TESINA DE GRADO

Universidad Nacional de La Plata

Adaptabilidad en familia de aplicaciones Web

Autor: Margiotta Hugo D.

Director: Gustavo Rossi

Carrera: Licenciatura en Sistemas

Índice general

Capitulo1. Introducción y presentación del problema

La Web, ha ido evolucionando a lo largo del tiempo, junto con las técnicas y tecnologías que se utilizan para brindar y recolectar información. Pero a pesar de esta evolución, la mayoría de estas técnicas y tecnologías no tienen presente, las nuevas necesidades crecientes de los usuarios finales. Donde estos, requieren de una mayor interacción, una mejor navegación y de una Web más personal. La mayoría de estas técnicas, recolectan y proveen de información al usuario y la manipulan a sus gustos para lograr mejorar la experiencia de los usuarios, pero al mismo tiempo, evitan que los mismos, tengan una interacción considerable en el uso de esta información. Ningún diseño, puede proveer información para cada situación y ningún diseñador, puede incluir información personalizada para cada usuario.

Con la llegada de la web 2.0, las aplicaciones pasaron de ser simples repositorios de información y se convirtieron en plataformas que permiten realizar procesos complejos. Esta nueva web, trajo consigo un nuevo enfoque para la adaptación de sitios Web. Este enfoque es conocido como DIY (do-it-yourselft), donde son los propios usuarios quienes mejoran o adaptan los sitios Web a sus propias necesidades. Un cliente popular, basado en la tecnología DIY es Javascript, usando plugins especiales, como por ejemplo, Greasemonkey [25]. Estos scripts, utilizan la infraestructura que provee el DOM (Document Object Model [42]), para poder obtener los diferentes elementos que componen a la página Web y que se encuentran representados mediantes los nodos del mismo. De esta forma, acceden a ellos y finalmente logran cambiar la pagina Web, para cumplir con las necesidades del usuario. En la Fig. 1, se ve un script utilizando Grasemonkey:

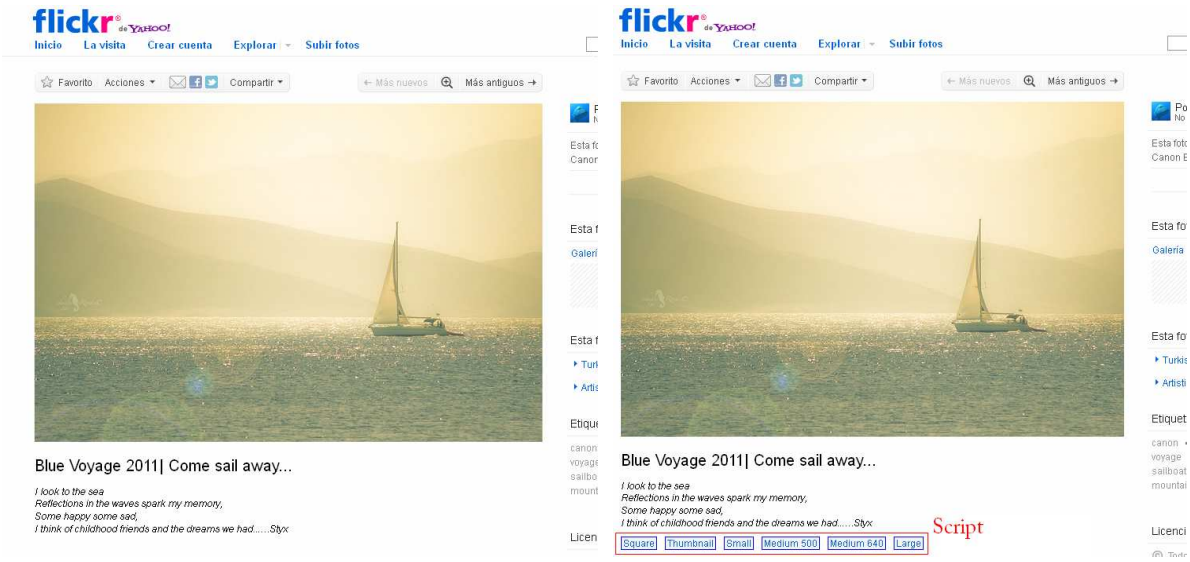

Fig. 1.1 a - Página sin el script.

Fig. 1.1 b - Greasemonkey script.

En la Fig. 1.1 b, se puede observar como el script agrega un nueva funcionalidad a la aplicación mediante una serie de etiquetas, que se insertan al DOM, con la creación de nuevos nodos utilizando Javascript. En este caso, esta nueva funcionalidad permite visualizar la imagen en diferentes tamaños.

Uno de los grandes atractivos de desarrollar scripts, para ser aplicados a través de Gracemonkey, es que pueden ser subidos a repositorios, como por ejemplo userscrips.org y ser compartidos por miles de usuarios. Entonces, un usuario podría bajar varios scripts desarrollados por otros y utilizarlos solo con agregarlos al plugin.

Pero qué pasa si se quisiera utilizar un script, en otra página que provee la misma información, incluso tal vez, casi la misma estructura. Consideremos el script anterior, a grandes rasgos podríamos decir que el script agrega una serie de etiquetas, a una foto, en un sitio donde se suben y comparten fotos. Se podría decir, que este script, se aplica a sitios donde se comparten fotos, por lo tanto debería ser posible aplicar el mismo script en otra página similar, por ejemplo en Photobucket [38] o deviantART [15]. Sin embargo, esto no es posible, ya que el script ha sido desarrollado sin tener en cuenta el concepto de familias de aplicaciones y con un nivel bajo de abstracción.

El script accede directamente a la estructura de la pagina (DOM), por ende si la estructura cambia, el script deja de funcionar. En este trabajo se propone una una forma de proteger el script de los cambios en las páginas, esto implica separar las partes más estables del script, de aquellas expuestas a los cambios más frecuentes.

El objetivo principal de esta tesis, es presentar un método para construir adaptaciones (desarrolladas a través de scripts), que puedan ser reutilizadas a lo largo de diversas aplicaciones, pertenecientes a una misma familia. Esta meta, se apoya, en la utilización de un framework, que se basa en la estructura presentada en [3], el cual provee un ambiente de desarrollo que permite construir adaptaciones, con un nivel de abstracción suficiente para ser reutilizables.

En lo que respecta a adaptaciones Web, muchos trabajos se han desarrollado en los últimos años, algunos de ellos, se encuentran relacionados con las adaptaciones o el entorno que se intentan construir aquí. Para lograr un mejor contraste y obtener una mayor experiencia, en la lectura de esta tesis, es que, a lo largo de las secciones de la misma, se analizan trabajos que se encuentran relacionados de diversas maneras, en general, apuntando principalmente al tema de adaptaciones o personalización en la Web. Estos trabajos, que pueden llegar a aportar desde técnicas o tecnologías hasta conceptos o tendencias, se analizan y comparan dentro del contexto de la tesis. Algunos de ellos, incluso, serán analizados desde distintos puntos vista a través de las diferentes secciones.

La tesis comienza en 1, introduciendo el problema que la misma intenta atacar, con una breve idea de la técnica que se utilizara para poder sobrellevar el mismo. En el capítulo 2, se presentan los conceptos básicos sobre los cuales se apoya el framework y se introduce al

lector, en un poco de la historia de la Web y algunas tecnólogas, técnicas y herramientas que la ayudaron a escalar a lo que es hoy en día. En 3, veremos en detalle la técnica de adaptaciones desde el lado del servidor y desde lado del cliente, y analizaremos para esta última, algunas de sus aplicaciones más importantes. En 4, introduciremos el concepto de familias de aplicaciones Web, analizaremos cómo es posible aplicarlo para reutilizar adaptaciones e iremos encaminándonos, hacia la idea de utilizar este concepto a través de un famrework. En 5, profundizaremos en el desarrollo del framework, mostraremos cómo se desarrolla este framework para poder ser utilizado en familias de aplicaciones Web y analizaremos cada una de sus partes y las técnicas y herramientas que lo complementan. Finalmente en 6, analizaremos un caso de estudio para poder comprobar la efectividad del framework y ver como el mismo se comporta.

Quisiera aprovechar, para agradecer en especial a una persona, quien sin su ayuda, este trabajo hubiera resultado en una tarea mucho más compleja. Sergio Firmenich, quien desarrollo el framework en [3], y quien además me aconsejo y guio a lo largo de la construcción del framework que presentamos aquí.

Capitulo 2. Conceptos Básicos

Antes de poder empezar a discutir sobre los objetivos de esta tesis y los medios para alcanzarlos, es necesario comprender las bases e ideas sobre las cuales se fundamenta. Además también, como en cualquier otro estudio, es siempre importante conocer un poco la historia, y como los sucesos de eventos que se comprendieron en ella llevaron al objeto de estudio a lo que es hoy en día. En nuestro caso por supuesto que estamos hablando de la Web y por lo tanto, veremos una pequeña reseña de aquellas tendencias, tecnologías y herramientas que la empujaron a evolucionar hasta lo que es hoy. Daremos un pequeño vistazo a todos aquellos factores que ayudaron a que la Web se convirtiera en algo cada vez más conectado con el usuario, con más interacción y lo que es más importante, al menos a los fines de esta tesis, a una Web más personalizada.

2.1 La nueva Web

Como ya hemos mencionado anteriormente la Web evoluciona con el tiempo, de hecho se encuentra en una permanente carrera de evolución. Uno de los factores principales que impulsa a la Web hacia esta continua evolución es el usuario, donde desde nuestro punto de vista, es la constante necesidad de nuevas formas de interacción de la Web con los usuarios, lo que la empuja hacia una nueva y mejor versión de sí misma. Donde esta nueva versión, no se refiere estrictamente a una actualización de las especificaciones técnicas de la misma, sino más bien a cambios importantes en la forma en la que desarrolladores de software y usuarios finales utilizan la Web. Gracias a esto ha nacido una nueva Web, normalmente conocida como Web 2.0, "el término Web 2.0 está asociado a aplicaciones web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración en la World Wide Web" [48]. Donde este diseño centrado en el usuario se refiere en parte a la creación de aplicaciones Web que consigan resolver las necesidades concretas de los usuarios, obteniendo la mayor satisfacción posible y la mejor experiencia al usar estas mismas aplicaciones.

Esta nueva Web 2.0 trajo consigo un nuevo enfoque que no se encontraba presente en la Web 1.0, y este fue "involucrar al usuario", y esto no solo en el contenido que agregan, sino también en la forma en que ayudan a organizarlo, compartirlo, modificarlo, criticarlo, actualizarlo, etc. En resumen, "Tu", es decir el usuario final, eres el aspecto más importante en el entorno de la Web 2.0.

Como ha de esperarse nuevas técnicas y tecnologías han surgido como resultado de las crecientes necesidades que se desprenden a partir de los nuevos conceptos introducidos en las tendencias de la Web 2.0. La antigua Web no pasaba de ser más que un Sistema de Hipertexto, podríamos decir que esta Web era un Sistema de Hipertexto de escala mundial, puesto que cualquier usuario podía poner su página en la red y establecer enlaces a cualquiera de los documentos disponibles en ella. El Sistema Hipertextual por excelencia es el sistema

de la World Wide Web (WWW) [49] basado en el lenguaje HTML (HyperText Markup Language) [28].

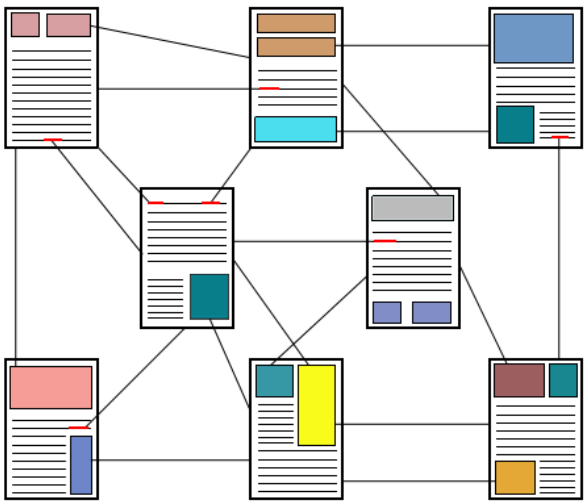

Fig. 2.1.1 – Sistemas de Hipertexto y la WWW

Las nuevas tecnologías que nacieron o se actualizaron para cubrir las crecientes necesidades de la Web, llevaron a que los sitios Web dejaran de ser simples repositorios de información compuestos solo por páginas Web y pasaran a ser plataformas, las cuales pueden llegar a contener aplicaciones Web. Estas aplicaciones permiten a los usuarios que las utilizan realizar procesos más ricos y complejos, permitiéndoles obtener una mejor experiencia.

Para cubrir parte del nuevo potencial que trajo el entorno de la Web 2.0 es que nacieron las aplicaciones Web, estas permitieron reducir de manera considerable la distancia que existía entre los sitios Web y sus usuarios. Agregando nuevas funcionalidades que les permitieran a los usuarios una mayor interacción y al mismo tiempo, tener algo de control sobre las acciones que podían realizar dentro del sitio Web. Sin embargo, todo tiene su límite y en cuanto a lo que las aplicaciones se refieren, están limitadas en lo que pueden llegar a ofrecer, es decir al funcionamiento que proveen, ya que este funcionamiento es impuesto por el sitio Web. En simples palabras una aplicación Web, solo puede brindar la funcionalidad para la que se encuentra diseñada.

Resulta obvio que a medida que la Web fue creciendo, los nuevos usos y acciones que se podían realizar a través de las páginas Web fueron complicando la implementación de las mismas. Al poco tiempo quedó presente que las prestaciones brindadas por el HTML, no eran suficientes para realizar todas las acciones que se pueden llegar a necesitar en una página web. En otras palabras, HTML se había quedado corto ya que sólo alcanzaba para presentar el texto de las páginas Web, definir su estilo y poco más. La nueva Web, necesitaba apoyarse en nuevas tecnologías que pudieran sacar a flote todo su potencial, entre muchas otras nosotros destacamos a JavaScript como una tecnología que ayudo a impulsar la Web hacia una nueva frontera.

Incluso antes de la Web 2.0 ya existía la idea y la implementación de una Web dinámica, para esto se utilizaban tecnologías que se apoyaban sobre el HTML para brindar ese dinamismo a las páginas Web, sin embargo la mayoría de los lenguajes que se utilizaban, necesitaban ser procesados y compilados del lado del servidor. JavaScript por su parte, es un lenguaje interpretado, que no requería ser compilado y podía ser embebido dentro del HTML de las páginas Web, en el lado del cliente (navegador). El hecho de que JavaScript se ejecute del lado del cliente, es sumamente importante, pues ahora los propios usuarios finales no solo pueden ver la lógica de las aplicaciones Web que se encuentran en el sitio, sino que además pueden modificarla, agregando nueva funcionalidad a estas mismas aplicaciones.

Estas nuevas tecnologías que mantenían todo el procesamiento de las aplicaciones Web del lado del cliente, evitando tener que ser tratadas en el servidor para luego recién ver los resultados, abrieron nuevos caminos para los usuarios finales. Una de las primeras tendencias nacidas a partir de estas, fue una en que los usuarios podían crear nuevas aplicaciones Web a partir de la combinación de otras existentes, esto se conoció como Mashup. Permitió a los usuarios finales sacar aun más provecho del potencial que proveía la nueva plataforma, que ahora eran las páginas Web, impuesta por la Web 2.0.

Esta nueva Web, en lo que a nosotros respecta, giro siempre alrededor del usuario final. Se trataba de lograr una mejor comunicación, y al mismo tiempo, lograr una mayor interacción con los mismos. Con la llegada de nuevas tecnologías, herramientas y la creación de técnicas innovadoras, como Mashup, que supieran sacar provecho al uso de las mismas, esta meta se encontraba mucho más cerca. Estas técnicas, no nacieron simplemente de un día para el otro, surgieron como parte de una nueva tendencia, un nuevo enfoque, que estaba comenzando a construirse dentro del ambiente de la Web 2.0. Este se conoció como DIY (Do-It-Yourself), que significa “hazlo tu mismo”. Tan simple y directo como suena, esta nueva tendencia hablaba sobre que sean los mismos usuarios finales, quienes utilizando los recursos de la Web, satisfagan sus propias necesidades y deseos.

Los usuarios comenzaron a experimentar con la plataforma que ahora era la Web. A través de los párrafos anteriores hay una idea u objetivo que se mantiene presente, aunque en algunas ocasiones resulte algo transparente, y es el de personalización. Para conseguir derribar, o al menos acortar, esa barrera que separa a los usuarios y a los desarrolladores de las aplicaciones, es necesario lograr esa mayor interacción y comunicación de la que venimos hablando. Con DIY, los usuarios empezaron a buscar sus propios medios para añadir ese extra de personal a la Web. Como todo en la vida, cuanto más personal se vuelven las cosas con las que uno se encuentra conectado, mayor y mejor será la experiencia que se obtenga de ellas, es como cuando un mecánico tunea su propio auto o un arquitecto diseña su propia casa. De esta forma, cuanto más personal se vuelva el sitio Web mayor y mejor será la experiencia que el usuario obtenga de él.

Los usuarios de estos mismos sitios, comprendieron que no solo era posible utilizar las aplicaciones que ya brindaba el sitio para crear nuevas, sino que también era posible modificarlas, para agregar nuevas funciones y procesos, llegando a un punto en el que incluso se podían crear totalmente desde cero nuevas aplicaciones Web, que funcionaran dentro del

sitio, como si el mismo sitio las hubiera implementado desde un principio. Con esto nació una nueva tendencia dentro de DIY, en la que los usuarios utilizando técnicas de Scripting y la tecnología dejada por JavaScript, utilizaban las plataformas para construir sus propias aplicaciones Web, llevando el sitio a algo cada vez más personalizado. Esta nueva tendencia, que hoy en día se conoce bajo el nombre de "tunnng" o "modding", está actualmente en auge y ha tenido una gran aceptación en el mundo Web, tanto así, que incluso los sitos Web luego incorporan funciones o aplicaciones Web enteras que son desarrolladas por los propios usuarios, ya que las mismas tienden a volverse muy populares entre los mismos. Existen incluso comunidades donde los usuarios comparten los scripts que desarrollan.

2.2 MASHUPS

Un concepto relacionado importante es el de Mashups, ya que ha sido una técnica que ha estado en constante crecimiento en los últimos años como medio para proveer personalización en la web y mejorar la experiencia del usuario. Existen una gran cantidad de trabajos para mejorar o enriquecer aplicaciones Web que se basan en el concepto de Mashup, algunos de estos, proponen ideas novedosas sobre técnicas para la aplicación del mismo, que le permiten comportarse, en algunos aspectos, como las adaptaciones que desarrollaremos aquí. Debido a lo anterior y a que ha sido, no solo, la predecesora de nuestras adaptaciones, sino que, aun se desarrollan técnicas novedosas para su aplicación, es que discutiremos sobre este concepto. Veremos una herramienta en particular, conocida como "Mash Maker" [27], y analizaremos las desventajas con respecto a nuestras adaptaciones.

Mientras la integración de datos se ha basado históricamente en permitir a los propietarios de los datos relacionar sus propios datos entre sí en formas bien planeadas y bien estructuradas, Mashups se trata de permitir a participantes/usuarios arbitrarios crear aplicaciones reutilizando una serie de fuentes de datos existentes, sin que los creadores de estos mismos datos tengan que estar involucrados. Esto es un punto importante ya que no son los propios diseñadores de las aplicaciones (como en el caso de los sistemas de adaptación hipermedia) quienes deben pensar en cómo presentar la información al usuario sino que son los mismos usuarios quienes satisfacen sus propias necesidades.

Básicamente, en el entorno del desarrollo web, un Mashup es una página web o aplicación que usa y combina datos, presentaciones y funcionalidad procedentes de una o más fuentes para crear nuevos servicios. El término implica integración fácil y rápida, usando a menudo APIs abiertas al público y fuentes de datos (provenientes de diferentes sitios) para producir una nueva información enriquecida.

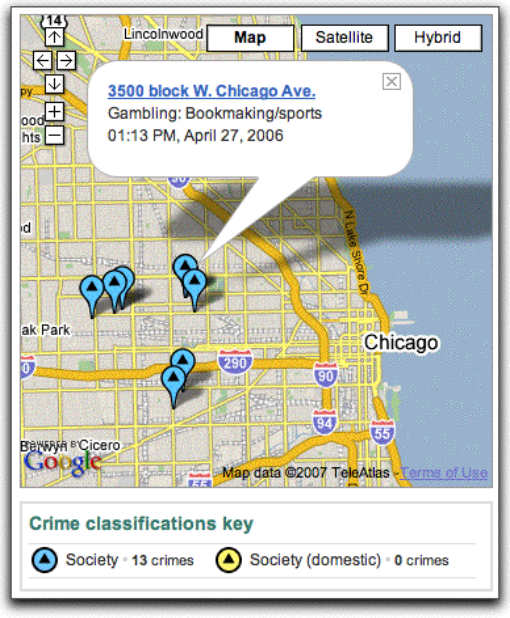

Fig. 2.2.1 – Departamento de policía de Chicago

En la Fig. 2.2.1 puede verse un ejemplo de un Mashup funcionando en el website del departamento de policía de Chicago (http://gis.chicagopolice.org/ CLEARMap_crime_sums/ startPage.htm) el cual integra la base de datos del departamento de crímenes reportados con Google Maps con el objetivo de ayudar a detener crímenes en ciertas áreas y avisar a los ciudadanos de áreas potencialmente más peligrosas.

Esta novedosa técnica, es una poderosa herramienta para proveer personalización en la web, pero como puede verse esto está lejos de acercarse a nuestra noción de una adaptación, ya que por definición Mashup implica proveer al usuario de un nuevo sitio o una nueva aplicación web y no adaptar una aplicación web existente. Sin embargo, así como el contenido de la web y sus nuevas forma de utilización han ido evolucionando, también se han desarrollado nuevas técnicas para poder utilizar Mashup de una forma que permita adaptar aplicaciones web.

En [27], se propone una forma de clasificar los métodos que se utilizan para realizar Mashups como "de composición" y "de personalización", de acuerdo a los roles que desempeñan las aplicaciones fuente. Los métodos de composición son semejantes a los esfuerzos de integración para construir nuevas aplicaciones a partir de recursos existentes. Este es el uso más común para el desarrollo de Mashups. Por contraste, los métodos de personalización se enfocan en una aplicación web dada, la cual es luego mejorada mediante la utilización de Mashups.

Este nuevo método para el desarrollo y aplicación de Mashups se encuentra más encaminado hacia nuestra idea de utilizar adaptaciones en aplicaciones web mejorando su estructura y por ende proveyendo una navegación más rica. Una de las pocas y novedosas herramientas que utiliza este método es Mash Maker [26]. De acuerdo a sus autores, "Mash Maker aumenta la familiar interfaz de navegación web que el usuario ya usa para navegar a través de los datos en la aplicación web, y la mejora con información mezclada ('mashed up')" [26].

“Mash Maker es un proyecto dentro de Intel Research, que apunta a impulsar el desarrollo o creación de Mashups unos pocos pasos más adelante” [26]. Con el fin de realizar este objetivo, Mash Maker utiliza datos estructurados de aplicaciones web existentes para crear nuevas interfaces mezcladas (“mashed up”) combinando información de muchas aplicaciones web o sitios web diferentes.

A medida que el usuario navega a través de la web, Mash Maker despliega una barra en el browser con botones que representan las adaptaciones que Mash Maker piensa que el usuario tal vez quiere aplicar en la aplicación web que se encuentra navegando en ese momento. El usuario solo necesita activar alguna combinación de estas adaptaciones propuestas para crear un nuevo Mashup (ej. para mapear ítems en un mapa, el usuario solo tiene que clicker en el botón Google Maps).

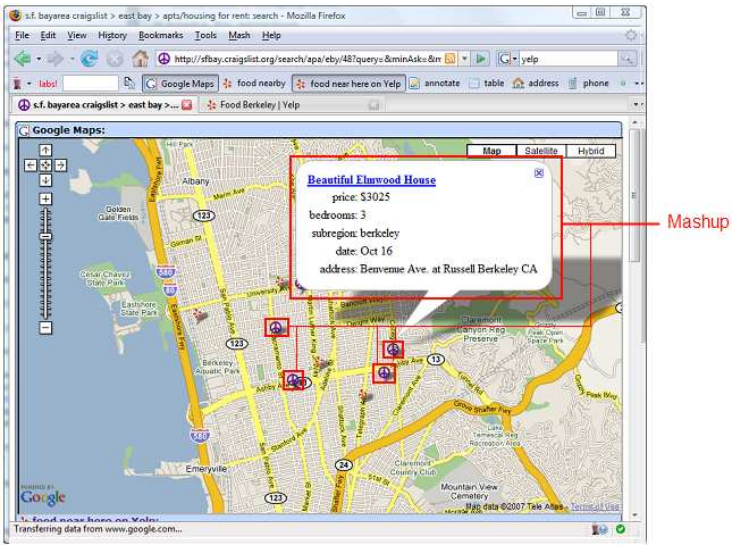

Fig. 2.2.2 – Mash Maker

En la Fig. 2.2.2 se ve un ejemplo de una aplicación web (ej. Google Maps) la cual ha sido mejorada utilizando Mash Maker. En este caso se mapean iconos sobre el mapa que representan departamentos disponibles que un usuario busco a través de Craigslist [13] (Craigslist es una red centralizada de comunidades urbanas en línea, ofreciendo anuncios clasificados gratis de empleo, vivienda, sentimentales, artículos para la venta/trueque/se busca, servicios, comunidad, etc.).

Aunque, Mash Maker implementa una nueva manera de utilizar Mashups para mejorar la estructura de aplicaciones web existentes, presenta ciertas desventajas con respecto a las adaptaciones creadas utilizando scripts. En primer lugar, Mash Maker, no deja de ser una herramienta que se basa en la utilización de Mashups, esto viene al hecho de que las adaptaciones que realiza en las aplicaciones web son o una combinación de datos pertenecientes a otras páginas web existentes o una nueva forma de visualizar los datos que se encuentran en la página web de forma que se adecue a las necesidades del usuario. Con lo cual no existe realmente nueva información, si bien la información de otras páginas puede ser

utilizada de una nueva forma, permitiendo que cumpla un objetivo diferente para la cual no fue diseñada, no deja de ser una forma de reutilizar la información. Aunque es cierto que la mayoría de las adaptaciones que se realizan actualmente están relacionadas con añadir link a otras páginas (ej. links a Wikipedia), enriquecer la estructura de un nodo hipermedia con datos de otras aplicaciones (ej. botones para compartir en redes sociales) o insertar cierta funcionalidad que probablemente puede tener su origen en otras aplicaciones web, un usuario podría querer crear nueva información que no se encuentra en ninguna otra página web o que no puede crearse a partir de la combinación de la información existente.

Por el contrario las adaptaciones que se desarrollan aquí son creadas utilizando scripts (desarrollados mediante Javascript), lo que permite un sinfín de posibilidades a la hora de enriquecer nodos hipermedia con nueva información o funcionalidades. Más aun las adaptaciones en Mash Maker están limitadas a realizarse cuando la página se carga, es decir estas adaptaciones solo se pueden asociar con el evento de carga de la página mientas que los scripts pueden ser asociados a cualquier evento DOM. Esto da mucha más flexibilidad a en el desarrollo de adaptaciones.

Otra desventaja es que cuando un usuario aplica una de las adaptaciones propuesta por Mash Maker directamente desde los botones de la barra, esta adaptación solo puede agregar información a la página web, las adaptaciones son “non-side-effecting”. Esto significa que no pueden realizar externamente acciones visibles. Por lo tanto si una adaptación desea realizar estas acciones, debe hacerlo insertando un widget que el usuario pueda utilizar para llevar a cabo tal acción. Esto no sucedería con las adaptaciones realizadas en forma de scripts, ya que el código puede contener todo lo necesario para realizar la tarea.

Entonces Mash Maker visualiza datos insertados en una página mediante la utilización de “widgets de visualización”. Estos widgets pueden ser desde unos simple widgets estáticos tales como imágenes y texto, hasta widgets interactivos como por ejemplo mapas. El problema es que para hacerlo el usuario debe escribir/programar el widget creando una nueva página web donde expone el código Javascript con una función especial que utiliza Mash Maker para invocar el widget (mashmaker_widget). Luego Mash Maker, reproduce el widget insertando esta página, en la página web que el usuario desea adaptar, como un iframe [22]. Esta necesidad de crear una nueva página para poder utilizar las adaptaciones es fácilmente evitable utilizando adaptaciones insertadas directamente sobre la aplicación web en forma de scripts.

Por último las adaptaciones en Mash Maker tienden a ser gadgets como componentes reutilizables los cuales pueden ser fácilmente insertados en la aplicación web actual por los usuarios finales. Mientas que las adaptaciones desarrolladas con scripts pueden insertar cualquier fragmento HTML en los nodos hipermedia de la aplicación Web. Otro punto importante es que en Mash Maker no se puede “remover información”, ya que por definición un Mashup agrega información. En ocasiones una adaptación podría involucrar remover fragmentos de datos de los nodos para poder centrar al usuario en el contexto que tiene en mente mientas navega la aplicación web, el ejemplo más clásico es remover banners o propagandas.

Hemos analizado las desventajas que presenta la implementación de Mashups (como medio para adaptar aplicaciones web existentes) utilizando Mash Maker, básicamente podríamos decir que cualquier adaptaciones propuesta utilizando Mash Maker puede ser realizada a través de scripting pero no al revés. Pese a estas desventajas, Mash Maker, sigue siendo una poderosa herramienta para la utilización de Mashups y más aun, presenta ciertas ideas de diseño que se verán asociadas a conceptos que veremos en las secciones siguientes.

2.3 Concerns de aplicaciones y de navegación

De acuerdo a [2] definimos un concern de aplicación como un "asunto de consideración en un sistema de software". Un concern puede reflejar aspectos funcionales de una aplicación como subir videos a youtube, búsqueda de productos en Amazon, o áreas temáticas tales como economía o historia en una enciclopedia.

"Los concern pueden ser genéricos, cuando aparecen en un amplio número de aplicaciones (ej. soporte para un logueo seguro), específicos del dominio cuando solo se aplican a un conjunto de aplicaciones (ej. agregando post o comentarios en algunos software web 2.0), o incluso específicos de la aplicación cuando aparecen solo en un tipo particular de software (ej. funciones de ruteo en google maps)" [12]. Algunos tipos de concerns pueden ser definidos abstractamente durante el desarrollo de la aplicación (ej. administración de categorías en una enciclopedia) y instanciados por los usuarios mientras usan el software (ej. tratando con una categoría en concreto).

Un concern de navegación, es un concern de aplicación que afecta la navegación, por ejemplo se manifiesta en la estructura de navegación de la aplicación mediante contenidos y links, y por lo tanto impacta en la forma que en el usuario navega a través de la aplicación.

Fig. 2.4.1.1 – Concerns de navegación en aplicaciones Web.

En este trabajo nos enfatizamos en este concern en particular ya que las adaptaciones aquí propuestas están dirigidas a mejorar la experiencia de navegación del usuario.

Capitulo 3. Personalizando aplicaciones Web

Como vimos, esta tesis se centra en mejorar la experiencia de navegación del usuario que obtiene al utilizar los sitos Web. Aquí veremos, más en detalle, como esta experiencia es obtenida mediante el uso de adaptaciones. En particular examinaremos dos enfoques posibles, uno en el que estas "adaptaciones" son implementadas en el mismo diseño de las aplicaciones Web, agregando el dinamismo necesario para generar esa nueva funcionalidad que se requiere para mejorar la interacción con el usuario. Luego veremos otra, y en nuestro caso la que nos acompañara a lo largo del desarrollo de esta tesis, en la que las adaptaciones son inyectadas dentro de la estructura de las aplicaciones Web existentes. Para cada enfoque, veremos algunas tecnologías y técnicas posibles, y analizaremos en un breve detalle las ventajas y desventajas de cada una.

3.1 Adaptaciones basadas en el servidor

Presentar una buena estructura de navegación en aplicaciones Web es un gran desafío para los desarrolladores, ya que muchas veces depende de las características del usuario o sus preocupaciones al momento de acceder a la aplicación, lo cual es muy difícil de determinar.

La idea de adaptar una aplicación web existente puede ser considerada como un problema de reingeniería de la aplicación. Desde este punto de vista, un enfoque posible es agregar los conceptos necesarios produciendo un nuevo modelo y generando una aplicación completamente nueva donde la idea presentar una estructura de navegación para mejorar la experiencia del usuario se encuentre presente.

Muchos enfoques de diseño han evolucionado junto con la Web, proveyendo herramientas concretas y conceptuales, con el fin de mejorar la navegación en aplicaciones web. Por ejemplo en OOHDM (Object-Oriented Hypermedia Desing Method) [36], una aplicación es comprendida como una vista de navegación sobre un modelo conceptual. Esto refleja una de las mayores innovaciones de OOHDM, el cual reconoce que los objetos (ítems) que el usuario navega no son los objetos conceptuales (como podría ser un "articulo"), sino otra clase de objetos que son construidos a partir de uno o más objetos conceptuales. Por lo tanto los atributos de un nodo (por ejemplo el titulo o la imagen de un artículo) son definidos como vistas de objetos orientadas a las clases conceptuales.

En OHDM, existe un conjunto predefinido de tipos de clases de navegación: links, nodes, anchors y estructuras de acceso. La semánticas de nodos, links y anchors son típicas en aplicaciones hipermedia. Estructuras de acceso, tales como índices, representan una posible forma de comenzar una navegación en una aplicación Web.

De la misma forma que los objetos de navegación, los links reflejan relaciones conceptuales que tienden a ser exploradas por los usuarios finales. Diferentes aplicaciones (incluso en el mismo dominio) pueden contener diferentes vinculaciones en las topologías de acuerdo a los

perfiles de usuario. Por ejemplo, un website de arquitectura podría incluir una categoría de información sobre construcciones para expertos en el tema, y no incluir esta información cuando el usuario presente es un usuario casual.

Una vez que las clases de navegación han sido decididas, es necesario estructurar el espacio de navegación que estará disponible para el usuario final. En OOHDM, esta estructura se define agrupando objetos de navegación en conjuntos llamados contextos. Cada definición de contexto incluye: los elementos que contendrá; la especificación de su estructura interna de navegación; un punto de acceso; restricciones de acceso en términos de la clase del usuario; y estructuras de acceso asociadas.

Como puede observarse, OOHDM, usa el concepto de contexto de navegación para enriquecer nodos hipermedia cuando son accedidos en un conjunto particular (por ejemplo la lista de mejores productos).

Otro claro ejemplo de tecnologías que han evolucionado para mejorar la navegación de aplicaciones Web es WebML (Web Modeling Language). En [46], se muestra como un website puede ser definido utilizando WebML mediante la especificación de perspectivas ortogonales. Donde una parte de estas perspectivas se encargan de administrar la navegación de la aplicación.

En el modelo de navegación (Navigation Model), se expresa como las paginas y los contenidos son relacionados para formar el hipertexto (ej. la pagina de un artista a la página de inicio del website). Luego en el modelo de personalización (Personalization Model) los usuarios y grupos de usuarios son expresamente modelados en el esquema de la estructura en forma de entidades predefinidas llamadas User and Goup, un grupo describe un conjunto de usuarios con características en común, por ejemplos aquellos usuarios que ingresan a un sistema como invitados. Las especificaciones de estas entidades son usadas para almacenar contenidos relacionados a un grupo o a un individuo, como por ejemplo recomendaciones de compras, lista de favoritos y recursos para personalización grafica.

WebML también permite definir reglas de negocio (Business rules) sobre el mismo modelo, que son un triple evento-condición-acción, el cual especifica el evento a ser monitoreado (ej. una compra en un e-commerce), la precondición a ser chequeada antes que el evento ocurra (ej. verificar que el usuario este logueado), y la acción a ser realizada cuando la condición es verdadera (ej. agregar el producto al carrito de compras). Esto permite por ejemplo generar nueva información relacionada con el usuario (ej. historia de compra) o actualizar el contenido del website (ej. insertando nuevas ofertas basándose en la historia de compra del usuario y sus preferencias).

La mayoría de estas las técnicas y tecnologías se basan en el principio de “separation of concern” [10], en el cual una aplicación web es dividida en diferentes componentes que realizan diferentes tareas. Por ejemplo WebML divide la aplicación en modelos que se encargan de diferentes aspectos, en nuestro caso nos interesa la forma en que WebML define la estructura de navegación de la aplicación y de qué manera realiza la adaptación en los nodos hipermedia ya sea usando el modelo de personalización o las reglas de negocio. Los

objetivos y beneficios de este enfoque incluyen reducir la complejidad, facilitar la cambios y actualizaciones, reutilización, simplificar la personalización.

Si bien estas técnicas permiten proveer a la aplicación web una mejor experiencia de navegación para el usuario final, el hecho de tener que reconstruir la aplicación la hace una alternativa demasiado costosa. Se necesita otra forma de adaptar aplicaciones web existentes de una manera menos intrusiva y más flexible, fácil y por supuesto no tan costosa.

3.2 Adaptaciones del lado del cliente, CSA (Client-Side Adaptation)

La Web 2.0 trajo consigo una nueva tendencia en la cual son los propios usuarios finales quienes personalizan las aplicaciones web a sus preferencias, modificando los contenidos y estilos a sus deseos y necesidades. Este tipo de intervención se conoce como "tuning" y ha estado en constante crecimiento en los últimos años. Los usuarios programan scripts que se ejecutan en el contexto del browser, a través de herramientas como Greasemonkey, el cual los aplica como plugins. Anteriormente observamos, como estos scripts pueden acceder a la estructura subyacente de la aplicación conocida como DOM y modificar, agregar o borrar nodos para adaptar los contenidos de la aplicación web a los deseos del usuario.

En este trabajo, se propone, usar esta técnica como medio para adaptar la estructura de navegación de la aplicación web, haciendo que los concerns de navegación estén más enfocados al contexto en el cual el usuario esta accediendo a la aplicación. Este contexto está determinado por la familia a la cual pertenece la aplicación y los conceptos que en ella se presentan (profundizaremos más sobre este tema en la siguiente sección). Por ahora, alcanza con decir, que una adaptación que se encuentra desarrollada para una familia, adaptara el contenido de la aplicación Web para mejorar la experiencia de navegación del usuario, en base a los conceptos de la familia. Por ejemplo, considérese una aplicación que proporciona información sobre tecnología, donde esta aplicación Web pertenece a la familia "software", seguramente los contenidos de los conceptos serán adaptados de manera diferente a si la aplicación perteneciera a la familia "hardware".

Alineando las propiedades de los nodos de hipermedia de la aplicación web, al contexto en el cual están siendo accedidos mejoramos la estructura de navegación de la misma, haciendo que los contenidos y links estén más enfocados en el contexto actual. De esta forma mejoramos la experiencia del usuario.

Para conseguir este objetivo, se plantea el desarrollo de un Framework, el cual al igual que Greasemonkey se ejecuta como un plugin de Firefox [34]. Pero a diferencia de Greasemonkey, el Framework proporciona un conjunto de abstracciones de alto nivel que permiten construir adaptaciones (desarrolladas a través de scripts) reutilizables para familias de aplicaciones Web. Protegiendo las partes estables del script de las partes frágiles, aquellas partes que acceden directamente a la estructura cruda del DOM y por lo tanto son las que más tienden a cambiar cuando las aplicaciones web se modifican o actualizan. Estas adaptaciones,

al igual que en Greasemonkey, se ejecutan automáticamente cada vez que una aplicación web termina de cargar. En [10], se presenta este tipo de adaptación como una adaptación estática (static adaptation). Estática en el sentido de que siempre se ejecuta cuando la aplicación termina de cargar, ya que las adaptaciones modifican dinámicamente la aplicación Web según el contexto del usuario. También presenta tres formas más posibles de realizar adaptaciones que se desarrollan del lado del cliente (client-side adaptation).

3.2.1 Adaptaciones CSN

La primera, son adaptaciones CSN (Concern-Sensitive navigation), las cuales son adaptaciones que mejoran la experiencia de navegación del usuario teniendo en cuenta las preocupaciones del mismo al momento de realizar la adaptación.

En [12], se presenta la idea de CSN como una herramienta práctica y conceptual para mejorar la estructura de navegación percibida por el usuario en aplicaciones Web. “CSN apunta a mejorar el acceso cognitivo y retorico a la información, lo que significa proveer a al usuario de la información necesitada en cada concern, de tal manera que es organizada y presentada en una forma más oportunista” [12].

Para entender esto pensemos en un usuario que navega en una aplicación web como puede ser amazon.com, donde el usuario visita miles de productos con diferentes concerns en mente.

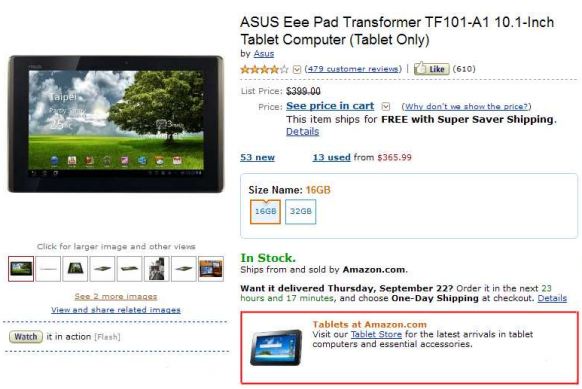

Fig. 3.2.1.1 - a

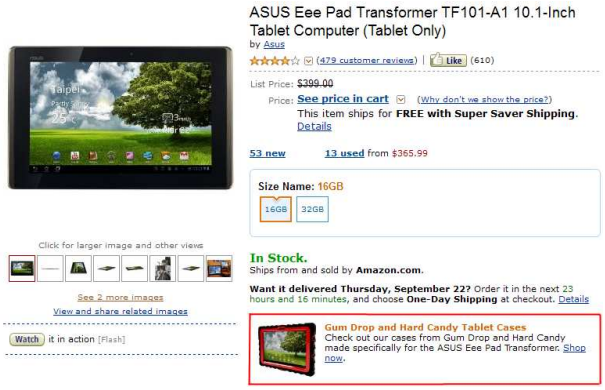

Fig. 3.2.1.1 - b

En Fig. 3.2.1.1 puede verse la página de un producto, en este caso una tablet en amazon.com. En la Fig. 3.2.1.1 - a aparece un banner que contiene información para poder visitar mas tablets en el sitio, ya que el producto fue alcanzado desde una oferta de tablets. En la Fig. 3.2.1.1 - b el mismo producto contiene información sobre artículos específicos para esa tablet de Asus, debido a que el producto fue buscado en la sección de tablet/asus de Amazon. Puede

observarse, como el contenido de la página del producto fue mejorada teniendo en cuenta el concern dominante en el momento de ser accedido.

Alineando las propiedades del objeto de navegación (en este caso el tablet) al concern en el cual está siendo accedido (como producto de una oferta o como un producto desde una lista específica), mejoramos la estructura de navegación de la aplicación, haciendo los contenidos y links más centrados al concern actual que el usuario esta navegando. En este caso agrega nueva información en base a la forma en que el usuario llego al producto.

"Los usuarios navegan utilizando aplicaciones Web para realizar una tarea específica; métodos de diseño modernos ya han presentado enfoques y notaciones para mapear descripciones de tareas (ej. usando casos de uso) en modelos conceptuales y de navegación (ej. OOHDM). CSN complementa estas ideas con una estrategia para enriquecer los objetos de navegación con información específica al concern que el usuario está atravesando" [12].

Si bien existen tecnologías y técnicas modernas que dedican una parte de su diseño a mejorar la navegación, la mayoría se basa en las preferencias de los usuarios o grupos predefinidos en el modelo para adaptar la aplicación Web (ej. WebML). Por su parte, CSN puede ser visto como un tipo de adaptación de software web, el cual no necesita administrar perfiles de usuarios. Esto es un punto importante, ya que las adaptaciones que en este trabajo de desarrollan, son creadas por los propios usuario finales, quienes probablemente no se encuentren contemplados dentro de los perfiles de usuario de la aplicación web.

3.2.2 Adaptaciones basadas en la historia de navegación

Estas adaptaciones registran la historia de navegación, que el usuario construye a medida que navega a través de diferentes aplicaciones web. De esta forma, se puede enriquecer los nodos hipermedia, de la aplicación actual, con la información recolectada de la navegación reciente del usuario. Por ejemplo, un usuario busca en Google cierta información, esto despliega una lista de resultados posibles. Seguramente el usuario elegirá uno de los posibles resultados basándose en su criterio, entonces hará clic en el link asociado al resultado y se dirigirá a la pagina destino. Esta aplicación destino, podría ser enriquecida, con un pequeño panel o menú que muestre los resultados recientemente obtenidos en Google, ya que seguramente la información que se encuentra en la página actual está relacionada con la de la búsqueda reciente.

3.2.3 Adaptaciones basadas en un una tarea

Son adaptaciones, que tienen en cuenta la actividad actual del usuario, para mejorar la aplicación web. El mejor ejemplo, para ver cómo funciona este tipo de adaptaciones, es cuando el usuario se encuentra utilizando varias aplicaciones para llevar a cabo una tarea.

Anteriormente, analizamos Mash Maker, como una herramienta para la creación y aplicación de Masups en aplicaciones web existentes y vimos las desventajas que presenta con respecto a las adaptaciones que se planean construir aquí. Sin embargo, Mash Maker presenta técnicas en la aplicación de Mashups que se alinean con el concepto de CSA (Client-side adaptation).

A medida que el usuario navega en la web, a través de la las diferentes aplicaciones, Mash Maker muestra botones (ubicados en su barra) que representan las diferentes adaptaciones que Mash Maker piensa que el usuario podría querer aplicar en la aplicación web actual.

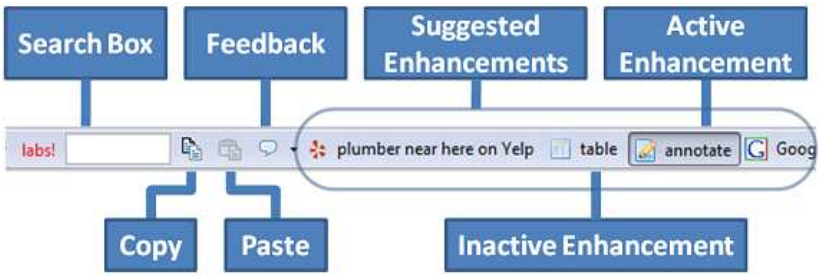

Fig. 3.2.3.1 – Barra de Mash Maker.

Mash Maker realiza estas sugerencias, de posibles adaptaciones, basándose en la información que el usuario se encuentra observando en la aplicación web, que esta navegando actualmente, y correlacionando el comportamiento del usuario con el comportamiento de otros usuarios que anteriormente aplicaron mashups en esa misma aplicación web.

Continuemos el ejemplo de departamentos que vimos antes. Un usuario busca departamentos disponibles en Craigslist, Mash Maker se da cuenta que los ítems de la pagina contienen direcciones y sugiere un Mashup (“Google maps”), que se muestra como un botón en la barra de Mash Maker. Al mismo tiempo, el usuario también está interesado en apartamentos que tengan buenos restaurantes cerca, entonces abre otra pestaña y busca restaurantes en la página de Yelp [52]. Mash Maker se da cuenta de que el usuario está interesado en restaurantes y actualiza la barra de Mashups para la página de Craigslist con un botón que sugiere un Mashup, para añadir una lista de restaurantes en el área.

Claramente, puede verse, como Mash Maker, hace uso del contexto para proveer una personalización de la aplicación web, que se acerque más a las necesidades del usuario, con el fin de mejorar la experiencia de navegación del mismo. En este caso, el contexto, esta dado por la actividad que el usuario se encuentra realizando, actividad, cuya información puede estar repartida en varias aplicaciones. Básicamente, esta es, la idea que plantea este tipo de adaptaciones, es decir adaptaciones que se encuentran pendiente de la actividad del usuario.

Capitulo 4. Trabajando con familias de aplicaciones

Utilizar abstracciones para reducir la complejidad y dependencia de un problema es una técnica muy utilizada en el desarrollo de software.

En POO (Program Oriented Object), se puede utilizar el concepto de familia mediante la implementación de generalizaciones y jerarquías, y gracias a la propiedad de herencia que proporciona. Para POO una clase representa un concepto, objeto o entidad a ser modelada por la aplicación. Esta contiene atributos que determinan el estado y características de dicha clase (ej. titulo, precio, cantidad, etc. de un CD en un sitio de música). Cuando varias clases presentan los mismos atributos y casi la misma funcionalidad pueden ser abstraídas mediante la creación de una tercera clase que reúne todas las características en común de las clases. Esta estructura se organiza de forma jerárquica, de esta forma las subclases heredan de la superclase las propiedades y funcionalidades (métodos). Por ejemplo, en un e-commerce que vende productos, estos productos podrían ser muchas cosas con lo cual podría decirse que "Producto" es una familia y que sus miembros podrían ser libros, muebles, autos, etc.

La Web, se encuentra compuesta de millones de aplicaciones que proporcionan servicios, información, entretenimiento, comunicación, etc. Si bien, cada una de estas aplicaciones, seguramente pertenece a dominios diferentes y presentan un modelo distinto, se pueden encontrar asociaciones de acuerdo a ciertas características o parámetros de manera tal que podemos decir que una aplicación pertenece a una familia. De esta forma, podríamos establecer que una aplicación pertenece a una familia porque presenta la misma estructura (ej. la forma en se presenta la información) o provee los mismos servicios (ej. subida de archivos, compartir imágenes, etc.) o tal vez podríamos decir que pertenece a una familia solo porque contiene información similar (ej. paginas sobre tecnología).

Como puede verse, el término familia, es demasiado amplio como para poder agrupar aplicaciones de una manera coherente a los fines de esta tesis. Es por eso que debemos acotar las propiedades, que hacen que una aplicación, pertenezca a una familia particular. En [19], se define una familia de aplicaciones, como aquellas aplicaciones que usualmente enfrentan conceptos similares (ej. un post en un blog) y por lo tanto se ven involucrados los mismos concerns de navegación. Por ejemplo, aplicaciones como Facebook[21], LinkedIn[32] y Google+[24], presentan estructuras similares y proveen un conjunto común de funcionalidad.

Actualmente existen tecnologías, que permiten construir adaptaciones para ser aplicadas a un conjunto de aplicaciones (que podrían considerarse parte de una familia). Greasemonkey [25] es una de ellas, permite a los desarrolladores construir una adaptación y especificar el conjunto de aplicaciones a las cuales la adaptación está destinada. Por ejemplo, el script BookInfoLine [8], permite comparar los precios en varios sitios de venta de libros (ej. Barnesandnoble, Half, Biblio, Ebooks, entre otros mas), mientras estas buscando en tu sitio preferido un libro en particular.

Fig. 4.1.1 – BookInfoLine

Como se ve, en la Fig. 4.1.1, el script utilizando Greasemonkey, agrega un panel flotante con una lista de precios de diferentes sitios de venta de libros. De forma que, si el usuario, encuentra un precio que se ajuste más a su bolcillo, que el precio que ofrece la pagina, puede clickear directamente sobre el panel donde aparece el nombre de la pagina que ofrece un mejor precio y lo redirige directamente a ese destino.

El usuario que desarrolla el script lo construye pensando que todas estas aplicaciones Web presentan una estructura similar, ya que seguramente al acceder a un libro particular se muestre una foto, el titulo, información sobre el autor, links recomendados a otros libros, precio, opciones de compra, etc. De esta forma, todas estas propiedades, que siempre se encuentran presente, cuando un usuario accede a un libro, en las diferentes aplicaciones web de venta de libros, se convierten en propiedades que hacen que una aplicación web sea admisible para pertenecer al conjunto de aplicaciones, en las cuales el script que el usuario creo, podría funcionar.

Si bien Greasemonkey, le permite al desarrollador, definir el conjunto de aplicaciones web (mediante la especificación de la URL) en las cuales el script puede aplicarse, tiene que rescribir este script para cada aplicación, pues el mismo accede directamente a la estructura de la pagina (DOM), para poder obtener información perteneciente a las diferentes propiedades y realizar las adaptaciones sobre los nodos hipermedia. Este hecho, de reescribir el script, para cada una de las aplicaciones resulta en una tarea tediosa y una gran pérdida de tiempo.

Aquí se propone el objetivo de agrupar aplicaciones en familias, no como una forma de especificar, solamente aquellas aplicaciones donde la adaptación funciona, sino como una forma de construir adaptaciones para la familia y no para la aplicación particular, de manera

que una adaptación, que se puede aplicar a una aplicación de la familia, podría funcionar para cualquier otra aplicación dentro de la misma familia.

Entonces, podríamos desarrollar una adaptación para una familia, continuando con el ejemplo, podría ser para la familia de "Sitios de venta de libros", y esta adaptación funcionaria en cualquier aplicación que se agregara a la familia y que por su puesto presentara las mismas propiedades. Pero como esto es posible?. Si bien, cada aplicación, dentro de la familia comparte las mismas propiedades y la funcionalidad que provee es similar, no es cierto que la implementación de la estructura sobre la que se apoya la aplicación sea igual, ni siquiera parecida. Entonces, como podríamos decir que la adaptación funciona en cualquier miembro de la familia. Para esto, tenemos que abstraernos de esta implementación y pensar en términos de los conceptos que comparten, conceptos que seguramente aparecen en cada miembro de la familia.

En primer lugar, debemos pensar en los objetos de navegación (ej. nodos de hipermedia), que las aplicaciones web utilizan para mostrar información y las adaptaciones tienen como medio para enriquecer la aplicación. Estos mismos nodos de datos, representan algo más que solo contener el conjunto de propiedades, que hace que una aplicación web pueda pertenecer a la familia. Cuando un usuario desarrolla una adaptación, ve estos nodos de datos de una forma abstracta y se refiere a ellos como entidades, que se encuentran conectadas con la información o funcionalidad de la página o incluso, entidades con las que él mismo se encuentra relacionado. En la página de venta de libros, seguramente el desarrollador vea estos nodos de datos y las propiedades que presentan (ej. foto, titulo, autor, etc.) y hable de todas ellas como un conjunto, el cual representa una entidad o más específicamente un concepto, que el usuario ve a través de la aplicación web. Entonces, el usuario podría desarrollar adaptaciones con el fin de adaptar conceptos presentes en las aplicaciones Web. Para el caso de la página de venta de libros, el usuario construirá una adaptación pensando que adapta un libro que tiene un título, una foto y un autor, y no el nodo de hipermedia que contiene las propiedades del libro, y lo representa en la aplicación web. Esto nos lleva a la conclusión, de que nuestro famework, deberá presentar un nivel de abstracción que permita al usuario definir estas abstracciones, que representan los conceptos presentes en las aplicaciones web y que permiten, construir una adaptación que es aplicable a cualquier miembro de la familia.

En [5], se presenta un framework llamado SWAT (Stateful Web Augmentation Toolkit), el cual permite a los desarrolladores identificar "registros" de datos en Websites, que proveen una base de datos. Los registros, son nodos de datos (ej. nodos de hipermedia), que se corresponden con las filas de las tablas en la base de datos de la aplicación. Mediante la utilización de este framework, los desarrolladores pueden agregar funcionalidad adicional a estos nodos de datos. Entonces, para SWAT, un registro de datos es cualquier nodo que represente los datos almacenados en la base de datos de la página, por ejemplo, en un sitio de venta de películas, seguramente se muestre una foto, el titulo y una descripción de la película que se corresponden con los datos almacenados en la base de datos.

Fig. 4.1.2 – Registros en SWAT

Lo más importante de SWAT, es que, introduce un grupo de abstracciones de alto nivel, que permiten la concepción y creación de interacciones con registros de datos, en Websites, que se apoyan en la utilización de bases de datos.

Básicamente, SWAT, consiste en tres módulos: "un modulo de perfil de sitio (Site Profile Module), que permite identificar los registros, un modulo de ajuste (Tweak Module), que define el comportamiento y la interface de una adaptación (widget) interactiva, y un modulo de almacenamiento (Storage Module), que persiste la adaptación a lo largo de diversas paginas y sesiones" [5].

Prestemos atención al primer modulo de SWAT, que es el que permite identificar los registros, que utiliza el framework en las páginas de un Website. Desde el punto de vista, de utilizar adaptaciones en familias de aplicaciones, esto es sumamente importante, ya que, los desarrolladores no diseñan adaptaciones para un nodo de datos, modelado en la aplicación, a través del HTML, sino que diseñan adaptaciones a ser aplicadas en "registros".

Fig. 4.1.3 – Adaptación en SWAT

En la Fig. 4.1.3, puede verse, el registro de un libro en Amazon y como una adaptación utilizando el framework (SWAT), agrega nueva funcionalidad a este registro. En este caso, agrega un campo que permite al usuario asociar un nombre con el libro (registro), que proviene de su lista de contactos.

La utilización de los registros en SWAT es similar a nuestra idea de "conceptos", solo que en vez de limitarnos a decir que todo aquello que el desarrollador tiene como objetivo para adaptar (ej. un libro, una película, un articulo, etc.), en una aplicación web son registros, dejamos que sea el mismo desarrollador quien defina los conceptos de la aplicación, en el caso de la pagina de venta de películas, el concepto seguramente terminaría siendo "película", de esta forma, los desarrolladores crearían adaptaciones para ser aplicadas en conceptos (en este caso seria para ser aplicadas en películas) y no directamente sobre los nodos hipermedia que los representan.

Es importante notar también, que SWAT, se encuentra un poco enfocado hacia la idea de familias de aplicaciones, pues para SWAT toda aquella aplicación web, cuyos nodos hipermedia, se pueden corresponder con los datos almacenados en las filas de las tablas de una base de datos y por lo tanto, pueden ser representados mediante un concepto abstracto, como es "registro", es una aplicación válida para formar parte de las aplicaciones de SWAT.

Sin embargo SWAT, no realiza un uso eficiente, o al menos desde nuestro punto de vista, en cuanto a familias de aplicaciones. En primer lugar, SWAT no permite definir explícitamente familias de aplicaciones, sino que, toda aplicación Web que cumpla las características necesarias impuestas por SWAT, pasara a formar parte de aquellas aplicaciones Web para las cuales los usuarios podrán desarrollar adaptaciones (Fig. 4.1.4).

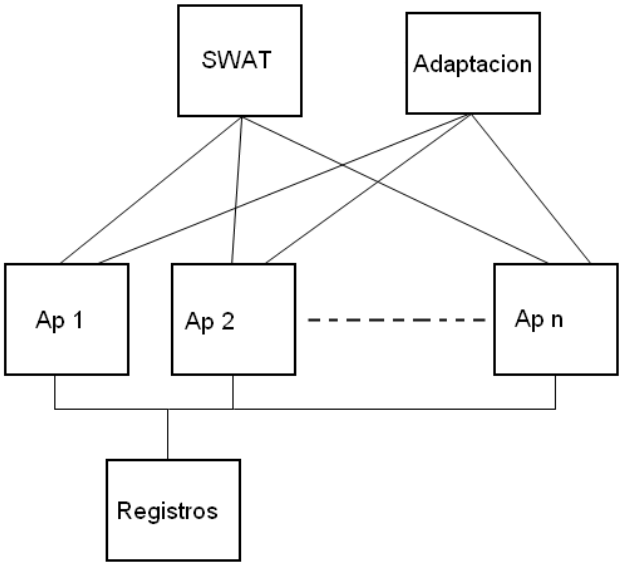

Fig. 4.1.4 - SWAT

Esto es una desventaja, en el sentido de poder brindar con más personalización a las aplicaciones web. En gran parte, esto se debe, a que una adaptación creada por un usuario se encuentra diseñada para mejorar la aplicación Web, de acuerdo al concepto que la aplicación representa para el usuario. Por ejemplo, supongamos que el usuario desarrolla adaptaciones para proveer de más funcionalidad a una aplicación Web, que pertenece a una página de noticias, donde la información presente puede ser interpretada de diferentes maneras. El usuario podría crear adaptaciones, por ejemplo, desde un punto de vista económico de la información agregando links o funcionalidades relacionadas con economía. Luego, el usuario vuelve a la misma aplicación web, pero viendo la información desde un punto de vista tecnológico y por ende, crea adaptaciones para mejorar los nodos de datos, teniendo en cuenta este nuevo interés. Si bien, un usuario, podría utilizar una aplicación Web teniendo en cuenta ambos intereses, lo más común, es que un usuario acceda a las aplicaciones con un objetivo o tarea en mente, por lo tanto, existe un conflicto en el objetivo de lo que se intenta adaptar. Esto se ve reflejado, en la experiencia de navegación del usuario, la cual se ve empobrecida, pues simplemente cuesta más centrarse en una tarea específica, si se muestra demasiada información, más aun, si esta misma información no está relacionada con la tarea que se desea desarrollar.

La forma más obvia, de solucionar este conflicto de intereses, es quitar todo aquello que me distrae o desvía la atención de mi objetivo. Una manera posible y seguramente la más sencilla de lograr esto, sería deshabilitando las adaptaciones que se encuentran fuera de contexto, de acuerdo al interés del usuario. Esto es similar a cuando se crean adaptaciones para remover banners o espacios de publicidad en las aplicaciones.

Dado que SWAT, agrupa todas las aplicaciones dentro de una gran y única familia, una de las posibles opciones que nos queda, es deshabilitar las adaptaciones, relacionadas con la aplicación web en conflicto, una por una. Si bien, seguramente esto signifique que para deshabilitar una adaptación, solo haya que tocar una pocas líneas de código, el hecho de tener que hacerlo uno por uno es una tarea un tanto tediosa, mas si se piensa, que una aplicación web, podría llegar a tener decenas o cientos de adaptaciones mezcladas. Además, al tener que hacerlo muchas veces y muy seguido, es más probable que se produzcan errores por parte del usuario, ya sea un error simple, como deshabilitar adaptaciones que no debería, hasta errores que harían que el modelo y por tanto el framework, no funcione como se supone o incluso deje de hacerlo.

Una forma rápida y más segura, de solucionar este problema, sería que las adaptaciones desarrolladas se encuentren asociadas a familias de aplicaciones y no directamente a las aplicaciones web. De esta forma, si una o varias adaptaciones, se encuentran en conflicto, ya que la intención con la que mejoran la aplicación es diferente, no es necesario buscar y deshabilitar una por una las adaptaciones que en cuestión generan este conflicto, sino que simplemente, se podría deshabilitar la familia a la que pertenecen las adaptaciones.

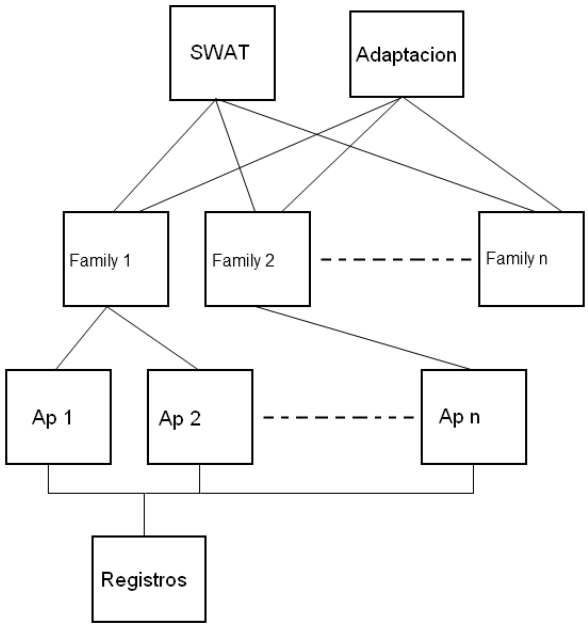

Fig. 4.1.5

En la Fig. 4.1.5 puede observarse una idea básica de como SWAT podría ser mejorado para soportar familias de aplicaciones.

Entonces, utilizar la idea de familia, para abstraernos de las aplicaciones, nos permite decir hasta un cierto punto que las adaptaciones son reutilizables en cualquier miembro de la familia. Pero, para que esto sea realmente aplicable, necesitamos definir una abstracción más que es el "concepto". Cuando una adaptación se construye para ser aplicada en una familia, esta va dirigida a un concepto perteneciente a la familia, concepto, que puede encontrarse presente en cualquier aplicación de la familia, por ejemplo, en el caso de la familia "Portal de noticias", un concepto valido posiblemente sea "noticia". Entonces, el usuario crea una adaptación para adaptar un concepto, que representa uno o un conjunto de nodos de datos de la aplicación web. Este concepto, posee un conjunto de propiedades, que hacen que se comporte como una entidad particular (ej. noticia).

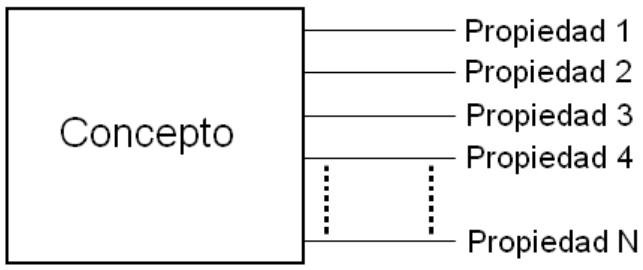

Fig. 4.1.6 – Propiedades de un concepto

Entonces este conjunto de propiedades, que pertenecen a un concepto, se corresponden con los atributos de los nodos de datos de una aplicación web. Estas propiedades representan contenidos multimedia, enlaces de los links u operaciones exhibidas por los nodos de la aplicación web. Es decir, cada una de estas propiedades, seguramente, terminara representado el título, el autor o el precio de, por ejemplo, un libro en Barnes&Noble.

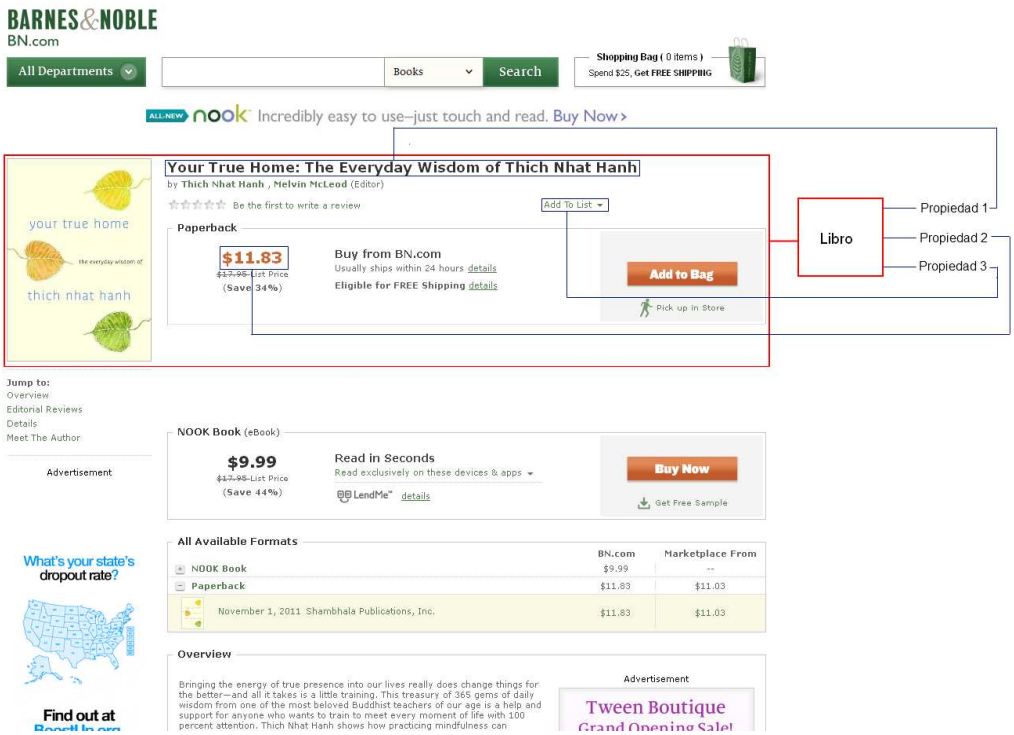

Fig. 4.1.7 – Propiedades del concepto libro

En la Fig. 4.1.7, podemos observar una forma posible de definir el concepto libro en Barnes&Noble, este concepto tiene 3 propiedades. La primera propiedad, es decir la número 1, representa el título del libro, la propiedad 2 representa una operación “Add To List” y la propiedad 3 representa el precio del libro.

Una vez que se define un concepto, para una familia aplicaciones junto con sus propiedades, estas siempre se encuentran presentes en los diferentes nodos de datos, que representan a los conceptos, en las distintas aplicaciones web. Donde, estas mismas aplicaciones, se encuentran dentro de una misma familia de aplicaciones. Por lo tanto, podríamos llamar a estas propiedades, como propiedades de núcleo o propiedades intrínsecas [11], en el sentido de que siempre se encuentran presentes, sin importar en qué aplicación web se encuentre el concepto.

Entonces, volviendo al ejemplo de la familia de “Portales de noticias”, se podría definir el concepto noticia junto con sus propiedades. Para simplificar, solo definiremos 2 propiedades para el concepto noticia, las cuales podrían ser el titulo de la noticia y su descripción.

Fig. 4.1.8 – Noticas en una aplicación web.

Por lo tanto, todo aquel nodo o nodos de datos, de una aplicación web, que presente estas propiedades, podrá ser considerado como un concepto valido de notica. Así mismo, todas aquellas aplicaciones, que presenten el concepto noticia, podrán formar parte de la misma familia de aplicaciones.

Fig. 4.1.9 – Portales de Noticias.

En la Fig. 20, se encuentran 2 aplicaciones, que pueden formar parte de la familia de “Portales de noticias”, ya que presentan el mismo concepto, con sus propiedades correspondientes. Resulta más obvio, al ver estas dos aplicaciones web, porque es necesario abstraer los nodos de datos, que representan el concepto, que se desea adaptar. Puede

observar, como una notica en la aplicación de la izquierda, presenta las propiedades del concepto de una forma un tanto distinta, a como lo hace la aplicación de la derecha. A simple vista el titulo de una noticia, en la aplicación de la izquierda, viene con una etiqueta y se en encuentra resaltada en azul, mientras que una noticia en la aplicación de la derecha, no contiene etiquetas y no esta resaltada. Si bien, estas pequeñas diferencias, parecen poco importantes, cuando un usuario diseña una adaptación, en algún punto, tendrá que tocar la estructura de bajo nivel que modela al concepto, por lo tanto, estas pequeñas diferencias, podrían terminar siendo grandes cambios en esta estructura, cambios que podrían hacer que la adaptación no se efectué adecuadamente. Veremos más sobre este tema y como el framework se encarga de ello en la siguiente sección.

Capitulo 5. Desarrollando adaptaciones para familias de aplicaciones

A lo largo de las secciones anteriores, fuimos introduciendo los conceptos e ideas, sobre las cuales se construyen las bases en las que se apoya nuestro Framework. Analizamos los diferentes enfoques, tecnologías, herramientas y técnicas que ayudan a aumentar la interacción de los usuarios finales para con los sitos de la Web, llevándolos a obtener una mejor experiencia, en particular nosotros apuntamos a mejorar, entre otras, la experiencia de navegación. Vimos, como es posible, aplicar todo esto a un nuevo enfoque, como son las "familias de aplicaciones Web". Ahora llega el momento de implementar todo, en una única herramienta, que cumpla con todas nuestras expectativas. Como mencionamos anteriormente, nosotros intentaremos cumplir este objetivo a través del uso de un Framework. Por lo tanto, en este capítulo, hablaremos del Framework en sí y de las diferentes partes que lo conforman. Comenzaremos con explicar el objetivo del Framework, aunque seguramente a este punto, se puede tener una idea de lo que se intenta alcanzar, para remarcarlo claramente, lo expresaremos de una manera formal.

5.1 Framework

Nuestro framework, tiene como uno de sus objetivos principales, proveer un ambiente de desarrollo, para la construcción de adaptaciones reutilizables en familias de aplicaciones Web. Al mismo tiempo, intenta introducir una serie de abstracciones de alto nivel, junto con un complemento de ideas de diseño, que permitan a los usuarios, con un poco de habilidades de scripting, construir estas adaptaciones y a los usuarios poco experimentados, en el tema de programación adaptaciones, utilizarlas con solo definir los conceptos e indicar las aplicaciones web, en las cuales las adaptaciones se aplicaran.

El framework, se encuentra desarrollado como una extensión de Firefox (en [33], se puede encontrar más información sobre la construcción de extensiones para este navegador). Las adaptaciones, se programan en el framework utilizando Javascript, ya que ha sido la tecnología más utilizada, en los últimos años, por los usuarios que navegan en la Web, para realizar tuning en aplicaciones Web.

Ahora que se han presentado, formalmente, los objetivos que nuestro framework intenta cumplir, hablaremos de las partes que hacen que el mismo funcione y que por su puesto, permiten que el framework alcance estos objetivos. En primer lugar, el framework, se encuentra conformado de dos grandes módulos, que trabajan en conjunto, para proveer a los usuarios de una serie de funciones y abstracciones, para el desarrollo y aplicación de adaptaciones en aplicaciones Web existentes.

El primer modulo, que llamaremos "Modulo de abstracción de datos", se encarga principalmente de reconocer y capturar los conceptos, en la aplicación web, que el usuario se encuentra navegando. El segundo modulo, que llamaremos "Modulo de adaptación de familias", se centra en proporcionar un conjunto de abstracciones y funciones para permitir al

desarrollador, crear las adaptaciones que se insertaran en las aplicaciones, a través de los conceptos capturados por el primer modulo.

A continuación veremos en detalle cada uno de estos módulos.

5.2 Modulo de abstracción de datos

En el capitulo anterior, vimos la importancia que aporta, para las adaptaciones a nivel de familias, el hecho de abstraer los nodos hipermedia, que las adaptaciones utilizan para insertar sus modificaciones o mejoras en las aplicaciones web, en conceptos. Ahora veremos detalladamente, cómo es que el framework o más específicamente este modulo, ayuda a que este mecanismo sea posible.

Primero que nada, tenemos que entender, que para lograr el nivel requerido de abstracción o al menos el suficiente, como para que las adaptaciones puedan utilizarse en cualquier aplicación web, dentro de una misma familia de aplicaciones, los módulos dentro del framework tienen que lidiar con dos tipos de conceptos diferentes. El primero de ellos, es un concepto puramente abstracto, del cual se encarga el otro modulo, es decir el "Modulo de adaptación de familias", por lo tanto, aquí no veremos cómo es que este concepto es definido. Por ahora, nos alcanza con saber, que el modulo de adaptación de familias, se encarga de definir un concepto abstracto junto con sus propiedades. Este sería el caso, de por ejemplo, el libro en la familia de venta de libros y sus propiedades (ej. titulo, autor, precio, etc.). Entonces este modulo, el de abstracción de datos, se dedica o mejor dicho define el otro tipo de concepto, conocido como concepto "concreto".

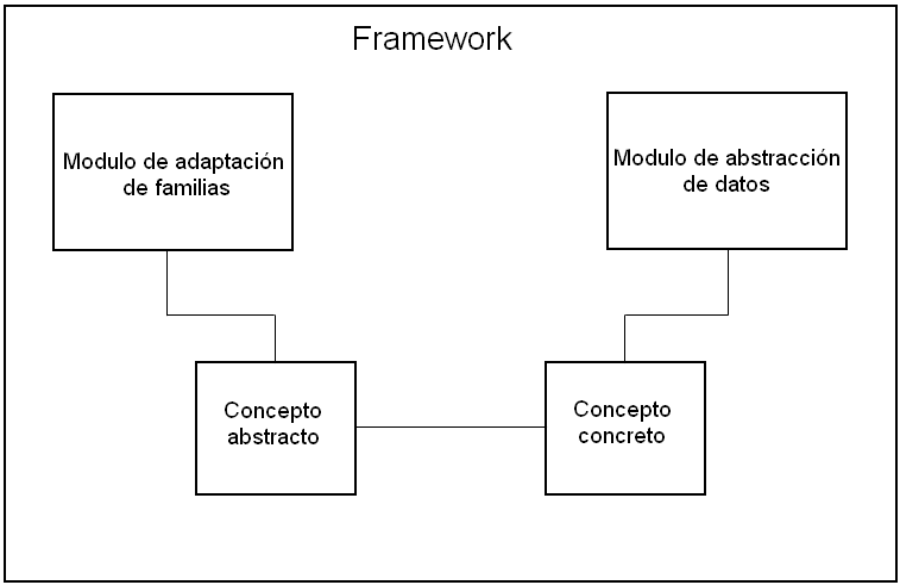

Fig. 5.2.1- Módulos

Seguramente, la primera pregunta que surja el ver la figura es, ¿porque es necesario que el framework maneje dos conceptos separados? Como puede verse, los conceptos se encuentran relacionados, ya que si bien, es cierto que cada modulo define el concepto, que le corresponde por separado, en conjunto ambos representan una sola entidad o idea, que el usuario, ve representada por la aplicación web, como es el ejemplo, de “libro” o “producto” en un e-commerce. Ambos módulos, cooperan entre sí, para poder proveer de la funcionalidad necesaria al framework. El modulo de abstracción de datos, utiliza los conceptos abstractos, definidos por el modulo de adaptación de familias, para poder construir los conceptos concretos, basados en las aplicaciones Web que el usuario navega y que por su puesto, se encuentran dentro de la misma familia, mientras que el modulo de adaptación de familias, utiliza los conceptos concretos, que captura el modulo de abstracción de datos, para poder insertar las adaptaciones creadas por los usuarios finales.

Entonces, este modulo, se encarga de definir los conceptos concretos, de las aplicaciones Web administrados por el framework. Por lo que, resulta primordial entender, que es un concepto concreto. Dado un concepto abstracto, que tiene un conjunto de propiedades, es necesario, identificar como el concepto y sus propiedades, son modeladas por la estructura de la aplicación web. Volvamos al ejemplo de la familia de venta de libros, donde se tenía un concepto que era libro, con tres propiedades definidas, las cuales podrían ser titulo, autor y precio. Entonces, el modulo de adaptación de familias, crea este concepto y sus propiedades abstractas, mientras que este modulo mapea este concepto y sus propiedades, directamente sobre la estructura de la aplicación web.

El acercamiento tradicional, para extraer datos desde la Web, es escribir programas especializados, llamados Wrappers, que identifican la información de interés sobre la aplicación web y la mapean a un formato más conveniente que puede ser procesado. En nuestro caso, esto es exactamente, de lo que se encargan nuestros conceptos concretos. El aspecto más desafiante al construir estos Wrappers, es que los mismos, deben ser capaces de reconocer la información de interés, entre muchas otras, no interesantes, piezas de texto sobre la aplicación Web (por ejemplo, markup tags, código embebido, consejos de navegación, etc.). “Esta información puede llegar a tener una estructura plana, pero también puede ser compleja y presentar una estructura jerárquica multinivel implícita, que a menudo no es rígida. Esto significa que la información puede llegar a exhibir variaciones estructurales que deben ser toleradas y manejadas de manera adecuada” [1].

El problema de generar Wrappers, para extracción de información en la Web, puede ser enunciado como sigue. “Dada una página Web S conteniendo un conjunto implícito de objetos, determinar un mapeo W que rellene un repositorio de datos R con los objetos en S. El mapeo W debe ser también capaz de reconocer y extraer información de cualquier otra pagina S′ similar a S” [1].

Esto significa, que nuestros conceptos abstractos serán mapeados por los conceptos concretos, donde, dada una página S, conteniendo un conjunto de instancias relacionadas al concepto abstracto, a través de nuestro concepto concreto, este mismo se encargara de rellenar el repositorio R, que es la colección de instancias para ese concepto abstracto. En

cuanto a ser capaz de reconocer y extraer información, de cualquier otra página similar, esto es uno de los objetivos, que principalmente, se intenta al agrupar las aplicaciones web en familias. Donde, estas "paginas similares", se relacionan por medio de los conceptos que comparten.

El desarrollo de Wrappers, presenta deficiencias que son bien conocidas, principalmente debidas a la dificultad que presentan al escribirlos y mantenerlos. Existe una gran cantidad de trabajos recientes, que presentan una mejor forma de encarar el problema, del desarrollo de wrappers para extracción de datos en la Web [6], [47], [39], entre otros.

Comencemos con como el concepto es capturado y mapeado sobre la aplicación. Piense en el libro, ¿qué es lo que representa para el usuario?. Posiblemente, a los ojos del usuario, indique un conjunto de características o propiedades, que se encuentran agrupadas sobre la aplicación Web. Pero es necesario definir concretamente, que entidad contiene estas propiedades o más específicamente, que parte de la estructura de la aplicación concuerda con el concepto. En la Fig. 4.1.7, se muestra un libro en la aplicación Web Barnes&Nobles. Puede verse, como todo lo comprendido dentro del rectángulo rojo, representa para el usuario lo que es el libro. Pero como hacer que el framework comprenda esto?. El usuario, no puede solo decirle al framework, "eso que está dentro del rectángulo es un libro", tiene que haber una forma más específica de indicarlo.

Esta forma, es utilizando la estructura subyacente sobre la que se apoya la aplicación Web, es decir el DOM. Entonces, para lograr que el framework, comprenda esta idea abstracta, bastaría con indicar que nodo [16] del DOM contiene o representa al concepto libro. A simple vista, existen dos formas directas o al menos claras de hacerlo, la primera, seria que sea el mismo usuario quien cree y setee por si mismo los atributos del concepto concreto, indicando por supuesto, el nodo que contiene al concepto. Sin embargo, esta alternativa requiere que el usuario tenga algo de experiencia en el desarrollo de aplicaciones Web y parte de la idea de este modulo, es hacer que el usuario final con poca o nada de experiencia sobre este tema, pueda definir los conceptos concretos de las aplicaciones Web. La segunda alternativa y la que utilizaremos aquí, es que el modulo cree estos conceptos concretos automáticamente, con un poco de ayuda del usuario. La intervención del usuario, recaería solo en indicarle al modulo, en que parte de la estructura de la aplicación, se encuentra el concepto.

Es importante notar, que para fuentes de datos estructuradas, tales como fuentes de bases de datos o documentos XML, el proceso de extracción de información podría llegar a ser conducido automáticamente, siguiendo el esquema de las tablas o los tag's XML. Sin embargo, este no es el caso para las no estructuradas o semi-estructuradas fuentes de información, tales como los documentos HTML o archivos de texto, debido a que la información no estructurada o semi-estructurada, es presentada sin propiedades de auto-descripción. Por lo tanto, uno de los objetivos del modulo, es realizar este proceso de extracción con una mínima intervención por parte del usuario.

La mayor parte de la información, que se encuentra actualmente en la web, reside en bases de datos estructuradas que se encuentran debajo de los mismos sitios Web (DOM). Sin embargo,

como los contemporáneos Web browsers, se manejan en términos de instrucciones de representación (HTML), el típico sitio Web debe transformar está estructurada información, que yace debajo del mismo en HTML, antes de servir esta información al browser. Por lo tanto, cuando la información alcanza el browser del usuario, ha perdido la mayor parte, sino toda, de su estructura original. Obviamente, esto hace que la información pueda ser digerida por los humanos, pero no puede seguir siendo utilizable para el procesamiento a nivel maquina.

Con el fin de lograr extraer esta información estructurada, que se encuentra enterrada dentro del código HTML y que puede ser utilizada por el framework, el modulo deja que sea el usuario quien indique, que parte de la estructura representa el concepto deseado. Para lograr esto, el modulo brinda una herramienta que da a los usuarios la habilidad de enseñar al modulo, como es que un concepto es definido sobre el HTML de la misma. Al observar estas demostraciones, el modulo aprende a extraer la información, que ha de necesitar luego para capturar un concepto, dentro de una aplicación Web, que pertenece a una familia de aplicaciones Web.

Entonces esta herramienta, permite que sean los usuarios finales, quienes definan el concepto sobre la aplicación Web. Ahora veremos, un poco más en detalle, cómo es que la herramienta hace esto posible. A medida que, el usuario navega sobre el código HTML de la aplicación Web, la herramienta resalta o remarca aquellos nodos del DOM, que representan la estructura modelada por el HTML, de esta forma, la herramienta da a entender al usuario que esa porción de la estructura, desde los ojos del usuario el HTML y desde el modulo el DOM, será considerada como el concepto que se intenta capturar. Para entender mejor como es que funciona esta captura, veremos un breve detalle sobre lo que representa el DOM y su estructura.

El Document Object Model o DOM, es la interface que te permite, mediante métodos de programación, acceder y manipular los contenidos de una página Web (o documento). Provee, una estructurada representación orientada a objetos de los elementos individuales y contenidos de una página, junto con un conjunto de métodos para recuperar y modificar las propiedades de tales objetos. Así mismo, el HTML DOM define una forma estándar para acceder y manipular documentos HTML.

Cuando un browser carga una página web, crea una representación jerárquica de los contenidos de la página, la cual resulta en una copia casi exacta de su estructura HTML. Esta nueva estructura conlleva a una organización de los nodos en forma de árbol, cada uno, representando un elemento, atributo, contenido o algún otro objeto modelado por la aplicación a través del HTML.

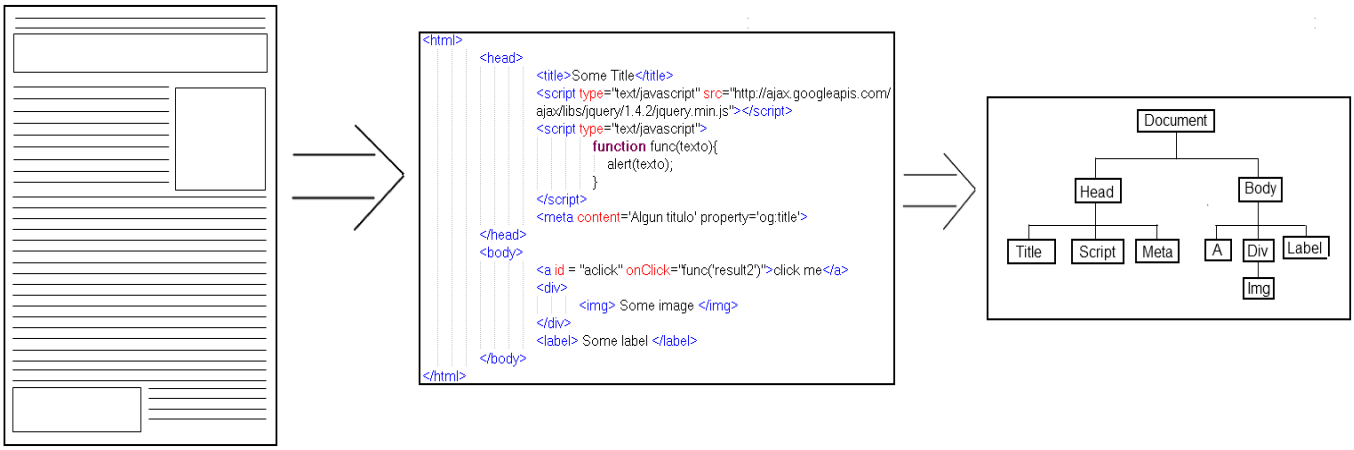

Fig. 5.2.2 – Pagina Web –> HTML –> DOM

Entonces, la herramienta, trabaja directamente sobre los nodos del árbol del DOM, para poder capturar el concepto que se encuentra en la aplicación Web. En particular, la herramienta le saca provecho, a la propiedad de jerarquía que proporciona la estructura de árbol. Cuando se manipula o se trabaja, con un nodo del árbol del DOM, al mismo tiempo se tiene todos aquellos nodos, que se encuentran relacionados con el primer nodo (conocido como nodo "raíz"), debido a la propiedad de jerarquía. Es decir, suponga que el concepto que se desea obtener, en la aplicación Web, se encuentra representado por el nodo "Body", que se puede apreciar en la Fig. 5.2.2, al obtener este nodo del árbol del DOM también es posible acceder y manipular todos aquellos nodos, que se desprenden a partir de este, como son "a", "div", "label" y "img". De esta forma, no es necesario capturar, cada una de las propiedades que están dentro del nodo, como por ejemplo, el titulo o la imagen de un libro, sino que basta con obtener el nodo que las contiene, al menos para definir el concepto concreto en sí.

La idea de utilizar la estructura proporcionada por el DOM, para automatizar la extracción de datos, a través de la web, no es nada nueva, muchos trabajos anteriores ya lo utilizan y han diseñado diversas técnicas para su aplicación [44], [14], [45], por nombrar algunas. Sin embargo, estos enfoques mayormente trabajan fuera del alcance del Web browser y no convierten la información extraída de las aplicaciones Web, en información que sea utilizable por los usuarios finales.

Por otro lado, existen trabajos, que se comienzan a aproximar a nuestra técnica de extracción de datos, uno de ellos es [5], el cual estuvimos analizando anteriormente. No obstante, sigue habiendo una distancia, entre la implementación de las técnicas a la hora de extraer datos. La principal diferencia y que permite una gran flexibilidad, por parte del framework hacia el usuario, es la forma en que se realiza la captura del nodo o mejor dicho del concepto.

Obviamente, en SWAT, también se utilizan los nodos del DOM, para poder construir los registros que utiliza el framework (SWAT), en los cuales, se insertan las adaptaciones (llamadas "tweaks") que desarrollan los usuarios. Pero la forma en que estos registros son extraídos de las aplicaciones Web es diferente. Esta principal diferencia, recae en que SWAT,

no proporciona una herramienta intuitiva que permita al usuario definir o capturar un registro, sobre la aplicación web de una manera simple. El usuario necesita examinar la estructura de la aplicación web, a bajo nivel, y proporcionar ciertos datos sobre el registro que desea que el framewrok capture.

Otra diferencia, o más bien podría considerarse una desventaja desde nuestro punto de vista, es que los registros si o si deben coincidir con datos almacenados en la base de datos de la aplicación. Aquí, un usuario, puede definir un concepto concreto, a partir de un nodo que no contiene datos relacionados con la aplicación en lo absoluto, por ejemplo, podría definir que un concepto es simplemente un banner, el cual obviamente no está en relación con la aplicación.

Por otra parte, en [43], podemos encontrar un acercamiento hacia la extracción de datos sobre la Web, que se aproxima más a nuestra técnica. Este acercamiento, es provisto por un sistema conocido como Thresher, el cual también se encuentra integrado dentro del browser como una extensión del mismo. Para realizar la extracción de datos, Thresher permite a los usuarios, con pocos conocimientos técnicos, enseñar a sus browsers, como extraer contenido Web semántico directo de los documentos HTML, que se encuentran en la World Wide Web.

Al igual que nuestro framework, Thresher, provee una herramienta de highlighting que permite a los usuarios definir, que parte de la estructura mostrada por el HTML, se desea que sea considerada como la entidad de interés, en la aplicación Web actual. Mientras que aquí, la herramienta de highlighting, realiza esta tarea resaltando el nodo raíz del sub árbol, que concuerda con la estructura HTML (la cual representa la entidad en cuestión), en Thresher su herramienta lo hace por medio de la selección, que realiza el usuario, directamente sobre el mismo contenido HTML. Ambas técnicas, son perfectamente validas y tienen sus ventajas y desventajas. Por nuestra parte, destacamos el hecho de que Thresher, permite realizar una extracción más fina de los datos sobre el modelo HTML, ya que permite en efecto definir una entidad sobre el HTML, que se mapea a un conjunto de nodos que pueden pertenecer a diferentes sub-arboles, dentro de la estructura del DOM. Por otro lado, nuestra herramienta, permite una selección un poco más rápida y prolija, ya que muchas veces es complicado y engorroso, realizar una selección directa sobre el HTML de las aplicaciones Web, más aun, si se trata de aplicaciones cargadas de información. Incluso, podría decirse que nuestra herramienta, es hasta un cierto punto un poco mas intuitiva, ya que al remarcar el nodo que contiene la entidad, ayuda al usuario a saber los límites de lo que representa y contiene a esta misma entidad que desea definir.

Veamos un poco más, sobre cómo es que, funciona nuestra herramienta de highlighting. Como siempre, la mejor forma de entender este tipo de implementaciones, es con un ejemplo. Ya que venimos hablando de libros, definiremos el concepto concreto para el concepto libro de la aplicación Web Barnes&Noble.

Obviamente, lo primero que debe de hacer el usuario es activar la herramienta. El framework, provee de un pequeño menú de opciones, que se encuentra ubicado en la parte superior del navegador como una barra de herramientas más de Firefox.

Fig. 5.2.3 – Barra de herramientas

Esta barra de herramientas provee un menú con dos ítems, "Model viewer" que veremos un poco más adelante, lo que ofrece esta opción y "Enable highlighting", que es la opción que permite que la herramienta comience a funcionar.

Fig. 5.2.4 – Menú ítems

Una vez activada la herramienta, es hora de capturar el concepto, como explicamos antes el usuario solo tiene que explorar la estructura de la aplicación Web con el mouse, mientras la herramienta ira resaltando los nodos del DOM, que se habrán de corresponder con el diseño modelado por el HTML y que representa el concepto concreto deseado en la aplicación Web.

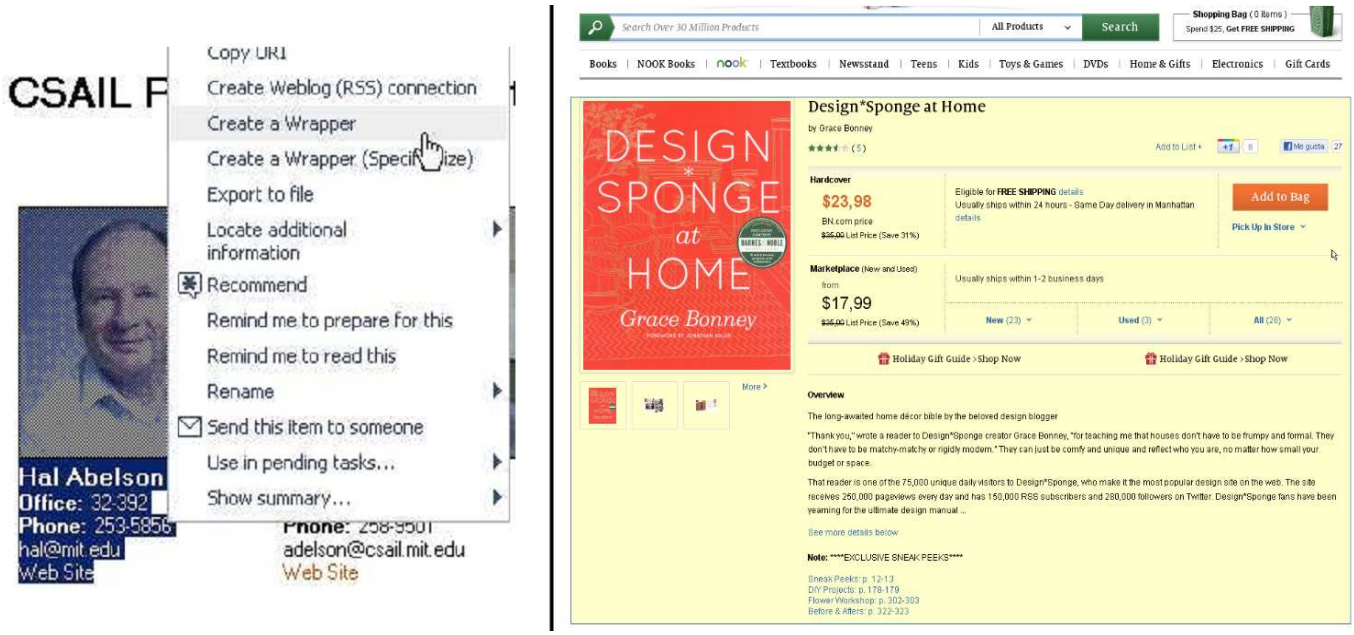

Fig. 5.2.5 – Herramientas de highlighting

En el lado izquierdo de la Fig. 5.2.5, se observa como Thresher implementa su herramienta de highlighting, sacando provecho de la habilidad de selección que brinda el browser. Mientras que a la derecha, puede verse como nuestra herramienta de highlighting funciona, aquí, puede apreciarse, como lo que se encuentra dentro del rectángulo remarcado, será utilizado por el modulo del framework, para capturar/construir el concepto concreto de la aplicación Web.

Ahora que conocemos, que porción del HTML, de la aplicación Web representa nuestro concepto concreto, es tiempo de introducirnos un poco más abajo en la estructura de esta aplicación, para poder terminar de definir, con más nivel de detalle nuestro concepto. En este punto, el modulo proporciona no una herramienta, sino una interfaz, que complementa nuestra herramienta y guía al usuario para poder terminar de definir el concepto concreto. Para acceder a esta interfaz, el usuario debe realizar un clic derecho con el mouse, sobre el rectángulo resaltado por la herramienta de highlighting, similar a como se ve en la Fig. 5.2.5 para la herramienta de Thresher, solo que en vez de tener una opción "Create a Wrapper", el usuario encontrara una opción llamada "Add as model concept…". Nótese, que la idea de wrapper en Thresher y concept en nuestro framework es similar, las dos buscan ayudar a abstraer la entidad que representan, construyendo un objeto que mapee esta entidad, directamente sobre la aplicación Web.

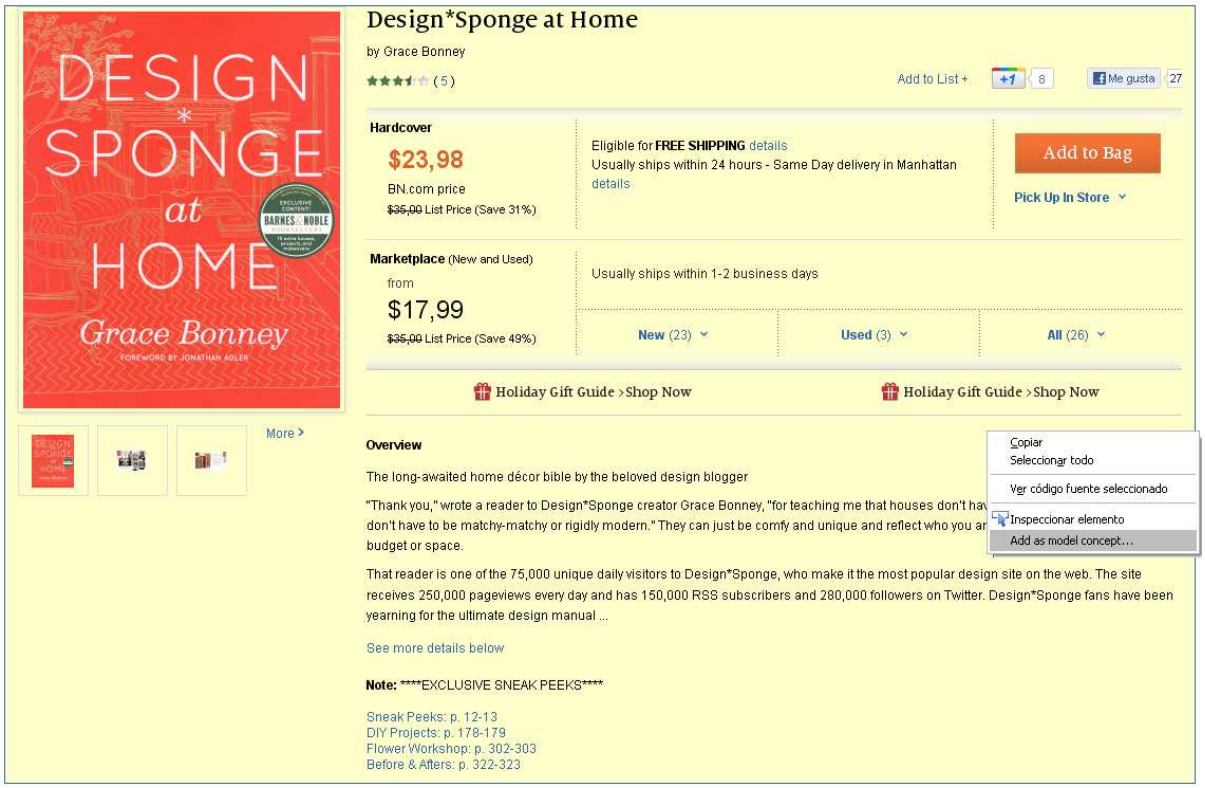

Fig. 5.2.6 – Menú contextual

Esta nueva interfaz, que brinda el modulo, permite definir con más detalle las propiedades que hacen que un concepto concreto se comporte como tal, es decir, que hacen que un "Libro" sea un libro. Recuerde que un libro, como concepto abstracto, tenía un conjunto de propiedades también abstractas, con lo cual es necesario definir donde y como estas propiedades se encuentran modeladas dentro del concepto concreto. Con esta pequeña introducción, damos comienzo a una parte esencial del modulo, y esta es, la de las propiedades de los conceptos concretos.

Entonces tenemos un concepto abstracto, que es un libro, que tiene un conjunto de propiedades abstractas, estas propiedades deberían ser en general aquellas propiedades esenciales que hacen que un libro sea un libro, decimos en general, porque el usuario podría definir cualquier conjunto de propiedades para cualquier concepto, pero por simplicidad asumiremos que se guía por la definición básica de un libro. Por lo tanto, siguiendo el sentido común, seguramente se tenga el titulo, el autor y aunque no tan convencional, pero dado a que se está en un sitio de venta de libros, se considera que el precio debería ser una prioridad, asique tendremos la propiedad precio.

Bien, comencemos por lo básico, como ubicar las propiedades que se encuentran en la aplicación Web y que pertenecen al concepto concreto. Viendo la figura de abajo, resulta obvio para el usuario, ver que es y donde, se encuentra cada una de ellas. Con lo cual, una forma rápida y sencilla de capturar estas propiedades, es que sea el mismo usuario quien le indique al modulo, que parte de la estructura dentro del concepto concreto representa cada una de las propiedades deseadas. Para esto, la interfaz que provee el modulo y que puede observarse en la Fig. 5.2.7, utiliza nuevamente nuestra herramienta de highlighting.

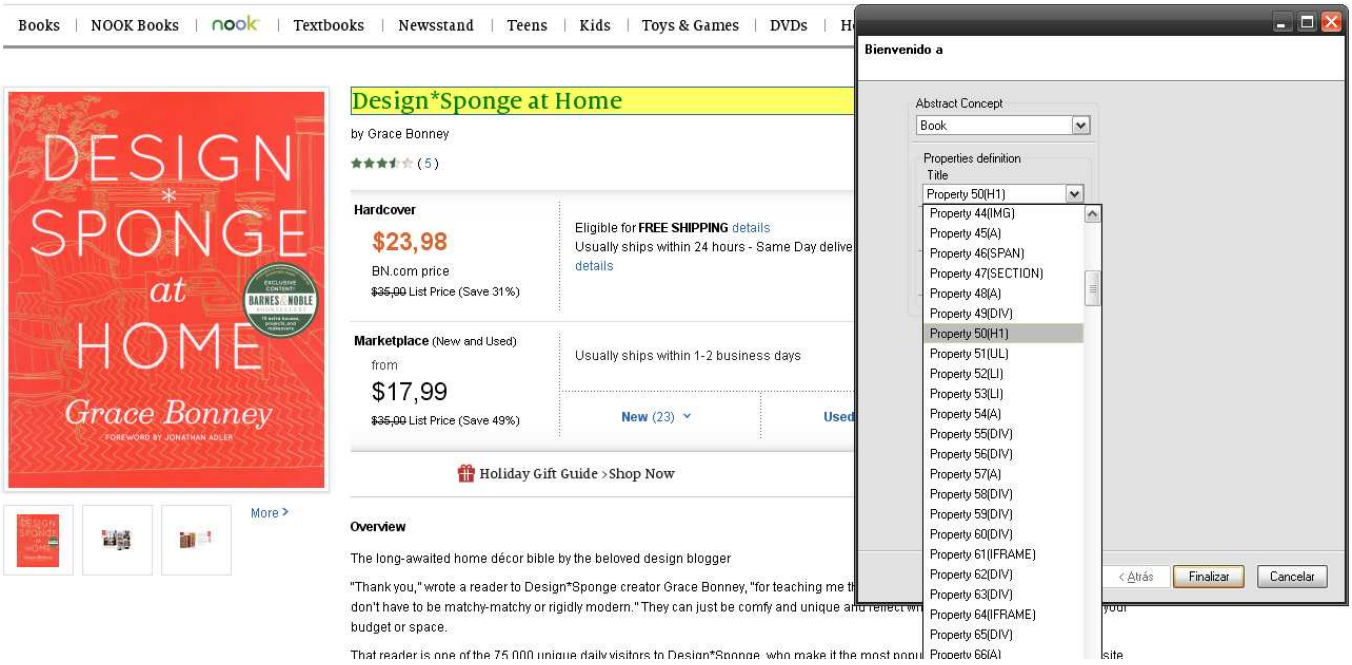

Fig. 5.2.7 – Capturando propiedades

Antes de comenzar a capturar las propiedades de los conceptos concretos, utilizando la herramienta de highlighting, el usuario debe especificar a qué concepto abstracto pertenecen estas propiedades. Para esto, el usuario solo debe seleccionar de la primera casilla, que se encuentra ubicada en el tope de la interfaz, un concepto abstracto de la lista. Luego se desplegara la colección de propiedades abstractas que posee el concepto. En cada una, de estas propiedades abstractas, el usuario deberá indicar que parte de la estructura, de la aplicación web, representa o contiene la propiedad en cuestión. Para esto la interfaz, provee una lista de todos los nodos del sub árbol del DOM, que representan las diferentes partes de la estructura mostrada por el HTML y que pertenecen al concepto concreto, capturado anteriormente por el usuario utilizando la herramienta.

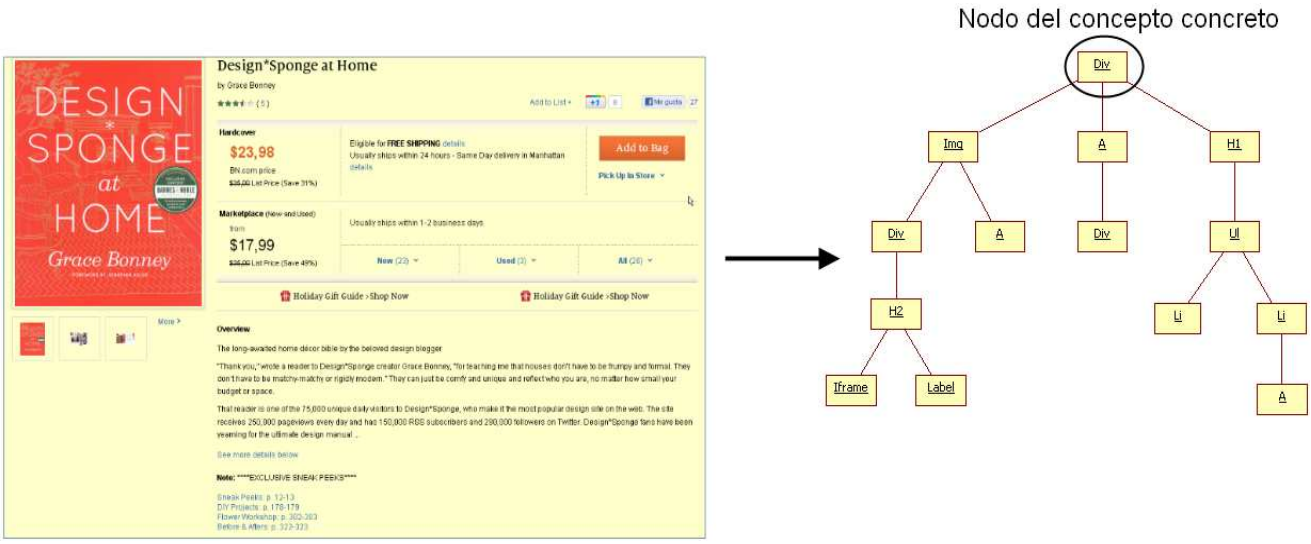

Fig. 5.2.8 – Sub árbol del concepto concreto

Esta posibilidad, de listar todos los nodos del sub árbol, que representa el concepto concreto (Fig. 5.2.8), es gracias a que previamente capturamos el nodo que contiene al mismo, utilizando la herramienta de highlighting, y a la propiedad de jerarquía que provee la estructura de árbol, la cual nos permite acceder a todos sus sub-nodos a través de la API [42] que nos brinda el DOM. Donde, cada uno de estos sub nodos, pude ser utilizado para construir las propiedades concretas de los conceptos concretos.

Entonces para terminar de capturar las propiedades del concepto concreto, el usuario solo debe recorrer, con el mouse, la lista de los nodos del sub árbol del concepto, mientras la herramienta ira resaltando con que parte de la estructura HTML, se concuerda cada nodo seleccionado. De esta forma, el usuario solo tiene que observar si la porción del HTML resaltada, concuerda con lo que él desea que sea considerado como la propiedad que se encuentra definiendo. En la Fig. 5.2.7, puede verse como la propiedad titulo, puede ser definida como un nodo del DOM, que representa una etiqueta HTML llamada “H1”. Por supuesto, el usuario deberá repetir estos pasos, para cada una de las propiedades del concepto

listada en el interfaz (titulo, autor, precio). Una vez, que cada propiedad, tenga especificado que nodo del DOM le corresponde se tendrá capturado completamente el concepto concreto.

Hasta este punto, hemos realizado una serie de pasos que concluyen lo que podríamos nombrar como el proceso de captura para los conceptos concretos. Analizamos como el modulo logro esta fase con ayuda del usuario, guiándolo mediante su herramienta de highlighting y proveyendo una pequeña interfaz. También, es aquí, donde termina la intervención del usuario con lo que respecta a la captura y definición de los conceptos concretos. De ahora en más, el modulo se encargara automáticamente de terminar de construir los conceptos concretos, siguiendo la guía dejada por el usuario.

Ahora que el modulo conoce exactamente que parte de la estructura de la aplicación Web, representa el concepto concreto y que parte dentro de esta misma estructura, representa cada una de sus propiedades, está en condiciones finalmente de crear este concepto junto con sus propiedades. Comencemos por el concepto en sí, recuerde que para esto capturamos el nodo que lo representa, es decir, que contiene todas sus propiedades o más fácilmente el que estaba resaltado con la herramienta de highlighting.

Lo importante ahora, es saber que parte del nodo o que características dentro de él, son necesarias o al menos esenciales para poder definir nuestro concepto concreto. Tal vez, el usuario considere, que ya sabe, cuáles son estas características, “son las propiedades que acabamos de capturar”. Pero si esto es así, para que molestarnos, perdiendo tiempo, en capturar el nodo que representa al concepto. Hubiera sido más fácil, directamente, capturar las propiedades una por una. Si bien es cierto, que estas propiedades son importantes y definen en un sentido al concepto, no lo son todo.

Cada nodo del DOM, guarda una gran cantidad de información dentro de él. En particular, a nuestro modulo, le interesa conocer la estructura del sub árbol, concebida por cada uno de los nodos que conforman la estructura del concepto concreto. La necesidad del modulo de conocer esta estructura, yace en el hecho de que la necesita para luego poder reconocer, basado en un concepto abstracto, un concepto concreto que se encuentra sobre una aplicación Web particular. Más adelante veremos cómo este proceso de detección se lleva a cabo, por ahora y en lo que concierne a la definición del mismo, solo basta con utilizar el nodo completo para construir el concepto concreto.

Ahora es el turno de definir las propiedades del concepto concreto. Al igual, que con el nodo del concepto, el modulo conoce los nodos que representan cada una de las propiedades concretas. Nuevamente, recae la pregunta, de qué es lo que debería de utilizarse de cada nodo para definir las propiedades. Una alternativa directa y simple, sería hacer lo mismo que para el nodo del concepto concreto y quedarse con el nodo en sí. Sin embargo, esta solución aunque atractiva, presenta un par de desventajas que luego traerían nada más que problemas. Esto es debido a una característica, que presentan los nodos con respecto al DOM, al que pertenecen.

Cuando un nodo es capturado del DOM, que actualmente representa la aplicación Web, este pierde “relación” con el DOM de la aplicación, una vez que la misma es recargada. Para

entender mejor este funcionamiento veamos un ejemplo. Suponga que se captura el nodo de una propiedad para el concepto abstracto. Este nodo capturado se encuentra en relación con una instancia del DOM, que fue creada cuando la aplicación Web se cargo por primera vez. En particular, esta relación se establece mediante una propiedad del nodo, conocida como "ownerDocument" [37], la cual obviamente apunta al DOM que lo contiene. Entonces, cuando la pagina es recargada, mas allá de que se trate de la misma URL, una nueva instancia del DOM es creada, pero nuestro nodo capturado previamente tiene como valor del ownerDocument el DOM anterior, es decir sigue apuntado a una instancia anterior o caduca del DOM. Más aun, cuando la aplicación Web es recargada, nuevas instancias de los mismos nodos del DOM son creadas. Por lo tanto, incluso el nodo que representa a la propiedad, ya no se corresponde con el nodo actual del DOM.

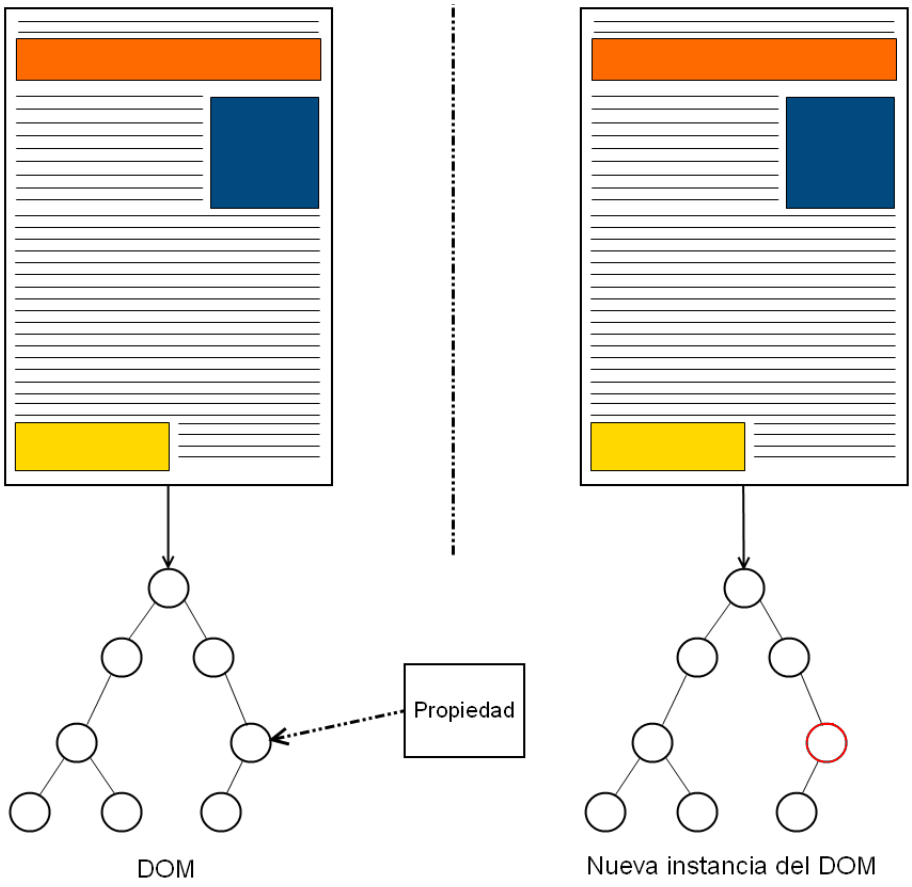

Fig. 5.2.9 – Instancias del DOM

En la Fig. 5.2.9, puede verse como una propiedad capturada a partir de un nodo del DOM, el cual, pierde relación con la nueva instancia del DOM, que se crea cuando la aplicación Web es cargada nuevamente.

Todo esto nos lleva al siguiente problema. Al tener un nodo, que pertenece a una instancia anterior del DOM, los cambios que se realicen por las adaptaciones más adelante, no se verán reflejados en la aplicación actual. Es decir, suponga que se captura el titulo de un libro, si una adaptación trata de manipular este nodo, para realizar cambios en la aplicación web, como poner el titulo en algún otro color o modificar el tamaño de la letra o su estilo, estos cambios jamás se verán en la aplicación actual, ya que el nodo, ya no se encuentra en relación con la misma.

La solución a este problema particular resulta clara, es necesario capturar los nodos que representan a las propiedades, para los conceptos concretos, en el mismo instante que las adaptaciones estén por ser aplicadas sobre la aplicación Web. De esta forma, nos aseguramos de que los nodos, pertenecen a la instancia actual del DOM, para la aplicación Web que las adaptaciones están intentando adaptar. Con esto no estamos diciendo que cada vez que una aplicación Web se termine de cargar, el usuario deberá volver a capturar las propiedades de los conceptos, sino que es necesario buscar una forma alternativa de especificar este conjunto de propiedades.

Tal vez resulte algo confuso, al decir que vamos a capturar las propiedades para los conceptos concretos luego, cuando las adaptaciones estén por ser aplicadas, puesto que dijimos que este modulo se encargaría de la extracción de datos para nuestro framework. Lo que se busca con este modulo si es capturar propiedades, pero no en el sentido de obtenerlas directamente del DOM para poder utilizarlas, sino mas como una forma de armar una guía o template, para que luego el framework pueda utilizarlo, para después si obtener estas propiedades.

Entonces, con el fin de capturar las propiedades de los conceptos concretos, que se encuentran en las aplicaciones Web, es necesario poder indicar con precisión, donde se encuentran localizadas tales propiedades, sobre la estructura de estas aplicaciones. Una forma simple, de localizar cualquier cosa en el mundo real es utilizando referencias, de esta forma solo bastaría con indicar que una propiedad se encuentra en cierta distancia o dirección con respecto de alguna otra cosa en la aplicación. En la Fig. 5.2.7, podemos ver el libro y una de sus propiedades es el autor, con lo cual, podríamos decir que el autor se encuentra “debajo del título” o “arriba del precio”. Sin embargo, estas referencias son un poco vagas o insuficientes para que el modulo pueda especificar con éxito las propiedades, recuerde que la información que el usuario ve a través del browser ha sido moldeada y transformada por el mismo, para poder ser atractiva a los ojos del usuario. Con lo cual, “arriba” y “abajo”, no es exactamente igual si miramos más adentro en la estructura de la aplicación Web.

Pero no todo está perdido, solo necesitamos una manera más específica de expresar estas referencias, una de las posibilidades es hacerlo utilizando direcciones. Por ejemplo, su casa es fácilmente ubicable con solo conocer el nombre de la calle y su número, lo que en conjunto hace una dirección. Entonces, podríamos usar este sistema de dirección, para ubicar las propiedades de los conceptos concretos dentro de la aplicación Web, ¿pero como indicarlo?, ¿cómo debería expresarse esa dirección de tal forma que permita ubicar una propiedad? En el caso de la casa, el sistema de dirección funciona porque puede contar con una infraestructura que respalda este sistema de direccionamiento, como es la estructura de calles y numeración

de su ciudad. De igual forma, se necesita una estructura presente, sobre la aplicación Web, que pueda utilizarse de referencia para ubicar nuestros conceptos. Por suerte, esta estructura existe y es el HTML.

Actualmente, existe una gran cantidad de herramientas, que confían en las características estructurales inherentes de los documentos HTML, de las aplicaciones Web, para poder realizar el proceso de extracción de datos. En [1], este tipo de herramientas son agrupadas bajo el nombre de "HTML-aware Tools". Dos herramientas de este tipo, que pueden ser consideradas de las más importantes y completas, en la construcción de wrappers, son W4F [9] y XWRAP [51], por lo tanto realizaremos un pequeño análisis de estas herramientas, para poder deducir una idea de cómo construir nuestras propias reglas de extracción de datos.

En W4F (World Wide Web Wrapper Factory), el proceso para construir los wrappers, consiste en tres fases: primero, el usuario describe como acceder al documento, segundo, describe que partes de la información desea extraer, y tercero, declara que estructura usar para almacenar la información. En W4F, un documento primero es recuperado desde la Web en base a una o más reglas de recuperación. Una vez recuperado, es entregado a un parser HTML, que construye un árbol de análisis siguiendo el DOM. A continuación, el usuario, puede escribir reglas de extracción para localizar información dentro del árbol de análisis. Estas reglas de extracción son una asignación entre una variable, o propiedad para nuestro framework, y un camino de expresión ("path expression").

Al igual que nuestro framework, W4F ofrece una herramienta que asiste al usuario durante el proceso de extracción de datos, incluso lo ayuda en la tarea de escribir las reglas de extracción, que serán aplicadas a los nodos del árbol, para extraer la información asociada.

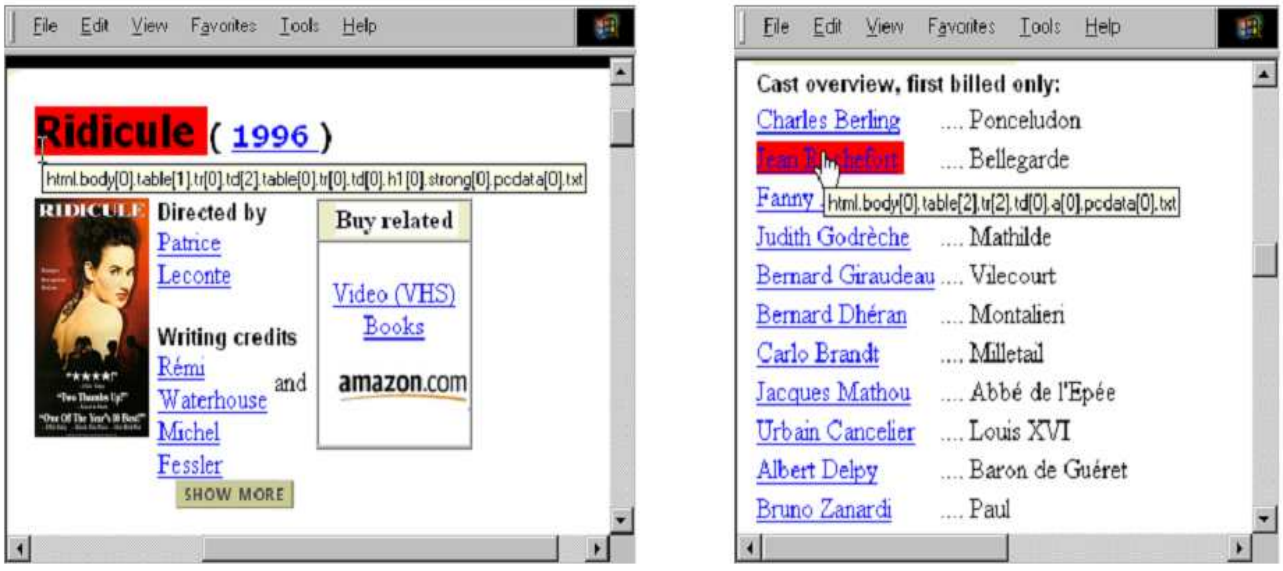

Fig. 5.2.10 – Herramienta de Highlighting en W4F.

En particular, esta herramienta en W4F, también se encuentra implementada como una herramienta de highlighting, donde el usuario cliquea en las piezas de información que son de su interés y la interfaz muestra la regla de extracción correspondiente. Estas reglas de extracción, son expresadas en un lenguaje de extracción de alto nivel, propio de W4F, conocido como HEL (HTML Extraction Language). Una regla de extracción, expresara una navegación a través del árbol y especificara que pedazos de información se han de recolectar y como juntarlas.

Es importante entender, que una regla de extracción, simplemente expresa algún interés sobre una pieza de información en el documento, pero no menciona nada sobre cómo esta pieza de información debe ser usada. Esto es esencial, ya que les da libertad a los usuarios de definir las propiedades de los conceptos concretos sobre las aplicaciones Web, sin estar atados a especificaciones que limitarían su elección. Sin embargo, esta flexibilidad tiene su contraparte, ya que los usuarios que desarrollen las adaptaciones, deben esperar a que los usuarios construyan las propiedades de los conceptos concretos con algún criterio coherente, en base a la propiedad que se encuentra definiendo.

En XWRAP, la herramienta cuenta con una biblioteca de componentes que provee bloques de construcción básicos para wrappers, y una interfaz amigable al usuario, para facilitar la tarea de desarrollo de de los mismos. Antes de enfrentar el proceso de extracción, la herramienta "limpia" las etiquetas HTML en malas condiciones y los errores sintácticos, y convierte el documento HTML en un árbol de análisis ("parsing tree"). La herramienta opera guiando al usuario a través de una serie de pasos, seleccionando en cada paso los componentes apropiados de su librería. En el paso de extracción del objeto, la herramienta despliega un conjunto predefinido de datos heurísticos para las extracción a medida de paginas HTML. Más concretamente, el proceso de extracción de información involucra tres pasos, cada paso genera un conjunto de reglas de extracción para ser usadas por la fase de generación de código para generar los wrappers en cuestión.

"El paso 1, es realizado a través de una interfaz, la cual deja al usuario la identificación de las regiones de interés sobre el documento fuente. La salida de este paso es un conjunto de reglas de extracción que pueden identificar regiones de interés sobre el árbol de análisis" [51]. Esto es similar a nuestra herramienta highlighting, que le permite definir, que parte de la estructura del HTML representa la entidad de interés. Solo que nuestro modulo, no captura la región por medio de una regla de extracción, expresada a través del árbol de análisis, sino que captura el nodo del DOM que representa la entidad.

"El paso 2, es llevado a delante por un programa interactivo, que permite a los desarrolladores de wrappers navegar a través de la estructura del árbol de análisis, y resaltar los tokens semánticos de interés en el documento fuente. Lo que resulta de este pasó, es un conjunto de reglas semánticas de extracción de tokens las cuales permiten localizar y extraer estos token semánticos de interés, y un archivo conteniendo todos los tipos de elementos y los pares de valores de interés" [51].

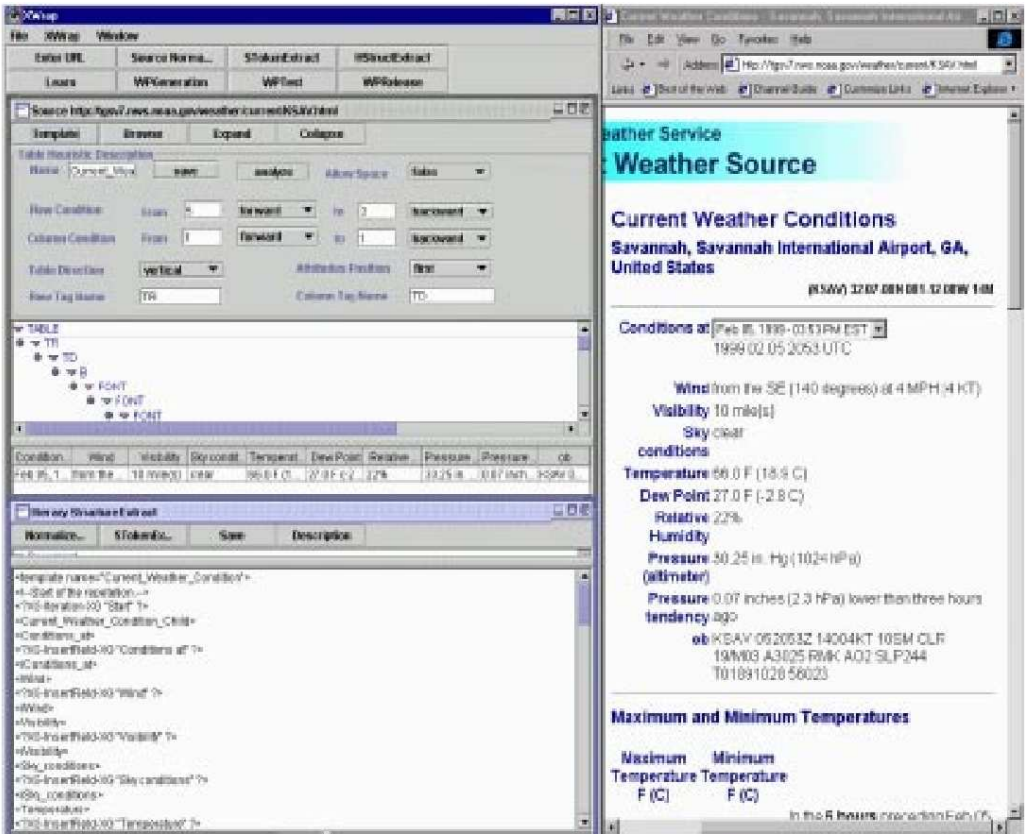

Fig. 5.2.11 – Herramienta de extracción en XWRAP

Estos tokens semánticos, de los que habla XWRAP, serian el equivalente en nuestro framewrok a las propiedades de los conceptos concretos, que también son representadas por medio de las etiquetas HTML del documento de la aplicación web.

“El paso 3, es realizado por el extractor de estructura jerárquica, el cual infiere y especifica la estructura de anidamiento de las secciones de la página web a ser wrappeadas. Tal especificación jerárquica será utilizada para extracción de contenido sensible a la información desde el documento fuente” [51]. El resultado de este paso, es el conjunto de reglas de extracción para la estructura jerárquica, especificadas en una gramática libre de contexto, describiendo la estructura sintáctica del documento fuente de la página.

Ahora que hemos visto estas dos herramientas, podemos ver un patrón en común. En primer lugar, tanto XWRAP como W4F, que confían en la estructura del HTML del documento web, antes de efectuar el proceso de extracción realizan una “limpieza” del documento HTML y luego se lo entregan a algún mecanismo o editor, que se encarga de convertir este mismo documento HTML de la aplicación web actual en un árbol de análisis (“parsing tree”), el cual es una representación que refleja la jerarquía de las etiquetas HTML del documento. A continuación, las reglas de extracción son generadas automática o semiautomáticamente y aplicadas sobre este mismo árbol.

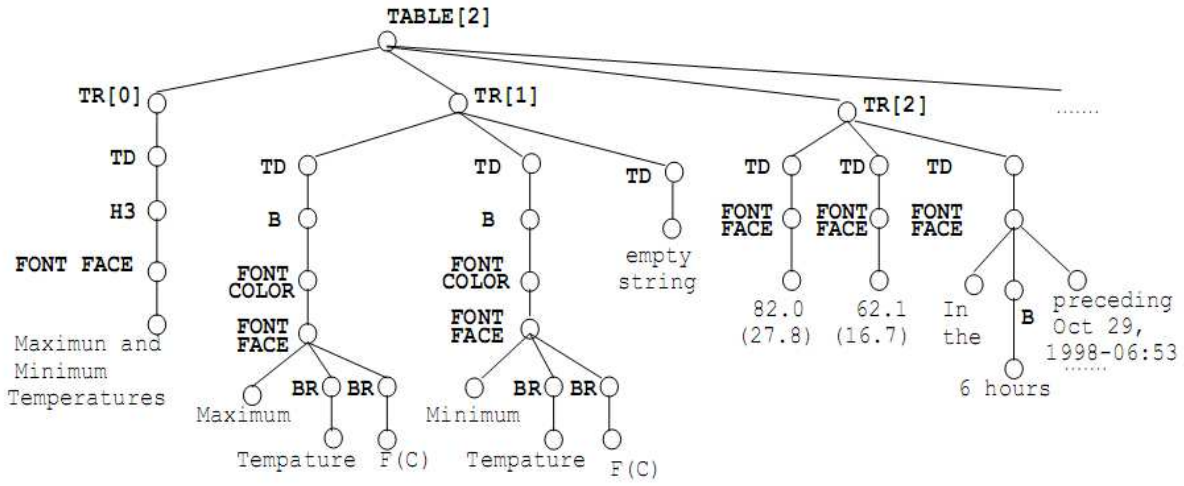

Fig. 5.2.12 – Árbol de análisis

Tanto W4F como XWRAP, expresan las reglas de extracción de los atributos, como una concatenación de las etiquetas o nodos del árbol, que se han de recorrer desde la raíz para alcanzar tal atributo. Por ejemplo, en la Fig. 5.2.10, que pertenece a W4F, se puede apreciar la regla de extracción para un atributo que se corresponde con el título de la película ("html.nody[0].table[2.tr[2].td[0].pcdata[0].txt"). Mientras que XWRAP, la regla de extracción para un atributo o token, que por ejemplo represente la "temperatura máxima" de una página del tiempo (Fig. 5.2.12), puede ser expresada como "table[2].tr[2].td[0]".

Si bien es cierto que estos árboles, permiten construir wrappers con reglas de extracción mucho más precisas y genéricas, también es cierto, que es necesario definir algún lenguaje de extracción de alto nivel, como el de HEL de W4F, para poder manejarlos. Esto requiere algún nivel de experiencia por parte del usuario para la construcción de las reglas de extracción de los wrappers, más específicamente, tanto W4F como XWRAP, requieren que el usuario seleccione mediante la interfaz que presenta algunas propiedades o características de los atributos, para poder escribir de forma correcta el código de extracción para los wrappers. Lo que presenta, en nuestro caso, una contradicción en cuanto a unos de los objetivos de la herramienta.

Nosotros buscamos una herramienta didáctica, con un interfaz simple, que permita a los usuarios con prácticamente nada de experiencia, definir las reglas de extracción, para las propiedades de los conceptos concretos, sin que ellos tengan que saber, siquiera, que están definiendo estas mismas reglas de extracción. Desde el punto de vista del usuario, solo está seleccionando, que es lo que desea que se capture, para el concepto que está definiendo. Considere que, posiblemente, los usuarios que definan los conceptos concretos sobre las aplicaciones Web, usando nuestra herramienta, no serán aquellos que han diseñando las adaptaciones y que utilizan las reglas de extracción, para poder realizar la adaptación. Con lo cual, estos usuarios, seguramente se encontraran ajenos a cualquier método de programación o habilidades de diseño en aplicaciones Web.

Al mismo tiempo, resulta claro, que definir las reglas de extracción, basadas en una estructura jerárquica como el árbol de análisis, es la solución que estamos buscando, para construir nuestras propias reglas de extracción. Con lo cual, nuestra herramienta, deberá utilizar alguna estructura, que represente la jerarquía conformada por las etiquetas HTML del documento Web. Anteriormente, ya estuvimos trabajando con esta estructura, la cual es conocida como DOM. Si recuerda, el DOM, contiene casi exactamente la misma estructura jerárquica que la que presentan las etiquetas HTML, en el documento Web. De esta forma, sería posible construir nuestras reglas de extracción en base a la estructura del DOM.

Al igual que en los árboles de análisis, que utilizan W4F y XWRAP, en el DOM un camino también puede ser especificado por medio de las etiquetas, de los elementos HTML, que los nodos del mismo representan. Pero para esto, también necesitamos emplear alguna clase de lenguaje de alto nivel, que nos permita referenciar de manera individual los elementos dentro del DOM. Si bien, podríamos crear un lenguaje simple, para la especificación de las reglas de extracción, esto nos llevaría de nuevo a las mismas desventajas explicadas anteriormente.

Actualmente, existen una gran variedad de lenguajes que permiten construir expresiones, que recorren y procesan documentos, los cuales presentan una estructura jerárquica. Algunos de ellos ampliamente difundidos y utilizados. Por ejemplo, W4F y XWRAP, permiten exportar el wrapper construido en un formato de archivo conocido como XML (Extensible Markup Language) [20].

XML es un formato muy utilizado en el ambiente de desarrollo de aplicaciones Web, ya que es muy fácil de emplear y escribir, esto se debe, más que nada, porque se asemeja de manera considerable al código HTML de los documentos Web, con lo cual, los usuarios que ya se encuentran familiarizados en el desarrollo de aplicaciones Web, se sienten relacionados con el mismo de forma inmediata. Por otro lado, que sea parecido, no quiere decir que sea lo mismo, XML provee mucha más funcionalidad que HMTL, de hecho, XML es realmente un meta-lenguaje para describir lenguajes etiquetados (“markup languages”). En pocas palabras, XML provee la facilidad de definir las mismas etiquetas de un documento Web y además las relaciones estructurales entre ellas. Todo esto, ha hecho, que XML sea ampliamente utilizado por diversas tecnologías en el mundo. Actualmente, soporta una gran variedad de aplicaciones (authoring, browsing, content analysis, etc.).

Todo el procesamiento, realizado con un fichero XML, está basado en la posibilidad de direccionar o acceder a cada una de las partes que lo componen, de modo que podamos tratar cada uno de los elementos de forma diferenciada. Para esto se ha desarrollado un lenguaje conocido como XPath o XML Path Language [50].

Dado a que, tanto XML como XPATH, se han vuelto una parte importante en la especificación, de una gran variedad de aplicación Web en los últimos años, es de esperar que una estructura fundamental de tales aplicaciones, como es el DOM, incorpore dentro de su API [17], un conjunto de funcionalidades que permitan direccionar y recuperar los elementos representados por sus nodos, por medio de direcciones XPATH.

Con lo anterior en mente y conociendo que nuestra herramienta, hace uso principalmente de la estructura provista por el DOM, para construir los wrappers a lo largo del proceso de extracción, resulta claro que XPATH es el lenguaje ideal para expresar las reglas de extracción, que nuestros conceptos concretos necesitan.

Ahora que sabemos exactamente, en qué formato y como habrán de expresarse las reglas de extracción (XPATH) de nuestra herramienta, continuaremos explicando detalladamente cómo serán especificadas las propiedades para los conceptos concretos de una aplicación Web.

Como ya dijimos, XPATH es una herramienta ampliamente utilizada por diversas aplicaciones, incluso dentro del desarrollo de aplicaciones para realizar tuning en la Web. En [18], se encuentra un trabajo que realiza una explicación sobre el desarrollo de un framework llamado Sitfer, el cual, al igual que nuestro framework, también se encuentra desarrollado como una extensión del browser del usuario y está diseñado para adaptar aplicaciones Web existentes.

Básicamente, Sitfer es una extensión que permite aumentar o enriquecer un arbitrario sitio Web con funcionalidades para filtrado y ordenamiento. Estas nuevas características trabajan dentro de las páginas del sitio web, como si el mismo sitio hubiera implementado estas nuevas funcionalidades. Esto significa, que al igual que nuestro framework, Sitfer necesita proveer alguna forma de extraer datos, directamente de las estructuras existentes de las aplicaciones web, donde estas nuevas funcionalidades van a ser implementadas. Con este fin, Sitfer contiene un algoritmo de extracción de datos basado en XPath.

Su algoritmo de extracción de datos comienza por capturar todos los XPath únicos a elementos <A>, en la aplicación Web actual. De esta forma, cada XPath a un elemento, es calculado encadenando todas las etiquetas HTML, hasta alcanzar el elemento desde el inicio del documento. Con lo cual, el XPath resultante podría terminar en algo parecido a lo siguiente: "/Html/Body/Tbody/Td/Div/Span/A". Con esto, Sitfer asegura que cada XPath, es lo suficiente general para cubrir más que solo el elemento original <A>, pero al mismo tiempo lo suficiente restrictivo para direccionar solo aquellos elementos que son similares a él.

Luego, cada <A> XPath, es expandido hasta abarcar completamente los ítems hipotéticos, donde los elementos <A> se encuentran contenidos (Fig. 5.2.13). Estos ítems hipotéticos de los que habla Sitfer serian nuestros conceptos concretos.

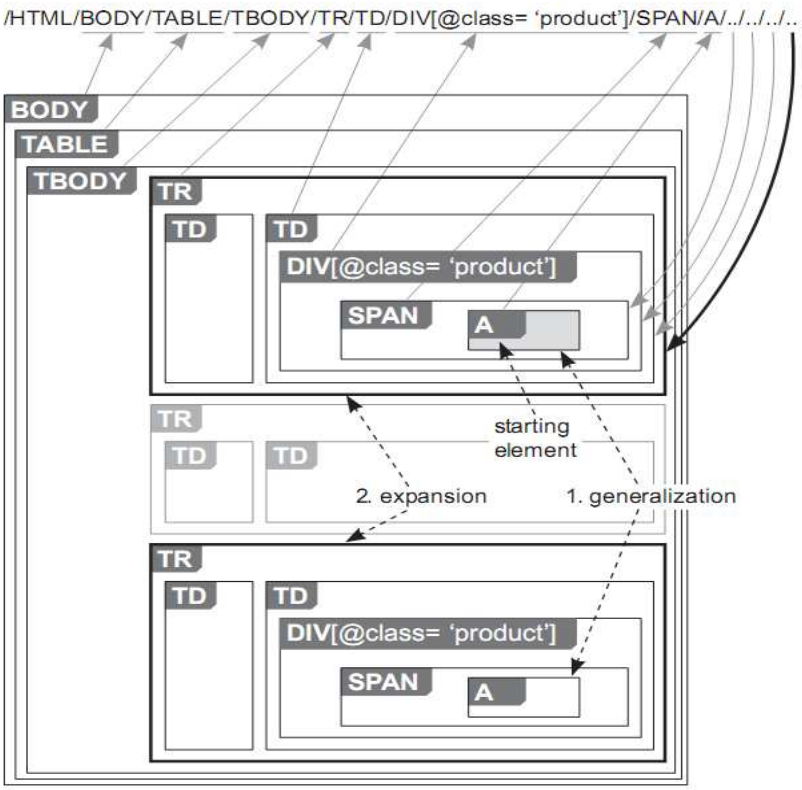

Fig. 5.2.13 – Algoritmo de extracción en Sitfer

Por lo tanto, al igual que en Sitfer, necesitamos expresar las reglas de extracción para las propiedades, basándonos en la estructura HTML de la aplicación Web, más específicamente en las etiquetas que la componen. De esta forma, para especificar la regla de extracción de una propiedad, necesitamos indicar el conjunto de etiquetas HTML, que se ha de recorrer hasta llegar a la propiedad.

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <div>                                          Concepto
            <h1>
                <label> Some title </label>   ← Propiedad 1
            </h1>
            <h2> Author </h2>                 ← Propiedad 2
        </div>
    </body>
</html>
```

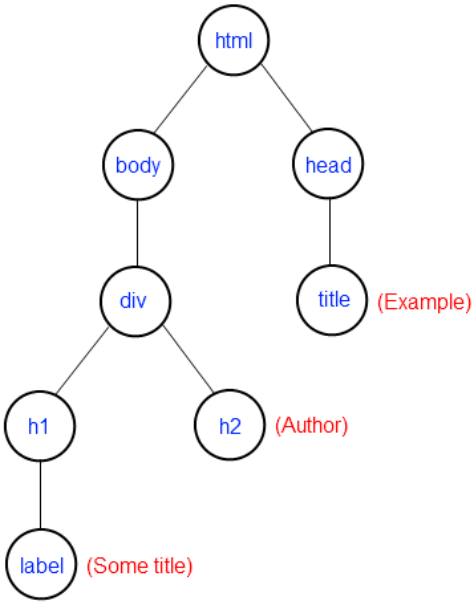

Fig. 5.2.14 – Cuerpo HTML de un concepto y el DOM

Suponga que la Fig. 5.2.14, representa el HTML de un concepto presente en la aplicación Web, este tiene 2 propiedades que se encuentran representadas por las etiquetas "label" y "h2" del HTML. Entonces, para expresar las reglas de extracción de estas dos propiedades, se necesita indicar las etiquetas, que se han de recorrer desde el comienzo del documento Web hasta alcanzar la propiedad. Para el caso de la propiedad 1, el XPATH de la regla de extracción seria "/html/body/div/h1/label" y para la propiedad 2, el XPATH se expresaría como "/html/body/div/h2".

Ahora conocemos exactamente como localizar las propiedades, de un concepto concreto, dentro del documento HTML de la aplicación Web, sin embargo, en nuestro caso no es posible especificar la regla de extracción para una propiedad desde el comienzo del documento Web, como lo hace Sitfer. Esto se debe, a que la instancia que representa al concepto concreto dentro del HTML, o más específicamente el conjunto de etiquetas HTML que lo conforman, podría cambiar de ubicación entre recargas de una misma aplicación Web. Por ejemplo, suponga que al "div" (Fig. 5.2.14), que representa la instancia del concepto, luego de una recarga de la pagina Web, aparece una etiqueta HTML cualquiera, que se antepone a este, como podría ser una etiqueta "p" y que de este nueva esta etiqueta, se desprende nuestro "div". A simple vista, esto no representa un gran cambio, prácticamente la estructura sigue siendo la misma, incluso nuestro concepto sigue contiendo el mismo conjunto de etiquetas, que lo representan. Pero ahora, nuestras reglas de extracción, no conducen a la ubicación de las propiedades, de hecho, no conducen a nada. Para que la especificación de las reglas de extracción sea válida, sería necesario modificar el XPATH para cada una. Por ejemplo, para el caso de la propiedad 1, el XPATH ahora seria "/html/body/p/div/h1/label".

Entonces, ¿cómo solucionamos esto? Bien, por un lado, se podría volver a calcular el XPATH, para el concepto concreto, cada vez que la aplicación cargue nuevamente. Sin embargo, esta alternativa no parece presentar una solución muy elegante. Por otra parte, una mejor solución, sería escribir el XPATH en base a la etiqueta HTML del concepto. Ya que, por más que el concepto cambie de lugar, el conjunto de etiquetas que se encuentran dentro del mismo permanece intacto o al menos eso es lo que se espera. Por lo tanto, si la dirección del XPATH para el concepto cambia, no afectaría los XPATH de las propiedades del mismo. Al igual que con la dirección de su casa, uno indica la dirección desde un punto de referencia, como es su ciudad. En nuestro caso, este punto de referencia esta dado por el concepto o más específicamente, por el nodo del DOM que representa al concepto.

Por lo tanto, de igual forma que con las propiedades del concepto, el nodo también tiene su etiqueta HTML que lo representa, en la figura de arriba, puede verse que este es modelado como una etiqueta “div”. Entonces, tomando esta etiqueta HTML, como punto de referencia, podemos construir una nueva dirección para cada propiedad. De esta forma, la dirección de la propiedad 1, terminaría siendo “/h1/label” y la propiedad 2 tendría la dirección “/h2”. Estas nuevas direcciones, que las llamaremos como direcciones relativas, serán utilizadas más adelante, por el modulo, para poder obtener los nodos del DOM que representan las propiedades de los conceptos.

Finalmente hemos concluido la captura del concepto concreto, tenemos el nodo del DOM, que lo representa sobre la estructura y sus propiedades expresadas en forma de XPath, que nos permiten ubicarlas y recuperarlas. Lo que queda ahora, es presentar algún método o técnica, que nos permita detectar la instancia del concepto (para un concepto abstracto), por medio de su concepto concreto en la aplicación Web, cada vez que esta es cargada. Sin embargo, esto no es tan simple.

Hasta ahora, hemos explicado el proceso de extracción de datos, como si solo existiera una instancia del concepto sobre la aplicación Web. Sin embargo, si recuerda en el capitulo anterior, cuando hablamos de familias de aplicaciones, dijimos que un concepto dentro de la familia podría y de hecho, seguramente, para la mayoría de las aplicaciones Web sea así, tener múltiples instancias dentro de las diferentes aplicaciones que se encuentran en la misma familia.

Cuando en este caso hablamos de concepto, nos referimos a un concepto abstracto, de esta forma un concepto como es “Producto”, “Articulo” o “Libro”, seguramente terminara teniendo múltiples instancias dentro de una aplicación Web, como por ejemplo en el caso de “amazon.com”, cada una de ellas se encontraran, representadas por un único concepto concreto asociado a la aplicación.

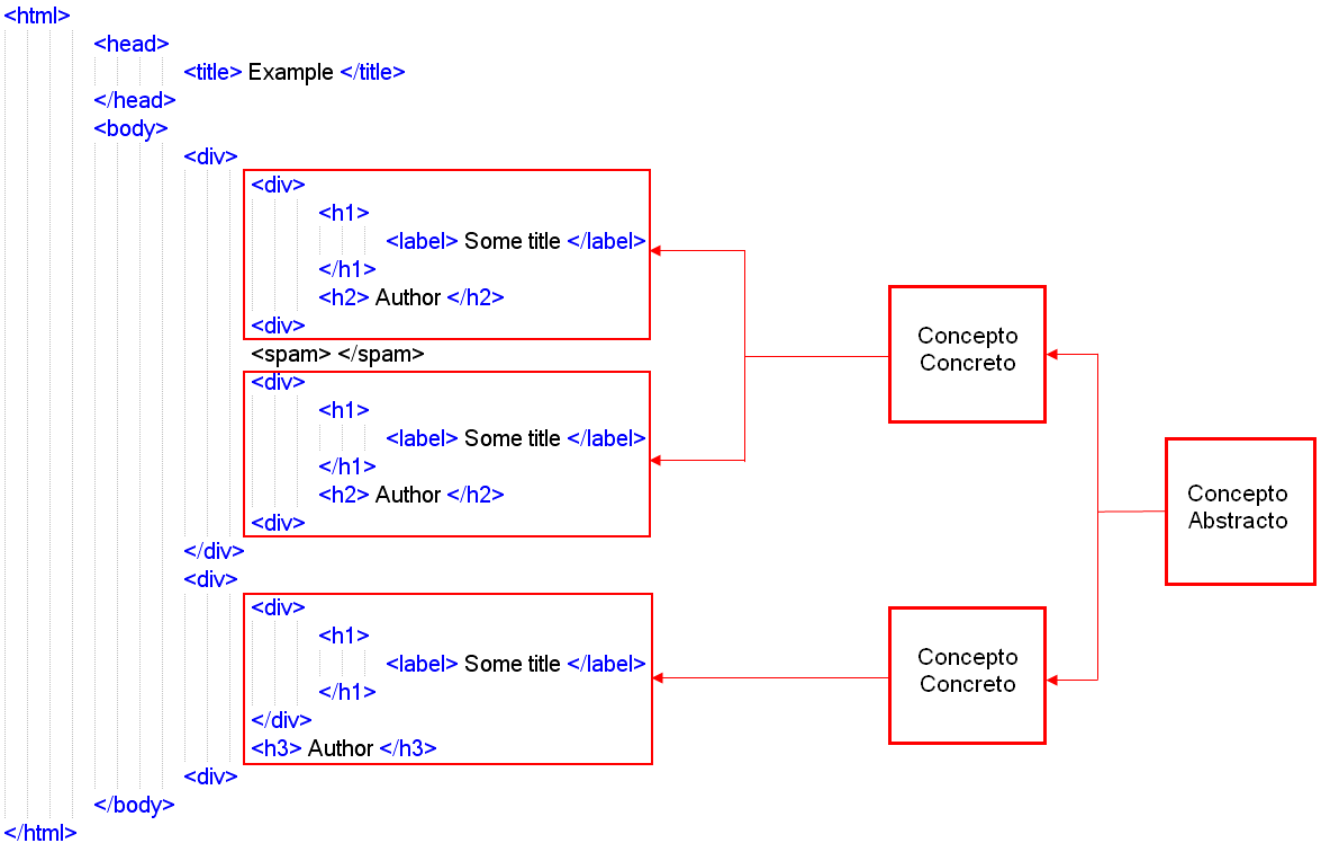

Fig. 5.2.15 – Instancias de los conceptos concretos

Piense, que una aplicación Web, podría llegar a tener una gran cantidad de conceptos abstractos, con lo cual cada uno requerirá que se defina, para la aplicación Web particular, su concepto concreto. Aun más, para una misma aplicación Web, un concepto abstracto podría llegar a tener más de un concepto concreto asociado a la misma (Fig. 5.2.15), con lo cual el modulo debería definir cada instancia, de cada uno de los conceptos asociados, a cada una de las aplicaciones.

Por lo tanto, la mejor opción para resolver esta situación, sigue siendo, que sea el mismo framework quien realice una detección automática. Entonces, el modulo también podría proveer alguna clase de herramienta o implementación, que permita detectar estas instancias dentro de la estructura de una aplicación Web y las entregue al framework, para poder ser utilizadas por los usuario, que desarrollan las adaptaciones, con el fin de modificar la aplicación. Teniendo en cuenta lo anterior, se ha desarrollado un algoritmo de detección que complementa y finaliza el proceso de extracción de datos para aplicaciones Web.

5.2.1 Algoritmo de detección

En esta sección explicaremos, como se encuentra desarrollado el algoritmo de detección, que utiliza nuestro framework, para poder detectar cada una de las instancias de los conceptos abstractos, que se encuentran representadas por sus respectivos conceptos concretos y están repartidas a lo largo de las aplicaciones Web. Con este objetivo en mente, nuestro algoritmo utiliza los wrappers o concepto concretos, construidos anteriormente, durante el proceso de extracción de datos, como templates o guías que lo ayudan a ubicar e identificar cada una de las instancias dentro de la aplicación Web.

Fig. 5.2.1.1 – Wrapper

En la Fig. 5.2.1.1, podemos observar un wrapper típico, que pudo ser construido a través del proceso de extracción, el cual mapea las propiedades abstractas del libro, directamente sobre la estructura de la aplicación Web. Este concepto concreto es construido, tomando como ejemplo, una de las instancias del mismo que se encuentra sobre la aplicación Web, en este caso, una instancia de un libro. Entonces, nuestro trabajo aquí es poder localizar y extraer todas las demás instancias que presentan, por decirlo de alguna manera, las mismas características. Por ejemplo, en la figura de arriba, resulta claro que existen 7 instancias del concepto libro, además de la que se utilizo para construir el concepto concreto. Es fundamental, para el éxito de nuestro framework y de las adaptaciones que se desarrollan a partir del mismo, el poder recuperar estos 7 libros.

Para lograr recuperar estas instancias, repartidas por la aplicación, el algoritmo confía nuevamente en la estructura jerárquica que presentan las etiquetas HTML y que modelan al concepto sobre la estructura de la aplicación Web. Si se analiza, el código HTML de los libros en la Fig. 5.2.1.1, seguramente se tendrá que el mismo conjunto de etiquetas HTML, se repiten para cada una de las instancias del libro y aun mas, respetan el mismo orden jerárquico. Entonces, sería posible que nuestro algoritmo, utilice este patrón para poder detectar cada una de las instancias.

Obviamente, no vamos a trabajar directamente sobre el código HTML, aunque existen trabajos, que si parsean el HTML para poder detectar este patrón en común, entre las distintas instancias que representan un mismo "elemento" en la aplicación Web. Por ejemplo, en Sitfer [18], vimos como el mismo utilizaba una característica en común, de las instancias, de los elementos que se repetían a lo largo de la aplicación Web y a partir del mismo, expandía el XPath, hasta encontrar un XPath, que direccionaba cada uno de los elementos en la lista. Sin embargo, esto funcionaba, porque el framwork consideraba que los elementos se encontraban siempre contiguos y la lista en sí, era la principal función de la aplicación, por ende abarcaba gran parte del código HTML. En nuestro caso, esto podría no suceder, o bien los elementos podrían estar repartidos de forma aleatoria o bien solo podría existir un solo elemento, con lo cual sería imposible construir este XPath.

Nuestro algoritmo, utiliza la estructura que presenta una representación jerárquica en forma de árbol del código HTML, mejor conocida como DOM, para poder realizar la detección de las instancias de un concepto en la aplicación Web. Recuerde que nuestro wrapper conoce el nodo del DOM, a partir del cual, se desprenden todos los demás elementos que forman al concepto. Entonces, tratando este nodo, que representa al concepto concreto completo, como el sub árbol que es dentro del DOM, nuestra idea es encontrar todos los demás sub arboles que son iguales a él y se encuentran esparcidos dentro del mismo DOM.

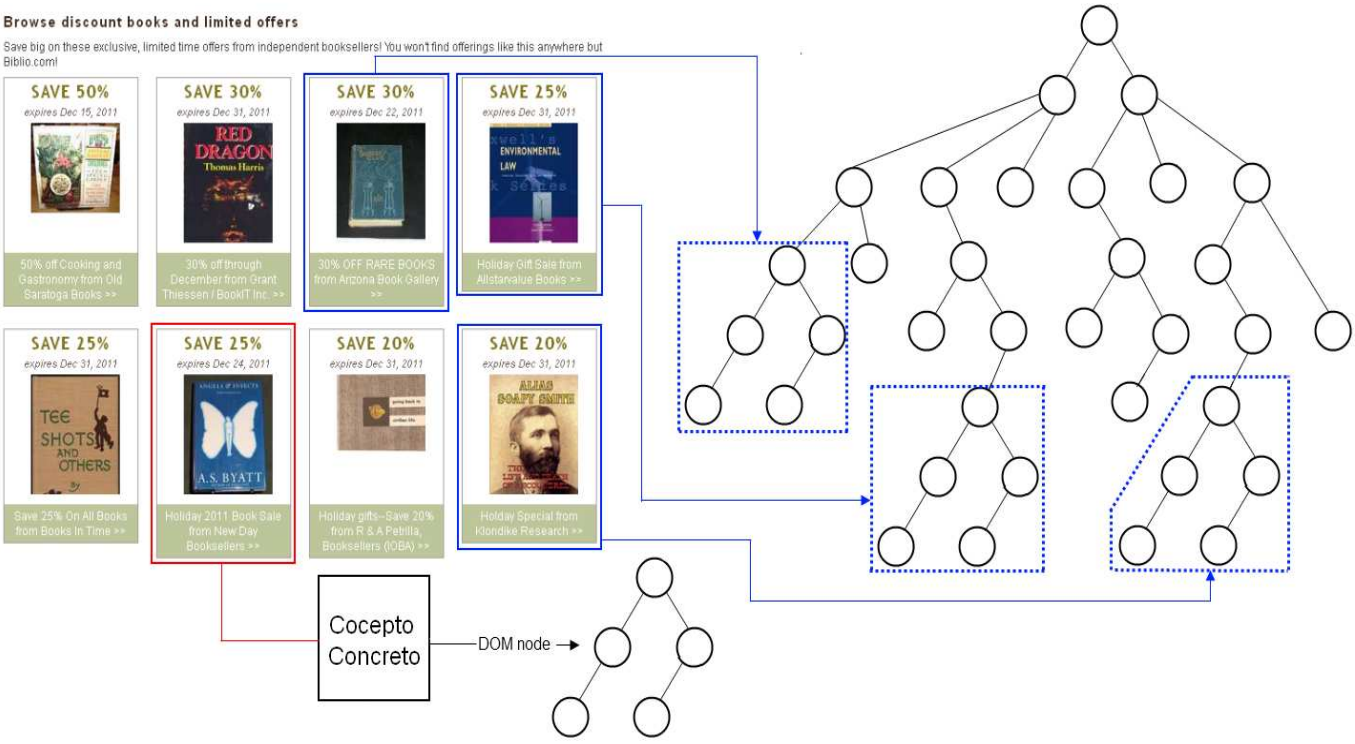

Fig. 5.2.1.2 – Sub arboles de las instancias dentro del DOM

En la Fig. 5.2.1.2, puede verse un concepto concreto, definido para un libro a partir de una de las instancias. De este, se obtiene el nodo del DOM, que contiene el sub árbol que representara a cada instancia. De esta forma, en el árbol de la derecha podemos ver, como tres instancias, que se coinciden con el sub árbol capturado de un libro, aparecen modeladas dentro de la estructura del árbol del DOM. Esta técnica, para extraer datos, es conocida como "Tree matching" [4] y es en la cual se basa nuestro algoritmo. Obviamente, esta técnica es posible de aplicar para la extracción de datos en la Web, gracias a que el HTML de las aplicaciones Web puede ser naturalmente expresado mediante una estructura de árbol.

Actualmente existe una gran variedad de técnicas, para la extracción automática de datos en la Web, basadas en el análisis y comparación de sub árboles del DOM, para localizar y extraer las instancias de los elementos/entidades de interés en las aplicaciones Web. Dentro de estas técnicas, existen tres muy conocidas o populares, además de la que utiliza nuestro algoritmo, estas son: "string matching" [23], parcial "tree aligment" [41] y "tree edit distance" [7].

Anteriormente, hemos estudiado un trabajo que utiliza una de estas técnicas y al igual que nuestro framework, se encuentra implementado como una extensión del browser, con el objetivo de aumentar o enriquecer el contenido de las aplicaciones Web, este trabajo era Thresher [43]. En particular, Thresher, también permite a los usuarios construir los wrappers y sus características relevantes o propiedades, por medio de herramientas e interfaces que provee el mismo framework, para las aplicaciones Web. Luego, utiliza estos mismos

ejemplos, para inducir un patrón flexible y reutilizable aplicando el algoritmo de “tree edit distance”, para encontrar el mejor mapeo entre estos ejemplos y sus respectivos sub árboles dentro del DOM.

De las tres tecinas nombradas anteriormente, la que más se asemeja a la de “tree matching” es “string matching”. La diferencia principal, es que en este algoritmo, se analiza el contenido semántico de los textos contenidos en los nodos de los sub arboles del DOM. De esta forma, el algoritmo, a la hora de determinar si dos sub arboles pueden llegar a ser equivalentes, analiza el texto teniendo en cuenta características especiales, algunos ejemplos de estas características son: existen atributos que contienen valores que proviene de un léxico limitado (ej. NBA team) o contienen términos representativos que se repiten con bastante frecuencia (ej. el término “prensa” es muy común en el valor de un atributo como es “editorial” de un “libro”). También, llega a toman en cuenta, que muchos valores consisten de un número relativamente fijo de “tokens” (ej. el nombre de una persona no suele contener más de 2 a 4 términos). Incluso, este algoritmo, pueden llegar a determinar que una variedad de atributos, se corresponden con valores compuestos por tipos específicos de caracteres (ej. valores con cantidades de dinero o precios suelen contener dígitos y símbolos “$”, “u$s”, etc.). Teniendo en cuenta estas características, el algoritmo de “string matching”, realiza una selección mucho más selectiva, de lo que podría llegar a ser el resultado obtenido por el algoritmo de “tree matching”.

Por otro lado, esta selección tan restrictiva, que se realiza utilizando “string matching”, tiene su contraparte. En [45], se encuentra un análisis que muestra que el algoritmo de string “matching”, no funciona tan bien debido a un uso extensivo de solo unas pocas tablas relacionadas de etiquetas HTML en páginas Web, lo que puede resultar en muchos aciertos erróneos. “El algoritmo de “tree matching”, es más conveniente, porque los arboles reflejan la estructura y layouts de las paginas naturalmente, y por lo tanto, pueden eliminar la mayoría de los aciertos erróneos” [4]. Bien, hasta aquí hemos dado un pequeño vistazo a algunas de las técnicas/algoritmosm que existen para la extracción automatizada de datos a través de estructuras de árbol. Sin embargo, se encuentra fuera de los límites de esta tesis el investigar cada una de estas técnicas, por lo que continuaremos explicando cómo es que se encuentra implementado nuestro algoritmo de “tree matching”.

El objetivo principal de nuestro algoritmo de “tree marching”, es encontrar todos aquellos sub arboles dentro del DOM, que son equivalentes al sub árbol del wrapper o concepto concreto de un concepto particular. Donde, cada uno de estos sub arboles, representa una instancia del concepto en la aplicación Web actual. Obviamente que, para que estos sub arboles sean equivalentes, no solo la distribución que presentan los nodos dentro del sub árbol debe ser la misma, sino que además las etiquetas de los elementos que representa cada uno de los nodos, deben ser también las mismas. Esta exigencia se debe, a que, para una distribución de un sub árbol, mas aun si se trata de una distribución de sus nodos relativamente pequeña, es muy probable que se encuentren una gran cantidad de coincidencias erróneas dentro del árbol del DOM.

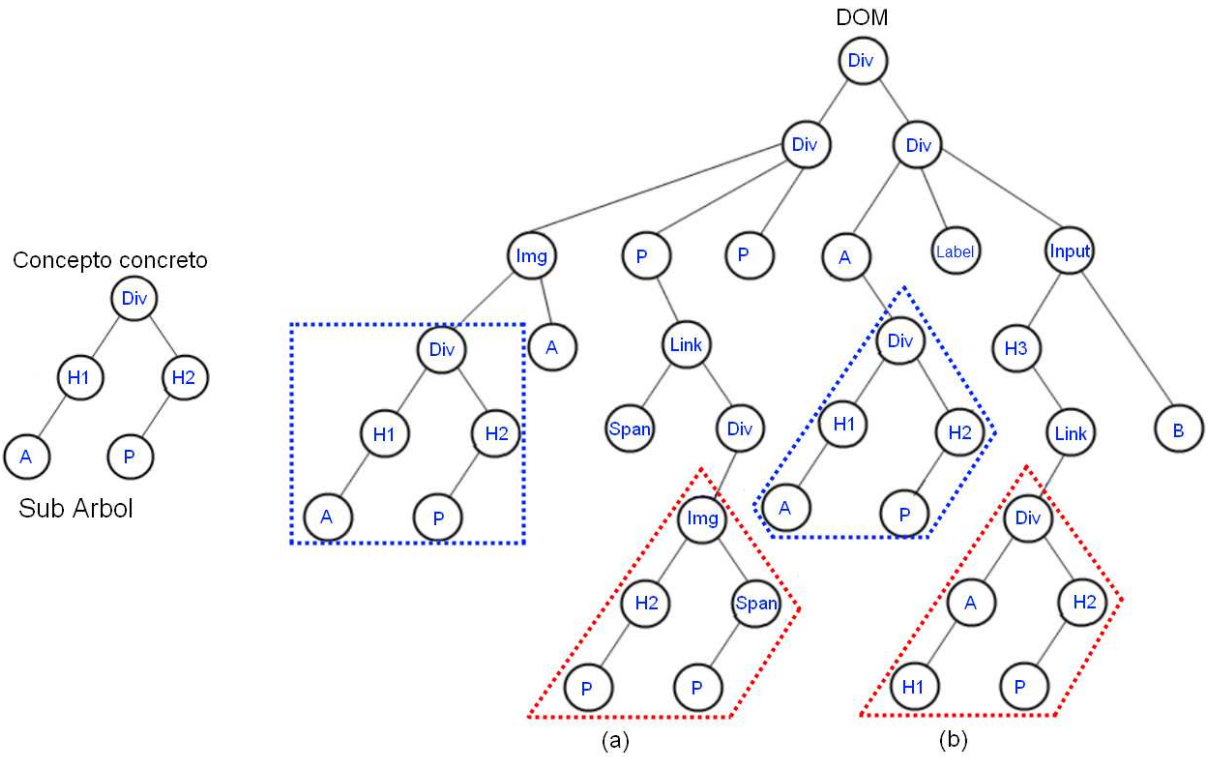

Fig. 5.2.1.3 – Sub arboles del DOM y las etiquetas HTML

En la Fig. 5.2.1.3, podemos ver un ejemplo de lo que estamos diciendo más arriba. Puede observarse, en la porción del árbol del DOM que se encuentra a la derecha, que para el sub árbol del concepto concreto, de acuerdo a la distribución de los nodos del mismo, se pueden encontrar cuatro ocurrencias de sub arboles que presentan la misma distribución en sus nodos. Sin embargo, puede verse como dos, de estos cuatro sub arboles, encontrados dentro del DOM no son iguales (los que están remarcados dentro de las líneas punteadas en rojo), esto es debido a que las etiquetas HTML exhibidas por los nodos de estos dos sub arboles, no se corresponden con el del sub árbol del concepto concreto. Estas diferencias, aunque visiblemente pequeñas, influyen de manera considerable en el funcionamiento de nuestro framework y más específicamente, en las adaptaciones que se aplicaran más adelante utilizando estos sub arboles.

Por lo tanto, nuestro algoritmo debe ser muy cuidadoso en este aspecto. Este debe controlar, no solo que el sub árbol que representa a la instancia buscada, presente el mismo conjunto de etiquetas HTML, sino que, además debe cerciorarse de que se encuentran en el mismo orden exacto dentro del árbol. Si, se observa la Fig. 5.2.1.3, de estos dos sub arboles erróneos encontrados, el (a) exhibe un conjunto de etiquetas casi totalmente distinto, por lo cual es fácilmente descartable. Mientras que el sub árbol (b), muestra casi el mismo conjunto de etiquetas, pero con una pequeña diferencia, dos de estas etiquetas están intercambiados de

lugar (el “A” con el “H1”), este pequeño cambio en el orden es suficiente para que nuestro algoritmo considere descartar esta instancia.

La principal necesidad de ser tan estricto con la estructura de los sub arboles, que representan a las instancias, es que las propiedades que se encuentran mapeadas por el concepto concreto, se encuentran expresadas mediante una dirección XPath, que define un camino único e invariable. De esta forma, la más pequeña variación en la distribución de los nodos de un sub árbol, hace que el camino expresado por el XPath sea invalido o peor aún, podría retornar cualquier otro elemento inesperado del documento Web. Por ejemplo, suponga que para el concepto concreto de la Fig. 5.2.1.3, se encuentra definida una propiedad que tiene especificado un XPath “/div/h1/a”. Entonces, para el árbol (a), resulta obvio que este XPath es imposible, porque de hecho ninguno de las etiquetas HTML se encuentran presentes. En el caso de (b), parece ser posible, pero de nuevo el hecho de que las etiquetas se encuentren fuera de orden hace que este XPath sea inválido, ya que expresa un recorrido en serie a través de los nodos del sub árbol.

Ahora sabemos bien los aspectos que deberá tener en cuenta nuestro algoritmo de “tree matching”, al momento de detectar los sub arboles dentro del DOM para obtener la menor cantidad de instancias erróneas. Por otro lado, también el tiempo de procesamiento que se requiere para buscar estas instancias dentro del DOM es relevante. Por eso, también es importante o al menos ideal tratar de reducir este tiempo, que consume la búsqueda, lo más que se pueda. Piense que, buscar coincidencias de un sub árbol particular dentro del DOM, podría ser costoso en tiempo, ya que el mismo puede llegar a ser extenso en su estructura y por ende en la cantidad de nodos que contiene. Además, este proceso se repetirá para cada uno de los conceptos concretos asociados con el concepto.

Una forma sencilla, de aumentar la eficiencia de nuestro algoritmo, seria reducir la cantidad de sub arboles a buscar dentro del DOM. Esta solución resulta lógica, menos sub arboles entre los que buscar, resulta en menos tiempo de búsqueda. Dado que, los sub arboles están compuestos de nodos, una solución directa seria limitar la cantidad de nodos dentro del DOM antes de que el algoritmo comience a efectuar esta búsqueda.

Con lo anterior en mente, se podría reducir la cantidad de nodos contenidos en el DOM, a solo aquellos nodos que se encuentran dentro del sub árbol, que representa al concepto concreto. Más específicamente, podríamos filtrar los nodos según la etiqueta del elemento HTML que estos mismos representan. Entonces, si el sub árbol contiene un conjunto de etiquetas HTML, como por ejemplo div, a, h1, etc., bastaría con que el DOM solo contenga aquellos nodos que se encuentran dentro de este pequeño conjunto. Sin embargo, si la estructura del sub árbol del concepto concreto, contiene una variedad amplia de etiquetas HTML, los nodos del DOM no se verían reducidos de manera considerable y de esta forma habríamos gastado tiempo valioso en una solución sin sentido.

Una forma más simple y rápida de solucionar esto, y que requiere muy poco esfuerzo de implementación y diseño, seria limitar la búsqueda a solo aquellos nodos del DOM que se coinciden con la raíz de nuestro sub árbol.

5.3 Modulo de adaptación de familia

El objetivo principal de este modulo, es presentar un conjunto de abstracciones e ideas de diseño, que guíen a los usuarios en el desarrollo correcto de adaptaciones para aplicaciones Web. Donde, este desarrollo "correcto", desde el punto de vista del modulo, tiene como fin el lograr que los usuarios construyan adaptaciones lo suficientemente genéricas y flexibles, como para poder ser reutilizadas en cada una de las aplicaciones que conforman una familia dentro de nuestro framework. A diferencia del primer modulo, que se dedicaba principalmente a la extracción de datos y donde era el usuario quien guiaba al mismo durante el proceso, aquí el modulo de adaptación de familias guía o enseña al usuario y no al revés. Donde estos usuarios, ya no son usuarios considerados comunes o poco experimentados, sino desarrolladores que se encuentran familiarizados en la construcción o desarrollo de adaptaciones y en el diseño de aplicaciones Web.

En los últimos años, ha habido un gran avance en tecnologías y técnicas para proveer personalización en las aplicaciones de la Web 2.0. En particular, un enfoque más que otros se ha destacado, este es conocido como DIY (Do-It-Yourself), donde son los mismos usuarios quienes modifican o tunean los sitios Web para poder satisfacer sus propias necesidades. Nuestro framework, se encuentra sincronizado con este nuevo enfoque, proveyendo un ambiente de desarrollo para que los usuarios puedan crear nuevas funciones o contenidos que le permitan modificar y mejorar sus aplicaciones Web.

Un cliente popular, basado en la tecnología DIY es Javascript. De esta forma, los usuarios utilizan esta tecnología, para crear sus nuevas funciones y contenidos. Para esto, desarrollan pequeños programas conocidos como scripts, los cuales les permiten mejorar o modificar sus aplicaciones Web favoritas. Muchas veces, estos mismos scripts son inyectados en los browsers de manera transparente, por medio de pluigns o weavers. Uno muy popular y que ha tenido un crecimiento considerable en estos años es Gracemonkey. Anteriormente, ya estuvimos analizando este plugin y más adelante lo estudiaremos un poco más aun.

Entonces, los usuarios crean adaptaciones que desarrollan por medio de scripts. Estos scripts, reaccionan a eventos cuando se interactúa con la aplicación Web o la misma se carga. Dado que, estos scripts se encuentran implementados dentro del ambiente del browser, pueden acceder a cualquier nodo del DOM de la aplicación Web y finalmente, pueden modificar esta misma aplicación a voluntad. Sin embargo, esta libertad que se la da a los scritps de los usuarios, tiene su contraparte. Para poder realizar estos cambios manualmente, los script deben tener conocimiento sobre la implementación de la aplicación Web. De esta forma, el script, seguramente terminara quedando ligado a la actual estructura, a la información y al estilo de la aplicación Web. El problema con esto, es que las aplicaciones Web están destinadas a cambiar frecuentemente (ej. debido a una actualización de los contenidos de la página Web). Por lo tanto, si la implementación de la aplicación cambia, el script deja de funcionar adecuadamente. Esto, obviamente, estropea todos los esfuerzos por parte de los usuarios para tunear la aplicación.

Nuestro framework intenta solucionar este problema, protegiendo los scripts de los usuarios, de las actualizaciones frecuentes de las páginas. Básicamente, la idea consiste en aislar aquellas partes del script, que tienden a sufrir modificaciones cuando las pagina se ven actualizas, de aquellas partes más estables del script, que no se preocupan por estos cambios frecuentes.

Un principio fundamental de la Ingeniera de Software es la ocultación de la información, por ejemplo, “el ocultamiento de decisiones de diseño en un programa de computadora que son más probables a cambios, por lo tanto protegiendo otras partes del programa de cambios si estas mismas decisiones de diseño son cambiadas” [35]. Desde el punto de vista, de las aplicaciones Web o más específicamente, desde el punto de vista de los documentos DOM, estas decisiones de diseño se encuentran realizadas en como el contenido dentro de las aplicaciones es estructurado, presentado y navegado.

Este ocultamiento de información, implica la existencia de una interface estable que aislé a los usuarios de la implementación. Esto significa, que la interfaz aislara los scripts desarrollados por los usuarios, de la implementación de la aplicación web, protegiendo aquellas partes estables del script, que no deberían verse afectadas por las actualizaciones de estas aplicaciones.

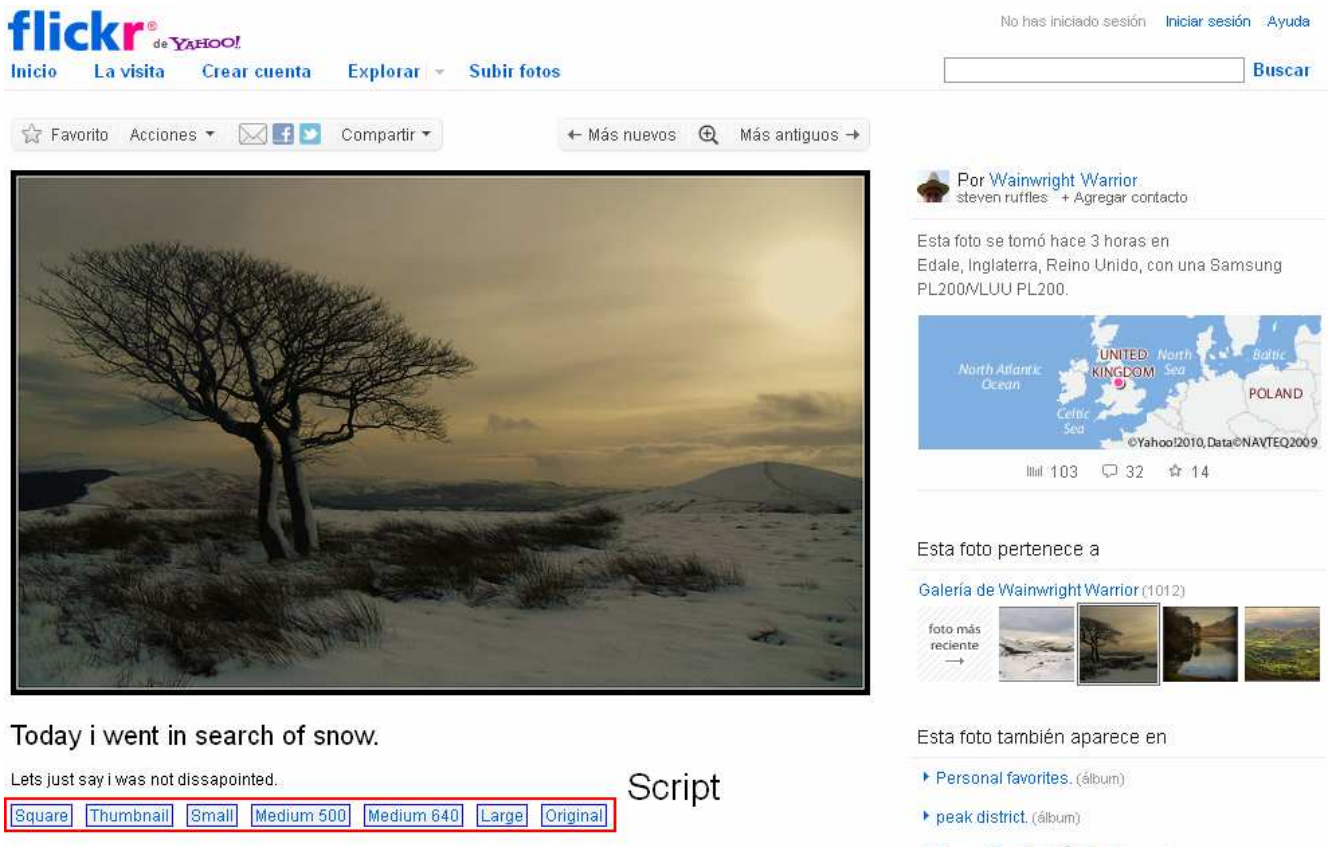

Fig. 5.3.1 – GM script

En la figura de arriba, podemos ver un script clásico de Grasemonkey (anteriormente vimos este script). Por si no recuerdan, el objetivo final de este script, es agregar nueva funcionalidad a la aplicación Web, por medio de un conjunto de etiquetas que permiten visualizar la imagen en distintos tamaños.

El script se encuentra desarrollado siguiendo las técnicas tradicionales, es decir, el mismo se encarga de indagar manualmente en la implementación de la aplicación Web, para insertar estas nuevas funcionalidades dentro de la aplicación. Esto, como ya vimos, hace que el script quede ligado a la estructura de la aplicación Web.

```
function addJQuery(mainScript) {
  var script = document.createElement("script");
  script.setAttribute("src", "http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js");
  script.addEventListener('load', function() {
    var script = document.createElement("script");
    script.textContent = "(" + mainScript.toString() + ")();";
    document.body.appendChild(script);
  }, false);
  document.body.appendChild(script);
}

// the guts of this userscript
function flickrSizeHackMain() {

  if($("div#main div#primary-column div#meta").length <= 0) {
    return;   // Not a photo page. No need to run.
  }


  // Create the HTML for display.
  var linkStyle = {"border": "1px solid #0000ff",
                   "padding-left" : "2px",
                   "padding-right" : "2px",
                   "font-family" : "arial",
                   "background" : "#f3f3f3",
                   "margin-right" : "10px",
                   "width" : "250px"};
  $("div#main div#primary-column div#meta").after('<div id="FlickrSizeHack"><a id="Hacksq" title="Square">' +
  '</a><a id="Hackt" title="Thumbnail"></a><a id="Hacks" title="Small"></a><a id="Hackm" title="Medium 500">' +
  '</a><a id="Hackz" title="Medium 640"></a><a id="Hackl" title="Large"></a><a id="Hacko" title="Original"></a>' +
  '<span class="cNew"><span class="L1">Loading</span> <span class="L2">all</span> <span class="L3">sizes...' +
  '</span></span></div>');
  $("div#FlickrSizeHack a, div#FlickrSizeHack span.cNew").css(linkStyle);
  $("div#FlickrSizeHack a").css("display", "none");



      // Animation of the 'loading' block.

      var Lone = $("div#FlickrSizeHack span.L1");
      var Ltwo = $("div#FlickrSizeHack span.L2");
      var Lthr = $("div#FlickrSizeHack span.L3");
      function doFading() {
          Lone.fadeOut(200);
```

Fig. 5.3.2 – Código del script.

En la figura de arriba, puede verse el código exhibido por el script que vimos antes. Este se encuentra escrito utilizando JSON (JavaScript Object Notation) [30], que es un formato ligero para el intercambio de datos. JSON, es un subconjunto de la notación literal de objetos de Java Script que no requiere el uso de XML. Por cuestiones de espacio no podemos mostrar todo el código contenido en el script, pero esta porción del mismo es suficiente para analizar las desventajas que provienen del scripting tradicional. Aquellos que deseen ver el script completo, en [40] pueden encontrarlo. De este ejemplo, podemos resaltar ciertas limitaciones clásicas:

- Una de las primeras cosas, que el script debe resolver, es cuando va a ser invocado. Es decir, cuando el script debe ejecutarse, por ejemplo, cuando la pagina se cargo o cuando alguien hiso click sobre algún elemento especifico, que pertenece al documento Web de la aplicación. La forma tradicional, de que los script definan en qué momento determinado van a entrar en acción sobre la aplicación Web, es a través del uso de eventos. Estos eventos, conocidos como "GUI events", son predefinidos por el DOM. En la Fig. 5.3.2, podemos ver como el script se asocia específicamente al evento "load" de la aplicación. Esto significa, que será invocado cuando la aplicación termine de cargarse. Este hecho, de tener que asociarse a eventos predefinidos por la implementación de la aplicación Web, hace que el script sea frágil a los cambios. Por ejemplo, el "load" podría ser remplazado por otro comando ante una eventual actualización del DOM.
- La información que se encuentra en la aplicación Web y que necesita el script, es recuperada manualmente, explorando la estructura de la misma. Esto obviamente, liga el script a la estructura actual de la aplicación Web. Por ejemplo, en la Fig. 5.3.2, podemos ver como el script hace uso de su conocimiento sobre la estructura implementada y de la localización del elemento que contiene a la imagen, a la que desea agregar las nuevas funciones. De esta forma, puede extraer la porción de la estructura directamente desde el DOM, indicando el camino de nodos a recorrer dentro del mismo. Nuevamente, esto hace que el script sufra ante una actualización de la aplicación. En este caso, si la estructura que representa la entidad de interés para el script cambia de lugar o se implementa con un conjunto diferente de elementos HTML.

Entonces, nuestra interface debe ser capaz de resolver como mínimo estas desventajas, que presenta el scripting tradicional. En [31], un grupo introduce la idea de una interface, la cual ellos dan a conocer como "modding interface". Al igual que con nuestra interfaz, ellos buscan proteger el script de "*decisiones de diseño que son más probables a cambiar*".

Con este fin, su modding interfase, especifica que puede ser cambiado en la aplicación Web, mediante la noción de "concept" y donde puede ser cambiado dentro de la misma aplicación, mediante la noción de "conceptual event". Donde un "concept", es una noción o idea

significativa dentro de un dominio, que es representada por un sitio web (por ejemplo, post, libros, productos, etc.), pero independiente de cómo el concepto termina siendo implementado (por ejemplo, div, label, table, etc.).

Esta interfaz, presentada por este grupo, es muy similar a la que nosotros hemos venido construyendo. Nótese, que su noción de "concetp", es casi lo mismo, que notros estamos planteando a través de la idea de "conceptos", como era el caso del libro para la aplicación Web de venta de libros. Al mismo tiempo, su idea de "conceptual event", que vendría a ser una ocurrencia del concepto dentro de la estructura de la aplicación Web, es similar a lo que nosotros tratamos con los conceptos concretos, por ejemplo, para las instancias de los libros en una aplicación Web particular.

Sin embargo, esta interface implementada, presentada por el grupo como una capa conceptual sobre la estructura del DOM de una aplicación web, requiere que sea desarrollada junto con la implementación de la aplicación misma. Nosotros, por otro lado, buscamos crear esta interface sobre la estructura de aplicaciones Web existentes, sin tener que redefinir la aplicación completa para ello. En [27], estos mismos autores, extienden la idea para permitir a los usuarios que desarrollan estos scripts, escribir sus propias abstracciones conceptuales para cortar la dependencia existente con los desarrolladores desconocidos de estas aplicaciones. De esta forma, el mantenimiento de los scripts es mucho más fácil, ya que cuando el DOM de la aplicación Web cambia, solo la correspondencia entre los conceptos y el DOM necesita ser redefinida. Esta idea, ya se aproxima mucho mas a nuestra búsqueda de la implementación de una interface, que pueda ser "instalada" sobre una aplicación web existente.

Nosotros cortamos parte de esta dependencia de la que habla [27], mediante el uso de los wrappers (o para nuestro framework), mediante el uso de los conceptos concretos. De esta forma, si la implementación de la aplicación Web, se ve modificada, debido a alguna actualización o cambio en la implementación de la página web, lo que seguramente terminara influyendo en la estructura del DOM, nosotros solo tenemos que redefinir el concepto concreto para la aplicación particular. Sin embargo, para que esto funcione, se necesita una interfaz soportada por un conjunto estable de abstracciones.

Entonces, este grupo propone crear estos scripts, más robustos y flexibles, mediante este nuevo enfoque que dan a conocer como "Interfaces for scripting". Básicamente, este nuevo acercamiento, sigue manteniendo el objetivo principal de [31], el cual era abstraer los script de preocupaciones innecesarias que podrían hacer que el mismo se vea afectado debido a cambios en la implementación de las aplicaciones Web. De esta forma, los scripts siguen subscribiéndose a los "conceptual events", en vez de subscribirse a eventos de bajo nivel, como eran los eventos GUI del DOM. La principal diferencia, recae en el proceso de desarrollo de estos mismos scripts, creados por los usuarios, que se encuentra divido en dos fases.

Durante el desarrollo, primero los scripts implementan la interface (llamada Class Script). Luego, un segundo script, provee el soporte lógico en el tope de esta interfaz (llamado Mod Script). De esta forma, la lógica es desacoplada/separada de la realización concreta de los conceptos, sobre los cuales esta lógica es aplicada.

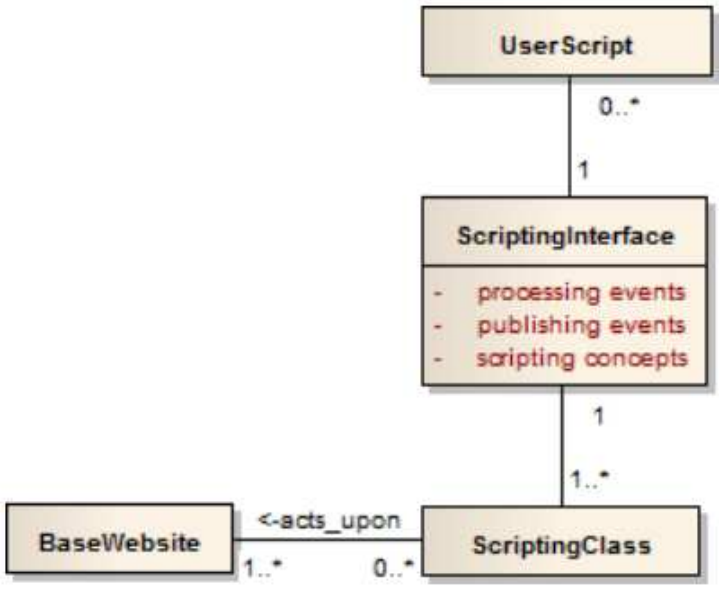

Fig. 5.3.3 – Scripting Interface

Para lograr este proceso, una "Scripting interface encapsula el árbol DOM de la aplicación web, y provee un conjunto de servicios que permiten, leer y escribir sobre este árbol DOM" [27]. De esta forma, la parte de lectura se encarga de incorporar la interfaz requerida como el conjunto de eventos, que la implementación de la interfaz solo señala, pero deja a los scripts su procesamiento. Esto básicamente, implica abstraerse de los UI events (por ejemplo, aquellos que se elevan cuando se manipulan elementos HTML) hacia conceptual events (por ejemplo, aquellos señalizados cuando se actúa sobre los conceptos).

En cuanto a la parte de escritura, trata de evitar que los scripts modifiquen directamente la estructura del DOM, y por ende terminen quedando ligados a la implementación actual de la aplicación Web. Esta parte, elimina esta dependencia identificando el lugar a ser modificado por el script, dentro de la estructura de la aplicación web, en términos de "ocurrencias de los conceptos" en lugar de a través de los nodos del DOM.

Mientras que nuestro framework, comparte la filosofía detrás de este trabajo, nosotros creemos que es necesario encaminarla un poco hacia nuestro rumbo, para que se encuentre acorde a las necesidades que se intentan satisfacer con nuestra interfaz. En primer lugar, entre los objetivos que nuestro framework intenta cumplir, se encuentra el de proveer un medio para mejorar la experiencia de navegación del usuario. Con este fin, las adaptaciones que han de ser aplicadas sobre las aplicaciones Web, deben ser efectuadas incluso antes de que el

usuario comience a utilizar la aplicación. Por lo tanto, estas adaptaciones tienen que ser introducidas dentro de la estructura de la aplicación web, inmediatamente después de que la misma ha sido traída desde el servidor y terminada de cargar dentro del browser (antes que el usuario comience siquiera a utilizarla). Esto es equivalente, a cuando los script de los usuarios, se asocian a la aplicación Web por medio del evento de carga ("load") del DOM. Como es el caso del script de la Fig. 5.3.2, que utiliza este evento "load" para fijar sus nuevas funciones a la aplicación Web o de manera similar en el caso de [27], cuando se asocian al conceptual event "loadBook", para el caso de un script que modifica una página Web que contiene libros. Aquí no estamos diciendo, que el evento "load" y "loadBook" sean exactamente lo mismo, sino que el objetivo por el cual los script los utilizan es equivalente, es decir, los dos buscar ser invocados ni bien la aplicación Web haya terminado de cargar dentro del espacio del browser.

Entonces, ya que nuestros script solo necesitan estar pendientes a un evento particular, que es el de carga, nosotros podemos cortar más aun esta dependencia entre estos scripts y la implementación de la aplicación Web. Con lo cual, los scripts no necesitan siquiera subscribirse a un evento predefino por el DOM o provisto por la interfaz, sino que nuestro framework estará pendiente de si la aplicación Web se termino de cargar. Una vez que esto suceda, será el mismo framework quien se encargara de avisar a los scripts que la aplicación Web ha terminado de cargar. De esta forma, los scripts solo tienen que preocuparse por realizar las modificaciones que los mismos desean introducir en las aplicaciones.

```
// ==UserScript==
// @name          Scripting Interface for Books
// @include       *
// ==/UserScript==
var bookInterface={
 "ScriptingConcepts":[{
  "conceptId":"Book",
  "attributes":[{"attributeId":"title","type":"string"},
                {"attributeId":"author","type":"array"},
                {"attributeId":"isbn","type":"string",
                 "pattern":"/[0-9]{10}/"},
                {"attributeId":"price","type":"number"}]
 }],
 "PublishingEvents":[{"id":"loadBook",
                      "payloadType":"Book",
                      "uiEventType":"load",
                      "cancelable":false}],
 "ProcessingEvents":[{"id":"appendChildBook",
                      "payloadType":"HTMLDivElement",
                      "operationType":"appendChild",
                      "targetConcept":"Book"}]
}
```

Fig. 5.3.4 – Interface de un libro

En la Fig. 5.3.4, puede verse una interface definida para un libro a trabes de la idea de Scripting Interface. Dejando de lado los “conceptual events” (“PublishingEvents” y “ProcessingEvents”) que utiliza la interfaz y que como explicamos nuestros scripts no necesitaran utilizar, el resto de esta interfaz provee un conjunto de especificaciones para el concepto, que esta interfaz encapsula y que se encuentran definidos dentro de la misma. Este conjunto de especificaciones, son en sí, los atributos que posee un concepto, en este caso los atributos de un libro (titulo, autor, isbn, id). Además de, por supuesto, el nombre del concepto que se encuentra dentro de la interfaz.

Nuestro modulo provee dentro de su conjunto de abstracciones, una en particular que desempeña exactamente el mismo papel que la Scripting Interface, de por ejemplo, un libro en [27]. Estos son, los conceptos abstractos, los cuales al igual que en la interfaz de arriba, contienen el conjunto de atributos que conforman a un concepto. Solo que en nuestro caso, estos atributos son llamados propiedades abstractas. Anteriormente, cuando vimos como crear el concepto concreto para un concepto abstracto “libro”, lo hicimos basándonos en un conjunto de propiedades, donde estas propiedades eran los atributos que pertenecían a un libro.

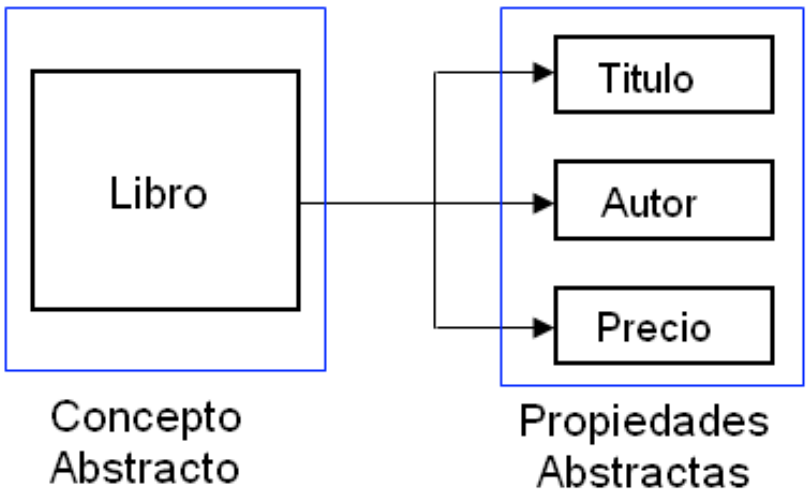

Fig. 5.3.5 - Interfaz

El objetivo que se busca aquí, al abstraer las propiedades de los conceptos, tanto de parte de [27], con su Scripting Interface, como nosotros con nuestra interfaz, es poder cortar parte de la dependencia entre los script que desarrollan los usuarios y la implementación de las aplicaciones Web.

En [27], esta dependencia viene dada por el hecho, de que comúnmente, los scripts utilizan su conocimiento sobre la implementación de la estructura de una aplicación Web, para recuperar

los pedazos de información que se corresponden con los atributos de un concepto y que se encuentran esparcidos dentro de la misma. Por ejemplo, utilizando direcciones Xpath para acceder a la información dentro de la aplicación, que estos scripts necesitan. Con esta nueva interfaz que provee, los script ya no acceden a la estructura de la aplicación para la recuperar información crudamente desde el DOM, sino que se comunican con la interfaz para obtener los atributos de los conceptos.

Por nuestra parte, la interfaz que implementa nuestro modulo, también busca ayudar a cortar parte de esta dependencia (donde los scripts escarban manualmente en la estructura de la aplicación Web en busca de información), con la abstracción de las propiedades para los conceptos que manejan los scripts. De esta forma, los usuarios especifican por medio de la interfaz, que atributos de un concepto desean y nuestra interfaz se encarga de mapearlos, directamente sobre la implementación de la estructura de la aplicación Web.

```
// ==UserScript==
// @name           Scripting Class for Amazon Books
// @include        http://www.amazon.com/*
// @require        http://userscripts.org/scripts/source/60315.user.js
// ==/UserScript==
var bookAmazonClass={
 "baseWebsite":"http://www.amazon.com/*",
 "implements":"http://userscripts.org/scripts/source/60315.user.js",
 "scrapers":[
  {"scrapedConcept":"Book","XPath":"//body[@class='dp']",
   "attributeScrapers":[
    {"scrapedAttribute":"title",
     "XPath":"//span[@id='btAsinTitle']"},
    {"scrapedAttribute":"author",
     "XPath":"//div[@class='buying']/span/a"},
    {"scrapedAttribute":"isbn",
     "function":function(book){
      var isbnNode=document.evaluate(
       ".//td[@class='bucket']/div[@class='content']/ul/li[4]",book,
null,XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null).snapshotItem(0);
      return isbnNode.innerHTML.match(/[0-9]{10}/)[0];}},
    {"scrapedAttribute":"price",
     "function":function(book){
      var price=document.evaluate(".//b[@class='priceLarge']",book,
null,XPathResult.UNORDERED_NODE_SNAPSHOT_TYPE,null).snapshotItem(0);
      return price.innerHTML.match(/[0-9]+(\.[0-9]+)?/)[0];}}]}]
}
window.registerScriptingClass(bookAmazonClass);
```

Fig. 5.3.6 – Class Script

En este punto, la interfaz para el script de [27] o en nuestro caso el concepto abstracto para nuestros scripts, solo contienen la descripción de una interfaz relacionada a un concepto

particular. Es necesario, implementar esta interface basada en una aplicación Web especifica. La cual, contendrá el mapeo que indicara como los conceptos de las interfaces son obtenidos, junto con sus respectivos atributos, desde las representaciones circunstanciales de estos conceptos, que se encuentran contenidas dentro de la implementación de una aplicación Web especifica.

En [27], esto se realiza a través de los Class Scripts, que justamente implementan una interfaz, basada en un sitio Web específico. En la figura de arriba (Fig. 5.3.6), puede verse la implementación de la interface del libro para la aplicación Web "Amazon.com". Esta mapea el concepto que representa la interfaz y sus atributos, sobre la estructura de la aplicación Web, empleando direcciones XPath. Por nuestra parte, nosotros implementamos la interfaz de una aplicación Web particular, mediante la construcción de wrappers o como los conoce nuestro framework, mediante los conceptos concretos.

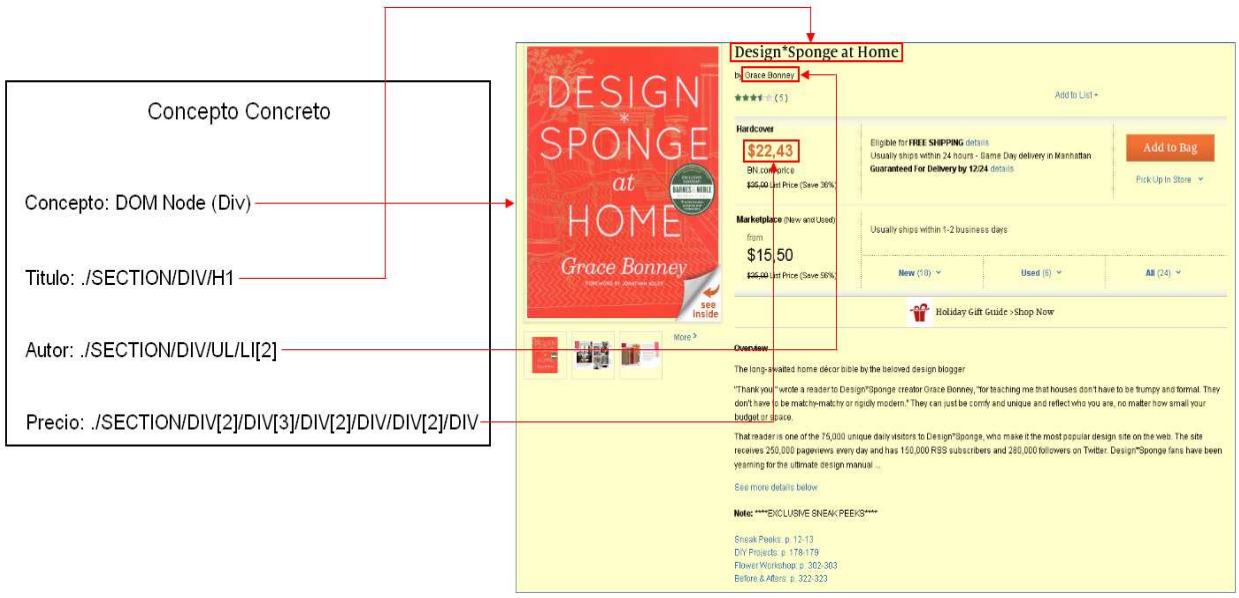

Fig. 5.3.7 – Implementación de un concepto concreto

Anteriormente, ya vimos como estos wrappers eran construidos como parte de nuestro proceso de extracción de datos, por lo que no entraremos en detalle sobre ese tema. En la Fig. 5.3.7, se observa un concepto concreto construido para la aplicación web Barnes&Nobles. Al igual que en la implementación de la interface de [27], nuestro concepto concreto también utiliza las direcciones XPath para mapear los atributos o propiedades de los conceptos. Es más, nuestros conceptos concretos, también especifican las propiedades utilizando direcciones XPath relativas, tomando como punto de referencia la estructura que representa el

concepto, en la aplicación Web. Por otro lado, a diferencia de [27], nosotros no mapeamos el concepto utilizando también el XPath, sino que lo hacemos a trabes del nodo del DOM que lo contiene. Más específicamente, a través del sub árbol que representa este nodo dentro del documento Web. Esto, esta implementado así, debido a nuestro algoritmo de detección que vimos en la sección anterior.

Si bien, el Class Script de [27], tiene el mismo objetivo básico que nuestros conceptos concretos, existen una serie de pequeñas diferencias, que desde el punto de vista de nuestro framework, podrían considerarse como una serie de ventajas a la hora de desarrollar las adaptaciones/scripts, que los usuarios utilizan para modificar las aplicaciones Web. En primer lugar, al dar un simple vistazo al código exhibido por el Class Script (implementado para un libro de Amazon.com en la Fig. 5.3.6), podemos ver que este código es mucho más "enredado" o complejo que nuestra implementación de la interfaz. Esto es debido a que este Class Script, debe ser construido a mano (es decir línea por línea), por los propios usuarios, quienes desarrollan los scripts. Por lo tanto, esta implementación, no solo necesita contener la especificación de cómo localizar, tanto el concepto que se encuentra encapsulado en la interfaz, como además, cada uno de los atributos que contiene. Sino que, también, los usuarios deben especificar, en esta interfaz, como extraer estos atributos desde la estructura de la aplicación Web. Todo esto hace que la implementación de la interfaz, sea un poco más sensible a los cambios que nuestra propia implementación. Esto es debido, a que nosotros, solo especificamos en la implementación de la interfaz, como ubicar las propiedades de los conceptos y el concepto en sí. De esta forma, si la estructura de la aplicación cambia, solo se necesita redefinir una línea de código bastante simple, como es el XPath para el título del libro en la Fig. 5.3.7 o en el peor de los casos, volver a capturar el nodo del DOM que representa el concepto. Sin embargo, en los Class Script, esto es un poco más complejo. Por ejemplo, si cambiara no solo la ubicación del título dentro del cuerpo del concepto, sino que además, la función o operación que utiliza para extraer esta información del documento DOM, debería reescribir algunas líneas de código un poco mas complejas (en el caso del título, por ejemplo, tendría que redefinir la función que utiliza en la línea 24 de la Fig. 5.3.6).

Por otra parte, nosotros separamos al usuario que desarrolla los scripts, de aquellos que los utilizan. Si bien es cierto, que el usuario que desarrollo estos scripts, podría terminar siendo el mismo que los utiliza después, esta separación que hace nuestro framework, viene mas por el lado de que los usuarios solo necesitan empeñar la mayor parte de su esfuerzo en desarrollar sus scripts, ósea solo aquello que estos scripts han de agregar o modificar sobre la estructura de la aplicación Web. Con respecto a la interfaz, solo necesitan especificar el concepto abstracto y sus propiedades. El resto de la implementación, como ya vimos, se construye usando nuestra herramienta, sin que el usuario que creó el script tenga que escribir una sola línea de código.

Otra de estas pequeñas diferencias, que ayuda a crear una interfaz más flexible, ante los cambios en la estructura de las aplicaciones Web, es el hecho de que nosotros (como explicamos antes), utilizamos el nodo del DOM, para definir al concepto en la implementación de esta interfaz. De esta forma, si el concepto cambia de lugar, no es necesario redefinir la implementación de la interfaz. Esto es gracias a nuestro algoritmo de

detección. Por el contrario, en los Class Script, al usar el XPath para definir el concepto es necesario redefinir esta dirección, ya que el conjunto de etiquetas HTML que indican el camino hacia el mismo ha cambiado y por lo tanto, ya no puede utilizarse para extraerlo.

Capitulo 6. Caso de estudio

Ahora que hemos terminado de explicar cada una de las partes, que hacen que nuestro framework funcione y las herramientas que ofrece para facilitar el desarrollo de adaptaciones reutilizables, pasaremos a un ejemplo práctico, donde mostraremos la efectividad completa del framework, en funcionamiento, para lograr mejorar la experiencia de navegación del usuario, a través de adaptaciones reutilizables en familias de aplicaciones Web.

6.1 Portales de noticias

Para este ejemplo práctico, hemos tomado como caso estudio dos aplicaciones Web, que ofrecen a los usuarios artículos y noticias diarias sobre la actualidad del mundo. Con lo cual, podría decirse, que estas aplicaciones pertenecen a la familia de aplicaciones Web, llamada "Portales de noticias". En la figura de abajo, se ven las dos aplicaciones seleccionadas para la prueba, una de ellas es Infobae.com y la otra es Ambito.com.

Fig. 6.1.1 – Portales de noticias

Para que estas aplicaciones Web, puedan pertenecer a una familia de aplicaciones en común, es mandatorio que compartan, al menos, un "concepto" en común. Siendo que ambos, son portales de noticias, el concepto "noticia" resulta más que satisfactorio. Entonces, esta noticia, será el concepto abstracto que utilizan y conectara, a estas dos aplicaciones. Ahora, es necesario, definir los atributos abstractos para este concepto "noticia", pues los necesitaremos para poder utilizar nuestras adaptaciones sobre estos mismos sitios Web.

Fig. 6.1.2 – Noticias

En general, las propiedades de un concepto abstracto, son aquellas que con frecuencia se atribuyen a la entidad, por ejemplo, si se piensa en un auto, las propiedades que seguramente saltan a la mente son las “ruedas”, el “volante,” el “motor”, etc. En la Fig. 6.1.2, pueden verse dos noticias típicas, que pueden llegar a exhibir cada una de las aplicaciones Web. Viendo estas noticias, podemos deducir, que dos propiedades abstractas básicas, que pueden ser incluidas dentro del concepto, son el titulo y la descripción.

6.2 Adaptaciones

Conocemos el concepto abstracto, que comparten estas aplicaciones Web, y la familia a la que pertenecen, es hora de crear los script o adaptaciones que se aplicaran para mejorar estas aplicaciones. Nosotros aquí, presentaremos tres adaptaciones, que pueden ser útiles en aplicaciones Web de portales de noticias. Sin embargo, recuerde que el usuario puede desarrollan cualquier clase de script para cambiar, adaptar o modificar los conceptos de cualquier forma que le parezca. El alcance de las adaptaciones, solo está limitado por la imaginación del usuario. Podría crear adaptaciones para exportar las noticias, resaltarlas, compartirlas, eliminarlas, etc.

A continuación explicaremos, con un breve detalle, estas tres adaptaciones.

6.2.1 YouTube Adapter

Esta adaptación, inserta un video de YouTube dentro del espacio de la notica. Este video, es buscado automáticamente por el script, utilizando la API [53] provista por YouTube y la tecnología de Jquery [29]. Nótese, que para que esto funcione, el script necesita esta librería, por lo tanto la agrega al ámbito de la aplicación web, nuestro framework tampoco restringe a los scripts sobre el uso de estas inclusiones.

Fig. 6.2.1.1 – YouTube Adapter

Aquí, se puede ver, esta adaptación funcionando en la sección de tecnología de la aplicación Web Infobae.com. Esta adaptación no agrega el video directamente, ya que es costoso, en tiempo de procesamiento y ancho de banda la búsqueda de videos, imagine que si hay cientos de noticias en un sitio Web, al tratar de cargar todos los videos a la vez tildaría la aplicación. Por lo tanto, primero se agrega un icono, luego al hacer click sobre este icono se despliega el

video. La búsqueda que se realiza para este video, se encuentra relacionada en base al título de la noticia.

Fig. 6.2.1.2 – YouTube Video

En esta figura, puede verse el resultado de la adaptación. Si se hace click nuevamente sobre el icono de YouTube, el video es ocultado.

6.2.2 I Like Adapter

Esta adaptación, es una reutilización del ya tan conocido “I Like”, de la red social Facebook [21]. Básicamente, esta adaptación, agrega el botón del “I Like”, para que se pueda publicar en el muro de una cuenta de Facebook con solo un click. Además, agrega un botón para comentar la publicación de la notica (“send”).

Fig. 6.2.2.1 – I Like Adapter

En la Fig. 6.2.2.1, puede verse la adaptación funcionado. La misma agrega, los dos botones que explicamos antes y al cliquear sobre el botón de “Like” despliega una ventana, para agregar un comentario y poder finalmente postear la publicación en el muro de Facebook.

6.2.3 Google Plus Adapter

Por último, el tercer adaptador, al igual que el “I Like” de Facebook, tiene como objetivo la difusión de información a través de las redes sociales. En particular, este está destinado a la nueva red social de Google, conocida como Google+ [24].

Fig. 6.2.3.1 – Google+ Adapter

Puede verse, como esta adaptador también agrega un icono, en el cual al cliquear sobre él, se postea la noticia en la red social de Google+.

Estas adaptaciones, recién presentadas, si bien pueden llegar a parecer un poco comunes, el objetivo con ellas es mostrar la clase de scripts, que se pueden llegar a desarrollar utilizando nuestro framework. Con lo cual, la idea de si son o no innovadoras, se encuentra fuera de la intención que se intenta alcanzar. Recuerde, que nuestro framework, presenta un nuevo método para la construcción de adaptaciones reutilizables, el hecho de que las mismas sean o no innovadores, viene por parte del mismo usuario que las desarrolla.

Entonces, ahora que hemos desarrollado las adaptaciones, y el concepto abstracto junto con sus propiedades, es hora de construir los conceptos concretos. Para esto, nuevamente utilizaremos nuestra herramienta de highlighting. Anteriormente, ya vimos, como este proceso es completado, por lo tanto aquí no lo vamos a volver a analizar. Lo que nos interesa, es el resultado del proceso de extracción de datos, que son los conceptos concretos.

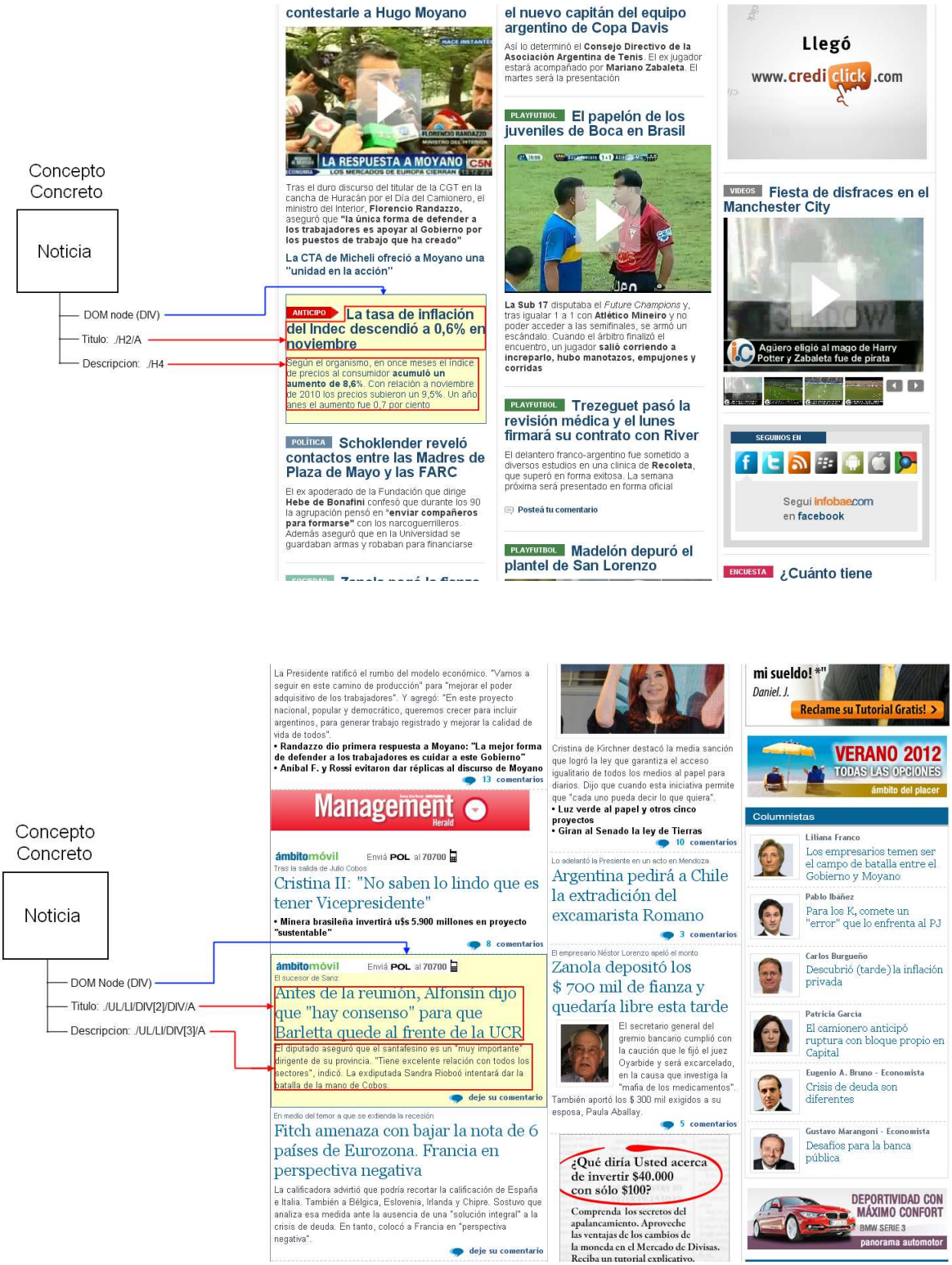

Fig. 6.1.3 – Conceptos Concretos

Recuerden, que estos conceptos concretos (wrappers), mapean la interfaz presentada a los scripts, directamente sobre la estructura del documento DOM de la aplicación web. En este caso, construimos los conceptos concretos necesarios para mapear la interfaz (Fig. 6.1.3), sobre la aplicación Web Infobae.com (la de arriba) y Ambito.com (la de abajo).

Finalmente, en la figura de la página siguiente (Fig. 6.1.4), podemos ver como el modelo del framework quedaría armado para estas dos aplicaciones web. Este se encuentra divido en dos grandes regiones, la parte de arriba corresponde a los usuarios experimentados que desarrollan las adaptaciones y la de abajo a los usuarios finales quienes utilizan estas adaptaciones y construyen los conceptos concretos usando las herramienta que proporciona el mismo framework.

Podemos ver, como en la parte de los desarrolladores, se encuentran tres abstracciones principales (marcadas en rojo), que presenta el modulo de adaptación de familias y que ayudan al usuario a construir estas adaptaciones mediante scripts, que se abstraen de la implementación de las aplicaciones Web. Aquí, también, podemos ver las tres adaptaciones de las que hablamos antes (“I Like Adapter”, “Google+ Adapter” y “YouTube Adapter”), donde cada una desarrollada como una adaptación, contiene en su interior (a bajo nivel), el script que agrega las nuevas funcionalidades y contenido a las aplicaciones.

Para este modelo particular, se tiene una familia de aplicaciones Web llamada “Portales de noticias”, que conoce un conjunto de aplicaciones web (Infobae.com y Ambito.com). Estas aplicaciones web al mismo tiempo, comparten un concepto en común (o podrían ser varios), el cual se conoce como “Noticia”. Esta, tiene un conjunto de atributos (o propiedades abstractas), que la definen como tal (“titulo” y “descripción”).

Por otra parte, en el lado que pertenece al usuario final y que se corresponde con el modulo de extracción de datos, podemos ver los conceptos concretos creados para cada aplicación Web de la familia. Estos son los “templates”, que se asocian a los conceptos concretos. También, estos conceptos concretos, son los que utilizan, finalmente, los scripts de las adaptaciones para insertar los nuevos contenidos y funcionalidades.

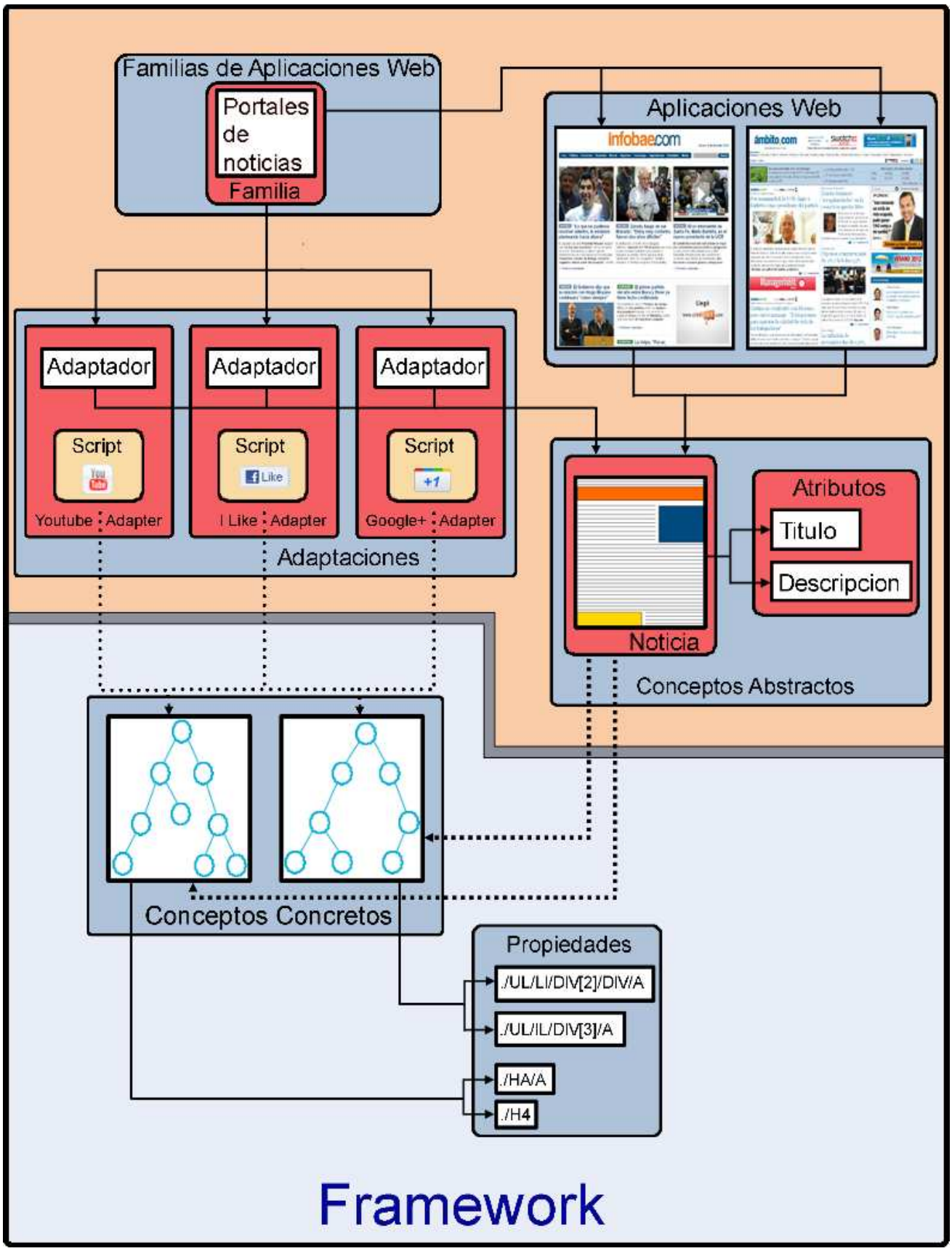

Fig. 6.1.4 – Modelo del Framework

Si bien, la abstracción, que representa a las aplicaciones Web en el modelo y que pertenece al modulo de adaptación de familias, puede ser creada a mano por los usuarios que desarrollan las adaptaciones, es decir, los usuarios mismos podrían definir un conjunto predefinido de aplicaciones Web, en las cuales los conceptos que ellos piensan modificar o mejorar, seguramente se encuentran presentes y por lo tanto, las adaptaciones pueden ser insertadas. Nosotros, apuntamos, a que estas abstracciones sean creadas por los usuarios finales, los cuales a medida que exploran el universo de la Web, pueden ir agregando nuevas aplicaciones Web a una familia, con solo unos pocos clics.

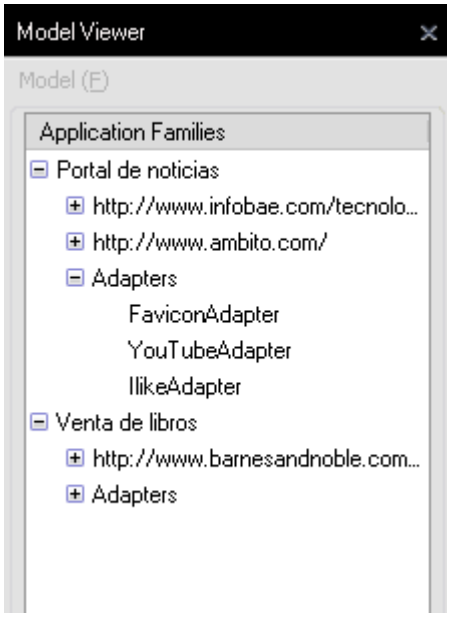

Fig. 6.1.5 – Model Viewer

Nuevamente, nuestro framework, ofrece una interfaz simple (implementada como un panel lateral del navegador), que proporciona una vista básica del modelo, que presentan las familias de aplicaciones y al mismo tiempo, permite agregar una nueva aplicación a una familia particular. En este caso, el ejemplo de arriba, presenta dos familias de aplicación web, para mostrar que es posible agregar una nueva aplicación a cualquiera de ellas. Una de estas familias, pertenece a las aplicaciones de libros que vimos antes. La otra, es la familia de "Portales de noticias" que venimos viendo en el modelo. En esta, pueden verse las dos aplicaciones web, "Infobae.com" y "Ambito.com", que pertenecen a la familia y las adaptaciones a asociadas a la misma.

Para agregar la aplicación Web, que el usuario se encuentra navegando en ese momento a una familia, solo tiene que cliquear con el botón derecho del mouse, sobre la familia a la que se desea agregar y seleccionar la opción "New Application". Luego la aplicación aparecerá listada en la respectiva familia y pasara a formar parte de ella.

Para terminar, podemos ver, como quedarían las tres adaptaciones en las aplicaciones Web (Infobae.com y Ambito.com), de la familia de “Portales de noticias”. Fíjese, como cada una, de las adaptaciones son insertadas en la estructura de de la aplicación web, dentro del espacio que le corresponde a cada concepto, en este caso, a cada “Noticia”.

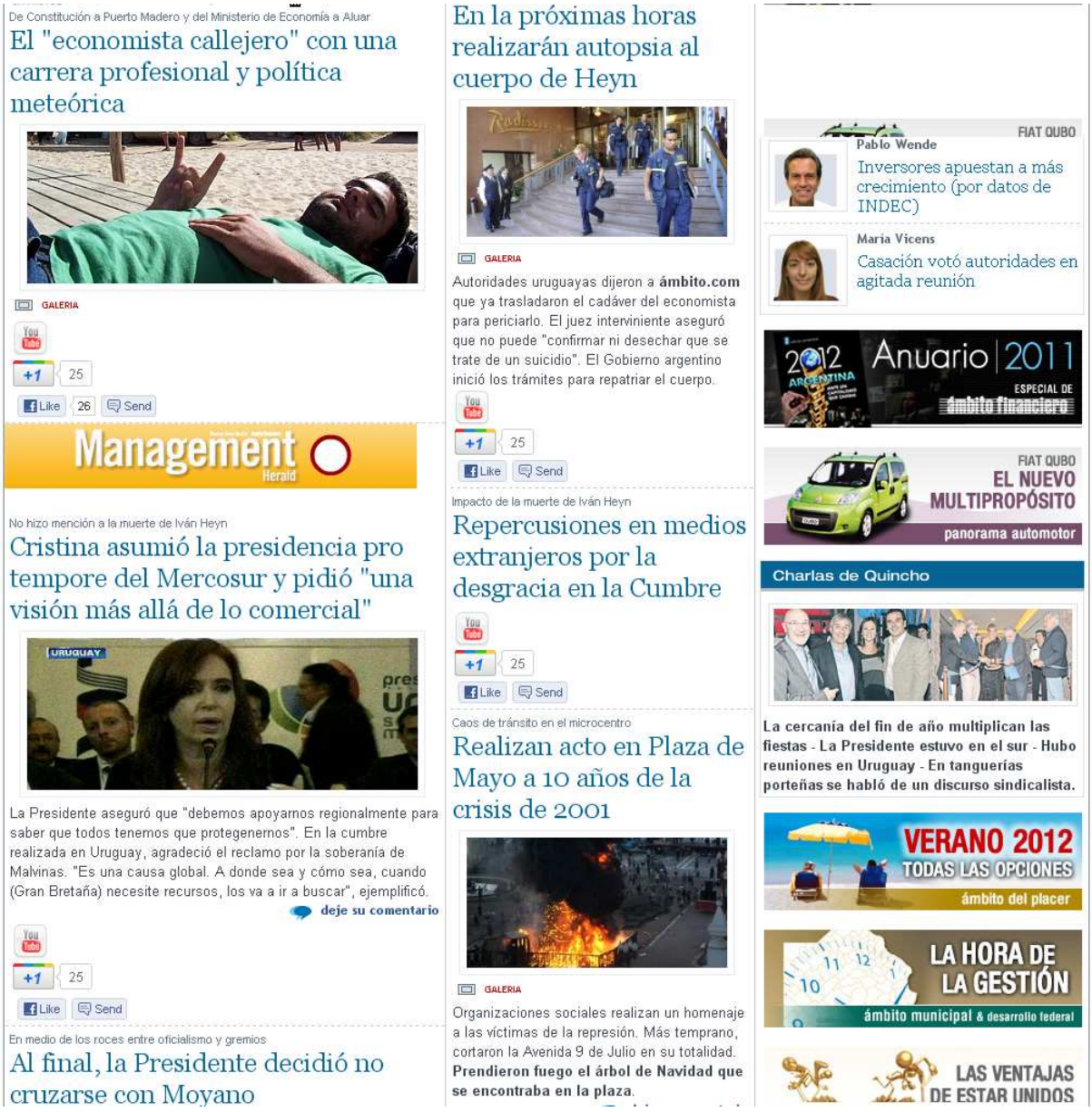

Fig. 6.1.6 – Adaptaciones en Ambito.com

Fig. 5.1.7 – Adaptaciones en Infobae.com

Capitulo 7. Conclusión y trabajo futuro

En esta tesis se introdujo un framework, construido como una extensión de un navegador, que trata de proveer a los usuarios finales, de una mayor personalización de la Web que utilizan y así lograr, que experimenten una mejor experiencia, al navegar a través de los sitios de la misma. Este se encuentra desarrollado, entorno, a la idea de la utilización de adaptaciones, que se ejecutan del lado del cliente, es decir, dentro del espacio del navegador. Estas adaptaciones, son las que finalmente, utilizan los usuarios como medio para mejorar las aplicaciones Web. El framework, ayuda a los usuarios en la construcción de estas adaptaciones, introduciendo técnicas e ideas de diseño, que les permiten a las mismas, no solo, ser más resistentes a los cambios de las aplicaciones Web, sino que además, poder ser reutilizadas en familias de aplicaciones Web.

El framework y la tesis en conjunto, pueden ser utilizados como una guía, para ayudar a desarrollar, aun más, las técnicas y tecnologías empleadas, con el fin, de mejorar la Web por medio de adaptaciones que se inyectan en la estructura de aplicaciones Web existentes. Todo, para lograr obtener una mejor experiencia al utilizar las mismas. Incluso, el framework mismo, puede ser mejorado llevándolo a un potencial aun mayor. Podría extenderse, por ejemplo, para poder ser utilizado en una variedad más amplia de familias de aplicaciones Web y esto con solo unos pocos recursos. Seria posible, integrar soporte para construir adaptaciones que pueden ser introducidas, incluso en aplicaciones que utilizan tecnologías modernas como flash o AJAX.

También, es necesario, tener en cuenta las limitaciones del framework. Un punto importante y que no tratamos aquí, es el efecto secundario de aplicar adaptaciones sucesivas en una misma aplicación Web. Esto podría llevar a que las adaptaciones se interpongan entre sí, afectando el funcionamiento correcto de las mismas. Por ejemplo, una adaptación previa, podría remover una etiqueta HTML o cambiar la estructura del DOM, de alguna forma, para un nodo que otra adaptación, que se aplique posteriormente, utiliza. Esto, obviamente, llevaría a que la adaptación no funcione adecuadamente. Aquí, las futuras investigaciones, podrían llevar a crear técnicas que prevengan este tipo de problemas.

La Web continuara evolucionando de aquí en adelante y con ella, nuevas tecnologías, técnicas, conceptos e ideas nacerán. Esta tesis, puede ser vista, como un escalón en la evolución de la Web, que podría servir de base para ayudar a desarrollar estas nuevas tecnologías, técnicas, conceptos e ideas, haciendo de la futura Web, una cada vez más centrada en el usuario, más personal y con más interacción.

Capitulo 8. Bibliografía

[1] – Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, Juliana S. Teixeira: A Brief Survey of Web Data Extraction Tools. SIGMOD Record (SIGMOD) 31(2):84-93 (2002)

[2] – Ana M. D. Moreira, João Araújo, Awais Rashid: A Concern-Oriented Requirements Engineering Model. CAiSE 2005:293-308.

[3] – Sergio Firmenich, Marco Winckler, Gustavo Rossi, Silvia E. Gordillo: A Framework for Concern-Sensitive, Client-Side Adaptation. ICWE 2011:198-213.

[4] – Nitin Jindal, Bing Liu: A Generalized Tree Matching Algorithm Considering Nested Lists for Web Data Extraction. SDM 2010:930-941.

[5] – Webber, Matthew J: A Stateful Web Augmentation Toolkit. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, February 2010.

[6] – Brad Adelberg: NoDoSE - A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents. SIGMOD 1998:283-294.

[7] – Davi de Castro Reis, Paulo Braz Golgher, Altigran Soares da Silva, Alberto H. F. Laender: Automatic web news extraction using tree edit distance. WWW 2004:502-511.

[8] – BookInfoLine. Compare book prices from various book stores. By EastWoodReaders, Last update May 10, 2011 http://userscripts.org/scripts/show/65482

[9] – Arnaud Sahuguet, Fabien Azavant: Building intelligent Web applications using lightweight wrappers. Data Knowl. Eng. (DKE) 36(3):283-316 (2001).

[10] – Sergio Firmenich, Silvia E. Gordillo, Gustavo Rossi, Marco Winckler: Client-Side Adaptation: An Approach Based in Reutilization Using Transversal Models. ICWE Workshops 2010:566-570.

[11] – Bent Bruun Kristensen, Kasper Østerbye: Roles: Conceptual Abstraction Theory and Practical Language Issues. TAPOS 2(3):143-160 (1996).

[12] – Jocelyne Nanard, Gustavo Rossi, Marc Nanard, Silvia E. Gordillo, Leandro Perez: Concern-Sensitive Navigation: Improving Navigation in Web Software through Separation of Concerns. CAiSE 2008:420-434.

[13] – Craigslist. www.craigslist.com/.

[14] – Wang, J.-Y., and F. Lochovsky. Data extraction and label assignment for Web databases. WWW 2003.

[15] – DeviantART: where ART meets application! www.deviantart.com/

[16] – DOM Node. https://developer.mozilla.org/en/DOM/node

[17] – World Wide Web Consortium (W3C). Document Object Model XPath - http://www.w3.org/TR/DOM-Level-3-XPath/xpath.html

[18] – David F. Huynh, Robert C. Miller, David R. Karger: Enabling web browsers to augment web sites' filtering and sorting functionalities. UIST 2006:125-134.

[19] – Sergio Firmenich, Gustavo Rossi, Matias Urbieta, Silvia E. Gordillo, Cecilia Challiol, Jocelyne Nanard, Marc Nanard, João Araújo: Engineering Concern-Sensitive Navigation Structures, Concepts, Tools and Examples. J. Web Eng. (JWE) 9(2):157-185 (2010).

[20] – World Wide Web Consortium (W3C). Extensible Markup Language (XML) - http://www.xml.com/

[21] – Facebook. http://www.facebook.com/

[22] – Frames in HTML documents. www.w3.org/TR/html4/present/frames.html

[23] – Qiang Hao, Rui Cai, Yanwei Pang, Lei Zhang: From one tree to a forest: a unified solution for structured web data extraction. SIGIR 2011:775-784.

[24] – Google+. https://plus.google.com/

[25] – GreaseMonkey: Power-ups for your browser. www.userscripts.org/

[26] – Robert Ennals, Eric A. Brewer, Minos N. Garofalakis, Michael Shadle, Prashant Gandhi: Intel Mash Maker: join the web. SIGMOD Record (SIGMOD) 36(4):27-33 (2007)

[27] – Oscar Díaz, Cristóbal Arellano, Jon Iturrioz: Interfaces for Scripting: Making Greasemonkey Scripts Resilient to Website Upgrades. ICWE 2010:233-247.

[28] – HyperText Markup Language (HTML). http://www.w3.org/html/

[29] – JQuery (The Write Less, Do More, JavaScript Library). http://jquery.com/

[30] – JSON (JavaScript Object Notation). http://www.json.org/.

[31] – Oscar Díaz, Cristóbal Arellano, Jon Iturrioz: Layman tuning of websites: facing change resilience. WWW 2008:1127-1128.

[32] – LinkedIn, World's Largest Professional Network. www.linkedin.com/

[33] – Mozilla Developers. https://developer.mozilla.org

[34] – Mozilla Firefox Web Browser. www.mozilla.org/en-US/firefox/new/

[35] – D. L. Parnas. On the criteria to be used in decomposing systems into models. Communications of the ACM, 15:1053-1058, 1972.

[36] – Daniel Schwabe, Rita de Almeida Pontes, Isbela Moura: OOHDM-Web: an environment for implementation of hypermedia applications in the WWW. Association for Computing Machinery (ACM): Volume 8, Issue 2, June 1999.

[37] – ownerDocument. https://developer.mozilla.org/En/DOM/Node.ownerDocument

[38] – Photobucket: Image hosting, free photo sharing & video sharing. www.photobucket.com/

[39] – Mary Elaine Califf, Raymond J. Mooney: Relational Learning of Pattern-Match Rules for Information Extraction. AAAI/IAAI 1999:328-334

[40] – Show Just Image 3. http://www. http://userscripts.org/scripts/show/109890. Last seen in 18/11/2011 . Author untamed0**.**

[41] – Yanhong Zhai, Bing Liu: Structured Data Extraction from the Web Based on Partial Tree Alignment. IEEE Trans. Knowl. Data Eng. (TKDE) 18(12):1614-1628 (2006).

[42] – World Wide Web Consortium (W3C). The Document Object Model (DOM) API - http://www.w3.org/DOM/

[43] – Andrew W. Hogue, David R. Karger: Thresher: automating the unwrapping of semantic content from the World Wide Web. WWW 2005:86-95.

[44] – Lerman, K., L. Getoor, S. Minton, and C. Knoblock. Using the structure of Web sites for automatic segmentation of tables. SIGMOD 2004.

[45] – Zhai, Y., and B. Liu. Web data extraction based on partial tree alignment. WWW 2005.

[46] – Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks (CN) 33(1-6):137-157 (2000).

[47] – Gustavo O. Arocena, Alberto O. Mendelzon: WebOQL: Restructuring Documents, Databases, and Webs. TAPOS 5(3):127-141 (1999)

[48] – Wikipedia , the free encyclopedia that anyone can edit.. www.wikipedia.org/

[49] – World Wide Web Consortium (W3C). www.w3.org/

[50] – World Wide Web Consortium (W3C). XML Path Language (XPath) Version 1.0 – W3C Recommendation, 1999. http://www.w3.org/TR/xpath/

[51] – Ling Liu, Calton Pu, Wei Han: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. ICDE 2000:611-621.

[52] – Yelp, Real people, Real reviews. www.yelp.com/

[53] – YouTube APIs and Tools.
http://code.google.com/apis/youtube/getting_started.html#data_api