCollectors* y uso del *Pocket*).

Analizar y diseñar mecanismos que permitan a los usuarios realizar adaptaciones bajo demanda en pos de satisfacer requerimientos volátiles referidos a la tarea actual de los mismos.
Mientras que en el capítulo 3 se analiza la importancia de la ejecución manual de adaptaciones, en el capítulo 4 se introducen herramientas concretas que le per-

miten al usuario ejecutar adaptaciones bajo demanda utilizando la información disponible en el *Pocket*. En este sentido se diseñó un objeto específico, *AdaptationDispatcher*, que puede ser utilizado manualmente a través del *Pocket*.

Analizar y diseñar mecanismos que den soporte al usuario cuando realiza tareas reiteradamente, siempre siguiendo un mismo escenario de uso.
Dar soporte a tareas de usuario conocidas ha sido, dentro de los objetivos específicos definidos, en el que mayor foco se ha hecho en el transcurso de la tesis. En el capítulo 3 se analizan las necesidades de un modo automático de ejecución de adaptaciones, lo cual está fuertemente ligado a soportar tareas que se realizan reiteradamente, ya que al ser conocidas, permiten la automatización. En el capítulo 4 se introducen las herramientas concretas para esta problemática: *ScenarioEditor* y *ScenarioPlayer*. En el capítulo 5 se detalla en profundidad el enfoque de *escenario* y se muestra en su totalidad las herramientas relacionadas.

Analizar y diseñar los artefactos de software que son necesarios para llevar a cabo estas adaptaciones.
El capítulo 4 se presenta el *framework* de adaptación del lado del cliente, que fue diseñado para involucrar y orquestar todos los componentes definidos en respuesta a los fundamentos planteados en el capítulo 3 acerca de una plataforma flexible que soporte las tareas de usuario mediante adaptación en el cliente Web. El *framework* se distribuye como una extensión del navegador Web Mozilla Firefox.

Mantener la filosofía subyacente en propuestas como Web augmentation y mash-ups respecto al desarrollo por parte de usuarios finales.
En la sección 2.4.2.1 del capítulo 2 se ha puesto en valor lo que se conoce como *end-user programming*. Haciendo foco en esto, en el diseño del *framework* incluido en el capítulo 4 se dejan ver diferentes puntos de extensión que pueden ser utilizados por usuario finales para crear nuevos artefactos en el marco del mismo. Además, se han contemplado diversos niveles de desarrollo, consiguiendo así una

estructura donde pueden desarrollarse nuevos componentes de muy baja abstracción, hasta nuevos *escenarios* que además de ser especificados utilizando el DSL presentado en el capítulo 5 pueden ser creados utilizando herramientas visuales.

Permitir a los usuarios a compartir los artefactos desarrollados. Todos los artefactos definidos pueden ser compartidos entre los usuarios. Esto no significa solo que existe un repositorio común sino que además se proveen desde el framework mecanismos sumamente sencillos para la administración (instalación y desinstalación) de los artefactos generados por usuarios. No es casual además, que artefactos como *escenarios* sean definidos a partir de otros artefactos como *augmenters*, lo cual implica que sin haber desarrollado ningún *augmenter*, y solo reutilizando aquellos implementados por otros usuarios, un nuevo *escenario* puede ser definido. Esto deja en evidencia el espíritu de colaboración y la delegación en la masa de usuarios que existe tras el enfoque.
Respecto a los *escenarios* no solo se posibilita a los usuarios compartir la definición base de los mismos, sino que además permite compartir las ejecuciones propias de un *escenario* con usuarios específicos que puedan sacar provecho de la información utilizada en ejecuciones previas.

Proveer una estructura de software robusta y lo suficientemente flexible para orquestar los artefactos desarrollados por usuarios, es decir tareas de augmentación, con tareas primitivas en pos de soportar las tareas de usuario.
Mediante el enfoque de *escenario*, y las herramientas correspondientes (presentadas en el capítulo 5) se logra la orquestación y combinación de las tareas de augmentación con las tareas primitivas (tareas primitivas que fueron definidas a partir de trabajos de investigación de terceros). Las herramientas y componentes del *framework* abocados a esta tarea permiten la edición y ejecución de *escenarios* como se ha mostrado en el capítulo 5 y en casos de estudio del capítulo 6.

Evaluar los resultados conseguidos para poder medir el impacto de los nuevos mecanismos de adaptación diseñados y orientando dichas evaluaciones a relevar en que medida es mejorada la experiencia de

usuario al realizar sus tareas.
El capítulo 7 esta absolutamente dedicado a la evaluación de las herramientas desarrolladas. Se han llevado a cabo mas de una evaluación utilizando diversas técnicas. Las evaluaciones reportaron una mejora significativa en la experiencia de los usuarios al realizar tareas en general, y también proveyeron una devolución importante referida, por ejemplo, a aspectos de usabilidad, que han disparado y motivado trabajos futuros.

9.2. Trabajos futuros

De la propuesta de adaptación presentada en esta tesis se desprenden trabajos futuros que se enumeran a continuación:

- Las evaluaciones realizadas y presentadas en el capítulo 7 han revelado pequeños problemas de usabilidad de algunas herramientas, los cuales deben ser corregidos en el futuro.

- Cuando se adaptan aplicaciones de terceros se corre un riesgo fundamental: no se controlan los documentos pertenecientes a las aplicaciones, con lo cual los artefactos de adaptación dependientes de la versión de tales documentos pueden dejar de funcionar ante los cambios que los documentos sufran. Aunque en esta tesis se han desarrollado mecanismos de adaptación independientes de cualquier documento, otras, las adaptaciones mas complejas no escapan a este riesgo. En este sentido, nuevas maneras de especificar los artefactos de adaptación deben ser investigados para incrementar la robustez de los mismos.

- Un punto relacionado con el anterior, respecto a la manipulación de los documentos, debe ser tenido en cuenta. La ejecución en paralelo de herramientas de adaptación que funcionen en el cliente Web pueden interferir entre sí. Siendo que aunque no podemos controlar las versiones originales, si podemos controlar las herramientas que se ejecutan. Así es que deben generarse a futuro mecanismos para sincronizar estas herramientas.

- La manera actual de ejecución de *escenarios* es asistida por herramientas, específicamente por *ScenarioPlayer*. Esto implica que se requiere que el usuario dispare la ejecución de un *escenario* para dar inicia a una determinada tarea. Aunque esto puede ser valioso para tareas especificas, como planificar un viaje, puede ser molesto (desde el punto de vista del usuario) tener que estar ejecutando un *escenario* mediante el *ScenarioPlayer* para lograr adaptaciones como la que se muestra en el ejemplo de la Figura 6.14. Actualmente se esta trabajando en la ejecución de *escenarios* en *background*, es decir que éstos están constantemente en estado activo y no solo cuando el usuario explícitamente utiliza el *ScenarioPlayer*. Dicha ejecución *background* requiere mayor estudio ya que puede caerse en problemas de rendimiento por estar ejecutando varios scripts al mismo tiempo. Note que a diferencia de motores estáticos como GreaseMonkey, donde el esfuerzo de aplicar la adaptación se realiza sólo al cargar un sitio Web, en el enfoque presentado en esta tesis las condiciones pueden ser varias otras.

- La definición de un *escenario*, es decir, de la composición y secuencia de tareas primitivas y de augmentación es sumamente simple utilizando el *ScenarioEditor*. Solo requiere seleccionar que tipo de tarea desea agregarse a la composición y luego que tarea en particular. Sin embargo, el orden y la especificidad de cada tarea requiere, para ser bien definido, que el usuario paralelamente a la composición vaya realizando la tarea manualmente; a no ser claro, que el usuario tenga la capacidad de conocer cada interacción que tiene lugar en las aplicaciones Web involucradas en una tarea determinada. Siendo así, se prevé nueva funcionalidad para la herramienta *ScenarioEditor* que, inspirada en la programación por ejemplo/demostración, permita al usuario registrar cada tarea realizada (tarea primitiva y de augmentación) para que sirva de base para la definición final del *escenario*.

- En esta presentación se ha vislumbrado a partir de la especificación de modelos de tareas para las tareas primitivas y las tareas de augmentación, cómo podrían utilizarse estos modelos para realizar análisis sobre las tareas de usuarios. En la tesis, se propone la notación HAMSTERS para tal objetivo. La generación automática (a partir del código correspondiente de

un *escenario*) de estos modelos no ha sido tratada en esta tesis y queda así planteada como trabajo futuro.

- La seguridad que concierne al enfoque en su totalidad requiere ser estudiada en profundidad. Introducir nuevos artefactos en el Browser Web es sumamente riesgoso ya que siendo parte de un sistema desktop como lo es un Browser Web, podría accederse a recursos que el usuario no desea exponer. La especificación de un DSL basado en XML, sumado a una estricta manera de desarrollar, instalar y ejecutar el resto de los artefactos son, en primer instancia, las precauciones de seguridad que se han tomado. Sin embargo mayores estudios son necesarios.

- Aunque hay una demostración empírica, basada en el conocimiento que tenemos sobre diversas comunidades, de que la masa de usuarios puede desarrollar artefactos por su cuenta, es necesario evaluar el nivel de facilidad con que los artefactos específicos a esta tesis son desarrollados; principalmente considerando *augmenters* y *escenarios*.

- Esta tesis, como muchas otras herramientas de adaptación del lado del cliente, se basa en la comunidad de usuarios. En términos potenciales, la diversidad de artefactos de distintas calidades que pueden ser generadas y además los requerimientos e ideas de usuarios que no tienen conocimientos para generarlos, hace esencial que se requiera (en un futuro) diseñar, especificar e implementar un método de relevamiento, desarrollo y prueba de artefactos de adaptación bajo la característica de crowdsourcing.

Acrónimos

- **API**: Application Programming Interface
- **DSL**: Domain-Specific Language
- **W3C**: World Wide Web Consortium
- **OOHDM**: Object Oriented Hypermedia Design Method
- **XML**: Extensible Markup Language
- **DOM**: Document Object Model
- **HTML**: HyperText Markup Language
- **HTTP**: Hypertext Transfer Protocol
- **PIM**: Personal Information Manager
- **CSA**: Client-Side Adaptation
- **CSN**: Concern-Sensitive Naveigation
- **URL**: Uniform Resource Locator
- **RIA**: Rich Internet Application
- **UML**: Unified Modeling Language
- **IT**: Information Technology
- **DUI**: Distributed User Interface

Glosario

CONCERN

Un concern según especifica un determinado tema o área de dominio de una aplicación, se define como algo que es materia de consideración de un sistema de software. En esta tesis, se refiere al término *concern* aplicado a la navegación. Un *concern navegacional* que afecta a la navegación, es decir, que repercute en la estructura navegacional de la aplicación.

CLIENTE

En una arquitectura de software cliente-servidor, el cliente es la aplicación que consume servicios/contenidos remotos, es decir del servidor.

BROWSER WEB

En el campo específico de las aplicaciones Web, donde existe una arquitectura cliente-servidor, el cliente es la aplicación utilizada por los usuarios para acceder a los recursos disponibles en el servidor y se llama Browser Web o Navegador Web.

LINK

Un link permite conectar dos nodos, posibilitando así la navegación desde uno de los nodos hacia el otro y no viceversa.

NODO

Un nodo de una aplicación Web representa contenido, el cual puede ser textual, audio, imágenes, etc., y que puede ser accedido desde un cliente.

NAVEGACION

Cuando dos nodos, A y B, están conectados por un link (de A hacia B) y el usuario selecciona ese link en particular se dice que *navega* hacia el nodo B. Esto es lo que se conoce como navegación.

PLUG-IN DE BROWSER WEB

Un plug-in, (también conocido como extensión o complemento) es una aplicación adicional que se relaciona y funciona con otra aplicación, generalmente de mayor importancia, para agregar nueva funcionalidad que suele ser muy específica.

EVENTO

Un evento tiene lugar ante a una acción realizada por el usuario y a partir de la cual puede realizarse algún proceso en respuesta a ese evento. En JavaScript, la interacción del usuario con las aplicaciones Web y con el mismo Browser Web puede ser observada gracias a los eventos.

SHADOW IT

En el contexto de IT, Shadow IT es un término referido a soluciones IT o sistemas IT que son desarrollados y utilizados dentro de una institución para llevar a cabo alguna labor, pero sin estar oficializada o aprobada por la institución en cuestión.

UML - Diagrama de Secuencia

Es un diagrama, definido por UML, para modelar las interacciones que ocurren entre los objetos de un sistema.

Referencias

Samur Araújo, Qi Gao, Erwin Leonardi, and Geert-Jan Houben. Carbon: Domain-independent automatic web form filling. In *ICWE*, pages 292–306, 2010. 55, 140

Cristóbal Arellano, Oscar Díaz, and Jon Iturrioz. Script programmers as value co-creators. In *ICWE Workshops*, volume 6385 of *Lecture Notes in Computer Science*, pages 417–420. Springer, 2010. 142

Chieko Asakawa and Hironobu Takagi. Transcoding. In Simon Harper and Yeliz Yesilada, editors, *Web Accessibility*, Human-Computer Interaction Series, pages 231–260. Springer, 2008. ISBN 978-1-84800-049-0. 46

Anne Aula, Natalie Jhaveri, and Mika Käki. Information search and re-access strategies of experienced web users. In *WWW*, pages 583–592, 2005. 54

Rob Barrett, Paul P. Maglio, and Daniel C. Kellem. How to personalize the web. In Steven Pemberton, editor, *CHI*, pages 75–82. ACM/Addison-Wesley, 1997. ISBN 0-201-32229-3. 133

Harini Bharadvaj, Anupam Joshi, and Sansanee Auephanwiriyakul. An active transcoding proxy to support mobile web access. In *SRDS*, pages 118–123, 1998. 12, 35

Jeffrey P. Bigham and Richard E. Ladner. Accessmonkey: a collaborative scripting framework for web users and developers. In *W4A*, pages 25–34, 2007. 46

Christopher Bogart, Margaret M. Burnett, Allen Cypher, and Christopher Scaffidi. End-user programming in the wild: A field study of coscripter scripts. In *VL/HCC*, pages 39–46, 2008. 137

Michael Bolin, Matthew Webber, Philip Rha, Tom Wilson, and Robert C. Miller. Automation and customization of rendered web pages. In Patrick Baudisch, Mary Czerwinski, and Dan R. Olsen, editors, *UIST*, pages 163–172. ACM, 2005. ISBN 1-59593-271-2. 137

Jorge Luis Borges. *Ficciones*. 1944. 14, 26

Niels Olof Bouvin. Unifying strategies for web augmentation. In *Hypertext*, pages 91–100. ACM, 1999. 4, 131

Lukasz Bownik, Wojciech Gorka, and Adam Piasecki. Assisted form filling. *Engineering the Computer Science and IT*, 2009. 140

J. Brooke. Sus: a "quick and dirty" usability scale. 1996. 123

Peter Brusilovsky. Adaptive hypermedia. *User Model. User-Adapt. Interact.*, 11 (1-2):87–110, 2001. 15

Peter Brusilovsky. Adaptive navigation support. pages 263–290, 2007. 21, 54

Peter Brusilovsky and Eva Millán. User models for adaptive hypermedia and adaptive educational systems. In Brusilovsky et al. [2007b], pages 3–53. ISBN 978-3-540-72078-2. 18, 19

Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, 2007a. Springer. ISBN 978-3-540-72078-2. 3

Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web, Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, 2007b. Springer. ISBN 978-3-540-72078-2. 20, 157, 161

Andrea Bunt, Giuseppe Carenini, and Cristina Conati. Adaptive content presentation for the web. In Brusilovsky et al. [2007b], pages 409–432. ISBN 978-3-540-72078-2. 22

Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945. 13

M.D. Byrne, B. John, N. Wehrle, and D. Crow. he tangled web we wove: a taskonomy of www use. In *SIGCHI*, pages 544–551. ACM, 1999. 6, 82, 83, 88

S. Card, T. Moran, and A. Newell. *The psychology of human-computer interaction*. Lawrence Erlbaum Associates, 1983. 127

Florian Daniel, Fabio Casati, Stefano Soi, Jonny Fox, David Zancarli, and Ming-Chien Shan. Hosted universal integration on the web: The mashart platform. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *ICSOC/ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 647–648, 2009. ISBN 978-3-642-10382-7. 132

D. Diaper and N. A. Stanton. *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates,, 2004. 141

Oscar Díaz and Cristóbal Arellano. Sticklet: An end-user client-side augmentation-based mashup tool. In *ICWE*, volume 7387 of *Lecture Notes in Computer Science*, pages 465–468. Springer, 2012. 138

Patrick Dubroy and Ravin Balakrishnan. A study of tabbed browsing among mozilla firefox users. In *CHI*, pages 673–682, 2010. 54

Oscar Díaz, Cristóbal Arellano, and Jon Iturrioz. Layman tuning of websites: facing change resilience. In *WWW*, pages 1127–1128, 2008. 56, 105, 106, 135

Oscar Díaz, Cristóbal Arellano, and Jon Iturrioz. Interfaces for scripting: Making greasemonkey scripts resilient to website upgrades. In *ICWE*, volume 6189 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2010. 135

Deborah M. Edwards and Lynda Hardman. 'lost in hyperspace': Cognitive mapping and navigation in a hypertext environment. In *UK Hypertext*, pages 105–125, 1988. 14

Sergio Firmenich, Silvia E. Gordillo, Gustavo Rossi, and Marco Winckler. Client-side adaptation: An approach based in reutilization using transversal models.

In *ICWE Workshops*, volume 6385 of *Lecture Notes in Computer Science*, pages 566–570. Springer, 2010a. 105

Sergio Firmenich, Gustavo Rossi, Matias Urbieta, Silvia E. Gordillo, Cecilia Challiol, Jocelyne Nanard, Marc Nanard, and João Araújo. Engineering concern-sensitive navigation structures, concepts, tools and examples. *J. Web Eng.*, 9 (2):157–185, 2010b. 25, 27, 33, 56, 59, 135

Sergio Firmenich, Marco Winckler, Gustavo Rossi, and Silvia E. Gordillo. A framework for concern-sensitive, client-side adaptation. In *ICWE*, volume 6757 of *Lecture Notes in Computer Science*, pages 198–213. Springer, 2011. 104, 198, 199

Sergio Firmenich, Vincent Gaits, Silvia E. Gordillo, Gustavo Rossi, and Marco Winckler. Supporting users tasks with personal information management and web forms augmentation. In *ICWE*, volume 7387 of *Lecture Notes in Computer Science*, pages 268–282. Springer, 2012. 44, 55, 109, 141

Martin Fowler and Rebecca Parsons. *Domain-Specific Languages*. 2010. 8, 77

Vincent Gaits. *Conception et developpement d'une technique d'interaction avec des formulaires Web*. 2011. 109

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Elements of Reusable Object-Oriented Software*. 2011. 69, 76

Alejandra Garrido, Sergio Firmenich, Gustavo Rossi, Julian Grigera, Nuria Medina Medina, and Ivana Harari. Personalized web accessibility using client-side refactoring. *IEEE Internet Computing*. 46

Irene Garrigós. *A-OOH: Extending Web Application Design with Dynamic Personalization*. 2008. 16

Andreas Girgensohn and Alison Lee. Seamless integration of interactive forms into the web. *Computer Networks*, 29(8-13):1531–1542, 1997. 139

Xiaonan Guo, Jochen Kranzdorf, Tim Furche, Giovanni Grasso, Giorgio Orsi, and Christian Schallhart. Opal: a passe-partout for web forms. In Alain Mille,

Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *WWW (Companion Volume)*, pages 353–356. ACM, 2012. ISBN 978-1-4503-1230-1. 140

Hao Han and Takehiro Tokuda. A method for integration of web applications based on information extraction. In Daniel Schwabe, Francisco Curbera, and Paul Dantzig, editors, *ICWE*, pages 189–195. IEEE, 2008. ISBN 978-0-7695-3261-5. 132

Tom Heath. A taskonomy for the semantic web. In *Semant. web 1*, pages 75–81, 2010. 6, 82

M. Cameron Jones and Elizabeth F. Churchill. Conversations in developer communities: a preliminary analysis of the yahoo! pipes community. In John M. Carroll, editor, *C&T*, pages 195–204. ACM, 2009. ISBN 978-1-60558-713-4. 132

Stanley M. Sutton Jr. and Isabelle Rouvellou. Modeling of software concerns in cosmos. In *AOSD*, pages 127–133, 2002. 26

Rohit Khare and Tantek Çelik. Microformats: a pragmatic path to the semantic web. In *WWW*, pages 865–866, 2006. 59, 70, 110, 140

Bent Bruun Kristensen and Kasper Østerbye. Roles: Conceptual abstraction theory and practical language issues. *TAPOS*, 2(3):143–160, 1996. 28

Stefania Leone, Michael Grossniklaus, Alexandre de Spindler, and Moira C. Norrie. Synchronising personal data with web 2.0 data sources. In Lei Chen, Peter Triantafillou, and Torsten Suel, editors, *WISE*, volume 6488 of *Lecture Notes in Computer Science*, pages 411–418. Springer, 2010. ISBN 978-3-642-17615-9. 141

Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa A. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In Mary Czerwinski, Arnold M. Lund, and Desney S. Tan, editors, *CHI*, pages 1719–1728. ACM, 2008. ISBN 978-1-60558-011-1. 137

C. Lewis. Using the "thinking aloud" method in cognitive interface design. 1982. 122

Quentin Limbourg, Costin Pribeanu, and Jean Vanderdonckt. Towards uniformed task models in a model-based approach. In Chris Johnson, editor, *DSV-IS*, volume 2220 of *Lecture Notes in Computer Science*, pages 164–182. Springer, 2001. ISBN 3-540-42807-0. 142

Célia Martinie, Philippe A. Palanque, and Marco Winckler. Structuring and composition mechanisms to address scalability issues in task models. In Pedro Campos, T. C. Nicholas Graham, Joaquim A. Jorge, Nuno Jardim Nunes, Philippe A. Palanque, and Marco Winckler, editors, *INTERACT (3)*, volume 6948 of *Lecture Notes in Computer Science*, pages 589–609. Springer, 2011. ISBN 978-3-642-23764-5. 84, 87, 96, 142

Matthias Meusel, Krzysztof Czarnecki, and Wolfgang Köpf. A model for structuring user documentation of object-oriented frameworks using patterns and hypertext. In *ECOOP*, pages 496–510, 1997. 66

Alessandro Micarelli, Fabio Gasparetti, Filippo Sciarrone, and Susan Gauch. Personalized search on the world wide web. In Brusilovsky et al. [2007b], pages 195–230. ISBN 978-3-540-72078-2. 18, 22

Jocelyne Nanard, Gustavo Rossi, Marc Nanard, Silvia E. Gordillo, and Leandro Perez. Concern-sensitive navigation: Improving navigation in web software through separation of concerns. In *CAiSE*, pages 420–434, 2008. 26, 27, 33, 196

Michael Nebeling, Stefania Leone, and Moira C. Norrie. Crowdsourced web engineering and design. In *ICWE*, pages 31–45, 2012. 40

Moira C. Norrie. Pim meets web 2.0. In Qing Li, Stefano Spaccapietra, Eric S. K. Yu, and Antoni Olivé, editors, *ER*, volume 5231 of *Lecture Notes in Computer Science*, pages 15–25. Springer, 2008. ISBN 978-3-540-87876-6. 141

Reinhard Oppermann. Adaptively supported adaptability. *Int. J. Hum.-Comput. Stud.*, 40(3):455–472, 1994. 11, 16

Fabio Paternò and Enrico Zini. Applying information visualization techniques to visual representations of task models. In Pavel Slavík and Philippe A. Palanque, editors, *TAMODIA*, pages 105–111. ACM, 2004. 142

Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In Steve Howard, Judy Hammond, and Gitte Lindgaard, editors, *INTERACT*, volume 96 of *IFIP Conference Proceedings*, pages 362–369. Chapman & Hall, 1997. ISBN 0-412-80950-8. 141

Fabio Paternò, Alexander Repenning, and Alistair G. Sutcliffe. End-user development. In *INTERACT*, 2003. 5

Mark Pilgrim. *Greasemonkey hacks - tips and tools for remixing the web with Firefox*. O'Reilly, 2005. 134

David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In Ari Juels, Marianne Winslett, and Atsuhiro Goto, editors, *Digital Identity Management*, pages 11–16. ACM, 2006. ISBN 1-59593-547-9. 140

Dirk Riehle and Thomas R. Gross. Role model based framework design and integration. In *OOPSLA*, pages 117–133, 1998. 28

Gustavo Rossi and Daniel Schwabe. Modeling and implementing web applications with oohdm. In *Web Engineering*, pages 109–155. 2008. 32, 34

J. Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Theresa Hudson (Ed.), 1994. 121, 123

Tom Stocky, Alexander Faaborg, and Henry Lieberman. A commonsense approach to predictive text entry. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *CHI Extended Abstracts*, pages 1163–1166. ACM, 2004. ISBN 1-58113-703-6. 140

Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: Automating web browsing tasks by demonstration. In *ACM Symposium on User Interface Software and Technology*, pages 9–18, 1998a. 132

Atsushi Sugiura and Yoshiyuki Koseki. Internet scrapbook: Automating web browsing tasks by programming-by-demonstration. *Computer Networks*, 30 (1-7):688–690, 1998b. 132

Jaime Teevan, William Jones, and Robert Capra. Personal information management (pim) 2008. *SIGIR Forum*, 42(2):96–103, 2008. 55, 141

Guilherme A. Toda, Eli Cortez, Altigran Soares da Silva, and Edleno Silva de Moura. A probabilistic approach for automatically filling form-based web interfaces. *PVLDB*, 4(3):151–160, 2010. 140

Y. Wang, T. Peng, W. Zuo, and R Li. Automatic filling forms of deep web entries based on ontology. In *Proceedings of the 2009 International Conference on Web Information Systems and Mining*, pages 376–380. IEEE Computer Society, 2009. 140

Noah Wardrip-Fruin and Nick Montfort. *The New Media Reader*. 2003. 14

Harald Weinreich, Hartmut Obendorf, Eelco Herder, and Matthias Mayer. Off the beaten tracks: exploring three aspects of web navigation. In *WWW*, pages 133–142, 2006. 54

Marco Winckler, Vincent Gaits, Dong-Bach Vo, Sergio Firmenich, and Gustavo Rossi. An approach and tool support for assisting users to fill-in web forms with personal information. In *SIGDOC2011*. 199

Jeffrey Wong and Jason I. Hong. Making mashups with marmite: towards end-user programming for the web. In Mary Beth Rosson and David J. Gilmore, editors, *CHI*, pages 1435–1444. ACM, 2007. ISBN 978-1-59593-593-9. 132

Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *IEEE Internet Computing*, 12(5):44–52, 2008. 4, 131

Anexo A: Desarrollo de Augmenters

En este anexo se explica paso a paso la construcción de un *augmenter*. En este caso se mostrará un *augmenter* que puede ser bajo demanda del usuario y también de manera automática mediante *escenarios*.

.1. Definición de un *augmenter*

Para crear un *augmenter* es necesario escribir un archivo JavaScript en el cual una nueva clase debe ser definida. Esta nueva clase debe heredar de la clase *AbstractAugmenter*, el punto de extensión del *framework* para *augmenters*.

En el listado 1 se muestran estos primeros pasos. Aquí, en primer lugar se muestra la variable *metadata*, en la cual se definen datos sobre el *augmenter* (nombre, autor, descripción, atributos, etc.). Luego, puede apreciarse la función *getAugmenterInstance*. Esta función debe estar en todo código de un *augmenter* dado que el *framework* al cargarlo ejecutara esta función para obtener la instancia del *augmenter*. Note que a priori el framework no conoce que clase es la que el desarrollor definió, en este caso se ha definido la clase *HighlightingAugmenter*, pero pudo haber sido cualquier otro nombre; es por esto que se requiere de la función *getAugmenterInstance*.

En la definición de la clase, es importante inicializar la variable *label*, ya que el valor asignado se utilizará en el menú contextual del *Pocket* cuando el usuario quiera ejecutar el *augmenter* manualmente.

Listing 1: Ejemplo completo de un *augmenter*

```
var metadata = {"name":"My Augmenter Name", "author":"Author Name", "description": "This
    augmenter ..........", "attributes":[]};


function getAugmenterInstance(){
   //return the augmenter instance
   return new HighlightingAugmenter();
};
function HighlightingAugmenter(){
   this.label = "Highlight";
};
HighlightingAugmenter.prototype = new AbstractAugmenter();
```

Finalmente, en la última linea de este primer extracto de código, se muestra como heredar de la clase *AbstractAugmenter*.

.1.1. Métodos a implementar en el *augmenter*

Una vez la clase del *augmenter* ha sido definida, deben proveerse implementaciones concretas para varios métodos (métodos que han sido o bien definidos de manera abstracta *AbstractAugmenter* o bien con un comportamiento por defecto):

- #isApplicableToConcept: como la información colectada por usuarios puede ser conceptualizada bajo un *tag* semántico (por ejemplo, pudo utilizarse el concepto "País" para el valor "Argentina", pero también pueden existir otras instancias para este concepto: "Francia", "Uruguay", etc.), el *Pocket* puede contener dos tipos de elementos: *PocketConcept* y *PocketInstance*. Como un *augmenter* puede requerir uno y solo un valor para ejecutarse, y siendo que un concepto puede tener mas de una instancia, es necesario que el desarrollador especifique si el *augmenter* puede ejecutarse con muchas instancias o si es necesario hacerlo sólo con una. En correspondencia, de este método debe retornarse *true* o *false*. Por ejemplo, el *augmenter CopyIntoInput* no soporta la ejecución con una colección de instancias, siendo que requiere solo un valor para copiar en un campo.

- #adaptTextPlainInstance: Las instancias de *PocketInstance* pueden corresponder a texto plano o bien a elementos del DOM. Claro está, que para llevar a cabo una adaptación (por ejemplo la de este *augmenter*: resaltar información para ser rápidamente encontrada) se deben tomar distintas medidas en concordancia al tipo de elemento que es. No es lo mismo resaltar texto plano, que un elemento *anchor* del DOM. El método #adaptTextPlainInstance debe contener el comportamiento necesario para adaptar texto plano.

- #adaptDOMElementInstance: este método es el equivalente al anterior pero para elementos del DOM.

- #isExecutableUnderDemand: en casos de querer desarrollar *augmenters* que sean solo ejecutables desde un *escenario* y no bajo demandas del usuario desde el *Pocket*, este método debe retornar *false*. De esta manera, cuando el *framework* construye el menú contextual del *Pocket* no tiene en cuenta a este *augmenter*.

Todos estos métodos son ejecutados oportunamente por el *framework*. Tanto #adaptTextPlainInstance como #adaptDOMElementInstance reciben instancias como argumentos. Para obtener el valor concreto de la instancia, estar proveen el mensaje #getValue(). Además los *augmenters* poseen el DOM objetivo, es decir, el DOM que se desea adaptar en la variable de instancia *doc*, definida en *AbstractAugmenter*. Esto es gracias a que para de ejecutar los métodos #adaptTextPlainInstance y #adaptDOMElementInstance (según corresponda) el *framework*, el framework envía al *augmenter* el mensaje #execute (definido en *AbstractAugmenter*) con dicho DOM objetivo como parámetro.

El código correspondiente a estos métodos es mostrado en el listado 2.

Listing 2: Ejemplo completo de un *augmenter*

```
HighlightingAugmenter.prototype.isApplicableToConcept = function(){
   return true;
};
```

```
HighlightingAugmenter.prototype.isExecutableUnderDemand = function(instance){
  return true;
};

HighlightingAugmenter.prototype.adaptDOMElementInstance = function(instance){
   var dom_element = instance.getValue();
   dom_element.setAttribute(``style'',``background-color:yellow;'');
};

HighlightingAugmenter.prototype.adaptTextPlainInstance = function(instance){
   this.highlightText(this.doc, instance.getValue(),true,false,false);
};
```

En este código todo lo necesario para un *augmenter* de ejecución manual es mostrado: cómo se obtiene el DOM objetivo, y cómo usar los valores de las instancias. El método *highlightText* que puede verse en el código, es solo un método que recorre el DOM buscando ocurrencias del texto. Por razones de espacio, y siendo que no es relevante para el total entendimiento de cómo desarrollar un nuevo *augmenter*, no es incluido.

.1.2. Información adicional disponible en los *augmenters*

Cuando un *augmenter* depende de la actividad del usuario, el *augmenter* puede agregar *listeners*. Este es el caso del *augmenter CopyIntoInput* que requiere que el usuario realice un *click* sobre un campo particular de un formulario. En el listado 3 se puede ver el código de este *augmenter*.

Listing 3: Ejemplo completo de un *augmenter*

```
function getAugmenterInstance(){
    return new SimpleCopyAdapter();
};

var data_for_simple_copy = null;
```

```
function SimpleCopyAdapter(){
  this.name = ''SimpleCopyAdapter'';
    this.label = ''Copy into input'';
};

SimpleCopyAdapter.prototype = new AbstractAdapter();

SimpleCopyAdapter.prototype.isApplicableToConcept = function(){
       return false;
};

SimpleCopyAdapter.prototype.isExecutableUnderDemand = function(instance){
  return true;
};

SimpleCopyAdapter.prototype.copyToInput = function(event){
  if(event.target.nodeName.toUpperCase() == ''INPUT''){
     event.target.focus();
       event.target.value = instance.getValue();
   }
};

SimpleCopyAdapter.prototype.adaptTextPlainInstance = function(instance){
   data_for_simple_copy = instance.getValue();
   this.expressInterestIn(''mousedown'', this.doc, this.copyToInput, true);
};
```

Note que en la última linea del método #adaptTextPlainInstance, el *augmenter* se envía así mismo el mensaje #expressInterestIn, en pos de expresar interés en un evento específico requerido para realizar la adaptación y en donde quiere que ocurra ese evento (*this.doc*). Además, para enviar este mensaje otros parámetros son necesarios, como ser una método que se ejecute cuando el evento ocurra. Finalmente, un cuarto parámetro, un *booleano*, permite determinar si el evento debe ser ejecutado mas de una vez.

.2. Instalación de augmenters *Augmenters*

Para instalar un nuevo *augmenter* el usuario debe abrir el archivo JavaScript correspondiente con Mozilla Firefox (teniendo instalado el *framework*). Esta acción abrirá una notificación donde se ofrece la instalación del mismo. Una vez instalado, y si corresponde, este será mostrado en el menú contextual del *Pocket*.

Si el desarrollador quiere editar el *augmenter* mientras este esta instalado, puede hacerlo mediante el menú principal del *framework* (*Client-Side Adaptation*), mediante la herramienta *UserArtefactManager*. Los cambios que se realicen serán tomados en cuenta luego de reiniciar el navegador. De otro modo, el desarrollador puede desinstalarlo y volver a instalar, reiniciando Firefox cada vez.

Anexo B: Escenario completo para la prueba de herramienta PIMI

En este anexo se presentan los escenarios completos que han sido especificados para la evaluación sobre *PIMI* presentada en el capítulo 7.

Se presentan, en dos subsecciones, la realización de la misma tarea con y sin utilización de *PIMI*.

.3. Escenario realizado sin PIMI

Cuadro 1: Lista de tareas de usuario y sus respectivas tareas específicas

Context Task	Action	Description	Time	K
Step 1: Buy Flights				
Step 1.1: Search Flights				
	H[Mouse]	Reach for mouse	0,4	
	P[BrowserAddressBar]	Move pointer to Browser Address Bar	1,1	

	K[mouse]	Select Browser Address bar by clicking over it.	0,2	
	H[Keyboard]		0,4	
	M15K[word]	Enter the text “www.expedia.com”	4,2	15
Once page is loaded:				
	H[Mouse]		0,4	
	P[RadioButton]	Move the point to the target product Flights	1,1	
	K[mouse]	Select the radio button “Flight” by clicking over it	0,2	
	P[LeavingFrom]	Move pointer to labeled “Leaving from” field	1,1	
	K[mouse]	Select input labeled “Leaving from” by clicking over it	0,2	
	H[Keyboard]		0,4	
	M12K[word]	Enter a text, for example “Buenos Aires”	3,6	12
	H[Mouse]		0,4	
	P[Departing]	Move pointer to labeled Departing field	1,1	
	K[mouse]	Click Departing field (this opens the calendar)	0,2	
	M[TargetDepartureDate]	Select Departure date	1,2	
	P[NextMonthButton]	Move pointer to next button month in the calendar	1,1	
	K[mouse]	Click Next Month button upto target month June	0,2	

	K[mouse]	Click Next Month button upto target month June	0,2	
	K[mouse]	Click Next Month button upto target month June	0,2	
	P[target date]	Move pointer to selected date	1,1	
	K[mouse]	Click over the target date, select 4-6-2012	0,2	
	P[GoingTo]	Move pointer to labeled Going to input	1,1	
	K[mouse]	Click in Going to input	0,2	
	H[Keyboard]		0,4	
	M4K[word]	Type target destination, Rome	2	4
	H[Mouse]		0,4	
	P[Departing]	Move pointer to labeled Returning field	1,1	
	K[mouse]	Click Returning field (this opens the calendar)	0,2	
	M[SelectReturningDate]		1,2	
	P[NextMonthButton]	Move pointer to next button month in the calendar	1,1	
	K[mouse]	Click Next Month button upto target month Jule	0,2	
	P[target date]	Move pointer to selected date	1,1	
	K[mouse]	Click over the target date, select 21-6-2012	0,2	
	P[SearchButton]	Move pointer to Search For Flights button	1,1	

	K[mouse]	Click Search for Flights button	0,2
Once page is loaded:			
	M[SelectDepartureAirport]	Select a Departure Airport	1,2
	P[AirportRadioButton]	Move pointer to the target radio button	1,1
	K[Mouse]	Click Target Radio Button	0,2
	P[ContinueButton]	Move pointer to Coninute Button	1,1
	K[Mouse]	Click Continue Button	0,2
Once page is loaded:			
Step 1.2: Select Flights	M[TargetDepartureFlight]	Select Departure Flight	1,2
	P[TargetDepartureFlight]	Move pointer to Select Button for the Departure Flight	1,1
	K[Mouse]	Click Select Button for the Departure Flight	0,2
	M[TargetReturnFlight]	Select Return Flight	1,2
	P[TargetReturnFlight]	Move pointer to Select Button for the Return Flight	1,1
	K[Mouse]	Click Select Button for the Return Flight	0,2
Once page is loaded:			
Step 1.3: Login			
	P[UserInput]	Move pointer to Username Input	1,1
	K[Mouse]	Click Username input	0,2
	H[Keyboard]		0,4

	M21K[Username]	Type user name, firmenich.s@gmail.com	5,4	21
	H[Mouse]		0,4	
	P[PasswordInput]	Move pointer to Password input	1,1	
	K[Mouse]	Click Password input	0,2	
	H[Keyboard]		0,4	
	M8K[Password]	Type password, *******	2,8	8
Once page is loaded:				
Step 1.4: Entring Passenger Information	H[Mouse]		0,4	
	P[CountrySelector]	Move pointer to Country Select	1,1	
	K[Mouse]	Click Target Country	0,2	
	P[CountrySelector]	Move pointer to phone number input	1,1	
	K[Mouse]	Click Phone Number Input	0,2	
	H[Keyboard]		0,4	
	M10K[PhoneNumber]	Type Phone Number, 2216043021	3,2	10
	H[Mouse]		0,4	
	P[ContactFirstName]	Move pointer to Firsta Name Contact Emergency Input	1,1	
	K[Mouse]	Click First Name Conctact Input	0,2	
	H[Keyboard]		0,4	
	M10K[ContactFirstName]	Type First Name	3,2	10
	H[Mouse]		0,4	

	P[ContactLastName]	Move pointer to Last Name Contact Emergency Input	1,1	
	K[Mouse]	Click Last Name Conctact Input	0,2	
	H[Keyboard]		0,4	
	M10K[ContactFirstName]	Type Last Name	3,2	10
	H[Mouse]		0,4	
	P[PassportCountry]	Move pointer to Passport Country Selector	1,1	
	K[Mouse]	Click target Country	0,2	
	P[PassportNumber]	Move pointer to Passport Number Input	1,1	
	K[Mouse]	Click Passport Number Input	0,2	
	H[Keyboard]		0,4	
	M10K[PassportNumber]	Type Passport Number, 31092679N	3,2	10
	H[Mouse]		0,4	
	P[ContinueButton]	Move pointer to Continue Button	1,1	
	K[Mouse]	Click Continue Button	0,2	
Once page is loaded:				
Step 1.5: Pay flights				
	H[Mouse]		0,4	
	P[CardNumber]	Move pointer to Card Number Input	1,1	
	K[Mouse]	Click Card Number Input	0,2	
	H[Keyboard]		0,4	
	M20K[CardNumber]	Type Card Number	5,2	20

	H[Mouse]		0,4	
	P[CardType]	Move pointer to CardType selector	1,1	
	K[Mouse]	Click Card Type Selector	0,2	
	M[CardType]	Select Card Type	1,2	
	K[Mouse]	Click target Card Type	0,2	
	P[ExpirationMonth]	Move pointer to Expiration Month Selector	1,1	
	K[Mouse]	Click Expiration Month Selector	0,2	
	M[SelectMonth]		1,2	
	K[Mouse]	Click target Expiration Month	0,2	
	P[ExpirationMonth]	Move pointer to Expiration Year Selector	1,1	
	K[Mouse]	Click Expiration Year Selector	0,2	
	M[SelectYear]		1,2	
	K[Mouse]	Click target Expiration Year	0,2	
	P[CardIdentification-Code]	Move pointer to Card Identification-code Input	1,1	
	K[Mouse]	Click Card Identification -code Input	0,2	
	H[Keyboard]		0,4	
	M3K[CVCCode]	Enter Card Ientification-Code	1,8	3
	H[Mouse]		0,4	

	P[CardholderFirstName]	Move pointer to CardHolder First Name Input	1,1	
	K[Mouse]	Click CardHolder First Name Input	0,2	
	H[Keyboard]		0,4	
	M8K[FirstName]	Type Cardholder First Name, SERGIO D	2,8	8
	H[Mouse]		0,4	
	P[CardholderLastName]	Move pointer to CardHolder Last Name Input	1,1	
	K[Mouse]	Click CardHolder Last Name Input	0,2	
	H[Keyboard]		0,4	
	M8K[LastName]	Type Cardholder Last Name, FIRMENICH	2,8	8
Provide Cardholder Address				
	H[Mouse]		0,4	
	P[Non United States]	Move pointer to Radio Button Non United States of America	1,1	
	K[Mouse]	Click Radio Button Non United States of America	0,2	
	P[StreetInput]	Move pointer to Street Input input	1,1	
	K[Mouse]	Click Street Input input	0,2	
	H[Keyboard]		0,4	
	M12K[Street]	Type Street, Plaza Italia	3,6	12

	H[Mouse]		0,4	
	P[TownCity]	Move pointer to Town or City input	1,1	
	K[Mouse]	Click Town or City input	0,2	
	H[Keyboard]		0,4	
	M8K[CityName]	Type City name, La Plata	2,8	8
	H[Mouse]		0,4	
	P[State]	Move pointer to State input	1,1	
	K[Mouse]	Click State input	0,2	
	H[Keyboard]		0,4	
	M12K[State]	Type State name, Buenos Aires	3,6	12
	H[Mouse]		0,4	
	P[ZipCode]	Move pointer to Zip-Code input	1,1	
	K[Mouse]	Click ZipCode input	0,2	
	H[Keyboard]		0,4	
	M4K[ZipCode]	Type Zipe Code, 1900	2	4
	H[Mouse]		0,4	
	P[CountrySelector]	Move pointer to Country or Region input	1,1	
	K[Mouse]	Click Country or Re-gion input	0,2	
	K[Mouse]	Click Target Country	0,2	
	P[CountryPhoneSelector]	Move pointer to Country input for Phone	1,1	
	K[Mouse]	Click Country in-put for Phone	0,2	

	K[Mouse]	Click Target Country	0,2	
	P[PhoneInput]	Move pointer to Phone number input	1,1	
	K[Mouse]	Click Phone number input	0,2	
	H[Keyboard]		0,4	
	M10K[PhoneNumber]	Type Phone Number, 2216043021	3,2	10
	H[Mouse]		0,4	
	K[Mouse]	Click Confirm Booking	0,2	
Step 2: Booking Hotel				
Step 2.1: Search Accommodation				
	H[Mouse]	Reach for mouse	0,4	
	P[BrowserAddressBar]	Move pointer to Browser Address Bar	1,1	
	K[mouse]	Select Browser Address bar by clicking over it.	0,2	
	H[Keyboard]		0,4	
	M15K[word]	Enter the text “www.booking.com”	4,2	15
Once page is loaded:				
	P[Destination]	Move pointer to Destination Input	1,1	
	K[Mouse]	Click Destination Input	0,2	
	H[Keyboard]		0,4	
	M4K[Destination]	Enter a text, for example Rome, it shows autocompletion	2	4
	H[Mouse]		0,4	

	P[TargetDestination]	Move pointer to taget Destination in the autocomplete options	1,1
	K[Mouse]	Click target Destination, Rome (it shows check-in-date calendar)	0,2
	P[CheckInDate]	Move pointer to Next Button Month in the calendar	1,1
	K[Mouse] x 4 times	Click Next Month button upto see target month	0,8
	M[SelectCheckInDate]	Select check in date	1,2
	K[Mouse]	Click target Checkin Date	0,2
	P[CheckOutDay]	Move pointer to Check-out date selector	1,1
	K[Mouse]	Click in selector	0,2
	P[TargetCheckOutDay]	Move pointer to Check-out target day	1,1
	K[Mouse]	Click target checkoun day, 21	0,2
Once page is loaded:			
Step 2.2: Select Hotel			
	M[SelectTargetHotel]	Select target hotel	1,2
	P[TargetHotel]	Move pointer to Book now button for the target hotel	1,1
	K[Mouse]	Click Book now button for the target hotel	0,2
Once page is loaded:			
	M[SelectRoom]	Select type of room	1,2

	P[NoRooms]	Move pointer to number of rooms selector for the target type of room	1,1	
	K[Mouse]	Click in number of room selector	0,2	
	K[Mouse]	Click in the targat number of Rooms, 1	0,2	
	P[BookNow]	Move pointer to Book now button	1,1	
	K[Mouse]	Click Book now button	0,2	
Once page is loaded:				
Step 2.3: Personal Information	P[FirstName]	Move pointer to labeled First name input	1,1	
	K[Mouse]	Click labeled First name input	0,2	
	H[Keyboard]		0,4	
	M6K[FirstName]	Enter first name, Sergio	2,4	6
	H[Mouse]		0,4	
	P[LastName]	Move pointer to labeled Last name input	1,1	
	K[Mouse]	Click labeled Last name input	0,2	
	H[Keyboard]		0,4	
	M9K[LastName]	Enter last name, Firmenich	3	9
	H[Mouse]		0,4	
	P[Email]	Move pointer to labeled Email address input	1,1	
	K[Mouse]	Click labeled Email address input	0,2	

	H[Keyboard]		0,4	
	M26K[Rmail]	Enter email address, sergio.firmenich@gmail.com	6,4	26
	H[Mouse]		0,4	
	P[ConfirmEmail]	Move pointer to labeled Confirm Email address input	1,1	
	K[Mouse]	Click labeled Confirm Email address input	0,2	
	H[Keyboard]		0,4	
	M26K[ConfirmEmail]	Enter email address, sergio.firmenich@gmail.com	6,4	26
	H[Mouse]		0,4	
	P[Smoking]	Move pointer to Smooking selector	1,1	
	K[Mouse]	Click Smooking selector	0,2	
	K[Mouse]	Click target option, no	0,2	
	P[ContinueButton]	Move pointer to Continue button	1,1	
	K[Mouse]	Click Continue Button	0,2	
Once page is loaded:				
Step 2.4.:Pay Acomodation				
	P[Address]	Move pointer to Address labeled input	1,1	
	K[Mouse]	Click Address input	0,2	
	H[Keyboard]		0,4	
	M22K[Address]	Enter address, Plaza Italia n¡ 31 8 B	5,6	22

	H[Mouse]		0,4	
	P[City]	Move pointer to City labeled input	1,1	
	K[Mouse]	Click City input	0,2	
	H[Keyboard]		0,4	
	M8K[City]	Enter address, La Plata	2,8	8
	H[Mouse]		0,4	
	P[ZipCode]	Move pointer to Zip/-Post code input	1,1	
	K[Mouse]	Click Zip/Post code input	0,2	
	H[Keyboard]		0,4	
	M4K[ZipCode]	Enter Zip/Post code, 1900	2	4
	H[Mouse]		0,4	
	P[Country]	Move pointer to Country Selector	1,1	
	K[Mouse]	Click Country Selector	0,2	
	K[Mouse]	Click target Country	0,2	
	P[Telephone]	Move pointer to Telephone input	1,1	
	K[Mouse]	Click Telephone input	0,2	
	H[Keyboard]		0,4	
	M12K[Telephone]	Enter telephone, 542216099021	3,6	12
	H[Mouse]		0,4	
Credit Card Information	P[CardType]	Move pointer to CardType selector	1,1	
	K[Mouse]	Click Card Type Selector	0,2	
	M[CardType]	Select Card Type	1,2	

	K[Mouse]	Click target Card Type	0,2	
	P[CardNumber]	Move pointer to Card Number Input	1,1	
	K[Mouse]	Click Card Number Input	0,2	
	H[Keyboard]		0,4	
	M20K[CardNumber]	Type Card Number	5,2	20
	H[Mouse]		0,4	
	P[CardholderName]	Move pointer to Card Holder Name Input	1,1	
	K[Mouse]	Click CardHolder Name Input	0,2	
	H[Keyboard]		0,4	
	M18K[CardholderName]	Type Card Holder's Name, SERGIO D FIRMENICH	4,8	18
	H[Mouse]		0,4	
	P[ExpirationMonth]	Move pointer to Expiration Month Selector	1,1	
	K[Mouse]	Click Expiration Month Selector	0,2	
	M[SelectMonth]		1,2	
	K[Mouse]	Click target Expiration Month	0,2	
	P[ExpirationMonth]	Move pointer to Expiration Year Selector	1,1	
	K[Mouse]	Click Expiration Year Selector	0,2	
	M[SelectYear]		1,2	
	K[Mouse]	Click target Expiration Year	0,2	

	P[CVC-Code]	Move pointer to CVC-code Input	1,1	
	K[Mouse]	Click CVC-code Input	0,2	
	H[Keyboard]		0,4	
	M3K[CVCCode]	Enter CVC-Code	1,8	3
	H[Mouse]		0,4	
	P[Book]	Move pointer to Book this Room button	1,1	
	K[Mouse]	Click Book this room button	0,2	
		Total time (seconds):	240,6	

.4. Escenario realizado con PIMI

Cuadro 3: Lista de tareas de usuario y sus respectivas tareas específicas

Context Task	Action	Description	Time	K
Step 1: Buy Flights				
Step 1.1: Search Flights				
	H[Mouse]	Reach for mouse	0,4	
	P[BrowserAddressBar]	Move pointer to Browser Address Bar	1,1	
	K[mouse]	Select Browser Address bar by clicking over it.	0,2	
	H[Keyboard]		0,4	
	M15K[word]	“Enter the text ” “www.expedia.com”	4,2	15
Once page is loaded:				

	H[Mouse]		0,4	
	P[RadioButton]	Move the point to the target product Flights	1,1	
	K[mouse]	“Select the radio button” “Flight” by clicking over it	0,2	
	P[LeavingFrom]	“Move pointer to labeled” “Leaving from” “ field”	1,1	
	K[mouse]	“Select input labeled” “Leaving from” “by clicking over it”	0,2	
	H[Keyboard]		0,4	
	M12K[word]	“Enter a text, for example” “Buenos Aires”	3,6	12
	H[Mouse]		0,2	
	P[Departing]	Move pointer to labeled “Departing” field	1,1	
	K[mouse]	Click “Departing” field (this opens the calendar)	0,2	
	M[TargetDepartureDate]	Select Departure date	1,2	
	P[NextMonthButton]	Move pointer to next button month in the calendar	1,1	
	K[mouse]	Click Next Month button upto target month “June”	0,2	
	K[mouse]	Click Next Month button upto target month “June”	0,2	
	K[mouse]	Click Next Month button upto target month “June”	0,2	
	P[target date]	Move pointer to selected date	1,1	

	K[mouse]	Click over the target date, select 4 6 2012	0,2	
	P[GoingTo]	Move pointer to labeled “Going to” input	1,1	
	K[mouse]	Click in “Going to” input	0,2	
	H[Keyboard]		0,4	
	M4K[word]	Type target destination, “Rome”	2	4
	H[Mouse]		0,4	
	P[Departing]	Move pointer to labeled “Returning” field	1,1	
	K[mouse]	Click “Returning” field (this opens the calendar)	0,2	
	M[SelectReturningDate]		1,2	
	P[NextMonthButton]	Move pointer to next button month in the calendar	1,1	
	K[mouse]	Click Next Month button upto target month “Jule”	0,2	
	P[target date]	Move pointer to selected date	1,1	
	K[mouse]	Click over the target date, select 21 6 2012	0,2	
	P[SearchButton]	Move pointer to “Search For Flights” button	1,1	
	K[mouse]	Click “Search for Flights” button	0,2	
Once page is loaded:				
	M[SelectDepartureAirport]	Select a Departure Airport	1,2	

	P[AirportRadioButton]	Move pointer to the target radio button	1,1	
	K[Mouse]	Click Target Radio Button	0,2	
	P[ContinueButton]	Move pointer to Coninute Button	1,1	
	K[Mouse]	Click Continue Button	0,2	
Once page is loaded:				
Step 1.2: Select Flights	M[TargetDepartureFlight]	Select Departure Flight	1,2	
	P[TargetDepartureFlight]	Move pointer to “Select” Button for the Departure Flight	1,1	
	K[Mouse]	Click “Select” Button for the Departure Flight	0,2	
	M[TargetReturnFlight]	Select Return Flight	1,2	
	P[TargetReturnFlight]	Move pointer to “Select” Button for the Return Flight	1,1	
	K[Mouse]	Click “Select” Button for the Return Flight	0,2	
Once page is loaded:				
Step 1.3: Login				
	P[UserInput]	Move pointer to Username Input	1,1	
	K[Mouse]	Click Username input	0,2	
	H[Keyboard]		0,4	
	M21K[Username]	Type user name, firmenich.s@gmail.com	5,4	21
	H[Mouse]		0,4	
	P[PasswordInput]	Move pointer to Password input	1,1	

	K[Mouse]	Click Password input	0,2	
	H[Keyboard]		0,4	
	M8K[Password]	Type password, *******	2,8	8
Once page is loaded:				
Step 1.4: Entring Passenger Information	H[Mouse]		0,4	
	P[IdentityPIM]	Move pointer to Identity PIM	1,1	
	K[Mouse] – For dragging	Click Identity PIM for dragging	0,2	
	P[PersonalInformation]	Move pointer to personal information inputs area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
	P[CountrySelector]	Move pointer to Country Phone Select	1,1	
	K[Mouse]	Click Country Phone Selector	0,2	
	K[Mouse]	Click Country Phone Target	0,2	
	P[ContactInformation]	Move pointer to Contact Information PIM	1,1	
	K[Mouse] – For dragging	Click Contact Information PIM for dragging	0,2	
	P[PersonalInformation]	Move pointer to phone input area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
	P[PassportCountry]	Move pointer to Passport Country Selector	1,1	

	K[Mouse]	Click Passport Country Selector	0,2	
	K[Mouse]	Click target Country	0,2	
	P[ContinueButton]	Move pointer to Continue Button	1,1	
	K[Mouse]	Click Continue Button	0,2	
Once page is loaded:				
Step 1.5: Pay flights				
	P[VisaCreditCardPim]	Move pointer to Visa Credit Card PIM	1,1	
	K[Mouse] – For dragging	Click Visa Credit Card PIM for dragging	0,2	
	P[CreditCardFields]	Move pointer to Credit Card inputs area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
	P[CardIdentification Code]	Move pointer to Card Identification code Input	1,1	
	K[Mouse]	Click Card Identification code Input	0,2	
	H[Keyboard]		0,4	
	M3K[CVCCode]	Enter Card Ientification Code	1,8	3
	H[Mouse]		0,4	
	P[CountrySelector]	Move pointer to “Country” input	1,1	
	K[Mouse]	Click “Country” input	0,2	
	K[Mouse]	Click Target Country	0,2	

	P[PersonalAddress]	Move pointer to Personnal Address PIM	1,1	
	K[Mouse] – For dragging	Click Personnal Address PIM for dragging	0,2	
	P[PersonalInformation]	Move pointer to Address inputs area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
	P[CompleteBooking]	Move pointer to Complete Booking button	1,1	
	K[Mouse]	Click Complete Booking	0,2	
Step 2: Booking Hotel				
Step 2.1: Search Accommodation				
	H[Mouse]	Reach for mouse	0,4	
	P[BrowserAddressBar]	Move pointer to Browser Address Bar	1,1	
	K[mouse]	Select Browser Address bar by clicking over it.	0,2	
	H[Keyboard]		0,4	
	M15K[word]	“Enter the text” “www.booking.com”	4,2	15
Once page is loaded:				
	P[Destination]	Move pointer to “Destination” Input	1,1	
	K[Mouse]	Click “Destination” Input	0,2	

	M4K[Destination]	Enter a text, for example “Rome”, it shows autocompletion	2	4
	H[Mouse]		0,4	
	P[TargetDestination]	Move pointer to taget Destination in the autocomplete options	1,1	
	K[Mouse]	Click target Destination, “Rome” (it shows check in date calendar)	0,2	
	P[CheckInDate]	Move pointer to Next Button Month in the calendar	1,1	
	K[Mouse] x 4 times	Click “Next Month” button upto see target month	0,8	
	M[SelectCheckInDate]	Select check in date	1,2	
	K[Mouse]	Click target Checkin Date	0,2	
	P[CheckOutDay]	Move pointer to Check out date selector	1,1	
	K[Mouse]	Click in selector	0,2	
	P[TargetCheckOutDay]	Move pointer to Check out target day	1,1	
	K[Mouse]	Click target checkoun day, “21“	0,2	
Once page is loaded:				
Step 2.2: Select Hotel				
	M[SelectTargetHotel]	Select target hotel	1,2	
	P[TargetHotel]	Move pointer to “”Book now” button for the target hotel	1,1	

	K[Mouse]	Click “Book now” button for the target hotel	0,2
Once page is loaded:			
	M[SelectRoom]	Select type of room	1,2
	P[NoRooms]	Move pointer to number of rooms selector for the target type of room	1,1
	K[Mouse]	Click in number of room selector	0,2
	K[Mouse]	Click in the targat number of Rooms, “1”	0,2
	P[BookNow]	Move pointer to “Book now” button	1,1
	K[Mouse]	Click “Book now” button	0,2
Once page is loaded:			
Step 2.3: Personal Information			
	H[Mouse]		0,4
	P[IdentityPIM]	Move pointer to Identity PIM	1,1
	K[Mouse] – For dragging	Click Identity PIM for dragging	0,2
	P[PersonalInformation]	Move pointer to personal information inputs area in the form	1,1
	K[Mouse] For Dropping	Release Click	0,2
	P[ContactInformation]	Move pointer to Contact Information PIM	1,1

	K[Mouse] – For dragging	Click Contact Information PIM for dragging	0,2
	P[PersonalInformation]	Move pointer to phone input area in the form	1,1
	K[Mouse] For Dropping	Release Click	0,2
	P[Smoking]	Move pointer to “Smooking” selector	1,1
	K[Mouse]	Click “Smooking” selector	0,2
	K[Mouse]	Click target option, “no”	0,2
	P[ContinueButton]	Move pointer to “Continue” button	1,1
	K[Mouse]	Click “Continue” Button	0,2
Once page is loaded:			
Step 2.4.:Pay Accomodation			
	P[PersonalAddress]	Move pointer to Personal Address PIM	1,1
	K[Mouse] – For dragging	Click Personal Address PIM for dragging	0,2
	P[PersonalAddress]	Move pointer to personal address inputs area in the form	1,1
	K[Mouse] For Dropping	Release Click	0,2
	P[ContactInformation]	Move pointer to Contact Information PIM	1,1
	K[Mouse] – For dragging	Click Contact Information PIM for dragging	0,2

	P[PersonalInformation]	Move pointer to phone input area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
Credit Card Information	P[VisaPim]	Move pointer to Visa Credit Card PIM	1,1	
	K[Mouse] – For dragging	Click Visa Credit Card PIM for dragging	0,2	
	P[CreditCardFields]	Move pointer to Credit Card inputs area in the form	1,1	
	K[Mouse] For Dropping	Release Click	0,2	
	P[CardIdentification Code]	Move pointer to Card Identification code Input	1,1	
	K[Mouse]	Click Card Identification code Input	0,2	
	H[Keyboard]		0,4	
	M3K[CVCCode]	Enter Card Ientification Code	1,8	3
	P[Book]	Move pointer to “Book this Room” button	1,1	
	K[Mouse]	Click “Book this room” button	0,2	
		Total time (seconds):	116,8	

Anexo C: Publicaciones Científicas

Sergio Firmenich, Gustavo Rossi, Matias Urbieta, Silvia E. Gordillo, Cecilia Challiol, Jocelyne Nanard, Marc Nanard and Joao Araújo. Engineering concern-sensitive navigation structures, concepts, tools and examples. Journal of Web Engineering, 9 (2):157–185, Rinton Press, 2010.

En el artículo se ha puesto en valor la teoría de navegación sensible al *concern* ampliando el uso de esta herramienta conceptual respecto de su definición original Nanard et al. [2008]. En este trabajo se hizo particular hincapié en la importancia de contemplar al usuario como usuario de varias aplicaciones llevando así la aplicación de los conceptos de CSN al cliente para poder soportar navegación inter-aplicación.

Además en este artículo se gestaron las primeras herramientas para asistir al usuario en la definición de adaptaciones basadas en el *concern* del usuario.

Sergio Firmenich, Silvia E. Gordillo, Gustavo Rossi and Marco Winckler. Client- side adaptation: An approach based in reutilization using transversal models. In ICWE Workshops, volume 6385 of Lecture Notes in Computer Science, pages 566–570. Springer, 2010.

En el artículo se enfatizo en la reutilización de adaptaciones basadas en modelos abstraídos del DOM. En este artículo además se definió una primera clasificación de los tipos de adaptaciones que pueden ser aplicadas del lado del cliente.

Sergio Firmenich, Gustavo Rossi, Silvia E. Gordillo and Marco Winckler. A flexible architecture for client-side adaptation. In ICWE Workshops, volume 7059 of Lecture Notes in Computer Science, pages 327–331. Springer, 2011.

Los fundamentos para una arquitectura flexible de adaptación del lado del cliente fueron presentados en el artículo. Aquí, además, se establecieron los objetivos principales de investigación que son parte de los objetivos presentados en esta tesis.

Sergio Firmenich, Marco Winckler, Gustavo Rossi and Silvia E. Gordillo. A framework for concern-sensitive, client-side adaptation. In ICWE, volume 6757 of Lecture Notes in Computer Science, pages 198–213. Springer, 2011.

El framework presentado en el capítulo 4 es una evolución de lo que originalmente fue publicado en el artículo. Este artículo incluyó por primera vez los conceptos de *Pocket*, *Augmenters* y *DataCollectors*. Este artículo fue distinguido con el premio *Best Paper Award*.

Sergio Firmenich, Marco Winckler and Gustavo Rossi. A tool support for web applications adaptation using navigation history. In INTERACT (4), volume 6949 of Lecture Notes in Computer Science, pages 340–348. Springer, 2011.

En este artículo se pone en valor el historial de navegación para ser utilizado como información en la ejecución de adaptaciones.

Sergio Firmenich, Marco Winckler, Gustavo Rossi and Silvia E. Gordillo. A crowdsourced approach for concern-sensitive integration of information across the web. Journal of Web Engineering., 10(4):289–315, Rinton Press, 2011.

En el artículo se retoma el framework definido en Firmenich et al. [2011] y se brinda un mejor enfoque respecto a las adaptaciones CSN basadas en modelos asbtraídos del DOM y al concepto de lo que se llamo *scenario*.

Marco Winckler, Vincent Gaits, Dong-Bach Vo, Sergio Firmenich and Gustavo Rossi. An approach and tool support for assisting users to fill-in web forms with personal information. 29th ACM international conference on Design of communication, 2011.

En este artículo se presentó un enfoque que basado en un espacio de información personal estructurado con Microformatos permiten completar formularios anotados. La herramienta *PIMI* presentada en esta tesis y con la que se han realizado los casos de estudio del capítulo 6 están inspiradas en las ideas de este artículo.

Sergio Firmenich, Vincent Gaits, Silvia E. Gordillo, Gustavo Rossi and Marco Winckler. Supporting users tasks with personal information management and web forms augmentation. In ICWE, volume 7387 of Lecture Notes in Computer Science, pages 268–282. Springer, 2012

En este artículo se utilizan técnicas de anotación de formularios en el cliente. De esta manera se complementa la herramientas de espacio personal de información presentada en Winckler et al., ya que ahora incluye tanto anotación como soporte para la compleción de formularios.

Además se propone un esquema colaborativo en donde las anotaciones realizadas por cualquier usuario en un sitio Web son tomadas en cuenta cuando cualquier otro usuario carga dicho sitio.

En este artículo también se presenta la evaluación que demuestra la conveniencia de estas herramientas para tareas que implican uso intensivo de formularios.

Alejandra Garrido, Sergio Firmenich, Gustavo Rossi, Julian Grigera, Nuria Medina Medina and Ivana Harari. Personalized web accessibility using client-side refactoring. IEEE Internet Computing.

En este artículo se abarca una categoría específica de todas las que se han definido en la taxonomía incluida en esta tesis. En este caso se utilizo el framework de adaptación Firmenich et al. [2011] como software base para obtener un enfoque de adaptación para la mejora de accesibilidad en aplicaciones existentes bajo la teoría de *refactoring*.

Marco Winckler, Sergio Firmenich, Gustavo Rossi and Philippe Palanque. An approach for supporting distributed user interface orchestration over the web. International Journal of Human-Computer Studies.

En este artículo se presenta el enfoque y las herramientas de *escenarios* basada en la composición de tareas primitivas y tareas de augmentación. Aquí también

se hace hincapié en la colaboración entre usuarios[1]. Es importante aclarar que en esta publicación, se ha dado el nombre de *Procedure* a lo que se llamó *Escenario* en el transcurso de esta presentación.

[1]Este artículo esta actualmente en la segunda ronda de revisión al 20/12/2012