Arquitectura para *Crawling* Dirigido de Información Contenida en la Web Oculta

Manuel Álvarez Díaz

TESIS DOCTORAL

Directores:

Dr. Carlos Alberto Pan Bermúdez

Dr. Fidel Cacheda Seijo

Departamento de Tecnoloxías da Información e as Comunicacións
UNIVERSIDADE DA CORUÑA

A Coruña, Diciembre 2007

A mis padres, Estrella y Amador,
porque a ellos les debo todo y se merecen todo

A MI hermana, Hortensia,
por ayudarme a creérmelo

AGRADECIMIENTOS

Todavía recuerdo el primer día. Mucho tiempo ha pasado desde entonces. Multitud de ideas pasaron por nuestras mentes y muchos fueron los escarceos investigadores necesarios hasta la definición concreta de los objetivos que han dado lugar a este trabajo. Un buen día, la sabiduría escondida tras una profunda capa de timidez vio la luz. Allí estaba, dormitando, la inmensidad oculta de la Web, esperando su oportunidad. Así fue como surgió la idea y me embarqué en esta apasionante aventura. Por ella comienzo mis agradecimientos, porque sin ella, todo habría sido diferente.

Pero no menos diferente habría sido sin el inestimable apoyo y colaboración de aquellas personas que me han acompañado en este viaje, que han vivido y compartido mis inquietudes y temores en los momentos más complicados. A ellos quiero dedicar mis agradecimientos.

En primer lugar, a mis codirectores, Alberto Pan y Fidel Cacheda. A Alberto, por su valiosa ayuda diaria, de principio a fin, por haber sabido enfocar la línea de trabajo en los momentos de duda y por proporcionarme los medios y las ideas iniciales en las que he basado el esfuerzo posterior. A Fidel, por su ayuda en la realización de este trabajo, sus importantes comentarios y su apoyo.

Agradezco también a Ángel Viña su apoyo y confianza durante todos estos años y sus útiles consejos en la preparación de este trabajo.

A Juan Raposo y Justo Hidalgo, compañeros de tripulación tanto tiempo, compartiendo momentos de tempestad y calma. A Juan Raposo desde la proximidad, por sus sugerencias y apoyo día a día. A Justo Hidalgo, *allende los mares*, por sus ánimos y consejos. ¡Tierra a la vista! Pronto nos reuniremos ;).

Quiero expresar también mi agradecimiento a Jacob Fernández y Oscar Arias, por sus contribuciones en la implementación del prototipo desarrollado para probar las técnicas presentadas. En particular, Jacob Fernández ha implementado las técnicas de *crawling* de formularios en el marco de su proyecto de fin de carrera de Ingeniería Informática de la Universidade da Coruña.

También debo mis agradecimientos al resto de compañeros, del Departamento de Tecnoloxías da Información e as Comunicacións de la Universidade da Coruña, y de DENODO Technologies, con los que he compartido escalas de mi travesía y me han ayudado a afinar el rumbo. En especial a Paula Montoto y Hoki Torres.

A mis amigos, compañeros de rutas, aquellos con quienes surco los mares, y conozco parajes increíbles, por seguir estando ahí.

Al mar y las estrellas, por sus momentos de calma y reflexión.

Por último, no puedo más que agradecer a quienes he dedicado esta tesis doctoral su apoyo y ánimos en todo momento. Sin ellos, esto no habría sido posible.

A todos ellos, Gracias

RESUMEN

La WWW (World Wide Web) constituye el mayor repositorio de información distribuida y heterogénea jamás construido. En la Web, como en cualquier biblioteca, tan importante como el almacenamiento de la información es disponer de un sistema que permita localizar, acceder y recopilar la que satisface las necesidades de un usuario.

La aproximación utilizada más habitualmente para recopilar y localizar información en Internet la constituyen los buscadores basados en técnicas de *crawling*. Los *crawlers* son programas software capaces de recorrer la Web automáticamente, recopilando las páginas accedidas para construir un índice que permita búsquedas sobre su contenido.

Pueden distinguirse dos tipos de estrategias de *crawling*: *crawling global* y *crawling dirigido*. Las tareas de *crawling dirigido* están orientadas a un propósito específico. Este tipo de tareas suelen aparecer en el ámbito corporativo y presentan requisitos más complejos que los que pueden satisfacerse con los buscadores convencionales.

Sin embargo, los *crawlers* actuales, tanto globales como dirigidos, sólo pueden acceder a la parte de la Web que se encuentra publicada y enlazada como páginas estáticas. Aunque estas páginas representan una gran cantidad de información, constituyen sólo una pequeña porción de toda la información web disponible. Existe gran cantidad de información que es generada dinámicamente por un servidor en respuesta a acciones del usuario. El ejemplo más paradigmático lo constituyen las páginas generadas como respuesta a una consulta efectuada por un usuario sobre un formulario web. A esta porción de la Web suele denominársele 'Web Oculta' (*Hidden Web*) o 'Web Profunda' (*Deep Web*).

Una gran parte de la información contenida en la Web Oculta tiene una estructura latente. Estudios recientes estiman que el 77% de las fuentes de la Web Oculta ofrecen formularios que permiten ejecutar consultas sobre una base de datos subyacente y devuelven los resultados obtenidos codificados en HTML. Por lo tanto, otro de los retos a los que se enfrenta el tratamiento de la información contenida en la Web Oculta es inferir la estructura subyacente presente en los datos de este tipo de páginas.

El objetivo de la presente tesis doctoral es definir una arquitectura y un conjunto de técnicas que posibilite tareas de *crawling* dirigido con acceso a información que se encuentra en la Web Oculta. Las principales contribuciones de este trabajo son las siguientes: (1) una arquitectura que contempla todos los pasos necesarios para construir de forma efectiva aplicaciones de *crawling* dirigido capaces de acceder a la Web Oculta, (2) un conjunto de técnicas y algoritmos para realizar *crawling* de la llamada 'Web Oculta del lado cliente', que se refiere a las páginas no accesibles a los *crawlers* convencionales debido a complejidades como *Ajax* y/o el uso de lenguajes de *script*, (3) un conjunto de técnicas y algoritmos para identificar y aprender a consultar automáticamente formularios de consulta web relevantes para una tarea de *crawling* dirigido especificada, (4) nuevas técnicas y algoritmos para extraer automáticamente los datos estructurados contenidos en las respuestas obtenidas a consultas efectuadas sobre formularios web.

ABSTRACT

The WWW (World Wide Web) is the biggest heterogeneous and distributed information repository that has ever been built. In the Web, as in any library, as important as information storage is to have efficient means for finding, accessing and retrieving the information satisfying the user needs.

The most frequently used approach to retrieve and find information from the Internet is by search engines based on crawling techniques. Crawlers are software programs that go through the Web automatically, compiling the accessed pages to build an index that allows queries about its content.

Two crawling strategies can be defined: global crawling and focused crawling. Focused crawling tasks are addressed to a specific purpose. This type of tasks usually appears in corporate environments and it shows more complex requirements that the ones addressed by conventional search engines.

Nevertheless, current crawlers, both global and focused, only can access to the part of the Web that is published and linked as static pages. Although these pages constitute a large amount of information, they are only a small portion of all the available information on the Web. Great amounts of information are dynamically generated by a server as a response to the user actions. The pages that are obtained in response to user queries submitted on a web query-form are the more paradigmatic example. This part of the Web is usually known as 'Hidden Web' or 'Deep Web'.

A large amount of the information contained in the Hidden Web has a latent structure. Recent studies estimate that about 77% of hidden web sources have forms allowing to execute queries over an underlying database and to return the obtained results encoded in HTML. So, dealing with the information contained in the Hidden Web tackles another challenge that is to infer the underlying structure of the data included in this type of pages.

The goal of this doctoral thesis is to define an architecture and a set of related techniques to allow focused crawling tasks for accessing to information located in the Hidden Web. The main contributions of this work are the following ones: (1) an architecture which takes into account all the steps needed to build focused crawling applications for accessing to the Hidden Web, (2) a set of techniques and algorithms to crawl the so-called 'client side Hidden Web', that is the set of pages that can not be accessed from conventional crawlers due to complexities like Ajax and/or scripting languages, (3) a set of techniques and algorithms to identify and learn to query relevant web forms for a specific focused crawling task automatically, (4) new techniques and algorithms to extract structured data from pages which are automatically obtained as response to web query-forms queries.

Índice de Contenidos

Índice de Figuras

Índice de Tablas

I. PLANTEAMIENTO Y CONTRIBUCIONES

I.1. ÁMBITO

I.1.1. LA WEB OCULTA

'La Web es el fenómeno más importante de Internet, demostrado por su crecimiento exponencial y su diversidad. Por su volumen y riqueza de datos, los buscadores de páginas se han convertido en una de las herramientas principales.

La Web tiene actualmente al menos unos cuatro mil millones de páginas estáticas y un número cientos de veces mayor de páginas dinámicas (aquellas que sólo se crean producto de un clic o de una consulta en un sitio Web).'

Web Mining. Ricardo Baeza-Yates. 2004

Internet o más concretamente la WWW (World Wide Web), constituye actualmente el mayor repositorio de información distribuida y heterogénea jamás construido. Puede considerarse como la mayor biblioteca de todos los tiempos, tanto por los conocimientos contenidos en ella, como por su disponibilidad (desde cualquier lugar del mundo y sin restricciones horarias) y accesibilidad (número de usuarios totales y concurrentes que permite). Ambas características se ven potenciadas por su carácter distribuido, que permite que todos sus usuarios puedan contribuir a darle forma y contenido. Por otra parte, sus contenidos no se limitan al formato textual, sino que constituye el sustrato para intercambiar cualquier tipo de información multimedia imaginable (texto, imágenes, audio, video).

En la Web, como en cualquier biblioteca, tan importante como el almacenamiento de la información es disponer de un sistema que permita localizar, acceder y recopilar en cada momento la que satisface las necesidades particulares de un usuario.

La Web comenzó su andadura en 1989 de la mano de Tim Berners-Lee, y debido a su rápido crecimiento, pronto surgió la necesidad de disponer de algún mecanismo que permitiese a sus usuarios localizar la información que les interesa entre toda la disponible.

La aproximación utilizada más habitualmente para recopilar y localizar información en Internet la constituyen los buscadores basados en técnicas de *crawling*. Los *crawlers* son programas software capaces de recorrer la Web automáticamente, recopilando las páginas accedidas para construir un índice que permita búsquedas por palabra clave sobre su contenido. Los *crawlers* convencionales se inicializan con un conjunto 'semilla' de páginas de entrada y obtienen nuevas páginas mediante la navegación automática a las páginas enlazadas desde el conjunto semilla. Las nuevas páginas son añadidas al conjunto semilla y el proceso se repite hasta que o bien no hay más páginas que examinar o bien se sobrepasa algún límite definido por el administrador (e.g. un límite de profundidad en la navegación o un límite de recursos consumidos).

El enfoque de *crawling* se encuentra en la base de los buscadores más populares y exitosos de la actualidad. Sin embargo, los *crawlers* convencionales sólo pueden acceder a la parte de la Web que se encuentra publicada y enlazada como páginas estáticas. Aunque estas páginas representan una gran cantidad de información en valor absoluto, constituyen sólo una pequeña porción de toda la información web disponible, ya que existe gran cantidad de información en Internet que es generada dinámicamente por un servidor en respuesta a acciones del usuario. El ejemplo más paradigmático lo constituyen las páginas generadas dinámicamente como respuesta a una consulta efectuada por un usuario sobre un formulario web.

Además, un gran número de documentos están accesibles en sitios web que utilizan lenguajes de *script* ejecutados en el cliente web (e.g. *JavaScript* [JAS]) y/o cuyos contenidos son dependientes del estado de la sesión del servidor, lo cuál también impide su acceso por parte de los *crawlers* convencionales.

A esta porción de la Web suele denominársele 'Web Oculta' (*Hidden Web*) o 'Web Profunda' (*Deep Web*) debido a que sus contenidos no pueden accederse desde los buscadores actuales, basados en *crawlers* convencionales. Por oposición, a la porción de la Web compuesta por páginas estáticas suele denominársele 'Web Visible' (*Visible Web*) o 'Web de Superficie' (*Surface Web*).

De esta forma, la Web puede, en cierto modo, verse como un iceberg (ver Figura 1). Existe una gran parte visible pero la parte que no puede verse directamente es de un tamaño mucho mayor y de más valor – constituye la base, los 'cimientos' del iceberg –. La Web de Superficie es sólo la punta del iceberg.

Bergman presenta en [Bergman00] el primer estudio que se ocupó de caracterizar la Web Oculta, llegando a la conclusión de que contiene más del 80% de toda la información disponible en la Web (entre 400 y 550 veces superior al contenido en la 'Web de Superficie'). Además, el contenido de calidad que contiene es entre 1000 y 2000 veces superior (entendiendo por este concepto, información altamente específica y elaborada). A pesar de su alto grado de calidad, el 95% de la información de la Web Oculta está accesible al público en general sin coste alguno.

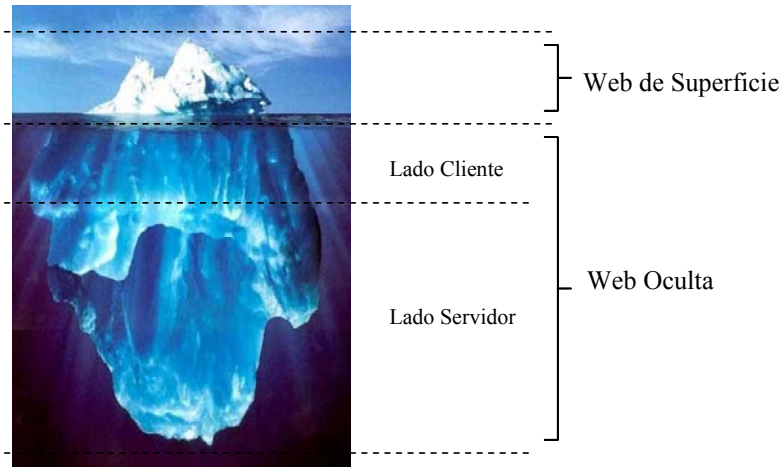


Figura 1 Internet es como un iceberg

Estudios más recientes realizados por Chang et al. en [CHLP+04] corroboran las principales conclusiones del estudio anterior y constatan además el rápido crecimiento del volumen de la Web Oculta: en el lapso que separa ambos estudios (4 años), el tamaño de la Web Oculta ha crecido entre 3 y 7 veces. Los autores de este estudio estiman en cerca de un millón el número de interfaces de consulta en la Web que generan páginas dinámicas como respuesta.

También de acuerdo a Chang et al. en [CHLP+04], en el 77% de los casos estas interfaces de consulta proporcionan acceso a una base de datos subyacente, por lo que las páginas de respuesta contienen un conjunto de resultados estructurados con un esquema definido. Por ejemplo, el formulario de consulta de una tienda electrónica de libros permite al usuario realizar consultas sobre la base de datos de productos de la tienda, devolviendo como respuesta una serie de registros, cada uno de ellos representando los datos de un libro concreto. Su alto nivel de estructuración es una de las causas principales de la percepción de que los datos de la Web Oculta tienen una mayor calidad: los datos estructurados permiten procesamientos más avanzados que los datos no estructurados.

Desde el punto de vista de las técnicas de *crawling*, el problema de tratar la Web Oculta puede dividirse en dos retos principales:

- *Crawling* del 'lado servidor'. Como ya se ha comentado, gran cantidad de sitios web ofrecen formularios de búsqueda para acceder a recursos almacenados en bases de datos subyacentes. Esta información no es accesible desde *crawlers* convencionales, porque estos no tienen la capacidad para ejecutar consultas sobre los formularios.
- *Crawling* del 'lado cliente'. Muchos sitios web usan tecnologías tales como lenguajes de *script* del lado cliente (e.g. *JavaScript*), y/o complejos sistemas de mantenimiento de sesión. Los *crawlers* convencionales no pueden alcanzar este tipo de páginas, puesto que no son capaces de ejecutar los *scripts* y no pueden mantener o recrear las sesiones adecuadas. Además, asociadas al dinamismo del lado cliente, surgen una serie de complejidades adicionales en el proceso de

crawling, como el que ciertos elementos puedan aparecer o desaparecer dinámicamente en función de las acciones del usuario. Por ejemplo, muchas páginas utilizan 'menús emergentes' que se generan dinámicamente cuando el usuario realiza un 'clic' sobre un elemento de la página. Estos menús emergentes contienen enlaces de navegación a nuevas páginas, que serían ignorados por un *crawler* convencional. La Web Oculta del lado cliente ha cobrado mayor importancia en los últimos tiempos debido al uso de las tecnologías de interactividad en el cliente web basadas en *Ajax* [AJAX05].

Debido a la imposibilidad de recolectar los contenidos de la Web Oculta a través de técnicas de *crawling*, los buscadores más populares de la actualidad acceden a parte de esta información a través de acuerdos específicos con los proveedores de contenido. Por ejemplo, algunas tiendas electrónicas proporcionan a los buscadores más populares su catálogo de productos en un formato que puede ser fácilmente indexado. Sin embargo, estas prácticas son relativamente raras, por lo que el grueso de la información disponible en la Web Oculta permanece inaccesible desde los buscadores convencionales.

Como se verá más adelante, entre los objetivos de este trabajo se encuentra proponer nuevas técnicas que ayuden a construir *crawlers* de nueva generación capaces acceder de forma automatizada a la Web Oculta.

I.1.2. INFORMACIÓN ESTRUCTURADA EN LA WEB OCULTA

La información contenida en la Web ha sido habitualmente clasificada dentro de la categoría de información *no estructurada* (junto a, por ejemplo, la información documental), en contraste con la información *estructurada* contenida en las bases de datos.

Los procesamientos que una aplicación software puede realizar con información no estructurada son muy limitados. Por el contrario, la información estructurada contenida en una típica base de datos relacional puede ser manipulada por un programa de ordenador con toda la complejidad que se desee, ya que esta información se ajusta a un esquema perfectamente estructurado y definido.

Una gran parte de la información contenida en la Web Oculta tiene una estructura latente. Tal y como ya se ha comentado, Chang et al. estiman en [CHLP+04] que el 77% de las fuentes de la Web Oculta ofrecen formularios que permiten ejecutar consultas sobre una base de datos subyacente y devuelven los resultados obtenidos, codificándolos en HTML [W3CHTM], para su visualización en un navegador de Internet.

Sin embargo, debido a que HTML no proporciona información que describa la estructura y semántica de sus contenidos, no existe una manera sencilla para que un programa software pueda identificar y extraer los elementos de información deseados.

Por lo tanto, otro de los retos a los que se enfrentan los sistemas modernos para el tratamiento de la información contenida en la Web Oculta es inferir la estructura subyacente presente en este tipo de páginas. Hasta la fecha se han propuesto dos enfoques principales que pueden ser empleados para este propósito: la *Web Semántica* [W3CSW] y el uso de *programas de extracción de datos web*.

La llamada Web Semántica define un conjunto de normas y protocolos para la construcción de fuentes web 'legibles para programas de ordenador' que, entre otras cosas, permiten al creador de un sitio web 'etiquetar' la información disponible con metainformación acerca de su estructura y su semántica. Sin embargo, ni estas iniciativas están teniendo todo el éxito que se esperaba – debido a su importante coste y al desconocimiento por parte de las instituciones y personas físicas que lanzan nuevas aplicaciones y páginas web –, ni solucionan el acceso al conjunto ingente de información que ya existe en la red en aplicaciones que no van a actualizarse a corto/medio plazo.

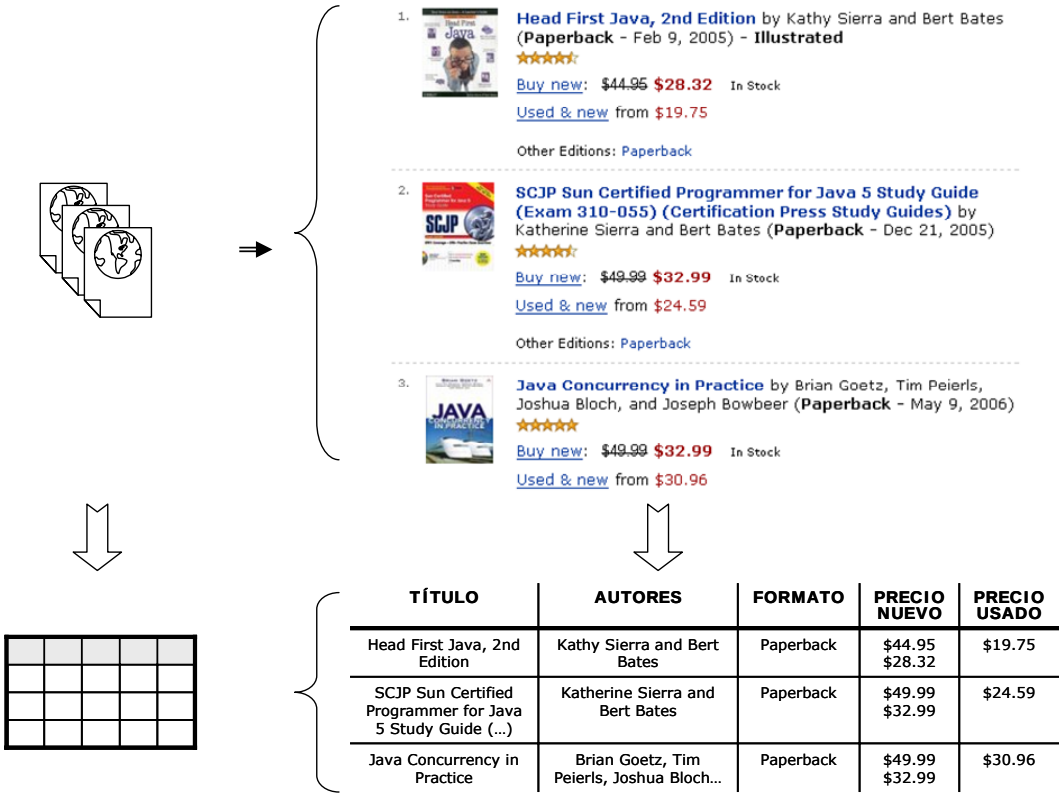


Figura 2 Programa de extracción de datos web

Un *programa de extracción de datos web* para una fuente web recibe como entrada una o varias páginas obtenidas como respuesta a una consulta ejecutada sobre el formulario de consulta de la fuente, y extrae de las mismas el conjunto de datos estructurados que componen la respuesta a la consulta efectuada. Por ejemplo, la Figura 2 esquematiza la actuación de un programa de extracción web para la tienda electrónica de libros Amazon[1]. Los programas de extracción de datos web se aprovechan de que los resultados obtenidos al consultar una fuente web son presentados de forma consistente. Esto es, todos los registros que componen las respuestas a las consultas son formateados de forma similar en la página

[1] http://www.amazon.com

HTML de respuesta, utilizando una *plantilla* subyacente. Por lo tanto, es posible construir programas que aprovechen dichas regularidades de presentación para extraer los datos deseados.

La programación manual de programas extractores web es difícil, propensa a errores y resulta en aplicaciones muy costosas de crear y mantener. Por ello, durante los últimos años se han propuesto numerosas técnicas para la creación semi-automática de programas extractores web, utilizando diferentes aproximaciones como el uso de algoritmos de aprendizaje inductivo [KWD97] [HD98] [MMK01] [MMK03] y/o la creación de herramientas gráficas supervisoras [SA01] [LPH00] [BFG01] [PRAH+02]. También se han propuesto técnicas para el mantenimiento automático de los mismos [RPAH07] [LMK03].

Las técnicas de generación de programas extractores web han permitido la construcción de aplicaciones avanzadas de búsqueda, agregación e integración de información web como comparadores de precios *on-line* [MySimon] o agregadores de contenidos [Yodlee]. Sin embargo, este enfoque requiere que un administrador cree *a priori* un programa extractor para cada fuente de la que se desee extraer datos. Esto presenta dos inconvenientes fundamentales:

- Si el número de fuentes es muy elevado, incluso a pesar de utilizar técnicas avanzadas de generación y mantenimiento, el enfoque puede no escalar adecuadamente.
- Existen aplicaciones en las que las fuentes de información no se conocen previamente, por lo que no es posible desarrollar *a priori* los envoltorios necesarios para tratarlas. Éste es justamente el caso de las aplicaciones de *crawling*, donde las nuevas fuentes de información son descubiertas dinámicamente al explorar la web.

Por lo tanto, ninguno de los dos enfoques (Web Semántica y programas de extracción web) es adecuado para aplicaciones que pretendan recopilar automáticamente la información de la Web Oculta. Otro de los objetivos de este trabajo se centrará en diseñar nuevos métodos capaces de inferir automáticamente la estructura de los datos contenidos en las páginas obtenidas como respuesta a una consulta sobre un formulario web, que sean adecuados para su utilización en este tipo de aplicaciones.

I.1.3. RECOPILACIÓN AUTOMÁTICA DE INFORMACIÓN DE LA WEB OCULTA

Pueden distinguirse dos tipos de estrategias de recopilación de la información de la web: *recopilación de información global* y *recopilación de información dirigida*.

Los buscadores convencionales (e.g. Google[2], Altavista[3], About[4]) utilizan el enfoque *global*, porque pretenden abarcar el contenido completo de la Web. Por lo tanto, en sus procesos de *crawling* los aspectos de escalabilidad son cruciales para construir un sistema exitoso. Para ello es clave que los componentes básicos del proceso de *crawling* realicen operaciones de escasa complejidad. Incluso en ese caso, los recursos computacionales necesarios en términos de ancho de banda, espacio de almacenamiento y capacidad de procesamiento son abrumadores.

Por el contrario, las tareas de recopilación de información *dirigida* están orientadas a un propósito específico. Este tipo de tareas suelen plantearse dentro del ámbito corporativo y presentan requisitos más complejos que los que pueden satisfacerse con los buscadores convencionales. Por ejemplo, considérese el caso de un analista de negocio interesado en construir un archivo de noticias, informes, patentes, productos y cualquier otra información relacionada con la industria del automóvil aparecida en los últimos 6 meses.

En casos como éste, el volumen de información a explorar y recopilar es mucho más restringido por lo que no son necesarios los recursos computacionales precisos para abarcar toda la Web. Sin embargo, el nivel de 'inteligencia' requerido en el proceso de recopilación es considerablemente mayor. Entre las tareas a las que un sistema de este tipo debe enfrentarse se encuentran:

- El sistema debe ser capaz de identificar la información relevante para un determinado propósito de forma automática. Por ejemplo, debe ser capaz de distinguir un documento relevante para el sector del automóvil de otro que no lo es. También debe ser capaz de distinguir formularios de consulta que permiten acceder a recursos sobre el sector del automóvil de aquellos que proporcionan acceso a otro tipo de recursos.
- La información debe ser almacenada conservando su estructura subyacente, de forma que puedan permitirse consultas precisas sobre la información recolectada, del tipo de las que podrían hacerse sobre una base de datos convencional. Por ejemplo, tiene que ser posible permitir a los usuarios realizar consultas como: 'obtener las patentes a nombre de la empresa Acme, cuyo inventor sea John Smith y otorgadas durante el año 2006'.

Las tecnologías disponibles en la actualidad para tareas de recopilación *dirigida* de información son de dos tipos: tecnologías de *crawling dirigido* y tecnologías de *automatización web*.

Las técnicas de *crawling dirigido* utilizan algoritmos de clasificación automática de contenidos para guiar los procesos de *crawling* hacia contenidos relevantes para un dominio de aplicación específico [CBD99] [DP94] [HJMP+98] [CGP98] [RM99] [DCLG+00]. Sin embargo, al igual que los *crawlers* tradicionales, los *crawlers* construidos mediante estas

[2] Google - http://www.google.com

[3] Altavista - http:// www.altavista.com

[4] About - http://www.about.com

técnicas no son capaces de acceder a las páginas de la Web Oculta ni de obtener ni utilizar la estructura subyacente de los datos contenidos en ellas.

Las tecnologías de *automatización web* proporcionan técnicas y herramientas para realizar automáticamente acciones de navegación y extracción de datos sobre la Web. Para realizar las tareas de extracción de datos, suelen apoyarse en las técnicas de generación de extractores web mencionadas en el apartado anterior.

Un ejemplo específico de aplicación de automatización web es la creación de programas envoltorio (*wrappers*). Un programa envoltorio para una determinada fuente web recibe como entrada una consulta emitida por una aplicación software sobre los datos de la fuente, y es capaz de ejecutar automáticamente la consulta recibida y de devolver a la aplicación llamante una lista de resultados estructurados como respuesta. Por lo tanto, las técnicas de automatización web permiten acceder a los contenidos de la Web Oculta y extraer los datos estructurados contenidos en la misma. En la Figura 3 se muestra un programa de automatización web para la tienda electrónica de libros Amazon.

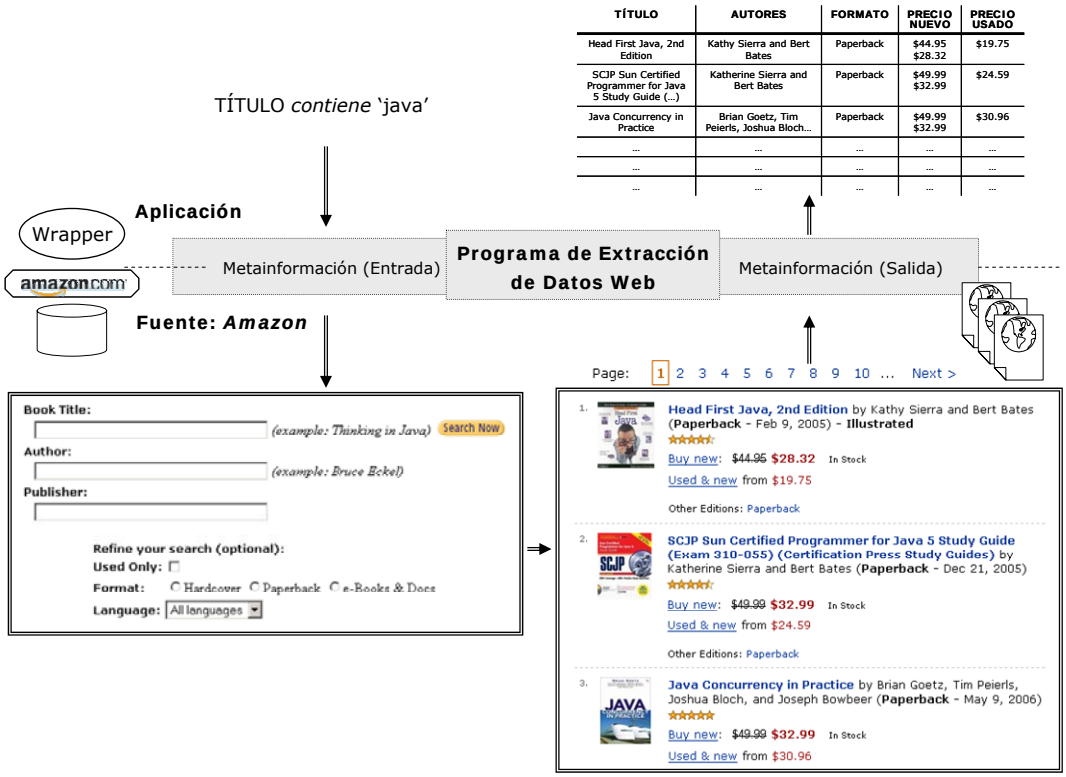


Figura 3 Aplicación de automatización web

Sin embargo, la creación de una tarea de automatización web requiere que el administrador indique específicamente cada paso de navegación a realizar y cómo rellenar cada formulario de consulta. El administrador también debe crear un *programa de extracción de datos web* para cada fuente de la que se deseen extraer datos. Por lo tanto,

este enfoque presenta inconvenientes análogos a los mencionados en la sección anterior para el problema de extracción de datos:

- Si el número de fuentes es muy elevado, el enfoque puede no escalar adecuadamente.
- Es preciso conocer *a priori* cómo navegar a toda la información deseada. También es preciso conocer *a priori* todos los programas de extracción de datos que serán necesarios para tratar la información obtenida. Por lo tanto, no es un enfoque válido en aquellas aplicaciones que descubren dinámicamente nuevas fuentes de información (e.g. aplicaciones de *crawling*).

Este trabajo se centra en la obtención de información de la Web Oculta en el contexto de las aplicaciones de recopilación de información dirigida, solucionando los problemas planteados por las técnicas actuales. El objetivo es dar soporte a las necesidades de información complejas que se plantean en los modernos entornos de negocio. El enfoque propuesto se basará en el de *crawling dirigido*: el crawler explorará periódicamente las fuentes objetivo para localizar información relevante para una tarea específica, y la indexará para su posterior consulta. El *crawler* incluirá las funcionalidades necesarias para acceder a la información contenida en la Web Oculta y para almacenar la información conservando su estructura subyacente.

I.2. OBJETIVOS

En esta tesis doctoral, se estudia el problema de la construcción de aplicaciones de recopilación automática de la información de la Web Oculta utilizando un enfoque dirigido. Se estructura el problema, mostrando los diversos aspectos de los que se compone, se analiza detalladamente el estado del arte, se propone una arquitectura global para construir aplicaciones de este tipo y se proponen un conjunto de técnicas originales para abordar los puntos cruciales del proceso en lo que se refiere al acceso a la Web Oculta. La arquitectura y las técnicas desarrolladas se han validado experimentalmente con tareas y fuentes de información web reales. Además, se presenta una implementación de las técnicas propuestas que contribuye a reforzar la aplicabilidad de las aportaciones de este trabajo.

Los objetivos detallados de este trabajo son:

1) Proponer una arquitectura para aplicaciones de recopilación automática de información de la Web utilizando el enfoque dirigido, que tenga en cuenta el acceso a la Web Oculta. Esta arquitectura debe identificar y tener en cuenta todos los pasos involucrados en la creación de este tipo de aplicaciones. Se pretende minimizar el grado de intervención humana requerida y el nivel de especialización de la misma. Partiendo de una descripción de la tarea específica que se desea abordar, la aplicación utilizará un enfoque de *crawling dirigido* para localizar e identificar información relevante para la tarea y la almacenará para su posterior consulta. La arquitectura incluirá módulos para acceder y procesar adecuadamente la información contenida en la Web Oculta, tanto del 'lado cliente' como del 'lado servidor'. Además, los datos deben ser extraídos y almacenados manteniendo su estructura subyacente (cuando ésta exista). Los datos recolectados deberán ser indexados de forma adecuada para permitir posteriores consultas complejas por parte de los usuarios.

2) Proponer nuevas técnicas y algoritmos para permitir a los sistemas actuales de *crawling dirigido* tratar con fuentes de información que hagan uso de técnicas de 'dinamismo en el lado cliente', tales como el uso de lenguajes de *script* (e.g. *JavaScript*), sistemas complejos de mantenimiento de sesión, etc. También es necesario tratar con otras implicaciones del dinamismo como la aparición o desaparición de elementos de la página en función de las acciones del usuario. Éste es un aspecto de importante interés práctico, sobre todo con la atención cada vez mayor que están recibiendo tecnologías para proporcionar interactividad en el cliente web, como *Ajax*.

3) Proponer nuevas técnicas y algoritmos para identificar y aprender a consultar automáticamente formularios de consulta web relevantes para la tarea especificada. Cuando un *crawler dirigido* capaz de tratar con la Web Oculta encuentra durante su exploración un formulario de consulta web, debe ser capaz de determinar si los contenidos que pueden ser accedidos a través de él son o no relevantes para su tarea. Además, si un formulario es considerado relevante, el sistema debe averiguar la forma correcta de rellenarlo para poder ejecutar consultas sobre él.

4) Proponer nuevas técnicas y algoritmos para extraer automáticamente los datos estructurados contenidos en las respuestas a consultas efectuadas utilizando formularios de consulta sobre bases de datos subyacentes. Tal y como se ha reflejado en estudios como [CHLP+04], la mayor parte de la información contenida en la Web Oculta muestra una estructura subyacente. Inferirla es necesario para permitir consultas complejas sobre los datos obtenidos. Tal y como se ha comentado en secciones previas, el método más habitual para abordar este problema, consistente en la creación de un *programa de extracción de datos web* para cada fuente objetivo, no es viable en este caso ya que las fuentes a tratar no son conocidas *a priori* y, además, su número puede ser muy elevado.

5) Validar la efectividad de las técnicas propuestas con experimentos con tareas y fuentes de información web reales, demostrando que es posible la construcción de sistemas de *crawling* dirigido capaces de acceder y procesar adecuadamente la información de la Web Oculta, y de llevar a cabo tareas reales de recopilación de información con un alto grado de efectividad.

I.3. PRINCIPALES CONTRIBUCIONES ORIGINALES

Este trabajo presenta las siguientes aportaciones principales:

1) Una arquitectura para aplicaciones de recopilación dirigida de información, que contempla el acceso a la Web Oculta. La arquitectura se basa en las arquitecturas existentes de *crawling dirigido* y las complementa y adapta para reflejar los componentes necesarios para el acceso a la Web Oculta. La arquitectura propuesta es descrita en el capítulo III.

2) Un conjunto de técnicas y algoritmos para realizar *crawling* de la 'Web Oculta de lado cliente'. Las técnicas de *crawling* propuestas identifican los recursos descargados mediante el concepto de *ruta*, que es una abstracción que extiende el concepto de URL para soportar mecanismos de mantenimiento de sesión. El proceso de *crawling* se basa en la utilización de componentes denominados '*mini-navegadores*', que son capaces de tratar con lenguajes de *script* (e.g. *JavaScript*). Además, se tienen en cuenta las variaciones en la página que pueden producirse dinámicamente en respuesta a las interacciones del usuario (e.g. nuevos enlaces que aparecen en la página al desplegar un menú emergente). Las técnicas de *crawling* de la Web Oculta 'del lado cliente' son descritas en la sección III.3 y se han presentado previamente en [APRV04] y [APRH06].

3) Un conjunto de técnicas y algoritmos para identificar y aprender a consultar automáticamente formularios de consulta web relevantes para la tarea especificada. El proceso de *crawling* dirigido admitirá como parte de su entrada un conjunto de *especificaciones de dominio de aplicación*. Cada especificación de dominio incluye cierta información que ayuda a identificar formularios de consulta relevantes para la tarea objetivo, así como un conjunto de consultas que se desearía ejecutar sobre tales formularios. Cuando el proceso de *crawling* encuentra durante su exploración un formulario de consulta web, utiliza diversas heurísticas basadas en técnicas de distancia visual y similitud textual para determinar si el formulario es relevante para alguno de los *dominios de aplicación* y, en ese caso, para aprender automáticamente a ejecutar consultas sobre el mismo, y obtener las respuestas a las consultas especificadas. Las técnicas de tratamiento de formularios (*crawling* de la Web Oculta 'del lado servidor') son descritas en la sección III.4 y se han presentado previamente en [ARCP06] y [APRC+07]. [RPB07] y [PRAC+07] aplican las técnicas de tratamiento de formularios a un sistema de mantenimiento automático para automatización web, para realizar el *crawling* requerido para la regeneración automática de secuencias de navegación web.

4) Nuevas técnicas y algoritmos para extraer automáticamente los datos estructurados contenidos en las respuestas obtenidas a consultas efectuadas utilizando formularios web de consulta sobre bases de datos subyacentes. Cuando el sistema de *crawling* descubre un nuevo formulario relevante y ejecuta un conjunto de consultas sobre el mismo, las páginas de respuesta son proporcionadas como entrada a un módulo de *estructuración automática*. Este módulo utiliza técnicas originales para obtener de

cada página HTML de respuesta los registros estructurados contenidos en ella. Las técnicas de estructuración automática de contenidos de la Web Oculta son descritas en la sección III.5. En el momento de escribir estas líneas, están aceptadas para su publicación en [APRB+07], [APRB+07b] y [APRB+08].

5) Un conjunto de herramientas software que permiten la creación sencilla de aplicaciones de *crawling* dirigido. Estas herramientas han sido utilizadas para la validación experimental de las técnicas propuestas, así como en diversas tareas de obtención de datos reales. Los detalles sobre estas herramientas se proporcionan en el anexo VIII.

El enfoque de este trabajo centra la descripción y discusión técnica de la arquitectura para el *crawling* dirigido de la Web Oculta en estas aportaciones. Otras partes de la misma, tales como los mecanismos de *crawling* dirigido 'tradicional' (sección II.2) son descritos a nivel general para proporcionar el adecuado contexto para los objetivos de este trabajo.

Los principales resultados son presentados en este documento y permiten, además de contribuir a validar el sistema aquí propuesto, obtener conclusiones relativas a los sistemas de *crawling* de la Web Oculta y su adecuación a las necesidades reales. Las conclusiones se describen en la sección V.2.2.

I.4. ESTRUCTURA DE LA TESIS

El Capítulo II, '*Estado del Arte*', comienza con una introducción en la que se identifican y describen los principales problemas que debe abordar un sistema capaz de realizar tareas de recopilación dirigida de información, incluyendo el acceso y procesamiento de la información contenida en la Web Oculta. La parte central del capítulo la constituye la exposición y discusión detallada de las principales técnicas propuestas hasta el momento para abordar cada una de las tareas identificadas. Si bien el capítulo hace especial énfasis en las técnicas más relacionadas con las principales aportaciones de este trabajo, también se describen algunas de las técnicas y sistemas más relevantes que, aún no abordando específicamente los problemas derivados del *crawling* dirigido de la Web Oculta, tratan problemas relacionados o complementarios.

El Capítulo III, '*Arquitectura de Crawling Dirigido de la Web Oculta*', describe en detalle la arquitectura y las nuevas técnicas presentadas en este trabajo. Se comienza con una descripción global de la arquitectura propuesta, detallando posteriormente las funciones y la estructura de cada uno de los módulos que la componen. Posteriormente se describen en detalle las técnicas y algoritmos originales propuestos para tres aspectos clave de la arquitectura: tratamiento de la 'Web Oculta de lado cliente', identificación e interpretación de formularios de consulta relevantes y extracción totalmente automática de datos estructurados de las páginas web conteniendo la respuesta a las consultas efectuadas.

El Capítulo IV, '*Experiencia Obtenida del Uso del Sistema*', valida la efectividad de la arquitectura y las nuevas técnicas para *crawling* dirigido de la Web Oculta propuestas en este trabajo, aplicadas en la práctica con tareas y fuentes web reales. Se comienza con la descripción de las distintas clases de experimentos realizados para la validación del prototipo desarrollado conforme a las técnicas diseñadas en esta tesis. A continuación se describen algunas aplicaciones significativas en las que se ha utilizado el sistema, con el objetivo de proporcionar una visión más clara de cómo el sistema puede utilizarse en aplicaciones reales. Posteriormente, se realiza una evaluación del cumplimiento de los objetivos planteados en función de los datos recogidos de los experimentos.

Finalmente, el Capítulo V, '*Discusión, Conclusiones y Trabajo Futuro*', comienza discutiendo los resultados y aportaciones de este trabajo con respecto al trabajo relacionado, que fue introducido en el Capítulo II, continúa exponiendo las conclusiones de la tesis doctoral, y finaliza esbozando las líneas básicas de trabajo futuro del autor.

II. ESTADO DEL ARTE

Este capítulo proporciona una introducción a las técnicas de *crawling* dirigido y tratamiento de la Web Oculta que se han propuesto hasta el momento en la literatura. Aunque hace especial énfasis en las técnicas más relacionadas con las principales aportaciones de este trabajo, también describe otras técnicas y sistemas que, aún no abordando específicamente los problemas derivados del *crawling* dirigido de la Web Oculta, se ocupan de problemas relacionados o complementarios.

La sección II.1 es una introducción general a las problemáticas que plantea la recuperación de información en la Web, y presenta la arquitectura general de un *crawler* web global.

La sección II.2 describe las principales aproximaciones a la construcción de *crawlers* dirigidos. Presenta técnicas que intentan que el proceso de *crawling* no diverja a sitios de temáticas diferentes a los de partida, para solucionar los problemas de 'falta de dirección' presentes en los sistemas convencionales.

Un problema común a estas dos arquitecturas es que no proporcionan ningún mecanismo para abordar la recopilación de información de la Web Oculta, ni en su vertiente cliente ni servidora. La sección II.3 es la parte central del capítulo y presenta la exposición y discusión detallada de las principales técnicas propuestas hasta el momento para abordar cada una de las tareas identificadas en el acceso y estructuración de la información contenida en la Web Oculta.

La sección II.4 presenta las conclusiones obtenidas del estudio del estado del arte, resumiendo las fortalezas y debilidades de los sistemas existentes para abordar los desafíos que presenta el acceso al contenido en la Web Oculta.

II.1. *CRAWLING* DE LA WEB

Diversos estudios se han ocupado de caracterizar la Web, tanto desde el punto de vista del número de documentos que contiene como por la forma en la que están interrelacionados. La estructura de la Web es compleja y evoluciona en el tiempo. Además del tamaño y de la rapidez en el cambio de los diferentes recursos web, su naturaleza basada en hipervínculos la hacen diferente de muchas otras colecciones de datos.

Un estudio reciente de Broder et al. [BKMR+00] sugiere que la estructura de hipervínculos de la Web puede verse como una pajarita. Como se muestra en la Figura 4, aproximadamente el 28% de las páginas constituyen un núcleo fuertemente conectado (el centro de la pajarita), un 21% está formado por páginas que pueden alcanzar el núcleo pero no pueden ser accedidas desde éste y otro 21% son páginas a las que se puede llegar desde el núcleo pero no a la inversa. Estos dos grupos constituyen los lazos de la pajarita (la entrada y la salida). Alrededor de estos dos grupos se encuentran los tentáculos, que contienen aproximadamente el 22% de las páginas y que son caminos sin salida, con la excepción de unos pocos que conectan el grupo de entrada con el de salida. Finalmente, el 8% restante está formado por islas, que son páginas que no están conectadas al resto de la Web.

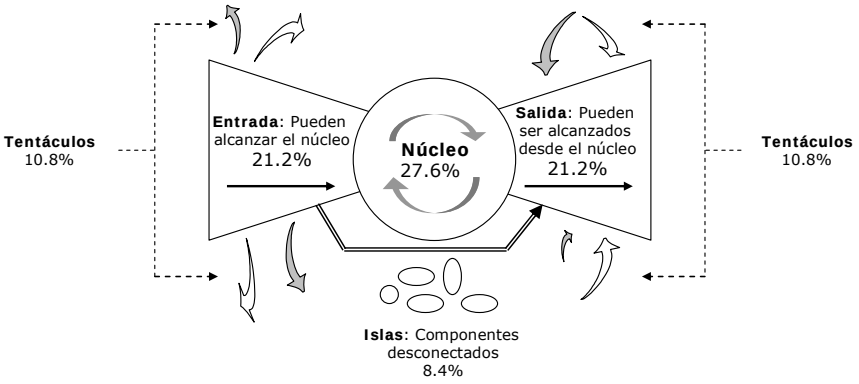


Figura 4 La Web es como una pajarita

La estructura en forma de pajarita de los hipervínculos entre páginas hace posible que partiendo de un sitio web determinado, sea posible alcanzar una gran parte de la Web, simplemente siguiendo enlaces para obtener nuevos documentos. Esta idea es utilizada por los sistemas de *crawling* para obtener una fracción de la información contenida en la Web, sobre la que permitir realizar búsquedas que faciliten a los usuarios la localización de información.

A continuación se presenta la arquitectura básica de un buscador web basado en *crawlers*, con sus diferentes componentes, como se describe en [ACGP+01]. La Figura 5 muestra el esquema básico. Se distinguen principalmente tres componentes: *crawling*, indexación y consulta.

El módulo de *crawlers* recibe como entrada un conjunto de URLs iniciales que apuntan a documentos que hay que descargar de la Web. Utilizando como base para su funcionamiento la estructura de hiperenlaces existente entre los diferentes documentos, extrae los URLs que aparecen en las páginas recuperadas y envía esa información al módulo de control del *crawler*. De forma adicional, también almacena las páginas recuperadas en un repositorio.

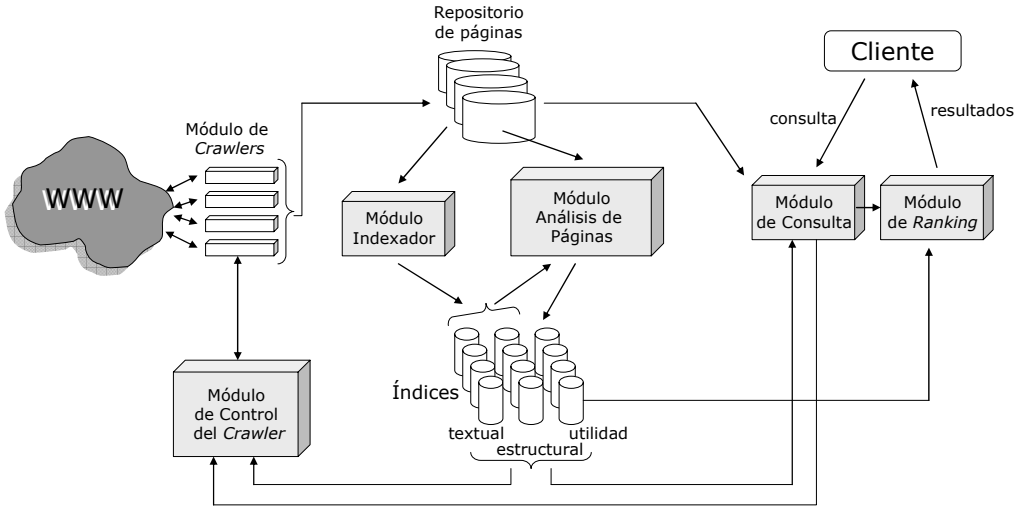


Figura 5 Arquitectura general de un buscador web basado en *crawlers*

El módulo de control del *crawler* determina qué enlaces visitar y el orden de los mismos, y alimenta con ellos al módulo de *crawlers*. Es el responsable de gestionar la dirección de exploración del *crawler* y establecer su criterio de parada. El algoritmo básico de *crawling* puede admitir diversas configuraciones. Por ejemplo, en función de la amplitud que se le quiera dar al *crawling* (es decir, el porcentaje de la Web a abordar) o de la temática objetivo para las páginas recolectadas, puede configurarse para visitar tantos sitios como sea posible, prescindiendo de las páginas que se encuentran a niveles más profundos en los sitios analizados; o puede configurarse para analizar sólo sitios de un dominio. Adicionalmente, el módulo de control de *crawling* puede utilizar el resultado de iteraciones anteriores, para decidir a qué URLs acceder y cuáles descartar. El módulo de control del *crawler* también puede utilizar consultas previas del usuario para guiar el proceso de *crawling*.

El módulo de indexación extrae todas las palabras de cada página y le asocia el URL en el que cada palabra ha aparecido. El resultado es una tabla de búsqueda generalmente de gran tamaño, que proporciona todos los URLs que apuntan a páginas donde una palabra dada aparece (representada por el índice 'textual' en la Figura 5). Lógicamente, el tamaño de la tabla estará limitado a las páginas obtenidas por el proceso de *crawling*. Debido a las dificultades inherentes en el tratamiento de volúmenes de información tan grandes y con frecuencias de cambio tan altas, normalmente se suele hacer uso de varios índices. Por ejemplo, suele utilizarse un índice separado para almacenar la estructura de enlaces entre los diferentes documentos. El módulo de análisis de páginas es el responsable de la creación de estos índices.

El repositorio de páginas representa la colección (posiblemente temporal) de páginas que son descargadas por el *crawler*. Esta colección de páginas suele mantenerse mientras no se finaliza el proceso de *crawling*/indexación. En algunos casos se mantienen más tiempo, para utilizarlas a modo de caché de documentos y acelerar el acceso a los documentos contenidos en las páginas de resultados de búsqueda sobre los contenidos recolectados por el *crawler*.

El módulo de consulta es el responsable de recibir y procesar las peticiones de los usuarios. Para resolver una petición utilizan la información almacenada en los índices y, en algunas ocasiones, también en el repositorio de páginas.

El módulo de consulta tiene especial interés debido a que la aplicación de técnicas de recuperación de información de forma directa sobre colecciones de documentos web plantea problemas de selectividad. Las técnicas tradicionales dependen de la similitud de los textos de la consulta con los textos en los documentos de la colección. Pero cuando las consultas incluyen pocas palabras, como suele ser el caso de consultas en la Web, el conjunto de documentos resultado es muy grande y es necesario utilizar algún mecanismo de filtrado para eliminar páginas irrelevantes. El módulo de ranking tiene como misión ordenar los resultados de tal forma que los que más interesen al usuario aparezcan al principio. Existen técnicas como *PageRank* [BP98] usado en Google o HITS [Kleinberg99] que complementan las técnicas de Recuperación de Información Tradicionales con el análisis de la estructura de hiperenlaces entre documentos para tratar de resolver estos problemas.

En el anexo VI se comenta en detalle la arquitectura y desafíos que presenta la construcción de un sistema de *crawling* que pretenda recorrer la Web de forma efectiva y eficiente.

La arquitectura presentada para un sistema de *crawling* global de la Web tiene como característica que partiendo de un conjunto de URLs iniciales (que pueden pertenecer todos a la misma temática), no garantiza que todos los documentos que se vayan a procesar pertenezcan también a ese mismo dominio de información. Además, sólo permite acceder a documentos que se encuentren enlazados, en uno o varios niveles, desde los documentos que constituyen el conjunto de inicio del sistema de *crawling*, imposibilitando por lo tanto la inclusión de documentos contenidos en la Web Oculta.

Por estos motivos, la aproximación de *crawling* general, por una parte no es válida porque no permite obtener información de la Web Oculta, y por otra parte no sería efectiva debido a que recuperaría mucha información irrelevante para las tareas de extracción consideradas.

II.2. *CRAWLING* DIRIGIDO

En sus inicios, la mayor parte de la Web de Superficie podía ser recorrida por un sistema de *crawling* de tamaño pequeño o medio. Entre 1996 y 1999 el nivel de exploración de la Web ha constituido un desafío muy importante: de un grado de exploración estimado del 35% en 1997 [BB98] se pasó a un 18% en 1999 [KEW01]. Después del año 1999, el crecimiento de la Web se detuvo un poco y se mejoró el nivel de conectividad de red, resultando en un nivel de exploración del 45%-55% con Google en el año 2000.

Debido al enorme tamaño de la Web, los sistemas de *crawling* de propósito general no son capaces de satisfacer las necesidades de acceso a información de todos los usuarios. Pero no es necesario que un *crawler* recorra toda la Web para ser efectivo. Cuando se trata con tales volúmenes de información, más importante que el número de documentos obtenidos es la selección de aquellos relevantes y su ordenación según los criterios de un usuario. En realidad, el conjunto de páginas indexadas por los sistemas de *crawling* más conocidos que son vistas por los usuarios, son una pequeña fracción del total de documentos que procesan.

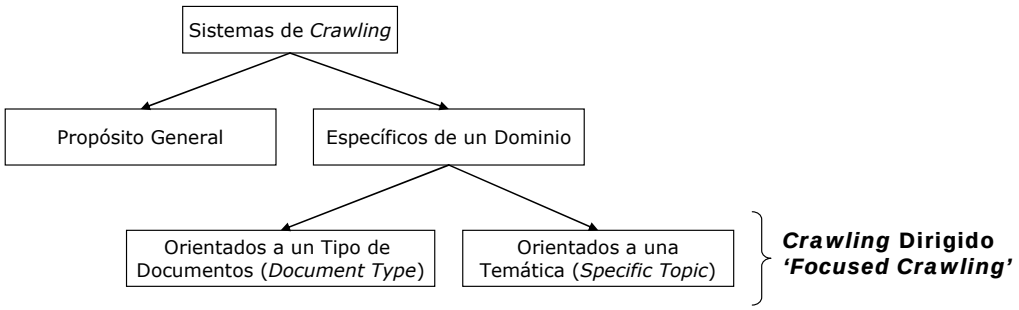


Figura 6 Clasificación de los sistemas de *crawling* web

Como se muestra en la Figura 6, los sistemas de *crawling* web pueden ser de propósito general u orientados a un dominio específico. Dentro de los sistemas orientados a un dominio específico se pueden distinguir entre aquellos que se centran en determinados tipos de documentos y aquellos otros orientados a una temática específica. Por ejemplo, *BuildingOnline*[5] está especializado en el dominio de la industria, *CollegeBot*[6] busca recursos relacionados con la educación, *LawCrawler*[7] está especializado en búsquedas de información legal en la Web, *Google Groups*[8] (anteriormente conocido como *DejaNews*) recopila información de artículos de noticias y *WebSeek*[9] imágenes. Como su tamaño es

[5] http://www.buildingonline.com/
[6] http://www.collegebot.com
[7] http://lawcrawler.findlaw.com/
[8] http://groups.google.com/
[9] http://persia.ee.columbia.edu:8008/

mucho más reducido, normalmente proporcionan resultados más precisos y es más sencillo mantenerlos actualizados.

El recorrido de la Web para *crawlers* especializados en un sitio determinado es relativamente sencillo. Simplemente tienen que definir unos filtros de URLs que restrinjan la descarga de documentos que pertenezcan a un nombre de dominio concreto. Sin embargo la tarea de seleccionar los recursos a indexar es más compleja para aquellos orientados a una temática específica, que tienen que identificar, a partir de una lista de URLs no visitados, los que enlazan la información más relevante y posteriormente determinan la relevancia de cada documento descargado, según el propósito específico del *crawler*, para descartar los documentos irrelevantes o de poca calidad.

El recorrido en anchura permite al *crawler* obtener, como mínimo, las páginas de inicio de los sitios web enlazados por las páginas exploradas. A pesar de su simplicidad, se utiliza de forma extensiva en sistemas de *crawling* especializado, debido a que es fácil de implementar y rápido en ejecución. Intuitivamente si un URL es relevante para un dominio destino, es probable que las páginas en su vecindad también lo sean. Esta idea está basada en los resultados obtenidos por Kumar et al. en [KRRT02], en el que se compara la Web con una sociedad, modelada como un grafo en el que los enlaces representan los arcos y las páginas los nodos. En dicho modelo, las páginas no apuntan a otras al azar, sino que reflejan páginas interesantes o relevantes para ésta, según el autor de la misma.

El recorrido en anchura funciona bien en sistemas orientados a un sitio específico, pero es más difícil utilizarlos para sistemas especializados en un tema porque no utilizan ninguna heurística para localizar e identificar páginas web relevantes, por lo que divergen fácilmente. En el apartado II.2.1 se describen técnicas que permiten restringir un proceso de *crawling* a un conjunto de páginas relevantes a una temática determinada, utilizando diferentes técnicas de clasificación de documentos.

II.2.1. APROXIMACIONES AL *CRAWLING* DIRIGIDO

Chakrabarti et al. [CBD99], acuñaron el nombre de *crawler* dirigido (*focused crawler*). Un *crawler* dirigido localiza, obtiene, indexa y mantiene páginas relacionadas con un conjunto de temáticas determinadas, que representan segmentos relativamente limitados de la Web.

El *crawler* dirigido ideal recupera el conjunto de páginas relevantes de un dominio, atravesando el menor número de documentos irrelevantes. Comienza con un pequeño conjunto de páginas relacionadas con una temática determinada y durante el proceso, un 'clasificador' determina qué enlaces seguir para cada página obtenida, en función de su relevancia potencial para la temática objetivo. En última instancia, el clasificador es el encargado de decidir la dirección de exploración del sistema de *crawling*.

Se pueden considerar tres aproximaciones para la construcción de un sistema de este tipo, en función de la información utilizada por el 'clasificador' para determinar la adecuación de cada una de las páginas a la temática general de una tarea de *crawling* concreta. Algunos hacen sólo uso del contenido de las páginas obtenidas, pero otros utilizan

la estructura de enlaces existentes entre los documentos web para obtener la relevancia de las páginas o el nivel de confianza en una ruta de exploración determinada. En realidad las diferentes aproximaciones no son exclusivas, sino que pueden complementarse para mejorar la efectividad del sistema final. Las siguientes subsecciones describen cada una de ellas.

II.2.1.1. SISTEMAS BASADOS EXCLUSIVAMENTE EN TÉCNICAS DE CLASIFICACIÓN

La primera aproximación se basa en el resultado de varios estudios que afirman que si p_1 y p_2 son páginas web directamente conectadas por un enlace desde p_1 a p_2, entonces la probabilidad de que p_2 sea relevante para una cierta temática T es mucho mayor si p_1 es relevante para T (cabe resaltar que esto no contradice la afirmación inicial de que un *crawler* no dirigido diverge rápidamente). En base a esta observación, estas técnicas ordenan los nuevos URLs descubiertos por el *crawler*, dando mayor prioridad a los obtenidos de enlaces contenidos en páginas relevantes a la temática *T*. Para determinar si una página dada es relevante para *T*, utilizan técnicas de clasificación automática (típicamente basados en expresiones regulares o en clasificadores Bayesianos entrenados previamente al proceso de *crawling*). Los sistemas *Fish* [DP94] y *Shark* [HJMP+98] utilizan esta aproximación.

Fish fue uno de los primeros sistemas de *crawling* dirigido. Fue concebido como una aplicación final para recuperación de información en tiempo real, que realiza un recorrido de la Web en profundidad, simulando la migración de un 'banco de peces'. Cada página se corresponde con un pez cuya supervivencia depende de la relevancia de la página visitada y de la velocidad del servidor remoto. Calcula la importancia de una página utilizando un clasificador binario (la página sólo puede ser relevante o irrelevante), en función de unas palabras clave o una expresión regular. Finaliza cuando ha atravesado un número determinado de páginas irrelevantes. De esta forma es capaz de acceder a páginas relevantes que están a más de un salto de otras relevantes. En cada documento atravesado, el pez se reproduce y la cantidad de descendientes depende de la relevancia de la página y del número de enlaces extraídos. Como consecuencia, el banco de peces migra en la dirección general de las páginas relevantes, que serán mostradas como resultado. El usuario especifica el punto de inicio como un conjunto de páginas semilla (o también puede utilizarse otro buscador para obtener los URLs de inicio). Esta técnica asume que los documentos relevantes para una temática deberían de encontrarse próximos en la estructura de enlaces. El sistema *Shark* extiende el algoritmo definido por *Fish*, priorizando los URLs de las páginas a descargar teniendo en cuenta una combinación lineal de la relevancia de la página origen, el texto del enlace y sus vecinos en la página origen, y la puntuación de relevancia heredada. La puntuación de relevancia heredada es la relevancia de la página padre, multiplicada por un factor de reducción. Además, la relevancia de la página se calcula utilizando medidas estándar de similitud entre el documento y la consulta, pudiendo ser cualquier número real entre `0` y `1`. Las puntuaciones del texto del enlace y de su contexto se calculan también como la similitud con la consulta.

Los sistemas de crawling dirigido que se basan únicamente en técnicas de clasificación automática presentan dos problemas:

- No se comportan bien con páginas formadas por un gran conjunto de enlaces relacionados con una temática determinada, denominadas páginas *hub*, por presentar pocos textos. Éste es obviamente un gran inconveniente debido a que se trata de páginas que contienen enlaces a los que el sistema debería asignar las prioridades más altas.
- No funcionan bien con páginas que, sin ser realmente relevantes por sí mismas para un tema objetivo *T*, sí son páginas que tienen una alta probabilidad de acabar llevando a páginas relevantes.

II.2.1.2. SISTEMAS QUE UTILIZAN TÉCNICAS DE ANÁLISIS DE HIPERVÍNCULOS

Los problemas de las técnicas del apartado anterior pueden ser abordados utilizando la información proporcionada por la estructura de hipervínculos existente entre las páginas para seleccionar de forma 'inteligente' los enlaces que deben ser considerados en el proceso de *crawling*, descartando aquellos menos relevantes. Los algoritmos PageRank [BP98] usado en Google o HITS [Kleinberg99], utilizan este concepto para ordenar los documentos web de acuerdo a su relevancia respecto a algún tema o consulta de un usuario. El más interesante para nuestros propósitos es HITS (*Hypertext Induced Topic Search*), que utiliza la estructura de enlaces entre documentos para asignar a cada página dos puntuaciones: una como autoridad y otra como *hub*. Las páginas autoridad son aquellas que son más relevantes para una consulta determinada. En cambio las páginas *hub* son aquellas que no son necesariamente autoridades por ellas mismas, pero tienen enlaces a muchas páginas relevantes (autoridades). El algoritmo se basa en una relación de refuerzo mutuo entre estos dos tipos de páginas: una página autoridad es una página que es enlazada por muchas páginas *hub* y los *hubs* son páginas que apuntan a muchas páginas autoridades.

Los sistemas basados en esta aproximación utilizan técnicas similares a HITS para tratar adecuadamente páginas que contienen muchos enlaces a páginas que son relevantes (*hubs*). Estas técnicas también son útiles para localizar de forma automática buenos sitios de inicio de *crawling* y para conseguir que el *crawling* 'salte' a otras zonas de la Web cuando el *crawler* no es capaz de obtener páginas relevantes en la zona actual (asignando prioridades más altas a los URLs de las páginas *hub*). Entre los sistemas basados en esta aproximación se encuentran los descritos por Cho et al. [CGP98] y Chakrabarti et al. [CBD99].

En [CGP98], Cho et al. utilizan el parámetro 'importancia de las páginas web' para reordenar la cola de URLs a ser accedidos. Los autores experimentaron con diferentes heurísticas de relevancia de páginas, como por ejemplo frecuencias de palabras clave en los documentos, similitud respecto a ejemplos iniciales, número de enlaces que apuntan a una página o puntuación PageRank. Demostraron experimentalmente sobre la web de la Universidad de Stanford que un *crawler* de este tipo, utilizando PageRank sobre el grafo inducido por las páginas descargadas hasta un momento determinado, se comportaba mejor

frente a otro que utilizase el número de enlaces que apuntan a una página o un recorrido en anchura.

Chakrabarti et al. [CBD99] utilizan una taxonomía de documentos existente (por ejemplo, las páginas del directorio de Yahoo![10] o Dmoz[11]) y un conjunto de documentos iniciales para crear un modelo para clasificar las páginas recuperadas en categorías. A diferencia de un *crawler* exhaustivo que sigue cada enlace de una página en anchura, este *crawler* dirigido da prioridad a los enlaces que pertenecen a páginas clasificadas como relevantes. En la Figura 7 se muestra la arquitectura de este sistema. Está formado por tres componentes separados: un *crawler*, un clasificador y un filtro (*distiller*). El clasificador – un clasificador Naïve Bayes – se utiliza para determinar la relevancia de una página (de acuerdo a la taxonomía), e influirá en la futura expansión de enlaces. Presenta dos reglas diferentes para expansión de enlaces: la regla estricta (*hard*) permite expansión de enlaces sólo si la clase a la que pertenece la página con mayor probabilidad forma parte de uno de los subconjuntos que interesan. La regla débil (*soft*) utiliza la suma de las probabilidades de que una página pertenezca a una de las clases relevantes para decidir visitar sus hijos (no se elimina ninguna página *a priori*). De forma periódica, el componente filtro identifica páginas *hub*. Para cada página calcula los valores de *hub* y autoridad con una versión extendida del algoritmo HITS. Intenta acceder a los enlaces dentro de las páginas con mayor puntuación *hub* primero, con la esperanza de localizar lo antes posible autoridades (páginas relevantes). El sistema funciona en dos fases: una de entrenamiento y otra de pruebas. En la fase de entrenamiento, el clasificador se entrena con datos relevantes para el tema objetivo.

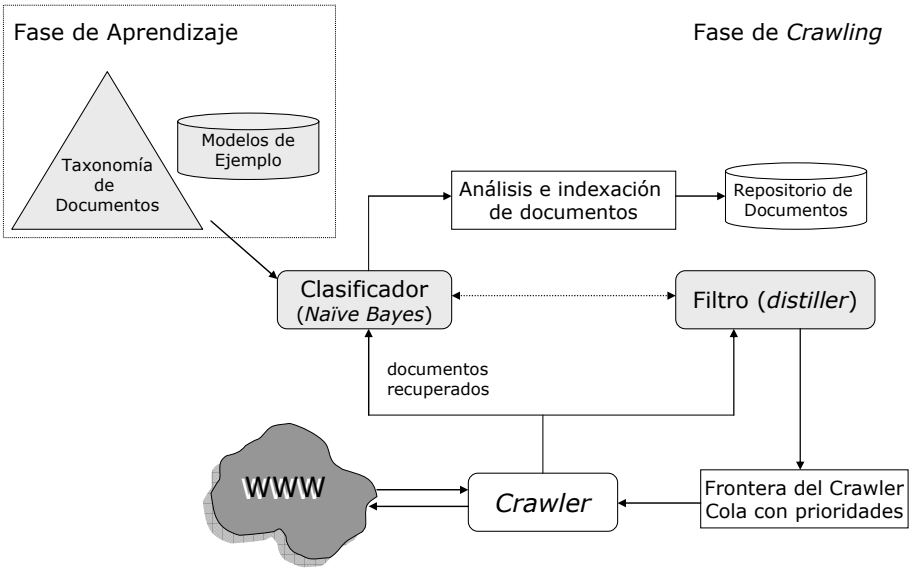


Figura 7 Arquitectura de un *crawler* dirigido

[10] http://dir.yahoo.com

[11] http://www.dmoz.org

Aunque esta estrategia es efectiva, no es óptima debido a que en dominios grandes, el número de enlaces que son irrelevantes puede ser muy alto. Para mitigar este problema, los autores mejoran el sistema añadiendo al *crawler* un clasificador adicional, el 'aprendiz' (*apprentice*), para seleccionar los enlaces más prometedores en una página relevante. El clasificador base calcula la relevancia de las páginas. El aprendiz utiliza esa información para aprender las características de buenos enlaces y así priorizarlos para alimentar el *crawler*.

II.2.1.3. SISTEMAS QUE UTILIZAN INFORMACIÓN DE RUTAS A DOCUMENTOS RELEVANTES

Los *crawlers* que utilizan sólo el contenido de las páginas recuperadas para determinar el recorrido a realizar sobre la Web tienen el inconveniente de que pueden perder páginas relevantes debido a que sólo consideran páginas que se espera que proporcionen un beneficio inmediato.

Los sistemas basados en esta aproximación (como los descritos por Rennie y McCallum [RM99] o Diligenti et al. [DCLG+00]) intentan solucionar esta limitación proponiendo algunas estrategias que entrenan a los clasificadores con características obtenidas de rutas que llevan a una página relevante, en lugar de considerar sólo el contenido de la página, para ser capaces de alcanzar la mayor parte de contenido relevante.

Rennie y McCallum [RM99] utilizaron aprendizaje por refuerzo para construir un *crawler* dirigido (*Cora*) efectivo para dominios en los que la información relevante se encuentra dispersa. En lugar de considerar sólo el contenido de una página individual y recorrer la Web siguiendo páginas que dan beneficio inmediato, entrenan un clasificador con características obtenidas de rutas que llevan a una página relevante: título y contenido del documento en el que se encuentra el enlace, palabras en el URL, texto del enlace y textos en las proximidades del enlace. Dado un enlace (p_1, p_2), el clasificador devuelve una estimación del número de páginas relevantes que pueden ser alcanzadas siguiendo (p_1, p_2). De forma repetida, recorren sitios de ejemplo para construir los grafos de conectividad con las rutas óptimas a los documentos relevantes. Esta aproximación fue diseñada para solucionar problemas de búsqueda sobre sitios web bien definidos. Por ejemplo, localizar artículos de investigación en sitios web de departamentos de ciencias de la computación: aunque la página 'home' de la web del departamento no sea considerada relevante porque no contiene artículos de investigación, hay una alta probabilidad de que acabe conduciendo a los mismos.

El *crawler* que proponen tiene dos fases: entrenamiento y pruebas. En la fase de entrenamiento aprende a asociar textos próximos a los URLs con un valor escalar, en base al recorrido de sitios de ejemplo. Este valor será la penalización por acceder a ese URL. Los premios se obtienen al visitar documentos de la temática de interés que son accesibles siguiendo ese URL, directa o indirectamente (puede estar a varios saltos desde una página dada). En la fase de pruebas, o fase de funcionamiento, el *crawler* asigna un valor escalar a cada URL no visitado en base a sus textos próximos. Este valor es la suma estimada de premios que se pueden obtener desde ese URL. El *crawler* obtiene los URLs desde una cola

de prioridad en la que los URLs se ordenan de acuerdo a la suma estimada de futuras penalizaciones. Una ventaja de esta aproximación es que considera premios futuros de enlaces en las prioridades de *crawling*, de forma que la probabilidad de acceder a un enlace dentro de un documento fuera de la temática que pueda llevar a una cantidad razonablemente alta de documentos del tema buscado, es alta.

Diligenti et al. [DCLG+00] también tienen en cuenta las rutas a las páginas relevantes. Sin embargo, su clasificador estima la distancia desde una página p_1 a alguna página relevante p_2, sin distinguir entre los diferentes enlaces de p_1 (si el clasificador determina que una página p_1 es buena, se recuperan todas las páginas directamente alcanzables desde p_1). Basados en la idea de que algunas páginas fuera de la temática pueden conducir a páginas del tema de interés, en algún nivel de profundidad desde la página de inicio, consideran que las páginas de la temática se pueden encontrar conociendo las temáticas de las páginas que las enlazan. Para resolver este problema proponen un *crawler* dirigido por el contexto, cuya arquitectura se muestra en la Figura 8, que usa motores de búsqueda de propósito general para localizar las páginas que apuntan a una dada (*back-links*), y las utiliza para construir un grafo con el contexto de cada página, en varios niveles. Esas páginas se utilizan para entrenar un conjunto de clasificadores para asignar documentos a varias clases, de acuerdo a su distancia esperada desde documentos fuera de la temática. Se construyen grafos y clasificadores para cada documento de inicio en cada capa particular, mostrando la distancia esperada a páginas objetivo. Se utiliza un clasificador Naïve-Bayes para cada capa. Los enlaces con poca distancia a los documentos objetivo serán analizados antes, en base a la división en varias colas de los URLs a ser accedidos por el *crawler* (una cola para los URLs clasificados en las diferentes distancias consideradas).

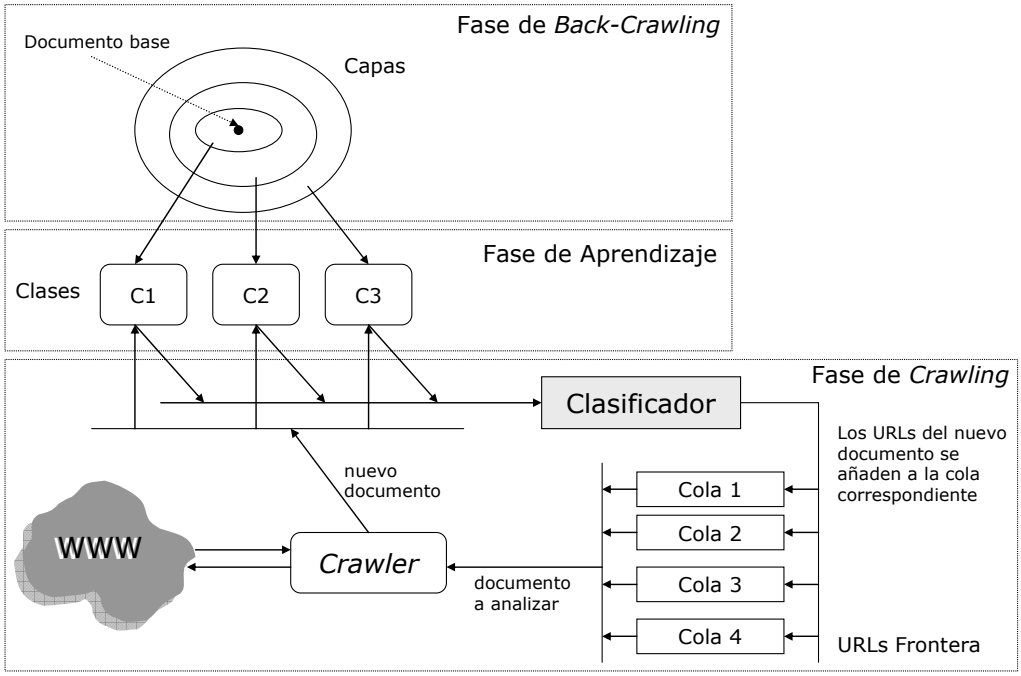


Figura 8 Arquitectura de un c*rawler* dirigido por el contexto

II.3. SISTEMAS DE TRATAMIENTO DE LA WEB OCULTA

II.3.1. INTRODUCCIÓN

En las secciones anteriores se ha descrito el funcionamiento de los sistemas de *crawling* convencionales. Tanto los sistemas de *crawling* global (sección II.1) como los sistemas de *crawling* dirigido a una temática determinada (sección II.2) se caracterizan por basar su modo de operación en el recorrido de la Web siguiendo los enlaces existentes entre las páginas. Es decir, en base a la división de la Web introducida en el apartado I.1.1, recorren la denominada Web de Superficie, sin proporcionar ningún mecanismo para acceder a la información contenida en la Web Oculta.

La Web Oculta es la parte de la Web a la que un usuario puede acceder navegando, pero los *crawlers* no. En general suele denominarse como Web Oculta a la porción de páginas web que se genera de forma dinámica. Se pueden distinguir dos tipos de dinamismo: el de contenido y el de navegación.

Raghavan y García-Molina definen el *dinamismo de contenido* en [RG00] de la siguiente forma: '*Una página P se dice que es dinámica si todo o parte de su contenido se genera en tiempo de ejecución (es decir, después de que la petición de la página P sea recibida por el servidor) por un programa ejecutado en el cliente o en el servidor. Esto contrasta con una página estática P', cuyo contenido ya existe en el servidor, listo para ser transmitido al cliente cuando éste lo solicite*'. Se pueden considerar los siguientes tipos de dinamismo de contenido:

- Dinamismo temporal. Es el que está presente en las páginas que varían su contenido en función del instante de tiempo en el que son solicitadas. Por ejemplo, una página que visualiza valores bursátiles o los últimos titulares de noticias pueden considerarse en esta categoría. Para tratar este tipo de páginas en los *crawlers* convencionales, pueden utilizarse técnicas para mantener actualizado el contenido obtenido por un sistema de *crawling*, en función de las diferentes frecuencias de cambio de las páginas web (ver anexo VI.9)
- Dinamismo dependiente del cliente. Está presente en aquellas páginas que adaptan su contenido en función del cliente o usuario que las solicitan. Por ejemplo, los sitios web personalizan sus páginas (en términos de apariencia, comportamientos y contenido) para adaptarse a un usuario o comunidad de usuarios particular. Esto supone la generación de páginas en tiempo real, utilizando la información de *cookies* del lado cliente o autenticaciones explícitas para identificar a un usuario particular. Como las páginas con dinamismo basado en el cliente tienen contenido personalizado, obtener esas páginas puede no ser útil para aplicaciones orientadas a poblaciones heterogéneas de usuarios. Sin embargo, para determinadas aplicaciones, un *crawler* convencional restringido a sitios específicos puede estar equipado con las *cookies* o información de autenticación necesaria para permitir recorrer un conjunto fijo de sitios.

– Dinamismo de entrada. Las páginas cuyo contenido depende de datos introducidos por el usuario muestran dinamismo de entrada. El ejemplo típico lo constituyen las páginas generadas por un servidor como respuesta a envíos de formularios. Éste es el tipo de dinamismo de contenido que constituye un problema real desde el punto de vista de un *crawler* convencional. El problema reside en cómo obtener esas páginas, ya que no existen enlaces que apunten a las mismas, y sólo pueden obtenerse como respuesta al envío de una consulta utilizando un formulario web.

Además del dinamismo de contenido, existe otro tipo de dinamismo, cada día más frecuente, y al que no se ha dedicado apenas esfuerzo investigador hasta la fecha. Se trata del *dinamismo de navegación*. Mientras las páginas con dinamismo de contenido suelen generarse de forma dinámica en el servidor web, en el caso del dinamismo de navegación, existe código que se descarga y se ejecuta en la máquina cliente, típicamente en un entorno controlado proporcionado por el navegador. *Applets* Java, controles *ActiveX*, código *JavaScript* o *Ajax* son ejemplos de tecnologías que pueden ejecutarse en el lado cliente. Estas tecnologías permiten que sea el cliente web el que genere parte de una página de forma dinámica, o que una página se modifique dinámicamente ante la recepción de un evento de interfaz generado por un usuario. También permiten la reescritura, en tiempo de ejecución, de los URLs seleccionados por un usuario durante su proceso de navegación y la implementación de complejos sistemas de redirecciones.

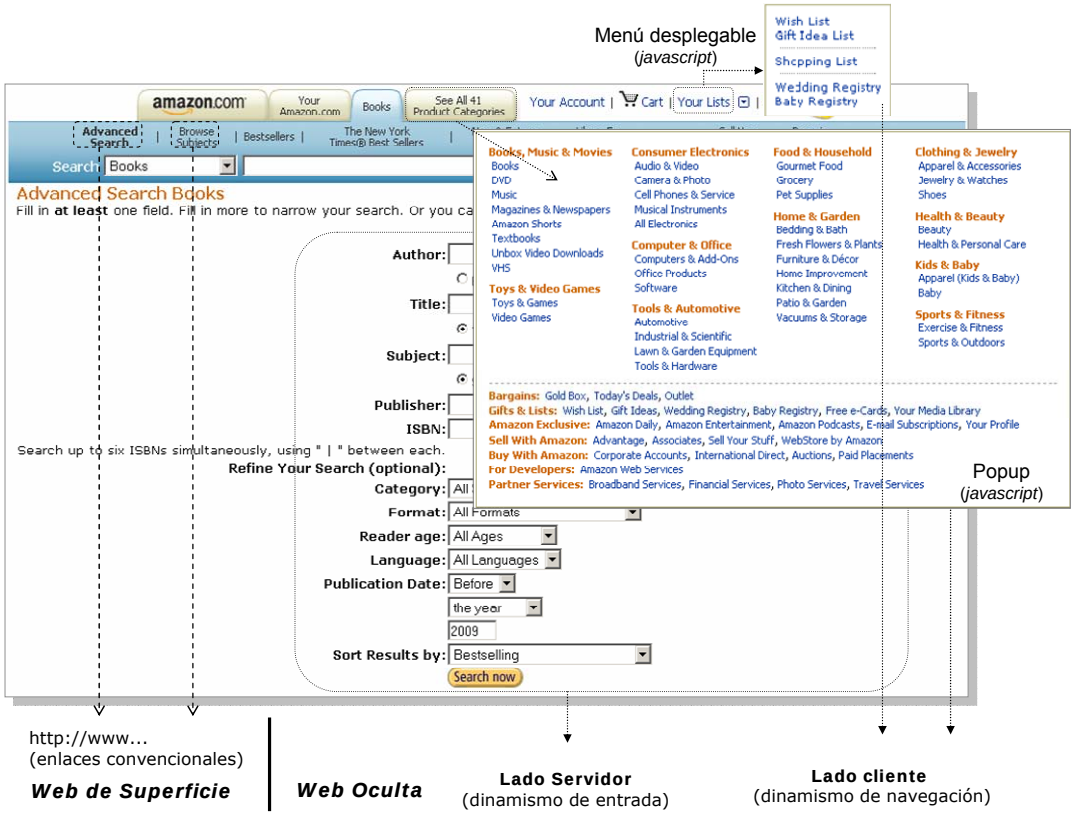


Figura 9 División de la Web en base a su visibilidad

Tanto los formularios web (dinamismo de entrada) como las tecnologías que presentan código embebido en el lado cliente (dinamismo de navegación) constituyen las barreras de la Web de Superficie que un *crawler* tiene que ser capaz de superar para acceder a la información contenida en la Web Oculta. En lo que resta de este trabajo, denominaremos a las técnicas que intentan solucionar los problemas derivados del dinamismo de navegación como técnicas para el tratamiento de la Web Oculta del lado cliente y, a las técnicas que intentan solucionar los problemas derivados del dinamismo de entrada, como técnicas para el tratamiento de la Web Oculta del lado servidor. En la Figura 9 se muestran los modos de acceso típicos a la información contenida en cada una de estas partes de la Web.

La Web Oculta del lado cliente está formada por aquellas páginas que aunque se encuentran directamente enlazadas desde recursos de la Web de Superficie, para obtener su contenido es necesario realizar algún procesamiento sobre ellas, en el que tratar con tecnologías del lado cliente. También incluye aquellos recursos que, aunque alcanzables siguiendo enlaces desde otras páginas, esos enlaces no incluyen la referencia completa al recurso, sino que para obtener el enlace real es necesario realizar algún procesamiento local tratando con tecnologías del lado cliente.

La Web Oculta del lado servidor está formada por todas aquellas páginas que presentan dinamismo de entrada, es decir, por el conjunto de páginas obtenidas como resultado de realizar una consulta sobre un formulario web. Los formularios constituyen el punto de acceso principal a la Web Oculta y suelen ser frontales de consulta contra un sistema de base de datos subyacente.

Como ya se ha comentado en la introducción (apartado I.1.1), se estima que el tamaño de la Web Oculta es varios órdenes de magnitud mayor que la Web de Superficie. Además, la información que contiene es de mayor calidad. Por estos motivos los motores de búsqueda han empezado a explorar diferentes alternativas para intentar abordar el *crawling* de la Web Oculta.

Existen directorios especializados en fuentes de la Web Oculta cuyo objetivo es permitir localizar páginas web con formularios que representan frontales de consulta contra bases de datos, organizados en diferentes categorías. Los más conocidos son *Companion Book Directory*[12], *Invisible Web*[13], *Complete Planet*[14], *LexiBot*[15] ó *Librarians' Internet Index*[16]. Según [CHLP+04], los directorios especializados sólo cubren un pequeño porcentaje de las bases de datos de la Web Oculta, y el procedimiento de clasificación utilizado es manual. Por otra parte, algunos sistemas poseen contratos especiales con determinados sitios de la Web Oculta, para proporcionarle algún mecanismo propietario para acceder a los datos que contiene. Existen también algunos protocolos como S*itemap* de Google [SiteMap] y OAI_PMH (*Open Archives Initiative Protocol for Metadata Harvesting*) [OAI_PMH], que permiten a un servidor hacer pública información que no está directamente accesible de otra

[12] http://www.invisible-Web.net
[13] http://www.invisibleweb.com
[14] http://www.completeplanet.com
[15] http://www.lexibot.com
[16] http://lii.org

forma. Algunos motores de búsqueda han empezado a utilizarlos, pero no dejan de ser intentos parciales, que dependen de que los servidores web proporcionen otros mecanismos para acceder a toda la información que contienen.

A nivel de investigación, la extracción de información contenida en la Web Oculta de forma automática se ha abordado desde dos aproximaciones diferentes: *crawling* y metabúsqueda. Un *crawler* de la Web Oculta recorre la Web de forma automática como un *crawler* convencional, pero adicionalmente localiza formularios web e intenta realizar consultas sobre ellos para obtener nuevos documentos. Un metabuscador es un sistema que proporciona un frontal de consulta común sobre diferentes formularios web correspondientes a sitios de una misma temática previamente definidos, para devolver la colección de resultados unificada y ordenada.

Para abordar la construcción de una arquitectura como la pretendida en este trabajo, es necesario un enfoque totalmente automático de exploración de la Web, como el que presentan los sistemas de *crawling*. Por ello, ese será el enfoque estudiado prioritariamente en este capítulo.

De todas formas, en el apartado II.3.2 se realiza una revisión bibliográfica de las técnicas utilizadas en la aproximación de metabúsqueda, porque ambas aproximaciones comparten algunos problemas. En el apartado II.3.3 se identifican los diferentes aspectos que es necesario considerar para una arquitectura de *crawling* dirigido de la Web Oculta como la objetivo de este trabajo y se expone el estudio de las principales problemáticas identificadas, haciendo especial énfasis en las más relacionadas con las principales aportaciones de este trabajo.

II.3.2. SISTEMAS DE METABÚSQUEDA

Un metabuscador es un sistema que agrupa interfaces de consulta de fuentes que pertenezcan a la misma temática, para proporcionar un frontal de búsqueda unificado. De esta forma, ante una consulta de un usuario, el sistema la realiza de forma automática sobre un conjunto de interfaces y devuelve al usuario un resultado global que agrupa las respuestas individuales obtenidas de cada uno de los sitios que integra.

Los sistemas de metabúsqueda surgieron inicialmente para intentar mejorar la calidad de los resultados devueltos por los buscadores web, debido a que aunque entre los motores de búsqueda más conocidos (Google, MSN, Ask/Teoma y Yahoo!) existe un cierto grado de solapamiento de contenidos, otros contenidos sólo están presentes en alguno de ellos [GS05]. En la actualidad, los sistemas de metabúsqueda se han utilizado para unificar consultas sobre formularios web de cualquier temática, no sólo de búsqueda web en general, y se utilizan en ámbitos como, por ejemplo, las herramientas de compra comparativa.

Si la gran cantidad de información disponible en Internet hace que los sistemas de *crawling* convencional no sean capaces de cubrir toda la Web de Superficie, el problema se agrava cuando el objetivo es acceder al contenido de la Web Oculta. Por este motivo, muchos trabajos de investigación relacionados con la extracción de información en la Web

Oculta utilizan la aproximación de metabúsqueda en lugar de la de *crawling*. En esta aproximación el objetivo es localizar puntos de acceso a la Web Oculta en el servidor (típicamente formularios de consulta) y caracterizarlos de forma automática para permitir realizar búsquedas de forma unificada desde un mismo frontal unificado.

A continuación se enumeran los principales problemas que es necesario abordar cuando se afronta la extracción de información de la Web Oculta siguiendo la aproximación de metabúsqueda. Estos enfoques dividen el problema en dos fases bien diferenciadas:

- Fase de generación del sistema.
    - Descubrimiento de formularios.
    - Caracterización de formularios.
    - Generación de la interfaz de consulta unificada.
- Fase de ejecución del sistema.
    - Selección de los formularios relevantes para resolver una consulta determinada.
    - Traducción de consultas, de un formato genérico a cada una de las fuentes involucradas.
    - Fusión de los resultados obtenidos de las diferentes fuentes.

En la fase de generación se obtienen y modelan los formularios web que va a considerar el sistema, para detectar las correspondencias entre campos de distintas interfaces y crear una interfaz de consulta unificada. También se obtiene información relativa al contenido proporcionado por cada fuente, que será utilizada en la siguiente fase para la selección de formularios relevantes.

En la fase de ejecución, ante una consulta de un usuario contra el frontal de consulta unificado, es necesario seleccionar los formularios sobre los que se realizará esa consulta de forma automática. Para esos formularios es necesario adaptar la consulta unificada al formato nativo de cada fuente. El sistema obtendrá los resultados de cada uno de los formularios seleccionados y los fusionará y ordenará, para devolver al usuario una visión unificada de los mismos.

El problema de descubrimiento y caracterización de formularios en sistemas de metabúsqueda también es aplicable a la aproximación de *crawling*, por lo que se comenta en detalle en el apartado II.3.3.

La detección de correspondencias de campos entre diferentes interfaces de forma automática es uno de los aspectos clave de este tipo de sistemas. Existen diferentes aproximaciones para calcular la concordancia entre esquemas: basadas en etiquetas y en instancias. En [RB01] se presenta una descripción de las diferentes aproximaciones. Los métodos basados en etiquetas sólo consideran la similitud entre las definiciones de etiquetas de los campos de las interfaces, y entre sus valores para aquellos campos que presentan una lista enumerada de valores posibles. Los métodos basados en instancias realizan consultas sobre diferentes formularios para comprobar, en función de los resultados devueltos, si dos campos en diferentes formularios identifican el mismo concepto. Estos últimos dependen

del solapamiento de contenidos o propiedades estadísticas como rango de datos y patrones para determinar la similitud entre dos atributos. Algunas de las soluciones presentes en los principales sistemas de metabúsqueda son [HC03] [HC06] [HMYW05b] [WWLM04] [WDC06].

En la fase de ejecución, para la selección de fuentes y ordenación de resultados se han propuesto algunos estándares. Por ejemplo, STARTS [GCGP97] permite a las fuentes exportar información que define sus contenidos como, por ejemplo, la lista de palabras contenidas en los documentos que indexan. Esta aproximación tiene el problema práctico de que en la Web no existe ningún mecanismo que haga que todas las interfaces de consulta implementen ese protocolo. Además, para permitir realizar ordenaciones de resultados de diferentes fuentes, requiere de algún tipo de cooperación entre las fuentes para que devuelvan un valor de importancia unificado, que *a priori* no tienen por qué conocerse. Las aproximaciones más utilizadas no asumen el uso de ningún estándar por parte de las fuentes, sino que realizan consultas cuidadosamente seleccionadas sobre las mismas para extraer documentos y generar una descripción del recurso a partir de los datos extraídos que permita decidir qué fuente seleccionar, como en [CC01] o [IGS01]. Para solucionar el problema de la ordenación, normalmente suelen ordenarse los resultados en base al análisis del contenido de cada uno de los documentos recuperados de cada interfaz de consulta.

Algunos de los sistemas más conocidos que utilizan la aproximación de metabúsqueda son MetaQuerier[17] [CHZ04] [HC06] [CHZ05] [ZHC05], WISE-Integrator [HMYW04] [HMYW05b] o QProber [GIS03] [IG02] [INCG05]. El sistema QProber está más orientado a la caracterización del contenido de las interfaces de búsqueda de la Web Oculta, para posteriormente utilizar esa información en los procesos de selección de los frontales más relevantes para resolver una consulta determinada.

Comparada con la aproximación de *crawling*, la metabúsqueda es más ligera puesto que no requiere indexar el contenido de las fuentes y además garantiza la actualidad de los datos. Sin embargo, los usuarios obtendrán tiempos de respuesta más altos debido a que las fuentes son consultadas en tiempo real.

Por otra parte, este tipo de sistemas no están pensados para un funcionamiento automático, sino que dependen de consultas realizadas por los usuarios, por lo que no son válidos para resolver el problema general planteado. En el apartado II.3.3 se comentarán en detalle las soluciones para el descubrimiento y caracterización de formularios que utilizan algunos sistemas de metabúsqueda, por ser una problemática aplicable a las aproximaciones de *crawling* y metabúsqueda.

[17] http://metaquerier.cs.uiuc.edu

II.3.3. SISTEMAS DE *CRAWLING* DE LA WEB OCULTA

II.3.3.1. INTRODUCCIÓN

Como se ha descrito en el capítulo I, se pretende definir una arquitectura de recopilación automática utilizando un enfoque dirigido, que habilite el acceso a la información contenida en la Web Oculta. El sistema parte de una especificación de una tarea, a la que vamos a denominar *dominio de aplicación*, y realiza un recorrido de la Web en base a ese dominio, a partir de unos sitios de inicio establecidos.

A continuación se enumeran los problemas que debe abordar un sistema de *crawling* guiado por un dominio, que sea capaz de recopilar, de forma estructurada, la información contenida en la Web Oculta:

- Tratamiento del dinamismo del lado cliente (dinamismo de navegación)
- Descubrimiento de formularios.
- Modelado de formularios.
- Establecimiento de la relevancia del formulario para el *dominio de aplicación* y aprendizaje de la forma de realizar las consultas deseadas sobre el mismo.
- Generación de consultas sobre el formulario e invocación del mismo para obtener páginas de resultados.
- Estructuración de las páginas de resultados.
- Indexación y búsqueda, tanto de páginas de resultados como de los registros extraídos de las mismas.

A continuación se describe brevemente cada uno de estos pasos. Las siguientes secciones describirán en detalle las aproximaciones propuestas para abordar cada uno de ellos.

Durante el recorrido de la Web, los *crawlers* tienen que considerar todos aquellos aspectos relacionados con el *crawling* de la Web Oculta del lado cliente, para poder alcanzar todos los sitios de la Web de Superficie a los que los usuarios convencionales pueden llegar. Además deben de realizar un *crawling* dirigido por la Web de Superficie para localizar los formularios relevantes para el tema objetivo.

Para tratar la Web Oculta del lado servidor, el primer desafío es la localización de interfaces de consulta web que constituyen los puntos de entrada a la información que se genera dinámicamente en los servidores web, en base a consultas realizadas por los usuarios. La mayor parte de las interfaces de consulta de la Web son formularios HTML, con lo que es posible ignorar otros tipos de interfaces como *applets* Java o *Flash* sin que esto suponga una limitación importante. Es sencillo realizar un recorrido de la Web para localizar páginas que contengan formularios, buscando etiquetas HTML de formularios. Sin embargo, no todos los formularios HTML representan interfaces de consulta. Por ejemplo, no entran en esta categoría los formularios de grupos de discusión, formularios de suscripción a listas de correo, formularios de compra en una tienda electrónica, formularios

de servicios web de correo electrónico o formularios de autenticación de usuarios en sitios web. De hecho, se estima que sólo algo más del 50% de los formularios son de consulta [CCH03]. Algunas aproximaciones existentes para la detección de interfaces de consulta utilizan clasificadores previamente entrenados a partir de las características que presentan conjuntos de formularios de búsqueda previamente seleccionados. Otras estrategias realizan consultas sobre los formularios y determinan si son interfaces de búsqueda analizando las respuestas obtenidas ante consultas predefinidas.

Una vez localizado un formulario web de consulta, es necesario realizar un análisis del mismo para crear un modelo que lo represente. Tras la caracterización de un formulario, es necesario aplicar algún algoritmo que permita determinar si el formulario es relevante para la temática considerada y, en ese caso, cómo debe rellenarse para ejecutar las consultas deseadas sobre el mismo.

Una vez se ha modelado el formulario y se ha caracterizado como perteneciente al dominio tratado, es necesario generar consultas sobre ese formulario, y ejecutarlas para obtener las páginas de resultado. Algunas aproximaciones consideran una serie de consultas predefinidas; otras en cambio intentan obtener toda la información contenida tras el formulario web.

Como resultado de realizar una búsqueda contra un formulario web se obtiene un documento que es necesario analizar para determinar si se ha producido algún error al realizar la búsqueda (invocación de un formulario sin especificar campos obligatorios, resultados no encontrados, etc.). En el caso de que no se haya producido ningún error, entonces se aplica algún algoritmo de estructuración que permita generar registros de información a partir de la estructura latente en la enumeración de resultados obtenidos en la página HTML resultado.

Como paso final, el sistema de *crawling* debe crear un índice que permita realizar tanto búsquedas por palabra clave sobre los documentos recolectados como consultas más precisas, estructuradas, que especifiquen condiciones complejas sobre los registros obtenidos como resultado del proceso de estructuración sobre las páginas de resultados.

En los siguientes apartados se comentan en detalle las aproximaciones existentes para resolver cada uno de los problemas apuntados. En el apartado II.3.3.2 se abordan las dificultades con las que se encuentra un *crawler* convencional para acceder a la denominada Web Oculta del lado cliente. En el apartado II.3.3.3 se presentan diversas aproximaciones para la localización de los formularios de consulta relevantes para una temática determinada. En el apartado II.3.3.4 se presentan los componentes de modelado de formularios de distintos sistemas existentes. En el apartado II.3.3.5 se analiza la problemática de cómo aprender a ejecutar las consultas deseadas sobre un formulario y en el apartado II.3.3.6 se comentan algunas aproximaciones para la generación de dichas consultas de forma automática. Por último, en II.3.3.7 se realiza una revisión de las principales técnicas de estructuración automática que pueden ser utilizadas para obtener los registros a partir de las páginas de resultados.

II.3.3.2. *CRAWLING* DE LA WEB OCULTA DEL LADO CLIENTE

Existen algunos problemas que dificultan a los sistemas de *crawling* tradicionales la obtención de datos de la parte cliente de la Web Oculta. Todos ellos están relacionados con la utilización de código que tiene que ejecutarse en los clientes HTTP [W3CHTTP] de los sistemas de *crawling*.

Las complejas páginas web actuales utilizan de forma intensiva lenguajes de *script* (principalmente *JavaScript*), mecanismos de mantenimiento de sesión, complejos sistemas de redirecciones, etc. Los desarrolladores utilizan las tecnologías de la parte cliente para añadir interactividad a las páginas web y para mejorar la navegabilidad a través de elementos de la interfaz como pueden ser menús emergentes, utilización de diferentes capas de datos, que se ocultan o hacen visibles dependiendo de las acciones del usuario, etc. Esta situación se ve agravada porque la mayoría de las herramientas utilizadas para crear sitios web de forma visual generan páginas que utilizan código de *script* para la generación del contenido y para mejorar los niveles de interacción en las navegaciones de los usuarios. Además, el reciente auge de las tecnologías que proporcionan interactividad cliente-servidor amigable, como *Ajax*, se basan fuertemente en el uso de *JavaScript*.

Aunque muchos de los sitios web más populares evitan la utilización de *JavaScript* y otras tecnologías similares para ser correctamente indexados por motores de búsqueda globales como Google, sitios web de tamaño medio conteniendo información de gran valor continúan utilizándolas de forma intensiva. Éste es el caso especialmente de sitios web que requieren suscripción o autenticación de usuario, debido a que estos sitios no tienen ningún incentivo para facilitar el trabajo a los grandes motores de búsqueda. Sin embargo, esta clase de sitios normalmente son los que proporcionan la información más valiosa para muchas aplicaciones de búsqueda dirigida, como vigilancia tecnológica o motores de búsqueda verticales (para unas temáticas determinadas).

En los siguientes apartados se describen en detalle los diferentes problemas que presentan las tecnologías del lado cliente a los sistemas de *crawling* convencionales, que a continuación se enumeran:

- Lenguajes de *script* de la parte cliente
- Mecanismos de mantenimiento de sesión
- *Applets* y código *Flash*
- Otras problemáticas

En cada apartado, cuando existan, se describen también las aproximaciones utilizadas en trabajos previos para abordar el problema. Cabe adelantar que, en líneas generales, el problema del *crawling* de la Web Oculta del lado cliente ha recibido muy poca atención hasta la fecha por parte de la comunidad investigadora, que ha tendido a centrarse en la Web Oculta del lado servidor.

II.3.3.2.1. LENGUAJES DE *SCRIPT* DE LA PARTE CLIENTE

Muchas páginas HTML hacen uso intensivo de *JavaScript* y de otros lenguajes de *script* que se ejecutan en el navegador web (como *Jscript* o *VBScript*) para diferentes propósitos como:

- Generar contenido en tiempo de ejecución (e.g. utilizando el método *JavaScript* `document.write`).
- Generar nuevas opciones de navegación en respuesta de acciones del usuario. Los sistemas de crawling convencional, cuando obtienen una nueva página, la analizan para obtener nuevos enlaces para procesar y añadir a la lista de URLs a acceder. El código de *script* complica esta situación porque puede ser utilizado para generar o eliminar enlaces de forma dinámica, en respuesta a algunos eventos. Por ejemplo, muchas páginas web utilizan enlaces para representar opciones de menús. Cuando se hace clic sobre un enlace que representa una opción, el código *script* genera de forma dinámica una lista conteniendo nuevos enlaces que representan las sub-opciones. Si se hace clic de nuevo sobre el enlace del menú principal, entonces el código de *script* pliega el menú de nuevo, eliminado los enlaces correspondientes a las sub-opciones. Un crawler que pretenda tratar con la Web Oculta del lado cliente debe de ser capaz de detectar estas situaciones y obtener todos los enlaces 'ocultos'.
- Generar dinámicamente el URL asociado a un enlace o formulario. Puede utilizarse código de *script* en el valor del atributo `href` de un enlace que genera dinámicamente el URL al que navegar cuando el usuario hace clic. También es posible utilizar código de *script* para generar o reescribir el URL asociado a un formulario.
- Utilización de tecnologías *Ajax* para realizar invocaciones al servidor web que no involucren recargar en el navegador la página activa, proporcionando mayor sensación de dinamismo al usuario.

Gestionar de forma apropiada lenguajes de *script* requiere que los clientes HTTP implementen todos los mecanismos que hacen posible a un navegador visualizar una página y generar nuevas navegaciones. Esto involucra seguir enlaces y ejecutar todas las acciones asociadas a los eventos que se disparen de forma automática al seguir esos enlaces.

Algunos sistemas de *crawling* [WC07] han incluido intérpretes de *JavaScript* [MR07] [MSM07] en los clientes HTTP que utilizan para proporcionar algún soporte, aunque limitado, para tratar con páginas que utilizan lenguajes de *script*. Sin embargo, estos sistemas sólo se ocupan de la parte de generación del contenido de la página, pero no tratan los problemas derivados de la aparición dinámica de nuevas opciones de navegación en respuesta a acciones del usuario ni las tecnologías *Ajax*.

Además, el uso de un intérprete de *script* específico no soluciona totalmente tampoco el problema de generación de nuevo contenido, debido a que los navegadores proporcionan un entorno de ejecución especial para los lenguajes de *script*, en el que determinados objetos propios del navegador están accesibles, por lo que se hace necesario reproducir ese entorno para que todos los *scripts* funcionen correctamente. Por lo tanto, en la práctica, estos

sistemas pueden ejecutar con éxito sólo una pequeña parte de los *scripts* encontrados dentro de las páginas web. De forma adicional, en algunas situaciones como páginas con varios marcos (*frames*), no es siempre sencillo localizar y extraer el código para ser interpretado. Esto es debido a que la mayor parte de los *crawlers* actuales, incluyendo los utilizados en los motores de búsqueda más populares, no proporcionan soporte para esa clase de páginas.

II.3.3.2.2. MECANISMOS DE MANTENIMIENTO DE SESIÓN

Muchos sitios web utilizan mecanismos de mantenimiento de sesión basados en recursos del lado cliente, como *cookies* o código de *script* para añadir parámetros de sesión a los URLs antes de enviarlos al servidor. Esto provoca los siguientes problemas:

- Como ya se ha mencionado, mientras la mayor parte de los *crawlers* son capaces de tratar con *cookies*, no sucede lo mismo con el código de *script*.
- El *crawling* distribuido constituye otro problema. Las arquitecturas convencionales para *crawling* se basan en una lista principal de URLs compartidos, de la que cada proceso de *crawling* (que pueden ejecutarse en diferentes máquinas) obtiene URLs y accede a ellos de forma independiente, en paralelo. Sin embargo, en sitios web que utilizan las sesiones para gestionar su modo de funcionamiento, es necesario asegurar que cada proceso de *crawling* tenga toda la información de sesión necesaria disponible (como *cookies* o el contexto para ejecutar código de s*cript*). Si el entorno no es el adecuado, entonces el acceso al documento en el servidor fallará. Los sistemas de *crawling* convencionales no tratan con estas situaciones.
- Otro problema lo constituye el acceso futuro a los documentos que han sido recolectados. Las páginas obtenidas no se almacenan normalmente en local, sino que se indexan en base a sus URLs. Cuando posteriormente un usuario obtiene la página como resultado de una consulta contra el índice, puede acceder a la página a través de su URL. Sin embargo, en un contexto en el que existen aspectos relacionados con el mantenimiento de sesión, el acceso a un URL puede no devolver el documento adecuado. Por ejemplo, el URL puede incluir números de sesión que expiren pocos minutos después de ser creado. Los sistemas de *crawling* presentados hasta la fecha tampoco se ocupan de este problema.

II.3.3.2.3. APPLETS Y CÓDIGO FLASH

Otros tipos de tecnologías utilizadas en la parte cliente son los *applets* y el código *Flash*. Se ejecutan en la parte cliente, con lo cual es necesario implementar un componente contenedor cliente que los procese. Aunque acceder al contenido mostrado por programas escritos en estos lenguajes es complejo debido a que están compilados, un *crawler* web al menos debería ser capaz de tratar con la situación habitual en la que estos componentes son utilizados como una introducción que finalmente redirige al usuario a una página convencional en la que el *crawler* puede continuar su tarea. Este problema no ha sido abordado por los sistemas de *crawling* previos.

II.3.3.2.4. OTRAS PROBLEMÁTICAS

Elementos de páginas web como marcos, HTML dinámico o HTTPs acentúan los problemas comentados previamente. En términos generales se puede decir que es muy difícil considerar todos los factores que pueden hacer un sitio web visible y accesible para un navegador.

II.3.3.3. LOCALIZACIÓN DE FORMULARIOS EN CRAWLING DIRIGIDO

Uno de los desafíos que presenta el tratamiento de la Web Oculta es la localización de los puntos de acceso a las bases de datos en ella embebidas. Un estudio reciente de Chang et al. [CHLP+04] estima que Google indexa 9000 millones de páginas y que el número de sitios en la Web Oculta es de 307000, con una media de 4.2 interfaces de consulta por sitio. Localizar formularios mediante un *crawling* completo de la Web sería altamente ineficiente, porque el número de formularios es demasiado elevado y la proporción de formularios respecto al número total de páginas web es pequeña. Por lo tanto, se hace necesario localizar formularios relevantes para una determinada temática mediante un enfoque de *crawler* dirigido.

Barbosa y Freire proponen en [BF05] una estrategia de *crawling* dirigido para localizar de forma automática bases de datos de la Web Oculta que ayude a alcanzar un equilibrio entre los dos requisitos conflictivos de este problema: la necesidad de realizar una búsqueda general mientras al mismo tiempo se evite la necesidad de recorrer un gran número de páginas irrelevantes. Los autores usan como punto de partida la arquitectura de *crawling* propuesta por Chakrabarti et al. [CBD99] (ver sección II.2.1), por lo que utilizan un clasificador de documentos para guiar el *crawler* y dirigir la búsqueda hacia páginas que pertenecen a una temática determinada. Adicionalmente, para conseguir mayor efectividad en la búsqueda, y al igual que hicieron Rennie y MacCallum en [RM99], este *crawler* aprende a identificar enlaces prometedores, incluyendo aquellos cuyo beneficio puede no ser inmediato – en este caso, aquellos que es probable que lleven a páginas que contengan formularios, en uno o varios pasos –. Al igual que Diligenti et al. en [DCLG+00], en lugar de crear el grafo de referencia a partir de varios recorridos sobre sitios seleccionados (lo cual puede ser prohibitivo para recorridos de gran amplitud), utilizan las facilidades de '*backward crawling*' proporcionadas por algunos buscadores para obtener las páginas que apuntan a una dada, para crear una aproximación de este grafo.

Adicionalmente, en base a características específicas de formularios, introducen un nuevo criterio de parada para guiar el *crawler* y evitar un exceso de trabajo especulativo en un único sitio web:

- El *crawler* abandona un sitio si recupera un número predefinido de formularios diferentes. Según Chang et al. [CHLP+04] los sitios web ocultos tienen un número reducido de interfaces de consulta (en media 4.2 interfaces).
- El *crawler* también abandona un sitio, si visita un número máximo de páginas permitidas. Esta condición es necesaria para evitar recorrer sitios web completos que no presenten formularios de consulta.

En la parte izquierda de la Figura 10 se muestra la arquitectura del *crawler* de formularios FFC (*Form Focused Crawler*). Utiliza dos clasificadores para guiar el recorrido: el de página y el de enlace. Adicionalmente utiliza un clasificador de formularios, para determinar si un formulario es de búsqueda o no.

El clasificador de formularios es un clasificador general (independiente de la temática) que utiliza un árbol de decisión para determinar si un formulario es de búsqueda.

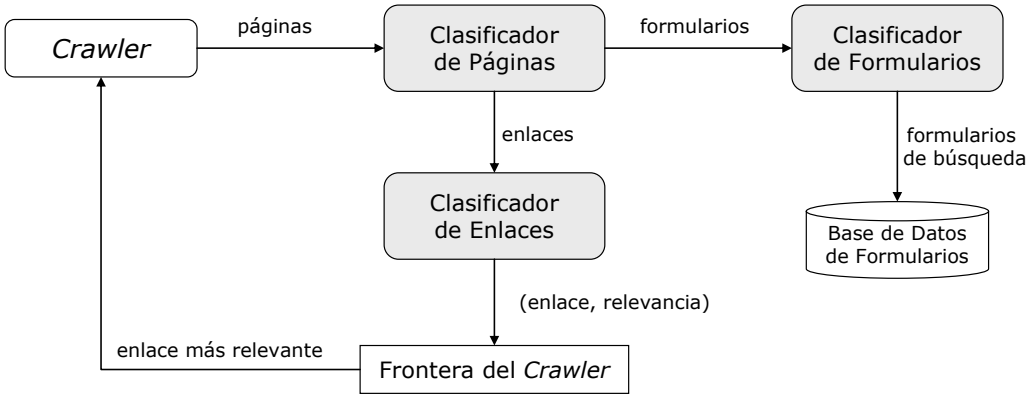


Figura 10 Arquitectura de un *crawler* de formularios (izquierda FFC, derecha ACHE)

La frontera del *crawler* está formada por *N* colas (*N* es el número de niveles en el clasificador de enlaces, y cada cola tiene asociados los enlaces clasificados como pertenecientes a su nivel). El *crawler* prioriza los enlaces que están más próximos de las colas correspondientes a los niveles más bajos. Dentro de una cola, los enlaces se ordenan por probabilidad de pertenecer al nivel respectivo. Sin embargo, las páginas próximas a la raíz de los servidores se sitúan antes en sus colas respectivas, debido a la observación realizada por Chang et al. en [CHLP+04] de que los formularios frecuentemente se encuentran próximos a las páginas principales de los sitios web.

Los resultados experimentales sobre tres dominios diferentes muestran que, incluso utilizando un grafo de conectividad aproximado, este *crawler* es más eficiente (hasta un orden de magnitud) que un conjunto de *crawlers* convencionales representativos. En este caso la eficiencia se define como la capacidad de procesar el menor número de páginas irrelevantes. Los experimentos además muestran la ventaja añadida por la combinación de un *crawler* dirigido con la identificación de enlaces que conducen a páginas que contienen formularios: el *crawling* dirigido sobre un tema ayuda a mejorar la efectividad del clasificador de enlaces, debido a que las características que son aprendidas para enlaces son frecuentemente específicas para un tema/dominio.

Sin embargo, este sistema posee dos limitaciones importantes. Por una parte, el conjunto de formularios recuperados es altamente heterogéneo al ser únicamente capaz de diferenciar formularios de búsqueda del resto, sin distinguir entre formularios de búsqueda de diferentes temáticas. Por otra parte, requiere una fase de entrenamiento manual para la creación del clasificador de enlaces, que además es bastante dependiente del conjunto de formularios utilizados como ejemplo, que tiene que ser representativo para el dominio tratado – esto puede ser complicado debido al tamaño y variabilidad de la Web –.

Los mismos autores, para solucionar las limitaciones comentadas, en [BF07] presentan ACHE (*Adaptative Crawler for Hidden-Web Entries*), que complementa FFC y cuya arquitectura se muestra en la Figura 10. ACHE parte de un conjunto de formularios de entrada y automáticamente localiza otros formularios pertenecientes al mismo dominio. Los autores modelan el problema como una tarea de aprendizaje, en la que el *crawler* aprende de experiencias previas. Incluyen en el sistema un componente para clasificar un formulario como perteneciente a una temática, en base a los textos contenidos en el formulario, y definen un algoritmo para seleccionar características de enlaces de las páginas que han sido consideradas satisfactorias para ir reconstruyendo el clasificador de enlaces. Se proponen y evalúan dos estrategias para el *crawling*: una completamente automática en la que el sistema construye el clasificador de enlaces desde cero, y otra semiautomática que parte de un clasificador de enlaces previamente creado y lo va refinando para adaptar la dirección del sistema a formularios de la temática objetivo. En sus resultados experimentales verifican la mejora en la efectividad de la nueva aproximación.

Existen otros sistemas en la bibliografía que se han ocupado del problema de localizar los formularios de consulta. Bergholz et al. Una de las aproximaciones de *crawling* dirigido utilizadas por Bergholz et al. en [BC03] se basa en obtener documentos pertenecientes a determinadas categorías del directorio de Google. El proceso de *crawling* es local a los sitios de inicio de *crawling* para así evitar el acceso a otros sitios web que puedan pertenecer a otras temáticas. Adicionalmente acotan el número de páginas descargadas por sitio y el nivel de profundidad. En base a sus experimentos concluyen que los formularios web están en niveles de profundidad bajos (1-3) lo que apoya la idea de que el *crawling* no necesita ser muy profundo para encontrar la mayor parte de los formularios.

Cope et al. [CCH03] proponen el recorrido de la Web para localizar formularios utilizando una técnica basada en la generación automática de características para formularios web y árboles de decisión. Los autores concluyen que uno de los árboles de decisión que utilizan es aplicable de forma general para la detección de cualquier formulario de consulta, aunque no se ocupa de clasificarlos según su temática.

Por último, existen servicios en Internet como *InvisibleWeb.com*, que proporcionan acceso sencillo a miles de bases de datos accesibles desde la Web, organizando los enlaces a esas bases de datos según su temática en un árbol de categorías (en base a la ayuda de expertos). Otros servicios como *BrightPlanet.com* automáticamente identifican, clasifican y categorizan el contenido almacenado en la Web Oculta. En ambos casos las técnicas son propietarias y no se ha publicado información sobre las mismas. De todas formas, constituyen un buen punto para obtener interfaces de consulta de la Web Oculta, previamente clasificadas en temáticas.

II.3.3.4. MODELADO DE FORMULARIOS

Una vez localizada una página conteniendo un formulario de consulta, un usuario rellena cada campo del formulario con los valores adecuados y posteriormente lo envía al servidor para obtener los datos de respuesta. Para llevar a cabo este proceso, el usuario debe

leer el formulario e interpretar la semántica de cada uno de sus campos, para después rellenarlo de tal forma que represente correctamente los requisitos de su consulta.

Para que un sistema automatizado pueda realizar esas tareas, el primer problema a resolver consiste en crear un modelo del formulario que permita descubrir la semántica de cada uno de sus elementos, es decir, las capacidades de consulta permitidas por el formulario web. En fases posteriores se utilizará esa información para generar de forma automática consultas sobre los mismos.

En esta sección se describen las principales técnicas propuestas para abordar este problema. En el apartado II.3.3.4.1 se describe en detalle el componente de análisis de formularios utilizado por el *crawler* de la Web Oculta HiWE. También se describen los componentes encargados de modelar los formularios de páginas web en los sistemas de metabúsqueda MetaQuerier (apartado II.3.3.4.2) y WISE-Integrator (apartado II.3.3.4.3). Por último, en el apartado II.3.3.4.4 se describe la aproximación utilizada por el sistema descrito por Kaljuvee et al. en [KBGP01], cuyo objetivo es la adaptación automática de formularios de consulta para ser mostrados en dispositivos móviles.

II.3.3.4.1. HIWE

En [RG01], Raghavan y García-Molina presentan el sistema de *crawling* HiWE (*Hidden Web Exposer*), que es capaz de reconocer y rellenar automáticamente formularios relevantes para una tarea de obtención de información determinada. En el anexo VII se presenta la arquitectura general del *crawler* HiWE. Este apartado describe en detalle su módulo de análisis de formularios.

El sistema HiWE representa un formulario F como un conjunto de elementos (o campos). La información sobre cada elemento contiene dos partes:

- Un dominio D_i, que representa el conjunto de valores que pueden ser asociados a ese elemento. Algunos elementos tienen dominios finitos, porque el conjunto de valores posibles está embebido en la página HTML (e.g. listas de selección). Sin embargo, otros elementos, como las cajas de texto, tienen dominios infinitos.
- Una etiqueta E_i, que es la información descriptiva asociada con ese elemento. Será nula cuando no esté disponible o no pueda ser extraída. El objetivo es obtener y asignar a cada campo el texto descriptivo que suele aparecer en el formulario para ayudar a los usuarios a entender la semántica de los elementos. Por ejemplo, en un formulario de búsqueda de libros, al lado del campo que permite buscar por título aparecerá una etiqueta 'Title:' o similar.

Además, el modelo de un formulario incluye información necesaria para su envío, como el URL e identificadores internos (S), y metainformación del formulario como la web en la que se encuentra, el conjunto de páginas que enlazan a la del formulario e incluso textos próximos al formulario (M). De forma abreviada, un formulario F se representa como $F= (\{(E_1, D_1), (E_2, D_2), \ldots, (E_n, D_n)\}, S, M)$. En las pruebas que realizaron no consideraron el parámetro M, es decir, los autores consideraron que era el conjunto vacío.

El objetivo del *Analizador de Formularios* es procesar una página que contiene formularios para extraer toda la información necesaria para construir el modelo del

formulario. El mayor desafío que presenta esta etapa consiste en extraer correctamente las etiquetas y dominios de los elementos del formulario. Para ello, se utiliza un motor de visualización simplificado denominado LITE (*Layout based Information Extraction Technique*) para calcular distancias visuales entre los elementos y etiquetas del formulario. También se usan heurísticas basadas en cómo un usuario humano interpreta las asociaciones entre textos y elementos de un formulario, de acuerdo con su disposición visual. Los pasos seguidos por el módulo son los siguientes:

- Podar la página para eliminar la parte que no influye directamente en la visualización de los elementos del formulario y sus etiquetas.
- Utilizar un motor de visualización para representar la página podada, descartando las imágenes, e ignorando tamaños y estilos de las fuentes y hojas de estilos.
- Utilizando el motor de visualización, identificar los 4 textos, si existen, que están visualmente más cercanos a cada elemento del formulario en dirección horizontal y vertical. Estos textos se consideran *candidatos*. Para elementos del formulario que involucren varios campos, tales como por ejemplo conjuntos de `checkboxes`, las distancias se calculan en relación al centro del grupo.
- Asignar una puntuación a cada texto *candidato* en función de un conjunto de medidas que tienen en cuenta su posición, tamaño y estilo de fuente, número de palabras, etc. Los textos a la izquierda o encima de un elemento tendrán más puntuación que el resto.
- Elegir el candidato con mayor puntuación como la etiqueta asociada con el elemento del formulario. En caso de empate, se selecciona uno al azar.
- Realizar los post-procesamientos que sean necesarios para normalizar el texto:
    - Eliminar etiquetas HTML y referencias a entidades HTML.
    - Eliminar caracteres no alfanuméricos.
    - Convertir caracteres a minúsculas.
    - Eliminar las palabras usuales (*stop words*).
    - Obtener las raíces de las etiquetas, utilizando el algoritmo de Porter [Porter80].

Esta aproximación presenta las siguientes debilidades:

- Los métodos utilizados para la selección de los textos son relativamente simples y pueden fallar con relativa facilidad. Por ejemplo, puede ocurrir que un campo se quede con un texto candidato de otro, en el caso de que visualmente quede más próximo.
- Sólo asocia un texto con cada campo del formulario, cuando en realidad podría haber varios y esta información es de gran utilidad para posteriormente ser capaces de realizar consultas sobre el formulario.
- No es capaz de reconocer dependencias entre campos del formulario (por ejemplo, si hay un selector para provincia y otro para localidad, los valores del selector de

localidad dependen de la provincia seleccionada). Esta última limitación la plantean los propios autores de HiWE como trabajo futuro.

II.3.3.4.2. METAQUERIER

Zhang et al. presentan en [ZHC04] una técnica para modelar interfaces de consulta web. Se basan en la observación de que los formularios de consulta comparten ciertas similitudes estructurales, desde el punto de vista de que todos se construyen modularmente, a partir de unos bloques básicos, como se muestra en la Figura 11. Para capturar esta idea parten de la hipótesis de que existe una gramática 'oculta', o implícita, sobre un lenguaje visual, conforme a la cual se construyen los formularios. Desde esta perspectiva el problema de comprender la semántica de un formulario de consulta se convierte en un problema de análisis sintáctico. Pero dado que la gramática *oculta* es hipotética, los autores deben de crear una en base a la observación. Esta gramática es inherentemente incompleta y ambigua, es decir, habrá patrones que no capture y podrán existir conflictos entre patrones.

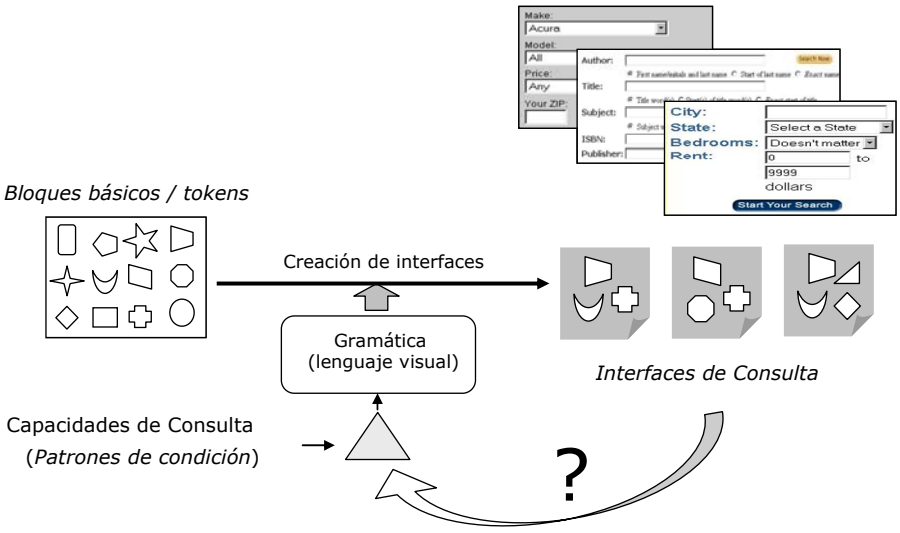


Figura 11 Observación e hipótesis base de MetaQuerier

A diferencia de los lenguajes de cadenas tradicionales (lenguajes de programación por ejemplo), la sintaxis utilizada para construir los patrones de condición considerados en los formularios usa información visual para expresar la semántica embebida en ellos. Por ejemplo un patrón de condición puede representar a una cadena situada a la izquierda de un campo de texto y ambos alineados verticalmente. La cadena de texto representaría probablemente la etiqueta que identifica la semántica del campo. Otro patrón común puede añadir al anterior una lista de *checkboxes* debajo del campo de texto, cada *checkbox* con una etiqueta de texto asociado, y donde cada *checkbox* permite escoger un operador de consulta (e.g. 'contiene alguna de las palabras', 'frase exacta', etc.).

Por tanto, los formularios de consulta pasan a ser vistos como composiciones de un lenguaje formal, en particular de un lenguaje visual, conformes a una gramática oculta, no establecida. La utilización de la gramática junto con un analizador sintáctico proporciona un marco de trabajo para especificar y reconocer los patrones comunes. El analizador no sólo debe tener en cuenta los patrones individuales, sino también su ensamblaje coherente para resolver conflictos locales a través del contexto global del formulario. Además, no puede rechazar ningún formulario de entrada como ilegal, aunque no sea capaz de analizarlo completamente.

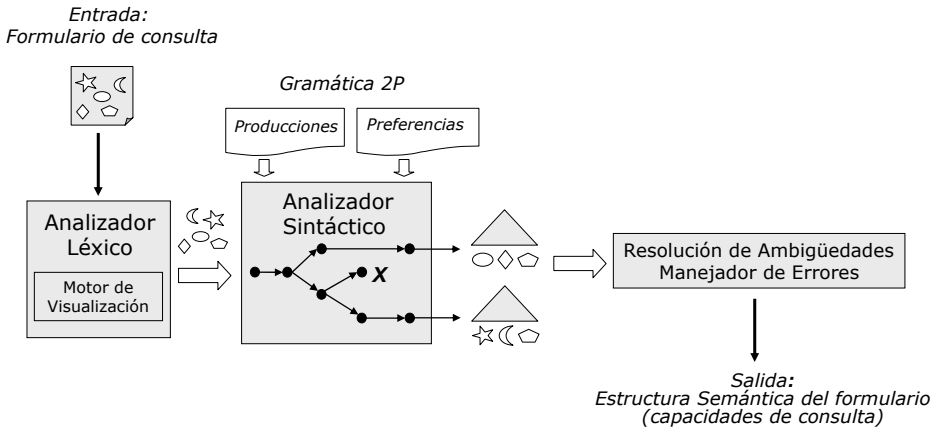


Figura 12 Arquitectura de extracción de consultas usando gramáticas

En la Figura 12 se muestra el funcionamiento global del sistema. Toma como entrada un formulario HTML de consulta y produce como salida su modelo semántico o capacidades de consulta. Como pre-procesamiento, un módulo prepara la entrada para el analizar sintáctico, convirtiendo el formulario HTML en un conjunto de *tokens* básicos, que constituyen las unidades atómicas de las composiciones gramaticales visuales. Cada *token* tiene un tipo y algunos atributos que almacenan propiedades necesarias para el análisis. Por ejemplo, existe una propiedad que se calcula para todos los *tokens* y que tiene como valor las coordenadas gráficas del elemento, obtenidas a partir de un motor de visualización de HTML. Como post-procesamiento, otro módulo integra la salida del analizador sintáctico para generar el modelo sintáctico final. En caso de que la salida del analizador produzca varios árboles parciales de análisis, que abarquen diferentes partes del formulario, la unión de algunos de ellos puede constituir un resultado final mejor. Este módulo también informa de errores, cuando se producen conflictos (si el mismo *token* se utiliza en diferentes condiciones), o cuando se *pierden* algunos elementos (no aparecen en ningún árbol de análisis).

Para representar las convenciones de presentación de las interfaces de consulta, los autores utilizan una gramática de dos dimensiones (gramática 2P), para especificar de forma declarativa y comprensiva patrones de condición (a través de las producciones) y sus precedencias (a través de las preferencias). La gramática 2P es una 5-upla que incluye:

- Conjuntos de símbolos terminales, que son los elementos que pueden aparecer en un formulario (text, textbox, checkbox, radiobutton, etc.).

- Conjuntos de símbolos no terminales, que definen formas posibles de disponer los elementos en el formulario para construir un patrón de condición (o una parte de un patrón de condición).
- El símbolo no terminal de inicio.
- Un conjunto de reglas de producción que utilizan predicados predefinidos para representar la posición relativa de los elementos: left(a,b), right(a,b), below(a,b), above(a,b).
- Un conjunto de preferencias que especifican precedencia entre producciones, para evitar ambigüedades.

La gramática utilizada (sin considerar las reglas de precedencia) es una instancia especial de una *attributed multiset grammar* [Golin91]. En este tipo de gramáticas se utilizan un conjunto de relaciones espaciales para capturar información topológica en las producciones. En las gramáticas bidimensionales, las producciones necesitan capturar relaciones espaciales, que básicamente son restricciones a ser verificadas por los constructores asociados a cada producción.

La idea básica consiste en crear de forma manual una gramática que permita definir la forma en que se disponen visualmente los 'patrones de condición' utilizados más habitualmente en la Web. En la Figura 13 se muestra un ejemplo de gramática.

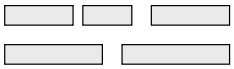
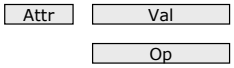
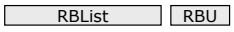
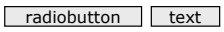
#	*Producciones*	*Patrones Visuales*
P1	**QI** ← HQI | Above(QI,HQI)	
P2	**HQI** ← CP | Left(HQI,CP)	
P3	**CP** ← TextVal | TextOp | EnumRB	
P4	**TextVal** ← Left(Attr,Val) | Above(Attr,Val) | Below(Attr,Val)	
P5	**TextOp** ← Left(Attr,Val) Λ Below (Op,Val)	
P6	**Op** ← RBList	
P7	**EnumRB** ← RBList	
P8	**RBList** ← RBU | Left(RBList, RBU)	
P9	**RBU** ← Left(radiobutton, text)	
P10	**Attr** ← text	
P11	**Val** ← textbox	

Figura 13 Ejemplo de gramática

En el ejemplo, para capturar el patrón TextOp se especifica que Attr está posicionado a la izquierda de Val y Op debajo de Val (en la Figura 14 se muestra el patrón visual que representa). A su vez, Op representa una lista de *radio buttons*, cada uno de ellos con un texto a su derecha. Es una disposición visual típica para representar un patrón de condición (Attr, {op1, …, opn}, text), donde los *radio buttons* de la parte inferior de la caja de texto permiten seleccionar el operador de consulta ('Contains', 'Starts with', 'Exact phrase').

Figura 14 Representación visual de un patrón de condición

Una vez definida esta gramática, el problema de obtener la estructura de un formulario se trata como el análisis sintáctico de un 'programa' escrito de acuerdo a la gramática: cada árbol de análisis válido para el formulario constituye una interpretación válida del mismo de acuerdo a los patrones de condición definidos en la gramática.

Como se comentó con anterioridad, las ambigüedades son inherentes al enfoque. Esto puede hacer que el análisis sea muy ineficiente. Por ello, se permite especificar reglas de precedencia que, en determinadas situaciones de conflicto, especifican cuál es la instancia ganadora. Una regla de precedencia es una 3-upla que indica los tipos de instancias sobre los que se produce el conflicto, una expresión lógica que especifica la situación de conflicto y cual es la instancia ganadora. Ejemplos de reglas de precedencia para la gramática anterior serían:

- Cuando una instancia `RBU` y una instancia `Attr` entran en conflicto por un token `text`, gana la primera. Esto se traduce en que si un campo de texto y un *radio button* compiten por el mismo texto, se le asigna al *radio button*.
- Cuando dos instancias de `RBList` entran en conflicto, si una subsume a la otra, se elige la más larga como la ganadora. Esto se traduce en que si dos listas de *radio buttons* entran en conflicto y una subsume a la otra, se considera solo la más larga.

El algoritmo de análisis sintáctico utiliza la técnica de evaluación de punto fijo [Golin91] [HMO91] [WWT91] para ir construyendo progresiva y concurrentemente múltiples árboles de análisis. La idea esencial consiste en ir creando nuevas instancias, aplicando las producciones hasta llegar a un punto fijo en el que no se puedan generar nuevas instancias. Debido a la posible incompletitud de la gramática utilizada, en este caso podría no llegar a generarse un árbol completo, pero sí varios árboles parciales. El algoritmo debe maximizar los resultados parciales para interpretar la mayor parte de la entrada posible. Para esto se buscan los árboles parciales que interpreten un conjunto máximo de *tokens* no subsumidos por ningún otro árbol de análisis.

Los autores presentan un estudio en el que han incluido 150 fuentes de tres dominios diferentes (libros, coches y líneas aéreas), en las que sólo han encontrado 25 patrones diferentes. Una de las conclusiones de este estudio es que según va aumentando el número de fuentes consideradas, el número de patrones diferentes encontrados crece con mucha mayor lentitud. Otras conclusiones son que los patrones son independientes del dominio de las fuentes y que siguen una distribución Zipf, es decir, que un pequeño número de patrones se usa con mucha frecuencia. También concluyen que dado un formulario cualquiera, es posible comprenderlo, descomponiéndolo en patrones conocidos utilizando un enfoque de 'divide y vencerás'.

Los experimentos presentados ofrecen buenos resultados. Entre otros experimentos, los autores realizan uno con 30 formularios seleccionados de forma aleatoria (en cualquier dominio) de *invisibleweb.net* y se obtiene `0.8` en precisión y `0.89` en alcance extrayendo patrones de condición. Se define en este caso la precisión como el número de patrones de

condición correctamente extraídos, y el alcance como el número de patrones de condición correctamente identificados del total de patrones de condición que deberían haber sido extraídos. Para el 55% se obtienen valores de precisión y alcance 1 (es decir, extracción perfecta).

La potencia del enfoque utilizado estriba en que las decisiones sobre cómo agrupar elementos en patrones de condición utilizan el contexto global proporcionado por el resto de elementos del formulario. De esta forma, por ejemplo, la decisión de asociar o no una etiqueta a un campo no se toma sólo en base a medidas locales (como la distancia visual) entre el campo y la etiqueta.

Como puntos débiles puede decirse que es muy sensible a los patrones básicos de construcción de formularios, considerados para generar la gramática hipotética utilizada, que han de ser obtenidos a través de la observación. También es muy sensible a las reglas de precedencia utilizadas, que son especificadas de forma ad-hoc. Estas reglas también disminuyen la ventaja del enfoque de tomar decisiones basadas en el contexto global del formulario, ya que su utilización implica la toma de decisiones en función de contextos locales para disminuir la complejidad del algoritmo. Además las reglas de precedencia son fijas y no es posible que en una parte local del árbol de análisis se dé preferencia a una de las opciones y en otra parte se de precedencia a la otra.

II.3.3.4.3. WISE-IEXTRACTOR

En [HMYW05a] se describe WISE-iExtractor (*Web Interfaces of Search Engines*), una herramienta de extracción automática de interfaces de consulta web complejas. Se corresponde con el componente de modelado de las interfaces de consulta web del sistema de metabúsqueda WISE-Integrator [HMYW05b].

El extractor de interfaces toma como entrada un conjunto de páginas HTML conteniendo las interfaces de consulta de múltiples fuentes del mismo dominio y para cada una de ellas identifica atributos lógicos agrupando etiquetas y elementos en cada interfaz para después obtener metainformación de los atributos, como su tipo o las relaciones existentes entre sus elementos.

Para modelar una interfaz de consulta, se utiliza una 3-upla, $F= (S, \{A_1, A_2, ..., A_n\}, C_f)$, donde S representa la información del sitio asociada al formulario, como el URL, el nombre del servidor y el método de comunicaciones HTTP a utilizar, $\{A_1, A_2, ..., A_n\}$ es una lista ordenada de los atributos de la interfaz y C_f representa las restricciones del formulario, es decir, las relaciones lógicas entre los atributos para la correcta ejecución de una consulta sobre el formulario.

Cada atributo A_i se representa con una 9-upla $(L, P, DT, DF, VT, U, R_e, \{E_j, E_{j+1}, ..., E_k\}, C_a)$, donde L es la etiqueta del atributo, P es su disposición visual en la página, DT es el tipo de dominio del atributo, DF es su valor por defecto, VT es el tipo del valor del atributo, U es la unidad en la que se expresa ese valor, $\{E_j, E_{j+1}, ..., E_k\}$ es una lista ordenada de elementos del dominio del atributo, R_e es el tipo de relaciones de los elementos del dominio del atributo y C_a son las restricciones del atributo. Cada elemento del dominio E_i se representa a su vez como (L_e, N, F_e, V, DV), donde L_e es una etiqueta (posiblemente vacía),

N es el nombre de E_i, F_e es el formato (textbox, checkbox, selection list, radiobutton), V es el conjunto de valores de E_i y DV es el valor por defecto de E_i (posiblemente nulo).

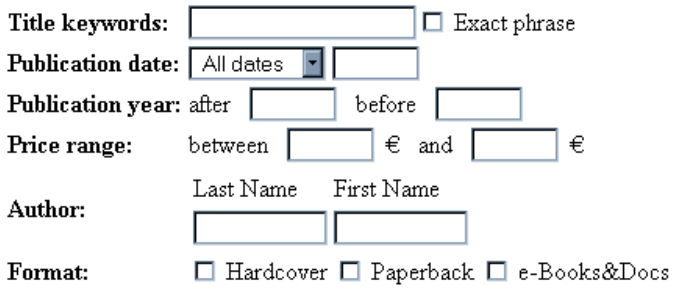


Figura 15 Ejemplo de formulario de una tienda de comercio electrónico

Un atributo puede estar formado por varios elementos, que pueden estar relacionados de 4 formas diferentes: rango, parte, grupo y restricción. Por ejemplo, en el formulario de la Figura 15, las relaciones entre los elementos de los atributos 'Publication date' y 'Price range' son de tipo rango para especificar las condiciones de consulta, la relación entre los campos 'Last name' y 'First name' es de tipo parte respecto al atributo 'Author', la relación entre los elementos del atributo 'Format' es de tipo grupo y 'Exact phrase' es una restricción del atributo 'Title keywords'.

Los atributos pueden ser de dos tipos: elementos del dominio o elementos restricción. Además, en función de la obligatoriedad de los diferentes campos para la realización de una consulta, se clasifican en conjuntivos (se necesitan todos), disyuntivos (se necesitan unos u otros), exclusivos (sólo se puede especificar uno) o híbridos (los tipos anteriores combinados sobre grupos de atributos).

Para capturar la disposición visual de las etiquetas y elementos de páginas HTML que representan interfaces de consulta, los autores introducen el concepto de expresiones de interfaz (*IEXP – Interface Expression*). Una expresión de interfaz organiza etiquetas y elementos en múltiples filas y proporciona una descripción de alto nivel de la disposición visual de las diferentes etiquetas y elementos. Se trata de una cadena formada por tres ítems básicos: t, que representa una etiqueta o texto, e un elemento del formulario y | un salto de línea. Por ejemplo la interfaz de búsqueda de la Figura 15 puede ser representada con la expresión teet|tee|ttete|ttetet|ttt|ee|tetetet.

Este sistema también es capaz de identificar atributos exclusivos, es decir aquellos que no pueden aparecer conjuntamente en el mismo formulario de búsqueda. Se trata de elementos cuyo valor proporciona el nombre del atributo, como por ejemplo una lista de selección – o un conjunto de checkboxes – que permite especificar si un área de texto obtendrá el título o el autor de un libro en una tienda de comercio electrónico.

Para identificar atributos exclusivos, los autores se basan en el descubrimiento realizado por He y Chang en [HC03], en el que se afirma que el vocabulario de los esquemas de un dominio se estabiliza rápidamente, incluso aunque el número de interfaces de búsqueda en el dominio sea muy grande. Además, los autores de WISE observaron que los nombres de atributos exclusivos suelen ser los nombres de atributo más comunes en un dominio. En

base a esto, los autores proponen una aproximación estadística para abordar el problema de identificar atributos exclusivos. La idea básica es la construcción de un vocabulario para un dominio, considerando múltiples interfaces en el mismo dominio al mismo tiempo.

Para construir el vocabulario, se calcula la frecuencia de ocurrencia de las etiquetas que aparecen en las interfaces de búsqueda consideradas para un dominio (LIF – *Label Interface Frequency*), seleccionando aquellas que superan un umbral. A continuación se localizan elementos en las interfaces de consulta que contengan en sus valores un porcentaje determinado de palabras del vocabulario – típicamente elementos input de tipo radiobutton o elementos select – y que tengan un elemento textbox próximo. En ese caso, se utiliza el valor seleccionado por el *radio button* o la lista de selección como nombre del atributo.

En base a la IEXP, las etiquetas y elementos relacionados se agrupan de modo que cada grupo representa un atributo. Este proceso se lleva a cabo utilizando la técnica de extracción basada en la disposición de las expresiones (LEX – *Layout-expression-based extraction technique*). Para cada elemento *e* en una fila, LEX encuentra la etiqueta en la misma fila o en filas adyacentes por encima de la fila del elemento (hasta 2 filas), que es la que mayor probabilidad presenta de ser la etiqueta para *e*, en base a un peso de asociación de la etiqueta con *e*. El peso de la asociación se calcula utilizando varias heurísticas, como por ejemplo que los nombres de los atributos suelen terminar en ':', la similitud textual de los textos de las etiquetas y nombres de elementos, la adyacencia de etiquetas y elementos o el alineamiento vertical.

Una vez se han extraído los atributos de una interfaz de consulta, el siguiente paso es obtener su semántica, diferenciando entre elementos del dominio del formulario y restricciones. Primero, se identifican los atributos cuyos elementos son todos de tipo radiobutton o checkbox para comprobar cuáles sólo tienen elementos del dominio. Para aquellos atributos que poseen varios elementos del dominio, se identifican sus relaciones. Se considera que la relación es de tipo 'grupo' si todos los elementos del atributo son de tipo checkbox o radiobutton y hay al menos 2; para las relaciones de tipo 'rango' se utilizan *tokens* independientes del dominio que suelen aparecen en elementos que representan rangos como 'between-and', 'from-to'. En el resto de casos, se asume la relación de tipo 'parte'.

Para obtener la semántica del resto de elementos, se utilizan diferentes heurísticas y un diccionario de sinónimos que se aplica a las etiquetas y valores de los elementos de un atributo. También se utiliza información independiente del dominio, como patrones comunes para fechas y horas. Por ejemplo, si se encuentran tres selectores, cada uno con los valores adecuados para mes/día/año, se considera que se trata de una fecha. De esta forma, se permite asociar de forma automática metainformación a los diferentes atributos como el tipo de dominio, que indica cuántos valores se pueden especificar en un atributo (range, finite, infinite, boolean), el tipo de valor – se consideran los tipos date, time, datetime, currency, id, number, char y se permite definir nuevos tipos en base a expresiones

regulares –, el valor por defecto (los marcados como checked o selected en el formulario), y sus unidades – se utiliza un diccionario de unidades[18] –.

Para detectar la obligatoriedad de los diferentes elementos de la interfaz, se realizan diferentes tipos de consultas para decidir en base a los resultados observados. La obtención de las consultas adecuadas para obtener resultados válidos puede ser complicada pero en algunas interfaces aparecen operadores de forma explícita y en las que no es así, se asume por defecto que todos los atributos son obligatorios.

En comparación con trabajos existentes, el extractor de interfaces WISE-iExtractor puede obtener más semántica para los diferentes elementos de las interfaces de consulta web, en base a la metainformación extraída para sus elementos. Está orientado a extraer atributos, que se definen como colecciones de elementos de la interfaz de consulta web. Además, permite extraer atributos exclusivos, obteniendo en sus experimentos muy buenos resultados – un porcentaje de éxito de 95% para la extracción de atributos de forma correcta, en relación al total de atributos que deberían de haber sido extraídos –.

Por otro lado, su aproximación totalmente heurística tanto para la extracción de atributos como de semántica, es su mayor debilidad. Además, a la hora de obtener la expresión que representa la disposición visual de los elementos no utilizan ningún motor de visualización, por lo que la expresión obtenida puede no representar fielmente el alineamiento de etiquetas y elementos presentes en la interfaz, con los errores que esto puede ocasionar al proceso completo.

II.3.3.4.4. SISTEMA DE ADAPTACIÓN DE FORMULARIOS WEB A DISPOSITIVOS DE PEQUEÑO TAMAÑO

Kaljuvee et al. presentan en [KBGP01] un sistema capaz de adaptar páginas web con formularios para que puedan ser visualizadas en dispositivos portátiles con pantallas de reducido tamaño, como PDAs (*Personal Digital Assistants*). Para cada formulario, se crea un modelo que lo represente, asociando etiquetas a cada uno de sus campos. A partir del modelo, el formulario se visualizará en los dispositivos manteniendo su semántica, pero en formato más reducido.

Para realizar el modelado del formulario, se divide la página en fragmentos teniendo en cuenta determinadas etiquetas HTML (<p>,
, <table>, etiquetas de formularios, etc.). Para cada uno de los fragmentos, se asocia el mejor texto utilizando heurísticas basadas en similitud textual.

La similitud textual se realiza con el nombre interno del campo del formulario en el código HTML, aplicando el algoritmo *n-gram*. Antes de realizar la comparación de los textos, se utilizan diferentes algoritmos para obtener un formato normalizado de los mismos. Por ejemplo, pueden utilizarse los algoritmos *Letter/Word* o *Word/Letter*, que consisten en obtener las dos primeras palabras de cada frase y concatenar el primer carácter de la primera palabra con la segunda palabra (*Letter/Word*; al revés para *Word/Letter*). Los

[18] http://www.unc.edu/~rowlett/units/

autores han observado que muchos formularios utilizan este criterio para generar los nombres de sus elementos. Otras veces funciona mejor la normalización *substring*, para permitir detectar similitudes entre textos y sus abreviaturas (detectar por ejemplo que 'Password' es un texto representativo para un elemento de nombre 'pwd').

En el caso de que el formulario utilice tablas para componer sus elementos, la tabla crea una disposición visual de los elementos determinada, que el usuario humano es capaz de comprender para asociar semántica a los diferentes campos. En ese caso, el sistema intenta aplicar las heurísticas que utiliza un usuario humano para realizar las asociaciones de textos. De esta forma, si un elemento de un formulario se encuentra en la misma celda que un texto, entonces se le asigna ese texto; si no es así, entonces se le asigna el primer texto que aparezca, buscando en las celdas contiguas, en el siguiente orden: izquierda, encima, debajo, derecha.

Cuando tras aplicar las heurísticas anteriores no se ha logrado asociar un texto representativo a un fragmento, se considera el texto anterior o siguiente a una distancia menor que 10 fragmentos, o en último caso el nombre del campo del formulario directamente.

Los checkboxes reciben un tratamiento especial, utilizando el nombre del primero para identificar el texto global del grupo, y el valor de cada uno de ellos para asociar sus textos correspondientes, aplicando las heurísticas de similitud.

Los autores han realizado pruebas con diferentes sitios web, combinando las diferentes heurísticas en un orden determinado, obteniendo un 80% de acierto en la identificación correcta de elementos de formularios.

De todas formas, este sistema es muy sensible a cómo se han generado los nombres internos de los elementos del formulario que, a menudo, no proporcionan realmente ninguna información sobre la semántica del campo. Los mismos autores identifican los siguientes escenarios adicionales de fallo típicos: divisiones en fragmentos erróneos, textos encapsulados en imágenes, *checkboxes* que no posean textos de grupo, elementos que no poseen textos como selectores de mes, día y año consecutivos, o etiquetas que agrupen a campos de texto o selectores.

II.3.3.5. ASOCIACIONES FORMULARIO-DOMINIO DE APLICACIÓN

Una vez que un formulario ha sido adecuadamente modelado, los siguientes pasos para un *crawler* dirigido de la Web Oculta son: 1) Determinar si el formulario es relevante para su tarea o *dominio de aplicación* objetivo, y 2) Si el formulario es relevante, rellenarlo asociando un valor de consulta a cada campo, de acuerdo a su semántica para ejecutar la consulta o consultas deseadas.

Para que este proceso pueda realizarse de forma automática por un sistema software, la especificación del *dominio de aplicación* para el *crawling* dirigido debe incluir cierta metainformación sobre los elementos que forman parte de los formularios de consulta para la tarea objetivo.

Por ejemplo, una especificación de dominio de aplicación orientada a localizar formularios de búsqueda de libros, puede indicar que los formularios objetivo frecuentemente incluyen campos de búsqueda para los atributos TÍTULO y AUTOR, y que dichos campos suelen ser identificados, respectivamente, por etiquetas textuales como 'Título' o 'Title' y 'Autor' o 'Author'. Esta metainformación es útil tanto para reconocer el formulario como relevante, como para aprender a rellenar adecuadamente el formulario para ejecutar una consulta determinada: por ejemplo, si el dominio de aplicación incluye entre sus tareas encontrar datos de libros sobre el lenguaje de programación Java, el *crawler* podría ejecutar una consulta que introdujese la palabra 'java' en el campo del formulario que tenga asociada una de las etiquetas especificadas por el dominio de aplicación para el campo TÍTULO.

A continuación se presentan los principales sistemas que se ocupan del problema de asociación de campos del formulario con atributos del dominio de aplicación que guía el proceso de *crawling* y del problema complementario de descartar formularios irrelevantes. En el apartado II.3.3.5.1 se describe la aproximación basada en clasificadores bayesianos presentada por Kushmerick en [Kushmerick03]. En el apartado II.3.3.5.2 se comenta en detalle la aproximación utilizada por el sistema de *crawling* de la Web Oculta HiWE.

II.3.3.5.1. CLASIFICACIÓN BAYESIANA DE FORMULARIOS

Kushmerick [Kushmerick03] utiliza una aproximación basada en clasificadores Bayesianos para clasificar formularios de consulta web en un dominio específico y predecir qué atributo del dominio de aplicación debe asociarse a cada elemento del formulario. En lugar de caracterizar cada campo del formulario por separado, se consideran todos los campos de un formulario de forma conjunta, y se intenta maximizar las probabilidades de todas las predicciones de forma global.

El objetivo es aprender la semántica de un formulario utilizando sólo información accesible desde el lado cliente, sin invocar el formulario para evitar sobrecargarlo o realizar acciones sobre él indeseadas. Por ejemplo la ejecución de un formulario podría tener como consecuencia la inserción de tuplas en una base de datos subyacente, con lo que se estaría modificando su contenido.

Para formalizar el problema de clasificación de formularios desde la aproximación de aprendizaje supervisado, se asume un conjunto de instancias previamente etiquetadas que forman el conjunto de datos de entrenamiento. Las instancias se corresponden con formularios web y las etiquetas son metadatos de una taxonomía asociados a los formularios y a sus campos. Un formulario está compuesto de uno o más campos, y cada campo a su vez está compuesto de uno o más términos. Más concretamente, un formulario F_i es una secuencia de campos $F_i=\{c_i^1, c_i^2, \ldots\}$, y cada campo c_i^j es un conjunto de términos, $c_i^j=\{t_i^j(1), t_i^j(2), \ldots\}$. Los términos representan palabras, etiquetas, atributos u otros *tokens* en documentos HTML.

El objetivo del algoritmo propuesto consiste en clasificar un formulario y sus campos de acuerdo a alguna taxonomía preexistente. Se asumen dos taxonomías para asociar metadatos semánticos a formularios y campos. Primero, se asume una taxonomía para dominios de aplicación, *D*. Los dominios capturan el objetivo general de un formulario, tal

como ‘búsqueda de libros’, ‘búsqueda de trabajos’, ‘consulta de horarios de vuelos’, etc. Asumen otra taxonomía, *T*, para tipos de datos. Los tipos de datos no representan aspectos de bajo nivel, como ‘texto’ o ‘entero’, sino la categoría semántica del atributo, como ‘título de libro’, ‘sueldo’, ‘aeropuerto destino’, etc.

El problema de aprendizaje se formula como sigue: La entrada es un conjunto de formularios y campos etiquetados, es decir, un conjunto $\{F_1, F_2, \ldots\}$ de formularios, junto con un dominio D_i perteneciente a D para cada formulario F_i, y un tipo de dato T_{ij} perteneciente a T para cada campo c_{ij} de F_i. La salida es un clasificador de formularios, es decir, una función que asocia un formulario no etiquetado a un dominio y un tipo de dato para cada campo. La estrategia presentada consiste en utilizar una aproximación estándar basada en el algoritmo de clasificación Naïve Bayes para determinar el tipo de cada campo en base a sus términos, a la vez que se calcula el dominio de un formulario considerando de forma conjunta los tipos asociados a sus campos.

El autor asume el siguiente modelo de generación de formularios web (en tres pasos)

- Selección del dominio
- Selección de tipos de datos para los campos del formulario.
- Selección de términos para los campos del formulario.

Esta aproximación presenta algunas limitaciones. Cada uno de los distintos elementos se obtiene de distribuciones, que se asumen independientes. Se asume también que los tipos de datos y los términos son independientes dados sus padres, y que todos los términos están asociados con campos (cuando en realidad existen algunos términos que describen genéricamente el formulario, sin estar asociados a ningún campo). El modelo también ignora el número de campos en un formulario y el número de términos de un campo.

II.3.3.5.2. HIWE

Este apartado describe el componente de HiWE que aprende a rellenar automáticamente los campos de un formulario para ejecutar una consulta deseada. Para los elementos de un formulario con un rango de valores finito (e.g. listas de selección o *checkboxes*), el conjunto de posibles valores que se le pueden asignar al elemento puede ser enumerado exhaustivamente. En cambio, para los elementos con un rango de valores infinito (e.g. campos de texto), es necesario utilizar información adicional para saber cómo rellenarlos.

HiWE utiliza una tabla de etiquetas-valores, LVS (*Label Value Set*) para definir la tarea de *crawling* a realizar. Es un concepto que juega un papel similar a la definición de dominio de aplicación a la que nos hemos referido al principio de la sección. Está compuesta por entradas que representan una etiqueta y un conjunto difuso de valores asociados. Cada conjunto de valores tiene una función asociada que asigna un peso entre 0 y 1 a cada miembro del conjunto. Cada valor del conjunto representa un valor que podría ser asignado a un elemento de un formulario si la etiqueta del elemento concuerda con la etiqueta de la entrada de la tabla correspondiente, y con un nivel de confianza expresado por el peso asignado a ese valor.

Para elementos con dominios infinitos, HiWE intenta realizar emparejamientos entre las etiquetas de esos elementos y las etiquetas de la tabla LVS. Para ello, primero se normalizan todas las etiquetas de la tabla a un formato común y un estilo estándar, aplicando la misma normalización que la utilizada en el apartado II.3.3.4.1 para las etiquetas asociadas a los campos del formulario. A continuación se utiliza un algoritmo de alineamiento de cadenas para calcular la distancia de edición mínima, teniendo en cuenta no solamente diferencias de caracteres, sino también reordenaciones de palabras.

Dado un elemento del formulario, el algoritmo utilizado devuelve la entrada de la tabla cuya etiqueta tiene la distancia de edición mínima con respecto a la etiqueta de ese elemento, siempre que sea inferior a un umbral. En el apartado II.3.3.6.1 se comenta en detalle cómo se obtienen los valores asociados a las etiquetas presentes en la tabla *LVS*.

Esta aproximación presenta las siguientes debilidades:

- Requiere que la tabla LVS contenga una definición de atributo que se asocie con cada campo del formulario. Es decir, considera asociaciones totales entre los campos de un formulario y los atributos especificados por el dominio de aplicación, descartando aquellos formularios que posean sólo parte de los campos.
- Limita los formularios tratados en función de un parámetro configurable que indica el número mínimo de campos que debe presentar un formulario.
- Además, selecciona para cada campo aquel atributo con mayor similitud (si supera un umbral). No parece haber ninguna comprobación adicional, con lo que podría suceder, por ejemplo, que se asignasen dos campos al mismo atributo.

II.3.3.6. EJECUCIÓN DE CONSULTAS SOBRE EL FORMULARIO

Una vez que un sistema de *crawling* dirigido ha determinado que un formulario es relevante para la tarea objetivo y ha aprendido como rellenarlo, el siguiente paso es ejecutar consultas sobre él. Una problemática añadida reside en determinar si las páginas obtenidas contienen los datos esperados o si por el contrario se ha producido algún tipo de error (e.g. página no encontrada, no se han encontrado resultados para la búsqueda, no se ha rellenado algún campo obligatorio, el valor de un campo es incorrecto, etc).

Existen diferentes aproximaciones para determinar qué consultas deben ejecutarse sobre un formulario. Una posible aproximación consiste en que la especificación del dominio de aplicación indique explícitamente una serie de consultas predefinidas en términos de los atributos del dominio. Otras aproximaciones consideran el problema de obtener todo el contenido almacenado tras un formulario de consulta web de forma independiente a un dominio concreto. Típicamente, los sistemas que utilizan esta última aproximación no siguen un enfoque dirigido sino de *crawling* global. Sin embargo, siempre que se haya establecido previamente por otros medios la relevancia del formulario para el dominio de aplicación, podrían utilizarse en el contexto de *crawling* dirigido, y por ello se describen también en esta sección.

En el apartado II.3.3.6.1 se describe el módulo de ejecución de consultas del sistema de *crawling* de la Web Oculta HiWE, que se basa en la utilización de una especificación de dominio de aplicación que contiene un conjunto de valores posibles para cada atributo. En el apartado II.3.3.6.2 se describen las principales técnicas propuestas para obtener todo el contenido almacenado tras un formulario de consulta web.

La idea de realizar consultas de forma automática contra bases de datos en la Web y examinar sus resultados ha sido aplicada anteriormente a otros contextos. Por ejemplo, para estimar el porcentaje de la Web que tratan diferentes motores de búsqueda [LG98] o para estimar la fracción de una base de datos textual que puede ser recuperada realizando consultas [AG03b] [AIG03] [WWLM06]. También se ha utilizado para identificar las bases de datos más relevantes dada una consulta de usuario [IGS01] [IG02] [CS96] [LRLC04] [HC03] [HYJS06]. En este último grupo se encuentran los sistemas centrados en la generación de resúmenes de bases de datos *online*, para permitir caracterizarlas, y de este modo ayudar en los procesos de selección de sistemas de metabúsqueda.

II.3.3.6.1. HIWE

Este apartado describe el componente de HiWE que realiza consultas sobre un formulario y analiza las páginas de respuesta. Utiliza bases de datos específicas a cada tarea de *crawling* para la realización de consultas de forma automática contra formularios de fuentes web.

Una vez se ha modelado el formulario y se conoce la semántica de cada campo, la tabla LVS introducida en el apartado II.3.3.5.2 permite obtener, para los campos del formulario con rango de valores posibles no acotado, el conjunto de valores para asignar al campo del formulario. Cada uno de esos valores tiene asignado un peso o nivel de confianza entre `0` y `1`. En el caso de campos con rango de valores acotado (e.g. listas de selección o *checkboxes*), el peso de cada posible valor es siempre `1`.

Una vez obtenidas las posibles asignaciones para cada campo individual, HiWE calcula todas las posibles asignaciones para el formulario haciendo las combinaciones, y las ordena utilizando una función de agregación que calcula un peso para la asignación global, en función del peso del nivel de confianza de cada asignación individual. Se limita el número de consultas a realizar sobre cada formulario y se utiliza ese nivel de confianza para determinar cuáles serán las consultas a realizar.

Los autores experimentaron con tres tipos diferentes de funciones: conjunción difusa (utiliza el nivel de confianza mínimo), media (calcula la media) y probabilística (trata los pesos como probabilidades y calcula la probabilidad de que la asignación sea útil). La más efectiva resulta ser la media. Los resultados de los experimentos pueden consultarse en [RG01].

Para determinar si una consulta ha sido exitosa o no, utiliza su motor de visualización para encontrar la zona central de la página de respuesta y la analiza de acuerdo a dos criterios:

- Busca mensajes típicos de error o de no hay resultados.

- Calcula un *hash* de la misma. Si se repite con frecuencia, considera que ese *hash* identifica la página de que no hay resultados.

Para rellenar la tabla LVS de asignación de valores a etiquetas, se utilizan varios mecanismos:

- Inicialización explícita. La proporcionada por el administrador inicialmente.
- Entradas precargadas. Contiene algunas categorías de uso común en diferentes aplicaciones, tales como, fechas, nombres de meses, días de la semana, etc.
- Fuentes de datos. Es posible configurar el sistema para que obtenga valores de consulta de diferentes fuentes de datos que pueden ser específicas de la aplicación o porciones de directorios genéricos disponibles a través de Internet. Sobre cada fuente de datos a utilizar debe construirse un programa envoltorio que permita realizar consultas para obtener conjuntos difusos de valores a partir de etiquetas, o a partir de otros conjuntos de valores que pertenezcan al mismo conjunto. Los autores han realizado experimentos con el directorio de Yahoo! para este propósito.
- Experiencia del *crawler*. Los elementos de formularios con rango de valores acotados son una fuente útil de etiquetas y conjunto de valores asociados. Esta información es particularmente útil cuando la misma etiqueta o una similar está asociada en un formulario con un elemento que tiene un rango acotado y en otro formulario con un elemento con rango no acotado (e.g. un campo de texto).

Para calcular los pesos asociados a los valores de la tabla LVS se usan las siguientes reglas:

- A los valores obtenidos a través de la inicialización explícita se les asigna peso fijo 1.
- El peso inicial de los valores procedentes de fuentes de datos externas debe ser proporcionado por el programa envoltorio.
- La confianza de cada valor usado en la búsqueda para un atributo se incrementa cada vez que es utilizada en una consulta exitosa y se reduce con el tiempo.

II.3.3.6.2. TÉCNICAS DE EXTRACCIÓN DE TODO EL CONTENIDO DETRÁS DE UN FORMULARIO

En este apartado se describen las principales técnicas propuestas para abordar el problema de obtener todo el contenido almacenado tras un formulario de consulta web de forma independiente a un dominio concreto.

Un primer sistema que aborda este problema es el propuesto por Liddle et al. [LYE01] [LESY02]. El sistema comienza realizando la consulta por defecto, es decir, el envío directo del formulario sin rellenar ningún campo y se analizan los resultados obtenidos. Los autores afirman que en bastantes casos la consulta por defecto permite obtener todos los resultados de la fuente. Para comprobar si se han obtenido todos los datos, se emiten varias consultas y se comprueba si se obtienen nuevos datos. Para que este método sea fiable, se aplica una técnica de muestreo estratificado para garantizar que las consultas se distribuyen

entre todos los atributos del formulario, para evitar que no se obtengan nuevos resultados debido a que las consultas han afectado al mismo subconjunto del espacio de búsqueda.

Si no se han obtenido todos los resultados, entonces se consulta de forma exhaustiva el formulario (calculando combinaciones de valores entre sus elementos, intentando distribuirlas al máximo en el espacio de búsqueda) hasta llegar a un determinado umbral de cantidad de datos extraídos, consultas ejecutadas, tiempo transcurrido o porcentaje de datos estimado que se ha extraído (este porcentaje se estima comparando las combinaciones ejecutadas frente a las combinaciones posibles). No se consideran campos de texto, sino que sólo se consideran campos con una lista enumerada de valores.

Tras el envío del formulario, se analiza la página de respuesta para determinar si contiene todos los datos, enlaces de intervalo siguiente, otros formularios o si se trata de una página de error. Para detectar enlaces en intervalo siguiente, aplican sencillas heurísticas buscando textos 'next' o 'more'.

Este sistema presenta las siguientes limitaciones:

- Sólo trata con formularios que no tienen campos de texto obligatorios. El formulario puede tener campos de texto, pero asumen que un envío del formulario con los campos textuales vacíos es equivalente a consultar dichos campos por todos los valores posibles. En el caso de que esto no sea cierto, permiten que el usuario los rellene de forma manual.
- No se tratan formularios en varios pasos, como puede ser el caso de la autenticación previa para tener acceso a algún formulario de búsqueda, o formularios que aparecen en páginas de resultados de búsqueda, para realizar diferentes filtrados de los datos.
- Si en una página existen varios formularios, sólo analizan el primero, sin realizar ningún análisis previo para seleccionar el más adecuado según su modo de funcionamiento.

Ntoulas et al. [NZC05] también abordan el problema de generación automática de consultas para obtener todo el contenido que haya tras un formulario web. Los autores proponen nuevas técnicas para generar de forma automática nuevas palabras de búsqueda a partir de resultados obtenidos por consultas previas y para priorizarlas con el objetivo de recuperar todo el contenido oculto tras el formulario, utilizando para ello el mínimo número de consultas. Intentan predecir el número de resultados obtenidos por cada término considerado, en función de su frecuencia de aparición en las páginas de resultados que va obteniendo. Los términos utilizados para las primeras consultas se obtienen en base a su frecuencia en una colección de documentos genérica.

Sin embargo, las técnicas presentadas sólo consideran formularios de búsqueda sobre lo que denominan bases de datos textuales, que normalmente presentan un único campo textual para permitir búsquedas por palabra clave. Contemplan como trabajos futuros la posibilidad de extender sus técnicas para que fuesen aplicables a interfaces de consulta web sobre bases de datos estructuradas. Tampoco consideran las páginas de resultado de intervalo siguiente.

II.3.3.7. ESTRUCTURACIÓN DE PÁGINAS DE RESULTADOS

Como ya se ha comentado en la introducción (apartado I.1.2), la información contenida en la Web de Superficie ha sido habitualmente clasificada dentro de la categoría de información *no estructurada* (junto a, por ejemplo, la información documental).

Los procesamientos que una aplicación software puede realizar con información no estructurada son muy limitados. Por el contrario, la información estructurada contenida en una típica base de datos relacional puede ser manipulada por un programa de ordenador con toda la complejidad que se desee, ya que esta información se ajusta a un esquema perfectamente estructurado y definido.

Sin embargo, a menudo las interfaces de consulta web no son más que frontales de acceso a bases de datos. Tal y como ya se ha comentado, Chang et al. [CHLP+04] estiman que el 77% de las fuentes de la Web Oculta ofrecen formularios que permiten ejecutar consultas sobre una base de datos subyacente. En general, el resultado de una consulta sobre uno de estos frontales está formado por una lista enumerada de tuplas, expresadas de forma semi-estructurada en HTML, conservando una estructura latente en su disposición visual.

Por lo tanto, otro de los retos a los que se enfrentan los sistemas modernos para el tratamiento de la información contenida en la Web Oculta es inferir la estructura subyacente presente en este tipo de páginas. Como ya se ha comentado en el apartado I.1.2, hasta la fecha se han propuesto dos enfoques principales que pueden ser empleados para este propósito: la *Web Semántica* [W3CSW] y el uso de *programas de extracción de datos web*. Ambos tienen importantes inconvenientes en el contexto de las aplicaciones de *crawling* dirigido, como se expone a continuación.

La Web Semántica define un conjunto de normas y protocolos para la construcción de fuentes web, de manera que el acceso a su información sea posible de forma sencilla para las aplicaciones software. De especial interés dentro de este ámbito es la interfaz que proporcionan los Servicios Web [W3CWS].

En una Web Semántica, las interfaces de consulta proporcionarían información concreta sobre los requisitos del usuario, y las páginas de respuesta presentarían resultados totalmente etiquetados, de modo que sería posible para un programa de ordenador, procesar de forma estructurada la información proporcionada. Además, la semántica asociada a la consulta ayudaría al motor de búsqueda en la localización de los resultados más adecuados.

Sin embargo, esta iniciativa no está teniendo todo el éxito que se esperaba, debido al importante coste que supone la etiquetación manual del contenido de las fuentes web, el desconocimiento por parte de las instituciones y personas que lanzan nuevas aplicaciones y páginas web y el conjunto ingente de información que ya existe en la red en aplicaciones que no van a actualizarse a corto/medio plazo. Queda mucho trabajo por hacer hasta que la Web Semántica adquiera la masa crítica imprescindible para hacerse realidad.

En cuanto a los programas de extracción de datos web, son muchos los trabajos existentes que abordan esta problemática, y pueden clasificarse de la siguiente forma, en

función de su modo de funcionamiento (Laender et al. [LRST02] realizan un análisis de este tipo de herramientas, que se presentan bajo una división parecida):

- Enfoque manual basado en lenguajes de definición de programas envoltorios. En esta aproximación se proporciona algún lenguaje de especificaciones basado en patrones para ayudar al usuario a crear programas de extracción. Son muchos los trabajos existentes que utilizan esta aproximación. Entre ellos se encuentran Minerva [CM98], TSIMMIS [HMG97], WebL [KM98], Web-OQL [AM98], FLORID [LHLM+98], Jedi [HFAN98], QEL/CESL [GRV98] o WIDL [Allen97]. Algunos disponen de herramientas gráficas para generación supervisada de los programas extractores, como W4F [SA01], XWRAP [LPH00], Lixto [BFG01] o Wargo [PRAH+02].
- Herramientas de aprendizaje inductivo. Estas técnicas utilizan aprendizaje supervisado para aprender las reglas de extracción de datos a partir de un conjunto de ejemplos etiquetados de forma manual. La etiquetación manual es un proceso bastante costoso en tiempo y no escalable. La razón de esto es que para distintos sitios web o incluso diferentes páginas en un mismo sitio web, puede ser necesario un nuevo proceso de etiquetación por tratarse de páginas que siguen diferentes plantillas. Ejemplos de sistemas que utilizan técnicas de inducción son WIEN [KWD97], SoftMealy [HD98] o STALKER [MMK01] [MMK03].
- Herramientas basadas en procesamiento de lenguaje natural. Son similares a las basadas en lenguajes de definición de programas envoltorios, pero en lugar de basarse en la estructura de representación de los elementos en las páginas, se basan en reglas sintácticas de lenguaje natural. Algunas herramientas representativas de este enfoque son RAPIER [CM99], SVR [Freitag00] y WHISK [Soderland99].
- Herramientas basadas en modelo de datos. Esta categoría incluye herramientas que, dada una estructura de los objetos de interés, tratan de localizar en las páginas web porciones de datos cuya representación gráfica es implícitamente conforme a tal estructura. La estructura es proporcionada de acuerdo a un conjunto de primitivas de modelado gráfico (por ejemplo tuplas, listas, etc.) que son conformes a un modelo de datos subyacente. A continuación, algoritmos similares a los utilizados por las herramientas de aprendizaje inductivo, identifican objetos con la estructura proporcionada en las páginas utilizadas. Algunas herramientas que adoptan esta aproximación son NoDoSE [Adelberg98] y DEByE [LRS02][RLS02].
- Herramientas de estructuración automática. Este conjunto de técnicas se basan en una serie de heurísticas generales de representación de datos en una página web y no requieren supervisión manual para generar un programa que extraiga datos de una determinada página web. Algunos de sus ejemplos más representativos son IEPAD [CL01], RoadRunner [CMM01], EXALG [AG03a] y DEPTA [ZL06].

Todos los tipos de técnicas mencionadas excepto las últimas requieren que un administrador cree *a priori* un programa extractor para cada fuente de la que se desee extraer datos. En las aplicaciones de *crawling* dirigido, las fuentes de información no se conocen previamente, sino que son descubiertas dinámicamente al explorar la Web. Por lo tanto, no es posible desarrollar *a priori* los programas extractores necesarios para tratarlas.

Por este motivo, el componente de estructuración de un *crawler* debe estar basado en técnicas de estructuración automática. Este tipo de herramientas pueden ser menos efectivas que las que utilizan ejemplos u otro tipo de intervención del usuario, pero suelen ser suficientes para una aplicación de *crawling* estructurado.

Las herramientas de estructuración automática se basan en que muchos sitios web contienen grandes conjuntos de páginas que son generadas utilizando una misma plantilla HTML, donde las partes variables de dicha plantilla representan los valores de las tuplas a ser extraídas, que normalmente proceden de una base de datos. Una vez deducida la plantilla, es posible utilizarla para extraer de cada página las partes variables que conforman las tuplas de resultados. En la Figura 16 se muestra el modelo de generación de páginas web que asumen los sistemas de estructuración automática.

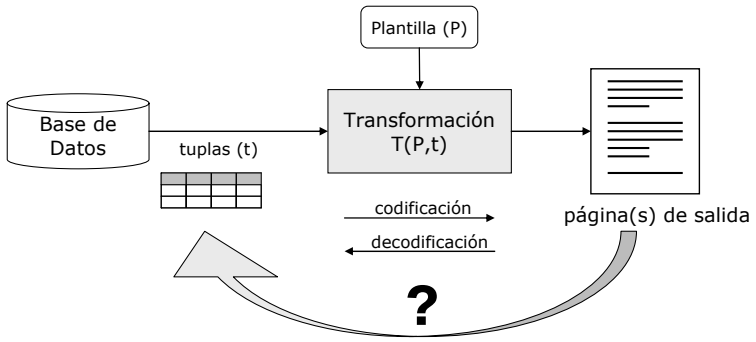


Figura 16 Modelo de generación de páginas

Un primer grupo de técnicas de este tipo (RoadRunner [CMM01], EXALG [AG03a], WebIQ [WDC06] o DeLa [WL03]) toman como entrada un conjunto de páginas de la misma clase (generadas a partir de la misma plantilla) para automáticamente inducir la plantilla a partir de la cual se han generado y el esquema de los datos codificados en ella. Sin embargo, la disponibilidad de múltiples páginas que contengan registros de datos similares constituye una limitación, ya que en ninguno de estos sistemas se aborda cómo obtener automáticamente dichas páginas y, además, es necesario recopilar múltiples páginas incluso si se desea extraer datos sólo de una de ellas. Los apartados II.3.3.7.2 y II.3.3.7.3 describen con más detalle los dos sistemas de este tipo más significativos.

Un segundo grupo de técnicas son capaces de inducir la plantilla aún tomando como entrada una única página que contenga varias tuplas de los datos a extraer, como IEPAD [CL01] o DEPTA [ZL06]. En este caso, para determinar cuales son las partes fijas de la plantilla y cuales son las partes variables correspondientes a campos de datos, se comparan entre sí los registros contenidos en una página, en lugar de utilizar comparaciones entre páginas. Ambos sistemas son descritos con más detalle en los apartados II.3.3.7.1 y II.3.3.7.4, respectivamente.

Lerman et al. proponen en [LGMK04] otro método cuya idea principal es utilizar información redundante en una página que presenta un conjunto de registros y páginas de detalle (cada una de esas páginas conteniendo información con detalles adicionales sobre el registro de datos correspondiente en la lista de páginas) para ayudar a la extracción de información. Esta técnica detecta aquellos datos que aparecen en la página principal y en la

de detalle y utiliza esta información para trocear adecuadamente la página de lista de resultados en fragmentos que representan cada uno de ellos un resultado de la consulta, y para identificar los valores para sus atributos. Este enfoque tiene la seria limitación de que en muchas ocasiones las tuplas presentes en este tipo páginas no tienen asociadas páginas de detalle. Además, asume que las páginas de detalle son proporcionadas como entrada al sistema (en sus experimentos son recolectadas manualmente), lo cual no sería sencillo para un sistema de *crawling* automático. La tarea de identificar automáticamente los enlaces que apuntan a una página de detalle, antes de tener identificados los registros de la página de partida, no es una tarea fácil de resolver.

Otro de los problemas a los que se enfrentan este tipo de sistemas es que no anotan los resultados extraídos, es decir, no proporcionan nombres significativos para los campos de las tuplas extraídas, ya que estos nombres normalmente no se encuentran codificados en las páginas. Arlota et al. [ACMM03] presentan LABELLER, una aproximación para anotar automáticamente los datos extraídos utilizando los textos que rodean a los fragmentos obtenidos de la página. Este enfoque no es válido en las situaciones en las que algunos de los campos de datos a extraer no tienen etiquetas asociadas en la página. Wang y Lochovsky presentan en [WL03] otra aproximación más compleja, basada en heurísticas, para asignar etiquetas a los datos extraídos. Las ideas básicas de las heurísticas utilizadas son las siguientes:

- Emparejar etiquetas de los elementos de los formularios de búsqueda con los atributos extraídos. Los valores de búsqueda enviados a través de un elemento de un formulario, normalmente aparecerán entre los valores del atributo correspondiente en la página de resultados. De esta manera es posible asociar un atributo con un elemento de un formulario y asignar a ese atributo la etiqueta asociada al elemento del formulario en la página de búsqueda.
- Buscar etiquetas en las cabeceras de las tablas. Las etiquetas presentes en las cabeceras de las tablas HTML pueden ser utilizadas como etiquetas para los valores extraídos de la columna correspondiente de la tabla.
- Buscar prefijos y sufijos comunes a los datos extraídos. Si todos los valores extraídos para un atributo concreto comparten un prefijo o sufijo común, normalmente, no forma parte del valor pero puede ser una etiqueta válida para el atributo.
- Identificar atributos con formatos convencionales. Si todos los valores extraídos para un atributo siguen un formato representativo de cierto tipo de datos, entonces puede asignárseles una etiqueta basada en ese tipo de datos. Por ejemplo las fechas normalmente se muestran como ‘dd-mm-yy’, ‘dd/mm/yy’, etc., las direcciones de correo contienen el símbolo ‘@’, los precios contienen los símbolos ‘$’,’€’, etc.

A pesar de la mayor complejidad de esta aproximación, tal y cómo reconocen los autores, existen ocasiones en las que la asignación de etiquetas realizada por el sistema es incompleta o errónea, ya que son bastante comunes las situaciones en las que ninguna de estas heurísticas es aplicable o, incluso, su aplicación lleva a resultados indeseados. En estos casos el usuario es el encargado de completar o corregir tales asignaciones.

A continuación se comentan detalladamente algunos de los sistemas de estructuración automática más relevantes. Se ocupan sólo del proceso de obtención de las tuplas de las páginas. Ninguno de ellos trata el problema de obtención de las colecciones de páginas ni la anotación de los datos, una vez han sido estructurados.

II.3.3.7.1. IEPAD

El sistema IEPAD [CL01] se basa en que las páginas web generadas por los motores de búsqueda generalmente presentan los resultados en forma de patrones regulares y repetitivos. Utiliza técnicas de búsqueda de patrones repetitivos y de alineamiento de cadenas de caracteres para identificar los registros de información contenidos en una página HTML.

El sistema realiza el descubrimiento de patrones repetitivos mediante una estructura de datos llamada árboles PAT [Morrison68]. Adicionalmente, los patrones generados son extendidos mediante técnicas de alineamiento múltiple de secuencias [Notredame02] [WBH05] [GBS92], de manera que el resultado reconozca registros con ligeras variaciones sobre los patrones repetitivos descubiertos.

El módulo generador de reglas de extracción es el encargado de buscar los patrones repetitivos. En la Figura 17 se muestran los diferentes componentes de los que está formado.

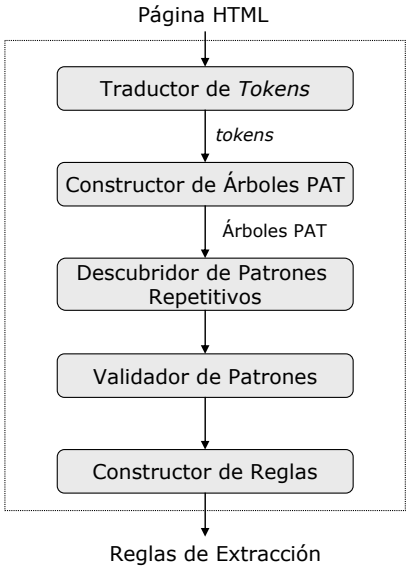


Figura 17 Generador de reglas de extracción

El *Traductor* recibe una página HTML y la traduce a una cadena de representaciones abstractas llamadas *tokens*. Cada *token* se representa con un código binario de longitud fija. Para cada marca HTML se crea un *token* diferente y los textos visibles (los que van entre dos marcas) se codifican todos con el mismo *token*. Los *tokens* correspondientes a marcas pueden clasificarse de diversas formas en función del nivel de detalle requerido en la

extracción. Por ejemplo las marcas del cuerpo de un documento pueden ser divididas en dos grupos: marcas de nivel de bloque (definen la estructura del documento) y marcas de nivel de texto (definen las características de formato de los contenidos de texto), de manera que si se ignoran todas las marcas de nivel de texto la abstracción será mayor y, si se consideran, el nivel de detalle será mayor.

El *Constructor de Árboles PAT* recibe la cadena binaria generada por el *Traductor* y construye un árbol PAT. Un árbol PAT es un árbol Patricia (*Practical Algorithm to Retrieve Information Coded in Alphanumeric*) [Morrison68] construido sobre todos los posibles sufijos de la cadena [GBS92].

El *Descubridor de patrones repetitivos* usa el árbol PAT para buscar patrones repetitivos. Por patrón repetitivo puede entenderse cualquier subcadena que aparece al menos dos veces en la cadena codificada. El número de patrones repetitivos puede ser muy elevado, por lo que se introduce el concepto de *repetición máxima*. Las repeticiones máximas son aquellos patrones repetitivos que no son una subcadena de ningún otro patrón repetitivo.

El *Validador* filtra de entre las repeticiones máximas los patrones no deseados para producir patrones de extracción candidatos. Además de la frecuencia de ocurrencias y de la longitud de los patrones, el *Validador* permite la utilización de una serie de criterios para determinar si un patrón candidato es válido, que incluyen:

- Regularidad. Se mide calculando la desviación estándar del intervalo de separación entre cada dos ocurrencias adyacentes de la repetición máxima. Favorece a aquellos patrones cuyas ocurrencias están espaciadas entre sí de forma uniforme.
- Compactación. Es una medida de la densidad de la repetición máxima. Favorece a aquellos patrones cuyas ocurrencias se encuentran concentradas en la misma región de la página.
- Cobertura. Mide el volumen de contenido de las ocurrencias de la repetición máxima. Favorece a aquellos patrones cuyas ocurrencias suponen una fracción significativa del volumen de la página.

Para cada uno de estos criterios el usuario puede establecer un umbral o utilizar el valor por defecto. En esta fase, para algunas fuentes que no cumplan los criterios establecidos, puede ser necesario dividir las ocurrencias del patrón, de manera que si las ocurrencias de alguna de las particiones cumplen los criterios por separado, entonces no son descartadas.

El *Constructor de Reglas* revisa cada patrón candidato para formar una regla de extracción en forma de expresión regular. Para permitir emparejamientos inexactos o aproximados se utiliza una técnica de alineamiento múltiple de secuencias descrita en [GBS92] para extender las repeticiones exactas encontradas. Para un patrón que tenga $k+1$ ocurrencias que empiecen en las posiciones $p_1, p_2, \ldots, p_{k+1}$ en la cadena codificada, si C_i denota a la cadena que comienza en p_i y acaba en $p_{i+1}-1$, el problema consiste en encontrar el alineamiento múltiple de las k cadenas $C_1, C_2, \ldots, C_k$. En la Figura 18 se muestra un ejemplo para el patrón '`adc`' y la cadena '`adcwbdadcxbadcxbdadcb`', obteniéndose el patrón de extracción '`adc[w|x|-]b[d|-]`' tras el alineamiento múltiple de las diferentes cadenas que contenían el patrón encontrado.

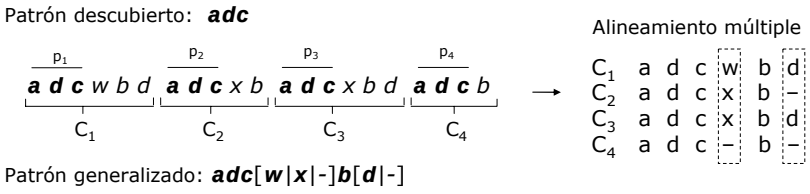


Figura 18 Alineamiento múltiple

Otro problema relativo a los patrones de extracción alineados, es que el patrón generado no necesariamente encajará con los límites de los registros de información. El patrón podría empezar a encajar en medio de un registro y emparejar con el final de ese registro y el comienzo del siguiente. Por tanto se necesita un paso adicional para ver cuál es el patrón correcto a utilizar, es decir, cuál es la posición dentro del patrón generado que encaja con el principio de los registros a extraer (una vez determinado ese punto, el patrón resultante se construye pasando al final del patrón la parte que va antes de esa posición). Para ello se generan posibles reglas de extracción teniendo en cuenta que los patrones de extracción normalmente empiezan y terminan por una marca de entre un conjunto predefinido (determinadas de una forma heurística). El problema de este paso es que se pueden generar muchos patrones de extracción candidatos.

Este sistema, a pesar de no precisar que el usuario proporcione ejemplos de entrada, requiere intervención del usuario en varias fases:

- Aunque los autores afirman que las marcas de bloque funcionan bien en la mayor parte de casos, en ciertos casos pueden no ser suficientes y es el usuario el que debe elegir las marcas a utilizar.
- Además, ya que no todos los patrones repetitivos aportan datos útiles, se utilizan una serie de heurísticas para identificar a los que sí los aportan. Si bien es posible utilizar valores por defecto, a menudo es el usuario quién debe variar los umbrales a utilizar por parte del validador para descartar los patrones candidatos no significativos. Aún así pueden quedar varios patrones repetitivos después del proceso de validación y el usuario debe elegir cuál utilizar.
- Como ya se ha comentado, los patrones generados pueden no encajar exactamente con los registros de información, y en lugar de ello encajar con la parte final de un registro y el principio de otro. Para corregir esta situación, se generan varias reglas de extracción posibles de acuerdo a una serie de heurísticas (marcas en las que suele empezar/acabar un patrón), pero todas ellas son consideradas como válidas y el usuario debe elegir la adecuada.
- En las reglas de extracción resultantes únicamente se extraen fragmentos de texto, pero no se etiquetan, ni se buscan partes de esos textos que sean fijas y que no formarían parte de los valores a extraer (e.g. prefijos y sufijos comunes).

Otro inconveniente es que se asume que las marcas HTML siempre forman parte de la plantilla a partir de la que se generan las páginas y que nunca forman parte de los valores a extraer. Esto es cierto en algunos casos pero existen múltiples excepciones en las que

algunas marcas (e.g. marcas de formateo como <b> o <i>) aparecen dentro de los valores a extraer.

Por último cabe reseñar que las técnicas utilizadas únicamente son útiles para extraer datos de tipos limitados a conjuntos de tuplas cuyo esquema sea plano (sin tipos de datos compuestos). En [WL03] se propone el sistema DeLa (*Data extraction and Label assignment*), que extiende a IEPAD para ser capaz de tratar con datos multivaluados. Necesita, sin embargo, múltiples páginas como entrada, que sigan la misma plantilla.

II.3.3.7.2. ROADRUNNER

Crescenzi et al. [CMM01] presentan RoadRunner, un sistema que partiendo de un cierto número de páginas generadas de acuerdo a una plantilla subyacente, trata de inducir dicha plantilla basándose en las similitudes y las diferencias existentes entre las páginas. El sistema no hace uso de ningún tipo de conocimiento previo sobre las páginas de la fuente, ni requiere interacción con el usuario, salvo en la fase final, para anotar la plantilla descubierta de acuerdo al esquema de salida deseado.

La base teórica del trabajo se fundamenta en un esquema de tipos, creados a partir de constructores básicos de listas y tuplas que pueden anidarse de forma arbitraria para representar el esquema de los datos contenidos en las páginas HTML. Para cada uno de estos tipos, genera una expresión regular *libre de uniones* (*UFRE*), que permitirá extraer los datos de la página. Las expresiones regulares UFRE no incluyen alternativas, es decir, operadores de unión. Esta limitación simplifica el proceso de inferencia, pero impide a RoadRunner tratar con aquellas páginas en las que en una determinada posición de la plantilla a inducir puede aparecer uno de entre varios atributos posibles. Los autores admiten esta limitación aunque afirman que su modelo es suficiente para un importante número de casos.

El problema de obtener la plantilla utilizada para generar las páginas se traduce en encontrar la mínima UFRE con la que concuerden todas las páginas de entrada. Para ello establecen un proceso iterativo que toma la primera página como UFRE inicial, y para cada nueva página comprueba si puede ser generada por la plantilla actual. Si no puede serlo, entonces se modifica la plantilla actual para que concuerde con todas las páginas consideradas hasta el momento.

Los autores utilizan el algoritmo ACME (*Align, Collapse under Mismatch, and Extract*) para el proceso de comparación y actualización de la plantilla con nuevas páginas. Las páginas de entrada al algoritmo se transforman a XHTML, y para cada una de ellas se obtiene una lista de *tokens*. Un *token* puede ser una marca HTML o un texto.

El algoritmo de emparejamiento trabaja con dos listas de *tokens* denominadas *ejemplo* y *envoltorio*. El ejemplo representa a la página considerada en la iteración actual, y el envoltorio es la expresión regular libre de uniones que representa a la plantilla actual. ACME recorre las dos listas de *tokens* de forma simultánea, encontrando y resolviendo *fallos de emparejamiento* entre ambas. Un fallo de emparejamiento ocurre cuando un *token* del ejemplo no concuerda con la posición correspondiente del envoltorio. En función del tipo de *tokens* que hayan provocado el fallo, se denominan fallos de cadenas, cuando textos

diferentes están en posiciones correspondientes en ambas listas, y fallos de marcas en el resto de casos. En la Figura 19 se muestra un ejemplo de funcionamiento del algoritmo.

Cuando se produce un fallo de cadena, se añade un nuevo atributo al esquema de salida. Los autores afirman que si dos páginas han sido generadas de acuerdo a la misma plantilla, entonces los fallos de cadena sólo pueden deberse a que las cadenas que han provocado el fallo constituyen diferentes valores para un mismo atributo.

Los fallos de marcas se utilizan para descubrir constructores de lista y atributos opcionales. Cuando se produce un fallo de marca, primero se intenta encontrar un patrón repetitivo que sugiera un constructor de lista. En otro caso se intenta encontrar un patrón opcional.

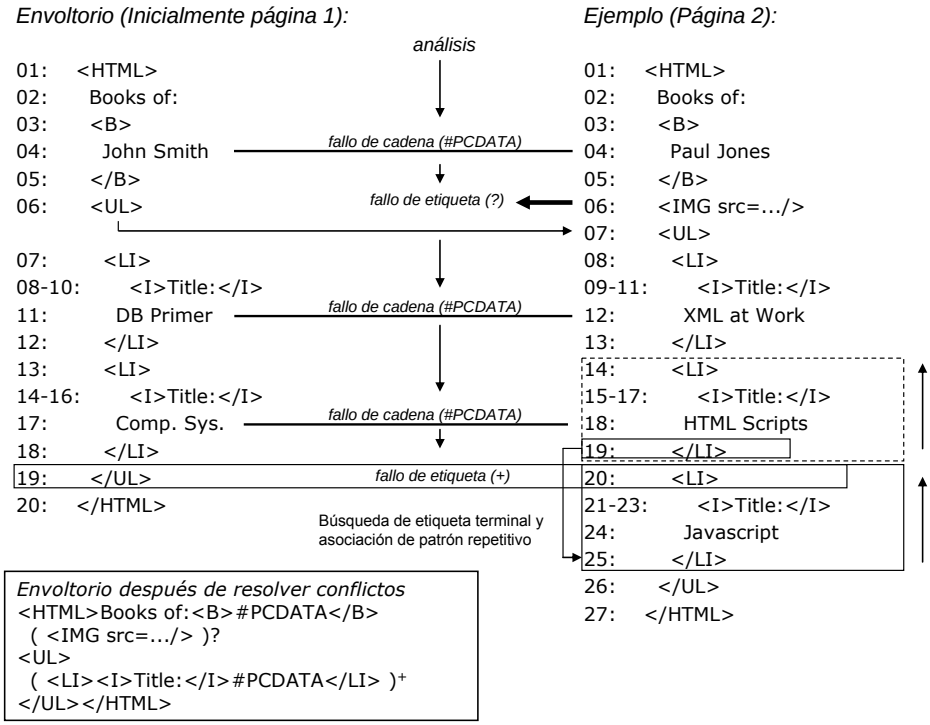


Figura 19 Ejemplo de funcionamiento de RoadRunner

Para buscar patrones opcionales se supone que en el envoltorio o en el ejemplo hay un trozo de código HTML que no está presente en el otro, de manera que saltándoselo sería posible continuar con el emparejamiento. Para encontrar ese fragmento se realiza una búsqueda cruzada en el envoltorio y el ejemplo, y una vez identificado, se generaliza el envoltorio como corresponda.

Para buscar patrones repetitivos (a los que los autores denominan *squares*) se siguen tres pasos:

- Se asume que tanto el envoltorio como el ejemplo contienen al menos una repetición del patrón repetitivo. Por lo tanto, considerando el *token* anterior al fallo de emparejamiento se obtendrá el último *token* del patrón repetitivo. En un caso se

tratará de la última ocurrencia del patrón repetitivo, pero en el otro a continuación vendrá otra ocurrencia y, por tanto, uno de los dos *tokens* que provocó el fallo tiene que ser el principio del patrón repetitivo. Se exploran ambas posibilidades, buscando la siguiente ocurrencia del *token* identificado como de fin de patrón repetitivo tanto en el envoltorio como en el ejemplo. En caso de encontrarse, se obtienen *ocurrencias candidatas* del patrón repetitivo.

- Para verificar si una *ocurrencia candidata* realmente identifica a un *square*, se intenta emparejar la ocurrencia del *square* candidato contra alguna porción anterior del envoltorio o ejemplo (según corresponda).
- Por último se generaliza el envoltorio. Para ello se buscan ocurrencias contiguas del patrón repetitivo identificado y se sustituyen por una expresión regular que concuerde con una o más repeticiones del mismo.

Hay que tener en cuenta que mientras se están resolviendo fallos de emparejamiento pueden aparecer otros fallos, denominados fallos de emparejamiento internos. Esto provoca que el algoritmo sea recursivo. Estos fallos se resuelven igual que los fallos externos, con la única diferencia que en este caso no se compara el envoltorio con el ejemplo, sino dos partes del mismo objeto (envoltorio o ejemplo, según corresponda).

Otra fuente de complejidad es que en el algoritmo muchas veces es necesario elegir entre diferentes alternativas posibles, y el camino seleccionado puede no llevar a una solución correcta, siendo necesario volver atrás para explorar otra alternativa. Por ejemplo cuando se está intentando localizar un patrón repetitivo, buscando la marca identificada como de fin de patrón repetitivo, pueden existir varias ocurrencias candidatas de esa marca.

Si bien es posible construir un árbol con todos los posibles caminos a seguir, la complejidad del algoritmo para recorrer ese árbol es exponencial. Para reducirla se utilizan técnicas de poda que eliminan aquellos caminos que se correspondan con opciones poco prometedoras. Por ejemplo se limita el número máximo de alternativas a explorar en cada situación, evaluando únicamente los patrones candidatos más cortos, y se eliminan los caminos que llevan a envoltorios en los que un patrón (opcional o iterador) está delimitado por cualquier lado por un patrón opcional.

Entre las limitaciones de este sistema cabe destacar que no es capaz de tratar ciertos tipos de fuentes, algunas debido al enfoque teórico, ya que hay clases de páginas que conforman un lenguaje regular que requiere de uniones, y otras debido a las limitaciones introducidas en la implementación para reducir la complejidad del algoritmo, como por ejemplo la imposibilidad de adyacencias entre constructores de tipo lista y opcionales.

Otro inconveniente es que se asume que las marcas HTML siempre forman parte de la plantilla a partir de la que se generan las páginas y que nunca forman parte de los valores a extraer. Esto es cierto en muchos casos, pero en un número significativo de fuentes las marcas aparecen dentro de los valores a extraer.

Otro aspecto a considerar es que las páginas que se proporcionan a la entrada deben ser representativas de todos los casos que puedan darse en las páginas de la fuente y estar bien elegidas. Por ejemplo, para que el sistema sea capaz de identificar las partes repetitivas es necesario que las páginas de entrada tengan diferente número de elementos en esa parte repetitiva.

II.3.3.7.3. EXALG

Arasu y García-Molina [AG03a] presentan EXALG, otro algoritmo para extraer automáticamente datos estructurados codificados en una colección de páginas web generadas utilizando la misma plantilla.

Los autores consideran como datos estructurados cualquier conjunto de valores conformes a un *esquema* o *tipo* común. Un *tipo* T_i se define de forma recursiva [AHV95] a partir de un tipo básico denominado β (representa una cadena de *tokens* que pueden ser palabras o marcas HTML), y dos constructores de tipo: el de tupla de orden n, $\langle T_1,\ldots,T_n\rangle$ y el de conjunto, $\{T\}$.

A partir de los constructores de tupla y conjunto, se definen otros dos constructores de tipos que ocurren normalmente en las páginas web: los atributos opcionales, $(T)?$ – equivalente a $\{T\}_\tau$ con la restricción de que cualquier instancia de τ tiene cardinalidad 0 o 1 – y las alternativas, $(T_1 \mid T_2)$ – equivalente a $\langle\{T_1\}_{\tau 1},\{T_2\}_{\tau 2}\rangle_\tau$, donde para cada instancia de τ, τ_1 o τ_2 tienen cardinalidad uno y el otro cardinalidad cero –.

Una plantilla P para un esquema E, se define como una función que asigna a cada constructor de tipo τ de E un conjunto ordenado de cadenas $P(\tau)$, tal que:

- Si τ es un constructor de tupla de orden n, $P(\tau)$ es un conjunto ordenado de $n+1$ cadenas $(C_{\tau 1},\ldots,C_{\tau(n+1)})$. Cada una de esas cadenas se correspondería con el fragmento de plantilla presente entre cada dos valores de la tupla.
- Si τ es un constructor de conjunto, $P(\tau)$ es una cadena C_τ. Esa cadena se correspondería con el fragmento de plantilla presente entre cada dos ocurrencias del conjunto.

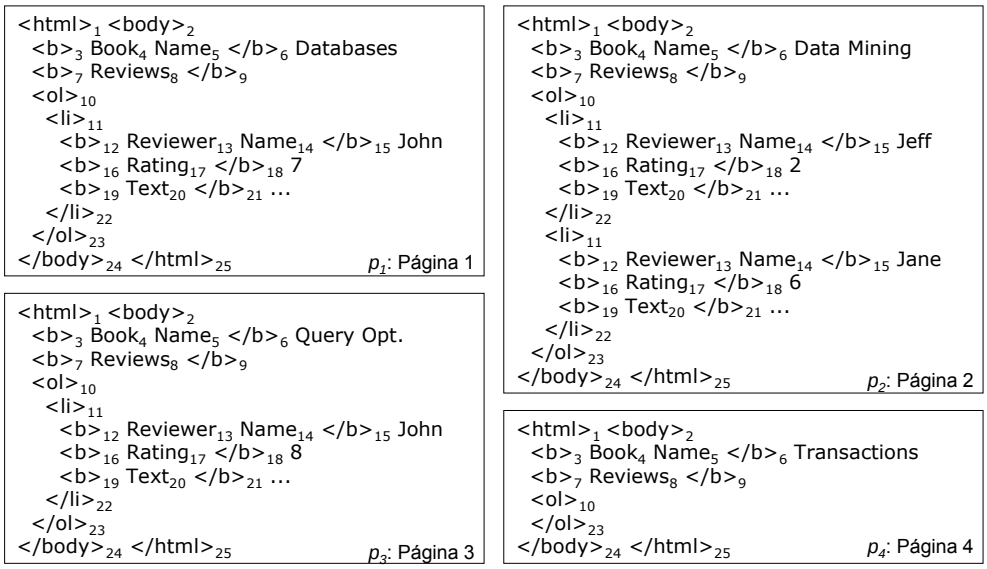


Figura 20 Páginas de entrada p_1, p_2, p_3, p_4

x_1 :	Databases	{ < John, 7, … > }
x_2 :	Data Mining	{ < Jeff, 2, … >, < Jane, 6, … > }
x_3 :	Query Opt.	{ < John, 8, … > }
x_4 :	Transactions	Ø

Valores correctos

```
<
<html>1 <body>2
 <b>3 Book4 Name5 </b>6 *
 <b>7 Reviews8 </b>9
 <ol>10
 { <
  <li>11
   <b>12 Reviewer13 Name14 </b>15 *
   <b>16 Rating17 </b>18 *
   <b>19 Text20 </b>21 *
  </li>22
 > }
 </ol>23
</body>24 </html>25
>
```

Plantilla P

Figura 21 Plantilla *P* y valores x_i a partir de los que se han generado las páginas de la Figura 20

Para entender mejor el algoritmo, considérese el siguiente ejemplo, donde se utilizan las cuatro páginas de ejemplo $\{p_1, p_2, p_3, p_4\}$, que se muestran en la Figura 20. Cada página p_i contiene el título y el conjunto de revisiones de un libro. Cada revisión contiene el nombre del revisor, la puntuación asignada y un comentario. El texto completo de los comentarios no se muestra por limitaciones de espacio. Las páginas fueron creadas a partir de la plantilla *P*, y los valores $\{x_1, x_2, x_3, x_4\}$ mostrados en la Figura 21. El esquema de los valores es $E=<\beta,\{<\beta, \beta, \beta>_{\tau 1}\}_{\tau 2}>_{\tau 3}$.

En la Figura 22 se muestra el diagrama de componentes del algoritmo utilizado para la extracción de datos.

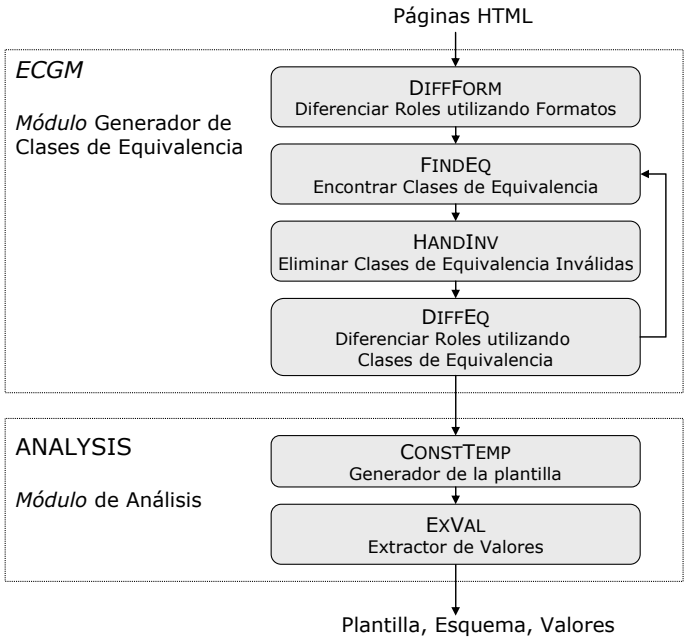


Figura 22 Algoritmo EXALG

EXALG divide su funcionamiento en dos fases. En la primera el módulo *ECGM (Equivalence Class Generator Module*) descubre *tokens* asociados con el mismo constructor de tipo en la plantilla (desconocida) usada para generar las páginas de entrada. En la segunda el módulo ANALYSIS usa esos *tokens* para deducir la plantilla. Finalmente, la plantilla deducida es usada para extraer los datos codificados en las páginas.

En la primera fase, el módulo FINDEQ analiza las páginas de entrada para calcular las llamadas *clases de equivalencia*. Una clase de equivalencia está formada por un conjunto de *tokens* que tienen la misma frecuencia de aparición en todas las páginas de entrada. En el ejemplo de la Figura 20 hay nueve clases de equivalencia. Un ejemplo de clase de equivalencia es ε_1, que está formada por el conjunto de 8 elementos {<html>, <body>, Book, Reviews, <ol>, , </body>, </html>}, donde cada *token* ocurre exactamente una vez en cada página de entrada. El subíndice especificado en las clases de equivalencia identifica el constructor de tupla al que corresponden los elementos de la clase de equivalencia.

Se consideran solamente aquellas clases de equivalencia que están formadas por un número mínimo de *tokens* y que aparecen en un número mínimo de las páginas de entrada (los umbrales utilizados han sido deducidos de forma heurística). A esas clases de equivalencia se les llama LFEQs (*Large and Frequently occurring EQuivalence classes*). En el ejemplo de la Figura 20 hay dos LFEQs. La ya citada ε_1, y ε_3, que consta de 5 elementos, {
, Reviewer, Rating, Text, }. Cada elemento de ε_3 ocurre una vez en p_1 y p_3, dos veces en p_2 y ninguna en p_4.

La intuición básica en la utilización de las LFEQs es que es poco probable que una LFEQ se forme por casualidad y casi siempre estarán formadas por *tokens* asociados al mismo constructor de tipo, en la plantilla utilizada para crear las páginas de entrada. Por tanto pueden ser utilizadas para deducir la plantilla y el esquema de los datos.

Sin embargo, siempre hay algunas LFEQs inválidas que sí se forman por casualidad. La detección y eliminación de LFEQs inválidas la realiza el submódulo HANDINV, en base a dos propiedades:

- Las clases de equivalencia válidas son ordenadas, es decir, se pueden ordenar sus *tokens* de forma que indiquen su orden de aparición en todas las páginas.
- Dos clases de equivalencia válidas deben estar anidadas, es decir, las ocurrencias en las páginas de los *tokens* de una y otra clase nunca se entremezclan, o si lo hacen entonces las ocurrencias de todos los *tokens* de una de las clases siempre ocurren entre ocurrencias de los dos mismos *tokens* de la otra clase.

Un problema adicional es que, en una página, puede haber ciertos *tokens* cuyas ocurrencias jueguen papeles diferentes (es decir, que estén asociados a más de un constructor de tipo). Por ejemplo una misma palabra o marca HTML puede aparecer en diferentes partes de la plantilla. De la diferenciación de papeles se encargan los submódulos DIFFFORM y DIFFEQ que utilizan para ello dos técnicas basadas, respectivamente, en las siguientes observaciones:

- La primera de ellas emplea información de formato de la página HTML vista como un árbol y dice que si dos ocurrencias de un *token* tienen diferente camino desde la raíz del árbol entonces juegan papeles diferentes.

- La segunda dice que si una ocurrencia de un *token*, aparece en una posición relativa a una clase de equivalencia, diferente a la de otra ocurrencia de ese mismo *token*, entonces ambas ocurrencias juegan papeles diferentes. Esto puede suceder porque una aparece entre dos ocurrencias de *tokens* de la clase de equivalencia y la otra no, o porque ocurren entre ocurrencias de diferentes elementos de la clase de equivalencia.

Se utiliza el termino *dtoken* para referirse a un *token* junto con su contexto identificado por el proceso de diferenciación. En la Figura 20 cada *dtoken* está identificado por un subíndice diferente. Si las LFEQs se construyesen a partir de los *dtokens*, entonces por ejemplo, $\texttt{Name}_5$ pertenecería a ε_1 y $\texttt{Name}_{14}$ a ε_3 (puesto que tienen el mismo número de ocurrencias en cada página que el resto de *tokens* de esas clases de equivalencia).

El funcionamiento del módulo *ECGM* es el siguiente: Primero, el submódulo DIFFFORM diferencia los papeles de los elementos utilizando la primera de las observaciones comentadas. A continuación los submódulos FINDEQ, HANDINV y DIFFEQ se ejecutan dentro de un bucle. En cada iteración FINDEQ determina las LFEQs existentes utilizando los *dtokens* actuales, HANDDIV procesa esas LFEQs para eliminar las inválidas produciendo como salida un conjunto ordenado de LFEQs anidadas. Finalmente DIFFEQ utiliza la segunda de las observaciones comentadas para diferenciar nuevos *tokens*, de manera que si se encuentra nuevos *dtokens* se pasa a una nueva iteración y en caso contrario se finaliza devolviendo las LFEQs.

Al final de esta fase, las clases de equivalencia detectadas para el ejemplo de la Figura 20 aumentan su número de elementos. La primera, ε_1, pasa de 8 a 13 elementos {<html>, <body>, , Book, Name, , , Reviews, , <ol>, , </body>, </html>}, donde cada *token* ocurre exactamente una vez en cada página de entrada. La segunda, ε_3, pasa de 5 a 12 elementos, {
, , Reviewer, Name, , , Rating, , , Text, , }.

El módulo ANALYSIS se encarga de crear la plantilla de salida utilizando las LFEQs construidas en la fase anterior. El módulo CONSTTEMP construye de forma recursiva una plantilla para cada clase de equivalencia y para cada posición no vacía de esa clase de equivalencia.

El procedimiento que se sigue es el siguiente: Se comienza identificando la LFEQ raíz, que será aquella cuyos *tokens* ocurren únicamente una vez en cada página de entrada. A continuación se determinan las posiciones entre *tokens* consecutivos de la clase de equivalencia que no estén vacías. Una posición entre dos *tokens* consecutivos está vacía si los dos *tokens* siempre ocurren consecutivamente en la página y no vacía en otro caso. A continuación se construye un constructor de tupla de orden *n* (un atributo para cada posición no vacía) y de forma recursiva se construye la plantilla correspondiente a cada posición no vacía analizando si todas las ocurrencias de las cadenas correspondientes a esa posición no vacía siguen algún patrón reconocible:

- Si son una cadena de *dtokens* y no contienen ocurrencias de ninguna clase de equivalencia, entonces el tipo del atributo es un tipo básico β.
- Si son una clase de equivalencia con varias ocurrencias, entonces el tipo de esa clase de equivalencia se inserta dentro de un constructor de conjunto. En este caso hay

que diferenciar dos situaciones que se tienen en cuenta a la hora de generar la plantilla:

  - Si las ocurrencias de la clase de equivalencia están consecutivas entonces no hay separador entre ellas.
  - Si hay un separador de *dtokens* entre cada ocurrencia de la clase de equivalencia entonces se considerará ese separador al generar la plantilla.

- Si a veces ocurre una clase de equivalencia y a veces otra, entonces los tipos de esas clases de equivalencia se insertan dentro de una disyunción.
- Si a veces ocurre una clase de equivalencia y a veces nada, entonces se inserta el tipo de la clase de equivalencia dentro de una opcionalidad.
- En cualquier otra situación se inserta un tipo básico β.

En cada paso, para generar la plantilla, se asigna a cada constructor de tipo un conjunto ordenado de cadenas. Para los constructores de tupla de orden n se construyen $n+1$ cadenas a partir de los *dtokens* de la LFEQ correspondiente, que quedan entre las posiciones no vacías. Para los constructores de tipo conjunto se busca el separador que hay entre cada dos ocurrencias de la LFEQ asociada al constructor y se asigna esa cadena (o la cadena vacía en caso de no haber tal separador).

En los experimentos realizados se muestra que en global son extraídos correctamente el 80% de los atributos considerados en las distintas fuentes. El resto de atributos se considera que son extraídos parcialmente correctos, situación que ocurre cuando el nivel de detalle de la extracción es mayor que el deseado, es decir, cuando el valor del atributo está contenido en un fragmento mayor considerado como un atributo único por el extractor.

El sistema se basa en una serie de premisas que como los autores reconocen no siempre se cumplen. Estas son:

- Existe un número suficientemente grande de *tokens* en la plantilla que tienen un papel único, que permite iniciar el proceso de formación de clases de equivalencia y consiguiente diferenciación de papeles.
- Un número suficientemente grande de tokens está asociado con cada constructor de tipo. Esto permite asegurar que la clase de equivalencia derivada de ese constructor de tipo se reconoce como una LFEQ.
- No existe ninguna regularidad en los datos codificados que lleve a la formación de clases de equivalencia inválidas.
- Hay separadores alrededor de los valores de datos.

En los resultados, los autores comentan que empíricamente han observado que el caso más común de fallo se da con la segunda premisa, es decir, hay constructores de tipos que tienen muy pocos *tokens* de la plantilla asociados con ellos. También han detectado casos en los que la primera premisa falla, y los *tokens* de la plantilla no pueden ser completamente diferenciados. Y finalmente la cuarta premisa también falla, aunque en raras

ocasiones, cuando los atributos están semánticamente muy relacionados (por ejemplo año, mes y día en fechas: '15 Nov 2006').

II.3.3.7.4. DEPTA

Zhai y Liu [ZL06] presenta otro sistema de estructuración automática, llamado DEPTA. Utilizan técnicas de alineamiento de árboles en lugar de alineamiento de cadenas como en IEPAD [CL01], explotando la estructura anidada del árbol para realizar una extracción de datos más precisa. Además, permite extraer datos de una única página, con múltiples registros.

El funcionamiento del sistema tiene lugar en dos fases: identificación de registros y obtención de los valores de los atributos de cada registro. En la Figura 23 se muestra el diagrama de módulos general del algoritmo.

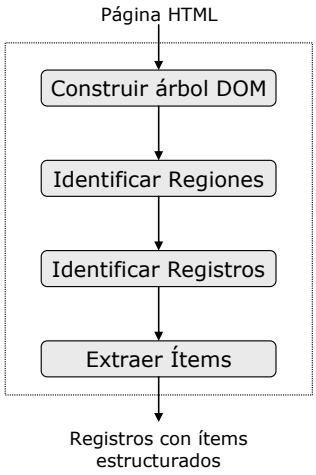


Figura 23 Diagrama de módulos del algoritmo de extracción DEPTA

En la primera fase se identifican los fragmentos de la página HTML que se corresponden con registros de datos presentes en las páginas, sin extraer el valor de sus atributos. El método empleado utiliza, entre otras, información de representación visual de la página obtenida de un navegador web, que ayuda al sistema de dos formas:

- Le permite identificar 'huecos' para separar los registros, utilizando la observación de que normalmente los huecos (o espacios visuales vacíos) dentro de un registro son más pequeños que el hueco entre dos registros.
- Se utiliza para inferir la relación estructural entre las marcas de la página y construir a partir de ella un árbol más robusto (que un árbol DOM [W3CDOM] u otro tipo de representación basada únicamente en las marcas HTML), debido a la mayor tolerancia ante errores en el código HTML que presentan los motores de visualización de los navegadores web.

En la segunda fase se obtienen los valores de los atributos de cada registro. Cada fragmento HTML conteniendo un registro se representa como un árbol. Dichos árboles se

proporcionan como entrada para un algoritmo de *alineamiento de árboles* [Tai79] [Valiente02] [Selkow77], que es similar conceptualmente a las técnicas de *alineamiento de cadenas de caracteres* [CRF03], pero adaptadas al caso de estructuras de datos en árbol. El alineamiento de dos árboles consiste en emparejar los nodos de un árbol con los del otro, de forma que se minimice la *distancia de edición* entre ambos. La *distancia de edición entre dos árboles* es el número mínimo de operaciones de edición (inserción, borrado y sustitución de nodos) que es necesario realizar sobre uno de los árboles para transformarlo en el otro. El sistema DEPTA considera que los nodos que quedan alineados entre sí en todos los árboles de entrada representan marcas HTML que juegan el mismo papel en todos los registros y, por lo tanto, son los 'separadores' entre atributos que definen la plantilla (los datos a extraer serán los textos entre dichos separadores). El sistema utiliza una técnica de alineamiento parcial que sólo alinea aquellos nodos que se pueden alinear con total certeza e ignora las demás partes (no se realizan suposiciones sobre las partes no alineadas).

El algoritmo de identificación de registros se basa en dos observaciones acerca de cómo suelen distribuirse los registros en una página y en un algoritmo de distancia de edición de cadenas [Baeza89]. Las dos observaciones son las siguientes:

- Un grupo de registros que contienen descripciones de un conjunto de objetos similares, normalmente se encuentran en una región contigua de la página y están formateados utilizando marcas HTML similares. Es la denominada *región de datos*. Una página puede contener más de una región de datos.
- La estructura de marcas anidadas de una página HTML forman un árbol. Un conjunto de registros de datos similares, están formados por una serie de subárboles hijos del mismo nodo padre (un registro no comienza ni termina en mitad de un subárbol).

El algoritmo de división en registros se divide en tres pasos:

- Se construye un árbol de marcas de la página. Para ello se utiliza información de visualización de un navegador web, basándose en que cada elemento HTML se visualiza como un rectángulo. El árbol se construye obteniendo las coordenadas de cada elemento, y en función de las relaciones de contención entre los rectángulos.
- Utilizando el árbol de marcas se buscan las regiones de datos en la página. Una región de datos se define como un área que contiene una lista de registros de datos con formato similar. Para calcular el grado de similitud entre dos registros, cada registro candidato se codifica como una cadena de caracteres y se utiliza un algoritmo de cálculo de distancia de edición entre cadenas. La idea en la que se basa esta búsqueda consiste en ir construyendo y comparando registros candidatos a partir de las marcas correspondientes a nodos individuales (incluyendo todos sus descendientes), o combinaciones de nodos adyacentes, que son hijos de un mismo nodo padre, a lo largo de toda la página. A cada nodo individual o combinación de nodos similares encontrados se le denomina *nodo generalizado*. Por tanto una secuencia de nodos generalizados adyacentes forman una región de datos. Para eliminar combinaciones de nodos falsas se utiliza información de distancia visual, basándose en la observación de que la distancia entre dos registros de una región de datos no debe ser mayor que cualquier espacio dentro de un registro.

- Se identifican los registros dentro de cada región de datos. Una vez identificada la región de datos, cada nodo generalizado (un nodo individual o una combinación de nodos) de cada región de datos podría no contener un registro de datos individual. En este paso se identifican los nodos que se corresponden con cada registro. Por ejemplo, si cada registro se corresponde con una columna de una tabla HTML, en lugar de una fila, la información del registro no estará contigua en el código HTML, y en el paso anterior se habrán identificado varias regiones de datos, pero juntando los nodos de cada región de datos se pueden obtener registros de datos no contiguos en el código HTML.

Una vez dividida la página en registros se procede a la extracción de datos. Para cada registro se construye un árbol, creando nodos artificiales en caso de que el registro esté contenido en más de un subárbol del árbol de marcas original de la página. Los árboles de todos los registros de cada región de datos se alinean utilizando un método de alineamiento parcial basado en emparejamientos de árboles. Se utilizan alineamientos y distancias de edición de árboles [Tai79] en vez de trabajar con cadenas porque así se tiene en cuenta la estructura del árbol, y se reducen notablemente los alineamientos posibles. Pero aún así puede haber conflictos cuando hay más de un posible alineamiento, en cuyo caso se sigue la regla de elegir el alineamiento que se corresponde con el subárbol que aparece antes.

El algoritmo de alineamiento múltiple propuesto, alinea múltiples árboles de marcas aumentando progresivamente un árbol semilla. Como árbol semilla se elige inicialmente, de entre todos los árboles, aquel con el mayor número de campos de datos. A continuación, para cada árbol, de entre los restantes, el algoritmo trata de encontrar para cada nodo un emparejamiento con un nodo del árbol semilla. Cuando para un nodo se encuentra un emparejamiento, se crea una asociación con el nodo correspondiente del árbol semilla. Si no se encuentra emparejamiento, entonces se intenta expandir el árbol semilla insertándole ese nodo, de manera que en los siguientes emparejamientos se utilice el árbol expandido. Un aspecto a comentar es que los campos de datos del árbol de marcas no se utilizan durante el emparejamiento ni durante el alineamiento.

Cuando en un árbol quedan nodos sin alinear, se insertan en una lista los árboles que necesitan procesarse de nuevo después de alinear el resto de árboles. Esto es así porque después de haber alineado los demás árboles, las modificaciones en el árbol semilla pueden hacer que las posiciones de inserción que antes eran ambiguas, dejen de serlo.

Este método de estructuración automática presenta los siguientes inconvenientes. En la fase de división en registros se utiliza una heurística que dice que el espacio visual entre dos registros siempre es mayor que los espacios que pueda haber dentro de un registro. Esto se cumple en bastantes casos, pero hay numerosas excepciones y la utilización de esta heurística llevaría a una división en registros errónea. Otro problema con el que se encontraría este sistema en algunas fuentes es que, aunque el sistema considera que un registro puede abarcar varios subárboles, asume que todos los registros de una región de datos ocupan el mismo número de subárboles. De nuevo esta observación se cumple en bastantes fuentes, pero no puede ser asumida en general, ya que hay fuentes en las que registros de una misma región de datos pueden abarcar diferente número de subárboles, debido a que contienen partes opcionales que abarcan subárboles completos.

II.4. DISCUSIÓN Y CONCLUSIONES

En esta sección se han estudiado las diversas aproximaciones propuestas hasta la fecha para recopilar y estructurar de forma automática información contenida en la Web Oculta que sea relevante para una temática determinada. Los principales objetivos del estudio del arte realizado son: (1) descomponer la problemática de localización y extracción de información relevante de la Web Oculta de forma estructurada, para evaluar qué sistemas y técnicas existentes permiten tratar todos o alguno de los desafíos que plantea, y (2) conocer las limitaciones de las técnicas actuales.

En cuanto al primer objetivo, en primer lugar se han descrito las soluciones que existen a un problema que afecta a todo tipo de sistemas de *crawling* de la Web orientados a un dominio de aplicación, con independencia de si la información que interesa obtener se encuentra en la Web Oculta o en la Web de Superficie. La utilización de las técnicas de *crawling* dirigido comentadas en la sección II.2 permite guiar un proceso de *crawling* por páginas relevantes a un dominio o por páginas que aunque no relevantes por ellas mismas, proporcionan acceso a páginas que sí lo son. La idea básica de estas técnicas es utilizar algún tipo de clasificador que permita decidir, para una página determinada, si es relevante o si forma parte del camino hacia páginas relevantes. La mayor parte requieren de una fase de entrenamiento del sistema, para que el clasificador aprenda a detectar las páginas interesantes.

Las aportaciones originales de esta tesis doctoral no se centran en esta parte del problema, aunque se han estudiado las soluciones existentes porque es necesario proporcionar alguna solución al mismo en la arquitectura general presentada.

A continuación se han estudiado las diversas aproximaciones propuestas para cada uno de los problemas que involucra el acceso a la información de la Web Oculta en el contexto de sistemas de *crawling* dirigido. La conclusión obtenida es que algunos de los problemas todavía no han sido abordados y las soluciones para otros presentan importantes limitaciones. A continuación, se repasa brevemente cada uno de ellos.

En la sección II.3 se realizó una primera división de los problemas que involucra el tratamiento de la Web Oculta, diferenciando la Web Oculta del lado cliente de la del lado servidor.

El desafío que presenta el tratamiento de la Web Oculta del lado cliente no ha sido apenas abordado hasta el momento. Esto hace que una parte importante de la Web de Superficie – cada vez mayor debido a la utilización creciente de tecnologías del lado cliente para mejorar la interacción de los sitios web – quede fuera del alcance de los sistemas de *crawling* tanto convencionales como dirigidos.

Algunos sistemas de *crawling* [WC07] han incluido intérpretes de *JavaScript* [MR07] [MSM07] en los clientes HTTP que utilizan para proporcionar algún soporte, aunque muy limitado, para este escenario. Sin embargo, sólo se ocupan de la parte de generación del contenido de la página, que es sólo uno de los desafíos que presentan las tecnologías utilizadas en la implementación de páginas HTML. No tratan con el dinamismo de

navegación (el presente en los enlaces) ni con las consecuencias que de él se derivan (e.g. la aparición y desaparición de nuevos enlaces como respuesta a acciones del usuario). Una de las contribuciones de esta tesis doctoral es un conjunto de técnicas que solucionan en gran medida el problema del *crawling* de la Web Oculta del lado cliente (ver sección III.3).

En cuanto a la Web Oculta del lado servidor, tampoco existen sistemas que realicen un tratamiento completo y automatizado para obtener una visión estructurada de la información que contiene. Existen, sin embargo, aproximaciones parciales para algunos de los desafíos que plantea.

En general, los desafíos que presenta el tratamiento de la Web Oculta del lado del servidor para un sistema guiado por un dominio, se pueden dividir en los siguientes: localización de formularios relevantes, modelado de formularios y aprendizaje de la forma de hacer consultas sobre ellos, generación de consultas a ejecutar, y extracción de los registros estructurados contenidos en las páginas de resultados obtenidas.

En la parte relativa a la localización de formularios, Barbosa y Freire [BF07] generalizan el concepto de *crawling* dirigido, considerando nuevos clasificadores que permiten dirigir el proceso hacia formularios relevantes para un dominio de aplicación. De esta forma, se permite que el recorrido no diverja a sitios web que no contienen páginas ni formularios relevantes para el dominio. Aunque presenta ciertas limitaciones, en esta tesis doctoral se ha considerado esta solución como adecuada y estas técnicas tendrán un sitio en la arquitectura propuesta en la sección III.2.

Para los problemas de modelado de formularios, reconocimiento de formularios relevantes y aprendizaje de la forma correcta de rellenarlos, los sistemas existentes presentan ciertas limitaciones. Se pueden dividir en función del enfoque para el que han sido diseñados en sistemas de *crawling* y sistemas de metabúsqueda.

Desde el punto de vista de los sistemas de *crawling*, el más completo es HiWE [RG01]. HiWE utiliza heurísticas de distancia visual y similitud textual para modelar los formularios y posee un concepto similar a un dominio de aplicación contra el que comparar el formulario modelado y decidir si es relevante. Sin embargo presenta diversas limitaciones:

- En la selección de los textos primero localiza los 4 textos más próximos al campo para después seleccionar uno de ellos en base a varias heurísticas. Estas heurísticas tienen en cuenta solamente la posición relativa de los textos candidatos respecto al campo (privilegia textos a la izquierda y encima del campo), tamaño de fuentes y estilos. Estas heurísticas fallan en un importante número de casos.
- Sólo asocia un texto con cada campo del formulario, cuando en realidad podría haber varios y esta información es de gran utilidad para posteriormente ser capaces de realizar consultas sobre el formulario.
- No es capaz de reconocer dependencias entre campos del formulario (por ejemplo, si hay un selector para provincia y otro para localidad, los valores del selector de localidad dependen de la provincia seleccionada). Esta última limitación la plantean los propios autores de HiWE como trabajo futuro.

Existen otras aproximaciones también de *crawling* como [KBGP01], [BC03], o [Kushmerick03], pero presentan limitaciones adicionales como soportar sólo subconjuntos

de los elementos del formulario (sólo campos textuales o sólo campos con un conjunto limitado de valores posibles) o no utilizar distancias visuales para modelar el formulario.

Existen otros enfoques que se ocupan de la problemática de modelado de formularios desde el punto de vista de metabúsqueda, pero sólo abordan una parte del problema. Por ejemplo, MetaQuerier [ZHC04] sólo se ocupa del análisis y modelado de los formularios, pero no de emparejar los atributos especificados con la metainformación de un dominio de aplicación con los campos del formulario, un paso necesario para las aplicaciones de *crawling* dirigido. Lo mismo sucede en el caso de las técnicas utilizadas por el sistema de metabúsqueda WISE [HMYW05a].

Por lo tanto, consideramos que el aspecto de modelado de formularios y aprendizaje del método para rellenarlos correctamente, no ha sido aún totalmente resuelto para los sistemas de *crawling* dirigido. Una de las contribuciones de esta tesis doctoral es el diseño de nuevas técnicas para estas tareas, que se describen en el apartado III.4.2.

Respecto al problema de generación de las consultas a ejecutar sobre cada formulario, el sistema HiWE presenta una aproximación basada en una serie de consultas predefinidas para la tarea de *crawling* utilizada en un momento determinado. En general, este enfoque nos parece adecuado para aplicaciones de *crawling* dirigido, especialmente en el caso de la Web Oculta, donde las consultas a efectuar tenderán a ser muy precisas y, normalmente, serán especificadas directamente por el administrador que especifique la metainformación del dominio de aplicación. Existen otras aproximaciones orientadas a extraer todo el contenido almacenado tras un formulario, que también podrían aplicarse en sistemas de *crawling* dirigido siempre que se estableciese previamente la relevancia del formulario. Sin embargo, las técnicas existentes presentan importantes limitaciones en el contexto de formularios compuestos por múltiples campos característico de la Web Oculta. Solucionar estas limitaciones queda fuera del alcance de esta tesis doctoral.

El tratamiento de la Web Oculta del lado del servidor plantea un último desafío que consiste en la estructuración automática de las páginas de resultados.

Las técnicas de generación de programas envoltorio para aplicaciones de extracción de datos web han sido un campo de investigación activo durante muchos años. Han sido propuestas muchas aproximaciones, ([LRST02] proporciona una breve revisión de algunas de las principales aproximaciones). Todas las aproximaciones para generación de envoltorios requieren alguna clase de intervención humana para crear y configurar el envoltorio como paso previo a la tarea de extracción de datos. Cuando las fuentes no son conocidas de antemano, como en aplicaciones de *crawling* dirigido, esta aproximación no es factible.

Varios trabajos han abordado el problema de realizar tareas de extracción de datos web sin requerir la intervención del usuario humano. Sin embargo, los sistemas propuestos presentan algunos inconvenientes, sobre todo desde el punto de vista de un sistema de *crawling* dirigido: algunos sistemas requieren un número suficiente de páginas representativas para hacer la estructuración, como RoadRunner [CMM01] o EXALG [AG03a]. Ninguno de los sistemas aborda el problema de cómo recopilar esas múltiples páginas de entrada, que en sus sistemas se obtienen de forma manual, un enfoque no válido en las aplicaciones de *crawling* dirigido.

Los sistemas que requieren una única página exigen que éstas cumplan una serie de condiciones bastante restrictivas. IEPAD [CL01] (descrito en el apartado II.3.3.7.1) utiliza árboles Patricia [GBS92] y técnicas de alineamiento de cadenas para buscar patrones repetitivos en las cadenas de etiquetas HTML de una página. El método utilizado por IEPAD es muy probable que genere patrones incorrectos mezclados con patrones correctos, de modo que es necesario que un usuario post-procese la salida. DEPTA [ZL06] requiere que cada registro esté compuesto del mismo número de subárboles y que se cumplan heurísticas visuales como que el espacio visual entre campos del mismo registro debe ser menor que entre registros. Estas heurísticas presentan un abundante número de excepciones.

Otra de las contribuciones originales de esta tesis doctoral es un nuevo método de estructuración automática que necesita una única página de entrada, que no requiere que se cumplan las heurísticas antes mencionadas, y que ha mostrado una efectividad muy alta en las evaluaciones experimentales realizadas. Dicho método es descrito en la sección III.5.

Por último, un problema a resolver relacionado con la estructuración automática es el problema de la etiquetación de los resultados estructurados, es decir, de la asignación de nombres significativos a cada uno de los atributos de los registros extraídos. Este problema fue tratado por Arlota et al. [ACMM03] y Wang y Lochovsky [WL03] y ha sido introducido en el apartado II.3.3.7. Las técnicas propuestas en estos trabajos se han considerado adecuadas para el enfoque de *crawling* dirigido ya que pueden beneficiarse en gran medida de la metainformación incluida en el *dominio de aplicación* que describe la tarea a realizar.

III. ARQUITECTURA DE CRAWLING DIRIGIDO DE LA WEB OCULTA

Este capítulo describe detalladamente las principales contribuciones originales de esta tesis doctoral: una arquitectura completa para la construcción de sistemas capaces de efectuar tareas de *crawling* dirigido de la Web, y un conjunto de técnicas innovadoras que abordan determinados aspectos clave del problema.

En la sección III.1 se introducen los diferentes modelos subyacentes a la arquitectura y las técnicas propuestas. En la sección III.2 se describe la arquitectura global del sistema de *crawling*. El resto de secciones describen los algoritmos y técnicas definidas para solucionar ciertos aspectos clave, que constituyen las principales contribuciones de esta tesis doctoral. La sección III.3 se ocupa de las complejidades existentes en el recorrido de la Web Oculta del lado cliente. La sección III.4 aborda el desafío que presenta tratar la Web Oculta del lado servidor y la sección III.5 se ocupa de la estructuración automática de resultados obtenidos como respuesta a las consultas realizadas por el sistema durante su recorrido de la Web.

III.1. MODELOS SUBYACENTES

En esta sección se describen los modelos subyacentes en los que se basan las técnicas utilizadas para abordar el *crawling* dirigido de la información contenida en la Web Oculta.

Un *crawler* de la Web Oculta tiene que ser capaz de acceder a páginas dinámicas. Para ello debe modelar los documentos web considerando los elementos que proporcionan acceso a este tipo de páginas, así como el soporte para el tratamiento de las propias páginas dinámicas. En particular, en el apartado III.1.1 se presenta el modelo de página utilizado para capturar los elementos generadores del dinamismo del lado cliente junto con el acceso a la Web Oculta del lado servidor a través de los formularios. En el apartado III.1.2 se presenta el nuevo modelo de navegación y *crawling* sobre dichas páginas.

El apartado III.1.3 define el concepto de dominio de aplicación, responsable de la dirección de exploración de la arquitectura de *crawling*, tanto para el recorrido sobre la Web de Superficie como para reconocer y tratar con formularios de búsqueda web.

III.1.1. MODELO DE PÁGINAS DINÁMICAS

Desde un punto de vista tradicional, la Web puede definirse como una colección estática de documentos HTML, referenciados entre ellos a través de enlaces. Si D es un documento y $E_1, \ldots E_n$ son los enlaces (elementos HTML anchor) que contiene apuntando a otros documentos, se puede modelar el documento D como $D = \{E_1, \ldots, E_n\}$. Éste es el modelo básico que utilizan los sistemas de exploración de la Web convencionales, que sólo son capaces de obtener documentos pertenecientes a la Web de Superficie.

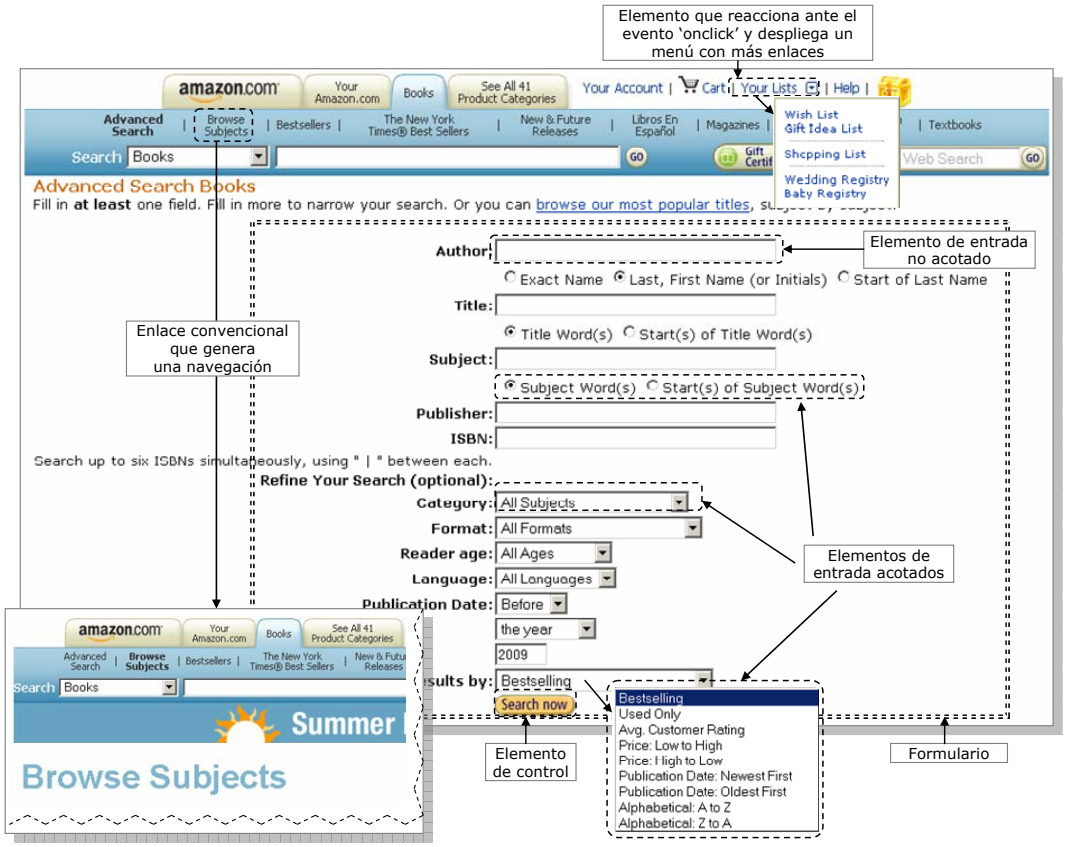


Figura 24 Elementos que forman parte del modelo extendido de páginas web

Pero en realidad las páginas web pueden presentar una estructura mucho más compleja y, con el éxito de tecnologías como *Ajax*, este tipo de páginas está aumentando. Un *crawler* que pretenda acceder a la Web Oculta debe tener en cuenta más elementos, además de los

enlaces estáticos entre páginas. Es necesario extender el modelo de página web que un *crawler* de la Web Oculta tiene que considerar.

Las páginas web actuales están compuestas por elementos que pueden reaccionar a diferentes eventos producidos por el usuario o eventos generados de forma automática por el navegador. Adicionalmente, en las páginas web también pueden aparecer elementos formulario, que constituyen el mecanismo más generalizado de acceso a la Web Oculta del lado del servidor. En la Figura 24 se muestran algunos de estos elementos. El texto 'Browse Subjects' representa un enlace convencional que especifica un URL a otra página HTML. El elemento 'Your Lists' despliega un menú, con nuevos enlaces para cada una de sus opciones, cuando un usuario hace clic sobre él, pero no genera navegaciones a nuevas páginas. En la figura también aparece un formulario web, sobre el que se especifica el tipo de alguno de sus elementos.

El modelo de documento web dinámico se define como $Dd = \{Et_1, \ldots, Et_m, Ed_1(ev_{11}, \ldots, ev_{1i}), \ldots, Ed_n(ev_{n1}, \ldots, ev_{nj}), En_1, \ldots, En_p, F_1, \ldots, F_q\}$, donde:

- $Et_1, \ldots, Et_m$ representa la colección de enlaces o anclas tradicionales, que apuntan directamente a otros documentos. Este tipo de enlace, una vez descartados aquellos que apuntan a diferentes zonas de la misma página, siempre generan nuevas navegaciones. Pueden aparecer representados en forma de textos, o a través de imágenes. Los *crawlers* convencionales se basan exclusivamente en este tipo de vínculos para realizar su recorrido de la Web.

- $Ed_1(ev_{11}, \ldots, ev_{1i}), \ldots, Ed_n(ev_{n1}, \ldots, ev_{nj})$ es la lista de elementos que son reactivos ante algún evento producido por el usuario (*Ed*), junto con los eventos ante los que puede reaccionar (*ev*). Por ejemplo aquellos elementos – típicamente enlaces, aunque no necesariamente, puesto que casi cualquier elemento HTML puede decidir manejar eventos – que ante un clic ejecutan algún *script* en el navegador. Otros elementos pueden reaccionar cuando el ratón pase sobre ellos, o cuando se intente hacer *drag&drop* de los mismos, etc. Además, tras la ejecución de ese *script* es posible que se generen una o varias navegaciones a otras páginas, o que no se produzca ninguna navegación pero sí se modifique parte del contenido de la página actual. Estos elementos incluyen a todos los componentes que una página presenta cuando utiliza tecnologías como *Ajax* para su definición. El menú 'Your Lists' en la Figura 24 es un ejemplo de este tipo de elementos, que reacciona ante el evento 'onclick' para generar nuevos enlaces en la página en forma de opciones de un menú.

- $En_1, \ldots, En_p$ es el conjunto de eventos del navegador ante los que puede reaccionar una página. Por ejemplo, una página puede ejecutar algún código en el navegador justo cuando termina de cargarse (evento 'onload').

- $F_1, \ldots, F_q$ es el conjunto de formularios presentes en la página. El modelo de formulario se describe en el apartado III.4.1.

Si una página estuviese compuesta por varios marcos (elemento HTML frame), entonces es posible definir su modelo en función de los marcos que la componen. En este caso, un documento estará formado por una colección de subdocumentos – cada uno con su propio identificador –, que a su vez pueden estar compuestos por otros documentos. En

última instancia, un documento sin marcos se define como hemos visto, a partir de sus elementos constituyentes.

III.1.2. MODELO DE NAVEGACIÓN

El tratamiento de los elementos que presentan las páginas dinámicas requiere un nuevo modelo de navegación.

Los sistemas de *crawling* convencional simplemente emiten peticiones HTTP para recuperar el contenido asociado a los diferentes URLs que van obteniendo durante el recorrido de la Web al analizar los enlaces convencionales. En cambio, las páginas dinámicas presentan otros elementos que permiten generar nuevo contenido en la página actual o nuevas navegaciones, además de formularios. Como se ha visto en el apartado III.1.1, estos elementos tienen manejadores asociados a diferentes eventos, ante los cuales se dispara la ejecución de alguna función *script*. Esa función se ejecuta en el lado cliente, con lo cual el cliente web debe disponer de un contenedor capaz de ejecutar ese código.

Para poder procesar los 'elementos dinámicos', se propone un enfoque de recorrido de la Web similar al efectuado por un usuario cuando navega. De hecho, el enfoque propuesto se basa en los eventos de navegación que genera un usuario humano cuando utiliza un navegador para acceder a una página determinada.

Definimos una secuencia de navegación como una lista de eventos efectuados sobre la interfaz de un navegador por un usuario hipotético que permite alcanzar una cierta página web, partiendo o no de otra página web. A cada evento producido por dicho usuario hipotético lo denominaremos paso de navegación. Por ejemplo, cuando un usuario quiere seguir un enlace de una página, realiza un clic sobre el elemento enlace en lugar de obtener el URL del enlace y realizar una petición HTTP directa de ese recurso. De esta forma, si el enlace tuviese asociado un evento a la acción de clic, la ejecución del código *script* asociado sería transparente al usuario.

El modelo de navegación considera todos los eventos definidos en los elementos de la página, para reproducir todas las interacciones que un usuario puede realizar sobre ellos. Para representar una secuencia de eventos de navegación en nuestro sistema, se utiliza el lenguaje NSEQL [PRAH+02] (ver apartado III.1.2.1).

Otro problema presente en los sistemas de *crawling* convencional está relacionado con el acceso futuro a documentos recolectados a partir de su URL. En un entorno dinámico como el considerado, a partir de un URL puede ser posible recuperar un recurso sólo si la sesión del usuario en ese servidor es la adecuada. Es el caso típico de recursos que requieren autenticación para acceder a ellos.

Por lo tanto, para representar un recurso en la Web introducimos el concepto de *ruta*. En el apartado III.3.1 se define en detalle qué es una ruta para nuestro sistema. En este momento nos referimos a ella como una forma de especificar un URL directo a un recurso web o, en caso de que no sea posible acceder a él directamente por problemas de sesiones o porque no se encuentre accesible en la Web de Superficie, una secuencia de pasos de navegación en base a eventos que permitan acceder a él.

El modelo de navegación en páginas dinámicas constituye una de las contribuciones que se comentará en más detalle en la sección III.3.

III.1.2.1. LENGUAJE DE NAVEGACIÓN NSEQL

NSEQL (*Navigation SEQuence Language*) [PRAHV02], es un lenguaje que permite crear declarativamente secuencias de navegación utilizando una representación de alto nivel basada en eventos. El entorno de ejecución de NSEQL está compuesto por un navegador web automatizado, más ligero que los navegadores convencionales. Ese navegador puede construirse sobre las APIs de programación que proporcionan los navegadores más populares, sobre sus controles de navegación. Existen implementaciones de NSEQL tanto para Microsoft Internet Explorer [MSIE07] como sobre Mozilla Firefox [MF07].

A través de NSEQL, es posible reproducir desde un programa de ordenador, cualquier secuencia de operaciones que un usuario humano pueda realizar a través de su navegador. El concepto fundamental de NSEQL es el de secuencia de navegación. Una secuencia de navegación se compone de una lista de comandos que se ejecutan consecutivamente sobre la interfaz de un navegador de Internet. NSEQL trabaja a nivel de navegador en lugar de a nivel HTTP. Esto hace posible que no sea necesario tratar problemas relacionados con interpretación de *JavaScript*, redirecciones o identificadores de sesión. Todos estos problemas son tratados de forma transparente por el navegador.

La sintaxis básica para especificar una secuencia de navegación es la siguiente:

$$<comando_1>; <comando_2>; \ldots <comando_m>$$

Cada comando tiene un nombre y puede recibir una lista de parámetros. El formato de cada comando es el siguiente:

$$NombreComando\ (<parámetro_1>; <parámetro_2>; \ldots <parámetro_n>)$$

A continuación, se explica mediante un ejemplo, cómo se especifica la ejecución de una búsqueda en un formulario HTML usando una secuencia de navegación NSEQL. Se ha seleccionado un ejemplo que incluye elementos dinámicos, para ilustrar cómo se gestionarían con el modelo de navegación propuesto. Se comenta cada comando ejecutado, así como el cambio que produce dicho comando en el componente de navegación. El ejemplo reproduce una búsqueda por título (‘Thinking in java’) y autor (‘Bruce Eckel’) en el formulario de búsqueda avanzada del sitio web Amazon[19]. La secuencia de navegación NSEQL para realizar dicha búsqueda es la que se muestra en la Figura 25. Se ha numerado cada una de las sentencias para mostrar en la Figura 26 cómo se van ejecutando sobre el navegador.

El comando `Navigate(url)` hace que el navegador se dirija al URL que recibe como parámetro. El efecto es el mismo que si un usuario humano hubiese tecleado el URL en la barra de direcciones del navegador y pulsado ENTER. El navegador responderá a la orden

[19] http://www.amazon.com

conectándose al URL solicitado y construyendo el árbol DOM de la página obtenida, que es la representación interna que el navegador utiliza para todos los componentes de la página.

El comando ExtendedWaitPages(n) espera a que se carguen n páginas en el navegador. Tras acceder a un URL normalmente se carga una única página pero hay situaciones en las que desde esa página que se carga hay redirecciones automáticas a otra página. Éste es el motivo por el cual en algunas ocasiones puede ser necesario esperar a que se cargue más de una página. Cuando se especifica como valor -1, el navegador detecta de forma automática cuándo ha finalizado la navegación para continuar ejecutando el resto de comandos de la secuencia.

```
(01) Navigate(http://www.amazon.com,0);
(02) ExtendedWaitPages(-1);
(03) FindElementByText(A,See all 41 Product Categories,0,true);
(04) FireEventOnSelectedElement(onmouseover)
(05) ClickOnAnchorByText(Books,0,true);
(06) ExtendedWaitPages(-1);
(07) ClickOnAnchorByText(Advanced Search,0,true);
(08) ExtendedWaitPages(-1);
(09) FindFormByAction(/gp/search/ref=sr_adv_b/,0,true);
(10) SetInputValue(author,0,bruce eckel);
(11) SetInputValue(title,0,thinking in java);
(12) FindElementByAttribute(INPUT,NAME,mysubmitbutton1,0,true);
(13) ClickOnElement();
(14) ExtendedWaitPages(-1);
```

Figura 25 Secuencia de navegación para búsqueda en un formulario HTML

El comando FindElementByText(tipo, texto, n, igual) busca el n-ésimo elemento del tipo especificado, cuyo texto coincida con el indicado (considerando que el valor 0 identifica el primer elemento de ese tipo). El parámetro igual es un valor lógico que indica si la búsqueda del enlace se hace por igualdad o por contención del parámetro texto en el texto del enlace. El comando FireEventOnSelectedElement(evento) dispara el evento especificado sobre el elemento seleccionado y como resultado se despliega una nueva capa con más enlaces (uno de ellos nos llevará al enlace de búsqueda de libros). Este comando produce el mismo efecto que si el usuario hubiese generado ese evento sobre el elemento. En nuestro caso, un evento 'onmouseover' sobre el enlace correspondiente.

El comando ClickOnAnchorByText (texto, n, igual) busca el n-ésimo hiperenlace de la página cuyo texto coincida con el indicado (considerando que el valor 0 identifica el primer enlace) y genera un evento de clic sobre él, provocando que el navegador acceda a la página destino. NSEQL permite la utilización de otras formas de localización de enlaces, como por ejemplo, por el valor del atributo href). De esta forma, tras hacer clic sobre dos enlaces, el navegador se posiciona en la página que contiene el formulario de búsqueda avanzada.

El comando FindFormByAction(url, n, igual) busca el n-ésimo formulario HTML en el árbol DOM de la página actual que tenga como valor del atributo action el especificado en el parámetro url, y lo establece como el formulario actual seleccionado (nuevamente, hay otras formas disponibles para localizar un formulario, por ejemplo por el valor de su atributo name).

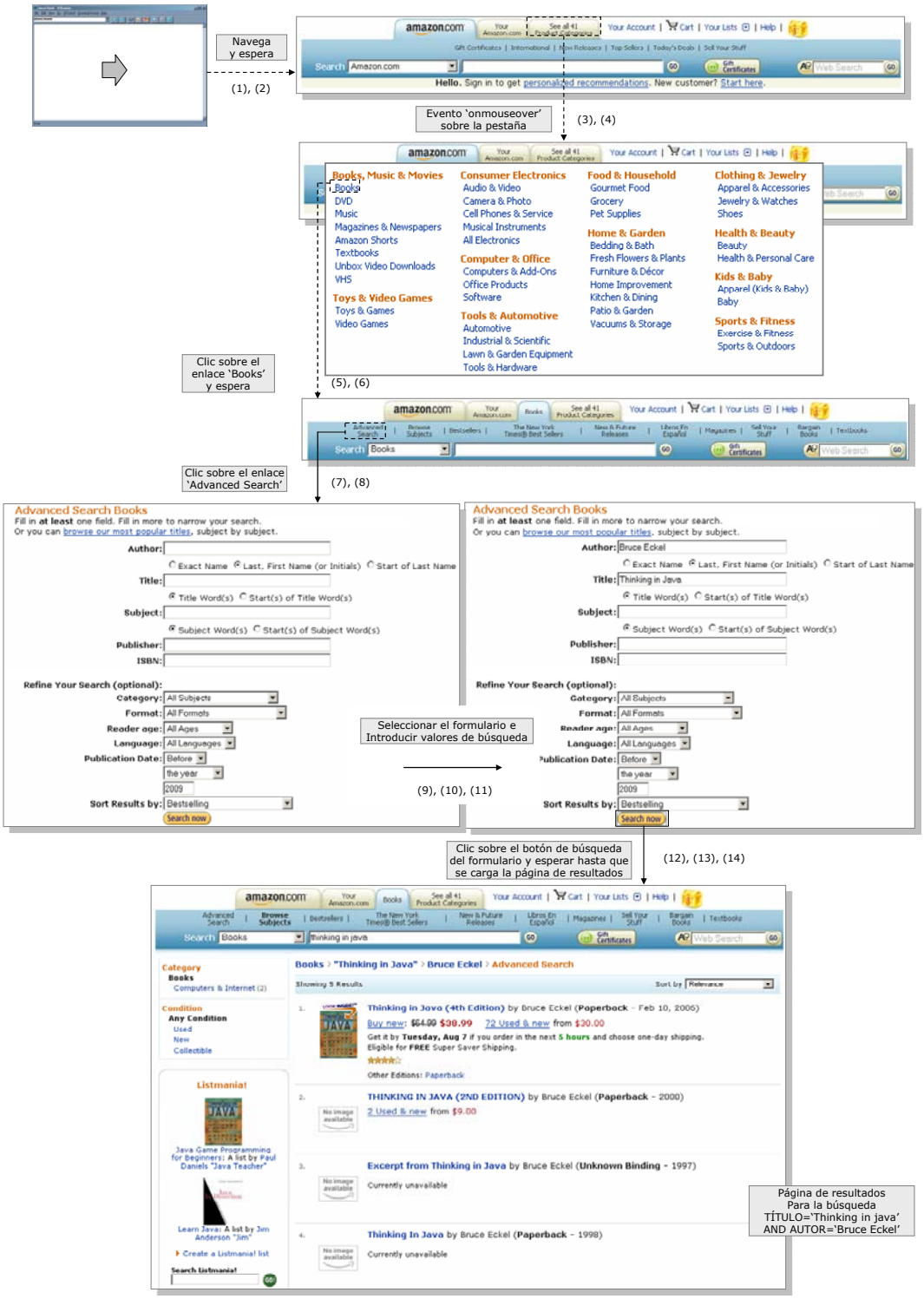


Figura 26 Ejecución de la secuencia de navegación de la Figura 25

A continuación se asignan los valores deseados a los campos del formulario. Para ello se utiliza el comando SetInputValue(nombre, n, valor), que asigna valor al n-ésimo campo del formulario de tipo input cuyo nombre es el especificado como primer parámetro. El parámetro valor es una cadena constante. En este caso se rellena el campo título con el valor 'Thinking in java' y el campo autor con el valor 'Bruce Eckel'.

El comando FindElementByAttribute(tipo, nombre, valor, n, igual) localiza el n-ésimo elemento del formulario seleccionado en ese momento, que tenga como tipo el especificado en el parámetro tipo y como valor del atributo indicado en nombre el especificado en valor. El comando ClickOnElement() provoca un clic sobre el elemento seleccionado. En el ejemplo se utilizan para hacer la invocación del formulario, y obtener la página de resultados de búsqueda.

Como resultado, se obtiene la página de respuesta con los datos de la búsqueda. Aunque no ocurre en este ejemplo, si la página de respuesta estuviese compuesta por varios marcos, podría seleccionarse uno de ellos con el comando SelectFrame, que permite establecer un marco para ejecutar el resto de comandos, por nombre, URL u orden en la página. NSEQL también incluye comandos para tratar otras características avanzadas como ventanas o diálogos emergentes.

III.1.3. MODELO DE DOMINIO DE APLICACIÓN

En este apartado definimos el modelo de dominio de aplicación que será utilizado por las técnicas que se van a describir en los apartados siguientes.

Un dominio de aplicación define la metainformación para una tarea de extracción de información estructurada de la Web. Por una parte, es el responsable de especificar la temática para la dirección de exploración del *crawler*, a nivel de la Web de Superficie, pero también es el referente para determinar si los formularios de consulta web localizados, que constituyen los puntos de entrada a la Web Oculta del lado servidor, son relevantes o no para la tarea objetivo. Además, los registros estructurados obtenidos de las páginas de resultados deben de ser etiquetados conforme a los elementos del dominio. Por último, el dominio también define los campos que tendrán los formularios que permitirán la realización de consultas estructuradas sobre la información recolectada por el sistema.

Un dominio de aplicación en este escenario puede definirse como una ontología, que contendrá al menos los siguientes elementos:

- El esquema con los atributos de las entidades que representan los elementos del dominio, con su nombre o conjunto de nombres habituales y el tipo al que pertenece cada uno de ellos (cadena, numérico, fecha, moneda, enumeración, etc.). Por ejemplo, si se desea recolectar información sobre libros, el esquema incorporará atributos tales como TÍTULO, AUTOR, etc.
- Relaciones existentes entre los atributos. Por ejemplo, los atributos DÍA, MES y AÑO conjuntamente representan una fecha.

- Un conjunto de consultas expresadas sobre el esquema del dominio, representativas para el objetivo de la tarea de extracción de datos. Estas consultas serán ejecutadas sobre los formularios relevantes descubiertos durante el proceso de *crawling*.
- Un conjunto de documentos representativos de ese dominio de aplicación. Típicamente, estos documentos se utilizarán para entrenar los clasificadores necesarios para las tareas de *crawling* dirigido en la Web de Superficie.
- Información adicional que permita decidir si un formulario que posea determinados atributos del dominio de aplicación, puede considerarse o no relevante para el mismo.

En el prototipo implementado, se ha optado por una definición de dominio de aplicación simplificada que contiene solamente los elementos necesarios para validar las contribuciones de esta tesis doctoral. Más concretamente, la definición utilizada está compuesta por:

- Un conjunto de atributos $A=\{a_1, a_2, ..., a_n\}$. Cada atributo a_i tiene asociado
  - Un nombre,
  - Un conjunto de alias $\{a_i\_alias_1, ..., a_i\_alias_k\}$, y
  - Un índice de especificidad, e_i.
- Un conjunto de consultas $C=\{c_1, c_2, ..., c_m\}$ que van a ser ejecutadas sobre los formularios descubiertos que sean relevantes para esta definición de dominio. Cada consulta c_j es una lista de pares *(atributo, valor)*, donde *atributo* es un atributo del dominio y *valor* es una cadena de caracteres, que puede ser vacía.
- Un umbral de relevancia denotado como μ.

Un atributo representa un campo que pueda aparecer en los formularios de consulta que son relevantes para una tarea de extracción de datos determinada, y que será utilizado para etiquetar los registros de datos obtenidos como resultado de consultas sobre los formularios considerados relevantes.

Los alias representan etiquetas o nombres alternativos que pueden identificar el atributo en un formulario de consulta. Por ejemplo, el atributo AUTOR, de un dominio utilizado para recuperar información relativa a libros de tiendas de comercio electrónico, podría tener como alias 'author', 'writer' o 'writen by'. Es importante destacar que el estudio descrito en [CHLP+04] concluyó que el vocabulario utilizado para definir el esquema de los formularios web de un mismo dominio tiende a converger en un tamaño relativamente pequeño. De forma adicional detectaron que la frecuencia de los atributos seguía una distribución Zipf, es decir, que un conjunto reducido de atributos son mucho más frecuentes en los formularios de un dominio que el resto. Esta afirmación permite concluir que es posible crear especificaciones de dominio efectivas de forma sencilla y rápida, siendo suficiente con explorar un conjunto reducido de fuentes en un dominio para encontrar los atributos y alias más relevantes para el mismo.

Para cada atributo, el dominio además incluye un índice de especificidad. El índice de especificidad (denotado por e_i) de un atributo a_i es un número entre 0 y 1 indicando la probabilidad de que un formulario de consulta que contenga ese atributo, sea realmente relevante para el dominio. Por ejemplo, en un dominio creado para obtener datos de libros, podríamos tener los siguientes índices de especificidad:

- El atributo ISBN tendría un valor alto (e.g. 0.95), puesto que un formulario que permita consultas por el atributo ISBN es casi con total seguridad un formulario perteneciente al dominio de libros. El ISBN es una propiedad característica y única de los libros, que no suele estar presente en otros dominios.
- El atributo PRECIO tendría un valor bajo (e.g. 0.05), debido a que un formulario que permite la realización de consultas por el atributo PRECIO podría ser un formulario de consulta que permite buscar cualquier clase de productos, no sólo libros.

Los índices de especificidad permitirán obtener la relevancia de un formulario para un dominio, a partir de los atributos de un dominio identificados en un formulario. Si el atributo identificado tiene un índice de especificidad muy alto, como por ejemplo el ISBN en el caso de tiendas electrónicas de libros, puede no ser necesario ningún otro atributo para reconocer el formulario como relevante para el dominio de aplicación. En cambio, en el caso de otros atributos como TÍTULO o PRECIO que pueden aparecer en multitud de dominios de aplicación (por ejemplo tiendas electrónicas de música o películas, además de libros), su identificación en un formulario no es suficiente para determinar su dominio. En esos casos, es necesario considerar la co-ocurrencia de diferentes atributos, como TÍTULO y AUTOR.

Dominio de Aplicación
'Libros'

Atributos: $A = \{a_1, a_2, a_3, a_4, a_5\}$

Atributo	Nombre	Alias	e_i (índice de especificidad)
a_1	TÍTULO	*'title', 'entitle', 'title of the book'*	0.6
a_2	AUTOR	*'author', 'writer', 'written by'*	0.7
a_3	ISBN		0.95
a_4	FORMATO	*'format', 'binding type'*	0.25
a_5	PRECIO	*'price', 'cost of book'*	0.05

Consultas: $C = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$

C_1 = { (TÍTULO, '*java*'), (FORMATO, 'hardcover') }
C_2 = { (TÍTULO, 'thinking in java'), (AUTOR, '*Bruce Eckel*') }
C_3 = { (TÍTULO, '*j2ee*') }
C_4 = { (TÍTULO, '*concurrent programming*') }
C_5 = { (TÍTULO, '*ejb3*') }
C_6 = { (TÍTULO, '*java server faces*') }
C_7 = { (AUTOR, '*Herbert Schildt* ') }
C_8 = { (TÍTULO, '*web services*') }

Umbral de relevancia: $\mu = 0.9$

Figura 27 Definición de dominio de aplicación para obtener datos de libros

Finalmente, la metainformación del dominio de aplicación también incluye un umbral de relevancia μ. Los índices de especificidad y el umbral serán utilizados para determinar si un formulario es relevante para un dominio, tal y como se comentará en la sección III.4.3.

La Figura 27 muestra un ejemplo de definición de dominio para la tarea de obtener páginas con datos relativos a libros de la temática de 'Java and XML programming'. El umbral de relevancia para este dominio se ha seleccionado a 0.9.

III.2. ARQUITECTURA PARA SISTEMAS DE *CRAWLING* DIRIGIDO

En esta sección se describe la arquitectura propuesta para sistemas de *crawling* dirigido para la Web Oculta. El sistema debe ser capaz de realizar un recorrido de la Web para localizar formularios de consulta, que constituyen frontales de acceso a bases de datos que contienen información estructurada, relevante para el dominio de aplicación.

El recorrido presenta determinadas problemáticas, debidas principalmente a las tecnologías que emplean en la parte cliente los navegadores web para hacer más amigable los frontales de los sitios web a los usuarios (ver apartado II.3.3.2). Una vez localizado un formulario, el sistema debe analizarlo para ser capaz de realizar consultas sobre él y tener acceso al contenido que almacena. Además, como ya se ha comentado, estos frontales de consulta proporcionan normalmente acceso a información estructurada contenida en bases de datos. Por lo tanto, el sistema de *crawling* debe ser capaz de inferir la estructura de la información de las páginas de resultados, para permitir a los usuarios finales la realización de consultas más precisas y por lo tanto de mayor calidad.

En la sección III.1 se han introducido los elementos básicos con los que tiene que tratar el sistema, y que condicionan su arquitectura. Las principales características del sistema son las siguientes:

- El modelo de páginas dinámicas y el modelo de navegación requerido para procesarlas, hace que el sistema tenga que tratar con *rutas* en lugar de simplemente con URLs. Esto afecta principalmente a los procesos de *crawling,* que no trabajan a nivel HTTP, sino que están basados en componentes de más alto nivel: mini-navegadores construidos utilizando APIs de navegadores estándares (Microsoft Internet Explorer [MSIE07] o Mozilla Firefox [MF07]). En la sección III.3 se describen las técnicas definidas para procesar de forma correcta la navegación en páginas dinámicas y permitir acceder a la información 'oculta' del lado cliente de la Web.
- El proceso de *crawling* está dirigido por un dominio (o conjunto de dominios) de aplicación. Para mantener la dirección de exploración y localizar formularios relevantes, se parte de la idea propuesta por Barbosa y Freire en [BF07] para un *crawler* de formularios adaptativo (ver apartado II.3.3.3), pero complementándolo con las técnicas de modelado y ejecución de consultas predefinidas sobre formularios relevantes descritas en la sección III.4.
- Las páginas de resultados de consultas realizadas de forma automática sobre formularios relevantes para el/los dominio(s) de aplicación, se estructuran y etiquetan para obtener registros de datos que permitan la realización de búsquedas estructuradas. En la sección III.5 se describe el componente de estructuración automática propuesto.

La arquitectura definida se divide en tres módulos principales: El módulo de *crawling*, responsable de recopilar la información web, el módulo de indexación, encargado de almacenar la información recolectada por el módulo anterior de forma adecuada para que el tercer módulo, el módulo de búsqueda, pueda permitir la realización de búsquedas y consultas. En la Figura 28 aparecen los tres módulos, junto con sus componentes básicos.

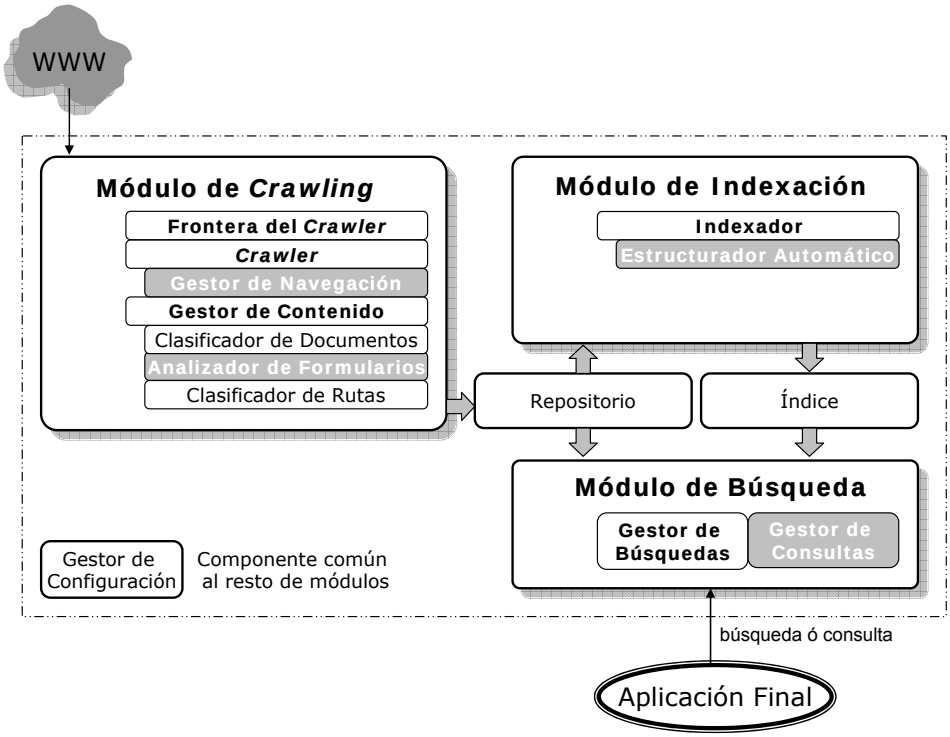


Figura 28 Arquitectura para Web Oculta – *crawling* / indexación / búsqueda

El componente Gestor de Configuración es común a todos los módulos, y contiene la información de arranque del *crawler* y la definición de los dominios de aplicación. Existen otros componentes que sirven de nexo de unión entre los diferentes módulos, como son el componente Repositorio y el componente Índice.

En tiempo de ejecución, distinguimos tres fases, cada una correspondiente a uno de los módulos principales. Las dos primeras se corresponden con la etapa de obtención de información del sistema y la última con la de ejecución de consultas sobre el mismo.

En la primera fase, el módulo de *crawling* recorre la Web guiado por los dominios de aplicación, recopilando páginas de la Web de Superficie relevantes y, además, localizando formularios de consulta web y realizando consultas sobre los relevantes para obtener nuevas páginas. En esta fase se asocia una ruta conteniendo una secuencia de navegación de alto nivel como las descritas en el apartado III.1.2, con cada nuevo recurso web obtenido. Esto permite reproducir el acceso a ese recurso en el futuro. Además, se almacenan los documentos y metainformación obtenidas en el componente Repositorio.

En la segunda fase, el módulo de indexación parte de la información almacenada en el componente Repositorio por el módulo de *crawling*, para realizar la indexación de los documentos y estructurar aquellos que son resultado de consultas sobre formularios web, en el componente Índice.

En la tercera fase, el módulo de búsqueda permite la realización de consultas sobre la información almacenada en el componente Índice.

El motivo para dividir la etapa de obtención de información en dos fases es la lentitud de los procesos de indexación, acentuada por los procesos de estructuración automática, que consumen un tiempo de CPU considerablemente mayor que los procesos de *crawling*, caracterizados por mayor consumo de entrada/salida. De todas formas, la arquitectura permite también realizar comunicaciones entre los módulos en cualquier momento, no sólo al finalizar la ejecución de la fase de *crawling*. Esto es porque aunque un *crawling* dirigido de la Web Oculta no suele llevar tanto tiempo como un *crawling* global, puede interesar crear un índice rápidamente para permitir la realización de consultas.

A continuación se describe la operación normal de los distintos componentes de la arquitectura.

Cuando el motor de *crawling* comienza su ejecución, lee su configuración del componente Gestor de Configuración. El siguiente paso consiste en inicializar la Frontera con la lista de sitios iniciales para el *crawling*, así como la inicialización del conjunto de procesos de *crawling*.

El componente Frontera es el responsable de mantener la lista global de rutas a ser accedidas, y que es compartida por todos los procesos de *crawling*.

Una vez se han inicializado todos los procesos de *crawling*, cada uno de ellos obtiene una ruta de la Frontera. Es importante destacar que cada proceso de *crawling* puede ser ejecutado local o remotamente al servidor, permitiendo *crawling* distribuido. Además, cada proceso de *crawling* utiliza un gestor de navegación de alto nivel, que es capaz de ejecutar secuencias NSEQL.

Una vez obtenida una ruta, cada proceso de *crawling* carga el objeto de sesión asociado a la ruta y descarga el documento asociado (utiliza el gestor de navegación para seleccionar el manejador adecuado para el documento, como PDF, MS Word, etc.). Si la sesión ha expirado, el proceso de *crawling* utiliza el programa NSEQL para acceder al documento de nuevo.

Como salida del componente de crawling se obtiene el documento modelado como se ha comentado en III.1.1. El componente Gestor de Contenido analiza el documento aplicando una cadena de filtros para decidir si el documento puede ser considerado relevante y, si es el caso, si debería ser almacenado o indexado.

En particular, un clasificador de documentos se encarga de determinar si el documento es relevante para alguno de los dominios de aplicación considerados, el analizador de formularios determina si contiene formularios relevantes, para obtener nuevas rutas a partir de consultas sobre esos formularios y el clasificador de rutas selecciona y prioriza las rutas que deben de ser añadidas a la frontera para ser accedidas en el futuro. Adicionalmente,

también posee filtros para almacenar documentos con su metainformación, y los formularios relevantes en el componente Repositorio.

Resaltar que las páginas resultado de ejecutar consultas contra formularios web se procesan de la misma forma que el resto de páginas, permitiendo el recorrido del *crawler* por las páginas dinámicas.

El proceso de *crawling* finaliza cuando no quedan rutas en la Frontera.

A partir de la información contenida en el componente Repositorio, el módulo de indexación recorre los diferentes documentos determinando cuáles han sido obtenidos a partir de consultas sobre formularios. En todos los casos se indexan esos contenidos en el componente Índice, y en el caso de páginas de resultado de consulta en formularios, el componente de estructuración automática obtiene los diferentes registros de datos contenidos en la página y los almacena en un índice multicampo para permitir la realización de consultas estructuradas sobre él.

Por último, el componente de búsqueda permite la realización de búsquedas sobre el índice de documentos, o de consultas por diferentes atributos presentes en los dominios definidos, sobre los registros de datos que han sido identificados en las diferentes páginas de resultados.

Los apartados III.2.1, III.2.2 y III.2.3 describen en detalle cada uno de los módulos de la arquitectura.

III.2.1. MÓDULO DE CRAWLING

En la Figura 29 se muestra la arquitectura del módulo responsable de recopilar la información de la Web Oculta, tanto del lado cliente como del lado servidor.

El componente *Gestor de Configuración* es el responsable de almacenar todos los parámetros requeridos por todos los componentes del sistema. En particular, el módulo de *crawling* requiere la siguiente metainformación:

- Una lista de rutas iniciales, identificando los sitios de partida del proceso de *crawling*. Destacar que una ruta inicial, al expresarse como una secuencia de comandos de alto nivel, puede hacer que un proceso de *crawling* se inicie en una zona de la Web que requiere autenticación para poder acceder a ella.
- La profundidad de *crawling* deseada para cada una de las rutas iniciales.
- La configuración de los gestores de navegación para los diferentes tipos de documentos tratados por el sistema, como pueden ser conversores de PDF, documentos Office, o RSS, entre otros.
- La configuración de los diferentes filtros del Gestor de Contenido, como las listas de expresiones regulares representando los nombres DNS (*Domain Name System*) a ser incluidos y excluidos del *crawling*.

– La definición de los dominios de aplicación que guían el proceso de *crawling*, como hemos descrito en el apartado III.1.3.

El componente *Repositorio* es el responsable de almacenar de forma persistente tanto los documentos descargados como su metainformación (las rutas que los identifican y que proporcionan la información necesaria para un acceso futuro a los mismos). También almacena la metainformación extraída a partir de los formularios relevantes.

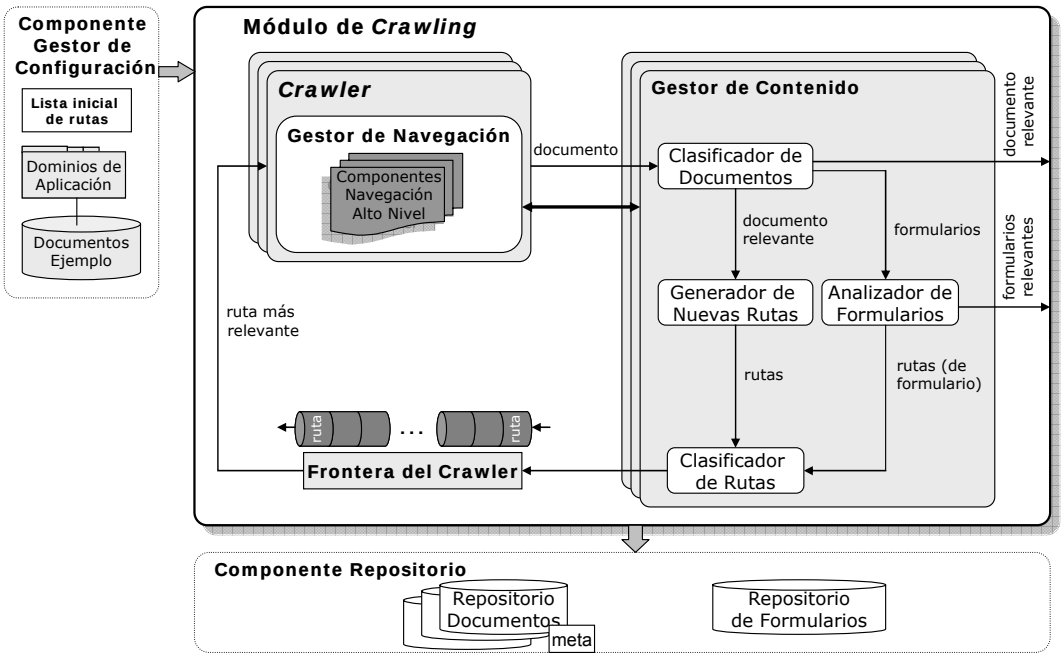


Figura 29 Arquitectura del módulo de *crawling*

El componente *Frontera del Crawler* almacena la cola de prioridades de rutas a ser accedidas, ordenadas por relevancia respecto al dominio de aplicación considerado.

El componente *Crawler* es el responsable de obtener las rutas del componente *Frontera* para acceder a ellas y recuperar los documentos asociados. La salida de este componente es un documento web junto con su modelo dinámico base (el comentado en el apartado III.1.1). Para ello, hace uso del componente *Gestor de Navegación*, que es el responsable de interpretar las rutas de alto nivel para obtener los recursos a los que apuntan. El componente *Gestor de Navegación* tiene como responsabilidad acceder e interpretar páginas generadas con tecnologías del lado cliente, pero también proporciona componentes utilizados por los filtros de generación de nuevas rutas de documentos dinámicos y formularios, como se comenta en la sección III.3.

Una vez obtenido un documento, el componente *Gestor de Contenido* es el encargado de almacenarlo en el componente *Repositorio* y generar nuevas rutas a añadir al componente *Frontera*, en el caso de que determine que el documento es relevante para los dominios que guían el proceso completo.

El componente *Gestor de Contenido* posee una serie de filtros encargados de diferentes aspectos, como el filtro de clasificación de documentos, el filtro de obtención de nuevas rutas, el filtro clasificador de rutas o el filtro de análisis de formularios.

El filtro de clasificación de documentos es el responsable de decidir si un documento recuperado pertenece a alguno de los dominios de aplicación tratados o aún no siéndolo, si es un documento prometedor para alcanzar documentos relevantes. Como complemento de este filtro, el filtro de clasificación de rutas permite seleccionar, del conjunto de rutas obtenidas a partir de un documento considerado relevante, aquellas que tienen más probabilidad de ser importantes para el *crawling*. En el modelo de dominio de aplicación se ha considerado un conjunto de documentos relevantes, cuya finalidad es entrenar los clasificadores aquí utilizados. Barbosa y Freire presentan en [BF07] un sistema de *crawling* de formularios web que define una posible implementación de estos clasificadores, proponiendo una forma alternativa que evita el proceso previo de clasificación de formularios, permitiendo que se vayan configurando en tiempo de ejecución del proceso de *crawling* (ver apartado II.3.3.3 para una descripción más detallada de este sistema).

En nuestro prototipo, se ha implementado una aproximación más sencilla para solucionar este problema, que se basa en una lista de palabras clave que tienen que aparecer en los documentos analizados y en una lista de expresiones regulares que deben cumplir las rutas contenidas en ellos para ser priorizadas en la *Frontera*.

Existe también otro filtro responsable del almacenamiento de los documentos y su metainformación asociada en el componente *Repositorio*.

El filtro de generación de nuevas rutas selecciona todos los enlaces en la página y genera una nueva ruta por cada uno. En el contexto de páginas dinámicas, para obtener todas las rutas posibles desde un documento es necesario delegar en el *Gestor de Navegación* esa tarea. En la sección III.3 se describe este componente en detalle.

El filtro de análisis de formularios analiza cada página relevante que contenga formularios, para determinar si es relevante para alguna de las definiciones de dominios especificadas. En caso de que un formulario sea considerado relevante, se añade una nueva ruta para cada consulta especificada en la definición del dominio. En la sección III.4 se describe el componente de acceso a la Web Oculta del lado servidor en detalle.

III.2.2. MÓDULO DE INDEXACIÓN

En la Figura 30 se muestra la arquitectura del módulo responsable de generar las representaciones internas que crearán el soporte para que el siguiente módulo pueda realizar búsquedas.

Este módulo comparte los componentes *Gestor de Configuración* y *Repositorio* con el módulo de *crawling*. Obtiene del componente *Gestor de Configuración* información relacionada con los dominios de aplicación definidos en el proceso de *crawling*, que permite interpretar la metainformación presente en las rutas a los documentos procesados.

Para todos los documentos obtenidos por el módulo de *crawling*, el componente *Indexador de Documentos* los añade al índice de documentos del componente *Índice*. Junto con el contenido del documento, añade campos adicionales en el índice para especificar su ruta remota y su ruta en el repositorio local de documentos.

Para aquellos documentos que sean resultado de búsquedas en formularios – detectables porque poseen metainformación adicional en la ruta especificando la consulta realizada sobre el formulario – el sistema los envía al componente de *Estructuración Automática*. Este componente se describe en detalle en la sección III.5, y es el encargado de obtener la lista de registros etiquetados, embebidos en la página de resultados de consultas en formularios. El componente *Indexador de Registros* es el responsable de almacenar cada registro en un índice diferente del componente *Índice*, en función del dominio de aplicación al que pertenezcan. Un documento del índice de un dominio tendrá tantos campos como atributos hay definidos en el dominio. Adicionalmente poseerá campos adicionales para identificar la página del formulario y la secuencia de navegación a reproducir para alcanzar ese documento.

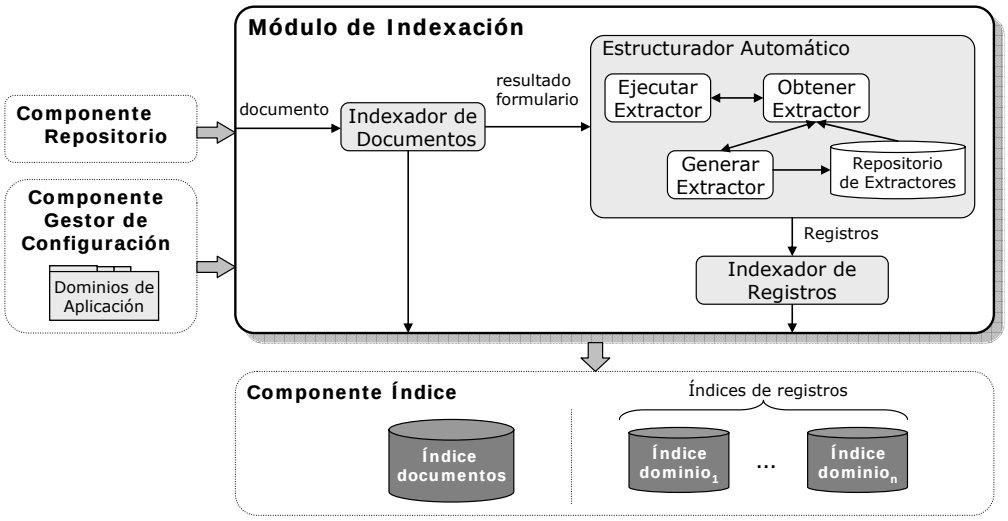


Figura 30 Arquitectura del módulo de indexación

Para optimizar el proceso de estructuración automática, la primera vez que se analiza una página de resultado de búsqueda en un formulario, se genera un programa extractor que será utilizado para obtener los registros del resto de las páginas de resultados para el mismo formulario. En nuestro prototipo, el programa extractor es un programa envoltorio generado de forma automática a partir de las tuplas de ejemplo obtenidas de la primera página de resultados analizada para cada formulario web. La implementación actual utiliza el sistema de generación de *wrappers* de Raposo et al. [RPB07] [RPAH07], basado en técnicas de inducción, que genera una especificación en un lenguaje que permite definir reglas de extracción de datos sobre documentos de texto semi-estructurado, denominado DEXTL (*Data EXTraction Language*) [PRAH+02].

Por último, el componente *Índice* contiene las diferentes estructuras de datos necesarias para que el módulo de búsqueda permita la realización de búsquedas.

III.2.3. MÓDULO DE BÚSQUEDA

En la Figura 31 se muestra la arquitectura del módulo que permite la realización de consultas contra el componente *Índice* de documentos.

Este componente permite la realización de diferentes tipos de consultas. Por una parte, permite la realización de búsquedas por palabra clave sobre los documentos indexados. Para ello se utilizan técnicas convencionales de recuperación de información no estructurada. Por ejemplo, en el contexto del dominio de aplicación de Libros, podría realizarse una consulta por palabra clave 'Thinking in Java', que devolvería como resultado los documentos relevantes para esos términos, de todos los obtenidos por el *crawler* durante su fase de recorrido de la Web.

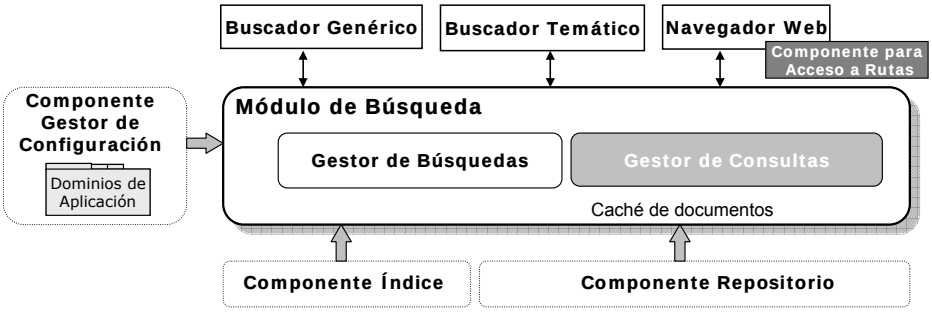


Figura 31 Arquitectura del módulo de búsqueda

Por otra parte, este componente permite también la realización de consultas precisas, sobre la información estructurada como resultado del proceso global. Este tipo de consultas permiten obtener resultados de mucha más calidad, debido a la naturaleza estructurada de la información base. Por ejemplo, considerando el mismo dominio de aplicación, el de Libros, y en particular la colección de registros de libros estructurados, permitiría resolver la consulta TÍTULO igualA 'Thinking in Java' AND AUTOR igualA 'Bruce Eckel', que devolvería todos aquellos registros de libros con el título y autor especificados.

El componente *Gestor de Configuración* proporciona los atributos de los dominios de aplicación, que definen los campos del índice por los que se pueden realizar las consultas.

Como resultado de una búsqueda sobre uno cualquiera de los índices, el sistema proporciona información de la ruta de acceso al documento. Si algún documento de la página de resultados no puede ser accedido directamente utilizando su URL por problemas relacionados con sesiones, se utilizará la secuencia NSEQL asociada a la ruta para reproducir el acceso al recurso.

En nuestro prototipo se ha implementado un componente AxtiveX de navegación de alto nivel para el navegador Microsoft Internet Explorer. Este componente ActiveX recibe como parámetro un programa NSEQL, que representa la ruta al documento. Una vez descargado por el navegador, el componente ejecuta la secuencia de navegación sobre el navegador, permitiendo reproducir los pasos seguidos durante el proceso de *crawling* para alcanzar el documento.

III.3. WEB OCULTA DEL LADO CLIENTE

En el apartado II.3.3.2 se han introducido las problemáticas que plantean las tecnologías del lado cliente a los sistemas de *crawling* convencionales. En esta sección se describen las técnicas propuestas en esta tesis doctoral para tratar con las dificultades que presenta el acceso a la parte oculta de la Web del lado cliente. Como ya se ha adelantado en apartados previos, la idea principal se basa en la utilización de componentes de *crawling* de alto nivel que permitan tratar con las tecnologías del lado cliente de una forma transparente para el sistema y, por otra parte, que permita utilizar un modelo de navegación especial para tratar páginas dinámicas que incluyan complejidades como con menús emergentes, etc.

El *Componente de Gestión de Navegación* de la arquitectura propuesta es el encargado de implementar las diferentes estrategias y algoritmos definidos para abordar el tratamiento de las tecnologías del lado cliente. Las principales características de este componente ya se han introducido previamente:

- Para solucionar el problema de mantenimiento de sesiones, se utiliza el concepto de *ruta* a un documento, que puede ser vista como una generalización del URL. Una ruta se compone de un URL, un objeto sesión conteniendo el contexto necesario para poder obtener el recurso al que apunta el URL y un programa NSEQL para acceder al documento cuando la sesión utilizada para obtener el documento ha expirado (ver apartado III.3.1).
- Los procesos de *crawling* utilizados en la arquitectura propuesta no se basan en clientes HTTP. En su lugar, están basados en 'mini navegadores web' automatizados, construidos utilizando APIs de navegadores estándar. Esto posibilita que el sistema pueda tratar con la ejecución de código de *script*, gestionar redirecciones, etc. (ver apartado III.3.2).
- Para tratar con menús emergentes y otros elementos dinámicos que pueden generar nuevos enlaces en la página actual, es necesario implementar algoritmos especiales para gestionar el proceso de generación de nuevas 'rutas a explorar' desde una página web (ver apartado III.3.3).

III.3.1. RUTAS PARA IDENTIFICAR RECURSOS WEB

En los sistemas de *crawling* convencional, las rutas o localizadores de recursos web, son simplemente URLs. Esto presenta los problemas con mecanismos de mantenimiento de sesión mencionados en el apartado II.3.3.2.2. En el modelo propuesto en esta tesis doctoral consideramos que una ruta está compuesta de tres elementos:

- Un URL convencional que apunta a un documento. En las rutas de la lista inicial de rutas en la frontera del *crawler*, este elemento también puede ser un programa NSEQL. Esto es útil para iniciar el *crawling* en un documento que no es

directamente accesible a través de un URL (por ejemplo, es el caso típico de sitios web que requieren autenticación).

- Un objeto sesión conteniendo toda la información requerida (*cookies*, etc.) para recuperar el entorno de ejecución que el proceso de *crawling* tenía en el momento de añadir la ruta a la frontera.
- Un programa NSEQL representando la secuencia de navegación seguida por el sistema para alcanzar el documento.

El segundo y tercer elementos son calculados por el sistema para cada ruta. El segundo elemento permite a un proceso de *crawling* acceder a un URL añadido por otro proceso (incluso si el proceso de *crawling* original se estaba ejecutando en otra máquina). El tercer elemento se utiliza para acceder al documento apuntado por la ruta, cuando la sesión originalmente utilizada para obtener el documento ha expirado. Esto es útil para permitir un acceso posterior a los documentos recolectados.

III.3.2. MINI-NAVEGADORES COMO PROCESOS DE *CRAWLING*

Los sistemas de *crawling* convencionales implementan los procesos de *crawling* utilizando clientes HTTP. En cambio, en las técnicas propuestas en este trabajo, los procesos de *crawling* están basados en 'mini-navegadores web' automatizados, construidos utilizando APIs estándar de navegadores y que son capaces de ejecutar programas NSEQL. Actualmente se dispone de implementaciones para los principales navegadores web: una basada en el componente de navegación web de Microsoft Internet Explorer [MSIE07] y la otra en Mozilla Firefox [MF07]. Esto permite que el sistema sea capaz de realizar de forma transparente, entre otras, las siguientes operaciones:

- Acceder al contenido dinámicamente generado a través de lenguajes de *script* (e.g. métodos `document.write` de *JavaScript*).
- Evaluar el código de *script* asociado con enlaces y formularios, de tal forma que se puedan obtener los URLs reales a los que apuntan esos elementos.
- Tratar con redirecciones del lado cliente. Después de ejecutar las nuevas navegaciones, el mini-navegador espera hasta que todos los eventos de navegación de la página actual hayan finalizado, antes de generar la siguiente navegación.
- Proporcionar un entorno de ejecución para tecnologías como Applets Java y código Flash. Aunque los mini-navegadores no pueden acceder al contenido mostrado por estos componentes compilados, pueden tratar con la frecuente situación en la que estos componentes se utilizan como introducción gráfica, que finalmente redirige el navegador a una página web convencional.

III.3.3. ALGORITMO PARA GENERAR NUEVAS RUTAS

Esta sección describe el algoritmo utilizado para generar nuevas rutas a partir de una página HTML. Este algoritmo trata con las dificultades asociadas a elementos HTML controlados por lenguajes de *script*.

En general, para obtener nuevas rutas a partir de un documento HTML, es necesario analizar la página buscando enlaces y elementos que tengan asociado algún código de *script* como respuesta a un evento. Los enlaces que no están controlados por código *script* pueden ser tratados como en *crawlers* convencionales, creando una nueva ruta a partir de su atributo `href`.

Sin embargo, si la página HTML contiene tecnología de *script* del lado cliente, la situación es más complicada. Cualquier elemento de la página puede contener *manejadores* asociados a diversos tipos de eventos. Un manejador es un código de *script* que se ejecutará cuando el evento al que está asociado se produzca sobre el elemento. Por ejemplo, un elemento puede tener un manejador asociado al evento llamado '`onmouseover`', que se disparará cuando el ratón se sitúe encima del elemento. Los elementos que tienen manejadores registrados y los tipos de eventos sobre los que estos se aplican, pueden detectarse a través de las APIs de los navegadores utilizados como base para los procesos de *crawling*. También pueden utilizarse estas APIs para generar automáticamente dichos eventos.

La idea principal del algoritmo consiste en generar automáticamente sobre cada elemento los eventos para los que dispone de un manejador registrado. Por ejemplo, si un elemento de la página tiene una acción definida para el elemento '`onmouseover`', el *crawler* detectará esta situación y ejecutará dicho evento sobre el elemento. La intención es comprobar si el efecto de los eventos permite obtener nuevos URLs válidos, a partir de los que crear nuevas rutas. El algoritmo tiene que considerar los diferentes escenarios que pueden ocurrir ante la ejecución de un evento, que a continuación se enumeran:

- Pueden generarse una o varias navegaciones a nuevas páginas (es posible que se abran nuevos diálogos para mostrar varias páginas). Esto ocurre si el código de *script* asociado al evento genera una o más navegaciones automáticas a nuevas páginas.
- Pueden no generarse nuevas navegaciones a páginas, pero modificarse el contenido de la página actual, apareciendo o desapareciendo algunos enlaces o elementos con manejadores asociados (e.g. menús emergentes).
- En páginas con varios marcos, es posible que se generen nuevos elementos en algunos marcos y navegaciones en otros.

En la Figura 24 se muestra un ejemplo de página dinámica en la que se presentan ocurrencias de los dos primeros escenarios: existen enlaces convencionales que llevan a nuevas páginas, pero también existen elementos controlados con código *script*, que generan menús emergentes con nuevos enlaces.

Durante el proceso de *crawling*, el mini-navegador puede estar en uno de dos estados:

- En el *estado de navegación* el navegador funciona como lo hace normalmente y cuando ocurre un evento sobre un elemento dinámico, su manejador se ejecuta sin ninguna restricción.
- En el *estado de simulación*, al ejecutar un evento sobre un elemento, el navegador sólo captura las acciones y eventos generados como consecuencia del manejador que ha sido ejecutado, pero no descarga las posibles nuevas páginas a las que navegaría automáticamente si el manejador se ejecutase completamente (aunque sí realiza las posibles transformaciones sobre la página local). El objetivo del estado de simulación es tratar elementos gestionados por código *script*, que generan automáticamente una o más navegaciones. 'Simulando' los eventos, el navegador puede obtener los nuevos URLs a los que estos elementos conducen para añadirlos a la frontera del *crawler*, pero sin que el navegador se vea redirigido a ellos. Este proceso es el análogo para este tipo de elementos al de obtener nuevos URLs examinando el atributo `href` para los enlaces convencionales.

A continuación se describe el algoritmo:

1. Sea P una página HTML que ha sido descargada por el navegador (en estado de navegación). Sea R el conjunto de rutas generadas a partir de la página P, inicialmente vacía.
2. El navegador ejecuta las secciones de *script* que no están asociadas a eventos (es decir, las secciones de *script* que forman parte del cuerpo de la página).
3. Sea E_p el conjunto formado por todos los enlaces y elementos con manejadores de código de *script* registrados de la página – es decir, elementos dinámicos de la página –.
4. Mientras queden elementos en E_p, para cada $e_i \in E_p$:
   - Eliminar e_i de E_p.
   - Si e_i es un enlace cuyo atributo `href` no contiene código de *script* asociado y no tiene registrado ningún manejador de eventos, se añade el enlace e_i a la lista de rutas obtenidas de la página: $R = e_i \cup R$.
   - En otro caso (es decir, si e_i es un enlace o cualquier otro elemento de una página web que posea manejadores de eventos de navegación registrados), el navegador cambia a estado de simulación y genera sobre ese elemento todos los eventos para los que tenga definidos manejadores. Tras la generación de cada evento sobre el elemento, el sistema procede como se indica a continuación:
     - Existen algunos elementos que ante determinados eventos, pueden generar acciones no deseadas (e.g. la invocación al método '`javascript:close()`' cierra el navegador). La aproximación seguida para evitar esto es capturar estos eventos no deseados e ignorarlos.
     - El *crawler* captura todos los eventos de nuevas navegaciones que se producen como consecuencia de la generación del evento correspondiente sobre el elemento tratado. Cada evento de

navegación genera un URL a partir del que se crea una nueva ruta r_i, que se añade a la lista de rutas obtenidas de la página: $R = r_i \cup R$.

- Una vez ha finalizado la ejecución de las acciones desencadenadas por los eventos generados sobre un elemento dinámico, el *crawler* analiza de nuevo la misma página buscando nuevos elementos dinámicos que hayan podido ser generados, E_{np}. Éste es el caso típico de un menú emergente, que tras activar una de sus opciones ejecutando algún evento, despliega un nuevo conjunto de elementos dinámicos. Estos elementos se añaden a la lista de elementos dinámicos a seguir procesando, por lo que $E_p = E_{np} \cup E_p$. En el caso de que aparezcan nuevos enlaces convencionales, se crearán nuevas rutas r_i a añadir a la lista de rutas obtenidas de la página: $R = r_i \cup R$.

  - Volver al paso 4.

5. El navegador cambia al estado de navegación y el *crawler* está listo para procesar un nuevo URL.

Si la página procesada está compuesta por varios marcos, entonces el sistema procesará cada uno de ellos de la misma forma. Es importante que la página contenedora de los diferentes marcos se encuentre cargada en el mini-navegador para que estén accesibles todas las funciones de *script* cuando se realiza la simulación de los eventos sobre los diferentes elementos, y puedan ejecutarse correctamente.

Destacar que el sistema procesa los enlaces y elementos con manejador de eventos en una página siguiendo una aproximación *bottom-up*, de tal forma que los nuevos elementos serán procesados antes de que el evento notificado sobre otros pueda eliminarlos de la página. Esto último está basado en el comportamiento habitual de los menús o árboles desplegables con código *script*, que normalmente modifican el árbol DOM desde su posición hacia abajo, manteniendo la estructura de los nodos superiores. Si se comenzase el proceso con los nodos superiores, cuando se intentase realizar la simulación sobre nodos inferiores, podría suceder que ya no formasen parte de la página y por lo tanto no se podría acceder a ellos.

Debe resaltarse que, al igual que cualquier otra ruta, las nuevas rutas añadidas tendrán que ser filtradas por el clasificador de rutas definido en el apartado III.2.1, cuando se describió la arquitectura del módulo de *crawling*.

En el prototipo de la arquitectura se ha implementado el algoritmo, considerando los enlaces y elementos que tengan registrado algún manejador para eventos ‘onclick’ u ‘onmouseover’.

III.4. WEB OCULTA DEL LADO SERVIDOR

En esta sección se describen las técnicas utilizadas para determinar si un determinado formulario web es relevante para una tarea de *crawling* dirigido y, en ese caso, para aprender a realizar consultas sobre él de forma automática. Una vez hecho esto, se añadirán a la frontera del *crawler* rutas para ejecutar sobre el formulario las consultas predefinidas en el dominio de aplicación.

Dado un formulario *f* localizado en una página HTML determinada y un dominio de aplicación *d* describiendo el esquema de los datos objetivo, se pretende determinar si *f* permite ejecutar consultas sobre los atributos de búsqueda especificados en la metainformación del dominio de aplicación *d*.

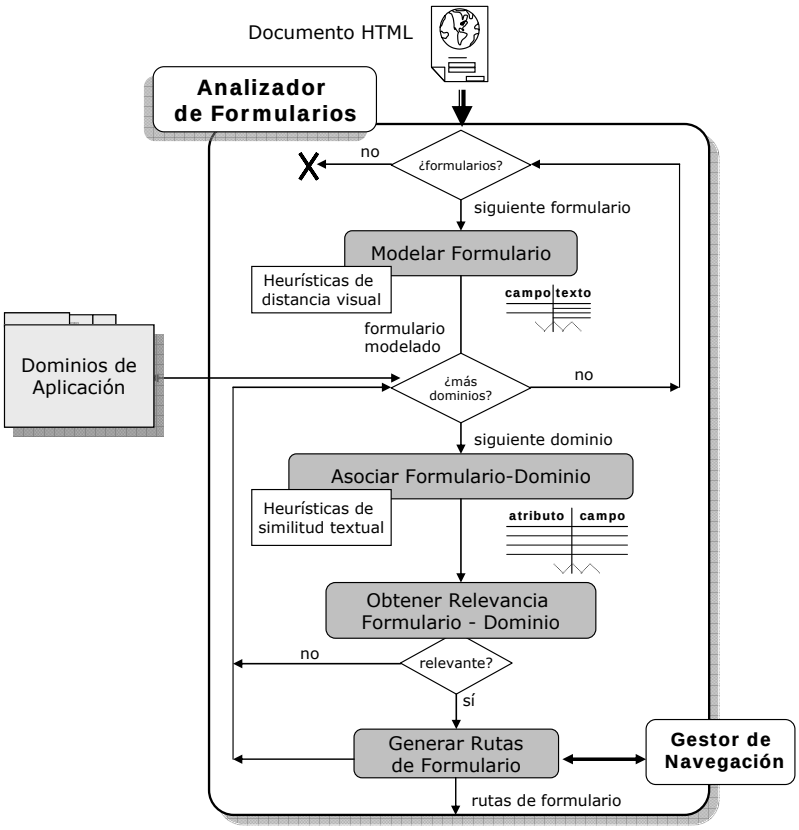


Figura 32 Arquitectura del analizador de formularios

En la Figura 32 se muestra la arquitectura del analizador de formularios. El algoritmo utilizado para procesar cada formulario *f* tiene las siguientes fases:

- Modelar el formulario *f*, estructurándolo en base a sus elementos constituyentes: inputs, labels, selects, options, etc. Esta fase se complementa con los siguientes

aspectos para intentar asociar a cada campo del formulario la semántica que tiene para un usuario:

  - Análisis de la página del formulario *f* para obtener todos los textos que contiene. Entre cada texto y elemento se calculará una medida de distancia visual. La distancia visual se calcula en base a la posición de ambos elementos en pantalla, que puede obtenerse utilizando la API de un navegador (por ejemplo Microsoft Internet Explorer o Mozilla).
  - Determinar qué textos están asociados a cada elemento del formulario. Este paso está basado en las medidas de distancia visual obtenidas en el paso previo y en varias heurísticas adicionales.

- Intentar relacionar los campos del formulario *f* con los atributos del dominio *d* (por ejemplo, en una tienda electrónica de libros intentarían asociarse campos en el formulario con los atributos del dominio TÍTULO, AUTOR y FORMATO). Este paso se lleva a cabo aplicando medidas de similitud textual entre los textos asociados con cada campo del formulario y el nombre y los alias de cada atributo del dominio *d*. Para los campos del formulario *f* que presenten una lista de valores asociados (tales como listas de selección, *radio buttons* o *checkboxes*), se compararán también dichos valores con los ejemplos de valores para el atributo incluidos en el dominio de aplicación.
- Como resultado de intentar asociar los campos del formulario con los atributos del dominio, se obtendrá un valor de similitud global entre el formulario y el dominio, que servirá para decidir si se considerará ese formulario como relevante para el dominio de aplicación.
- Si se ha determinado que el formulario pertenece al dominio de aplicación tratado, entonces se aprende a realizar consultas sobre él utilizando las asociaciones establecidas entre los elementos del formulario y del dominio. Como resultado se obtendrán rutas a páginas de resultados que serán añadidas a la frontera del *crawler* para ser accedidas posteriormente.

Los siguientes apartados detallan cada uno de estos pasos. El apartado III.4.1 describe cómo el sistema modela cada formulario, utilizando heurísticas de distancia visual entre los campos del formulario y los textos que lo rodean. El apartado III.4.2 describe el algoritmo de asociación de campos del formulario con atributos del dominio. El apartado III.4.3 describe el algoritmo que determina si un formulario es relevante para un dominio. Finalmente, en el apartado III.4.4 se describe el algoritmo de generación de rutas representando las consultas sobre el formulario, que permitirán obtener información contenida en la Web Oculta del lado servidor.

III.4.1. MODELADO DE FORMULARIOS

El objetivo de esta etapa es realizar el modelado de un formulario para detectar la semántica de cada uno de sus campos, de modo que en una etapa posterior (apartado III.4.2) se posea información suficiente como para determinar de forma unívoca la correspondencia entre campos del formulario y atributos del dominio.

El modelo de navegación de alto nivel utilizado, hace que en el modelado del formulario sólo haya que considerar aquellos elementos de los que es consciente un usuario cuando lo usa. Es decir, aquellos elementos 'ocultos', como elementos input de tipo hidden, no es necesario tenerlos en cuenta. El modelo de formulario utilizado es similar al usado por HiWE (ver apartado II.3.3.4.1). Un formulario f se modela como $f = \{(t_{11}, ..., t_{1i}, d_1), ..., (t_{n1}, ... t_{nj}, d_n), e_1, ..., e_m\}$, donde:

- Cada tupla $(t_{i1}, ... t_{ij}, d_i)$ representa un campo del formulario. $t_{i1}, ... t_{ij}$ son las diferentes etiquetas asociadas a ese campo – por ejemplo con elementos HTML label, atributos name de elementos button o input de tipo submit, o texto ALTernativo para navegadores, que puede aparecer en elementos input de tipo IMAGE, etc. –. d_i representa el rango de valores del campo. En base a su rango de valores, se distinguen dos tipos de campos:
  - Campos con valores acotados. Son aquellos que presentan un conjunto finito de valores posibles, como los campos selectores, *checkboxes* o *radio buttons*.
  - Campos con valores no acotados. Son aquellos cuyos valores de consulta posibles no están limitados, como las áreas de texto.
- Cada e_k representa un elemento de control del formulario, y expresa una posible alternativa para realizar el envío del formulario al servidor. Se consideran como elementos de control tanto a los elementos del formulario input de tipo submit, e image, y elementos button, como a aquellos elementos próximos al formulario que tengan registrado algún manejador de eventos. Posteriormente se determinará, de las opciones de envío candidatas, cuál es la válida.

En la Figura 24 se muestra una página que contiene un formulario de consulta avanzada, identificando un ejemplo de cada tipo de campo. Cada campo del formulario posee un nombre interno, utilizado para identificar el valor asociado cuando se envían los datos al servidor, pero que no sirve – o no necesariamente es válido – para asociar semántica al campo. Adicionalmente puede haber etiquetas (elementos HTML label) asociadas a los elementos del formulario. Cuando existen, lo cuál es poco común, su objetivo es asociar al campo una descripción de alto nivel. También se considera la colección de textos que se encuentran próximos al campo como etiquetas. Como se puede apreciar en la Figura 24, el diseñador web, es decir, el creador del documento HTML, asigna un texto o algún otro componente a los elementos del formulario, para que el usuario sepa que en el elemento que al lado tiene el texto 'Title', debe escribir el título del libro que desea buscar. Es decir, los textos próximos a un elemento contienen su semántica correcta.

Para asociar etiquetas o textos (semántica real de los campos del formulario) utilizamos heurísticas de distancia visual entre textos y elementos del formulario. El proceso se divide

en dos etapas: una primera en la que se calculan las distancias visuales entre los textos de la página y los campos de los formularios (descrita en el apartado III.4.1.2), y una segunda en la que se asocian los campos de los formularios con los textos más próximos (descrita en el apartado III.4.1.3).

Aunque el sistema está orientado al tratamiento de formularios de búsqueda multicampo que encapsulan el acceso a una base de datos subyacente, también es capaz de tratar con otros tipos de formularios. En el apartado III.4.1.1 se comenta el tratamiento realizado para algunos formularios especiales.

III.4.1.1. CASOS PARTICULARES

Existe un caso especial de formularios que poseen todos sus campos acotados (primer ejemplo de la Figura 33). Este tipo de formularios puede considerarse como otro mecanismo de la Web de Superficie para enumerar una serie de referencias a sitios web, más que un punto de entrada a la Web Oculta del lado del servidor. Para estos formularios, el sistema procede añadiendo una nueva ruta para cada combinación posible de los elementos acotados. En el caso de que el formulario contenga elementos con manejadores de eventos registrados, en el proceso de creación de las rutas es necesario tener en cuenta los pasos comentados en el apartado III.4.4 para formularios de búsqueda, para determinar cómo realizar el envío del formulario al servidor.

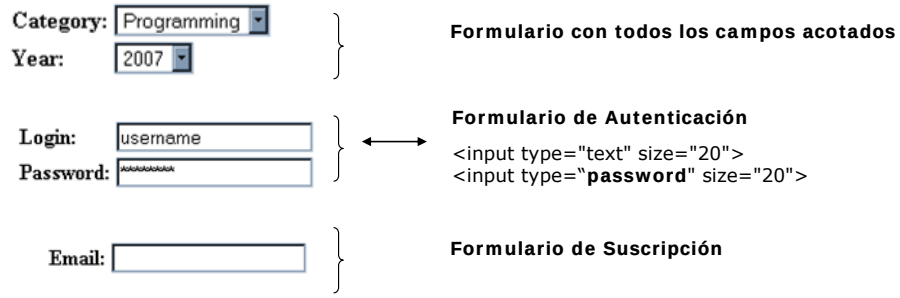


Figura 33 Ejemplos de Formularios

Existen otros tipos especiales de formularios, como son los formularios de autenticación, registro o suscripción (la Figura 33 muestra algunos ejemplos). Los formularios de autenticación son fácilmente detectables, debido a que suelen presentar un campo input de tipo password. En este caso, cuando el sistema posee información – establecida en la configuración inicial – de acceso para el formulario detectado, crea una ruta para acceder a la zona registrada y de esta forma poder continuar el *crawling*. En el resto de casos, el sistema detecta que no se trata de formularios relevantes a los dominios considerados, y los descarta.

III.4.1.2. MEDIDA DE DISTANCIAS VISUALES ENTRE TEXTOS Y CAMPOS DE FORMULARIOS

Este apartado describe la primera etapa del algoritmo de asociación de textos a campos de un formulario. El algoritmo considera todos los textos de la página y calcula sus distancias visuales con respecto a cada campo del formulario *c*. Es necesario resaltar que los elementos HTML de tipo *checkbox* y *radio button* con el mismo valor para el atributo name se consideran como un único campo del formulario. El sistema intenta asociar textos a cada *radio button* o *checkbox*, y textos para el campo agregado que representan (el grupo de *radio buttons* o *checkboxes*).

Las distancias visuales entre un texto *t* y un campo del formulario *c* se calculan de la siguiente forma:

1. Se utiliza la API de un navegador web para obtener las coordenadas del rectángulo que engloba al campo *c* y del rectángulo que engloba al texto *t* en sus representaciones visuales. Si el texto *t* se encuentra en la celda de una tabla, y es el único texto dentro de esa celda, entonces se le asignan al texto *t* las coordenadas del rectángulo que engloba la celda de la tabla.
2. A continuación se obtiene la distancia mínima entre los dos rectángulos. Esto involucra localizar la línea más corta que una ambos rectángulos. La longitud de esa línea será la distancia entre los rectángulos. Las distancias no se calculan en *píxels*, sino que se utilizan unidades de grano más grueso, para evitar la realización de asociaciones erróneas en situaciones en las que el texto no adecuado esté muy pocos *píxels* más cerca del campo, que el texto correcto. En nuestros experimentos se ha establecido heurísticamente un tamaño de celda para la normalización de las distancias de valor aproximado al tamaño de un carácter. En el apartado III.4.1.2.1 se describe en detalle el algoritmo de distancia utilizado.
3. Se obtiene el ángulo de la línea más corta que une ambos rectángulos y se aproxima al múltiplo de $\pi/4$ más cercano. De esta forma hay 8 valores posibles para el ángulo: 0 (el texto se encuentra aproximadamente a la derecha del campo), $\pi/4$ (arriba a la derecha), $\pi/2$ (encima), $3\pi/4$ (encima a la izquierda), π (a la izquierda), $-\pi/4$ (debajo a la derecha), $-\pi/2$ (debajo) y $-3\pi/4$ (debajo a la izquierda).

Para realizar una normalización más fina en el punto 2, se utilizan diferentes factores de normalización en función del tipo de elemento. En general es aplicable decir que los *checkboxes* y *radio buttons* tienen sus textos asociados mucho más próximos que el resto de elementos. Además, el tamaño de un carácter depende del tipo y tamaño de la fuente utilizada, que también es necesario tener en cuenta.

La Figura 34 muestra un ejemplo de formulario de consulta correspondiente a una tienda electrónica de libros. En la figura se muestran los rectángulos visuales que encierran a cada uno de los campos y textos del formulario, identificando cada campo con un nombre. Además, se muestran las distancias y ángulos entre el campo c_1 del formulario (el que permite buscar por el título del libro) y algunos de los textos que lo rodean. También se

muestra un ejemplo con todas las posiciones relativas consideradas entre un campo *c* y un texto *t*.

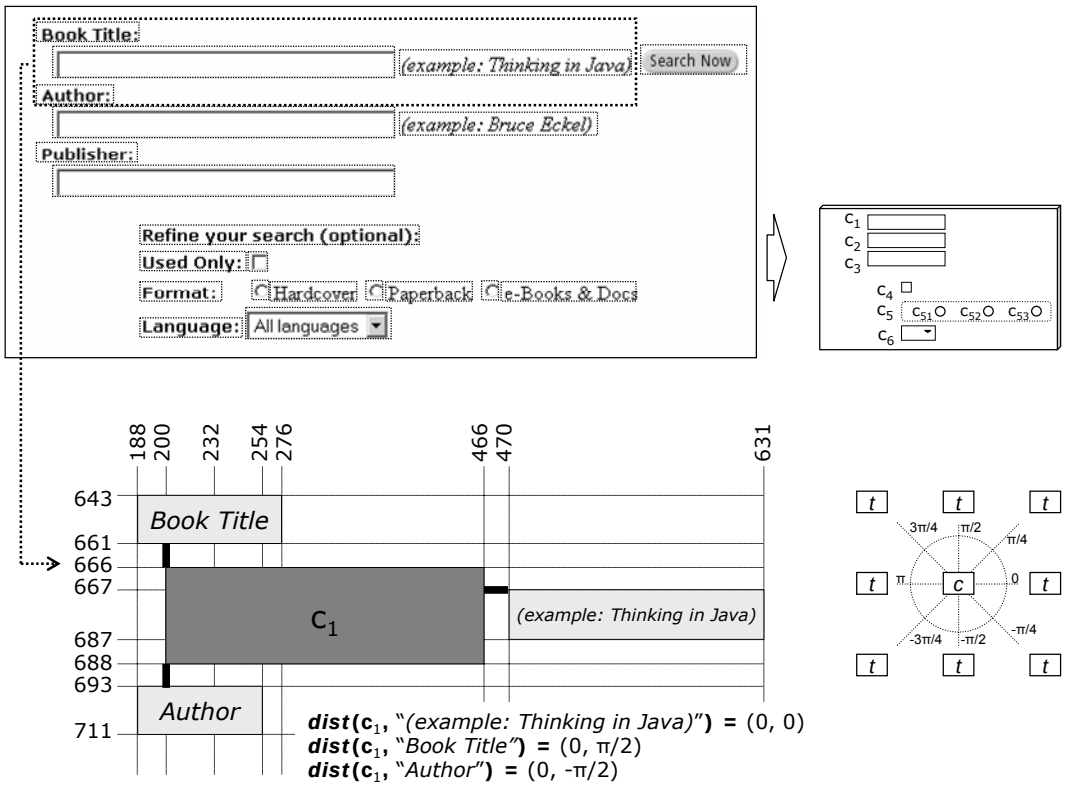


Figura 34 Formulario de consulta para una tienda electrónica de libros

III.4.1.2.1. ALGORITMO DE DISTANCIA

Para determinar la distancia entre los rectángulos se han considerado la estrategia de distancia mínima entre dos rectángulos. Se trata de un caso particular del algoritmo de distancia mínima entre polígonos. Para el caso de rectángulos se simplifica en gran medida. Para medir la distancia y ángulo entre dos rectángulos, el sistema comprueba primero si existe solapamiento entre ellos. Si no existe ningún tipo de solapamiento (ni horizontal ni vertical), entonces obtiene un punto en cada rectángulo y utiliza el Teorema de Pitágoras y la medida del coseno para obtener la distancia entre esos puntos. En la Figura 35 se marcan los puntos obtenidos de cada rectángulo, en diferentes escenarios, en función de la posición relativa de los mismos. Es importante destacar que en el caso de que exista solapamiento (es decir, cuando la distancia es 0), como en el caso e) de la figura, para el cálculo del ángulo se utilizan puntos pertenecientes a la línea de distancia mínima entre los rectángulos reducidos (para evitar el solapamiento y poder determinar correctamente las posiciones relativas).

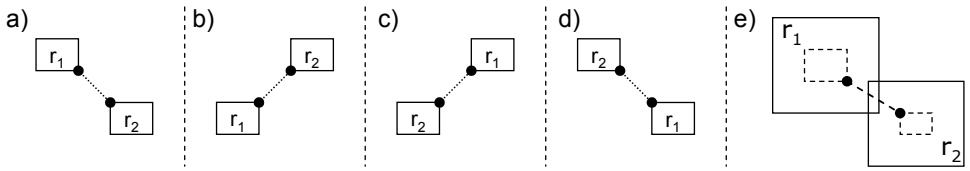


Figura 35 Posiciones relativas entre dos rectángulos, r_1 y r_2

III.4.1.3. ASOCIACIÓN DE TEXTOS Y CAMPOS DE FORMULARIOS

Este apartado describe la segunda etapa del algoritmo de asociación de textos a campos de un formulario. A partir de la salida obtenida de la etapa anterior, es decir, las distancias y ángulos de cada texto con respecto a los campos de un formulario, el objetivo de esta etapa es obtener para cada campo del formulario los textos 'asociados semánticamente' con él en la página. Por ejemplo, en la Figura 34 las cadenas asociadas semánticamente al primer campo son *'Book Title'* y *'(example: 'Thinking in Java')'*.

El proceso de asociación se basa en las siguientes observaciones:

Observación 1: Los textos semánticamente asociados a un campo de un formulario se muestran normalmente próximos al campo. Además, no sucede que textos que no están semánticamente relacionados con el campo se sitúen en la página significativamente más próximos que los semánticamente relacionados. Esta heurística se basa en que si fuese de otra forma, la interpretación de la semántica del campo podría ser confusa para el usuario.

Observación 2: Los textos semánticamente asociados a un campo de un formulario se muestran normalmente alineados con el campo. Esto significa que el ángulo que representa las posiciones relativas entre los rectángulos del texto y el campo del formulario típicamente serán un múltiplo de $\pi/2$ ($3\pi/2$, 0, $\pi/2$ o π). Para incrementar la precisión en los cálculos del algoritmo de asociación, en la fase anterior (ver apartado III.4.1.2) se ha normalizado a ángulos múltiplos de $\pi/4$.

Basándose en estas observaciones el sistema hace una primera preselección de los 'mejores textos' para cada campo *c* aplicando los siguientes pasos:

1. Primero, se añaden a la lista los textos que tienen la distancia *d* más corta con respecto a *c* en la lista.
2. Los textos que estén a una distancia más pequeña que *k*d* de *c* también se añaden a la lista ordenados por su distancia. Este paso descarta aquellos textos que están significativamente lejos del campo con respecto a los más cercanos. En los experimentos se ha determinado que con k=5 se obtienen buenos resultados.
3. Los textos que están a la misma distancia se ordenan de acuerdo a su ángulo (dado que la distancia se mide en unidades de grano más grueso que los *píxels*, es relativamente frecuente que varios textos estén a la misma distancia de un campo). Siguiendo la observación 2, el orden de preferencia privilegia aquellos textos que están alineados con los campos (es decir, ángulos que sean múltiplos de $\pi/2$). En

caso de empate, y dependiendo del tipo de elemento, también se privilegian direcciones concretas. Por ejemplo, en los elementos input de tipo radiobutton el texto suele aparecer a la derecha del elemento y en los input de tipo text, el texto suele aparecer a la izquierda o arriba. Cuando dos textos están a la misma distancia y ángulo, entonces se analiza el ángulo real (sin normalizar), y se dará precedencia al que más se aproxima a su valor normalizado.

Como salida de estos tres pasos se obtendrá una lista ordenada de textos para cada campo del formulario, en la que cada texto puede aparecer repetido en varias listas. El siguiente paso consiste en post-procesar las listas de acuerdo a las siguientes observaciones.

Observación 3: Normalmente, cada campo de un formulario tiene al menos un texto asociado semánticamente para permitir que el usuario identifique la función del campo. La razón de esto es que la semántica de un campo de un formulario sin ningún texto asociado no estaría clara para un usuario.

Observación 4: Podrá haber textos que estén semánticamente asociados a más de un campo (por ejemplo 'Refine your search:' en la Figura 34 está semánticamente asociado a los campos 'Publisher', 'Used Only', 'Format' y 'Language'). Sin embargo, precisamente debido a que estos textos se refieren a varios campos, normalmente no describen a ninguno de ellos de manera precisa, sino que proporcionan cierta información general aplicable a todos ellos.

Utilizando estas dos observaciones, las listas de textos se reducen de acuerdo a los siguientes criterios, para asociar de forma no ambigua textos con campos del formulario:

1. Un texto debe estar presente sólo en la lista de un único campo. Esto implica reducir el tamaño de las listas de textos, puesto que si un texto dado está presente en las listas de k campos (donde $k>1$), será eliminado de $k-1$ listas. Para elegir la lista en la que debe permanecer el texto, se utiliza el criterio de seleccionar la lista en la que el texto está en una posición más alta (recuérdese que los textos de las listas están ordenados en base a su distancia hasta el campo correspondiente).
2. De acuerdo a la observación 3, cuando se aplica la regla del punto anterior también hay que tratar de asegurar que ningún campo quede sin tener asociado al menos un texto. Por ejemplo, si la lista de textos de un campo c_1 contiene los textos t_1 y t_2 (en ese orden), y la lista de un campo c_2 solamente contiene el texto t_1, entonces debería eliminarse de la lista de c_1, ya que si se elimina de c_2 quedaría el campo con una lista vacía. Nótese que esto se haría incluso si t_1 estuviese más cerca de c_1 que de c_2.

Es importante tener en cuenta que, como se ha comentado al comienzo de este apartado, algunos campos del formulario pueden poseer textos asociados de forma única mediante etiquetas label o textos alternativos, entre otros. Estos elementos podrían quedar sin nuevos textos asociados en esta etapa del algoritmo por ya poseer textos propios (es decir, no pueden competir por textos con elementos que no poseen ninguno asociado por sintaxis del formulario). Otros casos a tener en cuenta son los elementos *radio button* y *checkboxes* que sólo presenten un elemento, en cuyo caso el elemento que los agrupa no competirá para obtener textos.

La Figura 36 muestra un ejemplo del proceso completo de asociación de textos y campos de formularios para el formulario de ejemplo de la Figura 34. Para cada campo del

formulario, se muestra la lista ordenada de textos obtenidos utilizando las heurísticas de distancia y ángulos visuales. Debe destacarse cómo el sistema modela el campo de tipo *checkbox* FORMAT como un campo con tres subcampos. c_5 se refiere al conjunto completo de *checkboxes*, mientras que c_{51}, c_{52} y c_{53} se refieren a los *checkboxes* individuales. Los textos que se mantienen en las listas, después del paso de reducción, se muestran en negrilla. Por ejemplo, para el campo c_1 los textos asociados finalmente son '(example: Thinking in Java)' y 'Book Title:'.

Campos		Textos	(dist, α)
c_1	√	**(example: Thinking in Java)**	(0,0)
	√	**Book Title:**	(0, π/2)
		Author:	(0, -π/2)
		(example: Bruce Eckel)	(1, -π/2)
		Publisher:	(3, -π/2)
c_2	√	**(example: Bruce Eckel)**	(0, 0)
	√	**Author:**	(0, π/2)
		Publisher:	(0, -π/2)
		(example: Thinking in Java)	(2, π/2)
		Book Title:	(3, π/2)
c_3	√	**Publisher:**	(0, π/2)
		Refine your search (optional):	(1, -π/2)
		(example: Bruce Eckel)	(2, π/2)
		Author:	(3, π/2)
		Used Only:	(3, -π/2)
		Format:	(4, -π/2)
		Hardcover	(4, -π/2)
		Paperback	(4, -π/2)
c_4	√	**Used Only:**	(0, π)
	√	**Refine your search (optional):**	(0, π/2)
		Hardcover	(0, -π/2)
		Format:	(1, π)
		Language:	(2, -π/2)
c_5		e-Books & Docs	(0, 0)
		Used Only:	(0, 3π/4)
		Language	(0, -3π/4)
	√	**Format:**	(1, π)
		Refine your search (optional):	(1, π/2)

Campos		Textos	(dist, α)
c_{51}	√	**Hardcover**	(0, 0)
		Used Only:	(0, 3π/4)
		Language:	(0, -3π/4)
		Format:	(1, π)
		Refine your search (optional):	(1, π/2)
c_{52}		Hardcover	(0, π)
	√	**Paperback**	(0, 0)
		Refine your search (optional):	(1, π/2)
c_{53}		Paperback	(0, π)
	√	**e-Books & Docs**	(0, 0)
		Refine your search (optional):	(2, 3π/4)
c_6	√	**Language:**	(0, π)
		Hardcover	(0, π/2)
		Paperback	(0, π/2)
		Format:	(1, π)
		Used Only:	(1, π/2)
		Refine your search (optional):	(3, π/2)

c_1 [*(example: Thinking in Java)*] [*Book Title:*]
c_2 [*(example: Bruce Eckel)*] [*Author:*]
c_3 [*Publisher:*]
c_4 [*Used Only:*] [*Refine your search (optional):*]
c_5 [*Format:*]
c_{51} [*Hardcover*]
c_{52} [*Paperback*]
c_{53} [*e-Books & Docs*]
c_6 [*Language*]

Figura 36 Textos asociados a cada campo del formulario de la Figura 34

III.4.2. ASOCIACIONES FORMULARIO-DOMINIO

El objetivo de esta etapa es intentar detectar los campos del formulario que se corresponden con atributos del dominio de aplicación tratado.

Se distinguen dos tipos de campos: los campos con valores acotados y aquellos otros con un número ilimitado de valores posibles.

La idea básica para obtener la similitud entre un campo *c* y un atributo *a* consiste en medir la similitud textual entre los textos asociados con *c* en la página (obtenidos como se explicó en los apartados previos) y los textos asociados con *a* en el dominio (el nombre del atributo y su lista de alias). Cuando el campo es acotado, el sistema también tiene en cuenta las similitudes textuales entre los valores posibles de *c* en la página y los valores asociados al atributo *a* en las consultas especificadas para el dominio. Obtener estos valores es trivial para etiquetas de tipo select-option, ya que los posibles valores aparecen en el código HTML encerrados en las marcas option. Para las marcas de tipo checkbox y radiobutton, es necesario utilizar técnicas de distancia visual, como se ha visto en el apartado anterior.

Las medidas de similitud se obtienen utilizando un método propuesto por Cohen et al. en [CRF03], que combina TFIDF y el algoritmo de distancia de edición Jaro-Winkler. Los experimentos llevados a cabo en dicho trabajo estudiaron un amplio abanico de métodos para estimar la similitud entre dos textos y concluyeron que éste es el más efectivo para estimar la similitud de nombres y etiquetas cortas.

Antes de calcular similitudes, se realiza una normalización de los textos eliminando palabras usuales (*stop words*), y caracteres no alfanuméricos. Adicionalmente, las palabras en el texto se ordenan alfabéticamente. No se aplican técnicas de lematización de palabras porque los sufijos son de gran utilidad para diferenciar alias de campos, mejorando la precisión de las asociaciones (e.g. 'Published' y 'Publisher').

Para puntuar la similitud entre un campo *c* y un atributo *a* se utiliza el siguiente método:

1. Los textos que describen el atributo *a* (es decir, su nombre y lista de alias especificados en el dominio) los denotaremos como el conjunto $\{a\_alias_1, \ldots, a\_alias_n\}$. Los textos de la página asociados con un campo *c* del formulario obtenidos utilizando el método expuesto en el apartado previo, los denotaremos como $\{c\_texto_1, \ldots, c\_texto_m\}$.
2. Si *c* es un campo acotado, entonces se obtienen de la página sus valores posibles $\{c\_valor_1, \ldots, c\_valor_p\}$. También se examinan las consultas especificadas en el dominio para obtener el conjunto de valores usados en ellas para el atributo *a* $\{a\_valor_1, \ldots, a\_valor_q\}$.
3. A continuación se calcula la similitud textual entre cada par $(a\_alias_i, c\_texto_j)_{i=1..n,\ j=1..m}$. Y se calcula sim_1 como el máximo de las similitudes textuales obtenidas (III.4-1).

$$sim_1 = \max\left\{ similitudTextual\left(a\_alias_i, c\_texto_j\right)_{i=1..n, j=1..m} \right\} \qquad \textbf{(III.4-1)}$$

4. Si *c* es un campo acotado, entonces también se calcula la similitud textual entre cada par $(c\_valor_k, a\_valor_r)_{k=1..p,\ r=1..q}$. Se calcula sim_2 como la media de las máximas similitudes obtenidas para cada $a\_valor_{r,r=1..q}$ (III.4-2).

$$sim_2 = \frac{\sum_{r=1..q} \max\left\{ similitudTextual\left(a\_valor_r, c\_valor_k\right)_{k=1..p} \right\}}{q} \qquad \textbf{(III.4-2)}$$

5. Si *c* es acotado, entonces la similitud entre *a* y *c* es igual a $max\{sim_1, sim_2\}$. Si *c* no es acotado, entonces la similitud es igual a sim_1.

Como resultado de aplicar los pasos previos a cada atributo del dominio de aplicación y a cada campo del formulario, se obtiene una tabla con la similitud estimada entre cada campo del formulario y cada atributo. A continuación se procede de la siguiente forma:

1. Los pares de la tabla que no alcancen un mínimo umbral de similitud se descartan.
2. Si la tabla no contiene más de una entrada para el mismo atributo, el proceso termina y la tabla contiene las asociaciones válidas entre campos del formulario y atributos del dominio de aplicación.
3. Si la tabla contiene más de una entrada para el mismo atributo, se elige para cada atributo la entrada con similitud más alta pero intentando garantizar que ningún campo con alguna entrada por encima del umbral queda sin asignar. Por ejemplo, si el campo c_2 fuese la mejor opción para dos atributos a_1 y a_2, y c_1 fuese también una opción adicional para a_1, las asignaciones que se harían serían c_1-a_1 y c_2-a_2 en lugar de hacer solamente la asignación c_2-a_1 y dejar c_1 sin asociar.

La salida de esta fase es un conjunto de asociaciones $\{A_1, ..., A_k\}$ entre campos del formulario y atributos del dominio. El sistema establece un nivel de confianza a cada una de las asociaciones, en base a la similitud obtenida entre el campo y el atributo.

Asociación	*Campo del Formulario*	*Atributo del Dominio*	C_i *(confianza)*
A_1	c_1	a_1 = TÍTULO	0.71
A_2	c_2	a_2 = AUTOR	1
	c_3	*(sin asociar)*	
	c_4	*(sin asociar)*	
A_3	c_5	a_4 = FORMATO	1
	c_6	*(sin asociar)*	

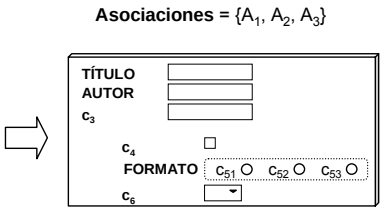


Figura 37 Asociaciones obtenidas para el formulario de la Figura 34 utilizando el dominio definido en la Figura 27

La Figura 37 muestra las asociaciones obtenidas para el formulario de la Figura 34, utilizando el dominio mostrado en la Figura 27.

III.4.3. RELEVANCIA DE UN FORMULARIOS PARA UN DOMINIO

La salida de la fase descrita en el apartado anterior es un conjunto de asociaciones $\{A_1, ..., A_k\}$ entre campos del formulario y atributos del dominio de aplicación. Cada una de estas asociaciones tiene un grado de confianza expresado como un número entre 0 y 1. Denotamos la confianza de la asociación A_i como C_i.

El método utilizado para determinar si un formulario es relevante para un dominio de aplicación determinado consiste en la suma de la confianza que aporta cada una de las asociaciones, ponderadas por el índice de especificidad del atributo involucrado en la relación, y comprobando si la suma excede el umbral de relevancia para el dominio, μ. Es decir, el sistema comprueba si se verifica la siguiente desigualdad (III.4-3):

$$\sum_{i=1..k} C_i e_i > \mu \qquad \textbf{(III.4-3)}$$

Por ejemplo, considerando la definición de dominio mostrada en la Figura 27, y las asociaciones mostradas en la Figura 37, obtendríamos el siguiente resultado (III.4-4):

$$0.71 \cdot 0.6 + 1 \cdot 0.7 + 1 \cdot 0.25 = 1.376 > \mu = 0.9 \qquad \textbf{(III.4-4)}$$

III.4.4. GENERACIÓN DE NUEVAS RUTAS

Una vez el sistema ha determinado que un formulario es relevante para un dominio de aplicación *d*, debe obtenerse una nueva ruta para cada consulta especificada en *d*. Representar la ejecución de una consulta al formulario como una ruta involucra rellenar los campos del formulario de acuerdo a la consulta y realizar la invocación del formulario.

La primera tarea se puede realizar de forma sencilla a partir de las asociaciones entre campos del formulario y atributos del dominio.

La segunda tarea presenta algunas complicaciones. Aunque los mini-navegadores utilizados por el sistema de *crawling* pueden invocar directamente la operación de submit sobre el formulario, una vez ha sido rellenado, esta estrategia no es válida para algunos sitios web. La razón es que muchos sitios web hacen uso de forma frecuente de lenguajes de *script* del lado cliente para gestionar la invocación de formularios. Por ejemplo, es bastante habitual que al realizar el envío del formulario se ejecute algún código de *script* asociado a eventos como 'onsubmit', que cambie la propiedad action del formulario o el valor de alguno de sus parámetros. En otras ocasiones el código de *script* es utilizado para validar las entradas realizadas sobre el formulario, antes de permitir la invocación del mismo. Es también bastante habitual que el envío del formulario tenga lugar al hacer clic sobre un enlace o una imagen.

Para tratar con estas complejidades, el sistema procede con los siguientes pasos, en el orden especificado, para intentar determinar la forma de invocación adecuada que considere los manejadores asociados al envío del formulario:

1. El sistema busca elementos input de tipo submit, image o button en el formulario (la búsqueda la realiza en ese orden). Cada elemento encontrado se utiliza para intentar realizar la invocación del formulario, generando un evento de clic sobre él. Después de cada intento, el sistema comprueba si el evento ha generado una nueva navegación en el mini-navegador. Si no fue el caso, entonces el sistema realiza la misma prueba con el siguiente elemento.
2. Si el paso previo no ha tenido éxito (típicamente porque no existen los tipos de elementos input buscados), el sistema concluye que la forma utilizada para invocar el formulario debe de ser a través de algún *script*, ejecutado al invocar algún evento sobre un elemento del formulario o próximo al mismo, que tenga registrado un manejador de ese evento. De esta forma, el sistema busca enlaces u otros elementos que tengan registrados manejadores de eventos (principalmente

onclick, pero no exclusivamente), y cuya localización visual sea próxima al formulario. Los elementos obtenidos se ordenan de acuerdo a su proximidad visual con respecto al formulario y la similitud textual entre sus textos asociados y un conjunto de textos predefinidos que se utilizan habitualmente para especificar invocación de formularios (e.g. 'search', 'go', 'submit', etc.). El sistema genera el evento manejado por cada elemento de la lista y comprueba si provoca una nueva navegación en el mini-navegador. Si no es el caso, entonces el sistema lo intenta con el siguiente paso.

3. Los pasos anteriores intentan localizar un elemento de la interfaz que realice el envío del formulario de la misma forma que un usuario, para tener en cuenta cualquier manejador que pueda estar asociado al envío de ese formulario. Si todos los pasos previos han fallado, el sistema realiza un envío directo del formulario (a través de la API del navegador).

Para determinar si un intento de envío es correcto, se comprueba si genera una nueva navegación. Destacar que se considera también como navegación cualquier petición que desde código de *script* se haga desde el navegador a un servidor – este último es el comportamiento típico de muchas aplicaciones implementadas utilizando la tecnología del lado cliente *Ajax* –.

Una vez obtenida la forma de invocar el formulario, el sistema genera nuevas rutas de consulta al formulario para cada una de las consultas predefinidas en el dominio de aplicación y las añade a la frontera del *crawler*. Además, se marcarán como rutas asociadas a páginas de respuesta de consultas en formularios web, para que puedan ser identificadas en la fase de indexación.

III.5. ESTRUCTURACIÓN Y ETIQUETACIÓN

En esta sección se describen las técnicas propuestas en este trabajo para automatizar la extracción de tuplas a partir de las páginas resultado de realizar consultas sobre formularios web considerados relevantes para uno de los dominios de aplicación que guían el proceso de *crawling*.

Típicamente, los formularios de consulta web devuelven los resultados de la consulta embebidos en páginas HTML conforme a una plantilla determinada. Estas fuentes de datos se denominan normalmente fuentes de datos web semi-estructuradas. La Figura 38 muestra un ejemplo de página conteniendo una lista de registros de datos, cada uno representando la información de un libro en una tienda electrónica.

Figura 38 Ejemplo de página HTML conteniendo una lista de registros de datos

Las técnicas presentadas requieren una única página como entrada y han sido validadas contra un gran número de sitios web reales, mostrando un alto nivel de efectividad (ver apartado IV.1.2).

El apartado III.5.1 describe algunas definiciones básicas y establece el modelo de generación de páginas asumido por las técnicas. El apartado III.5.2 presenta la arquitectura general del componente de estructuración automática.

Los apartados III.5.3, III.5.4 y III.5.5 describen las técnicas propuestas y constituyen la parte central del sistema de estructuración. El apartado III.5.3 describe el método utilizado para detectar la región de datos en la página que contiene la lista de registros. El apartado III.5.4 explica cómo se segmenta la sección de datos en registros de datos individuales. El apartado III.5.5 describe cómo se extraen los valores de cada atributo individual del registro. El apartado III.5.6 comenta brevemente los pasos seguidos para la etiquetación de los atributos de los registros de datos obtenidos, es decir para asignar un nombre a cada atributo.

III.5.1. DEFINICIONES Y MODELADO DEL PROBLEMA

Este apartado introduce un conjunto de definiciones y modelos que serán usados como punto de partida para describir los algoritmos de estructuración.

III.5.1.1. LISTAS DE REGISTROS DE DATOS ESTRUCTURADOS

El objetivo que se pretende es detectar y extraer listas de registros de datos estructurados que se encuentran embebidos en páginas HTML. A continuación se definen formalmente los conceptos de datos estructurados y el concepto de lista de registros de datos.

Se considera que datos estructurados son cualquier conjunto de valores de datos conformes a un *esquema* o *tipo* común. Un *tipo* se define de forma recursiva como sigue (Abiteboul et al. [AHV95], Arasu y García-Molina [AG03a]):

- El tipo básico, denotado como β, representa una cadena de *tokens* (un *token* se define como una palabra o una marca HTML).
- Si $T_1, ..., T_n$ son tipos, entonces su lista ordenada $\langle T_1, \ldots, T_n \rangle$ también es un tipo construido a partir de $T_1, \ldots, T_n$ usando un constructor de tupla de orden n.
- Si T es un tipo, entonces $\{T\}$ también es un tipo construido a partir de T usando un constructor de conjunto.

El término constructor de tipo es utilizado para referirse tanto al constructor de tupla como al de conjunto.

Una instancia de un esquema se puede definir de forma recursiva de la siguiente forma:

- Una instancia de un tipo básico, β, es cualquier cadena de *tokens*.
- Una instancia del tipo $\langle T_1, T_2, \ldots, T_n \rangle$ es una tupla de la forma $\langle i_1, i_2, ..., i_n \rangle$, donde $i_1, i_2, ..., i_n$ son instancias de los tipos $T_1, T_2, \ldots, T_n$ respectivamente. Las instancias $i_1, i_2, ..., i_n$ se denominan atributos de la tupla.

- Una instancia del tipo *{T}* es cualquier conjunto de elementos $\{e_1, ..., e_m\}$, de tal forma que cada e_i ($1 \leq i \leq m$) es una instancia del tipo *T*.

Además de los constructores de tupla y conjunto hay otros dos tipos de constructores de tipos que ocurren normalmente en las páginas web: los atributos opcionales y las alternativas. Estos dos últimos constructores de tipos pueden ser considerados como constructores de tipos especiales construidos a partir de los constructores de tupla y de conjunto:

- Si *T* es un tipo, entonces *(T)?* es el tipo opcional *T* y es equivalente a $\{T\}_\tau$ con la restricción de que cualquier instancia de τ tiene cardinalidad 0 o 1.
- Si T_1 y T_2 son tipos, $(T_1 \mid T_2)$ representa a un tipo que es la disyunción de T_1 y T_2 y es equivalente a $<\{T_1\}_{\tau 1},\{T_2\}_{\tau 2}>_\tau$, donde para cada instancia de τ, τ_1 o τ_2 tienen cardinalidad uno y el otro cardinalidad cero.

De acuerdo a esta definición, una lista de registros de datos estructurados es una instancia de un tipo de la forma $\{<T_1, ..., T_n>\}$, donde $T_1, T_2, ..., T_n$ son los tipos de los atributos del registro en la lista.

Por ejemplo, el tipo de una lista de libros en la que cada libro incluye título, autor, formato y precio podría representarse como {<TÍTULO, AUTOR, FORMATO, PRECIO>}, donde TÍTULO, AUTOR, FORMATO y PRECIO representan tipos básicos.

III.5.1.2. LISTAS DE REGISTROS DE DATOS EN PÁGINAS HTML

La respuesta a una consulta realizada contra un repositorio de datos estructurado será una lista de registros de datos estructurados conforme al modelo descrito en el apartado anterior. Esta lista necesita ser embebida por un programa en una página HTML antes de que sea presentada al usuario. El modelo utilizado por nuestras técnicas para la creación de la página es el mismo que el descrito por Arasu y García-Molina en [AG03a].

Un valor *x* de una base de datos se codifica en una página utilizando una plantilla *P*. Denotamos la página resultado de la codificación de *x* utilizando *P*, como $\lambda(P, x)$.

Una plantilla *P* para un esquema *E*, se define como una función que asigna a cada constructor de tipo τ de *E* un conjunto ordenado de cadenas $P(\tau)$, tal que:

- Si τ es un constructor de tupla de orden *n*, $P(\tau)$ es un conjunto ordenado de $n+1$ cadenas $<C_{\tau 1},...,C_{\tau(n+1)}>$. Cada una de esas cadenas se correspondería con el fragmento de plantilla presente entre cada dos valores de la tupla, el de comienzo de tupla y el de fin de tupla.
- Si τ es un constructor de conjunto, $P(\tau)$ es una cadena C_τ. Esa cadena se correspondería con el fragmento de plantilla presente entre cada dos ocurrencias del conjunto.

Dada una plantilla *P* para un esquema *E*, la codificación $\lambda(P, x)$ de una instancia *x* de *E* se define de forma recursiva en términos de la codificación de los subvalores de *x*:

- Si x es un tipo básico, β, $\lambda(P, x)$ se define directamente como x.
- Si x es una tupla de la forma $<x_1, ..., x_n> \tau_t$, $\lambda(P, x)$ es la cadena $C_1\ \lambda(P, x_1)\ C_2\ \lambda(P, x_2)\ ...\ C_n\ \lambda(P, x_n)\ C_{n+1}$. Aquí, x es una instancia del subesquema encabezado por el constructor de tipo τ_t en E, y $P(\tau_t)=<C_1, ..., C_{n+1}>$.
- Si x es un constructor de conjunto de la forma $\{e_1, ..., e_m\}\ \tau_c$, $\lambda(P, x)$ se define como la cadena $\lambda(P, e_1)\ C\ \lambda(P, e_2)\ C\ ...\ C\ \lambda(P, e_m)$. Aquí x es una instancia del subesquema encabezado por el constructor de tipo τ_c en E, y $P(\tau_s)=C$.

Este modelo de creación de páginas define una forma precisa para embeber listas de registros de datos en páginas HTML. Por ejemplo, la Figura 39 muestra un fragmento de código HTML de la página de la Figura 38, junto con la plantilla utilizada.

```
(                                                                    PLANTILLA
   <tr><td><table>
    <tr>
     <td>
      <span><a>$TÍTULO</a></span>
      <br>by <span>$AUTOR</span>
      <br><span>$FORMATO</span> - $FECHAPUBLICACIÓN</td>
     <td><img></td></tr></table></td></tr>
   <tr><td>Buy new: <span>$PRECIO€</span>
        ¿ Price used: <span>$PRICIOUSADO€</span> ?
     </td></tr>
 ¿ <tr><td>Other editions: <span>$OTRASEDICIONES</span></td></tr> ?
) *
```

```
<html><body>
 <div> ... </div>
 <div> ... </div>
 <div>
  <table> ... </table>
  <table>
   <tr><td><table>
    <tr><td>
        <span><a>Head First Java. 2nd Edition</a></span>
        <br>by <span>Kathy Sierra and Bert Bates</span>
        <br><span>Paperback</span> - Feb 9. 2005</td>
       <td><img></td></tr></table></td></tr>
   <tr><td>Buy new: <span>29.67€</span>
        Price used: <span>20.00€</span></td></tr>

   <tr><td><table>
    <tr><td>
        <span><a>Java Persistence with Hibernate</a></span>
        <br>by <span>Christian Bauer and Gavin King</span>
        <br><span>Paperback</span> - Nov 24. 2006</td>
       <td><img></td></tr></table></td></tr>
   <tr><td>Buy new: <span>37.79€</span></td></tr>
   <tr><td>Other editions: <span>e-Books & Docs</span></td></tr>
   ...
  </table>
  ...
 </div>
</body></html>                                                  CÓDIGO HTML
```

Figura 39 Código fuente HTML y plantilla de la página de la Figura 38

El modelo captura la intuición básica de que cada registro de datos en una lista está formateado de forma consistente, es decir, las ocurrencias de cada atributo en varios registros están formateadas de la misma forma y siempre ocurren en la misma posición relativa respecto a los restantes atributos. Además, esta definición también establece que los registros de datos de una lista se muestran de forma contigua en la página.

III.5.1.3. LISTAS DE REGISTROS EN EL ÁRBOL DOM DE UNA PÁGINA HTML

Otra representación posible para las páginas HTML es la representación como árboles DOM (*Document Object Model*) [W3CDOM]. Por ejemplo, la Figura 40 muestra una parte del árbol DOM del código HTML de la Figura 39.

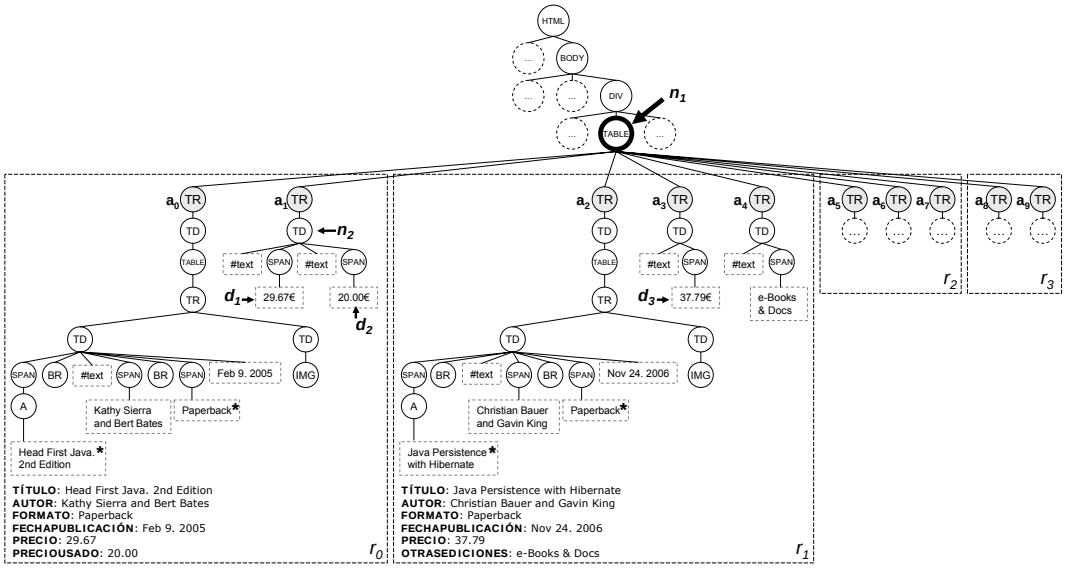


Figura 40 Árbol DOM para la página HTML de la Figura 38

Del modelo definido en el apartado anterior para embeber la lista de registros de datos en HTML, podemos derivar las siguientes propiedades de sus representaciones como árboles DOM:

- **Propiedad 1**: Cada registro en el árbol DOM está compuesto por un conjunto de subárboles consecutivos, con la misma raíz. Adicionalmente, aunque no se deriva estrictamente del modelo de creación de página, se comprueba de forma heurística que un registro de datos está compuesto por un número determinado de subárboles completos. Por ejemplo, en la Figura 40 los dos primeros subárboles forman el primer registro y los tres siguientes subárboles forman el segundo registro.
- **Propiedad 2**: Las ocurrencias de cada atributo en varios registros tienen la misma ruta XPath [W3CXPT] desde la raíz del árbol DOM del documento. Por ejemplo, en la Figura 40 se puede observar cómo todas las instancias del atributo TÍTULO tienen la misma ruta XPath desde la raíz del árbol DOM, y lo mismo es aplicable a los restantes atributos.

III.5.2. ARQUITECTURA DEL COMPONENTE DE ESTRUCTURACIÓN

En este apartado presentamos la arquitectura general del componente de estructuración automática. La Figura 41 resume las diferentes etapas del algoritmo de estructuración.

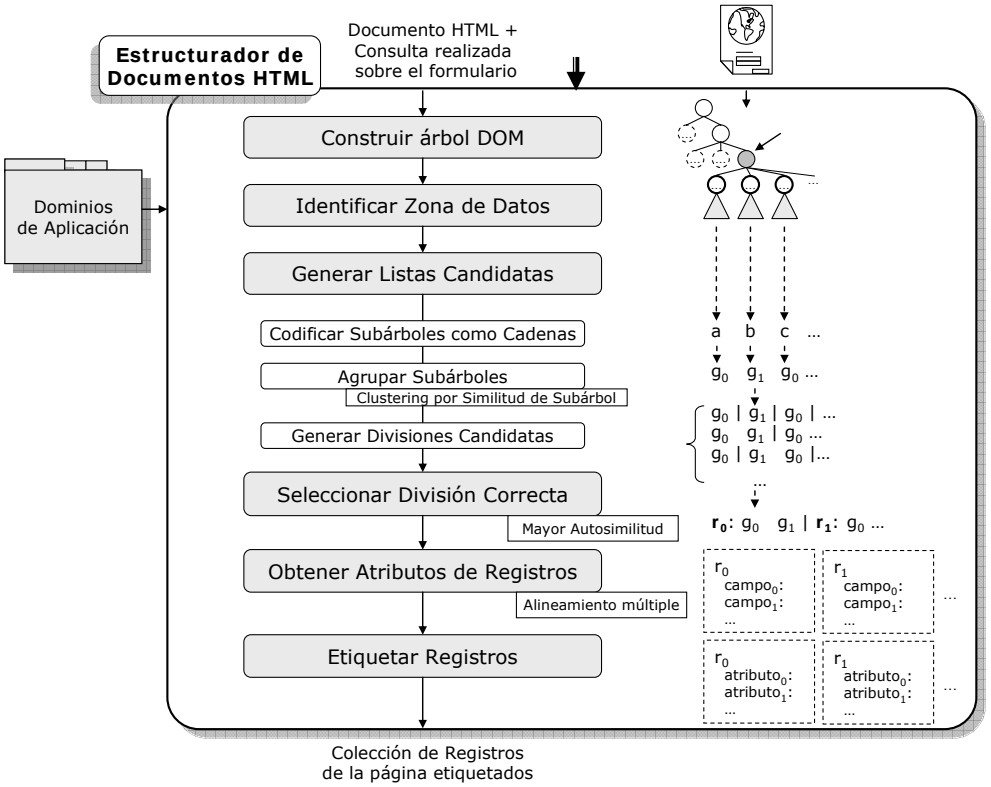


Figura 41 Arquitectura del componente de estructuración automática

El componente de estructuración automática recibe como entrada una página resultado de realizar una consulta contra un formulario web perteneciente a un dominio de aplicación relevante para el *crawler*, junto con las palabras clave de la consulta.

A partir del código fuente de la página HTML, se obtiene su representación interna como un árbol DOM y sobre esa representación en árbol DOM se identifica la región de la página que contiene los datos.

Una vez identificada la zona de datos, se genera un conjunto de registros candidatos. Para realizar las divisiones candidatas, se parte del nodo raíz de la zona de datos, agrupando los subárboles en función de su similitud, y generando una serie de divisiones candidatas en base a determinadas propiedades que serán descritas en el apartado III.5.1.3. Para el agrupamiento de los subárboles, se aplican técnicas de *clustering*.

La siguiente fase consiste en seleccionar la división de registros candidatos correcta. La idea básica en esta fase consiste en calcular la similitud media entre los registros que forman cada división candidata. Se escogerá la división candidata cuyos registros muestren una mayor similitud entre sí.

El siguiente paso es analizar los registros obtenidos para dividirlos en los atributos que los componen, y obtener el valor de cada atributo para cada registro.

El proceso finaliza etiquetando los atributos de los registros obtenidos, asociando los nombres de los atributos del dominio de aplicación con los que se corresponden.

Los siguientes apartados describen cada una de estas etapas en detalle.

III.5.3. LOCALIZACIÓN DE LA ZONA DE DATOS DE UNA PÁGINA

En este apartado se describe cómo localizar la región de datos de la página que contiene la lista de registros dominante.

De la propiedad 1 del apartado III.5.1.3 se deduce que el problema de localizar la región de datos en una página es equivalente a localizar el nodo padre común de los subárboles hermanos que forman los registros de datos. El subárbol que tenga como raíz ese nodo será la región de datos buscada. Por ejemplo, en el árbol DOM mostrado en la Figura 40 el nodo padre que debe ser descubierto es n_1.

Nuestro método para localizar la región conteniendo la lista dominante de registros en la página *P* consta de los siguientes pasos:

1. Sea *N* el conjunto compuesto por todos los nodos en el árbol DOM de *P*. A cada nodo $n_i \in N$, se le asignará una puntuación denominada p_i. Inicialmente, $\forall_{i=1..|N|} p_i = 0$.
2. Calcular *T*, como el conjunto de todos los nodos de texto de *N*.
3. Dividir *T* en subconjuntos $t_1, t_2, \ldots, t_m$ de tal forma que todos los nodos de texto con la misma ruta XPath desde la raíz en el árbol DOM, estén contenidos en el mismo t_i. Para calcular las rutas XPath desde la raíz, se ignoran los atributos de las diferentes etiquetas HTML.
4. Para cada par de nodos de texto pertenecientes al mismo grupo, calcular n_j como su nodo padre común más profundo en el árbol DOM, y añadir `1` a p_j (la puntuación de n_j).
5. Sea n_{max} el nodo que ha obtenido la puntuación más alta. El subárbol DOM que tiene n_{max} como raíz es la zona de datos seleccionada.

Una vez descrito el algoritmo para seleccionar la zona de datos de una página, a continuación se procede a su justificación. Primero, por definición, la región de datos destino contiene una lista de registros y cada registro de datos está compuesto de una serie de atributos. Por la propiedad 2 del apartado III.5.1.3 sabemos que todas las ocurrencias del mismo atributo tienen la misma ruta XPath desde la raíz. Por esta razón, el subárbol conteniendo la lista de registros dominante en la página típicamente contendrá más textos con la misma ruta desde la raíz que otras regiones. Además, dados dos nodos de texto con la misma ruta en el árbol DOM, pueden ocurrir las siguientes dos situaciones:

1. Por la propiedad 1, si los nodos de texto son ocurrencias de nodos de texto en diferentes registros (por ejemplo dos valores de datos del mismo atributo en diferentes registros), entonces su nodo padre común más profundo en el árbol DOM será el nodo raíz de la zona de datos que contiene todos los registros. De esta forma, cuando se considere ese par de nodos en el paso 4 del algoritmo, se incrementará la puntuación del nodo correcto. Por ejemplo, en la Figura 40 el padre común más profundo de los nodos d_1 y d_3 es n_1, el nodo raíz del subárbol que contiene la zona de datos completa.
2. Si los nodos de texto son ocurrencias de diferentes atributos del mismo registro, entonces en algunos casos su nodo padre común más profundo podría ser un nodo más profundo que el que estamos buscando y se incrementará la puntuación de un nodo incorrecto. Por ejemplo, en la Figura 40 el nodo padre común más profundo para d_1 y d_2 es n_2.

Por la propiedad 2 podemos inferir que habrá normalmente más ocurrencias del caso 1 y por tanto el algoritmo devolverá como resultado el nodo correcto. A continuación explicamos la razón para que esto sea así. Por ejemplo, consideremos un hipotético par de nodos de texto (t_{11}, t_{12}) que se corresponden con las ocurrencias de $atributo_1$ y $atributo_2$ en el registro $registro_1$. (t_{11}, t_{12}) es un par de nodos en el caso 2. Pero, por la propiedad 2, para cada registro $registro_i$ en el que aparezcan $atributo_1$ y $atributo_2$, habrá pares (t_{11}, t_{i1}), (t_{11}, t_{i2}), (t_{12}, t_{i1}), (t_{12}, t_{i2}), pertenecientes al caso 1.

Así pues, en ausencia de campos opcionales en el esquema, puede demostrarse fácilmente que habrá más pares en el caso 1. Cuando existen campos opcionales, también es sencillo ver que es todavía muy probable que así sea.

Este método tiende a encontrar la lista en la página con el mayor número de registros y el mayor número de atributos en cada registro. Cuando las páginas de las que queremos extraer datos han sido obtenidas tras ejecutar una consulta sobre un formulario web, normalmente estamos interesados en extraer registros de datos que constituyen respuestas a la consulta, incluso aunque no formen parte de la lista de registros más larga de la página (esto puede suceder si la consulta tiene pocos resultados). En las aplicaciones de *crawling* dirigido, la consulta ejecutada es conocida, por lo que esta información puede ser utilizada para refinar el método descrito. La idea es muy sencilla: en el paso 2 del algoritmo, en lugar de utilizar todos los nodos de texto del árbol DOM de la página, se utilizarán sólo aquellos nodos de texto que contengan valores usados en la consulta. Por ejemplo, supóngase que la página mostrada en la Figura 38 fue obtenida como resultado de realizar la consulta que podríamos expresar como (TÍTULO contiene 'java') AND (FORMATO igualA 'paperback'). En este caso, los únicos nodos considerados en el paso 2 serían los marcados con '*' en la Figura 40.

La razón para este refinamiento es clara: los valores utilizados en la consulta típicamente aparecerán con mucha mayor probabilidad en la lista de resultados que en otras listas de la página.

III.5.4. DIVISIÓN DE LA LISTA DE DATOS EN REGISTROS

Esta sección describe las técnicas propuestas para segmentar la región de datos en fragmentos, cada uno de ellos conteniendo al menos un registro de datos.

El método puede ser dividido en las siguientes fases:

- Generar un conjunto de listas de registros candidatos. Cada lista de registros candidatos propondrá una división particular de la región de datos en registros.
- Elegir la mejor lista de registros candidatos. El método propuesto está basado en calcular una medida de auto-similitud entre los registros de cada la lista de registros candidata. Se escogerá la división candidata cuyos registros muestren una auto-similitud mayor.

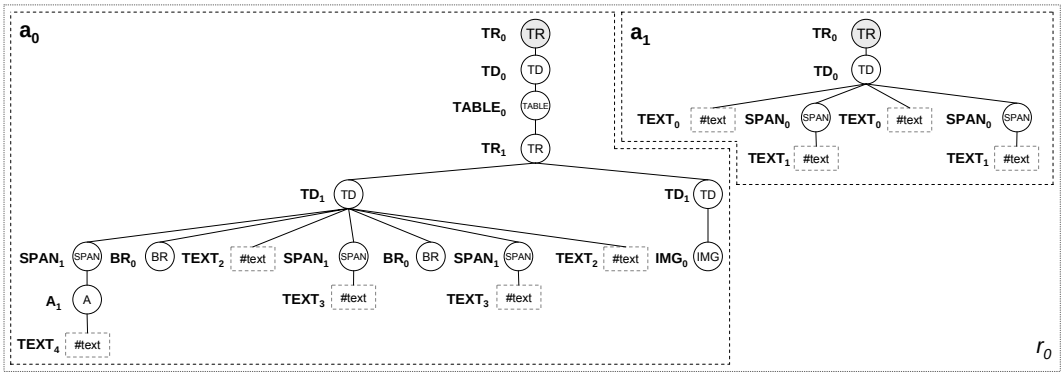
Las secciones III.5.4.2 y III.5.4.3 describen en detalle cada una de estas fases. Ambas tareas necesitan un mecanismo para estimar la similitud entre dos secuencias de árboles hermanos consecutivos en el árbol DOM de una página. El método utilizado para esto se describe en el apartado III.5.4.1.

III.5.4.1. MEDIDA DE SIMILITUD DE DISTANCIA DE EDICIÓN

Para calcular las medidas de similitud se utilizan técnicas basadas en algoritmos de distancia de edición entre cadenas. Concretamente, para calcular la similitud *se* entre dos secuencias de subárboles hermanos consecutivos denominados r_i y r_j en el árbol DOM de una página, se llevan a cabo los siguientes pasos:

1. Representar r_i y r_j como cadenas, que se denominarán c_i y c_j respectivamente. Esto se realiza de la siguiente forma:
   a. Sustituir cada nodo de texto por una etiqueta especial llamada *texto*.
   b. Atravesar cada subárbol en profundidad y, para cada nodo, generar un carácter en la cadena. Se asigna un carácter diferente a cada etiqueta con una ruta diferente desde la raíz del árbol DOM. La Figura 42 muestra las cadenas c_0 y c_1 obtenidas para los registros r_0 y r_1 de la Figura 40, respectivamente.
2. Calcular la similitud entre r_i y r_j, denotada como $se(r_i, r_j)$, como la distancia de edición de cadenas entre c_i y c_j, denotada como $de(c_i, c_j)$. La distancia de edición entre c_i y c_j se define como el número de operaciones de edición (cada operación afecta sólo a un carácter cada vez) que son necesarias para transformar una cadena en otra. Para calcular la distancia de edición entre cadenas, se utiliza una variante del algoritmo de Levenshtein [Levenshtein66], que sólo permite inserciones y borrados (otras implementaciones también permiten operaciones de sustitución). Para obtener un valor de similitud entre `0` y `1`, el resultado se normaliza dividiéndolo entre $(longitud(c_i) + longitud(c_j))$, y el resultado obtenido se resta de `1` (III.5-1). En el ejemplo de la Figura 42, la similitud entre r_0 y r_1 es 1-(2 / (26 + 28)) = 0.96.

$$se(r_i, r_j) = 1 - \frac{de(c_i, c_j)}{longitud(c_i) + longitud(c_j)} \qquad \textbf{(III.5-1)}$$



c_0 → a_0: **TR$_0$ TD$_0$ TABLE$_0$ TR$_1$ TD$_1$ SPAN$_1$ A$_1$ TEXT$_4$ BR$_0$ TEXT$_2$ SPAN$_1$ TEXT$_3$ BR$_0$ SPAN$_1$ TEXT$_3$ TEXT$_2$ TD$_1$ IMG$_0$**
a_1: **TR$_0$ TD$_0$ TEXT$_0$ SPAN$_0$ TEXT$_1$ TEXT$_0$ SPAN$_0$ TEXT$_1$**

c_1 → a_2: **TR$_0$ TD$_0$ TABLE$_0$ TR$_1$ TD$_1$ SPAN$_1$ A$_1$ TEXT$_4$ BR$_0$ TEXT$_2$ SPAN$_1$ TEXT$_3$ BR$_0$ SPAN$_1$ TEXT$_3$ TEXT$_2$ TD$_1$ IMG$_0$**
a_3: **TR$_0$ TD$_0$ TEXT$_0$ SPAN$_0$ TEXT$_1$**
a_4: **TR$_0$ TD$_0$ TEXT$_0$ SPAN$_0$ TEXT$_1$** longitud(c_0) = 26 longitud(c_1) = 28

$de(r_0, r_1) = de(c_0, c_1) = 2$

Figura 42 Cadenas obtenidas para los registros r_0 y r_1 de la Figura 40

III.5.4.2. GENERACIÓN DE LISTAS CANDIDATAS DE REGISTROS

Este apartado describe cómo generar un conjunto de listas de registros candidatas dentro de la región de datos previamente seleccionada. Cada lista de registros candidata propone una división concreta de la región de datos en registros.

Por la propiedad 1, se asume que todos los registros están compuestos de uno o varios subárboles hermanos consecutivos, que son descendientes directos del nodo seleccionado como raíz de la región de datos.

Partiendo de esa propiedad, se puede generar una lista de registros candidata para cada posible división en registros que la satisfaga. Sin embargo, el número de combinaciones posibles sería muy elevado: si el número de subárboles es *n*, el número posible de divisiones verificando la propiedad 1 es 2^{n-1} (resaltar que registros diferentes en la misma lista pueden estar formados por un número diferente de subárboles, como por ejemplo r_0 y r_1 en la Figura 40). En algunas fuentes, *n* puede ser bajo, pero en otras puede alcanzar valores del orden de cientos (por ejemplo, una fuente con 25 registros compuestos cada uno de ellos en media de 4 subárboles). Por lo tanto, la aproximación exhaustiva no es factible. El resto de este apartado explica cómo resolver este problema. El método consta de dos fases:

1. Agrupar los subárboles de acuerdo a su similitud (fase de *clustering*).
2. Utilizar los grupos obtenidos en el paso 1 para generar las divisiones de registros candidatas.

Las siguientes subsecciones respectivamente detallan cada una de estas fases.

III.5.4.2.1. AGRUPANDO LOS SUBÁRBOLES

Para agrupar los subárboles de acuerdo a sus similitudes, se utiliza un proceso basado en *clustering* siguiendo los pasos que se describen a continuación:

1. Considerar el conjunto $\{a_1, ..., a_n\}$ de todos los subárboles que son hijos directos del nodo seleccionado como raíz de la región de datos. Cada a_i puede representarse como una cadena utilizando el método descrito en el apartado III.5.4.1. Denominaremos a esas cadenas como $c_1, ..., c_n$.
2. Calcular la matriz de similitudes. Esta es una matriz *nxn* donde la posición *(i,j)* (denotada como m_{ij}) se obtiene como $se(a_i,a_j)$, la similitud de distancia de edición entre a_i y a_j.
3. Se define la *similitud de columna* entre a_i y a_j, denotada como $sc(a_i, a_j)$, como la inversa del error absoluto medio entre las columnas correspondientes a a_i y a_j en la matriz de similitud (III.5-2). Por lo tanto, para considerar dos subárboles como similares, la medida de *similitud de columna* requiere que sus columnas en la matriz de similitudes sean muy parecidas. Esto significa que los dos subárboles tienen que tener distancias de edición parecidas con el resto de subárboles del conjunto para ser considerados similares. En los experimentos realizados, se ha detectado que la similitud de columna es una medida más robusta para estimar la similitud entre a_i y a_j en el proceso de *clustering*, que la utilización directa de $se(a_i,a_j)$.

$$sc(a_i, a_j) = 1 - \sum_{k=1..n} \frac{|m_{ik} - m_{jk}|}{n} \qquad \textbf{(III.5-2)}$$

4. El siguiente paso es aplicar un algoritmo de *clustering bottom-up* [Chakrabarti03] para agrupar los subárboles. La idea básica de este algoritmo es comenzar con un grupo para cada elemento y sucesivamente combinarlos para obtener nuevos grupos en los que la similitud entre sus elementos sea alta, compactándolos hasta obtener tantos grupos como se deseen.

La Figura 43 muestra el pseudocódigo para el algoritmo de *clustering bottom-up*. La similitud entre elementos de un conjunto Φ se estima utilizando la medida de auto-similitud. La auto-similitud de un conjunto Φ se denota como $s(\Phi)$ y se calcula como la media de las similitudes entre cada par de elementos en el conjunto (III.5-3).

$$s(\Phi) = \frac{1}{\binom{\Phi}{2}} \sum_{a_i, a_j \in \Phi} sc(a_i, a_j) = \frac{2}{|\Phi|(|\Phi| - 1)} \sum_{a_i, a_j \in \Phi} sc(a_i, a_j) \qquad \textbf{(III.5-3)}$$

1. Considerar cada subárbol *a* en un grupo diferente *{a}*
2. Sea *G* el conjunto de todos los grupos
3. Sea Ω_g el umbral de similitud de grupo y
   Ω_e el umbral de similitud entre elementos de un grupo.
4. *MIENTRAS |G| > 1* HACER
   - 4.1 Elegir $\Gamma, \Delta \in G$, un par de grupos que maximicen la medida de autosimilitud $s(\Gamma \cup \Delta)$ *(*ver ecuación III.5-3). El conjunto $\Gamma \cup \Delta$ *debe verificar:*
     - a) $s(\Gamma \cup \Delta) > \Omega_g$
     - b) $\forall i \in \Gamma \cup \Delta,\ j \in \Gamma \cup \Delta,\ sc(i, j) > \Omega_e$
   - 4.2 Si ningún par verifica las condiciones anteriores, TERMINAR
   - 4.3 Eliminar *Γ* y *Δ* de *G*
   - 4.4 Sea $\Phi = \Gamma \cup \Delta$
   - 4.5 Insertar Φ en *G*
5. *fin MIENTRAS*

Figura 43 Pseudocódigo para el algoritmo de *clustering bottom-up*

Se utiliza la similitud de columna como la medida de similitud entre a_i y a_j. Para permitir que se forme un nuevo grupo, debe verificar dos umbrales:

- La auto-similitud global del grupo debe ser superior al umbral de auto-similitud Ω_g. En la implementación actual se ha establecido este umbral a 0.9.
- La similitud de columna entre cada par de elemento del grupo debe ser superior al umbral de similitud entre elementos Ω_e. Este umbral se utiliza para evitar la creación de grupos que, aunque muestren un alto nivel de auto-similitud media entre todos sus elementos, contienen algunos elementos que son muy diferentes. En la implementación actual se ha establecido este umbral a 0.8.

La Figura 44 muestra el resultado de aplicar el algoritmo de *clustering* a la página de ejemplo de la Figura 40. Como puede verse, se crean tres grupos a los que se ha denominado g_0, g_1 y g_2.

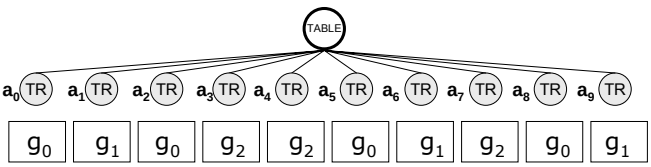


Figura 44 Resultado de aplicar el algoritmo de *clustering* a la página de ejemplo de la Figura 40

III.5.4.2.2. GENERANDO LAS DIVISIONES CANDIDATAS EN REGISTROS

Este apartado describe el método utilizado para generar el conjunto de divisiones candidatas en registros.

Primero se asigna un identificador a cada uno de los grupos generados. A continuación se construye una secuencia enumerando en orden los subárboles de la región de datos, y representando cada subárbol con el identificador del grupo al que pertenece como se muestra en la Figura 44.

La región de datos puede contener, tanto al principio como al final, algunos subárboles que no son en realidad parte de los datos, sino que constituyen información auxiliar. Por ejemplo, estos subárboles pueden contener información sobre el número de resultados o formularios web para refinar la consulta o para navegar a otros intervalos de resultados. Estos subárboles típicamente estarán solos en un grupo, debido a que no existen otros subárboles similares en la región de datos. Por lo tanto, como paso previo a la generación de divisiones en registros candidatos, se pre-procesa la cadena desde el principio (y desde el final) para eliminar *tokens* hasta encontrar el primer identificador de grupo que aparece más de una vez en la secuencia. En algunos casos, este pre-procesamiento puede no eliminar todos los subárboles de este tipo. Los restantes serán eliminados normalmente en la fase de extracción de atributos de los registros de datos, como se describe en el apartado III.5.5.

Una vez ha finalizado la fase de pre-procesamiento, se procede a generar la lista de divisiones en registros candidatas. Por la propiedad 1 del apartado III.5.1.3, se deduce que cada registro está formado por una lista de subárboles consecutivos (es decir, caracteres en la cadena). Del modelo de página utilizado, se deduce que los registros están codificados de forma consistente. Por lo tanto, la cadena tenderá a estar formada por una secuencia repetitiva de identificadores de grupos, cada secuencia correspondiendo a un registro de datos. Como hay campos que pueden ser opcionales en los registros extraídos, la secuencia para un registro puede ser ligeramente diferente de la secuencia correspondiente a otro registro. Sin embargo, se asumirá que cada registro siempre comienza o termina con un subárbol que pertenece al mismo grupo (es decir, todos los registros de datos siempre empiezan o terminan de la misma forma). Para ello nos basamos en la observación de las siguientes heurísticas:

- En muchas fuentes, los registros están delimitados visualmente de una forma no ambigua para mejorar la claridad para el usuario humano. Este delimitador está presente antes o después de cada uno de los registros.
- Cuando no hay un delimitador explícito entre registros, los primeros campos de datos de los registros suelen ser campos obligatorios, que aparecen en todos los registros (en el ejemplo de la Figura 40 el delimitador puede ser el fragmento que se corresponde con el campo TÍTULO que aparece en cada registro).

En base a las observaciones anteriores, se generan las siguientes listas de registros candidatas:

- Para cada grupo g_i, $i=1..k$, se generan dos divisiones candidatas: una asumiendo que cada registro comienza con el grupo g_i y otras asumiendo que cada registro finaliza

con el grupo g_i. Por ejemplo, en la Figura 45 se muestran las divisiones en registros candidatas obtenidas para el ejemplo de la Figura 40.

- Adicionalmente se añade una división en registros candidata considerando que cada registro está formado exactamente por un único subárbol.

Esto reduce el número de divisiones en registros candidatas de 2^{n-1}, donde n es el número de subárboles, a $1+2k$, donde k es el número de grupos generados, haciendo posible la evaluación de cada lista candidata para seleccionar la mejor.

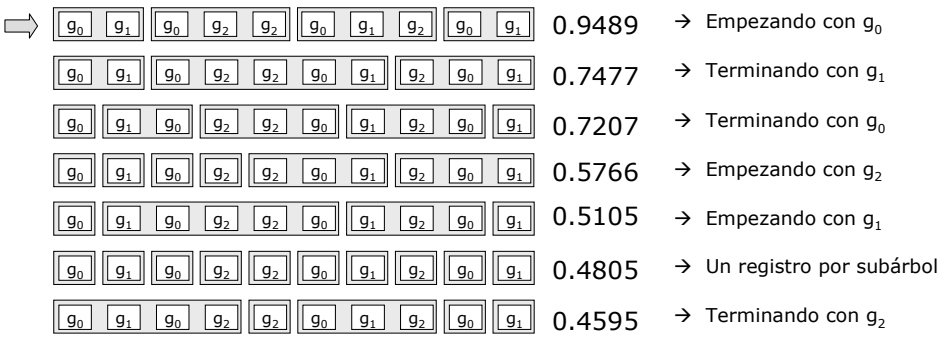


Figura 45 Conjunto de listas de registros candidatas para el ejemplo de la Figura 40

III.5.4.3. SELECCIÓN DE LA LISTA DE REGISTROS CORRECTA

Para seleccionar la lista de registros candidata correcta, se utiliza la observación de que los registros de una lista tienden a ser similares unos a otros. Esto se deriva fácilmente del modelo de página descrito en el apartado III.5.1.2. Por lo tanto, se seleccionará la lista de registros candidata que muestre una mayor auto-similitud.

Como ya se ha comentado en secciones previas, cada lista candidata está compuesta por una lista de registros y cada registro es una secuencia de subárboles hermanos consecutivos.

Dada una lista candidata compuesta de registros $<r_1, \ldots, r_n>$, calculamos su auto-similitud como la media ponderada de las similitudes de distancia de edición entre cada par de registros de la lista. La contribución de cada par a la media se pondera por la longitud de los registros comparados, como se muestra en la ecuación (III.5-4).

$$\frac{\sum_{i=1..n, j=1..n, i\neq j} se(r_i, r_j)\left(longitud(r_i)+longitud(r_j)\right)}{\sum_{i=1..n, j=1..n, i\neq j} longitud(r_i)+longitud(r_j)} \qquad \textbf{(III.5-4)}$$

Por ejemplo, en la Figura 45, la primera división en registros candidata es la que muestra mayor auto-similitud.

III.5.5. EXTRAER ATRIBUTOS DE REGISTROS

Este apartado describe las técnicas utilizadas para la extracción de los valores de los atributos de los registros de datos identificados en el apartado previo.

La idea básica consiste en transformar cada registro de datos de la lista en una cadena, utilizando el método descrito en el apartado III.5.4.1, para después utilizar técnicas de alineamiento de cadenas para identificar los atributos de cada registro. Un alineamiento entre dos cadenas es una correspondencia entre los caracteres en una cadena y los de la otra, de tal forma que se minimice la distancia de edición entre las dos cadenas. Podría haber más de un alineamiento óptimo entre dos cadenas. En ese caso, se seleccionará uno cualquiera de ellos.

Por ejemplo, la Figura 46 muestra un fragmento del alineamiento entre las cadenas que representan los registros del ejemplo de la Figura 40. Cada *token* de texto alineado se corresponde aproximadamente con un atributo del registro. Destacar que para obtener el valor actual de un atributo puede ser necesario eliminar prefijos o sufijos comunes que aparecen en todas las ocurrencias de un atributo. Por ejemplo, en la página de la Figura 40, para obtener el valor del atributo PRECIO se detectaría y eliminaría el sufijo común '€'. Adicionalmente, los nodos de texto alineados que tienen el mismo valor en todos los registros (e.g. 'Buy new:', 'Price used:') se consideran como etiquetas en lugar de valores de atributos y por tanto no aparecerán en la salida.

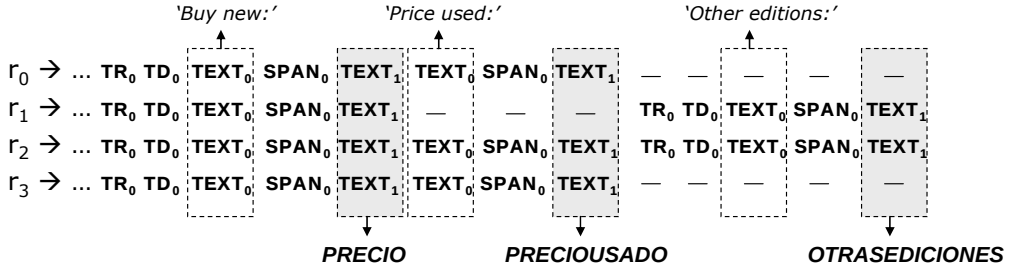


Figura 46 Alineamiento entre cadenas representando los registros para el ejemplo de la Figura 40

Para alcanzar el objetivo, no es suficiente con alinear dos registros. Es necesario alinear todos los registros. Sin embargo, los algoritmos de alineamiento múltiple de cadenas óptimos tienen una complejidad $O(n^k)$. Por lo tanto, se necesita un algoritmo que permita una aproximación al resultado deseado. Se han propuesto diferentes algoritmos en la literatura para esta tarea [Notredame02] [GBS92]. Nuestro prototipo utiliza una variante del algoritmo de aproximación '*center star*' [GBS92], que es también similar a una variante utilizada en [ZL06] (aunque dicho trabajo utiliza alineamiento de árboles en lugar de alineamiento de cadenas). El funcionamiento del algoritmo sigue los pasos que se describen a continuación:

1. Elegir la cadena de mayor longitud como la cadena *maestra*, *m*.
2. Inicialmente, el conjunto de cadenas todavía no alineadas, *C*, contiene todas las cadenas excepto la maestra, *m*.

3. Para cada cadena *c* $\in$ *C:*
    a. Alinear *c* con *m*.
    b. Si hay un único alineamiento óptimo entre *c* y *m*:
        i. Si el alineamiento hace corresponder una posición nula de *m* con un carácter de *c*, entonces se añade el carácter a *m*, reemplazando la posición nula.
        ii. Eliminar *c* de *C*.
4. Repetir el paso 3 hasta que el conjunto *C* esté vacío o la cadena maestra no cambie.

El paso clave del algoritmo es 3.b.i, donde se alinea cada cadena con la cadena maestra y eventualmente se utiliza para extenderla. La Figura 47 muestra un ejemplo de este paso. Se parte de la cadena maestra 'abcbc' ante la que se presenta una nueva cadena 'abdcb'. El alineamiento óptimo entre ambas cadenas hace corresponder la tercera posición de la cadena, 'd', con un valor nulo en el maestro. El nuevo maestro se obtiene insertando ese carácter en la posición determinada por el alineamiento, generándose la cadena 'abdcbc'.

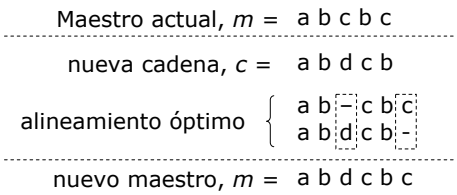


Figura 47 Ejemplo de alineamiento de cadenas con la cadena maestra

Como se mencionó en el apartado III.5.4.2, la región de datos puede contener, tanto al principio como al final, algunos subárboles que no forman realmente parte de los datos, sino que constituyen información auxiliar (e.g. información sobre el número de resultados, formularios web para navegar a otros intervalos de resultados, etc.). Por esta razón, antes de generar la división en registros candidata, el sistema lleva a cabo un primer intento para eliminar esos subárboles. Sin embargo, puede ocurrir que algunos subárboles todavía estén presentes al principio del primer registro o al final del último. Por lo tanto, tras el proceso de alineamiento múltiple, si el primer registro (respectivamente el último) empieza (respectivamente termina) con una secuencia de caracteres que no están alineados con ningún otro registro, entonces estos caracteres se eliminan.

III.5.6. ETIQUETACIÓN

Como resultado del proceso de estructuración automática, se obtiene una colección de registros con los diferentes atributos identificados. Para los registros obtenidos a partir de una misma página, los *tokens* que ocupan la misma posición se corresponden con valores para el mismo atributo del dominio de aplicación, pero no se sabe a cuál de ellos.

Tanto para posibilitar la realización de búsquedas estructuradas sobre estos registros como para permitir integrar registros obtenidos de diferentes fuentes, es conveniente conocer con qué atributo del dominio se corresponde cada valor atómico identificado. El proceso de etiquetación es el responsable de realizar esta asociación.

Como el proceso de estructuración se encuentra integrado en un componente dentro de la arquitectura de *crawling* dirigido, es posible utilizar metainformación asociada al dominio para mejorar los resultados de la etiquetación.

El algoritmo de etiquetación procede en dos fases. En la primera fase obtiene metainformación a partir de los valores que ocupan la misma posición en todos los registros, y en la segunda determina el nombre de atributo que se corresponde con cada una de las posiciones. Es importante tener en cuenta que los campos estructurados se encuentran ordenados respecto a su aparición relativa en la página, y que el proceso de estructuración identificó como campos tanto los valores constantes de la plantilla generadora de la página de resultados como los valores de los registros propiamente dichos.

En la primera fase se obtiene la siguiente información para cada campo del esquema global del registro – el identificado tras la fase de alineamiento múltiple –:

- Obligatoriedad: Un campo se considera obligatorio cuando presenta valores para todos los registros extraídos.
- Relevancia: Se considera irrelevante un campo si aparece en menos del 20% de los registros. Esos campos no se consideran para el proceso de asociación.
- Tipo del campo: En base a expresiones regulares se obtiene el tipo del campo, clasificándolo en uno de los siguientes: Numérico, Fecha, Moneda o Texto.
- Constante: Un campo se considera constante cuando el valor de ese campo para todos los registros es el mismo. Los campos identificados como constantes se consideran también no relevantes.
- Separador: Un campo se considera separador si el valor de ese campo para todos los elementos está formado sólo por caracteres separadores. Los siguientes caracteres se consideran separadores: ‘ .-:,;\n\t(){}$%€£’. Los campos identificados como separadores se consideran también no relevantes.
- Prefijo y sufijo común para los valores de un campo en todos los registros. El prefijo y el sufijo tienen que ser una palabra completa, es decir, debe estar separada del resto por caracteres separadores.

En la segunda fase, en base a la metainformación recolectada, se recorren todos los campos del esquema. En primer lugar:

- Si un campo es Constante, entonces su valor se considerará como una etiqueta candidata para el siguiente valor no constante del alineamiento. Por ejemplo, el campo con valor constante ‘Buy new:’ produciría una etiqueta candidata para el siguiente campo no constante (que, en el ejemplo utilizado, se corresponde con el precio del libro).

- Si el campo no es constante y tiene un prefijo asociado, entonces se asocia dicho prefijo como etiqueta candidata para el campo. Adicionalmente, los prefijos y sufijos comunes son eliminados de los valores de los elementos.

Una vez obtenidas las etiquetas candidatas para cada campo, se realiza un proceso similar al de asociación de campos de formularios y atributos del dominio (ver apartado III.4.2), aplicando similitud textual entre esas etiquetas y los nombres y alias de atributos en el dominio de aplicación.

A continuación se utiliza la información de las consultas de ejemplo incluidas en el dominio de aplicación. Las palabras clave utilizadas para cada atributo en la consulta son conocidas, y dichas palabras deben aparecer en la mayor parte de los valores del campo que se corresponda con dicho atributo. De esta forma, es posible identificar los campos de los registros que se corresponden con dichos atributos. Por ejemplo, si se ha realizado una consulta por la palabra clave '`java`' en el campo `TÍTULO` del formulario, dicha palabra aparecerá en los valores del campo de los registros que se corresponde con el título de los libros.

Como resultado se obtendrá la lista etiquetada de registros. A los campos que no han podido ser asociados a un atributo del dominio se le asocian nombres generados de forma automática, que no permitirán añadir semántica pero permiten no perder la información que representan.

IV. EXPERIENCIA OBTENIDA DEL USO DEL SISTEMA

Este capítulo describe la validación experimental realizada de los principales algoritmos y técnicas propuestos en esta tesis doctoral. La estructura del capítulo es la siguiente. En la sección IV.1 se detalla el conjunto de experimentos realizados sobre un prototipo implementado de acuerdo a las técnicas presentadas en el capítulo III, junto con los resultados obtenidos. En la sección IV.2 se comentan brevemente algunas aplicaciones de las técnicas propuestas para otras tareas diferentes al *crawling* dirigido de la Web Oculta y en la sección IV.3, se describen aplicaciones industriales en las que se han utilizado. Finalmente, en la sección IV.4 se realiza una evaluación del cumplimiento de los objetivos, expuestos en la sección I.2, tras los resultados obtenidos en las secciones anteriores.

IV.1. EXPERIMENTOS

Para comprobar la validez de las técnicas se ha implementado un prototipo de *crawling* de la Web Oculta conforme a ellas (llamado DeepBot) y se han realizado una serie de experimentos que se detallan en esta sección.

Se han realizado experimentos en varias etapas, para probar de forma independiente las técnicas definidas para los principales módulos de la arquitectura propuesta. En la primera de ellas se han diseñado una serie de experimentos para validar las técnicas de *crawling* de la Web Oculta. En la segunda se han diseñado una serie de experimentos para validar las técnicas de estructuración automática.

IV.1.1. EXPERIENCIA EN *CRAWLING DE LA* WEB OCULTA

Para evaluar la efectividad de las técnicas definidas para establecer la relevancia y aprender a consultar formularios de acceso a la Web Oculta, se ha definido un conjunto de experimentos en tres dominios de aplicación diferentes. Los dominios seleccionados son tiendas electrónicas de libros, música y películas.

Para la definición de cada uno de los dominios, se han seleccionado 10 sitios web de forma aleatoria de la categoría correspondiente del directorio de búsqueda de Yahoo![20]. A partir de esos sitios, se han definido los atributos y alias del dominio de aplicación que guía el proceso de *crawling*. Los índices de especificidad y el umbral de relevancia de los atributos del dominio y del dominio respectivamente, fueron seleccionados de forma manual a partir de la experiencia obtenida recorriendo esos sitios. Las definiciones de dominio resultantes se muestran en la Figura 48.

Dominio de Aplicación 'Libros'

Atributos

Nombre	Alias	e_l (índice de especificidad)
TÍTULO	*'title', 'title of book'*	0.6
AUTOR	*'author', 'author's name'*	0.7
PUBLICADOPOR	*'publisher'*	0.8
ISBN		0.95
FECHAPUBLICACIÓN	*'publication date'*	0.7
CATEGORÍA	*'subject', 'section', 'category', 'department', 'subject Category'*	0.05
FORMATO	*'format', 'binding type'*	0.25
PRECIO	*'price'*	0.05

Umbral de relevancia: $\mu = 0.9$

Dominio de Aplicación 'Música'

Atributos

Nombre	Alias	e_l (índice de especificidad)
ARTISTA	*'artist', 'artist name', 'composer/author/artist'*	0.6
TÍTULO	*'song', 'soundtrack title','song title'*	0.95
ÁLBUM	*'album', 'album title'*	0.95
DISCOGRÁFICA	*'label', 'vendor'*	0.8
GÉNERO	*'genre', 'style'*	0.05
FORMATO	*'format', 'media type', 'product type', 'item types'*	0.25
PRECIO	*'price'*	0.05

Umbral de relevancia: $\mu = 0.9$

Dominio de Aplicación 'Películas'

Atributos

Nombre	Alias	e_l (índice de especificidad)
TÍTULO	*'movie title'*	0.6
LEYENDA	*'legend'*	0.7
PROTAGONISTAS	'starring', 'star', 'actor', 'cast', 'featuring (cast/crew)', 'cast name', 'artisties'	0.7
DIRECTOR		0.7
PRODUCTOR	*'producer'*	0.7
EDITOR		0.7
BANDA SONORA	*'sound', 'music'*	0.7
FORMATO	*'format', 'media'*	0.05
GÉNERO	*'genre', 'movie type', 'category'*	0.05
PRECIO	*'price'*	0.05

Umbral de relevancia: $\mu = 0.9$

Figura 48 Definiciones de dominios utilizados en las pruebas

[20] http://dir.yahoo.com

Una vez creados los dominios, se utilizó el prototipo implementado, para recorrer 20 sitios web de las mismas categorías del directorio de Yahoo!. Los sitios web visitados para cada dominio se muestran en la Tabla 1. Los utilizados para definir los atributos y alias se agrupan en el denominado conjunto de entrenamiento, mientras que los restantes sitios se agrupan en el conjunto de pruebas.

Tabla 1 Conjuntos de entrenamiento y de pruebas para los dominios de tiendas electrónicas de libros, música y películas

► Fuentes ▼ Dominios	***Entrenamiento (C_1)***	***Pruebas (C_2)***
Libros	Amazon.com - *http://www.amazon.com* Barnes&Noble - *http://www.barnesandnoble.com* Bolen - *http://www.bolen.bc.ca* BookFinder4U.com - *http://www.bookfinder4u.com* Cody's Books - *http://www.codysbooks.com* Daedalus Books&Music - *http://www.daedalusbooks.com* Dymocks Booksellers - *http://www.dymocks.com.au* eCampus.com - *http://www.ecampus.com* Powell's Books - *http://www.powells.com* Tattered Cover Bookstore - *http://www.tatteredcover.com*	Blackwell's Bookshop - *http://bookshop.blackwell.co.uk* Globe Pequot Press - *http://www.globepequot.com* Green Apple Books&Music-*http://www.greenapplebooks.com* Northshire Bookstore - *http://www.northshire.com* Oxbow Books - *http://www.oxbowbooks.com* Strand Book Store - *http://www.strandbooks.com* The American Book Center - *http://www.abc.nl* The Book Pl@ce - *http://www.thebookplace.com* The Scholar's Bookshelf - *http://www.scholarsbookshelf.com* Thomson Gale - *http://www.gale.com*
Música	A&B Sound - *http://catalog2.absound.ca* Amazon.com Music - *http://www.amazon.com* AudibleFaith – *http://www.audiblefaith.com* Barnes&Noble Music - *http://www.barnesandnoble.com* CD Quest - *http://www.cdquest.com* Collectors' Choice Music - *http://www.ccmusic.com* Rough Trade - *http://www.roughtrade.com* Sam Goody - *http://www.samgoody.com* Schott Musik International – *http://www.schott-music.com* Tower Records - *http://www.towerrecords.com*	Buy Cd - *http://www.buycd.com* ClassicTrax.co.uk - *http://www.classictrax.ltd.uk* Cyber Music Surplus - *http://www.cybermusicsurplus.com* Ladyslipper Music - *http://www.ladyslipper.org* Looney Tunes - *http://www.looneytunescds.com* Mojo Sounds - *http://www.mojosounds.com* Musica Obscura - *http://www.musicaobscura.com* MyMusic.com - *http://www.mymusic.com* ProMusicFind.com - *http://www.promusicfind.com* Record Exchange - *http://www.buymusichere.net*
Películas	Amazon.com DVD – *http://www.amazon.com* BollyVista.com - *http://www.bollyvista.com* Docuseek - *http://www.docuseek.com* GlobeAndMail.com - *http://www.theglobeandmail.com* Half.com - *http://www.half.ebay.com* Kansas.com - *http://ae.kansas.com* Movie Tickets.com - *http://www.movietickets.com* Saregama - *http://hamaracd.com* The Spinning Image - *http://www.thespinningimage.co.uk/* Yahoo! Movies - *http://movies.yahoo.com*	Bestvideobuys - *http://www.bestwebbuys.com* Blockbuster.com - *http://www.blockbuster.com* Blockbuster UK - *http://www.blockbuster.co.uk* BrickFilms - *http://www.brickfilms.com* Fandango - *http://www.fandango.com* Human Rights Film Directory - *http://db.lib.washington.edu* IGN.COM - *http://search.ign.com* Intelliflix - *http://www.intelliflix.com* Sensasian.com - *http://sensasian.com* Video*ezy - *http://www.videoezy.com.au*

Una primera medida que se obtuvo fue el número de sitios que utilizaban lenguajes de *script* o bien para la gestión de los formularios de consulta o bien en las páginas que era necesario recorrer para alcanzar el formulario. Estos sitios no hubiesen podido ser tratados por sistemas que no tuviesen en cuenta las complejidades de acceso a la web oculta de lado cliente. Los porcentajes se muestran en la Tabla 2.

Tabla 2 Porcentaje de utilización de lenguajes de *script* en los dominios considerados

Libros	Música	Películas	Total
20%	15%	25%	20%

A continuación, para validar el nivel de precisión en los resultados obtenidos en el tratamiento de los formularios, se analizaron los sitios web de forma manual y se comparó el resultado obtenido de las observaciones realizadas contra los obtenidos por DeepBot. Se realizaron validaciones en distintas fases del proceso:

- ***Fase 1***: Asociación de textos con campos del formulario.
- ***Fase 2***: Asociación de campos del formulario con atributos de dominio.

- ***Fase 3***: Relevancia de un formulario para un dominio.
- ***Fase 4***: Ejecución de consultas sobre los formularios relevantes.

Para cuantificar los resultados se hace uso de métricas estándar de recuperación de información [BR99]: *precisión*, *alcance (recall)* y *media armónica* (F_1). La precisión representa el número de asociaciones correctas reconocidas por el sistema respecto al número total de asociaciones reconocidas por el sistema. El alcance indica el número de asociaciones correctas reconocidas por el sistema respecto al número total de asociaciones. La media armónica representa una medida de compromiso entre precisión y alcance. El valor de la media armónica será mayor, cuanto mayor sea el valor de precisión y alcance.

Las métricas definidas para medir el rendimiento del sistema utilizan las siguientes variables:

- $ATextoCampo_{DeepBot}$: representa el conjunto de asociaciones entre textos y campos del formulario descubiertas por DeepBot.
- $ATextoCampo_{Real}$: representa el conjunto de asociaciones entre textos y campos del formulario establecidas durante el análisis manual.
- $ACampoAtributo_{DeepBot}$: representa el conjunto de asociaciones entre campos del formulario y atributos del dominio descubiertas por DeepBot.
- $ACampoAtributo_{Real}$: representa el conjunto de asociaciones entre campos del formulario y atributos del dominio establecidas durante el análisis manual.
- $AFormularioDominio_{DeepBot}$: representa el conjunto de asociaciones entre formularios y dominios descubiertas por DeepBot.
- $AFormularioDominio_{Real}$: representa el conjunto de asociaciones entre formularios y dominios establecidas durante el análisis manual.
- $FormulariosInvocados_{DeepBot}$: representa el conjunto de formularios exitosamente invocados por DeepBot.

En base a estas variables, definimos las siguientes métricas:

- Métricas para asociación de textos y campos del formulario:

$$\mathrm{Precisión}_{ATextoCampo} := \frac{\left|ATextoCampo_{DeepBot} \cap ATextoCampo_{Real}\right|}{\left|\mathrm{ATextoCampo}_{DeepBot}\right|} \qquad \textbf{(IV.1-1)}$$

$$Alcance_{ATextoCampo} := \frac{\left|ATextoCampo_{DeepBot} \cap ATextoCampo_{Real}\right|}{\left|ATextoCampo_{Real}\right|}$$

$$F_{1\,ATextoCampo} := \frac{2 \times \mathrm{Precisión}_{ATextoCampo} \times Alcance_{ATextoCampo}}{\mathrm{Precisión}_{ATextoCampo} + Alcance_{ATextoCampo}}$$

– Métricas para asociaciones entre campos del formulario y atributos del dominio.

$$\text{Precisión}_{ACampoAtributo} := \frac{\left|ACampoAtributo_{DeepBot} \cap ACampoAtributo_{Real}\right|}{\left|ACampoAtributo_{DeepBot}\right|} \quad \textbf{(IV.1-2)}$$

$$Alcance_{ACampoAtributo} := \frac{\left|ACampoAtributo_{DeepBot} \cap ACampoAtributo_{Real}\right|}{\left|ACampoAtributo_{Real}\right|}$$

$$F_{1\,ACampoAtributo} := \frac{2 \times \text{Precisión}_{ACampoAtributo} \times Alcance_{ACampoAtributo}}{\text{Precisión}_{ACampoAtributo} + Alcance_{ACampoAtributo}}$$

– Métricas para asociaciones globales entre formularios y dominios.

$$\text{Precisión}_{AFormularioDominio} := \frac{\left|AFormularioDominio_{DeepBot} \cap AFormularioDominio_{Real}\right|}{\left|AFormularioDominio_{DeepBot}\right|} \quad \textbf{(IV.1-3)}$$

$$Alcance_{AFormularioDominio} := \frac{\left|AFormularioDominio_{DeepBot} \cap AFormularioDominio_{Real}\right|}{\left|AFormularioDominio_{Real}\right|}$$

$$F_{1\,AFormularioDominio} := \frac{2 \times \text{Precisión}_{AFormularioDominio} \times Alcance_{AFormularioDominio}}{\text{Precisión}_{AFormularioDominio} + Alcance_{AFormularioDominio}}$$

$$\text{Precisión}_{FormulariosInvocados} := \frac{\left|FormulariosInvocados_{DeepBot}\right|}{\left|AFormularioDominio_{DeepBot} \cap AFormularioDominio_{Real}\right|}$$

La Tabla 3 resume los resultados experimentales obtenidos. Para cada dominio, se muestran los valores obtenidos para todas las métricas en el conjunto de entrenamiento (C_1, los sitios utilizados para definir los dominios), en el conjunto de pruebas (C_2, los sitios restantes) y el conjunto global formato por todos los sitios analizados (C_1+C_2, entrenamiento + pruebas).

Es importante destacar que, para calcular las medidas para las asociaciones formulario-dominio y campo-atributo no se han considerado formularios de búsqueda rápida ni formularios de autenticación. Los resultados sólo incluyen formularios multi-campo como los utilizados en las interfaces de consulta de búsqueda avanzada. Además, los resultados para las asociaciones campo-atributo se han medido de forma independiente a los resultados del paso previo (asociaciones texto-campo).

Los resultados obtenidos son bastante prometedores: todas las métricas muestran valores altos y algunos incluso alcanzan el 100%. A continuación se discuten las causas de los errores cometidos en cada una de las fases.

El alcance en la asociación entre formularios y dominios llegó al 100% en casi todos los casos, salvo en el conjunto de pruebas de los dominios de música y películas, en los que se obtuvo un 95%. En el dominio de música la razón es que la fuente ProMusicFind utiliza un alias para el atributo ARTISTA que no encaja con ninguno de los alias definidos en el dominio. Además, el formulario sólo tiene dos campos, con lo que aunque el sistema asigna el otro campo con el atributo del dominio TÍTULO, esto no es suficiente para superar el umbral de pertenencia al dominio. En el dominio de películas, el formulario de consulta de la fuente IGN.COM sólo presentaba dos campos de búsqueda (TÍTULO y GÉNERO) que se

pudieron asociar con atributos en la definición de dominio utilizada. Aunque el sistema asocia correctamente ambos, tampoco es suficiente para superar el umbral.

Tabla 3 Resultados experimentales para pruebas de modelado de formularios

	Libros			Música			Películas		
	C_1	C_2	C_1+C_2	C_1	C_2	C_1+C_2	C_1	C_2	C_1+C_2
Fase 1: Asociaciones Textos-Campo									
Precisión	*129/142* ***0.91***	*101/137* ***0.73***	*230/279* ***0.82***	*93/110* ***0.83***	*107/132* ***0.81***	*199/242* ***0.82***	*154/179* ***0.86***	*163/184* ***0.89***	*317/363* ***0.87***
Alcance	*129/132* ***0.98***	*101/127* ***0.79***	*230/259* ***0.88***	*92/94* ***0.98***	*107/109* ***0.98***	*199/203* ***0.98***	*154/168* ***0.92***	*163/181* ***0.90***	*317/349* ***0.91***
F_1	***0.94***	***0.76***	***0.85***	***0.90***	***0.89***	***0.89***	***0.89***	***0.89***	***0.89***
Fase 2: Asociaciones Campo–Atributo									
Precisión	*54/55* ***0.98***	*50/50* ***1.00***	*104/105* ***0.99***	*37/37* ***1.00***	*31/33* ***0.94***	*68/70* ***0.97***	*45/46* ***0.98***	*33/33* ***1.00***	*78/79* ***0.99***
Alcance	*54/54* ***1.00***	*50/53* ***0.94***	*104/107* ***0.97***	*37/37* ***1.00***	*31/37* ***0.84***	*68/74* ***0.92***	*45/45* ***1.00***	*33/35* ***0.94***	*78/80* ***0.98***
F_1	***0.99***	***0.97***	***0.98***	***1.00***	***0.89***	***0.94***	***0.99***	***0.97***	***0.98***
Fase 3: Asociaciones Formulario-Dominio									
Precisión	*13/13* ***1.00***	*11/11* ***1.00***	*24/24* ***1.00***	*10/10* ***1.00***	*9/9* ***1.00***	*19/19* ***1.00***	*12/12* ***1.00***	*9/9* ***1.00***	*20/20* ***1.00***
Alcance	*13/13* ***1.00***	*11/11* ***1.00***	*24/24* ***1.00***	*10/10* ***1.00***	*9/10* ***0.90***	*19/20* ***0.95***	*12/12* ***1.00***	*9/10* ***0.90***	*21/22* ***0.95***
F_1	***1.00***	***1.00***	***1.00***	***1.00***	***0.95***	***0.97***	***1.00***	***0.95***	***0.97***
Fase 4: Formularios Enviados									
Precisión	*13/13* ***1.00***	*11/11* ***1.00***	*24/24* ***1.00***	*10/10* ***1.00***	*9/9* ***1.00***	*19/19* ***1.00***	*12/12* ***1.00***	*9/9* ***1.00***	*21/21* ***1.00***

Los valores de precisión y alcance obtenidos para las asociaciones entre textos y campos del formulario excedió el 80%, excepto en el conjunto de pruebas del dominio de libros (precisión 0.73 y alcance 0.79). La mayor parte de los errores en este conjunto proviene de una única fuente (Blackwell's Bookshop). Si no se hubiese considerado esta fuente, las medidas tomarían valores similares a los alcanzados por los otros conjuntos.

Los fallos en esta fase fueron debidos principalmente a campos con valores acotados que no tenían ningún texto asociado de forma global en el formulario (el formulario sólo incluyó los textos correspondientes a sus valores). Esto hace que no se cumpla una de las heurísticas consideradas, que asume que cada campo del formulario debería tener al menos un texto asociado para explicar la función del campo al usuario. Lo que sucede es que algunas veces los valores de un campo acotado son autoexplicativos, y los creadores del formulario optan por no incluir una etiqueta de texto adicional. Por ejemplo, a partir de una lista de *radio buttons* que incluyan las opciones 'hardcover' y 'paperback', el usuario puede inferir que los botones se utilizan para expresar una condición de consulta sobre el atributo FORMATO, incluso aunque no incluya ninguna etiqueta de texto adicional indicándolo.

Finalmente, los valores de precisión y alcance son también altos (mayor que el 90% excepto en un caso) en las asociaciones entre campos del formulario y atributos del dominio de aplicación. Los fallos en esta etapa ocurrieron porque el dominio no incluyó los alias utilizados en los formularios para algunos atributos.

Los resultados experimentales para cada sitio en el dominio de tiendas electrónicas de libros se muestran en la Tabla 4. La Tabla 5 muestra los resultados para el dominio de música (en la parte superior) y para el dominio de películas (en la parte inferior).

Tabla 4 Resultados experimentales detallados para pruebas de modelado de formularios del dominio de aplicación de libros

► Métricas ▼ Fuentes	**Fase 1** **Asociaciones Texto-Campo**		**Fase 2** **Asocaciones Campo-Atributo**		**Fases 3 y 4** **Asociaciones Formulario-Dominio**		
	Precisión	*Alcance*	*Precisión*	*Alcance*	*Precisión*	*Alcance*	*Precisión* Formularios enviados
Amazon.com	17/19 = **0.89**	17/17 = **1.00**	7/7= **1.00**	7/7 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Barnes&Noble	9/11 = **0.82**	9/9 = **1.00**	6/7 = **0.86**	6/6 = **1.00**	2/2 = **1.00**	2/2 = **1.00**	2/2 = **1.00**
Bolen	5/5 = **1.00**	5/5 = **1.00**	4/4 = **1.00**	4/4 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
BookFinder4U.com	24/27 = **0.89**	24/25 = **0.92**	6/6 = **1.00**	6/6 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Cody's Books	7/8 = **0.88**	7/7 = **1.00**	5/5 = **1.00**	5/5 = **1.00**	2/2 = **1.00**	2/2 = **1.00**	2/2 = **1.00**
Daedalus Books&Music	30/32 = **0.94**	30/32 = **0.94**	7/7 = **1.00**	7/7 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Dymocks Booksellers	7/8 = **0.88**	7/7 = **1.00**	4/4 = **1.00**	4/4 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
eCampus.com	7/7 = **1.00**	7/7 = **1.00**	4/4 = **1.00**	4/4 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Powell's Books	16/17 = **0.94**	16/16 = **1.00**	6/6 = **1.00**	6/6 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Tattered Cover Bookstore	7/8 = **0.88**	7/7 = **1.00**	5/5 = **1.00**	5/5 = **1.00**	2/2 = **1.00**	2/2 = **1.00**	1/1 = **1.00**
Blackwell's Bookshop	12/33 = **0.36**	12/28 = **0.43**	7/7 = **1.00**	7/7 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Globe Pequot Press	6/6 = **1.00**	6/6 = **1.00**	5/5 = **1.00**	5/5 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Green Apple Books&Music	6/9 = **0.67**	6/9 = **0.67**	3/3 = **1.00**	3/4= **0.75**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Northshire Bookstore	13/14 = **0.93**	13/14 = **0.93**	6/6 = **1.00**	6/6 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Oxbow Books	12/15 = **0.80**	12/12 = **1.00**	6/6 = **1.00**	6/6 = **1.00**	2/2 = **1.00**	2/2 = **1.00**	2/2 = **1.00**
Strand Book Store	11/13 = **0.85**	11/12 = **0.92**	7/7 = **1.00**	7/8 = **0.88**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
The Book Pl@ce	8/9 = **0.89**	8/9 = **0.89**	4/4 = **1.00**	4/4 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
The American Book Center	13/13 = **1.00**	13/13 = **1.00**	5/5 = **1.00**	5/6 = **0.83**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
The Scholar's Bookshelf	14/18 = **0.78**	14/17 = **0.83**	3/3 = **1.00**	3/3 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**
Thomson Gale	6/7 = **0.86**	6/7 = **0.86**	4/4 = **1.00**	4/4 = **1.00**	1/1 = **1.00**	1/1 = **1.00**	1/1 = **1.00**

Tabla 5 Resultados experimentales detallados para pruebas de modelado de formularios del dominio de aplicación de música (arriba) y de películas (abajo)

► Métricas ▼ Fuentes	Fase 1 Asociaciones Texto-Campo		Fase 2 Asocaciones Campo-Atributo		Fases 3 y 4 Asociaciones Formulario-Dominio		
	Precisión	*Alcance*	*Precisión*	*Alcance*	*Precisión*	*Alcance*	*Precisión* Formularios enviados
A&B Sound	1/4 = 0.25	1/3 = 0.33	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Amazon.com Music	14/16 = 0.88	14/14 = 1.00	4/4 = 0.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
AudibleFaith	13/14 = 0.93	13/13 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Barnes&Noble Music	7/10 = 0.70	7/7 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
CD Quest	10/11 = 0.91	10/10 = 1.00	2/2 = 1.00	2/2 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Collectors' Choice Music	5/8 = 0.63	5/5 = 1.00	5/5 = 1.00	5/5 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Rough Trade	3/4 = 0.75	3/4 = 0.75	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Sam Goody	8/10 = 0.80	8/8 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Schott Musik International	12/14 = 0.86	12/12 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Tower Records	19/19 = 1.00	19/19 = 1.00	6/6 = 1.00	6/6 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Buy Cd	9/14 = 0.64	9/9 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
ClassicTrax.co.uk	8/9 = 0.88	8/9 = 0.89	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Cyber Music Surplus	20/22 = 0.91	20/20 = 1.00	4/4 = 1.00	4/6 = 0.67	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Ladyslipper Music	13/18 = 0.72	13/13 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Looney Tunes	13/18 = 0.72	13/14 = 0.93	3/4 = 0.75	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Mojo Sounds	11/12 = 0.92	11/11 = 1.00	3/4 = 0.75	3/4 = 0.75	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Musica Obscura	13/13 = 1.00	13/13 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
MyMusic.com	9/11 =0.82	9/9 = 1.00	4/4 = 1.00	4/6 = 0.67	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
ProMusicFind.com	2/12 = 0.17	2/2 = 1.00	1/1 = 1.00	1/2 = 0.50	-	0/1 = 0.00	-
Record Exchange	9/13 = 0.69	9/9 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00

► Métricas ▼ Fuentes	Fase 1 Asociaciones Texto-Campo		Fase 2 Asocaciones Campo-Atributo		Fases 3 y 4 Asociaciones Formulario-Dominio		
	Precisión	*Alcance*	*Precisión*	*Alcance*	*Precisión*	*Alcance*	*Precisión* Formularios enviados
Amazon.com DVD	39/41 = 0.95	39/39 = 1.00	10/10 = 1.00	10/10 = 1.00	2/2 = 1.00	2/2 = 1.00	2/2 = 1.00
BollyVista.com	7/7 = 1.00	7/7 = 1.00	5/5 = 1.00	5/5 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Docuseek	25/41 = 0.61	25/39 = 0.64	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
GlobeAndMail.com	10/10 = 1.00	10/10 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Half.com	5/8 = 0.63	5/5 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Kansas.com	9/9 = 1.00	9/9 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Movie Tickets.com	14/15 = 0.93	14/14 = 1.00	5/5 = 1.00	5/5 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Saregama	13/14 = 0.93	13/13 = 1.00	5/5 = 1.00	5/5 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
The Spinning Image	13/14 = 0.93	13/13 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Yahoo! Movies	19/20 = 0.95	19/19 = 1.00	3/4 = 0.75	3/3 = 1.00	2/2 = 1.00	2/2 = 1.00	2/2 = 1.00
Bestvideobuys	5/5 = 1.00	5/5 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Blockbuster.com	16/18 = 0.89	16/17 = 0.94	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Blockbuster UK	12/14 = 0.86	12/14 = 0.86	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
BrickFilms	16/22 = 0.73	16/16 = 1.00	3/3 = 1.00	3/4 = 0.75	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Fandango	9/10 = 0.90	9/9 = 1.00	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Human Rights Film Dir.	0/9 = 0.00	0/9 = 0.00	2/2 = 1.00	2/3 = 0.67	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
IGN.COM	65/65 = 1.00	65/70 = 0.93	2/2 = 1.00	2/2 = 1.00	-	0/1 = 0.00	-
Intelliflix	18/18 = 1.00	18/18 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Sensasian.com	13/14 = 0.93	13/14 = 0.93	4/4 = 1.00	4/4 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00
Video*ezy	9/9 = 1.00	9/9 = 1.00	3/3 = 1.00	3/3 = 1.00	1/1 = 1.00	1/1 = 1.00	1/1 = 1.00

IV.1.2. EXPERIENCIA EN ESTRUCTURACIÓN AUTOMÁTICA

Este apartado describe la evaluación experimental con fuentes reales de las técnicas propuestas para los procesos de estructuración automática.

Durante el desarrollo de las técnicas, se utilizó un conjunto de 20 páginas de 20 fuentes web diferentes. Los resultados obtenidos para estas páginas fueron utilizados para ajustar el algoritmo y seleccionar los valores adecuados para los umbrales utilizados por el proceso de *clustering* (ver apartado III.5.4.2).

Para las pruebas experimentales, se han seleccionado 200 nuevos sitios en diferentes dominios de aplicación (tiendas electrónicas de libros, música, sitios de información de patentes, sitios de proyectos de I+D con financiación pública, sitios de información de películas, etc.). Se ha realizado una consulta en cada sitio web para obtener la primera página de resultados. Se han seleccionado las consultas para garantizar que la colección incluye páginas con un número de resultados variable (algunas consultas devuelven sólo 2-3 resultados, mientras que otras devuelven cientos). La colección de páginas utilizadas en los experimentos está disponible online[21].

Durante la recolección de páginas para los experimentos, se encontraron tres fuentes de datos en las que el modelo de creación de páginas propuesto no es correcto. El modelo propuesto asume que todos los atributos de un registro de datos se muestran de forma contigua en la página. En estas tres fuentes no se cumple y por lo tanto el sistema falla con ellas. Para solucionar este problema, podrían utilizarse técnicas propuestas anteriormente en el estado del arte que complementarían nuestra solución, como la presentada por Zhai y Liu en DEPTA [ZL06] (véase la sección II.3.3.7.4 para más detalles). Estas fuentes no han sido consideradas en los experimentos.

Se han medido los resultados en tres fases del proceso:

- ***Fase 1***: Después de seleccionar la región de datos conteniendo la lista de registros dominante.
- ***Fase 2***: Después de seleccionar la mejor división en registros candidata.
- ***Fase 3***: Después de extraer los datos estructurados contenidos en la página.

La evaluación del proceso en estas tres fases permite evaluar de forma separada la efectividad de las técnicas utilizadas en cada fase. Además, los resultados obtenidos después de las dos primeras fases son útiles de forma aislada. Por ejemplo, algunas aplicaciones necesitan generar de forma automática una representación reducida de una página para mostrarla en un *portlet* dentro de un portal web o en un dispositivo móvil con pantalla de reducidas dimensiones. Para crear esta versión reducida para páginas que contienen las respuestas a una consulta, las aplicaciones pueden permitir elegir mostrar la

[21] http://www.tic.udc.es/~mad/resources/projects/dataextraction/testcollection_0507.htm

región de datos completa o los primeros registros de la lista, descartando el resto de la página.

La Tabla 6 muestra los resultados obtenidos en la evaluación experimental.

Tabla 6 Resultados experimentales para pruebas de estructuración automática

Fase 1	**# Correctas**	**# Incorrectas**	**% Correctas**
	198	*2*	*99.00*

Fase 2	**# Correctas**	**# Incorrectas**	**% Correctas**
	192	*8*	*96.00*

Fase 3	**# Registros a extraer**	*3557*
	# Registros extraídos	*3570*
	# Registros correctamente extraídos	*3496*
	Precisión	**0.9793**
	Alcance	**0.9829**

En la primera fase se utilizó información de la consulta ejecutada, como se explicó al final del apartado III.5.3. Los resultados muestran que la región de datos se identifica de forma correcta en todas las páginas con excepción de dos. En estos casos, la respuesta a la consulta devuelve pocos resultados y había una lista más larga en la barra lateral de la página conteniendo ítems relacionados con la consulta.

Los resultados de la segunda fase se pueden clasificar en dos categorías:

- Correcta: La división en registros seleccionada es la correcta. Como ya se ha comentado antes, es importante resaltar que en algunos casos, la región de datos contiene, al principio o al final, algunos subárboles que no son realmente parte de datos, sino que representan información auxiliar (como por ejemplo información del número de resultados de la página, formularios web para refinar la consulta o controles para navegar a otros intervalos de resultados). Esto no es un problema porque, como ya se ha explicado en el apartado III.5.5, el proceso en la fase 3 elimina estas partes irrelevantes, al no ser alineadas con los restantes registros. Por lo tanto, estos casos son considerados como correctos.
- Incorrecta: La división en registros seleccionada contiene algunos registros incorrectos (no necesariamente todos). Por ejemplo, dos registros diferentes pueden estar concatenados en uno o un registro puede aparecer segmentado en dos.

Como se observa en la Tabla 6, la división en registros seleccionada es correcta en el 96% de los casos. Es importante resaltar que, incluso en páginas para las que se seleccionan divisiones incorrectas, habrá normalmente muchos registros correctos. Por lo tanto, la fase 3 puede todavía funcionar bien sobre ellas. La razón principal de fallos en esta fase es que, en unas pocas fuentes, la medida de auto-similitud descrita en el apartado III.5.4.1 falla en la detección de la división correcta. Aunque la división de registros correcta está entre las candidatas, la medida de auto-similitud le da un valor superior a otra. Esto sucede porque, en estas fuentes, algunos registros de datos difieren bastante de otros. Por ejemplo, en un

caso en el que había dos registros de datos consecutivos mucho más cortos que el resto, el sistema eligió una división candidata que agrupó estos registros en uno. Es importante resaltar que, en la mayoría de los casos, sólo se identificaron incorrectamente uno o dos registros.

En la tercera fase se utilizaron métricas estándar de precisión y alcance. La precisión se calcula como el ratio entre el número de registros extraídos correctamente y el número total de registros extraídos por el sistema. El alcance se calcula como el ratio entre el número de registros extraídos correctamente por el sistema y el número total de registros que deberían de haber sido extraídos.

Estas son las métricas más importantes en lo que se refiere a aplicaciones de extracción de datos web, porque miden el rendimiento del sistema al final del proceso. Como se puede ver en la Tabla 6, los resultados obtenidos son muy altos, alcanzando respectivamente 0.9793 y 0.9829. La mayor parte de los fallos provienen de errores propagados de fases previas.

Se han medido también los tiempos de ejecución de las técnicas propuestas para estructuración automática. Las etapas más costosas del proceso son las siguientes:

- La generación de de la matriz de similitudes descrita en la sección III.5.4.2. Esto involucra el cálculo de la similitud entre cada par de subárboles de primer nivel de la región de datos.
- El cálculo de las auto-similitudes de las divisiones candidatas en registros, para seleccionar la mejor (como se ha descrito en la sección III.5.4.3). Esto involucra calcular la similitud entre cada par de registros en cada división candidata.
- El alineamiento múltiple de los registros de datos, para identificar sus atributos.

Afortunadamente, las tres fases pueden ser optimizadas de forma importante cacheando las medidas de similitudes:

- Cuando se genera la matriz de similitudes, nuestra implementación de las técnicas comprueba la igualdad entre las cadenas generadas para los árboles de primer nivel. De esta forma sólo es necesario realizar las comparaciones entre subárboles diferentes. Destacar también que como los subárboles representan una lista de registros de datos con estructura similar, normalmente habrá muchos subárboles idénticos.
- Se puede realizar una optimización similar en la fase de cálculo de auto-similitudes de las divisiones candidatas en registros. Debido a la estructura regular de los datos, normalmente habrá muchos registros candidatos iguales a lo largo de las divisiones en registros candidatas.
- En la etapa de alineamiento múltiple es también posible realizar el cálculo sólo una vez para cada par de registros idénticos.

Como consecuencia, las técnicas propuestas muestran un alto nivel de eficiencia. En las pruebas realizadas en un PC medio (Intel Centrino Core Duo 2 GHz, 1GB RAM), con el sistema operativo Windows XP y utilizando para la ejecución del proceso la última versión

de la máquina virtual Java (J2SE 1.6.0_02), el tiempo de ejecución medio para cada página en los experimentos comentados fue de 506 milisegundos (incluyendo el tiempo de analisis de la página HTML y de generación del árbol DOM). El proceso se ejecutó en un tiempo inferior a un segundo en el 94% de las páginas de la colección.

Las siguientes tablas muestran el resultado obtenido para cada una de las páginas consideradas en las pruebas de estructuración automática. Se especifica las fuentes para las que las fases 1 y 2 han sido exitosas (marcadas con el símbolo '✓') o se han producido fallos ('×') y los valores de precisión y alcance de cada una de ellas en la fase 3.

Tabla 7 Resultados experimentales de estructuración automática

Fuente	Fase 1	Fase 2	Fase 3 Precisión	Fase 3 Alcance
101CD.COM - http://www.101cd.com	✓	✓	*47/47 = 1.00*	*47/47 = 1.00*
121 Music - One To One Music - http://www.121music.com	✓	✓	*6/6 = 1.00*	*6/6 = 1.00*
1BookStreet - http://www.1bookstreet.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
1stcyberstore - http://1stcyberstore.com	✓	✓	*6/6 = 1.00*	*6/6 = 1.00*
2buycars - http://www.2buycars.net	✓	✓	*11/11 = 1.00*	*11/11 = 1.00*
401 Carfinder.ca - http://www.401carfinder.com	✓	✓	*6/6 = 1.00*	*6/6 = 1.00*
949 online.com - http://949online.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
A&B Sound - http://www.absound.ca	✓	✓	*12/12 = 1.00*	*12/12 = 1.00*
a1BOOKS - http://www.a1books.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
AARONBOOKS.COM - http://www.aaronbooks.com	✓	✓	*4/4 = 1.00*	*4/4 = 1.00*
AbeBooks.com - http://www.abebooks.com	✓	✓	*30/30 = 1.00*	*30/30 = 1.00*
AddALL - http://www.addall.com	✓	✓	*51/51 = 1.00*	*51/51 = 1.00*
alibris - http://www.alibris.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
AllBooks4Less.com (antiguo) (extracto) - http://www.allbooks4less.com	✓	✓	*34/34 = 1.00*	*34/34 = 1.00*
AllBooks4Less.com - http://www.allbooks4less.com	✓	✓	*22/22 = 1.00*	*22/22 = 1.00*
allclassical - http://www.allclassical.com	✓	✓	*56/62 = 0.90*	*56/63 = 0.89*
ALLEGRO - http://www.allegro-music.com	✓	✓	*19/25 = 0.76*	*19/25 = 0.76*
allmusic - http://www.allmusic.com	✓	✓	*63/63 = 1.00*	*63/63 = 1.00*
Amadeus.net (antiguo) (extracto) - http://www.amadeus.net	✓	×	*0/32 = 0.00*	*0/12 = 0.00*
Amazon DVD - http://www.amazon.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Amazon Music - http://www.amazon.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Amazon.com - http://www.amazon.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
AnyBook - http://www.anybook.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*

Tabla 8 Resultados experimentales de estructuración automática (continuación)

Fuente	Fase 1	Fase 2	Fase 3 Precisión	Fase 3 Alcance
AudibleFaith - http://www.audiblefaith.com	✓	✓	*17/17 = 1.00*	*17/17 = 1.00*
Autos 100.com - http://www.100autos.com	✓	✓	*9/9 = 1.00*	*9/9 = 1.00*
BAMM.COM BOOKSAMILLION.COM - http://www.booksamillion.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Barnes & Noble.com - http://video.barnesandnoble.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Barnes & Noble.com - http://www.barnesandnoble.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Barnes & Noble.com - http://www.barnesandnoble.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Best Music Buys - http://www.bestwebbuys.com	✓	✓	*50/50 = 1.00*	*50/50 = 1.00*
BEST VIDEO - http://www.bestvideo.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Bestvideobuys - http://www.bestwebbuys.com	✓	✓	*50/50 = 1.00*	*50/50 = 1.00*
Biblio.com - http://www.biblio.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Big Movie Zone - http://www.bigmoviezone.com	✓	✓	*4/4 = 1.00*	*4/4 = 1.00*
BiggerBooks.com - http://www.biggerbooks.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
BigTreeBooks.com - http://bigtreebooks.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
BIGWORDS.COM - http://www.bigwords.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
BizRate shopping search - http://www.bizrate.com	✓	✓	*7/7 = 1.00*	*7/7 = 1.00*
Blockbuster UK - http://www.blockbuster.co.uk	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Blockbuster.com - http://www.blockbuster.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Bolen - http://www.bolen.bc.ca	✓	✓	*9/9 = 1.00*	*9/10 = 0.90*
BOOK CLOSEOUTS.com - http://www.bookcloseouts.com	✓	✓	*17/17 = 1.00*	*17/17 = 1.00*
BookBrain - http://www.bookbrain.co.uk	✓	✓	*50/50 = 1.00*	*50/50 = 1.00*
BookFinder4U.com - http://www.bookfinder4u.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
bookpool.com - http://www.bookpool.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Books & Collectibles - http://www.booksandcollectibles.com.au	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
BOOKS INC. - http://www.booksinc.net	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
bookstore.co.uk - http://bookstore.tbpcontrol.co.uk	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
BrickFilms - http://www.brickfilms.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Buy Cd - http://www.buycd.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Buy.com - http://www.buy.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
Buy.com Music - http://www.buy.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
CampusI. - http://www.campusi.com	✓	✓	*65/65 = 1.00*	*65/65 = 1.00*
Carbuyer.com - http://www.carbuyer.com	✓	✓	*12/12 = 1.00*	*12/12 = 1.00*
CD connection.com - http://www.cdconnection.com	✓	✓	*100/100 = 1.00*	*100/100 = 1.00*
cd EXPRESS - rockhouse records - http://www.cdexpress.com	✓	✓	*14/14 = 1.00*	*14/14 = 1.00*
CD Quest - http://www.cdquest.com	✓	✓	*50/50 = 1.00*	*50/50 = 1.00*
CDEUROPE.COM - http://www.cdeurope.com	✓	✓	*15/15 = 1.00*	*15/15 = 1.00*
CDPLUS.COM - http://www.cdplus.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
ClassBook.com - http://www.classbook.com	✓	✓	*14/14 = 1.00*	*14/14 = 1.00*
ClassicTrax.co.uk - http://www.classictrax.ltd.uk	✓	✓	*4/4 = 1.00*	*4/4 = 1.00*
Cody's Books - http://www.codysbooks.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Cokesbury.com - http://www.cokesbury.com	✓	✓	*6/6 = 1.00*	*6/6 = 1.00*
Computer Manuals - http://www.compman.co.uk	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
ConsumerRecords.org - http://www.consumerreports.org	✓	×	*9/10 = 0.90*	*9/10 = 0.90*
Cordis (extracto) - http://cordis.europa.eu	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
CRAIG MOERER - RECORDS BY MAIL - http://www.recordsbymail.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
Cyber Music Surplus - http://www.cybermusicsurplus.com	✓	✓	*5/5 = 1.00*	*5/5 = 1.00*
chapters.indigo.ca - http://chapters.indigo.ca	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
CheapestBookPrice.com - http://www.cheapesttextbooks.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
Christianbook.com - http://www.christianbook.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Daedalus Books&Music - http://www.daedalusbooks.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
djangos - http://www.djangomusic.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Djangos - http://www.djangomusic.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*
Docuseek - http://www.docuseek.com	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
DVD Empire - http://www.dvdempire.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
DVD Empire.com - http://www.dvdempire.com	✓	✓	*25/25 = 1.00*	*25/25 = 1.00*
DVDs & Videos Stores Online - http://www.bizrate.com	✓	✓	*6/6 = 1.00*	*6/6 = 1.00*
Dymocks Booksellers - http://www.dymocks.com.au	✓	✓	*14/14 = 1.00*	*14/14 = 1.00*
eCampus.com - http://www.ecampus.com	×	✓	*10/10 = 1.00*	*10/10 = 1.00*
El Corte Inglés - El sitio de los libros - http://www.elcorteingles.es	✓	✓	*10/10 = 1.00*	*10/10 = 1.00*
ELSEVIER science & technology books - http://books.elsevier.com	✓	✓	*20/20 = 1.00*	*20/20 = 1.00*

Tabla 9 Resultados experimentales de estructuración automática (continuación)

Fuente	Fase 1	Fase 2	Fase 3 Precisión	Fase 3 Alcance
EXCALIBUR FILMS - http://excaliburfilms.com	✓	✓	13/13 = 1.00	13/13 = 1.00
f.y.e. for your entertainment - http://www.fye.com	✓	✓	25/25 = 1.00	25/25 = 1.00
f.y.e. for your entertainment - http://www.fye.com	✓	✓	25/25 = 1.00	25/25 = 1.00
Fandango - http://www.fandango.com	✓	✓	20/20 = 1.00	20/20 = 1.00
fbo ENTERTAINMENT - http://www.fbo.com.au	✓	✓	6/6 = 1.00	6/6 = 1.00
FIRSTANDSECOND.COM - http://www.firstandsecond.com	✓	✓	3/3 = 1.00	3/3 = 1.00
FIRSTANDSECOND.COM - http://www.firstandsecond.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Fnac.es - http://www.fnac.es	✓	✓	15/15 = 1.00	15/15 = 1.00
for Movies.com - http://www.formovies.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Globe Pequot Press - http://www.globepequot.com	✓	✓	2/2 = 1.00	2/2 = 1.00
GlobeAndMail.com - http://www.theglobeandmail.com	✓	✓	16/16 = 1.00	16/16 = 1.00
GMN your arts network - http://www.gmn.com	✓	✓	25/25 = 1.00	25/25 = 1.00
gracenote - http://www.gracenote.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Green Apple Books&Music - http://www.greenapplebooks.com	✓	✓	6/6 = 1.00	6/6 = 1.00
Half.com - http://www.half.ebay.com	✓	✓	16/16 = 1.00	16/16 = 1.00
half.com [by ebay] - http://half.ebay.com	✓	✓	20/20 = 1.00	20/20 = 1.00
half.com Music [by ebay] - http://half.ebay.com	✓	✓	20/20 = 1.00	20/20 = 1.00
HamiltonBook.com - http://www.hamiltonbook.com	✓	✓	3/3 = 1.00	3/3 = 1.00
HARCOURT International - http://www.harcourt-international.com	✓	✓	8/8 = 1.00	8/8 = 1.00
Harvard Book Store - http://www.harvard.com	✓	✓	26/26 = 1.00	26/26 = 1.00
HOLLYWOOD.com - http://www.hollywood.com	✓	✓	10/13 = 0.77	10/10 = 1.00
Human Rights Film Directory - http://db.lib.washington.edu	✓	✓	20/20 = 1.00	20/20 = 1.00
IAN BRABNER Bookseller - http://www.bookgarden.com	✓	✓	10/10 = 1.00	10/10 = 1.00
IGN.COM - http://search.ign.com	✓	✓	10/10 = 1.00	10/10 = 1.00
ILAB LILA - http://www.ilabdatabase.com	✓	✓	25/25 = 1.00	25/25 = 1.00
IMDb - http://us.imdb.com	✓	✓	97/97 = 1.00	97/97 = 1.00
indiaplaza.in [formerly fabmall.com] - http://www.indiaplaza.in	✓	✓	10/10 = 1.00	10/10 = 1.00
Intelliflix - http://www.intelliflix.com	✓	✓	20/20 = 1.00	20/20 = 1.00
IRAN Melody - http://www.iranmelody.com/Music	✓	✓	10/10 = 1.00	10/10 = 1.00
isbn.nu - http://isbn.nu	✓	✓	50/50 = 1.00	50/50 = 1.00
JR.com - http://www.jr.com	✓	✓	20/20 = 1.00	20/20 = 1.00
JR.com - http://www.jr.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Kansas.com - http://ae.kansas.com	✓	✓	10/10 = 1.00	10/10 = 1.00
KHAZANA - http://www.khazana.com	✓	×	0/6 = 0.00	0/5 = 0.00
Ladyslipper Music - http://www.ladyslipper.org	✓	✓	8/8 = 1.00	8/8 = 1.00
LifeWay christian stores - http://www.lifewaystores.com	✓	✓	16/16 = 1.00	16/16 = 1.00
Looney Tunes - http://www.looneytunescds.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Mojo Sounds - http://www.mojosounds.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Movie Tickets.com - http://www.movietickets.com	✓	✓	6/6 = 1.00	6/6 = 1.00
Moviefone - http://www.moviefone.com	✓	✓	25/25 = 1.00	25/25 = 1.00
Movies Unlimited - http://www.moviesunlimited.com	✓	✓	15/15 = 1.00	15/15 = 1.00
Musica Obscura - http://www.musicaobscura.com	✓	✓	24/24 = 1.00	24/24 = 1.00
musicsafe.nl - http://www.themusicsafe.com	✓	✓	10/10 = 1.00	10/10 = 1.00
MVD Entertainment Group - http://mvdb2b.com	✓	✓	22/22 = 1.00	22/22 = 1.00
MyMusic.com - http://www.mymusic.com	✓	✓	20/20 = 1.00	20/20 = 1.00
mySimon - http://www.mysimon.com	✓	✓	10/10 = 1.00	10/10 = 1.00
mySimon Music - http://www.mysimon.com	✓	✓	10/10 = 1.00	10/10 = 1.00
mySimon Video - http://www.mysimon.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Northshire Bookstore - http://www.northshire.com	✓	✓	25/25 = 1.00	25/25 = 1.00
Nostalgia Family Video - http://www.nostalgiafamilyvideo.com	✓	✓	5/5 = 1.00	5/5 = 1.00
Oxbow Books - http://www.oxbowbooks.com	✓	×	3/4 = 0.75	3/5 = 0.60
PAGE1ONE - http://www.page1book.com	✓	✓	20/20 = 1.00	20/20 = 1.00
pearsoned.co.uk - http://www.pearsoned.co.uk	✓	✓	10/10 = 1.00	10/10 = 1.00
PlanetOut.com - http://www.planetout.com	✓	✓	23/23 = 1.00	23/23 = 1.00
PlayCentric.com - http://www.playcentric.com	✓	✓	15/15 = 1.00	15/15 = 1.00
PlayCentric.com - http://www.playcentric.com	✓	✓	3/3 = 1.00	3/3 = 1.00
Powell's Books - http://www.powells.com	✓	×	24/24 = 1.00	24/25 = 0.96
Pricing Central.com PriceGrabber - http://pricingcentral.pricegrabber.com	✓	✓	20/20 = 1.00	20/20 = 1.00
ProMusicFind.com - http://www.promusicfind.com	✓	✓	18/18 = 1.00	18/18 = 1.00

Tabla 10 Resultados experimentales de estructuración automática (continuación)

Fuente	Fase 1	Fase 2	Fase 3 Precisión	Fase 3 Alcance
RANDOM HOUSE, INC. - http://www.randomhouse.com	✓	✓	2/2 = 1.00	2/2 = 1.00
RARE HOLLYWOOD.COM - http://www.rarevideo.com	✓	✓	12/12 = 1.00	12/12 = 1.00
Record Exchange - http://www.buymusichere.net	✓	✓	20/20 = 1.00	20/20 = 1.00
Reel.com - http://www.reel.com	✓	✓	9/9 = 1.00	9/9 = 1.00
ROBERTS hard to find VIDEOS - http://robertsvideos.co	✓	✓	7/7 = 1.00	7/7 = 1.00
Rotten Tomatoes - http://www.rottentomatoes.com	✓	✓	9/9 = 1.00	9/9 = 1.00
Rough Trade - http://www.roughtrade.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Sam Goody - http://www.samgoody.com	×	✓	25/25 = 1.00	25/25 = 1.00
SATURN RECORDS - http://www.saturnrecords.com	✓	✓	50/50 = 1.00	50/50 = 1.00
Schott Musik International - http://www.schott-music.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Sensasian.com - http://sensasian.com	✓	✓	2/2 = 1.00	2/2 = 1.00
Sensasian.com Music - http://sensasian.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Shopping.com - http://uk.shopping.com	✓	✓	30/30 = 1.00	30/30 = 1.00
SONGSEARCH - http://www.songsearch.com	✓	✓	22/22 = 1.00	22/22 = 1.00
SONGSEARCH music, movies & more - http://www.songsearch.com	✓	✓	35/35 = 1.00	35/35 = 1.00
Strand Book Store - http://www.strandbooks.com	✓	✓	25/25 = 1.00	25/25 = 1.00
street online - http://www.streetsonline.co.uk	✓	✓	12/12 = 1.00	12/12 = 1.00
Student BOOKWORLD.COM - http://www.studentbookworld.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Tamdem Library Books - http://www.tandemlibrarybooks.com	✓	✓	25/25 = 1.00	25/25 = 1.00
Tattered Cover Bookstore - http://www.tatteredcover.com	✓	✓	20/20 = 1.00	20/20 = 1.00
Taylor & Francis CRC Press - http://www.crcpress.com	✓	✓	11/11 = 1.00	11/11 = 1.00
TEXTBOOKX.COM - http://www.textbookx.com	✓	✓	10/10 = 1.00	10/10 = 1.00
TEXTBOOKX.COM (avanzado) - http://www.textbookx.com	✓	✓	19/19 = 1.00	19/19 = 1.00
The American Book Center - http://www.abc.nl	✓	✓	15/15 = 1.00	15/15 = 1.00
The Book Pl@ce - http://www.thebookplace.com	✓	✓	10/10 = 1.00	10/10 = 1.00
The DRAMA BOOK SHOP since 1917 - http://www.dramabookshop.com	✓	✓	20/21 = 0.95	20/20 = 1.00
The HTML Writers Guild - http://www.hwg.org	✓	✓	10/10 = 1.00	10/10 = 1.00
THE LAST UNICORN - http://www.thestore24.com	✓	✓	23/23 = 1.00	23/23 = 1.00
The MIT Press - http://mitpress.mit.edu	✓	✓	4/4 = 1.00	4/4 = 1.00
The Scholar's Bookshelf - http://www.scholarsbookshelf.com	✓	✓	2/2 = 1.00	2/2 = 1.00
The Spinning Image - http://www.thespinningimage.co.uk	✓	✓	20/20 = 1.00	20/20 = 1.00
The University of Chicago Press - http://www.press.uchicago.edu	✓	✓	10/10 = 1.00	10/10 = 1.00
The Video Collection - http://www.videocollection.com	✓	✓	11/11 = 1.00	11/11 = 1.00
Thomson Gale - http://www.gale.com	✓	✓	13/13 = 1.00	13/13 = 1.00
tlavideo.com - http://www.tlavideo.com	✓	×	0/8 = 0.00	0/13 = 0.00
Tower Records - http://www.towerrecords.com	✓	✓	25/25 = 1.00	25/25 = 1.00
Traverse Motors Toyota - http://toyota.traversemotors.com	✓	✓	3/3 = 1.00	3/3 = 1.00
University Book Store - http://www3.bookstore.washington.edu	✓	✓	20/20 = 1.00	20/20 = 1.00
VARSITY BOOKS - https://www.varsitybooks.com	✓	✓	6/6 = 1.00	6/6 = 1.00
VCD Gallery - http://www.vcdgallery.com	✓	✓	10/10 = 1.00	10/10 = 1.00
Video Universe - http://www.cduniverse.com	✓	✓	34/34 = 1.00	34/34 = 1.00
Video*ezy - http://www.videoezy.com.au	✓	✓	10/10 = 1.00	10/10 = 1.00
VIDEOMATICA.CA - http://www.videomatica.com	✓	✓	15/15 = 1.00	15/15 = 1.00
Vidiots - http://www.vidiotsvideo.com	✓	✓	15/15 = 1.00	15/15 = 1.00
VINTAGE VINYL - http://www.vvinyl.com	✓	✓	13/13 = 1.00	13/13 = 1.00
VMH sales.com - http://www.vmhsales.com	✓	✓	20/20 = 1.00	20/20 = 1.00
WAL*MART - http://www.walmart.com	✓	✓	24/24 = 1.00	24/24 = 1.00
WAL*MART - http://www.walmart.com	✓	✓	20/20 = 1.00	20/20 = 1.00
WAL*MART Music - http://www.walmart.com	✓	✓	14/24 = 0.58	14/24 = 0.58
WHSmith - http://www.whsmith.co.uk	✓	×	23/23 = 1.00	23/25 = 0.92
WHSmith - http://www.whsmith.co.uk	✓	✓	9/9 = 1.00	9/9 = 1.00
Wolters Kluwer - Lippincott Williams & Wilkins - http://www.lww.com	✓	✓	3/3 = 1.00	3/3 = 1.00
wsbradio.com: DVD store search - http://wsbradio.com	✓	✓	20/20 = 1.00	20/20 = 1.00
WWW Music Database - http://www.onlinemusicdatabase.com	✓	✓	10/10 = 1.00	10/10 = 1.00
www.folkweb.com - http://www.folkweb.com	✓	✓	5/5 = 1.00	5/5 = 1.00
www2.vinylvendors.com - http://www2.vinylvendors.com	✓	✓	9/9 = 1.00	9/9 = 1.00
Yahoo! Movies - http://movies.yahoo.com	✓	×	8/8 = 1.00	8/9 = 0.89
ZEVELEKAKIS Online Bookstore - http://www.zevelekakis.gr	✓	✓	25/25 = 1.00	25/25 = 1.00
ZoomMovie - http://www.zoommovie.com	✓	✓	40/40 = 1.00	40/40 = 1.00

IV.2. APLICACIONES DE LAS TÉCNICAS

Parte de las técnicas desarrolladas en esta tesis se han empleado para otros propósitos además del *crawling* dirigido de la Web Oculta. Concretamente, las técnicas de modelado de formularios web han sido aplicadas por Raposo et al. [RPAH07] [RPB07] para el mantenimiento de secuencias de navegación en programas envoltorio.

En el apartado I.1.2 se ha comentado la posibilidad de utilizar programas envoltorio (*wrappers*) para extraer información estructurada de páginas web. En los últimos años se han desarrollado diversas técnicas para generar semi-automáticamente programas envoltorio. Sin embargo, las fuentes web semi-estructuradas presentan un elevado nivel de autonomía, y tanto las interfaces de consulta como la forma de representar los resultados pueden cambiar sin previo aviso. Mantener automáticamente el envoltorio implica adaptarlo a los cambios que se produzcan en la fuente.

El mantenimiento automático de envoltorios puede dividirse en dos subproblemas. Dependiendo de la naturaleza del cambio, puede ser necesario abordar ambos o solamente uno de ellos.

Por una parte puede ser necesario generar nuevas secuencias de navegación para acceder a las páginas que contienen los datos (si las secuencias utilizadas por el envoltorio antiguo han dejado de funcionar). Por otra parte, puede ser necesario generar nuevos programas de extracción capaces de obtener los datos deseados.

Parte de las técnicas definidas para el tratamiento de la Web Oculta del lado servidor han sido utilizadas para resolver el primero de los subproblemas: la generación de nuevas secuencias de navegación para acceder a las páginas que contienen los datos.

El sistema recibe como entrada los nombres de campos y alias para cada atributo en el momento en el que se creó el envoltorio, la secuencia de navegación previa (antes de producirse el cambio en la fuente) y un conjunto de consultas almacenadas durante la operación normal del envoltorio, junto con sus resultados. La idea básica para su funcionamiento se basa en un proceso de *crawling* dirigido, que comienza en la página inicial de la fuente y busca formularios de consulta candidatos a través de todo el sitio web. Para acotar este proceso es posible configurar una profundidad máxima de *crawling*. Para cada formulario encontrado, aplica las técnicas descritas en la sección III.4 para modelarlo y obtener su relevancia respecto a la secuencia y envoltorio previos, que actúan de forma similar al dominio de aplicación. Por último, se aplican las técnicas de generación de consultas, para obtener la forma de invocar el formulario.

IV.3. EXPERIENCIA EN APLICACIONES INDUSTRIALES

Las técnicas diseñadas en esta tesis doctoral relacionadas con el tratamiento de las dificultades de acceso a la Web Oculta del lado cliente, han servido como base para la implementación de un sistema de recorrido de la Web que realiza un tratamiento parcial de la Web Oculta, y que está incluido dentro de una plataforma comercial de extracción de información web.

El producto concreto que incluye el *crawler* de la Web Oculta del lado cliente es Denodo Aracne [DNDARN]. Este producto permite automatizar el recorrido de la Web, incluyendo aquella porción que utiliza mecanismos de dinamismo del lado cliente, para aplicar una serie de filtros de contenido sobre los documentos extraídos y clasificarlos en diferentes índices. También proporciona una interfaz de programación para poder realizar consultas sobre la información recolectada.

El sistema de *crawling* se ha utilizado en varios proyectos de diferentes ámbitos en los que era necesario extraer información no estructurada de fuentes web que utilizaban tecnologías del lado cliente de forma exhaustiva o complejos sistemas de redirecciones. Algunos de los ámbitos de los proyectos en los que se ha utilizado el sistema son el de inteligencia competitiva y vigilancia tecnológica.

Respecto a las técnicas de modelado de formularios y ejecución, parte de las mismas se han aplicado en un sistema comercial de mantenimiento automático de las secuencias de navegación de programas envoltorio, como se ha comentado en la sección IV.2. El producto concreto que incluye el sistema de mantenimiento es Denodo ITPilot [DNDITP]. Este producto permite automatizar cualquier tipo de navegación sobre fuentes web y extraer la información deseada, de forma que los datos estructurados puedan ser usados directamente por las aplicaciones o puedan ser combinados con datos procedentes de otras fuentes. Además, ofrece la funcionalidad, ya comentada, de mantener automáticamente los envoltorios utilizados para navegar y extraer la información de las fuentes web.

Finalmente, respecto a las técnicas de estructuración automática, constituyen la base de un componente de estructuración automática que formará parte de la siguiente versión del producto Denodo ITPilot [DNDITP].

IV.4. EVALUACIÓN DE CUMPLIMIENTO DE OBJETIVOS

En esta sección se evalúa el grado de cumplimiento de los objetivos de esta tesis doctoral, que fueron descritos en la sección I.2.

1) Proponer una arquitectura para aplicaciones de recopilación automática de información de la Web utilizando el enfoque dirigido, que tenga en cuenta el acceso a la Web Oculta. Esta arquitectura debe identificar y tener en cuenta todos los pasos involucrados en la creación de este tipo de aplicaciones. Se pretende minimizar el grado de intervención humana requerida y el nivel de especialización de la misma.

La arquitectura que se ha propuesto en la sección III.2 contempla todos los aspectos que debe de tratar una aplicación de recopilación automática de información relevante de la Web Oculta para unos dominios de aplicación concretos.

Para tratar con la Web Oculta del lado cliente, el módulo de *crawling* ha sido extendido para utilizar componentes de navegación de alto nivel que son capaces de tratar con las tecnologías de *script* y de implementar los nuevos métodos de navegación que requieren este tipo de páginas.

Para tratar la Web Oculta del lado servidor, la arquitectura incluye un componente encargado de detectar formularios de consulta y determinar si son relevantes en base a una serie de especificaciones de dominios de aplicación de ejemplo, previamente modelados. Para conseguir que el *crawling* no diverja, se propone la utilización de técnicas existentes en la literatura previa, como las presentadas por Barbosa y Freire [BF07]. Para aquellos formularios que cumplan los criterios de adecuación a uno de los dominios considerados, se definen nuevos algoritmos para aprender a ejecutar consultas sobre ellos. Las nuevas páginas obtenidas como resultado de las consultas sobre los formularios relevantes localizados, contienen normalmente enumeraciones de elementos de resultado. El sistema analiza esas páginas y obtiene una visión estructurada para cada uno de los resultados. Para ello se utiliza un componente de estructuración automática, que tiene en cuenta la información contenida en el dominio de aplicación para mejorar su efectividad. La arquitectura se completa con un módulo de indexación de la información recolectada, aplicando para ello algoritmos de indexación existentes, para la generación de buscadores sobre el conjunto completo de información recolectada. La arquitectura propuesta resuelve también el problema asociado al acceso a los recursos indexados no directamente accesibles a través de URLs mediante la utilización de las secuencias de navegación de alto nivel asociadas a las *rutas*.

La arquitectura propuesta minimiza tanto el grado de intervención humana como su especialización. La configuración inicial del *crawler* es similar a la de un sistema de *crawling* convencional, con la salvedad de la definición de dominios de aplicación. Por otra parte, el resto de las técnicas operan de forma totalmente automática, por lo que no requieren ningún tipo de intervención humana.

2) Proponer nuevas técnicas y algoritmos para permitir a los sistemas actuales de crawling dirigido tratar con fuentes de información que hagan uso de técnicas de 'dinamismo en el lado cliente', tales como el uso de lenguajes de script, sistemas complejos de mantenimiento de sesión, etc. También es necesario tratar con otras implicaciones del dinamismo como la aparición o desaparición de elementos de la página en función de las acciones del usuario.

Las técnicas para el tratamiento de los problemas planteados por las tecnologías utilizadas en la parte cliente por los sitios web han sido descritas en la sección III.3. Se trata de un aspecto que apenas ha sido abordado en la literatura previa. En el enfoque propuesto, al hacer uso de un mini-navegador web para el acceso a los recursos, el gestor de descarga del *crawler* dispone del mismo entorno de ejecución que cualquier usuario para tratar con páginas que utilicen tecnologías del lado cliente. De forma adicional, el algoritmo definido para gestión de navegaciones sobre páginas dinámicas permite el tratamiento del resto de aspectos, como la aparición o desaparición de elementos de la página en función de las acciones del *crawler*.

Estas técnicas han sido utilizadas en varios proyectos reales en el área de vigilancia tecnológica, para permitir obtener información de sitios web caracterizados por utilizar exhaustivamente tecnologías del lado cliente y complejos sistemas de redirecciones.

3) Proponer nuevas técnicas y algoritmos para identificar y aprender a consultar automáticamente formularios de consulta web relevantes para la tarea especificada.

Las técnicas y algoritmos para abordar los problemas que plantea el acceso a la Web Oculta del lado servidor han sido presentados en la sección III.4. Estas técnicas han mostrado porcentajes de efectividad muy altos en los experimentos con fuentes reales descritos en la sección IV.1.1. Además, las técnicas propuestas presentan diversas ventajas sobre las aproximaciones propuestas en trabajos previos. Estas ventajas se presentan en el apartado V.1.3.

4) Proponer nuevas técnicas y algoritmos para extraer automáticamente los datos estructurados contenidos en las respuestas a consultas efectuadas utilizando formularios de consulta sobre bases de datos subyacentes.

La sección III.5 presenta los nuevos algoritmos definidos para la extracción de forma automática de datos estructurados contenidos en las páginas de respuesta de consultas a formularios web relevantes para el dominio considerado.

Con respecto a los trabajos previos, el modelo propuesto puede tratar con páginas que no verifican las condiciones requeridas por aproximaciones previas (ver apartado V.1.4, para más detalle). El modelo propuesto ha sido validado con un gran número de sitios web reales, obteniendo un alto grado de efectividad. Los experimentos se han descrito en el apartado IV.1.2.

5) Validar la efectividad de las técnicas propuestas con experimentos con tareas y fuentes de información web reales, demostrando que es posible la construcción de sistemas de crawling dirigido capaces de acceder y procesar adecuadamente la información de la Web Oculta, y de llevar a cabo tareas reales de recopilación de información con un alto grado de efectividad.

Los experimentos descritos en la sección IV.1 se han realizado con fuentes web reales, de diversos tipos y en diferentes dominios de aplicación. Se han realizado de forma independiente para los principales componentes de la arquitectura propuesta, para obtener la efectividad de forma independiente.

En los experimentos se obtuvieron porcentajes de efectividad muy altos, superando en prácticamente todos los casos el 90% tanto en precisión como en alcance (*recall*) y alcanzando el 100% en numerosos casos.

V. DISCUSIÓN, CONCLUSIONES Y TRABAJO FUTURO

Este capítulo concluye la tesis doctoral. La sección V.1 discute las técnicas propuestas en este trabajo con respecto al estado del arte previo. La sección V.2 repasa las principales contribuciones originales y la sección V.2.2 presenta las conclusiones de este trabajo. Finalmente, la sección V.3 esboza las líneas de trabajo futuro del autor.

V.1. DISCUSIÓN

En esta sección se discuten las aportaciones de este trabajo con respecto al estado del arte actual, que fue descrito en el capítulo II.

Esta sección se organiza como sigue. En primer lugar, el apartado V.1.1 discute la arquitectura propuesta para el *crawling* estructurado de la información contenida en la Web Oculta. El apartado V.1.2 se ocupa del modelo de navegación definido para permitir tratar la Web Oculta del lado cliente. A continuación, el apartado V.1.3 compara las técnicas propuestas en esta tesis doctoral para el tratamiento de la Web Oculta del lado servidor con las propuestas en trabajos previos. Por último, el apartado V.1.4 realiza dicha comparación con las técnicas de estructuración automática de los resultados obtenidos.

V.1.1. ARQUITECTURA DE CRAWLING DIRIGIDO DE LA WEB OCULTA

En la sección III.2 se ha propuesto una arquitectura completa para sistemas que tengan como objetivo realizar un *crawling* dirigido y estructurado de la información contenida en la Web Oculta relevante para un dominio de aplicación determinado.

En la sección II.3 se han considerado diversos sistemas que abordan el tratamiento de la información contenida en la Web Oculta. La arquitectura más completa presentada en trabajos previos para aplicaciones de *crawling* dirigido es la propuesta por HiWE [RG01]. Sin embargo, esta arquitectura no aborda el proceso completo para resolver el problema objetivo de este trabajo:

- La arquitectura de *crawling* de HiWE no tiene en cuenta el acceso a la Web Oculta del lado cliente. La arquitectura presentada en este trabajo introduce nuevos conceptos y modificaciones en los algoritmos de *crawling* para tener en cuenta este problema.
- La arquitectura de HiWE no aborda el problema de cómo obtener una visión estructurada de la información extraída de la Web Oculta. La arquitectura presentada en este trabajo contempla este aspecto, así como sus implicaciones sobre el proceso de indexación y consulta.
- La arquitectura de HiWE tampoco aborda el problema de cómo localizar formularios relevantes para un dominio de aplicación, ni cómo integrar las técnicas de acceso a la Web Oculta con las técnicas de *crawling* dirigido de la Web de Superficie. Aunque este trabajo no presenta nuevas técnicas para abordar estos problemas, sí incluye en la arquitectura propuesta los componentes necesarios para realizarlas, pudiendo utilizarse para su implementación las técnicas propuestas en trabajos previos que se describen en detalle en los apartados II.2 y II.3.3.3.

V.1.2. TRATAMIENTO DE LA WEB OCULTA DEL LADO CLIENTE

En la sección III.3 se han descrito un conjunto de técnicas para facilitar a los sistemas de *crawling* el tratamiento de las dificultades que presenta el acceso a la información contenida en la Web Oculta del lado cliente.

El desafío que presenta el tratamiento de la Web Oculta del lado cliente no ha sido apenas abordado hasta el momento. Algunos sistemas de *crawling* [WC07] han incluido intérpretes de *JavaScript* [MR07] [MSM07] en los clientes HTTP que utilizan para proporcionar algún soporte, aunque limitado, para tratar con páginas con soporte de lenguajes de *script*. Sin embargo, el enfoque propuesto ofrece varias ventajas sobre ellos:

- Al ser los procesos de *crawling* mini-navegadores, el modelo se independiza de los problemas de bajo nivel relacionados con las tecnologías utilizadas en la creación de las páginas web, como pueden ser la interpretación de código de *script* o del tratamiento de complejos sistemas de redirecciones (incluyendo aquellas generadas por Applets Java y programas Flash).
- Se generaliza el concepto de URL en el de *ruta*, permitiendo de esta forma independizarse de los problemas relacionados con los mecanismos de mantenimiento de sesión utilizados por algunos sitios web, tanto en tiempo de recorrido de la Web como para el acceso posterior a los documentos.
- El modelo de *crawling* definido es capaz de tratar con elementos de navegación (enlaces o cualquier otro elemento que tenga algún manejador de evento registrado que pueda generar una nueva navegación) generados de forma dinámica en respuesta a eventos producidos por el *crawler*, como por ejemplo la aparición de menús emergentes con nuevos enlaces a otras páginas. En el apartado III.3.3 se

describe el algoritmo propuesto para tratar el modelo de navegación requerido por las páginas dinámicas.

V.1.3. TRATAMIENTO DE LA WEB OCULTA DEL LADO SERVIDOR

En la sección III.4 se han descrito las diferentes técnicas propuestas para modelar y aprender a consultar formularios web, que constituyen el punto de entrada a la Web Oculta del lado servidor.

En los últimos años han sido varios los trabajos que se han ocupado de este problema, utilizando distintas aproximaciones.

El enfoque más similar al que se propone es el utilizado por Raghavan y García-Molina en HiWE [RG00] [RG01]. HiWE es un *crawler* orientado a un dominio de aplicación determinado, capaz de reconocer y rellenar formularios relevantes a una tarea de obtención de datos concreta, de forma automática. Del mismo modo que el modelo que se propone en este trabajo, HiWE utiliza medidas de distancia visual para encontrar los textos asociados a cada campo en un formulario, y medidas de similitud textual para asociar campos del formulario con atributos del dominio. Para aprender cómo rellenar un formulario, HiWE encaja los textos asociados con cada campo del formulario y las etiquetas asociadas a los atributos, definidas en su tabla LVS (un concepto que juega un papel similar al de una especificación de dominio de aplicación). El proceso seguido por el enfoque propuesto tiene las siguientes ventajas respecto al realizado por HiWE:

- Puede utilizar un formulario, incluso aunque tenga campos que no se asocien con ningún atributo del dominio de aplicación. Por ejemplo, la especificación de dominio de aplicación de la Figura 27 no tiene un atributo que se pueda asociar con el campo PUBLICADOPOR de la Figura 34, pero de todas formas sería capaz de utilizar el formulario. En cambio HiWE tiene como restricción que requiere que la tabla LVS contenga una definición de atributo que se asocie con cada campo del formulario no acotado.
- Detecta de forma correcta cuando un campo tiene más de un texto asociado, lo cual mejora la precisión del proceso de asociación de campos del formulario con atributos del dominio.
- La decisión de asociar un texto a un campo no está basada sólo en condiciones locales al campo, sino que las heurísticas que se consideran tienen en cuenta el contexto proporcionado por el formulario completo. Por ejemplo, en el formulario de ejemplo de la Figura 34, HiWE asignaría de forma errónea el texto 'Hardcover' al segundo elemento *radio button* (c_{52}), por ser el texto más próximo, además de estar a la izquierda del campo. Sin embargo, el enfoque que se propone asignaría de forma correcta el texto 'e-Books & Docs' a c_{53}, 'Paperback' a c_{52} y 'Hardcover' a c_{51}. Esto es debido a que se garantiza que todos los campos tendrán asociado al menos un texto (las asociaciones realizadas por HiWE dejarían c_{51} sin textos asociados o asignarían el mismo texto a dos campos).

- Para invocar formularios y obtener páginas de resultados, se aplican las técnicas definidas para el tratamiento de la Web Oculta del lado cliente, permitiendo, por ejemplo, atravesar formularios que estén controlados por código de *script*.

Bergholz y Chidlovski [BC03] presentan otro sistema para *crawling* basado en dominios de la Web Oculta. Sin embargo, sólo trata formularios de búsqueda de texto, es decir, formularios que tienen un único campo que permite buscar por palabra clave sobre una colección de documentos no estructurados. En cambio el enfoque propuesto en este trabajo soporta formularios con varios campos, que son los normalmente utilizados para consultar datos estructurados.

Ntoulas et al. [NZC05] tratan el problema de generación automática de consultas para obtener todo el contenido que haya tras un formulario web. Los autores proponen nuevas técnicas para generar de forma automática nuevas palabras de búsqueda a partir de resultados obtenidos por consultas previas y para priorizarlas con el objetivo de recuperar todo el contenido oculto tras un formulario, utilizando el mínimo número de consultas. La capacidad de generar de forma automática nuevas consultas a partir de los resultados obtenidos por consultas previas sería una característica interesante para el enfoque propuesto, de forma que este trabajo lo complementa. Sin embargo, las técnicas que presentan los autores necesitarían ser adaptadas de forma importante, puesto que sólo consideran formularios de búsqueda por palabra clave y no tratan con formularios constituidos por varios campos.

El problema de extraer el contenido completo que se oculta detrás de un formulario web también ha sido tratado por Liddle et al. [LESY02]. Sin embargo, el modelo que los autores hacen de los formularios no trata con campos de texto, lo cuál limita enormemente su aplicabilidad práctica.

Existen otros enfoques que se ocupan de la problemática de modelado de formularios desde el punto de vista de metabúsqueda, pero sólo abordan una parte del problema. Por ejemplo, MetaQuerier [ZHC04] sólo se ocupa del análisis y modelado de los formularios, pero no de emparejar los atributos especificados en la metainformación de un dominio de aplicación con los campos del formulario, un paso necesario para las aplicaciones de *crawling* dirigido. Lo mismo sucede en el caso de las técnicas utilizadas por el sistema de metabúsqueda WISE [HMYW05a].

V.1.4. ESTRUCTURACIÓN AUTOMÁTICA

En la sección III.5 se ha presentado un nuevo método para detectar y extraer de forma automática una lista de registros de datos estructurados, de una página web de resultados de una consulta sobre un formulario de búsqueda.

Varios trabajos han abordado el problema de realizar tareas de extracción de datos web sin requerir la intervención del usuario humano. Un primer intento es IEPAD [CL01] (descrito en el apartado II.3.3.7.1) que utiliza árboles Patricia [GBS92] y técnicas de alineamiento de cadenas para buscar patrones repetitivos en las cadenas de etiquetas HTML de una página. El método utilizado por IEPAD es muy probable que genere patrones

incorrectos mezclados con patrones correctos, de modo que es necesario que un usuario post-procese la salida. Éste es un inconveniente muy importante respecto a la aproximación que se propone.

Otro sistema es RoadRunner [CMM01] (descrito en el apartado II.3.3.7.2), que recibe como entrada múltiples páginas cumpliendo la misma plantilla y las utiliza para inducir una expresión regular libre de uniones que puede ser utilizada para extraer datos de páginas que cumplen esa plantilla. RoadRunner no puede tratar con disyunciones en el esquema de entrada. Otro inconveniente con respecto a la aproximación que se propone es que requiere recibir como entrada múltiples páginas cumpliendo la misma plantilla.

Como ya se había comentado antes, el modelo de creación de página descrito en el apartado III.5.1 fue previamente introducido por Arasu y García-Molina en ExAlg [AG03a]. Del mismo modo que RoadRunner, EXALG (descrito en el apartado II.3.3.7.3) recibe como entrada múltiples páginas que cumplen la misma plantilla y las utiliza para inducir la plantilla y derivar un conjunto de reglas de extracción de datos. EXALG realiza algunas suposiciones sobre las páginas web que, de acuerdo a los propios experimentos de sus autores, no se cumplen en un número significativo de casos: por ejemplo, asumen que la plantilla asigna un número relativamente grande de *tokens* a cada constructor de tipos. Además su enfoque asume que un subconjunto substancialmente numeroso de campos de datos a ser extraídos tiene una ruta desde la raíz del árbol DOM de la página, única. Finalmente, otro inconveniente de EXALG es que requiere recibir múltiples páginas de entrada.

Zhai y Liu presentan DEPTA [ZL06] (descrito en el apartado II.3.3.7.4), un método que utiliza la disposición visual de la información en una página junto con técnicas de distancia de edición para detectar listas de registros en una página y para extraer los registros de datos estructurados que contiene. Del mismo modo que el método que se propone, DEPTA necesita una única página de entrada conteniendo una lista de registros de datos estructurados. Los autores también utilizan la observación de que en el árbol DOM de una página, cada registro en una lista está compuesto de una secuencia de subárboles hermanos consecutivos. Sin embargo, los autores asumen otras dos observaciones adicionales: 1) que todos los registros están formados por exactamente el mismo número de subárboles, y 2) que el espacio visual entre dos registros de datos en una lista es mayor que el espacio visual entre cualquier par de valores dentro del mismo registro. Es relativamente fácil encontrar ejemplos en sitios web reales que no lo cumplan. Por ejemplo, en la página de ejemplo de la Figura 40 no se cumple ninguna de las dos. Adicionalmente, el método utilizado por DEPTA para detectar las regiones de datos es considerablemente más costoso que el que se propone, porque involucra un número potencialmente elevado de cálculos de distancia de edición.

Respecto al método propuesto por Lerman et al. [LGMK04], se basa en la observación de que las páginas que contienen listas de resultados de una consulta en una fuente web semi-estructurada, normalmente incluyen un enlace para cada registro, permitiendo acceso a información adicional, de detalle, relativa a cada uno de ellos. El método se basa en localizar información redundante entre páginas de listado y páginas de detalle para utilizarla como ayuda al proceso de estructuración. Este método no es válido para fuentes

en las que no existen páginas de detalle y además requiere múltiples páginas para que pueda funcionar.

El método de estructuración automática que se ha propuesto necesita una única página de entrada y no requiere que se cumplan las heurísticas antes mencionadas. Por ejemplo, no asume que el espacio visual entre dos registros consecutivos tiene que ser mayor que el existente entre cualquier par de valores dentro del mismo registro ni que todos los registros de una página tengan que estar formados exactamente por el mismo número de subárboles.

V.2. CONCLUSIONES

Esta sección presenta las conclusiones de este trabajo. En primer lugar el apartado V.2.1 repasa las principales contribuciones de esta tesis doctoral. El apartado V.2.2 establece y justifica una lista de conclusiones.

V.2.1. RESUMEN DE LAS PRINCIPALES CONTRIBUCIONES

En función de la discusión con respecto al estado del arte realizada en el apartado anterior, concluimos que este trabajo presenta las siguientes aportaciones principales (que fueron ya adelantadas en la sección I.3):

1) Una arquitectura para aplicaciones de recopilación dirigida de información, que contempla el acceso a la Web Oculta. La arquitectura se basa en las arquitecturas existentes de *crawling dirigido* y las complementa y adapta para reflejar los componentes necesarios para el acceso a la Web Oculta.

2) Un conjunto de técnicas y algoritmos para realizar *crawling* de la 'Web Oculta de lado cliente'. Las técnicas de crawling propuestas identifican los recursos descargados mediante el concepto de 'ruta', que es una abstracción que extiende el concepto de URL para soportar mecanismos de mantenimiento de sesión. El proceso de *crawling* se basa en la utilización de componentes denominados '*mini-navegadores*', que son capaces de tratar con lenguajes de script (e.g. *JavaScript*). Además, se tienen en cuenta las variaciones en la página que pueden producirse dinámicamente en respuesta a las interacciones del usuario (e.g. nuevos enlaces que aparecen en la página al desplegar un menú emergente).

3) Un conjunto de técnicas y algoritmos para identificar y aprender a consultar automáticamente formularios de consulta web relevantes para la tarea especificada. El proceso de *crawling* dirigido admitirá como parte de su entrada un conjunto de *especificaciones de dominio de aplicación*. Cada especificación de dominio incluye cierta información que ayuda a identificar formularios de consulta relevantes para la tarea objetivo, así como un conjunto de consultas que se desearía ejecutar sobre tales formularios. Cuando el proceso de *crawling* encuentra durante su exploración un formulario de consulta web, utiliza diversas heurísticas basadas en técnicas de distancia visual y similitud textual para determinar si el formulario es relevante para alguno de los *dominios de aplicación* y, en ese caso, para aprender automáticamente a ejecutar consultas sobre el mismo, y obtener las respuestas a las consultas especificadas.

4) Nuevas técnicas y algoritmos para extraer automáticamente los datos estructurados contenidos en las respuestas obtenidas a consultas efectuadas utilizando formularios web de consulta sobre bases de datos subyacentes. Cuando el sistema de *crawling* descubre un nuevo formulario relevante y ejecuta un conjunto de consultas sobre el mismo, las páginas de respuesta son proporcionadas como entrada a un módulo de

estructuración automática. Este módulo utiliza técnicas originales para obtener de cada página HTML de respuesta los registros estructurados contenidos en ella.

5) Un conjunto de herramientas software que permiten la creación sencilla de aplicaciones de crawling dirigido. Estas herramientas han sido utilizadas para la validación experimental de las técnicas propuestas, así como en diversas tareas de obtención de datos reales.

V.2.2. CONCLUSIONES OBTENIDAS

En este apartado se enumeran y justifican las principales conclusiones de este trabajo.

CONCLUSIÓN 1: Es posible diseñar soluciones de crawling dirigido para extracción de información de forma automatizada de la Web Oculta.

La arquitectura presentada en esta tesis doctoral considera todas las tareas necesarias para realizar la extracción de información dirigida de la Web Oculta. En esta tesis se han propuesto soluciones para aquellas tareas para las que no existían técnicas adecuadas (acceso a la Web Oculta del lado cliente) y se han presentado nuevas técnicas que mejoran el estado del arte actual en otras tareas (acceso a la Web Oculta del lado servidor y estructuración automática). El resto de tareas involucradas en la arquitectura (como *crawling* dirigido) pueden ser implementadas haciendo uso de técnicas presentadas previamente en la literatura, como se ha comentado en la sección III.2, donde se describe la arquitectura propuesta.

Las técnicas propuestas han sido implementadas y probadas con un amplio número de fuentes web reales, pertenecientes a diferentes dominios de aplicación. Los resultados de estos experimentos han sido muy prometedores, avalando la eficacia de este enfoque. Los experimentos se han descrito en la sección IV.1.

CONCLUSIÓN 2: Es posible automatizar el acceso a la información contenida en la Web Oculta del lado cliente.

La arquitectura presentada en la sección III.2 permite el acceso a la información contenida en la Web Oculta del lado cliente, mediante la utilización de mini-navegadores como componentes para la descarga de documentos web e implementando sobre ellos los algoritmos de navegación sobre páginas dinámicas definidos en el apartado III.3.

El problema del tratamiento de la Web Oculta del lado cliente ha recibido muy poca atención hasta el momento. En la actualidad, con el gran éxito que están teniendo tecnologías como *Ajax*, los sitios web están tendiendo a utilizar cada día más este tipo de tecnologías. Aunque el uso de estas técnicas para un *crawling* global puede ser aún dificultoso por razones de eficiencia (ver la sección V.3.6 para una discusión más detallada), el uso de estas técnicas está plenamente justificado para un *crawler* dirigido, orientado a obtener la mayor cantidad de información relevante para una temática objetivo.

Podemos concluir que no sólo es importante disponer de un contenedor capaz de tratar con las tecnologías del lado cliente, sino que es también necesario un nuevo modelo de navegación sobre las páginas dinámicas, caracterizadas por poder generar nuevos elementos en una página a partir de las interacciones que un usuario realice sobre ella. Las técnicas propuestas en este trabajo satisfacen ambos requisitos.

Finalmente, se quiere resaltar que el modelo presentado para abordar las problemáticas que el tratamiento de la Web Oculta del lado cliente introduce, ha sido utilizado de forma exitosa en varias aplicaciones empresariales, en los campos de búsqueda corporativa y vigilancia tecnológica, como parte de un producto real como se ha comentado en la sección IV.3.

CONCLUSIÓN 3: Es posible definir dominios de aplicación válidos para el algoritmo de reconocimiento de formularios, de forma sencilla y considerando un número reducido de fuentes del dominio.

El estudio realizado por Chang et al. en [CHLP+04] apuntaba en esta dirección, y en nuestros experimentos del apartado IV.1.1 hemos llegado a la misma conclusión, siendo posible crear especificaciones de dominio efectivas de forma sencilla y rápida, explorando sólo un conjunto reducido de fuentes en un dominio de aplicación para encontrar los atributos y alias más relevantes para el mismo. Incluso con especificaciones de dominio sencillas como las utilizadas en el prototipo realizado, se consiguen resultados muy buenos. Esto se debe a que el vocabulario utilizado para identificar los campos de los formularios para un dominio de aplicación determinado suele converger con gran rapidez.

CONCLUSIÓN 4: Las técnicas de reconocimiento de formularios web propuestas en este trabajo son efectivas con fuentes web reales.

Las técnicas presentadas en la sección III.4 crean el modelo de formularios web en base a una serie de heurísticas de distancia visual entre textos y elementos del formulario. A partir de un formulario modelado, obtienen su relevancia respecto a una especificación de dominio de aplicación y en el caso de que determinen que un formulario puede ser considerado como relevante, generan de forma automática nuevas rutas que permitan acceder a la información contenida en la base de datos subyacente al formulario.

Los resultados obtenidos de los experimentos descritos en el apartado IV.1.1 son bastante prometedores: todas las métricas muestran valores altos y algunos incluso alcanzan el 100%. Estos experimentos utilizaron fuentes web reales en diferentes dominios de aplicación, utilizando formularios de consulta diversos con un número y disposición de campos muy variable.

Estas técnicas también han sido aplicadas con éxito en un sistema real de mantenimiento de secuencias de navegación de programas envoltorio de fuentes web, como se ha comentado en la sección IV.2.

CONCLUSIÓN 5: Las técnicas de estructuración automática de registros de datos contenidos en páginas de resultados de formularios de consulta web son efectivas con fuentes web reales.

El método propuesto para estructuración automática de páginas de resultados sólo necesita como entrada una página conteniendo una lista de registros. El método comienza localizando la región de datos que contiene la lista dominante. Como siguiente paso, se realiza un proceso de agrupamiento ('*clustering*') para limitar el número de divisiones en registros candidatas en la región de datos. Posteriormente, se selecciona la que proporciona una mayor auto-similitud de acuerdo a técnicas basadas en distancia de edición entre cadenas. Finalmente se utiliza un algoritmo de alineamiento múltiple de secuencias para extraer los valores de los atributos de cada registro de datos.

En los experimentos que se comentan en el apartado V.1.4 se han obtenido valores de precisión y alcance muy altos 0.9793 y 0.9829 respectivamente. Estos experimentos utilizaron fuentes web reales en diversos dominios de aplicación, tales como comercio electrónico, búsqueda de patentes o vuelos.

CONCLUSIÓN 6: La implementación de las técnicas propuestas supone una solución a los principales problemas que plantea la extracción de información estructurada perteneciente a la Web Oculta, relevante para un dominio.

Aunque el prototipo realizado no constituye una implementación completa de la arquitectura propuesta, permite concluir que las técnicas propuestas constituyen una solución efectiva a los problemas de obtención de información estructurada de la Web Oculta. Los resultados obtenidos en los diferentes experimentos realizados con fuentes web reales han sido muy prometedores, consiguiendo extraer información relevante para un dominio de aplicación, de la Web Oculta.

De todas formas, el prototipo realizado no implementa la arquitectura propuesta de forma completa porque no incluye los clasificadores de documentos y formularios utilizados para que el proceso de *crawling* no diverja a sitios web no relevantes. Como se ha comentado, el prototipo basa su dirección sólo en la aplicación de expresiones regulares sobre las rutas, para determinar la dirección del *crawling*. Una implementación completa podría utilizar para este propósito las técnicas presentadas previamente en otros trabajos y que han sido descritas en los apartados II.2 y II.3.3.3.

CONCLUSIÓN 7: Las tareas de crawling dirigido pueden beneficiarse en gran medida del acceso a los contenidos de la Web Oculta.

Diversos estudios se han ocupado de caracterizar la Web Oculta [Bergman00] [CHLP+04], llegando a la conclusión de que contiene mucha más información que la que es accesible directamente a través de enlaces, y además de mucha mayor calidad.

Actualmente los procesos de *crawling* global no son capaces de considerar el volumen de páginas de la Web de Superficie, continuamente en aumento. Sin embargo, la inclusión de información de la Web Oculta permitiría obtener información de más calidad y, algo

todavía más importante considerando la visión estructurada de esa información, permitiría la realización de consultas más precisas.

Adicionalmente, parte de la Web formada por páginas estáticas no está accesible por los problemas introducidos por las tecnologías de cliente utilizadas en la construcción de páginas web. Aunque muchos sitios web populares evitan la utilización de código *script* y otras tecnologías similares para ser correctamente indexados por motores de búsqueda globales como Google, sitios web de tamaño medio conteniendo información de gran valor continúan utilizándolas de forma intensiva. Es especialmente el caso de sitios web que requieren suscripción o autenticación de usuarios. Debido a que estos sitios no tienen ningún incentivo para facilitar el trabajo a los grandes motores de búsqueda, continúan utilizando de forma intensiva tecnologías para proporcionar dinamismo en el lado cliente. Sin embargo, esta clase de sitios normalmente son los que proporcionan la información más valiosa para muchas aplicaciones de *crawling* dirigido, como vigilancia tecnológica o motores de búsqueda verticales (de unas temáticas determinadas). Además, la aparición de tecnologías como *Ajax* convierte esta necesidad en todavía más importante.

Por esta razón se concluye que los esfuerzos para acceder a la información contenida en la Web Oculta son valiosos y deben de ser continuados.

V.3. LÍNEAS DE TRABAJO FUTURO

Esta sección describe algunas de las líneas de trabajo actuales y futuras del autor.

V.3.1. INFERIR DOMINIOS DE FORMA AUTOMÁTICA

El modelo utilizado asume un dominio de aplicación creado manualmente por un usuario y que no se modifica durante la fase de funcionamiento del mismo.

Por una parte, podría resultar útil utilizar las tuplas estructuradas resultado de consultas sobre los formularios para extender la meta-información de los dominios de aplicación. Por ejemplo, podrían añadirse nuevas consultas a ejecutar o nuevos términos relevantes que ayuden en la definición de la semántica de cada atributo.

Una aproximación más ambiciosa consistiría en aplicar las técnicas propuestas para definir una arquitectura de *crawling* que, a partir de unos documentos relevantes que definan el criterio de extracción, recorra la Web localizando formularios de consulta, los modele y genere esquemas de dominios de aplicación candidatos. Para cada nueva especificación de dominio generada, se necesitaría algún algoritmo de *clustering* que determinase si varios 'dominios de aplicación candidatos' pueden corresponderse con el mismo 'dominio de aplicación', obteniendo a su vez las correspondencias entre los campos en los diferentes dominios de aplicación candidatos. Esta aproximación permitiría, tanto la generación automática de aplicaciones de *crawling* como de metabúsqueda, para diferentes temáticas.

Un dominio de aplicación candidato podría modelarse de acuerdo al modelo propuesto para un formulario en el apartado III.4.1. Para determinar si dos dominios de aplicación candidatos se corresponden con el mismo 'dominio de aplicación', se podrían aplicar las técnicas propuestas para determinar la relevancia entre un formulario y un dominio en el apartado III.4.2 y III.4.3. Sería necesario utilizar algún algoritmo nuevo para obtener una definición de dominio global a partir de dos dominios candidatos que se consideren equivalentes, para utilizarlo como referencia en posteriores comparaciones.

Para la realización de las primeras consultas del formulario podrían utilizarse palabras relevantes de los documentos que definen el criterio del *crawling* o un conjunto de consultas semilla. Una vez obtenidos los primeros resultados de consultas, estos podrían utilizarse para obtener nuevos valores de consulta. Además, es importante considerar que la propia forma de generar el dominio global permite obtener de forma automática ejemplos de consulta. Por ejemplo, si un campo en el dominio aparece como enumerado en algún formulario, los valores de consulta incluidos pueden utilizarse como valores de consulta válidos para ese campo en el dominio.

V.3.2. INFERIR RELACIONES ENTRE DOMINIOS DE APLICACIÓN DE FORMA AUTOMÁTICA

La información obtenida de las páginas web recolectadas durante el proceso de *crawling* es almacenada en índices sobre los que se pueden realizar consultas. Se dispone de dos tipos de índices: en uno se almacenan los documentos obtenidos para permitir búsquedas por palabra clave; en el otro se almacenan los campos de los registros estructurados para permitir la realización de consultas precisas sobre ellos.

En el caso de los índices con campos estructurados, habrá uno diferente para cada dominio de aplicación configurado en un proceso de *crawling*. Cada domino posee su propio esquema de datos. La información almacenada en un índice con múltiples campos puede verse como datos almacenados en una tabla de una base de datos.

La idea que se propone es aplicar técnicas que en base a los datos obtenidos para cada uno de los dominios estructurados, detecte relaciones entre sus esquemas que permitan definir nuevos esquemas agregados para consultar la información recolectada. Por ejemplo, si el sistema tuviese un dominio de tiendas electrónicas y otro dominio de críticas de productos, analizando los datos estructurados obtenidos se podría obtener un esquema de consulta de productos con sus críticas (equivalente a un esquema *join* en una base de datos convencional).

V.3.3. EXTENDER EL MODELO DE FORMULARIO

El modelo propuesto para los formularios presenta limitaciones para aquellos en los que no existan etiquetas de texto para alguno de sus elementos. Existen formularios que utilizan imágenes u otros elementos para especificar a un usuario la semántica de un elemento. Podrían utilizarse sistemas de OCR (*Optical Character Recognition*) para obtener el texto asociado a esos elementos gráficos. De esta forma pasarían a estar integrados en el modelo de forma transparente.

V.3.4. EXTENDER EL MODELO DE PÁGINA PARA ESTRUCTURACIÓN

El modelo de dominio de aplicación propuesto define sólo campos atómicos. Aunque esto no supone un problema desde el punto de vista de reconocimiento de formularios, sí puede serlo para la estructuración de las páginas de resultados.

Algunas páginas de resultados presentan, para algún campo, una colección de valores. Por ejemplo un libro puede presentar múltiples autores. En otros casos, los elementos de la colección de valores pueden presentar un esquema propio, es decir, valores con una estructura de datos anidada.

Aunque el modelo de estructuración automática propuesto no falla en estos casos (los datos con estructura anidada aparecerán formando parte del mismo campo), es cierto que podría obtener una estructuración más fina de los datos, para permitir consultas todavía más

precisas. El dominio en este caso presentaría, además de tipos simples, tipos colección y registros para permitir representar elementos multivaluados con diferentes esquemas.

V.3.5. COMPLETAR EL PROTOTIPO PARA TAREAS DE EXTRACCIÓN DIRIGIDA DE INFORMACIÓN DE LA WEB OCULTA

Como ya se ha comentado en secciones previas, el prototipo utilizado para validar las técnicas no constituye una implementación completa de la arquitectura propuesta, aunque es suficiente para validar las técnicas propuestas. Para completar la implementación, podría implementarse un componente de descubrimiento de fuentes. En este trabajo se ha considerado este componente, pero no se ha propuesto un modelo para el mismo por considerar suficientemente válida la aproximación presentada por Barbosa y Freire en [BF07] (ver apartado II.3.3.3 para más detalle).

V.3.6. TRATAMIENTO DE LA WEB OCULTA DEL LADO CLIENTE EN *CRAWLING* GLOBAL

La principal diferencia entre un sistema de *crawling* dirigido y un sistema de *crawling* global la constituye el volumen de información que debe tratar cada uno de ellos. Para aplicar las técnicas de tratamiento de la Web Oculta de lado cliente a un escenario de *crawling* global, sería necesario validar que también permiten la construcción de sistemas eficientes y escalables en ese nuevo contexto.

Las técnicas propuestas para el tratamiento de la Web Oculta del lado cliente, utilizan mini-navegadores de alto nivel para acceder a los recursos. Los mini-navegadores posibilitan el acceso a información que de otra forma no sería posible alcanzar, pero suponen un coste bastante superior, tanto a nivel de consumo de memoria como de tiempos de acceso a los recursos, que los clientes HTTP que utiliza un sistema de *crawling* convencional.

Para poder aplicar estas técnicas a un sistema de *crawling* global, sería posible considerar la implementación de los componentes de navegación de alto nivel utilizando clientes web más ligeros con soporte para tecnologías del lado cliente sobre HTTP. También podrían usarse los mini-navegadores sólo para aquellas páginas en las que se detecten tecnologías del lado cliente, utilizándolos simplemente para ‘abrir las puertas’ a la información contenida en la Web Oculta, pero utilizando clientes HTTP en el resto de casos. Aunque existen sitios web que utilizan de forma exhaustiva técnicas del lado cliente, para los que sería necesario utilizar los mini-navegadores para el acceso a todos sus recursos, son también muchos los sitios web que presentan tecnologías del lado cliente sólo para algunas navegaciones, siendo posible realizar el resto con componentes más eficientes.

V.3.7. ESTUDIO DEL NIVEL DE UTILIZACIÓN DE LAS TECNOLOGÍAS DEL LADO CLIENTE EN SITIOS WEB

El nivel de utilización de lenguajes de *script* en el diseño de sitios web ha ido variando a lo largo del tiempo. Tras el gran auge de sus comienzos, fue perdiendo protagonismo, en gran parte debido a las dificultades de los *crawlers* globales para acceder a su información. Actualmente, y principalmente desde la aparición de tecnologías como *Ajax* y a un nuevo modelo de diseño de sitios web en los que el lado cliente gana mucha importancia, las tecnologías de *scripting* han resurgido con fuerza.

Desde el punto de vista de los sistemas de *crawling global* sería útil, no sólo conocer el porcentaje de fuentes en Internet que utilizan este tipo de tecnologías en la actualidad, sino también para qué las están utilizando.

Se trataría de definir una serie de niveles de utilización de tecnologías de *script*, estimando para cada uno de ellos el coste que supondría tratarlo para un sistema de *crawling*. La escala podría tener en cuenta aspectos tales como: sitios que requieren *login*/*password*, enlaces estáticos, enlaces con *script* en `href`, enlaces con *script* en algún manejador de evento, generación de código HTML utilizando la función `document.write`, etc.

La escala definida se utilizaría para dos propósitos: por una parte, para determinar el nivel de utilización de las tecnologías del lado cliente en la Web de hoy y, por otra parte, para caracterizar el nivel de efectividad de diferentes sistemas de *crawling* en el tratamiento de las tecnologías del lado cliente.

VI. ANEXO A. CRAWLING DE ALTAS PRESTACIONES

La primera pregunta que se plantea ante la construcción de un sistema que obtenga de forma total o parcial la información contenida en la Web es cómo 'descubrir' todas las páginas existentes. Antes de la aparición de la Web, los grandes conjuntos de información, como por ejemplo, bibliotecas o periódicos, proporcionaban un sistema directo de acceso a toda la información. Sin embargo, no hay ningún catálogo con los URLs de todos los documentos disponibles en la Web. El único procedimiento disponible para obtener URLs consiste en analizar las páginas recolectadas, para obtener los enlaces a otras páginas. Éste es el principio básico de los sistemas de *crawling*.

```
C0 = { Conjunto inicial de URLs }
C = C0
MIENTRAS el proceso de crawling no termine HACER
        Obtener el siguiente URL de C
        Conectar al servidor
        Obtener la página (método GET HTTP)
        Cuando se reciba la página
                Añadir a C los URLs obtenidos de la página
                (Opcional) Realizar otros procesamientos
fin MIENTRAS
```

Figura 49 Algoritmo de funcionamiento general de un *crawler* web

Como todo sistema software, un *crawler* tiene un conjunto de datos de entrada, un proceso y un conjunto de datos de salida. Conceptualmente, la entrada de un *crawler* está formada por un conjunto inicial de URLs, C_0. Al iniciarse, C_0 se almacena en una cola, en la que los diferentes URLs serán priorizados. El *crawler*, durante su proceso, progresivamente, obtendrá URLs de esa cola en algún orden, descargará las páginas y las analizará para obtener el conjunto de páginas a las que apuntan para formar un nuevo conjunto de URLs, que añadir a la cola. Este proceso se repite hasta que el *crawler* decide detenerse. El *crawler*, durante su operación o al final de la misma, obtiene como resultado el conjunto de todos los documentos web accedidos mediante los URLs tratados. El algoritmo general de *crawling* se muestra en la Figura 49.

Uno de los principales desafíos que deben abordar estos sistemas es cómo tratar de forma eficiente el 'mayor' y 'mejor' número de documentos contenidos en la Web.

Numerosos trabajos de investigación han estudiado los diferentes problemas que plantea la obtención de información de la Web. También han sido muchas e importantes las empresas que han dedicado esfuerzo en el desarrollo de sistemas de *crawling* web. Desafortunadamente, muchas de las técnicas que utilizan estas compañías, y en especial las que tienen que ver con temas de optimización y rendimiento, no se han hecho públicas por considerarse de valor estratégico. Existen sólo unos cuantos documentos de dominio público que proporcionan algunos detalles. Entre ellos se pueden citar una publicación referente al *crawler* Mercator implementado por Compaq para Altavista[22] [HN99] y una descripción de la primera generación del *crawler* que utiliza Google[23] [BP98]. Basado de forma parcial en esa información, en [Chakrabarti03] se presenta la arquitectura genérica de un sistema de *crawling* eficiente y escalable para el procesamiento de grandes cantidades de documentos de la Web. La arquitectura se muestra en la Figura 50.

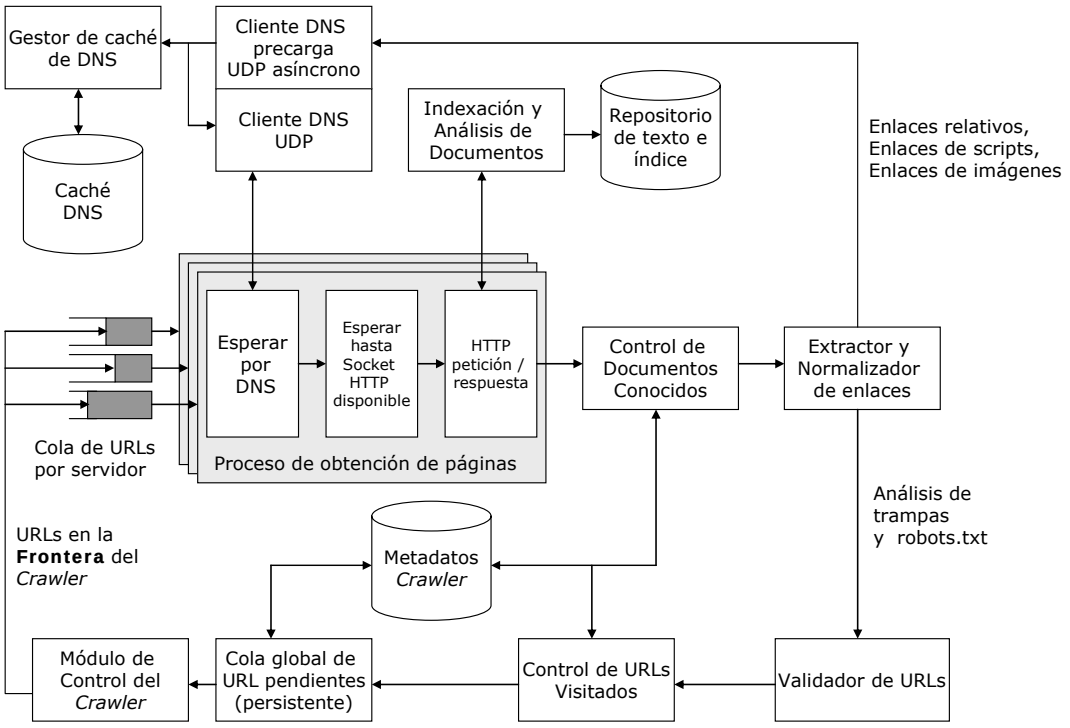


Figura 50 Arquitectura de un *crawler* web de altas prestaciones

A continuación se describen los diferentes módulos considerados en la Figura 50, que tiene que tener en cuenta un *crawler* para ser eficiente y que escale a una gran porción de la Web. Básicamente los módulos intentan establecer la arquitectura que permita responder a las siguientes preguntas: ¿Qué páginas debería descargar? ¿Cómo debería refrescar las

[22] http://www.altavista.com
[23] http://www.google.com

páginas? ¿Cómo minimizar la carga sobre los sitios visitados? ¿Cómo debería paralelizarse el proceso de *crawling*?

El ciclo de vida completo para la obtención de una página está gestionado por un hilo de ejecución lógico que no tiene por qué ser un hilo o un proceso del sistema operativo, sino que puede haber sido programado de forma específica para mejorar su eficiencia. Su tarea comienza con la resolución de nombres DNS (*Domain Name System*) y finaliza cuando la página completa ha sido obtenida vía HTTP (o se ha recibido algún mensaje de error). Una vez finalizada esta tarea, la página normalmente se almacena de forma persistente en formato comprimido y se analiza para obtener los enlaces de páginas a las que apunta. El procesamiento de los enlaces salientes se realiza en un *pool* de procesos, y un gestor de carga se encarga de chequear la correcta utilización de la red, para no sobrecargarla. Este proceso continúa hasta que el *crawler* ha recolectado un número 'suficiente' de páginas. Este número puede depender de muchos factores, que forman parte de la definición de la lógica del *crawler*.

El principal objetivo de un *crawler* es obtener el mayor número de páginas en el menor tiempo posible. Para ello tiene que minimizar, o si es posible eliminar, los diferentes momentos de inactividad que pueden producirse por alguna de las siguientes operaciones:

- Retardos introducidos durante la traducción del nombre de la máquina especificado en el URL a una dirección IP utilizando DNS.
- Retardos debidos a tiempos de establecimiento de conexión y envío de la petición del recurso al servidor.
- Tiempos de espera y procesamiento por la recepción de la página de respuesta, su almacenamiento en un repositorio local y su análisis para obtener nuevos enlaces.

La reducción de los dos primeros tiempos es misión de los módulos de precarga y caché de DNS de la Figura 50.

El otro gran cuello de botella a tener en cuenta es el tiempo de espera para la descarga de una página, que puede ser del orden de segundos. Si el *crawler* se mantuviese en espera durante segundos por una sola página web, su eficiencia se vería terriblemente penalizada. Por este motivo es necesario que un sistema de *crawling* pueda procesar múltiples conexiones de forma concurrente a documentos web.

En los siguientes apartados se comentan en detalle los restantes módulos y aspectos que ha de tratar un sistema de *crawling* para que sea escalable a las dimensiones que actualmente presenta la Web.

VI.1. NORMALIZACIÓN DE URLS

Una vez descargados los documentos web, el *crawler* debe analizarlos para poder extraer los enlaces que contengan. Pero antes de añadir esos enlaces a la cola de URLs para que puedan ser accedidos nuevamente, es necesario normalizarlos para evitar que un documento sea accedido varias veces por no haber detectado que dos URLs representan un enlace al mismo recurso.

Por ejemplo, si se analiza la página principal del Departamento de Tecnoloxías da Información e as Comunicacións de la Universidade da Coruña, el *crawler* debería detectar que los tres siguientes URLs apuntan al mismo recurso: http://www.tic.udc.es/index.html, http://www.tic.udc.es/./index.html o http://www.tic.udc.es.

Es importante tener en cuenta que la normalización no va a garantizar que se eliminen todos los URLs que apunten al mismo documento debido a que un servidor puede tener múltiples nombres. Podría pensarse en normalizar en base a la dirección IP, pero en algunos casos se utilizan balanceadores de carga que hacen que el acceso a un mismo nombre de dominio sea servido por diferentes máquinas, por temas de escalabilidad, tolerancia a fallos y, en resumen, para dar un mejor servicio a los usuarios. Tampoco sería válida la normalización por IP debido a que algunos sitios utilizan lo que se conoce como '*virtual hosting*', con lo que se consigue que una misma máquina, en función del nombre con el que se acceda, proporcione diferentes contenidos (útil cuando una organización dispone de pocas direcciones IPs y necesita publicar múltiples sitios lógicos).

VI.2. POLÍTICAS DE EXCLUSIÓN DE ROBOTS

Los servidores web pueden definir si desean o no ser indexados por los motores de búsqueda web. Para ello tienen que crear un fichero robots.txt en su directorio raíz, en el que especifican que no desean ser indexados total o parcialmente, indicando en el último caso las rutas en las que se deniega el acceso.

Se trata realmente de un protocolo de buena conducta más que de una prohibición real de los servidores. Los sistemas de *crawling* 'deben' comprobar si el servidor define un fichero robots.txt, y actuar en consecuencia, descartando los URLs que no cumplan lo especificado en ese fichero.

Otro mecanismo que pueden utilizar los administradores de sitios web para controlar la actividad de los sistemas de *crawling* es utilizar etiquetas META en las páginas, para especificar si se quiere permitir que sean indexadas o si se quiere que sean analizadas para obtener nuevos enlaces.

VI.3. CONTROL DE URLS VISITADOS

Antes de añadir un nuevo URL a procesar, los *crawlers* chequean si ya ha sido analizado previamente para evitar tratar más de una vez un mismo documento. Los sitios web contienen multitud de enlaces repetidos, por lo que la comprobación de si el URL ya ha sido visitado debe ser muy eficiente, porque es una operación que se ejecuta con cada URL obtenido. Normalmente tiene lugar calculando una función *hash* sobre el URL, diferenciando el nombre de máquina del resto del URL.

La lista de URLs visitados puede llegar a ser de un tamaño considerable, con lo que suele estar parcialmente almacenada en disco. Los sistemas de *crawling* aprovechan la

localidad espacio-temporal de avance de la exploración para tener en memoria los URLs ya visitados de los sitios más frecuentemente comprobados en cada periodo de tiempo.

Por último, si el URL no había sido previamente accedido, se añade a la cola de documentos pendientes a acceder en disco y también a la tabla de comprobaciones de URLs visitados.

VI.4. VALIDACIÓN DE URLS

Debido a que no existe control sobre el contenido de la Web, el sistema tiene que prestar especial atención a las páginas y enlaces procesados para evitar ser víctima de algún código o enlace maliciosamente añadido.

Por una parte, los analizadores de páginas HTML utilizados deben de ser implementados de forma que garanticen el tratamiento de posibles errores de forma robusta, pudiendo descartar páginas si fuese necesario. No suelen ser válidos analizadores convencionales, porque las páginas HTML suelen seguir formatos no estrictos, debido a la permisividad de los navegadores de páginas web.

Por otra parte, en la construcción de sitios web pueden utilizarse enlaces simbólicos y reescrituras de URLs que posibiliten la creación de bucles infinitos. El módulo de validación de URLs debe de intentar detectar esos casos. La mejor política es realizar estadísticas periódicas sobre el funcionamiento del *crawler* y cuando un sitio comience a dominar la colección de documentos, configurar este módulo para que elimine los URLs de ese sitio.

El módulo de validación de URLs también puede utilizarse para deshabilitar el *crawling* de ciertas partes de un sitio o para eliminar URLs que apunten a tipos de datos que claramente no son textuales como imágenes, ejecutables, etc.

VI.5. CONTROL DE DOCUMENTOS CONOCIDOS

Los *crawlers* deben de evitar acceder múltiples veces a los mismos recursos. Ese es el objetivo del módulo de control de URLs visitados (ver apartado VI.3), pero en algunos casos pese a esas comprobaciones y debido principalmente a la existencia de réplicas de sitios web, puede intentarse acceder varias veces a un recurso equivalente. En esas situaciones, lo menos grave es detectar que esa página ha sido ya accedida, para evitar tener que extraer sus enlaces.

La detección de duplicados exactos es una tarea sencilla. Simplemente es necesario añadir un *hash* del contenido de la página junto con el documento almacenado. De esta forma, cuando se obtiene un nuevo documento, se realiza su *hash* y se comprueba si existe algún documento al que ya se haya accedido con el mismo *hash*.

El problema real consiste en que, en el caso de réplicas de sitios web, es posible que haya alguna modificación en las páginas, como la fecha de actualización, o el nombre o

dirección de correo electrónico del administrador del sitio. En esos casos un *hash* no es válido y es necesario utilizar técnicas más sofisticadas, como las basadas en distancia de edición para comprobar que dos páginas son iguales, con un margen de error determinado.

VI.6. CONTROL DEL *CRAWLER*

El módulo de control del *crawler* monitoriza las peticiones y mantiene una serie de estadísticas sobre la carga actual de la red como tiempos de latencia en las peticiones o número actual y máximo de conexiones a establecer. Utiliza esta información para seleccionar y distribuir los URLs a ser encolados para ser accedidos por los procesos del *crawler*. Estas colas que almacenan los URLs a ser accedidos constituyen lo que se conoce como la frontera del *crawler*.

Los sistemas de *crawling* convencionales no son capaces de descargar todas las páginas de la Web. Los motores de búsqueda más conocidos se limitan a una pequeña fracción de la Web. En base a este hecho, es importante para los *crawlers* la selección de las páginas, priorizando los URLs en la cola de forma adecuada, de modo que la fracción de la Web visitada y que mantienen actualizada sea de la mayor calidad.

En general, suelen considerarse las siguientes políticas de ordenación de URLs:

- Recorrido en anchura (cola FIFO, *First In, First Out*). Es la política comúnmente utilizada por los sistemas de *crawling* convencional. Las razones son que permite distribuir la actividad entre diferentes servidores y esto también ayuda a evitar posibles trampas de *crawling*, como bucles infinitos.
- Recorrido en profundidad (cola LIFO, *Last In, First Out*). Este tipo de recorrido se caracteriza por premiar el recorrido de un servidor web (en todos sus niveles) antes de analizar el siguiente.
- Los mejores primero. Cola de prioridades en base a determinadas métricas, por ejemplo los documentos más enlazados primero. Ver sección II.2 para más detalles.
- Recorrido aleatorio. Se selecciona cualquiera de los URLs de la cola de forma aleatoria como siguiente recurso a ser accedido.

VI.7. COLAS DE URLS POR SERVIDOR

Cuando un *crawler* obtiene páginas de la Web, consume recursos que pertenecen a otras organizaciones. Por ejemplo, cuando el *crawler* descarga la página *p* del sitio *S*, el sitio *S* necesita recuperar la página *p* de su sistema de ficheros, consumiendo recursos de disco y CPU. Además, la página tiene que ser transferida por la red, que constituye otro recurso compartido. El *crawler* debe minimizar su impacto sobre estos recursos.

Muchos servidores HTTP comerciales están protegidos contra ataques de denegación de servicio. Este tipo de ataques consiste en inundar el servidor con peticiones frecuentes desde un mismo sitio cliente. Los servidores pueden responder de varias formas: limitando

la velocidad o frecuencia de las respuestas a una dirección IP fija, por ejemplo a tres páginas por segundo; o penalizando activamente las respuestas (esto es lo que suelen hacer los servidores ante páginas que requieren procesamiento interno en el servidor, páginas generadas dinámicamente).

Un *crawler* necesita evitar tales situaciones, no sólo por motivos de rendimiento, sino también para evitar problemas legales. Para ello suelen limitar el número de peticiones activas contra un servidor en un momento determinado, utilizando una cola de peticiones por cada servidor, que van rotando en base a un límite de tiempo. Es importante destacar que para lograr un *crawling* eficiente es necesario buscar un equilibrio entre localidad de URLs y las prácticas de buenas conductas con los servidores.

VI.8. ALMACENAMIENTO DE DOCUMENTOS

Las tareas de un *crawler* finalizan cuando almacena en un repositorio los documentos obtenidos durante su ejecución. Este repositorio de documentos podrá ser usado por multitud de sistemas y servicios, como es el caso de herramientas de indexación y clasificación de documentos.

La información relativa a los documentos web se divide en dos partes: metadatos y contenido del documento.

Los metadatos contienen información acerca de los documentos descargados como fecha de modificación, tipo del documento, tamaño, URL y cualquier otro dato que pueda ser interesante para los sistemas que vayan a utilizar el repositorio generado. La metainformación es relacional por naturaleza, pero normalmente se gestiona por software especializado debido a que las bases de datos relacionales no están orientadas para soportar de forma eficiente actualizaciones concurrentes.

El contenido suele almacenarse en formato comprimido, distribuido en múltiples servidores, etc., dependiendo de los recursos disponibles y de la finalidad que tenga el repositorio generado.

VI.9. ACTUALIDAD DE LOS DOCUMENTOS

Idealmente, el índice de un buscador debería de mantenerse constantemente actualizado, reflejando la versión más reciente de los documentos recolectados. En la práctica no es factible, porque no existe ningún mecanismo general que notifique los cambios en las páginas. En realidad, un *crawler* web nunca termina su trabajo; simplemente se detiene cuando ha obtenido 'suficientes' páginas. La mayor parte de los motores de búsqueda, en cuanto finalizan un *crawling* comienzan de nuevo.

La frecuencia de cambio en las páginas web puede ser muy diferente. Muchas de las páginas no cambian entre dos recorridos, pero existen otras que cambian varias veces. El *crawler* necesita decidir qué páginas volver a visitar y cuáles no. Existen algunos trabajos

que se ocupan de realizar estimaciones respecto al tiempo de cambio de las páginas, para mejorar la eficiencia de los *crawlers* incrementales como [CG00][CG00b].

VI.10. PROCESAMIENTO DISTRIBUIDO

Muchos motores de búsqueda web utilizan una arquitectura centralizada como la comentada, pero existen diferentes variantes. Debido al enorme tamaño de la Web, los *crawlers* suelen ejecutarse sobre múltiples máquinas y descargar páginas en paralelo, para obtener una gran cantidad de recursos en un tiempo razonable.

Los *crawlers* paralelos deben de estar adecuadamente coordinados, para que diferentes *crawlers* no visiten los mismos sitios web en múltiples ocasiones. Esta coordinación puede requerir un gran número de comunicaciones entre los diferentes componentes, pudiendo limitar el número de componentes de *crawling* simultáneos. En [CG02] se presenta la arquitectura de un *crawler* paralelo, considerando las diferentes aproximaciones posibles a cada una de las problemáticas que es necesario abordar para implementarlo, y las ventajas de cada una de ellas.

VII. ANEXO B. ARQUITECTURA DE HIWE

Crawling the Hidden Web [RG01] es uno de los artículos pioneros sobre *crawling* de la Web Oculta. Los autores desarrollaron un *crawler* llamado HiWE (*Hidden Web Exposer*), semi-automático y orientado a tareas de *crawling* específicas. Este *crawler* necesita asistencia humana para proporcionar un conjunto inicial de valores de consulta relevantes para la tarea o aplicación considerada. Estos valores serán utilizados por el *crawler* para rellenar los formularios de búsqueda que encuentre, si bien una vez iniciado el proceso, el propio sistema puede ir añadiendo nuevos valores de búsqueda. Está orientado hacia la extracción selectiva de porciones de la Web Oculta, es decir, extrae documentos en base a los requisitos de una tarea o aplicación concreta, no todos los documentos posibles.

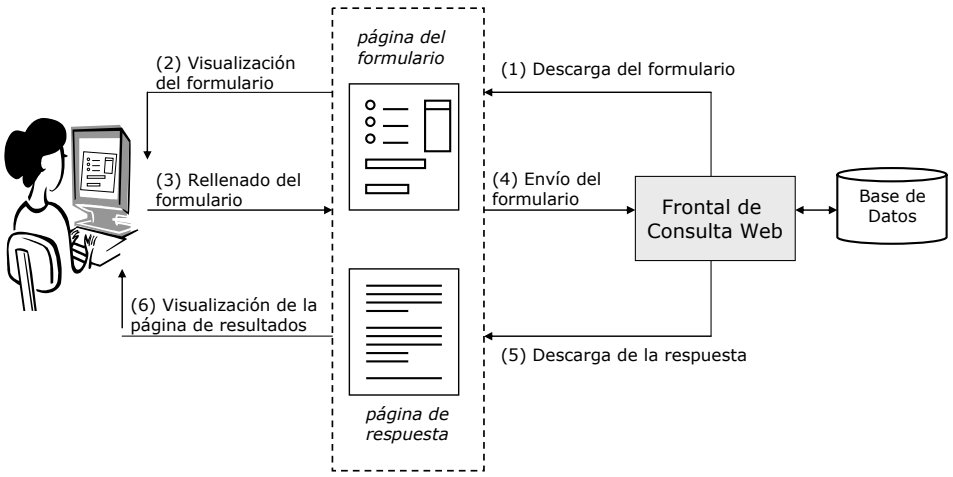


Figura 51 Interacción típica de un usuario contra una interfaz de consulta web

La idea del sistema que los autores proponen es intentar automatizar los pasos que sigue un usuario humano para consultar un formulario web. En la Figura 51 se muestra la secuencia de pasos típica de un usuario ante un formulario de consulta. Un usuario solicita la descarga de una página que contiene un formulario web (1). La página es visualizada en el navegador del usuario (2). El usuario rellena cada campo del formulario con los datos

adecuados (3), y lo envía al servidor (4). Ante ese envío, el servidor devuelve una página de resultado (5) que se visualiza en el navegador del usuario (6).

En la Figura 52 se muestran los elementos necesarios para un sistema que sea capaz de automatizar la interacción de un usuario contra una interfaz de consulta, restringiendo su actividad a un dominio de información concreto, definido por una tarea que guíe el proceso.

– Un componente que analice el formulario para obtener una representación interna del mismo como un conjunto de elementos de entrada estándares (listas de selección, *checkboxes*, cajas de texto, áreas de texto y *radio buttons*), información de envío (lo necesario para hacer su envío) y metainformación acerca del formulario (página y sitio web donde se encuentra, otros textos en la página no relativos al formulario, etc.).

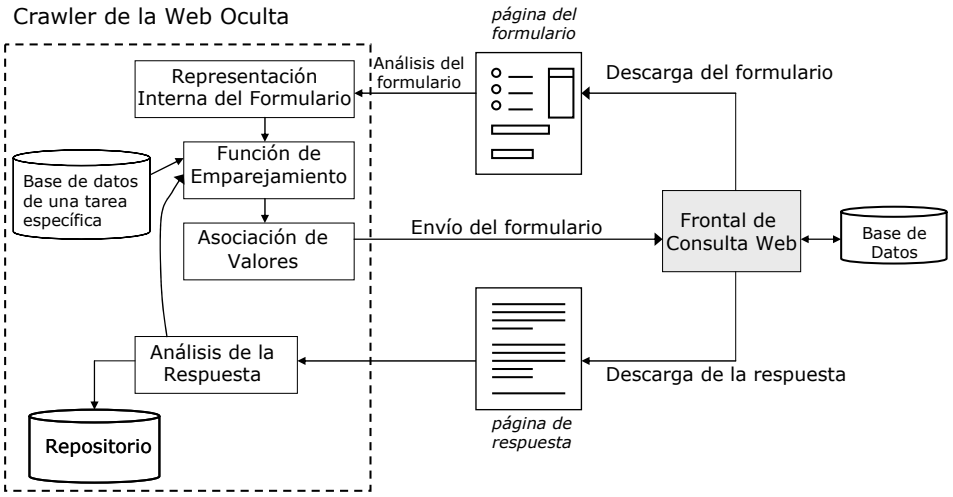


Figura 52 Modelo de operación genérico para un *crawler* de la Web Oculta

– Una base de datos relativa a la tarea o aplicación, que contiene información que el *crawler* necesita para poder realizar consultas de búsqueda relevantes para la tarea.

– Una función de emparejamiento que toma como entrada una representación interna de un formulario, y el contenido de la base de datos comentada en el punto anterior, y genera como salida un conjunto de asignaciones entre valores contenidos en la base de datos y elementos del formulario. Se trata del componente encargado de rellenar el formulario.

– Un módulo de análisis de la página de respuesta al envío del formulario al servidor, para almacenarlas en un repositorio, intentando distinguir entre páginas que contienen resultados y páginas de error.

En la Figura 53 se presenta la arquitectura general del *crawler* HiWE.

La *Frontera* del *crawler* es el módulo que contiene la lista de URLs a los que el *crawler* debe acceder para realizar su procesamiento. Se inicializa con un conjunto de URLs, que se va completando de forma automática durante el proceso de *crawling* con los enlaces de las páginas analizadas.

El módulo *Crawler* controla el proceso de *crawling*, decide qué URL de la lista de URLs es el siguiente a visitar y descarga el documento referenciado.

El *Analizador de documentos* se encarga de analizar los documentos web descargados, extrayendo los URLs convencionales y los puntos de acceso a la Web Oculta del lado servidor, es decir, los formularios. Las páginas que no contienen formularios sólo son tratadas por este módulo y los anteriores.

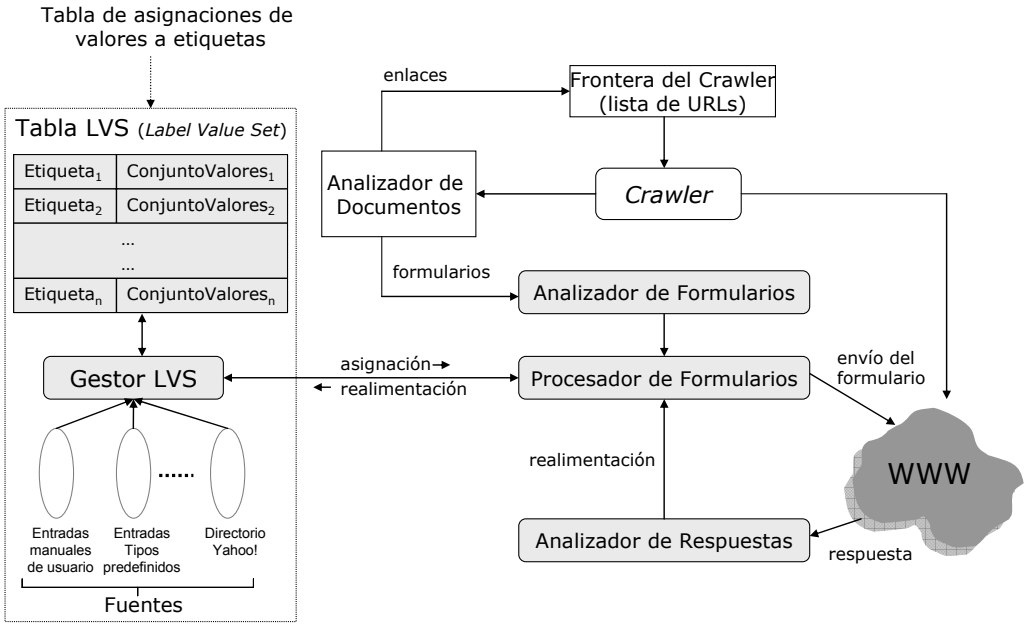


Figura 53 Arquitectura de alto nivel del *crawler* HiWE

El *Analizador de Formularios*, el *Procesador de Formularios* y el *Analizador de Respuestas* implementan las operaciones de modelado, procesamiento y envío de formularios respectivamente. El modelado de formularios consiste en establecer relaciones entre elementos del formulario y los identificadores que los desarrolladores web sitúan en los documentos, para que el usuario sea capaz de identificar el significado del elemento. Las relaciones se determinan mediante heurísticas basadas en distancias visuales entre elementos e identificadores. Una vez modelado el formulario, el sistema asigna posibles valores a los elementos del formulario para realizar consultas. Estos valores deben ser coherentes con el contexto al que pertenece el formulario. Para ello utiliza bases de datos de dominios específicos para mantener una tabla de asignaciones entre etiquetas y conceptos relevantes denominada LVS (*Label Value Set*). Una vez asociados los valores de consulta a todos los elementos del formulario, el sistema es capaz de ejecutar la consulta y descargar el documento generado para continuar con el proceso cíclico de *crawling*.

Estos tres últimos módulos se describen en detalle en los apartados II.3.3.4, II.3.3.5 y II.3.3.6 respectivamente.

VIII. ANEXO C. PROTOTIPO

Para validar las técnicas propuestas se ha implementado DeepBot, un prototipo de sistema de *crawling* dirigido de la Web Oculta. DeepBot está implementado sobre un software de *crawling* convencional denominado WebBot, incluido en el producto Denodo Aracne [DNDARN], que ha sido extendido para proporcionarle las funcionalidades de acceso a la información contenida en la Web Oculta, implementando las técnicas propuestas.

WebBot es un software de *crawling* de propósito general. Parte de una lista inicial de URLs, un límite de la profundidad del recorrido a realizar y un conjunto de expresiones regulares que limitan su amplitud. Como gestor de descargas de recursos web utiliza un cliente HTTP. El gestor de contenido está formado por una serie de filtros que se aplican sobre los documentos descargados. Entre esos filtros se incluyen un analizador de HTML que extrae y reescribe enlaces, uno que elimina enlaces a los que ya se ha accedido previamente, un filtro de URLs en base a la lista de expresiones regulares establecidas en la configuración inicial del proceso de *crawling* y otro filtro que almacena los documentos en el repositorio de datos y de metainformación. Entre la metainformación del documento se incluyen datos como su tipo MIME y su URL real. El sistema es multitarea y cada tarea realiza el proceso que se acaba de comentar, obteniendo y añadiendo nuevos URLs de la frontera del *crawler*. Adicionalmente, WebBot indexa los documentos obtenidos utilizando el software de libre distribución Apache Lucene[24], y proporciona una API de búsqueda sobre el índice generado, para permitir la creación de buscadores sobre el contenido recolectado por el sistema de *crawling*.

DeepBot cambia radicalmente la filosofía de WebBot, para adaptarse a la arquitectura propuesta en el apartado III.2, e incluir la implementación de todas las técnicas y algoritmos definidos. La arquitectura de DeepBot se muestra en la Figura 54. El sistema se divide en tres partes bien diferenciadas: el módulo de *crawling*, el módulo de indexación y el módulo de búsqueda.

Los siguientes apartados describen cada uno de los módulos del sistema.

[24] http://lucene.apache.org

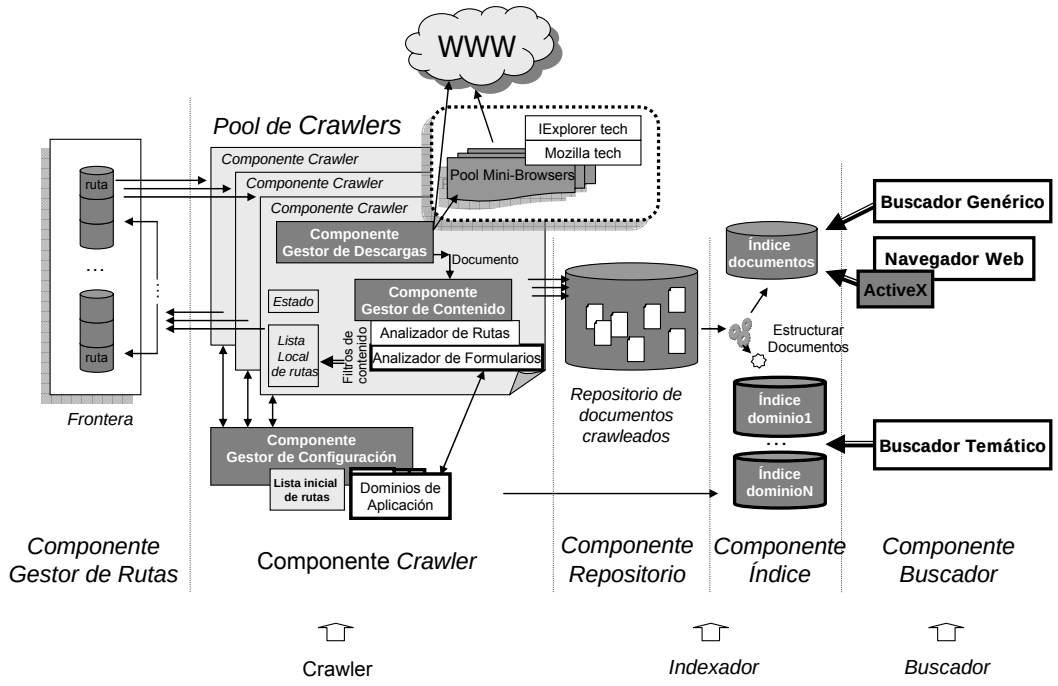


Figura 54 Arquitectura del prototipo

VIII.1. MÓDULO DE CRAWLING

Para posibilitar el acceso a los recursos situados en la Web Oculta, DeepBot considera *rutas* en lugar de URLs (ver apartado III.3.1), utiliza un conjunto de definiciones de dominios de aplicación como las introducidas en III.1.3 para guiar el proceso de *crawling* hacia páginas y formularios relevantes y extiende sus componentes de acceso y análisis de documentos para que soporten páginas dinámicas:

- DeepBot utiliza un mini-navegador web como gestor de descargas. Esto posibilita el acceso a los recursos web sin tener que considerar los problemas relacionados con el tratamiento de tecnologías del lado cliente. Este mini-navegador es capaz de ejecutar secuencias de navegación expresadas en el lenguaje NSEQL (descrito en el apartado III.1.2.1), e incluye la implementación del algoritmo de gestión de navegaciones sobre páginas dinámicas. Actualmente se dispone de implementaciones sobre los navegadores más conocidos [MF07] [MSIE07].
- DeepBot añade un nuevo filtro al gestor de contenido, encargado de detectar la existencia de formularios en las páginas analizadas. Además de añadir nuevas rutas a partir de los enlaces de los documentos analizados, también obtiene nuevas rutas de otros elementos dinámicos que generen nuevas navegaciones o enlaces en la página, y otras representando consultas sobre aquellos formularios que hayan sido

considerados relevantes respecto a las definiciones de dominios especificadas en su configuración.

- DeepBot utiliza una implementación sencilla de técnicas de *crawling* dirigido a nivel de superficie, basado en las expresiones regulares especificadas en la configuración inicial del sistema y que se aplican sobre las rutas de los diferentes recursos.
- DeepBot extiende también el modelo de documento, para considerar todos los elementos de las páginas dinámicas. El nuevo modelo de documento incluye, además de formularios, textos y elementos que hayan registrado algún manejador de eventos, las coordenadas visuales de los diferentes elementos, necesarias para poder realizar el modelo completo del formulario.

VIII.2. MÓDULO DE INDEXACIÓN

A partir de los documentos recolectados, DeepBot genera dos índices. En uno de ellos almacena los documentos y su metainformación, para permitir realizar búsquedas no estructuradas sobre ellos. Entre su metainformación incluye tanto el URL del documento como la secuencia NSEQL de acceso al mismo.

Adicionalmente, para cada documento que haya sido generado a partir de una consulta sobre un formulario, DeepBot extrae de forma automática los registros de datos contenidos en esa página de resultados, y los etiqueta de acuerdo al dominio al que pertenezca el formulario de origen. Para cada registro obtenido, DeepBot almacena metainformación que permite acceder tanto a la página del formulario como a la que contiene el registro, y la consulta que es necesario realizar sobre el formulario para generar esa página.

VIII.3. MÓDULO DE BÚSQUEDA

Sobre los documentos recolectados, DeepBot permite realizar consultas por palabra clave para obtener aquellos documentos relevantes para un usuario. Entre los documentos obtenidos como resultado de la consulta es posible que aparezcan referenciados recursos que se encuentran en la Web Oculta.

Adicionalmente, también permite realizar consultas más precisas sobre el índice que contiene los registros de datos estructurados. En la Figura 55 se muestra la interfaz para consultas estructuradas. En el ejemplo se está buscando ‘Thinking in java’ sobre el campo TÍTULO de los registros de datos. Como resultado se muestran los registros que concuerdan con esa búsqueda y la referencia al documento que lo contiene. También se posibilita el acceso a la página del formulario y a la consulta que se ha realizado.

Para permitir el acceso a páginas de la Web Oculta se hace uso de la secuencia NSEQL asociada a cada uno de los recursos. En la Figura 55 se muestra un fragmento de la secuencia asociada a la página en la que se encuentra el registro devuelto. Para reproducir

el acceso desde el navegador a ese recurso, se ha desarrollado un componente ActiveX para el navegador Microsoft Internet Explorer, que ejecuta de forma secuencial cada comando de la secuencia de navegación, visualizando el resultado en el navegador.

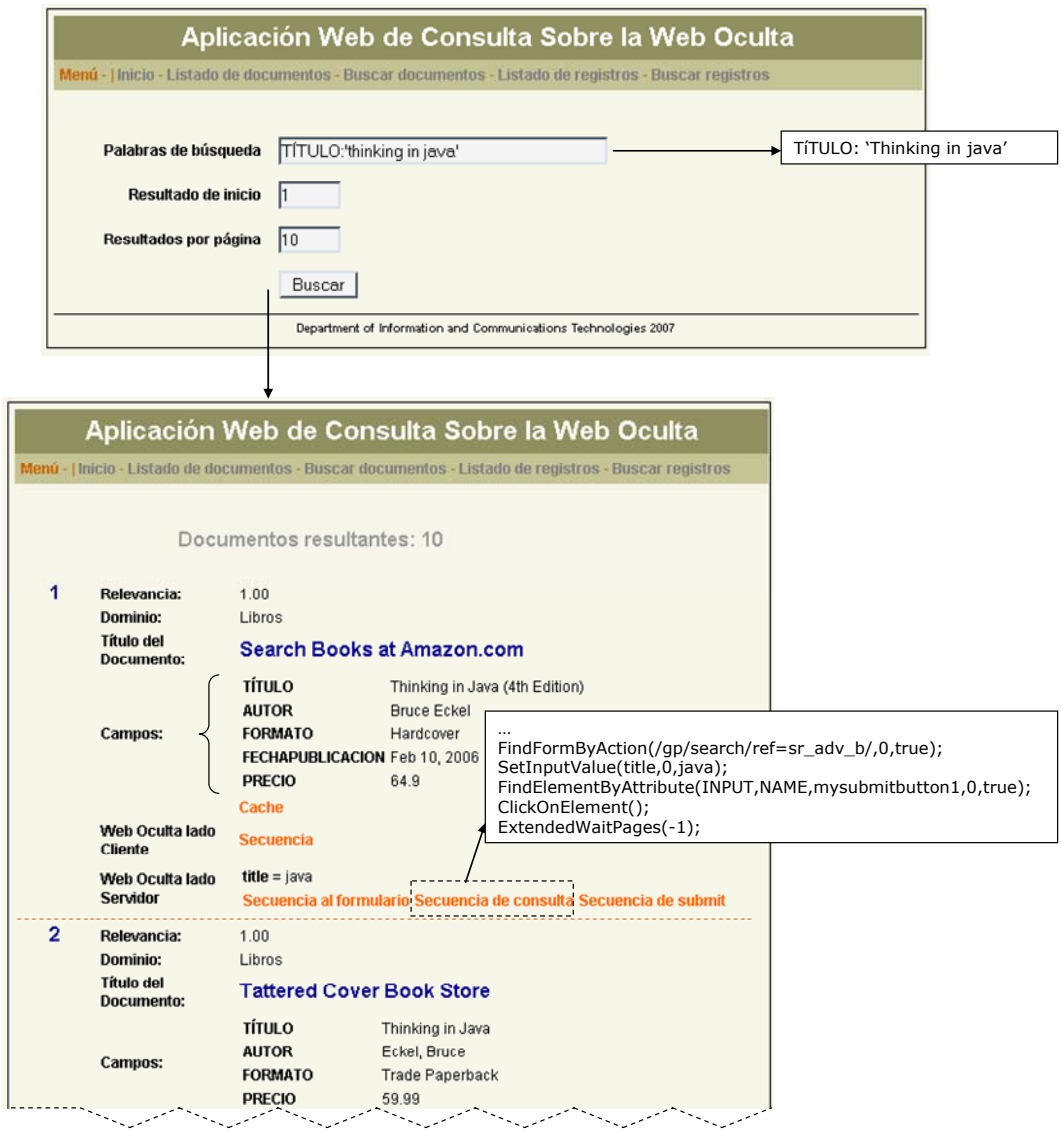


Figura 55 Aplicación web de consulta del prototipo

IX. REFERENCIAS

[Adelberg98] B. Adelberg. NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents. *SIGMOD Record 27(2):283-294*. 1998

[ACMM03] L. Arlota, V. Crescenzi, G. Mecca, and P. Merialdo. Automatic annotation of data extracted from large websites. *En Proceedings of the WebDB Workshop, pp. 7-12*. 2003

[ACGP+01] A. Arasu, J. Cho, H. García-Molina, A Paepcke, and S. Raghavan. Searching the Web. *Journal of ACM Transactions on Internet Technology, vol. 1(1)*. Agosto 2001

[AG03a] A. Arasu, and H. García-Molina. Extracting Structured Data from Web Pages. *En Proceedings of the ACM SIGMOD International Conference on Management of data, pp. 337-348*. 2003

[AG03b] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. *En Proceedings of 19th International Conference on Data Engineering (ICDE)*. 2003

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. *Addison Wesley*. 1995

[AIG03] E. Agichtein, P. Ipeirotis, and L. Gravano. Modelling query-based access to text databases. *En Proceedings of 6th International Workshop on the Web and Databases (WebDB)*. 2003

[AJAX05] Jesse James Garrett. Ajax: A New Approach to Web Applications. *Adaptive Path*. Febrero 2005

[Allen97] C. Allen. WIDL: Application Integration with XML. *Journal of World Wide Web 2(4)*. 1997

[AM98] G. O. Arocena, and A. O. Mendelzon. WebOQL: Restructuring Documents, Databases, and Webs. *En Proceedings of the 14th International Conference on Data Engineering, pp. 24-33*. 1998

[APRB+07] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. Using Clustering and Edit Distance Techniques for Automatic Web Data

Extraction. *En Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE). Lecture Notes in Computer Science. Springer Berlin/Heidelberg.* Aceptado para publicación. Diciembre 2007

[APRB+07b] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. Using Finding and Extracting Data Records from Web Pages. *En Proceedings of the 2007 IFIP International Conference on Embedded and Ubiquitous Computing (EUC). Lecture Notes in Computer Science. Springer Berlin/Heidelberg.* Aceptado para publicación. Diciembre 2007

[APRB+08] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. Extracting Lists of Data Records from Semi-structured Web Pages. *Data & Knowledge Engineering Journal*. Aceptado para publicación. 2008

[APRC+07] M. Álvarez, A. Pan, J. Raposo, F. Cacheda, F. Bellas, and V. Carneiro. DeepBot: A Focused Crawler for Accessing Hidden Web Content. *En Proceedings of the 3rd International Workshop on Data Engineering Issues in E-Commerce and Services, in conjunction with ACM Conference on Electronic Commerce (EC) (DEECS). ACM Digital Library. ISSN: 978-1-59593-856-5.* Junio 2007

[APRH06] M. Álvarez, A. Pan, J. Raposo, and J. Hidalgo. Crawling Web Pages with Support for Client-Side Dynamism. *En Proceedings of the 7th International Conference, Advances in Web-Age Information Management (WAIM). Lecture Notes in Computer Science. Springer Berlin/Heidelberg. ISSN: 0302-9743, ISBN-10: 3-540-35225-2, ISBN-13: 978-3-540-35225-9. Vol. 4016, pp. 252-262.* Junio 2006

[APRV04] M. Álvarez, A. Pan, J. Raposo, and A. Viña. Client-Side Deep Web Data Extraction. *En Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East). ISBN: 0-7695-2206-8, pp.158-161.* Septiembre 2004

[ARCP06] M. Álvarez, J. Raposo, F. Cacheda, and A. Pan. A Task-specific Approach for Crawling the Deep Web. *En las Journal Engineering Letters. Special Issue: 'Advances in Information Engineering' (EL). International Association of Engineers. ISSN: 1816-0948, ISBN: 1816-093X, 13:2, pp. 204-215.* Septiembre 2006.

[Baeza89] R. A. Baeza-Yates. Algorithms for String Searching: A Survey. *SIGIR Forum 23(3-4):34-58*. 1989

[BB98] K. Bharat and A. Bröder. A technique for measuring the relative size and overlap of public Web search engines. *En Proceedings of the 7th International World Wide Web Conference (WWW7)*. 1998

[BC03] A. Bergholz, B. Chidlovski. *Crawling* for Domain-Specific Hidden Web Resources. *En Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE)*. 2003

[Bergman00] M. Bergman. The Deep Web. Surfacing Hidden Value. *Technical report, BrightPlanet LLC*. Diciembre 2000

[BF05] L. Barbosa, and J. Freire. Searching for Hidden-Web Databases. *En Proceedings of 8th International Workshop on the Web and Databases (WebDB)*. 2005

[BF07] L. Barbosa, and J. Freire. An Adaptative Crawler for Locating Hidden-Web Entry Points. *En Proceedings of the 16th International World Wide Web Conference (WWW16), pp. 441-450*. Mayo 2007

[BFG01] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. *En Proceedings of 27th Conference on Very Large Databases (VLDB), pp. 119-128*. 2001

[BKMR+00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. *En Proceedings of the 9th International Conference on the World Wide Web*. 2000

[BP98] S. Brin, and L. Page. The Anatomy of a Large-Scale Hypertextual Search Engine. *En Proceedings of the 7th International World Wide Web Conference*. 1998

[BR99] R. Baeza-Yates, and B. Ribeiro-Neto. Modern Information Retrieval. *ISBN: 0-201-39829-X. Addison Wesley*. 1999

[CBD99] S. Chakrabarti, M. van den Berg, and B. Dom. Focused *Crawling*: a new Approach to Topic-Specific Web resource Discovery. *En Proceedings of the 8th International World Wide Web Conference (WWW8). Computer Networks, 31, pp. 1623-1640*. 1999

[CC01] J. Callan, and M. Connell. Query-based Sampling of Text Databases. *En ACM Transactions on Information Systems Journal, Vol. 19, pp. 97-130*. 2001

[CCH03] J. Cope, N. Craswell, and D. Hawking. Automated Discovery of Search Interfaces on the Web. *En Proceedings of the 14th Australasian Database Conference (ADC)*. 2003

[CG02] J. Cho, and H. García-Molina. Parallel Crawlers. *En Proceedings of the 12nd International World Wide Web Conference (WWW12)*. Mayo 2002

[CGP98] J. Cho, H. García-Molina, and L. Page, L. Efficient *Crawling* through URL Ordering. *En Proceedings of the 7th International World Wide Web Conference (WWW7)*. 1998

[Chakrabarti03] S. Chakrabarti. Mining The Web: Discovering Knowledge from Hypertext Data. *ISBN: 1-55860-754-4. Morgan Kaufmann Publishers*. 2003

[CHLP+04] K. C.-C. Chang, B. He, M. Patel, C. Li, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record, 33(3)*. Septiembre 2004

[CHZ04] K. C.-C. Chang, B. He, and Z. Zhang. MetaQuerier over the Deep Web: Shallow Integration Across Holistic Sources. *En Proceedings of the VLDB Workshop on Information Integration on the Web (VLDB-IIWeb)*. 2004

[CHZ05] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large-Scale Integration: Building a MetaQuerier over Databases on the Web. *En Proceedings of CIDR, pp. 44-55*. 2005

[CL01] C.H. Chang, and S.C. Lui. IEPAD: information extraction based on pattern discovery. *En Proceedings of 10th International Wold Wide Web Conference (WWW10), pp. 681-688*. 2001

[CM99] M. E. Califf, and R. J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. *En Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, pp. 328-334*. 1999

[CMM01] V. Crescenzi, G. Mecca, and P. Merialdo. ROADRUNNER: Towards Automatic Data Extraction from Large Web Sites. *En Proceedings of the 27th International Conference on Very Large Databases (VLDB), pp. 109-118*. 2001

[CM98] V. Crescenzi, and G. Mecca. Grammars Have Exceptions. *Information Systems 23(8):539-565*. 1998

[CRF03] W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. *En Proceedings of IJCAI-03 Workshop (IIWeb)*. 2003

[CS96] W. Cohen and Y. Singer. Learning to query the web. *En Proceedings of AAAI Workshop on Internet-Based Information Systems*. 1996

[DCLG+00] M. Diligenti, F. Coetzee, S. Lawrence, C.L. Giles, and M. Gori. Focused *Crawling* using context graphs. *En Proceedings of 26th International Conference on Very Large Databases (VLDB), pp. 527-534*. 2000

[DNDARN] Denodo Aracne. *http://www.denodo.com/castellano/aracne.html*

[DNDITP] Denodo ITPilot. *http://www.denodo.com/castellano/denodo_itpilot.html*

[DP94] P. M. E. De Bra, and R. D. J. Post. Information retrieval in the World Wide Web: Making client-based searching feasible. *En Proceedings of the 1st International World Wide Web Conference (WWW1)*. 1994

[Freitag00] D. Freitag. Machine Learning for Information Extraction in Informal Domains. *Machine Learning 39(2/3):169-202*. 2000

[GBS92] G.H. Gonnet, R.A. Baeza-Yates, and T. Snider. New Indices for Text: Pat trees and Pat Arrays. *Information Retrieval: Data Structures and Algorithms, Prentice Hall, New Jersey, pp. 66-82*. 1992

[CG00] J. Cho, and H. García-Molina. The evolution of the web and implications for an incremental *crawler*. *En Proceedings of the 26th International Conference on Very Large Databases*. 2000

[CG00b] J. Cho, and He. García-Molina. Synchronizing a database to improve freshness. *En Proceedings of the International Conference on Management of Data*. 2000

[GCGP97] L. Gravano, C.-C. K. Chang, H. García-Molina, and A. Paepcke. STARTS: Stanford Proposal for Internet Meta-searching, *pp. 207-218*. 1997

[GIS03] L. Gravano, P. Ipeirotis, and M. Sahami. QProber: A System for Automatic Classification of Hidden-Web Databases. *En ACM Transactions on Information Systems, vol. 21(1)*. Enero 2003

[Golin91] E. J. Golin. Parsing visual languages with picture layout grammars. *Journal of Visual Languages and Computing, 4(2):371-394*. 1991

[GRV98] J. Gruser, L. Raschid, M. Vidal, and L. Bright. Wrapper Generation for Web Accesible Data Sources. *En Proceedings of the Third International Conference on Cooperative Information Systems, pp. 14-23*. 1998

[GS05] A. Gulli, and A. Signorini. The Indexable Web is More than 11.5 Billion Pages. *En Proceedings of Wold Wide Web Conference (WWW)*. 2005

[HC03] B. He and K. C.-C. Chang. Statistical Schema Matching Across Web Query Interfaces. *En Proceedings of the ACM SIGMOD Conference, pp. 217-228*. 2003

[HC06] B. He, and K. C.-C. Chang. Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach. *En ACM Transactions on Database Systems (TODS), vol. 31(1)*. Marzo 2006

[HD98] C. N. Hsu, and M.T. Dung. Generating Finite-state transducers for semi-structured data extraction from the Web. *Information Systems 23(8):521-538*. 1998

[HFAN98] G. Huck, P. Fankhauser, K. Aberer, and E. J. Neuhold. Jedi: Extracting and Synthesizing Information from the Web. *En Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, pp. 32-43*. 1998

[HJMP+98] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm – an application: Tailored Web site mapping. *En Proceedings of the 7th World Wide Web Conference (WWW7)*. 1998

[HMG97] J. Hammer, J. McHugh, and H. García-Molina. Semistructured Data: The Tsimmis Experience. *En Proceedings of the 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS), pp. 1-8*. 1997

[HMYW04] H. He, W. Meng, C. Yu, and Z. Wu. Automatic Integration of Web Search Interfaces with WISE-Integrator. *Very Large Databases Journal, Vol.13, No.3, pp. 256-273*. Septiembre 2004. (Mejor artículo en VLDB 2003)

[HMYW05a] H. He, W. Meng, C. Yu, and Z. Wu. Constructing Interface Schemas for Search Interfaces of Web Databases. *En Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE)*. 2005

[HMYW05b] H. He, W. Meng, C. Yu, and Z. Wu. WISE-Integrator: A System for Extracting and Integrating Complex Web Search Interfaces on the Deep Web. *En Proceedings of the 31th International Conference on Very Large Databases (VLDB)*. 2005

[HMO91] R. Heml, K. Marriot and M. Odersky. Building visual language parsers. *En Proceedings on Human Factors in Computing Systems (CHI), pp. 105-112*. 1991

[HYJS06] Y. Hedley, M. Younas, A. James, and M. Sanderson. Sampling, information extraction and summarisation of Hidden Web databases. *Data & Knowledge Engineering Journal*. 2006

[HN99] A. Heydon, and M. Najork. Mercator: A scalable, extensible Web crawler. *En Proceedings of the 8th International World Wide Web Conference (WWW8), 2(4), pp. 219-229*. 1999

[IG02] P. Ipeirotis, and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. *En Proceedings of the 28th International Conference on Very Large Databases (VLDB)*. 2002

[IGS01] P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden web databases. *En Proceedings of the ACM SIGMOD Conference*. 2001

[INCG05] P. Ipeirotis, A. Ntoulas, J. Cho, and L. Gravano. Modeling and Managing Content Changes in Text Databases. *En Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE)*. 2005

[JAS] JavaScript Language Specification.

http://developer.mozilla.org/en/docs/JavaScript

[KBGP01] O. Kaljuvee, O. Buyukkokten, H. García-Molina, and A. Paepcke. Efficient Web Form Entry on PDAs. *En proceedings of the 10th Wold Wide Web Conference (WWW10), ACM 1-58113-348-0/01/0005*. 2001

[KEW01] C. Kwork, O. Etzioni, and D.S. Weld. Scaling question answering to the Web. *En Proceedings of the 10th International World Wide Web Conference (WWW10), vol. 10, pp. 150-161*. Mayo 2001

[KM98] T. Kistlera, and H. Marais. WebL: A Programming Language for the Web. En *Proceedings of the 7th International World Wide Web Conference (WWW7)*, pp 259-270. 1998

[KWD97] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper Induction for Information Extraction. *En Proccedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI), pp. 729-737*. 1997

[Kleinberg99] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM, Vol. 46, No 5, pp. 604-632*. 1999

[KRRT02] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The Web and Social Networks. *IEEE Computer Journal, pp. 32-36*. 2002

[Kushmerick03] N. Kushmerick. Learning to Invoke Web Forms. *En Proceedings of the Internacional Conference Ontologies, Databases and Applications of Semantics (CoopIS/DOA/ODBASE), pp. 997-1013*. 2003

[LESY02] S. Liddle, D. Embley, D. Scott, and S. Yau Ho. Extracting Data Behind Web Forms. *En Proceedings of the 28th International Conference on Very Large Databases (VLDB)*. 2002

[Levenshtein66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10, pp. 707–710. 1966

[LGMK04] K. Lerman, L. Getoor, S. Minton, and C. Knoblock. Using the structure of web sites for automatic segmentation of tables. *En Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. pp. 119-130*. 2004

[LG98] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science, 280(5360):98-100*. 1998

[LHLM+98] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. *Information Systems 23(8):589-613*. 1998

[LMK03] K. Lerman, S. Minton, C. Knoblock. Wrapper Maintenance: A Machine Learning Approach. *Journal of Artificial Intelligence Research 18, pp. 149-181*. 2003

[LPH00] L. Liu, C. Pu and W. Han. XWRAP: An XML-enabled wrapper construction system for web information sources. *En Proceedings of the 16th International Conference on Data Engineering, pp. 611-621*. 2000

[LRLC04] V. Z. Liu, J. C. Richard, C. Luo, and W. W. Chu. DPro: A probabilistic approach for hidden web database selection using dynamic probing. *En*

Proceedings of 20th International Conference on Data Engineering (ICDE). 2004

[LRS02] A.H.F. Laender, B.A. Ribeiro-Neto, and A. S. da Silva, Data Extraction By Example. *Data and Knowledge Engineering 40(2):121-154*. 2002

[LRST02] A.H.F. Laender, B.A. Ribeiro-Neto, A. S. da Silva, J.S. Teixeira. A Brief Survey of Web Data Extraction Tools. *ACM SIGMOD Record 31(2):84-93*. 2002

[LYE01] S.W. Liddle, S.H. Yau, and D.W. Embley. On the Automatic Extraction of Data from the Hidden Web. *En Proceedings of the International Workshop on Data Semantics in Web Information Systems (DASWIS), pp. 27-30*. 2001.

[MF07] Mozilla Firefox. http://www.mozilla.com/

[MMK01] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical Wrapper Induction for Semistructured Information Sources. *Autonomous Agents and Multi-Agent Systems 4(1-2):93-114*. 2001

[MMK03] I. Muslea, S. Minton, and C. A. Knoblock. Active learning with strong and weak views: A case study on wrapper induction. *En Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 415-420*. 2003

[Morrison68] D.R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded in Alphanumeric. *The Journal of ACM 15(4):514-534*. 1968

[MR07] Mozilla Rhino - *JavaScript* Engine (Java). *http://www.mozilla.org/rhino/*

[MSM07] Mozilla SpiderMonkey - *JavaScript* engine (C). *http://www.mozilla.org/js/spidermonkey/*

[MSIE07] Microsoft Internet Explorer.

http://www.microsoft.com/windows/ie/default.mspx

[MySimon] mySimon - *http://www.mysimon.com*

[Notredame02] C. Notredame. Recent progresses in multiple sequence alignment: a survey, Pharmacogenomics 31(1):131-144. 2002

[NZC05] A. Ntoulas, P. Zerfos, and J. Cho. Downloading Textual Hidden Web Content Through Keyword Queries. *En Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 2005

[OAI_PMH] *http://www.openarchives.org/OAI/openarchivesprotocol.html*

[Porter80] M. F. Porter. An algorithm for suffix stripping. *Program 14, 3, pp. 130–137*. 1980

[PRAC+07] A. Pan, J. Raposo, M. Álvarez, V. Carneiro, and F. Bellas. Automatically Maintaining Navigation Sequences for Querying Semi-structured Web Sources. *Data & Knowledge Engineering Journal (DKE). Elsevier. ISSN: 0169-023X. Vol. 63, issue 3, pp.793-808*. Diciembre 2007

[PRAH+02] A. Pan, J. Raposo, M. Álvarez, J. Hidalgo, and A. Viña. Semi-Automatic Wrapper Generation for Commercial Web Sources. *En Proceedings of IFIP WG8.1 Working Conference on Engineering Information Systems in the Internet Context (EISIC)*. 2002

[RPB07] J. Raposo, A. Pan, F. Bellas. Técnicas de Mantenimiento Automático de Programas 'Envoltorio' para Fuentes de Datos Web Semi-estructuradas. *Tesis Doctoral Universidade da Coruña, España.* 2007

[RB01] E. Rahm, and P. A. Bernstein. A survey of approaches to automatic schema matching. *Very Large Databases Journal, Vol.10, No.4, pp. 334-350*. 2001

[RG00] S. Raghavan, and H. García-Molina. *Crawling* the hidden Web. *Technical Report 2000-36, Computer Science Department, Stanford University*. Diciembre 2000. *(Disponible en http://dbpubs.stanford.edu/pub/2000-36)*

[RG01] S. Raghavan, and H. García-Molina. *Crawling* the Hidden Web. *En Proceedings of the 27th International Conference on Very Large Databases (VLDB)*. 2001

[RLS02] B.A. Ribeiro-Neto, A.H.F. Laender, and A. S. da Silva. Extracting Semi-Structured Data Through Examples. *En Proceedings of the Eighth ACM International Conference on Information and Knowledge Management, pp. 94-101*. 1999

[RM99] J. Rennie, and A. MacCallum. Using reinforcement learning to spider the Web efficiently. *En Proceedings of 16th International Conference on Machine Learning, pp. 335-343*. 1999

[RPAH07] J. Raposo, A. Pan, M. Álvarez, and J. Hidalgo. Automatically Maintaining Wrappers for Semi-Structured Web Sources. *Data & Knowledge Engineering Journal (DKE). Elsevier. ISSN: 0169-023X. Vol. 61, issue 2, pp.331-358*. Mayo 2007

[SA01] A. Sahuguet, F. Azavant. Building Intelligent Web Applications Using Lightweight Wrappers. *Journal of Data & Knowledge Engineering 36(3):283-316*. 2001

[Selkow77] S. M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters 6(6):184-186*. 1977

[SiteMap] *https://www.google.com/webmasters/tools/docs/en/protocol.html*

[Soderland99] S. Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning 34(1-3):233-272*. 1999

[Tai79] K. Tai. The tree-to-tree correction problem. *Journal of the ACM 26(3): 422-433*. 1979

[Valiente02] G. Valiente. Tree edit distance and common subtrees. *Research Report LSI-02-20-R, Universitat Politécnica de Catalunya*. 2002

[W3CDOM] The W3 Consortium. The Document Object Model. *http://www.w3.org/DOM/*

[W3CHTM] The W3 Consortium. HTML 4.01 Specification. *http://www.w3.org/TR/html4/*

[W3CHTTP] The W3 Consortium. HyperText Transfer Protocol v1.1. *http://www.w3.org/Protocols/*

[W3CSW] The W3 Consortium. Semantic Web. *http://www.w3.org/2001/sw/*

[W3CWS] The W3 Consortium. Web Services. *http://www.w3.org/2002/ws/*

[W3CXPT] The W3 Consortium. XML Path Language (XPath). *http://www.w3.org/TR/xpath/*

[WBH05] I.M.Wallace, G. Blackshields, and D.G. Higgins. Multiple sequence alignments. *Current Opinion in Structural Biology 15(3):261-266*. 2005

[WC07] WebCopier. Feel the Internet in your Hands. *http://www.maximumsoft.com/*

[WDC06] W. Wu, A. Doan, and Clement Yu. WebIQ: Learning from the Web to Match Deep-Web Query Interfaces. *En Proceedings of the 22nd International Conference on Data Engineering (ICDE)*. 2006

[WL03] J. Wang, and F. Lochovsky. Data Extraction and Label Assignment for Web Databases. *En Proceedings of the 12th International World Wide Web Conference (WWW12)*. 2003

[WWLM04] J. Wang, J.-R. Wen, F. H. Lochovsky, and W.-Y. Ma. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. *En Proceedings of the 30th International Conference on Very Large Databases (VLDB), pp. 208-419*. 2004

[WWLM06] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query Selection Techniques for Efficient *Crawling* of Structured Web Sources. *En Proceedings of the 22nd International Conference on Data Engineering* (ICDE)*, p. 47*. 2006

[WWT91] K. Wittenburg, L. Weitzman, J. Talley. Unification-based grammars and tabular parsing for graphical languages. *Journal of Visual Languages and Computing, 4(2):347-370*. 1991

[Yodlee] Yodlee - *http://corporate.yodlee.com*

[ZHC04] Z. Zhang, B. He, and K. C.-C. Chang. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. *En Proceedings of the 2004 ACM SIGMOD Conference*. 2004

[ZHC05] Z. Zhang, B. He, and K. C.-C. Chang. Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. *En Proceedings of the 31st International Conference on Very Large Databases (VLDB)*. 2005

[ZL06] Y. Zhai, and B. Liu. Structured Data Extraction from the Web Based on Partial Tree Alignment. *En IEEE Transactions on Knowledge and Data Engineering Journal, vol. 18, n°12*. 2006