Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Montevideo - Uruguay

Proyecto de Grado

"Extracción e integración de información en una arquitectura de Web Warehouse"

Diciembre 2004

Verónica Giaudrone
Marcelo Guerra
Marcelo Vaccaro

Tutoras:
Adriana Marotta
Regina Motz

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Resumen

Actualmente se encuentra publicada en la Web gran cantidad de información diversa que podría ser utilizada para el apoyo a la toma de decisiones. Este proyecto está motivado por la necesidad de extraer e integrar este tipo de información. Dado lo costoso y complejo de un proceso de extracción e integración manual, se plantea la construcción de una herramienta, en el marco de una arquitectura Web Warehousing, basada en Wrappers y Mediadores. Los Wrappers se encargan de la extracción de información, que debe permitir seleccionar páginas Web relevantes según una consulta en un dominio de interés, y obtener de ellas la información solicitada. Por otro lado, los Mediadores realizan la integración de datos, que busca complementar la información proveniente de varias fuentes, y resolver conflictos que puedan surgir por información contradictoria. Además, dados los cambios constantes de la información contenida en la Web surge la necesidad de contar con un mecanismo que permita soportar la evolución del sistema, manteniendo trazabilidad del proceso para poder reflejar un cambio ocurrido en cierta fuente en la información obtenida.

Nuestro proyecto comprende la implementación de una herramienta para extraer e integrar la información de las páginas Web y generar metadata que permita soportar la evolución. El proceso de extracción se basa en la estructura de las páginas Web, considerando las diferentes posibilidades de presentación y cubriendo algunas de ellas. Además, buscamos enriquecer la consulta, utilizando una ontología del dominio de búsqueda. La integración se realiza a partir de la información obtenida en la extracción. El método de integración propuesto se basa en la confiabilidad de las fuentes. Además, se analiza el efecto de cambios en las páginas Web (evolución) en el sistema, de forma de minimizar el impacto de éstos en los procesos de extracción e integración de la información. En cada etapa se genera la metadata necesaria, independientemente de la información extraída para ganar flexibilidad, para una futura implementación del manejo de evolución.

Se implementó un prototipo que cumple las funcionalidades descriptas, utilizando tecnologías como XPath, XML, XML Schema, OWL y Java. Se presenta un caso de prueba de aceptación y se realiza un testing que busca satisfacerlo, así como casos de otros posibles dominios. Se utilizan métricas para facilitar la detección de mejoras o desmejoras.

Palabras clave: Web Warehouse, Wrapper, Mediador, Ontología, Extracción de información, Integración de información.

Índice

Índice de figuras

Capítulo 1. Introducción

El presente documento es resultado del proyecto “Extracción e integración de información en una arquitectura de Web Warehouse” correspondiente a la asignatura Proyecto de Grado del Instituto de Computación de la Facultad de Ingeniería, dentro del área de Concepción de Sistemas de Información.

El proyecto se basa en el artículo “Managing Source Schema Evolution in Web Warehouses” [1] que propone una arquitectura de Web Warehouse. Este tipo de sistemas tiene como objetivo permitir la carga de datos provenientes de la Web, en un Data Warehouse.

Se requirió el estudio de los temas relacionados a la arquitectura propuesta, así como la investigación de tecnologías involucradas. Siguiendo la arquitectura planteada, se llevó a cabo el relevamiento de requerimientos, el diseño, la implementación de la herramienta y el testeo, en el marco de un proceso iterativo e incremental.

Si bien existen varios proyectos realizados en torno a la extracción de información de la Web (algunos ejemplos son [4][5][6]), en este caso se propone no sólo la extracción sino también la integración de esta información, así como la generación de metadata. El principal objetivo de esta metadata es ser un soporte para el manejo de la evolución de las páginas Web y por lo tanto de la información obtenida como resultado. La información en la Web cambia constantemente y es importante mantener la información obtenida actualizada, ya que apoyará a la toma de decisiones.

Este documento describe los aspectos más importantes del proyecto, introduciendo los temas involucrados, desarrollando la solución propuesta, comentando las tecnologías utilizadas, así como las dificultades encontradas, las limitaciones y las posibles futuras extensiones.

1.1. Motivación y objetivos

La Web es actualmente una herramienta de uso global y habitual para un grupo importante de personas, tanto para trabajar, como para compartir información, experiencias y comunicar ideas, lo que constituye una gran fuente de datos, distribuida alrededor del mundo. Además, es una fuente de información que crece continuamente de forma exponencial, tanto en términos de cantidad como de diversidad de la información [2][3][27].
Por otro lado, cada usuario publica la información de la manera que prefiere, no existiendo un estándar sobre la forma de compartir la información.
Se presenta entonces, una inminente necesidad de comprender la dinámica de la Web y de mecanismos para obtener la información allí contenida.

En la actualidad, dada la necesidad de buscar información en la Web, debemos recurrir inevitablemente a un buscador, como Google. En el buscador se ingresan ciertas palabras claves, y como resultado se obtiene un conjunto de páginas Web, que esperamos contengan la información que necesitamos.

En general, el resultado que obtenemos no cumple con nuestras expectativas, ya que está constituido por varias páginas que contienen las palabras solicitadas y debemos analizarlas para obtener la información que nos resulte útil. El resultado obtenido se caracteriza por contener una gran cantidad de páginas Web, donde no todas están relacionadas al tema de la consulta, por otro lado, no todas las páginas contienen toda la información que solicitamos e incluso puede haber inconsistencias que deberíamos evaluar entre las diferentes fuentes de datos. Es cada vez más necesario obtener la información deseada de forma lo más exacta y concreta posible, de manera que pueda ser utilizada como apoyo a la toma de decisiones.

Como solución a este problema se plantea una arquitectura de extracción e integración de información basada en técnicas de “wrappers y mediadores” [1], que permite realizar la carga de un Data Warehouse con información obtenida de la Web. Luego, el usuario realiza sus consultas directamente sobre el Data Warehouse.

Dado el cambio constante de la información existente en la Web, se presenta como una importante necesidad manejar la evolución de la información en el sistema, o sea reflejar en la información recabada, los cambios que suceden en las fuentes de datos. Con este objetivo, es que se genera metadata que permite mantener la trazabilidad del proceso.

Objetivo general:

- Análisis, diseño e implementación de una herramienta de extracción e integración de información proveniente de la Web siguiendo la arquitectura propuesta en [1].

Objetivos particulares:

- Proponer soluciones para los problemas de extracción de información de las páginas Web, utilizando ontologías de dominio.
- Proponer soluciones para los problemas de integración de datos, utilizando ontologías de dominio.
- Analizar el manejo de evolución de las fuentes.

- Proponer metadata adecuada para el soporte del sistema y la evolución de las fuentes.
- Implementar las soluciones propuestas

1.2. Alcance del proyecto

Dado que la propuesta de este proyecto no comprende toda la arquitectura de wrappers y mediadores especificada en el artículo de base [1], sino que se centra en la extracción de la información de la Web y su integración a nivel de datos, presentaremos el diagrama de la arquitectura, donde indicamos su alcance.
En las siguientes secciones describiremos en detalle esta arquitectura, pero en la Figura 1 podemos observar que como entrada se presentan varios conjuntos de páginas Web (estos corresponden a dominios de información relacionados), y finalmente la arquitectura concluye en un Data Warehouse.
El alcance de este proyecto, incluye la extracción de información de las páginas Web y la integración a nivel de datos (lo indicamos con el recuadro anaranjado punteado, más grande, en la Figura 1), y deja afuera la integración de información a nivel de diferentes dominios relacionados y la carga del Data Warehouse. Además, podemos observar que la arquitectura para cada conjunto de páginas es la misma, por lo cual nuestro sistema la implementa una sola vez, pudiendo ser instanciada para los diferentes conjuntos de páginas (lo indicamos con el recuadro celeste punteado, más pequeño, en la Figura 1).

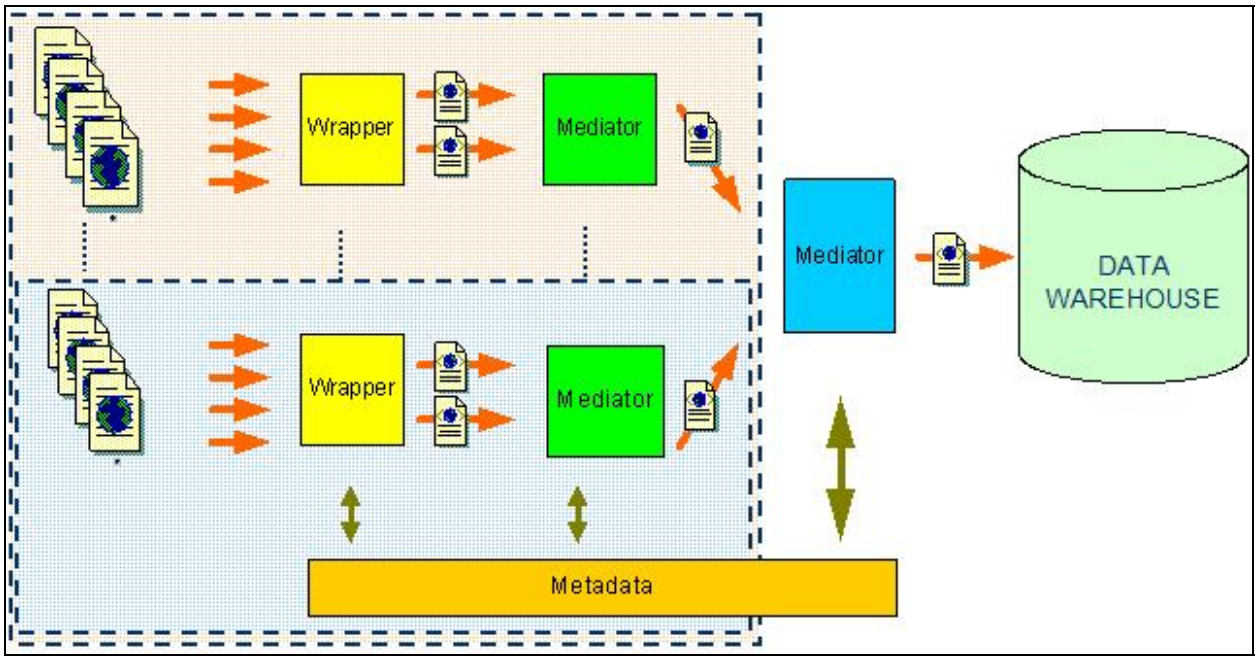


Figura 1 – Arquitectura y alcance del proyecto

Por otro lado, el alcance incluye el análisis de la metadata necesaria para soportar la evolución y su generación, pero no comprende la implementación de los mecanismos de evolución.

1.3. Metodología de trabajo

Se eligió como metodología de trabajo un proceso iterativo e incremental que consta de dos iteraciones principales. En la primer iteración se construyó una "maqueta" del prototipo para alcanzar ciertos objetivos, que incluyó un testeo

básico, luego de la segunda iteración se obtuvo la versión final del prototipo que cumple con el nivel de calidad requerido.

Esta metodología de trabajo se diseñó en torno a los requerimientos de calidad. Dado que el proyecto se enmarca en un proyecto mayor que se continuará en el futuro, es de suma importancia generar un sistema que alcance cierto nivel de calidad.

Inicialmente se realizó un estudio de los temas relacionados al proyecto, a partir de la guía brindada por las tutoras, se relevaron los requerimientos, y se validó la comprensión del problema. Se mantuvieron reuniones semanales con las tutoras en las primeras etapas, dada la necesidad de comprender con claridad el problema a resolver, en las siguientes etapas se realizaron reuniones quincenales. El equipo de trabajo mantuvo reuniones semanales, para la discusión de temas importantes y la toma de decisiones, así como la coordinación de las actividades y la evaluación de los resultados de cada semana.

1.4. Organización del documento

Este documento está estructurado en 7 capítulos, en los cuales se pretende ofrecer al lector los aspectos más sobresalientes del proyecto. Se puede profundizar en los diferentes temas, en los documentos que se entregan de manera complementaria.
Los capítulos se organizan como se describe a continuación:

- Capítulo 2: Se incluyen conceptos que deben tenerse presentes para comprender el resto del documento y la solución desarrollada.
- Capítulo 3: Se presenta el desarrollo de la solución, describiendo la arquitectura en la que se basa la herramienta, y cada módulo identificado. Este capítulo es esencial para comprender los siguientes, ya que profundizan en cada módulo de la arquitectura.
- Capítulo 4: Este capítulo se centra en la extracción de la información, desarrollando la arquitectura del wrapper, las decisiones de diseño tomadas y la metadata generada.
- Capítulo 5: Desarrollamos la integración a nivel de datos, correspondiente al primer mediador de la arquitectura. Profundizamos en la arquitectura del mediador, el criterio de integración utilizado y la metadata generada.
- Capítulo 6: Se presenta la metodología de testing utilizada, así como las métricas y los resultados obtenidos.
- Capítulo 7: Finalmente, se presentan las conclusiones obtenidas, producto del trabajo realizado y el trabajo futuro.

Al final del documento se encuentran las referencias a la bibliografía que utilizamos durante todo el proyecto.

Capítulo 2. Conceptos generales

La toma de decisiones en general necesita información de múltiples fuentes, pero es difícil para las personas involucradas obtener la información requerida e integrarla en una cantidad de tiempo razonable, dado el acceso que deben realizar a diferentes fuentes, y las inconsistencias o contradicciones que pueden encontrar. Con el objetivo de manejar esta gran cantidad de información es que los investigadores proponen herramientas para organizarla [1][34][35].

En este capítulo proponemos revisar algunos de los principales conceptos y herramientas en los que se basa este proyecto.

2.1. Arquitectura de Web Warehouse

En el trabajo [1] se presenta una arquitectura para Web Warehouse. Esta arquitectura se basa en el uso de dos tipos de módulos: Wrappers y Mediadores [34][35]. El objetivo de un wrapper es extraer información relevante a partir de una fuente de datos y presentar esta información en un formato especificado. El rol de un mediador es de integrar la información producida por diferentes wrappers u otros mediadores. En la Figura 2 se presenta el diagrama de la arquitectura.

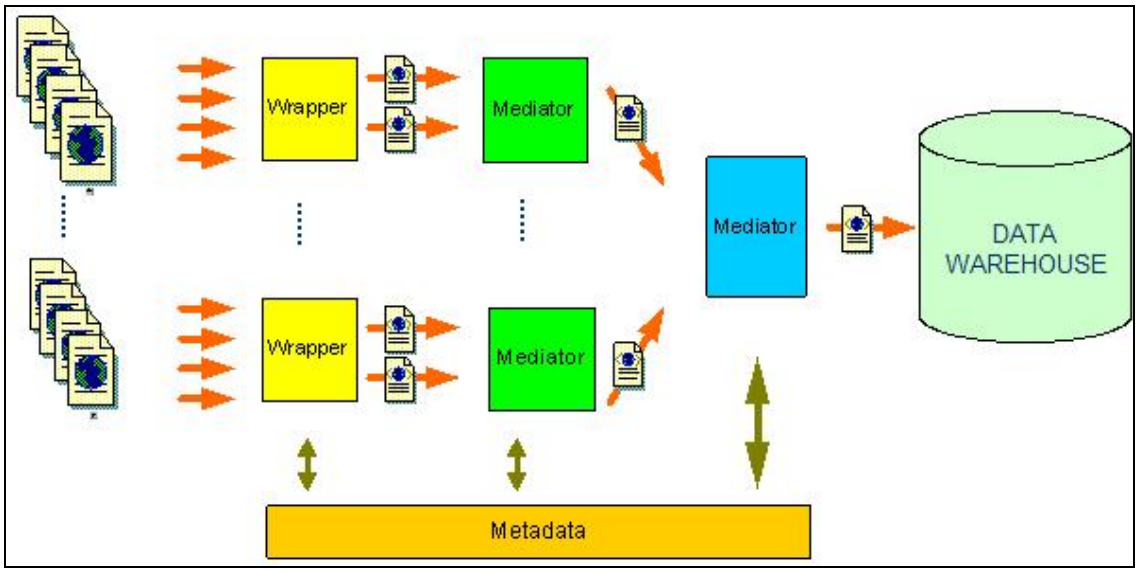


Figura 2 – Arquitectura de Web Warehouse

Los wrappers extraen información de conjuntos de páginas Web, donde cada uno se ocupa de un dominio particular. Un wrapper devuelve la información relevante contenida en cada página del conjunto. Se decidió la utilización XML [24] para la salida y entrada de información, por lo cual la información relevante viene dada por un pedido que realiza el Data Warehouse en formato de esquema XML. El resultado que se obtiene son varios archivos XML (uno para cada página Web) que cumplen con cierto esquema especificado.
Luego un mediador integra la información a nivel de datos, tomando como entrada los archivos XML generados por el wrapper, integrando la información obtenida de diversas fuentes para cierto dominio. La integración comprende la resolución de conflictos de datos que pueden provenir por ejemplo de inconsistencias de las diversas fuentes, por lo cual deben definirse criterios de integración de la información [29][30][31].

Finalmente, se considera un segundo mediador que realiza una integración a nivel de esquemas, que integra la información de diversos dominios (relacionados). Este tipo de integración comprende la reestructuración de la información, ya que toma como entrada archivos con diferentes estructuras, pero de contenido semántico relacionado. La información obtenida permite cargar un Data Warehouse al cual se le pueden realizar consultas sobre ese dominio.
Por otro lado, dados los continuos cambios que ocurren en la Web se considera la generación de metadata, que registra todo el proceso realizado para la extracción e integración, y que permitirá reflejar de forma automática los cambios en la información obtenida.

2.2. Ontologías

La nueva generación de la Web, está caracterizada como la "Web Semántica", ya que la información no sólo debe ser interpretada por humanos, sino que también debe poder ser procesada por máquinas. La Web Semántica requiere de interoperabilidad a nivel semántico, lo que requiere estándares no sólo sintácticos, sino también del contenido semántico [13][28]. La W3C (World Wide Web Consortium) está realizando grandes esfuerzos de estandarización por medio de XML/XML Schema, RDF/RDF Schema, OWL, etc. [28].

De manera sencilla, una ontología es una especificación formal de un vocabulario que describe objetos, conceptos u otras entidades relevantes de una realidad y las relaciones entre estos. [11]
Según [10], una ontología es una especificación explícita de cierto tema, es una representación formal y declarativa que incluye vocabulario (o nombres) para referirse a términos en el área de interés y sentencias lógicas que describen qué son los términos, cómo están relacionados, y cómo pueden o no estar relacionados. Las ontologías proveen un vocabulario para representar y comunicar conocimiento sobre cierto tema y el conjunto de relaciones existentes entre los términos de ese vocabulario.
Según Tom Gruber [12], una ontología es una especificación de una conceptualización, lo que significa que es una descripción de conceptos y relaciones que pueden existir para un agente o comunidad de agentes.

Las ontologías pueden verse como esquemas de metadatos, que proveen un vocabulario controlado de términos, cada uno de ellos definido explícitamente con una semántica procesable por una máquina.[4]

Existen varios lenguajes para especificar ontologías como rdf [16], daml+oil [27][9] y owl [15][19][32][33]. En particular, nosotros utilizamos OWL, que es un lenguaje diseñado para el uso de aplicaciones que necesitan procesar el contenido de la información en lugar de la presentación a través de la cual sería publicada a un ser humano. OWL facilita la interpretación por parte de las máquinas proveyendo de vocabulario con una semántica formal.

Las ontologías deben ser desarrolladas y mantenidas para cada dominio de interés, lo que puede convertirse en una tarea tediosa de realizar manualmente. Otro problema que presentan, es que los expertos en la áreas (que podrían escribir las ontologías de forma más exacta), no conocen (o tienen por qué conocer) la forma de escribir las especificaciones.

2.3. Estructuración de las páginas Web

En la Web se utiliza html como el lenguaje para publicar la información, este lenguaje tiene como característica que la información se encuentra estructurada, pero de forma flexible. En este sentido es que, muchas de las páginas publicadas no necesariamente están estructuralmente bien escritas, pero de todas formas se visualizan correctamente. Por otro lado no existe un estándar en la estructuración que debe realizarse en las páginas y esta flexibilidad lo hace atractivo para muchos contextos. En particular, esta característica hace difícil el trabajo de procesamiento de la información y la obtención de conocimiento para las máquinas, dada la variedad de formas en las que puede presentarse.

La estructuración concierne a la extracción de la información, ya que debe decidirse de que forma reconocer la información útil. Dado que la estructuración existe pero puede presentarse de una gran variedad de formas, es que en este proyecto elegimos algunos formatos de estructuración que permitieron generar reglas de extracción de información. Pudimos observar que en general las páginas Web utilizan el formato de tablas que permite ordenar la información y visualizarla de mejor manera.

2.4. Metadata

La metadata se define como datos acerca los datos, o una forma de expresar información acerca de la información.
En nuestro caso podemos distinguir metadata que contiene por ejemplo, información que permite seguir la trazabilidad del proceso que realiza la herramienta, información acerca del formato de los archivos XML que se generan, información que utiliza la herramienta internamente y puede enriquecerse para mejorar su comportamiento.

Las ontologías también son un tipo de metadata, la cual está orientada a describir información sobre un dominio determinado.

La metadata resulta un componente importante en este proyecto, ya que se toma como base para permitir la evolución del sistema, en cuanto a la actualización de la información obtenida.

2.5. Evolución

Al referirnos a evolución, como ya hemos mencionado, nos estamos refiriendo a poder reflejar en la información obtenida los cambios que ocurren en las fuentes de datos.
La Web es una fuente de continuos cambios, en el sentido más amplio, cierta página Web puede estar publicada hoy y no mañana; la información cambia en su contenido, tanto por actualizaciones de datos existentes, como por cambio de contexto (o sea se utiliza la misma URL para publicar distinto tipo de información); cambia la forma de presentar información en la página y por lo tanto su estructura.
Para poder realizar una propagación de cambios automática resulta necesario registrar información sobre el proceso realizado, de forma de detectar qué porción de la información debe ser cambiada o actualizada, sin necesidad de ejecutar la herramienta nuevamente [1][36].

2.6. Lenguaje XPath

El propósito de XPath (XML Path Language) es referenciar partes de un documento XML. Para satisfacer este objetivo, provee facilidades de manipulación de strings, números y expresiones booleanas. Tiene una sintaxis compacta, que opera en la estructura lógica de un documento XML. Utiliza una notación de caminos (paths), como en las URLs, para navegar en la estructura jerárquica de un documento XML.
XPath modela un documento XML como un árbol de nodos. Hay diferentes tipos de nodos, incluyendo nodos de elemento, nodos de atributo, nodos de texto.
El lenguaje también está diseñado para utilizar el matcheo, o sea testear si un nodo satisface cierto patrón.

La principal construcción sintáctica de XPath es la expresión. Los caminos de ubicación (location paths) son una de las expresiones más importantes. Un camino de ubicación selecciona un conjunto de nodos relativo a cierto nodo, llamado de contexto. El resultado de evaluar una expresión de este tipo es un conjunto con los nodos seleccionados por el camino.
Además, proporciona una biblioteca de funciones que pueden utilizarse para evaluar expresiones. Cada función toma uno o más argumentos y devuelve un resultado, donde los argumentos y las funciones son de tipos básicos (conjunto de nodos, booleano, número o string).

La especificación de este lenguaje puede encontrarse en [22] y un manual sencillo se encuentra en [23].

Capítulo 3. Análisis del sistema

Este capítulo pretende brindar una visión global de la solución desarrollada, se especificarán los requerimientos relevados y se plantearán las subdivisiones que se realizaron del problema.
Se utilizará como guía el caso de prueba en el que se basó el desarrollo.

3.1. Requerimientos

Se resumen los requerimientos relevados en términos del tipo de usuario del sistema, requerimientos funcionales, no funcionales y restricciones. El documento completo se encuentra en el anexo “Documento de requerimientos”.

3.1.1. Usuarios

El sistema a construir forma parte de otro sistema que se implementará en un futuro y completará la propuesta [1]. Por lo tanto nuestro sistema será utilizado por el sistema global que se encargará de solicitarle la información necesaria, para realizar el proceso de integración de esquemas y carga del Data Warehouse. La comunicación se realizará a través archivos XML.

El usuario del sistema global podrá ser un usuario que realice consultas al Data Warehouse, por lo cual será un usuario no experto.
La herramienta construida en este proyecto podrá ser utilizada por dos tipos de usuario, uno que tome al sistema con un domino determinado, con las configuraciones ya realizadas, utilizando páginas Web de similares características; y otro que cumple el rol de instalador de la herramienta, que debe conocer las tecnologías XML, XPath y ontologías, que brinde al sistema la ontología del dominio que desee utilizar, el esquema XML con la solicitud de la información, y extienda la herramienta (agregando reglas de extracción que se explicarán más adelante), en caso de ser necesario.
Para comprender en detalle el funcionamiento de la herramienta se proporciona el “Manual de usuario”, como anexo.

3.1.2. Requerimientos funcionales

Clasificación de páginas web:
Deben seleccionarse las páginas web del dominio que son relevantes al pedido del Data Warehouse. Éstas serán las páginas que contengan la información a devolver. Debe considerarse la reutilización de algunos trabajos ya realizados en este tema.

Extracción de información de una página:
La extracción consiste en seleccionar información relevante de cada página Web que se clasificó y devolver un archivo XML, basado en un esquema dado (que se obtendría a partir de las necesidades del Data Warehouse), que permita su procesamiento posterior.
La búsqueda de la información se orienta con el esquema de salida y conocimiento específico del dominio (ontología). Para esto se considerará la utilización de heurísticas que determinarán donde se encuentra la información a extraer.

Integración de los datos obtenidos de las páginas:
La extracción de información devolverá un archivo de datos para cada página relevante. O sea, que se obtienen tantos archivos con información como páginas relevantes existan, pero validan el mismo XML-Schema (con formato a definir). La

integración se encargará de devolver un solo archivo para el conjunto de páginas relevantes con la información solicitada. Mantendrá el formato de los archivos de entrada pero realizará una "identificación de mismos datos provenientes de distintas fuentes", en caso de conflictos de datos deberá seguir estrategias para determinar que datos se mantienen.

Generación de metadata para soporte a la evolución:
Es necesaria la generación de metadata (datos sobre los datos) que indique como se realizan los procesos, para poder soportar la evolución. Esto refiere a que ante futuros cambios en las páginas la metadata permita identificar donde afecta en la información obtenida (propagación de cambios fuentes- mediador).

3.1.3. Requerimientos no funcionales

Elección de un dominio a utilizar:
Se plantea que el desarrollo se realice basado en cierto dominio a elegir. Esto es importante para orientar los casos de prueba y la búsqueda de información.

Ontologías para obtener conocimiento del dominio:
Es necesario enriquecer el vocabulario correspondiente al pedido del DataWarehouse, con conocimiento del dominio. El conocimiento debe ser obtenido mediante ontologías sobre ese dominio. Esto es importante porque en la Web se utiliza mayoritariamente lenguaje natural para especificar información lo cual implica que hay muchas formas de decir lo mismo.

Utilización de XML para entrada/salida de datos:
La modularidad del sistema implica comunicación entre diferentes componentes. Esta comunicación se realizará mediante archivos con un formato específico. Se decidió la utilización de un formato estándar como lo es XML.

Documentación del sistema:
Debe documentarse el diseño y la implementación del sistema, así como realizar un manual de usuario. Se espera que la documentación contenga la información necesaria para la comprensión, reutilización y extensión de la solución propuesta.

3.1.4. Restricciones

Nivel de calidad
Se construirá una "maqueta" que permitirá evaluar la arquitectura de solución propuesta, su alcance será limitado a cumplir ciertos casos de prueba específicos. Para esta etapa se deberá alcanzar un nivel de calidad 2, según la especificación en www.fing.edu.uy/~dvallesp/Tesis/webActividades/index.htm. [7]
Luego para el prototipo final se requerirá saltar al nivel 3 de calidad siguiendo la especificación antes mencionada.

3.1.5. Interfaces de Usuario

Se utilizará la consola del sistema.

3.2. División en módulos

Hemos descrito la arquitectura y los requerimientos que se relevaron, lo que llevó al planteo de la siguiente división.

Se dividió en tres módulos principales:
- Clasificador
- Extractor
- Mediador

El clasificador y el extractor conformarían el wrapper, pero se decidió separarlos dada la posibilidad de reutilizar un clasificador ya implementado en un proyecto anterior [5].
Por lo tanto, se decidió comenzar con el proceso de extracción, luego continuar con el mediador y finalmente realizar la clasificación.

Se realiza un análisis del problema de la evolución en las páginas Web y su propagación al sistema, el cual permitirá identificar la metadata que debe ser registrada, para una posterior implementación de los mecanismos evolutivos. Las salidas del extractor y el mediador consisten en la información obtenida de las fuentes y archivos con la metadata necesaria, que se detallarán en las siguientes secciones.
En el anexo “Documento de diseño” se especifica el diseño de los módulos analizados.

3.3. Elección de dominio

Es necesario especificar el dominio a utilizar de forma de poder contar con las entradas adecuadas al sistema, entradas que dependen del dominio. Necesitamos páginas Web seleccionadas de ese dominio, la ontología y esquemas XML.
El sistema no debería perder generalidad por basarse en un dominio específico, pero se necesita uno para la realización del prototipo.
Para elegir el dominio adecuado debemos tener en cuenta varios factores que permitan demostrar las características más relevantes del sistema. Los factores principales son la integración de datos provenientes de distintas fuentes, luego de la extracción de la información y el soporte a cambios (evolución). Se decidió que el dominio a elegir no tenía que ser ideal para ninguno de los factores en particular, pero si debía cubrir lo mejor posible la arquitectura del sistema. La explicación detallada se encuentra en el anexo “Elección del dominio”.

Se buscó dentro del dominio de medicina en idioma inglés, donde se consideraron los médicos, hospitales y medicamentos, enfocándonos principalmente en los médicos. Este dominio permitió iniciar el análisis del problema, aunque en esa etapa aún no se contaba con el dominio definitivo. Se encontró poca información que pudiera crear conflictos en la integración.
Se sugirió buscar dentro del dominio de turismo y cine. El dominio turismo se propuso principalmente por su evolución en el tiempo, el problema que se presentó era que muchas páginas contenían imágenes para presentar la información y que los paquetes de viaje no tienen una fácil identificación, como para poder determinar qué paquetes son iguales pero ofrecidos por diferentes agencias.
Sobre el dominio de cine se buscó información en idioma español, se consideraron como dominios relacionados: las películas de cine y los actores. Se detectó que las carteleras de cine cambian con frecuencia, y se encontró información contradictoria

como el género y la duración. Lamentablemente, no encontramos una ontología que aplicara a nuestras necesidades dentro de este dominio [9][14], por lo cual debimos construir una con la información necesaria. Esta ontología se puede consultar en el anexo “Ontologías”. Para su generación se utilizó la herramienta [20].
Luego de este análisis, se decidió plantear el caso de prueba con el dominio de películas de cine.

3.4. Caso de prueba

Se presentan las principales características del caso de prueba utilizado para ejemplificar las funcionalidades con este caso. El caso de prueba de aceptación se encuentra especificado en el anexo “Caso de prueba de aceptación”.

En la Figura 3 podemos observar el diagrama de la arquitectura instanciado para el dominio de cine. Se muestran dos conjuntos de páginas relacionados: películas y actores; que corresponden a dos líneas de la arquitectura. El caso de prueba se basa en la segunda línea correspondiente a las películas de cine.

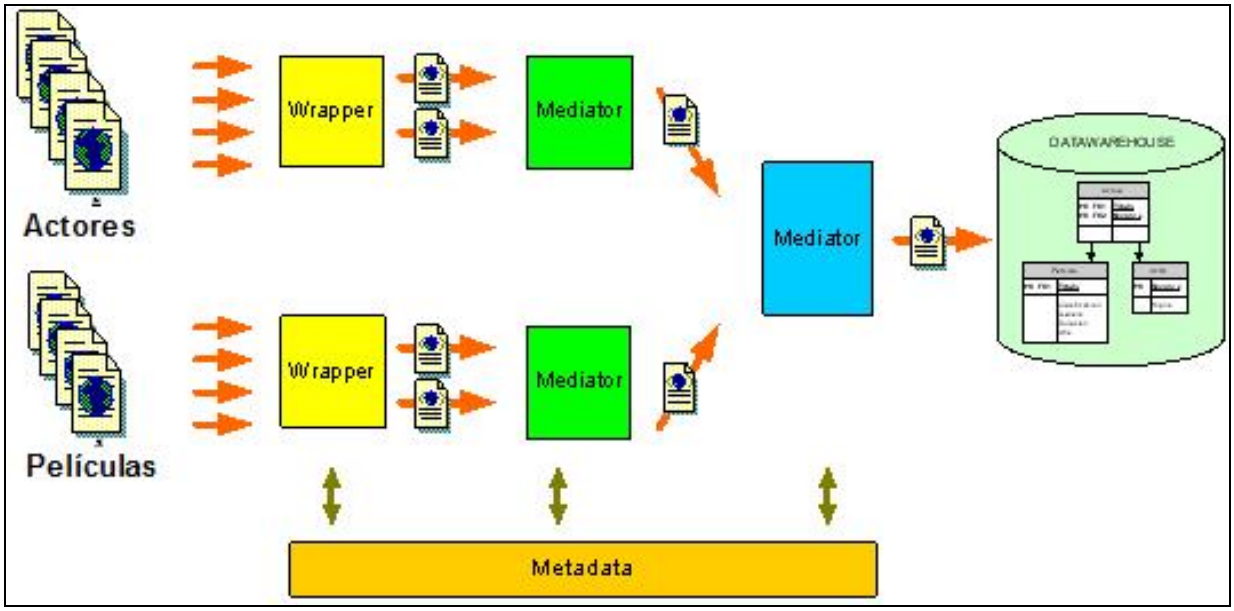


Figura 3 – Arquitectura instanciada

Debe brindase al sistema un conjunto de páginas Web, una ontología del dominio de interés, el esquema XML de solicitud de información y la confiabilidad de la fuentes, que se utilizará en caso de conflictos de datos en la etapa de integración.

Se utilizaron páginas seleccionadas de una búsqueda realizada en Google sobre las películas de cine, incluyendo carteleras de cine del Uruguay. Se trabajó con 11 páginas Web, de diferentes fuentes y características.

Las páginas Web serán presentadas en caso de ser necesario en las siguientes secciones, y pueden consultarse en el resto de la documentación entregada. En la Figura 4 se muestran fragmentos de algunas páginas a modo de ejemplo.

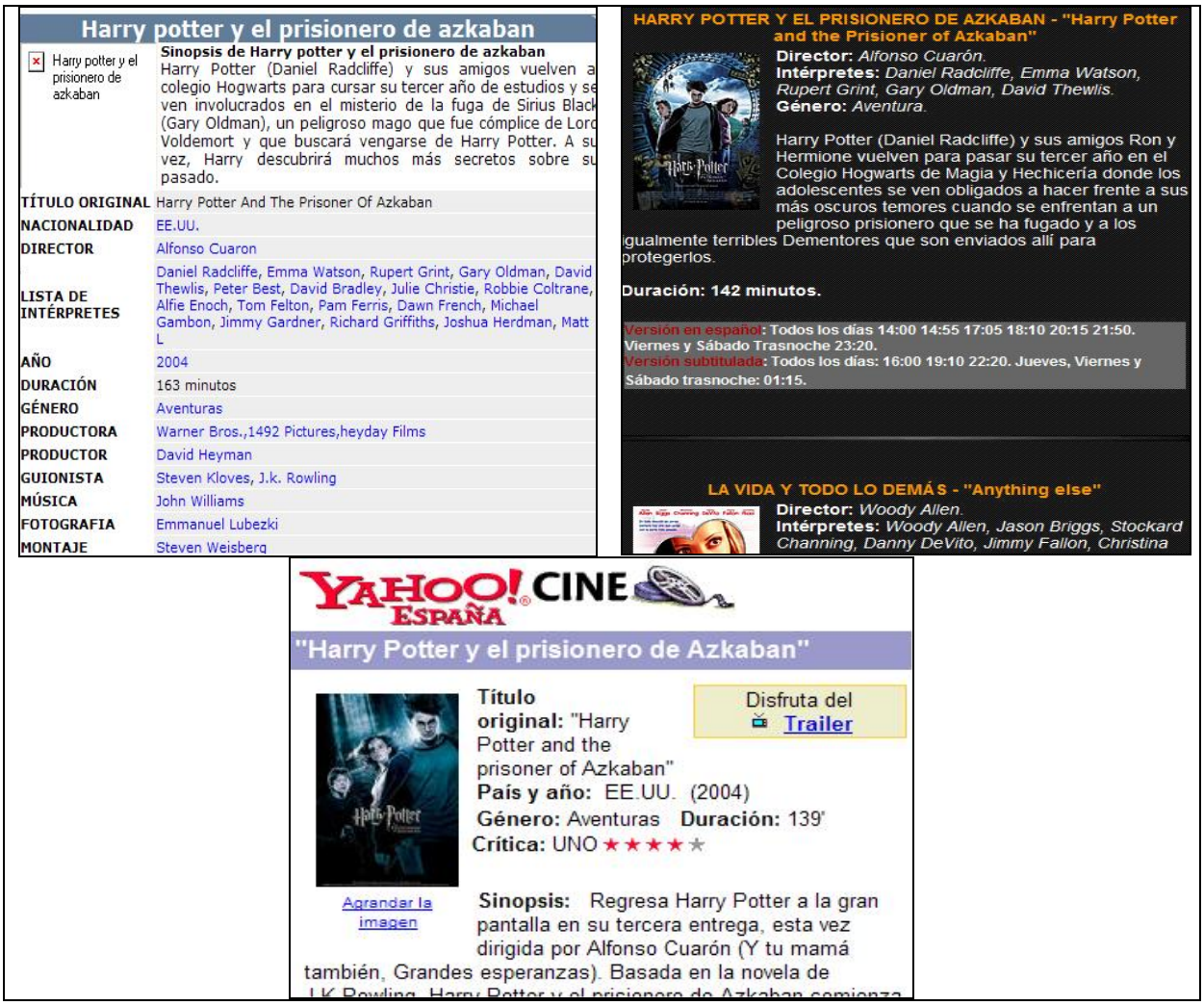


Figura 4 – Páginas Web

Podemos observar diferentes páginas de las cuales, una pertenece a una cartelera por lo cual contiene varias películas y las otras dos son específicas de cierta película (utilizamos como películas principales “Harry Potter y el prisionero de Azkabán” y “Troya”, por lo cual los resultados para esas películas serán los que obtengan más información, ya que del resto no incluimos más páginas, sólo se cuenta con la información de las carteleras).

La ontología contiene la información necesaria, así como individuos que instancian los conceptos (se consultó [21]). Dado lo extensa de la ontología presentamos sólo un extracto de ella en la Figura 5. El lenguaje utilizado es OWL. La ontología completa se puede consultar en el anexo “Ontologías”.

```
- <owl:Class rdf:ID="sinopsis">
  - <owl:equivalentClass>
      <owl:Class rdf:ID="resumen" />
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="intérpretes" />
  <owl:Class rdf:ID="productor" />
  <owl:Class rdf:ID="idioma" />
  <owl:Class rdf:ID="dirección" />
  <owl:Class rdf:ID="director" />
  <owl:Class rdf:ID="apta" />
  <owl:Class rdf:ID="interpretación" />
- <owl:Class rdf:ID="país">
  - <owl:equivalentClass>
      <owl:Class rdf:ID="nacionalidad" />
    </owl:equivalentClass>
  </owl:Class>
  <país rdf:ID="usa">
  - <owl:sameAs>
    - <país rdf:ID="estados_unidos">
      - <owl:sameAs>
        - <país rdf:ID="u.s.a.">
            <owl:sameAs rdf:resource="#estados_unidos" />
          </país>
        </owl:sameAs>
      - <owl:sameAs>
        - <país rdf:ID="ee.uu">
            <owl:sameAs rdf:resource="#estados_unidos" />
          </país>
        </owl:sameAs>
        <owl:sameAs rdf:resource="#usa" />
      </país>
    </owl:sameAs>
  </país>
```

Figura 5 – Ontología de películas de cine

Los conceptos se encuentran declarados como clases dentro del namespace [17] películas, y se definen conceptos que tienen instancias equivalentes y disjuntas, así como para las instancias se define cuales son iguales y cuales diferentes. Se define una clase como disjunta de otra explícitamente, en caso que pueda dar lugar a confusión en la extracción, como el título y el título original, donde una palabra está incluida en la otra. Un razonamiento similar se aplica a las instancias, donde por ejemplo el género comedia es diferente a comedia romántica. No se toman en cuenta las jerarquías sino que se utiliza la etiqueta de equivalencia para que los conceptos sean tomados como sinónimos. Podría extenderse la utilización de la ontología para utilizar jerarquías, pero nuestra herramienta sólo considera sinónimos e instancias.
Por otro lado, en este caso no se muestran etiquetas para los conceptos ya que el nombre puede escribirse directamente en el identificador (los infra-guiones son cambiados a espacios por la herramienta), pero no todos los caracteres son permitidos en el identificador, por lo cual para ese fin se utiliza el atributo “label”, que permite especificar el nombre del concepto.

Se definen en la ontología los títulos de las películas, como instancias de título ya que no se encontró un formato estándar en el que aparecieran presentados en las páginas Web, con lo cual la forma de identificarlos es obtenerlos de la ontología. Esto puede verse como una restricción dado que debería encontrarse una forma de completar periódicamente la ontología con títulos de películas. Podría definirse una página Web que se utilice como referencia para obtener películas nuevas, o una base de datos de películas con la cual actualizar la ontología regularmente.

El esquema que se utiliza como solicitud de información se presenta en la Figura 6.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
   - <xs:element name="pelicula">
      - <xs:complexType>
            <xs:attribute name="título" type="xs:string" use="required" />
            <xs:attribute name="título_original" type="xs:string" />
            <xs:attribute name="país" type="xs:string" />
            <xs:attribute name="año" type="xs:gYear" />
            <xs:attribute name="duración" type="xs:duration" />
            <xs:attribute name="género" type="xs:string" />
            <xs:attribute name="productora" type="xs:string" />
            <xs:attribute name="audiencia" type="xs:string" />
            <xs:attribute name="idioma" type="xs:string" />
            <xs:attribute name="recaudación_total" type="xs:long" />
            <xs:attribute name="actores" type="xs:string" />
            <xs:attribute name="directores" type="xs:string" />
            <xs:attribute name="productores" type="xs:string" />
        </xs:complexType>
     </xs:element>
  </xs:schema>
```

Figura 6 – Esquema de solicitud de información

La solicitud permite incluir un solo elemento de búsqueda (en este caso película) que contenga la cantidad de atributos deseada. Se identifica un atributo como requerido, en este caso el título que se utilizará como clave en la integración. Además se especifica para cada atributo el tipo de dato, para posibilitar el chequeo de dominio.

La confiabilidad de las fuentes se presentará en la sección de integración donde se detallará su utilidad. Consiste de un archivo XML que define para cada fuente de información un número entre 0 y 1 que indica la confiabilidad de esa fuente.

Capítulo 4. Extracción de información

Parte importante del sistema desarrollado consiste en obtener información de páginas Web a partir de un pedido. No existe entre los diseñadores de páginas Web un criterio definido de cómo colocar la información y las formas de hacerlo son muy variadas.
Este capítulo muestra como el sistema ataca este problema e intenta obtener información con una precisión aceptable

4.1. Arquitectura de la solución

El Wrapper consiste en 5 módulos principales que se muestran en la Figura 7.

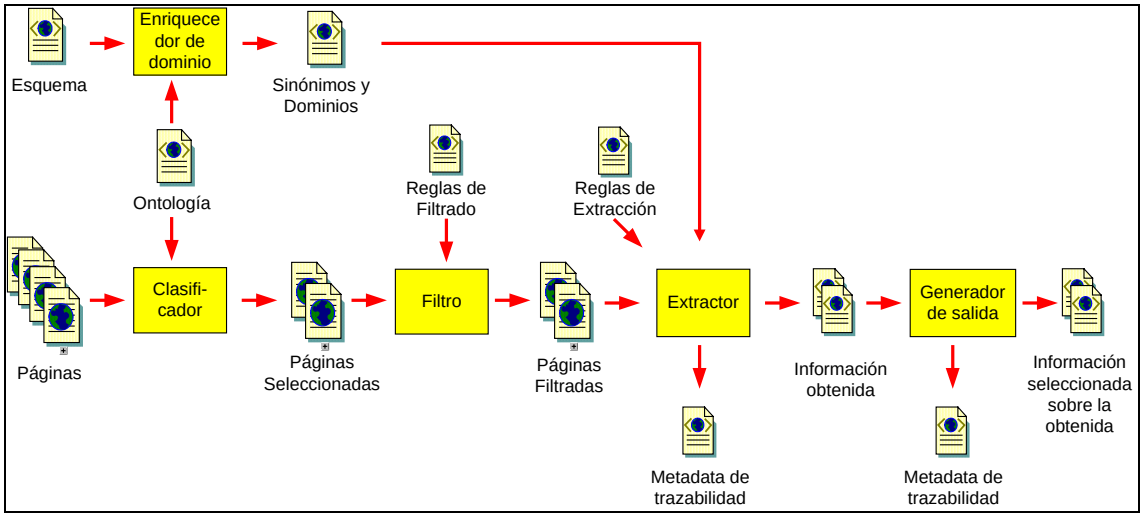


Figura 7 – Arquitectura del módulo Extractor.

En las siguientes secciones describiremos cada uno de los módulos.

4.1.1. Enriquecedor del dominio

El pedido de información representado por el esquema XML que ingresa al Wrapper solicita la información utilizando términos en particular que llamamos representante, el mismo es elegido por quien genera el esquema. Sin embargo, la forma en que esta información aparecerá en las páginas no necesariamente contiene ese representante, podría contener alguna variante del mismo (por ejemplo un sinónimo). Para enriquecer el dominio proponemos 2 variantes al pedido.
La primera consiste en utilizar sinónimos de los conceptos solicitados, esto es, si la solicitud habla sobre actores de películas, también es de interés la información sobre protagonistas, intérpretes, etc.
La segunda consiste en utilizar instancias de los conceptos solicitados, o sea, si la solicitud incluye información sobre países también intentaremos encontrar información que incluya Uruguay, Estados Unidos, etc.
A continuación mostramos la estructura de los archivos utilizados.

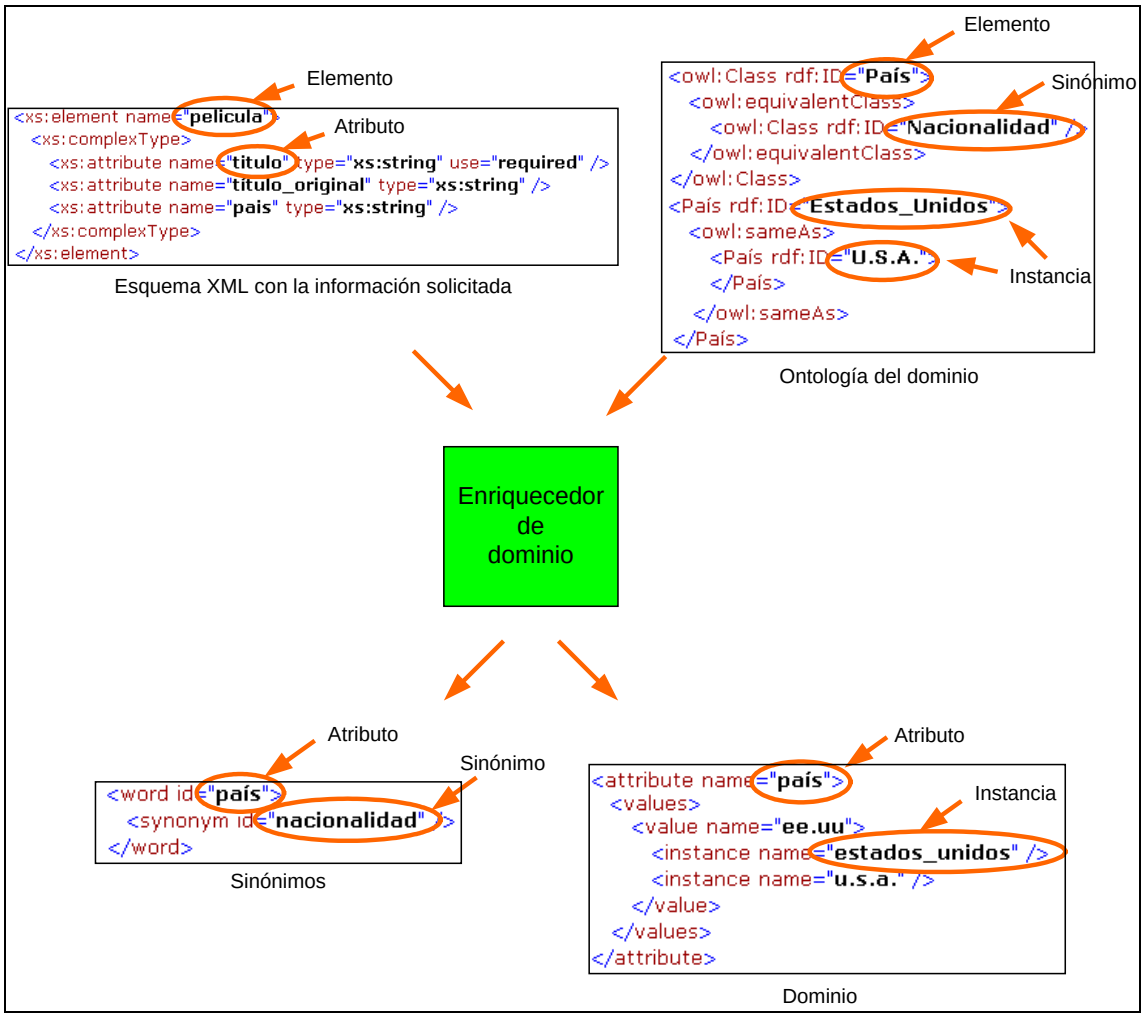


Figura 8 – Ejemplo de salida del Enriquecedor de dominio

Como puede verse en la figura 8 el enriquecedor de dominio produce para cada atributo un archivo con sus sinónimos y un archivo con sus instancias, obtenidos de la ontología (utilizando [26]).

4.1.2. Clasificador

La clasificación consiste en identificar páginas relevantes al pedido de un determinado conjunto.
La clasificación de páginas Web es un tema que ha sido tratado en proyectos anteriores que hemos estudiado. Entre ellos seleccionamos la tesis de maestría "Generador de Wrappers – de Álvaro Fernández (2003-2004)" [5] para reutilizar su clasificador, en forma de módulo de caja negra que anexamos a los nuestros. Este clasificador utiliza una consulta (para guiar la clasificación), un conjunto de páginas Web, una ontología y la frecuencia con la que se deben encontrar los conceptos en la página Web. Una explicación más detallada sobre este proceso se encuentra en el anexo "Proceso de clasificación".
Para poder reutilizarlo fue necesario averiguar como utiliza estas entradas. La idea del algoritmo de clasificación es la siguiente: a partir de la consulta se buscan sus palabras en la ontología para identificar sinónimos y/o términos relacionados, los términos encontrados son considerados relevantes y serán los que se buscarán en las páginas para decidir su relevancia, conjuntamente con la frecuencia de aparición necesaria para cada uno.

Nuestro sistema no se basa en una consulta como este módulo, sino en un esquema XML, por esto fue necesario evaluar la conversión de nuestro esquema en una consulta, o estudiar el funcionamiento del módulo para analizar si es posible utilizar una entrada más similar a nuestro esquema. Al analizar estas opciones encontramos que la más conveniente era utilizar como entrada una similar a nuestro esquema que incluyera además sinónimos e instancias. Esto se debió principalmente a que la forma en que el clasificador calcula sinónimos e instancias no es compatible con las ontologías que utiliza nuestro sistema.

Veamos ahora el resto de las entradas:

- El conjunto de páginas Web: este conjunto corresponde a páginas Web relevantes y no relevantes disponible para completar la información que nos fue solicitada en el esquema XML.
- La frecuencia con que debe aparecer cada concepto en las páginas Web: Actualmente se toma de un archivo de texto. Esta frecuencia depende de los conceptos de la ontología, nosotros no recibimos esa información por lo cual decidimos fijarla en el valor de 0,5 para todos los conceptos, podría indicarse esa información para cada concepto pero debería brindarla el usuario. Colocarla en la ontología no tendría sentido dado que la importancia de cada concepto dependería del pedido de información que se realiza.

Por lo cual para nuestro “clasificador adaptado” las entradas son: el conjunto de páginas Web a clasificar y el archivo de sinónimos, que construimos a partir de la ontología y el esquema XML a completar (el pedido de información).

El clasificador utiliza el método Vector Space Model (VSM) [5] (que detecta la similitud entre los conceptos a buscar y la página Web).

Finalmente devuelve una serie de páginas seleccionadas del conjunto de páginas originales.

4.1.3. Filtro

Una vez que hemos seleccionado las páginas relevantes, nos encontramos con el problema de que las páginas Web no son necesariamente un archivo XML, por ejemplo, no todos los tags que se abren tienen que ser cerrados. Para solucionar este problema utilizamos la herramienta Tidy [8] que convierte páginas escritas en HTML a páginas escritas en XHTML.
Además existe una serie de información que se encuentra en las páginas que nuestro sistema no utilizará como las imágenes, por lo que removemos los tags que refieran a imágenes. Así como otras que decidimos no utilizar en este prototipo como el formato o tamaño de las letras de manera que esta información también es eliminada.
Un tercer uso del filtro es convertir estructuras de forma de poder ser reconocidas por los mecanismos de extracción (que más adelante explicaremos).
Las reglas de filtrado se especifican en un archivo XML y se expresan de la forma: la expresión regular A se sustituye por la expresión regular B.

En la Figura 9 mostramos un ejemplo de uso del filtro donde se puede observar como los formatos (colores, negrita, cursiva, etc.) así como las imágenes y otros elementos de presentación son eliminados por el filtro. Manteniéndose las estructuras básicas y el contenido que podría ser útil.

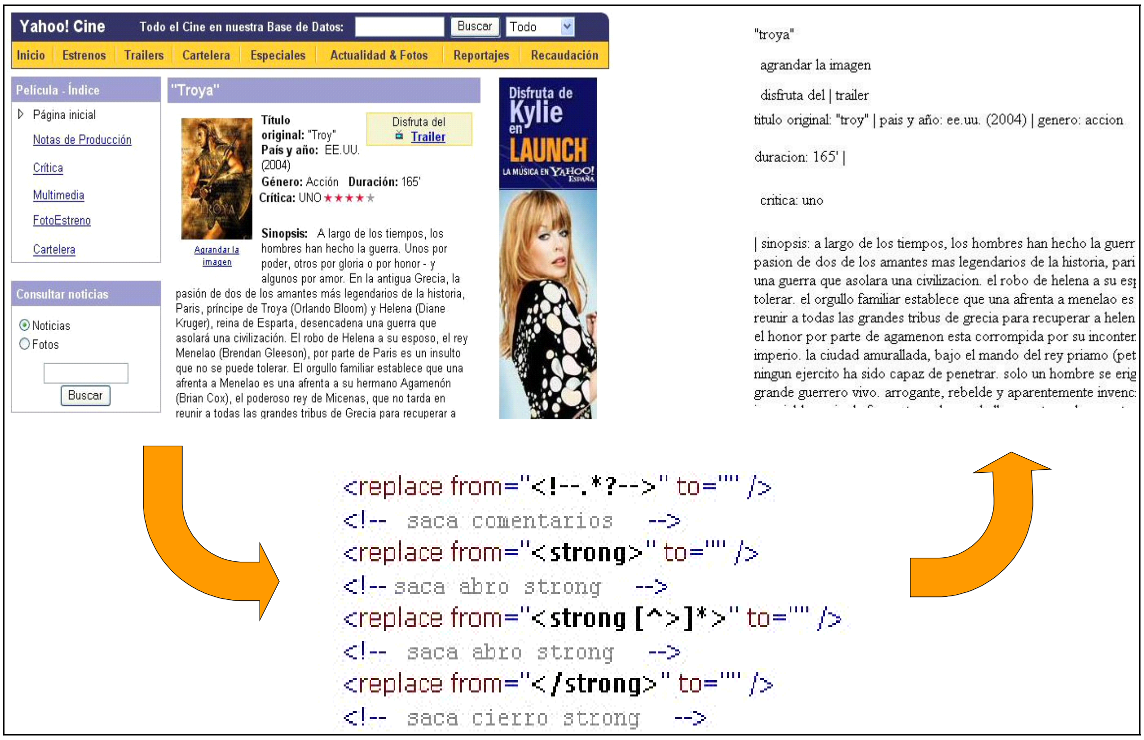


Figura 9 – Ejemplo de filtrado

4.1.4. Extractor

Existen varias alternativas a la hora de intentar extraer información de páginas Web.
Una opción es utilizar técnicas de manejo de lenguaje natural, por ejemplo las utilizadas en el proyecto: "*Técnicas de estado finito para la extracción automática de Metadata en la Web*" [4].
Una segunda opción es tratar de explotar la estructura de las páginas.
Una tercera alternativa sería una combinación de las técnicas anteriores donde se utiliza la segunda para localizar la información y la primera para extraer.
Este proyecto implementa un extractor basado en la segunda alternativa.
Para esto utilizamos una serie de reglas basadas en la estructura de la página para encontrar la información.
El algoritmo de extracción se encarga de evaluar las reglas (que se detallan más adelante) para cada atributo del elemento a buscar en la página. Las reglas están parametrizadas, por lo cual son evaluadas sustituyendo el valor comodín (%synonym%, que se explica a continuación), por los sinónimos e instancias de cada atributo obtenidos de la ontología. De esta forma se identifica en la página Web, la ubicación de la información solicitada.
El algoritmo de extracción se describe de forma más detallada en el anexo "Algoritmo de Extracción".

Reglas de extracción

Las reglas se basan en los siguientes conceptos:

- **Elemento:** representa un elemento en el esquema XML que se ingresa como pedido de información al sistema. Toda la información de la página que se encuentra en la ubicación del elemento pertenece a una misma

entidad. Ejemplos de elemento podrían ser: película, actor, hospital, médico, etc.

- **Meta-contenido:** representa un atributo en el esquema XML. Su ubicación consiste en la información necesaria para poder completar el valor de un cierto atributo, por ejemplo la aparición de “actores:” indica que el valor con el que se debe llenar el atributo actores se encuentra a continuación.
- **Contenido:** representa el valor con el que se completa un atributo del esquema XML (descrito por el meta-contenido). Ejemplos de contenido son: “Harry Potter” , “Troya”, etc.
- **Variables:** representan valores que en una regla pueden ser calculados una vez y luego reutilizados, su uso puede ser necesario como forma de mejorar la perfomance de una regla.
- **%synonym%**: es una variable que permite hacer referencia a un sinónimo o instancia de un atributo buscado.

Estos conceptos se ven reflejados en la definición de reglas mediante el esquema de la Figura 10:

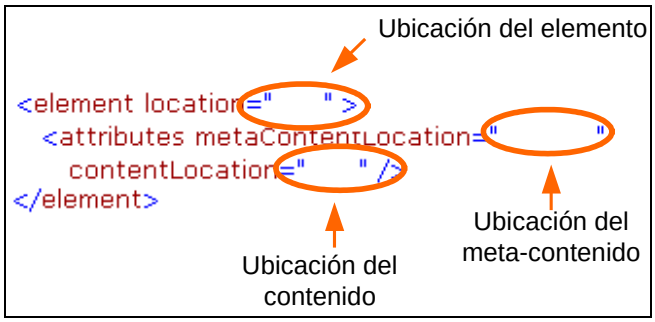


Figura 10 – Esquema de reglas

Reglas actuales:

Actualmente hay tres reglas que se utilizan en la implementación del prototipo. El orden de evaluación de las reglas no es relevante en el proceso de extracción. Dado que el proceso de extracción se basa en la estructura de las páginas Web, las reglas generadas cubren la mayoría de las estructuras incluidas en las páginas utilizadas en los casos de prueba.
Utilizaremos el siguiente esquema para mostrar las reglas que utiliza el sistema actualmente:

Estructura:	Estructura que intenta reconocer la regla
HTML:	HTML que se utiliza para representar la estructura
Ubicación del elemento:	Estructura que delimita el elemento en la página Web
Ubicación del elemento en X-Path:	X-Path a la estructura que delimita el elemento en la página Web
Ubicación del meta-contenido:	Estructura que delimita el meta-contenido en la página Web
Ubicación del meta-contenido en X-Path:	X-Path a la estructura que delimita el meta-contenido en la página Web

Ubicación del contenido:	Estructura que delimita el contenido en la página Web
Ubicación del contenido en X-Path:	X-Path a la estructura que delimita el contenido en la página Web
Variables:	Variables utilizadas por la regla
Ejemplo	Figura que muestra un ejemplo de aplicación de la regla

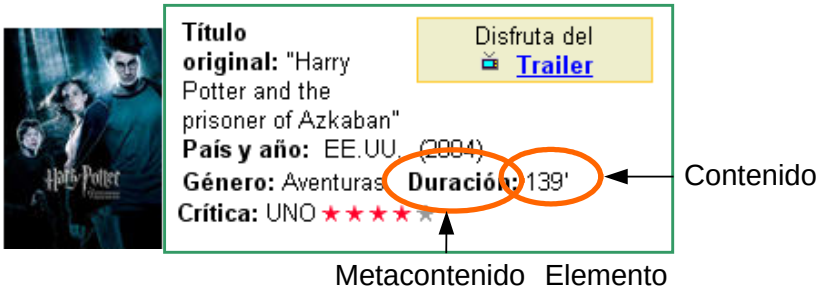
Estas reglas se encuentran en un archivo XML independiente, en el cual podrían agregarse nuevas reglas para atacar más estructuras o buscar información de forma más específica.

Regla 1:

Esta regla intenta buscar información tal que el meta-contenido y el contenido se encuentran dentro del mismo tag HTML (por ejemplo dentro de un párrafo o una celda de una tabla) utilizando el carácter ":" para separar uno del otro.

Estructura:	Párrafos que contienen Metacontenido: Contenido, dentro de alguna estructura HTML como <p>, , etc.
HTML:	<?> Metacontenido: Contenido </?>
Ubicación del elemento:	Celda de una tabla donde se encuentra la información.
Ubicación del elemento en X-Path:	//td/descendant::text()[contains(.,'%synonym%')]/ ancestor::td[1]
Ubicación del meta-contenido:	Tag donde se encuentra la información
Ubicación del meta-contenido en X-Path:	./descendant::text()[contains(.,'%synonym%')]
Ubicación del contenido:	El mismo lugar donde está el meta-contenido
Ubicación del contenido en X-Path:	.

Ejemplo



Regla 2:

Esta regla considera que existe en la página una tabla donde se encuentra toda la información de un elemento (ej: película) que tiene dos columnas donde la primera es el meta-contenido y la segunda es el contenido.

Estructura:	Información contenida en una tabla donde el meta-contenido y el contenido se encuentran en celdas contiguas.
HTML:	<td> Metacontent </td> <td> Content </td>
Ubicación del elemento:	Tabla donde se encuentra la información.
Ubicación del elemento en X-Path:	//td/descendant-or-self::*/text() [contains(.,'%synonym%')]/ancestor::table[1]
Ubicación del meta-contenido:	Celda donde se encuentra el sinónimo del atributo
Ubicación del meta-contenido en X-Path:	./descendant-or-self::*/text()[contains(.,'%synonym%')]
Ubicación del contenido:	Celda frente a la anterior
Ubicación del contenido en X-Path:	./ancestor::td[1]/following-sibling::*/descendant-or-self::text()[1]

Ejemplo

Regla 3:

Esta regla busca en la página una tabla con cabezal, donde el cabezal es el meta-contenido y las filas de la tabla son una serie de elementos.

Estructura:	Información en tabla con títulos.
HTML:	<td> Metacontent </td> <td> Content </td>
Ubicación del elemento:	Cada fila de la tabla.
Ubicación del elemento en X-Path:	//td/descendant-or-self::* [contains(text(),'%synonym%')] /ancestor::table[1]/descendant::tr [count(preceding-sibling::*)>%fila%][count(td)>1]
Ubicación del meta-contenido:	El cabezal de la tabla.
Ubicación del meta-contenido en X-Path:	./ancestor::table/descendant-or-self::*/text()[.='%synonym%']
Ubicación del contenido:	Celda en la fila y columna correspondiente dentro de la tabla. (Evaluado desde el elemento)-
Ubicación del contenido en X-Path:	./descendant::td[count(preceding-sibling::td)=%columna%]/descendant-or-self::text()[1]
Variables:	Fila = count(//td/descendant-or-self::*[contains(text(),'%synonym%')]/ ancestor::tr/preceding-sibling::tr) Columna = count(//td/descendant-or-self::*[contains(text(),'%synonym%')]/ ancestor-or-self::td[1]/preceding-sibling::td)

Ejemplo

Título	Recaudación semanal	Recaudación total
El día de mañana	2.389.679	8.270.064
Troya	1.329.807	15.531.436
El Castigador	659.602	761.818
The Ladykillers	368.995	2.072.748
Van Helsing	296.190	9.158.050
FBI: Frikis buscan incordiar	231.235	231.235
Giro inesperado	213.386	1.361.761
La matanza de Texas 2004	105.191	814.424
El efecto mariposa	100.795	2.491.544
Héctor	51.374	703.398

Problemas encontrados.

¿Qué sucede cuando no hay meta-contenido?

Esta pregunta surge del siguiente ejemplo:

Figura 11 – Información sin meta-contenido

La respuesta podría atacarse por dos frentes:

1. Se sabe que ese es el título de una película (representado en que se encuentra en la ontología como instancia de titulo de película).
2. Se puede deducir por el contexto de la página que ese es el título.

Analizando los diversos casos en que se presentó el problema no pudimos encontrar un patrón común que nos permitiera deducir información que no posee meta-contenido de la página, por lo que optamos por la opción 1.

¿Qué sucede cuando la información de una película se encuentra en diferentes estructuras dentro de la página? ¿La página pasa a ser el elemento?

Un ejemplo de este problema es:

Figura 12 – Información particionada

Donde la información se encuentra complementada en distintas estructuras, para esto el sistema intenta complementar información de diferentes estructuras según el siguiente lineamiento: “Si dos elementos del resultado contienen información complementaria entonces se unen para formar un único elemento”.

¿Los atributos pueden ser multi-valuados?

La implementación actual no posee soporte para multi-valuados de forma de no agregar complejidad a la implementación, la información de actores por ejemplo se obtiene como el string que se encuentra en la página.

Algunas limitaciones.

De todo lo anterior se deduce que información presentada en formas que no son soportadas por las reglas no será extraída. Algunos ejemplos:

El meta-contenido se encuentra en una celda superior a la del contenido:

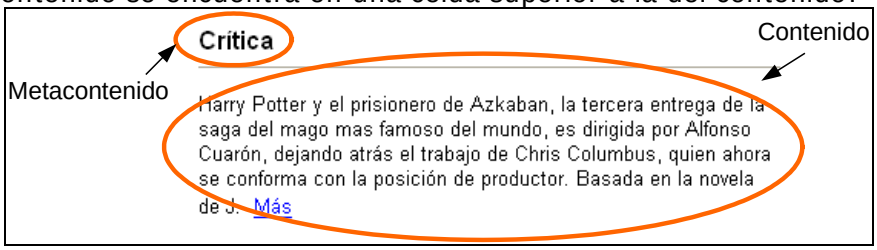


Figura 13 – Información en estructura no soportada

Para encontrar la información es necesario hacer clic en un hipervínculo.

Figura 14 – Información contenida en un hipervínculo

La información se encuentra contenida en una imagen

Figura 15 – Información contenida en imagen

4.1.5. Generador de salida.

El generador de salida tiene por objetivo eliminar la información no relevante que rodea a la información extraída.
Utiliza expresiones regulares para seleccionar la información.

Ejemplos:

- Si el resultado obtenido es <meta-contenido>:<contenido> como se obtendría con la primer regla mostrada elimina <meta-contenido>: del resultado
- Si el resultado tiene tipo año, trata de ubicar un entero de 4 dígitos.

Esto podría sustituirse por el uso de técnicas más avanzadas como reconocimiento de lenguaje natural y/o inteligencia artificial.

4.2. Metadata.

La metadata de este módulo consiste en:

- **Esquema de reglas:** Este archivo posee el esquema de definición de las reglas, cada regla que pertenece al sistema debe seguir este esquema. Este esquema se muestra en la Figura 10.
- **Reglas:** Este archivo posee la definición de las reglas que el sistema utiliza actualmente, de requerirse agregar reglas estas deben ser incluidas en este archivo. Un ejemplo del contenido del archivo se muestra en la Figura 16.

```
<element location="//td/descendant::text()[contains(.,'%synonym%')]/ancestor::td[1]" >
  <attributes metaContentLocation="./descendant::text()[contains(.,'%synonym%')]" contentLocation="." />
</element>
```

Figura 16 – Extracto de especificación en XML de la regla 1

- **Archivo de reglas utilizadas:** Este archivo contiene las reglas que se ejecutaron sobre la página y el resultado de dicha ejecución. En la Figura 17 se muestra un ejemplo de archivo de reglas utilizadas.

```
<elementRule location="/*[01]/*[02]/*[04]/*[01]/*[01]/*[01]/*[01]/*[01]/*[01]/*[02]/*[01]/*[01]/*[01]"
  aceptance="A0" friendlyLocation="//td/descendant::text()[contains(.,"%synonym%")]/ancestor::td[1]">
  <attributes metaContentLocation="./descendant::text()[contains(.,"%synonym%")]" contentLocation=".">
    <attribute name="título" type="xs:string" required="true">
      <metaContent synonym="harry potter y el prisionero de azkaban" isInstance="true" />
      <content result="harry potter y el prisionero de azkaban" />
    </attribute>
    <attribute name="título_original" type="xs:string" required="false">
      <metaContent synonym="harry potter and the prisoner of azkaban" isInstance="true" />
      <content result="harry potter and the prisoner of azkaban" />
    </attribute>
    <attribute name="país" type="xs:string" required="false">
      <metaContent synonym="ee.uu" isInstance="true" />
      <content result="ee.uu" />
    </attribute>
    <attribute name="duración" type="xs:duration" required="false">
      <metaContent synonym="duracion" isInstance="false" />
      <content result="duracion:139' |" />
    </attribute>
```

Figura 17 - Reglas utilizadas

Existen además, otros archivos que ayudan a la ejecución del sistema (como la definición de los filtros, el archivo de reglas instanciadas, etc.).

4.3. Análisis de manejo de evolución de las fuentes

Para analizar el efecto de los cambios es necesario considerar que existe un notificador de cambios (según relevamiento realizado en [6]) que devuelve una lista de:

- XPath: conjunto de XPath de las dos páginas combinados (o sea si es el mismo no lo agrego)
  - Valor anterior del Xpath: resultado de evaluarlo en la página anterior
  - Valor actual del Xpath: resultado de evaluarlo en la página actualizada.

Caso	Valor anterior	Valor actual	Descripción
1	X	X	No hubo cambio
2	X	Y	Cambio contenido
3	X	Null (ej. Se borró el td)	Cambio de estructura (eliminación)
4	Null	X	Cambio de estructura (agregado)

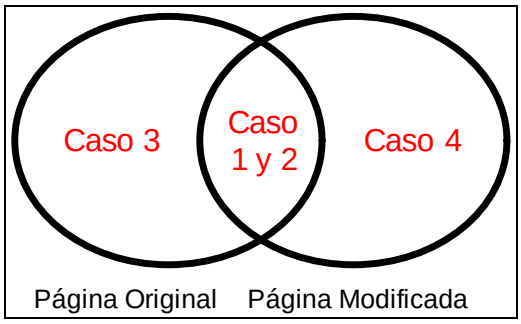


Figura 18 - Formas de evolución de las fuentes

Para cada caso de cambio de la Figura 18, estudiamos el impacto y su solución. Mostramos ejemplos en cada caso, basados en la siguiente entrada.

Esquema

```
<xs:element name="pelicula">
    <xs:complexType>
        <xs:attribute name="título_original" type="xs:string"/>
        <xs:attribute name="idioma" type="xs:string"/>
        <xs:attribute name="actores" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

Reglas evaluadas (Metadata)

```
<attribute name="título_original">
    <metaContent synonym="TÍTULO ORIGINAL"/>
    <content result="Harry Potter And The Prisoner Of Azkaban"/>
</attribute>
<attribute name="duración">
    <metaContent synonym="DURACIÓN"/>
    <content result="163 minutos"/>
</attribute>
<attribute name="idioma">
    <metaContent synonym=""/>
    <content result=""/>
</attribute>
```

4.3.1. Cambio en el contenido de la página

El cambio en este caso será notificado indicando un XPath, su evaluación en la página anterior (valor anterior) y su evaluación en la página actualizada (valor nuevo)
Para este caso debemos considerar dos opciones:

- Que el cambio sea de contenido
- Que el cambio sea de metacontenido

Si el cambio es de contenido entonces, debemos cambiar en el contenido del atributo que corresponda, el valor anterior por el nuevo.
Si el cambio es de metacontenido entonces, debemos chequear para el atributo que corresponda si el nuevo valor es sinónimo del metacontenido anterior o no.

- Si es sinónimo entonces bastará con cambiar el sinónimo asociado a ese atributo.
- Si no es sinónimo entonces debemos averiguar si el nuevo valor es sinónimo de alguno de los demás atributos que buscamos, si lo es entonces, a ese atributo le asigno el sinónimo encontrado y como contenido le asigno el resultado que tenía el atributo anterior. Además, tendremos que borrar el sinónimo y el contenido que tenemos en el archivo de reglas evaluadas para el atributo que se eliminó.

Ejemplo 1: cambio de contenido

Cambia la duración de la película.

Reporte del cambio:

XPath: ...
Valor anterior: 163 minutos
Valor actual: 180 minutos

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="duracion">
    <metaContent synonym="DURACIÓN"/>
    <content result="180 minutos "/>
</attribute>
```

Ejemplo 2: cambio de meta contenido

Cambia LISTA DE INTÉRPRETES por PROTAGONISTAS.

Reporte del cambio:

XPath:...

Valor anterior: LISTA DE INTÉRPRETES
Valor actual: PROTAGONISTAS

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="actores">
    <metaContent synonym="PROTAGONISTAS"/>
    <content result="Daniel Radcliffe"/>
</attribute>
```

Ejemplo 3: cambio de meta contenido

Cambia DURACIÓN por IDIOMA

Reporte del cambio:

XPath:...
Valor anterior: DURACIÓN
Valor actual: IDIOMA

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="duracion">
    <metaContent synonym=""/>
    <content result=""/>
</attribute>
<attribute name="idioma">
    <metaContent synonym="IDIOMA"/>
    <content result="163 minutos"/>
</attribute>
```

Ejemplo 4: cambio de meta contenido

Cambia DURACIÓN por PRODUCTORES

Reporte del cambio:

XPath:...
Valor anterior: DURACIÓN
Valor actual: PRODUCTORES

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="duracion">
    <metaContent synonym=""/>
    <content result=""/>
</attribute>
```

4.3.2. Cambio de estructura (eliminación)

Consideramos ahora otro caso de la tabla, que se haya eliminado algo en la estructura.
La estructura que se eliminó podía tener contenido o metacontenido, de todas formas los casos se tratan prácticamente de la misma manera.
Debemos eliminar el string correspondiente al sinónimo y resultado del contenido para el atributo que corresponda.

Ejemplo 5: cambio de estructura (eliminación)

Se elimina la fila que contiene duración

Reporte del cambio:

XPath:...
Valor anterior: DURACIÓN
Valor actual: NULL

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="duracion">
    <metaContent synonym=""/>
    <content result=""/>
</attribute>
```

4.3.3. Cambio de estructura (agregado)

En este caso debemos averiguar si el resultado del XPath nuevo es sinónimo de algún atributo de los que buscamos, si lo es, entonces vemos si hay alguna regla para el metacontenido que nos lleve a esa ubicación. En caso afirmativo trataremos de ejecutar la regla para el contenido, si aplica entonces, asignamos el sinónimo y el resultado al contenido del atributo que corresponde.

Ejemplo 6: cambio de estructura (agregado)

Se agrega una fila que contiene idioma

Reporte del cambio:

XPath:...
Valor anterior: NULL
Valor actual: IDIOMA

Aplicando el mecanismo explicado anteriormente se obtiene:

```
<attribute name="idioma">
    <metaContent synonym="IDIOMA"/>
    <content result="Inglés"/>
</attribute>
```

Capítulo 5. Integración de información

5.1. Introducción

Para diseñar la arquitectura debemos considerar cual es la función del mediador, sus entradas y salidas. Como entrada disponemos de las salidas del extractor, que consisten en varias instancias del esquema XSD inicial. Además, debemos tener en cuenta la ontología, ya que nos puede brindar información útil a la hora de integrar los datos, por ejemplo, conjuntos de instancias posibles que permiten que términos que corresponden al mismo concepto de las diferentes páginas pasen a ser el mismo término.
El mediador recibe varias instancias del XSD inicial correspondiente a diferentes páginas Web y debe integrarlas de forma de devolver una sola instancia con la información requerida. La salida corresponde a esa instancia. Otro factor importante a considerar, es la generación de la metadata, que se conservará en archivos XML al igual que en el extractor (ver Figura 19).

5.2. Arquitectura del mediador

El mediador lo diseñamos en base a módulos definidos teniendo en cuenta las etapas que debe seguir el proceso. En la Figura 19 mostramos la arquitectura.

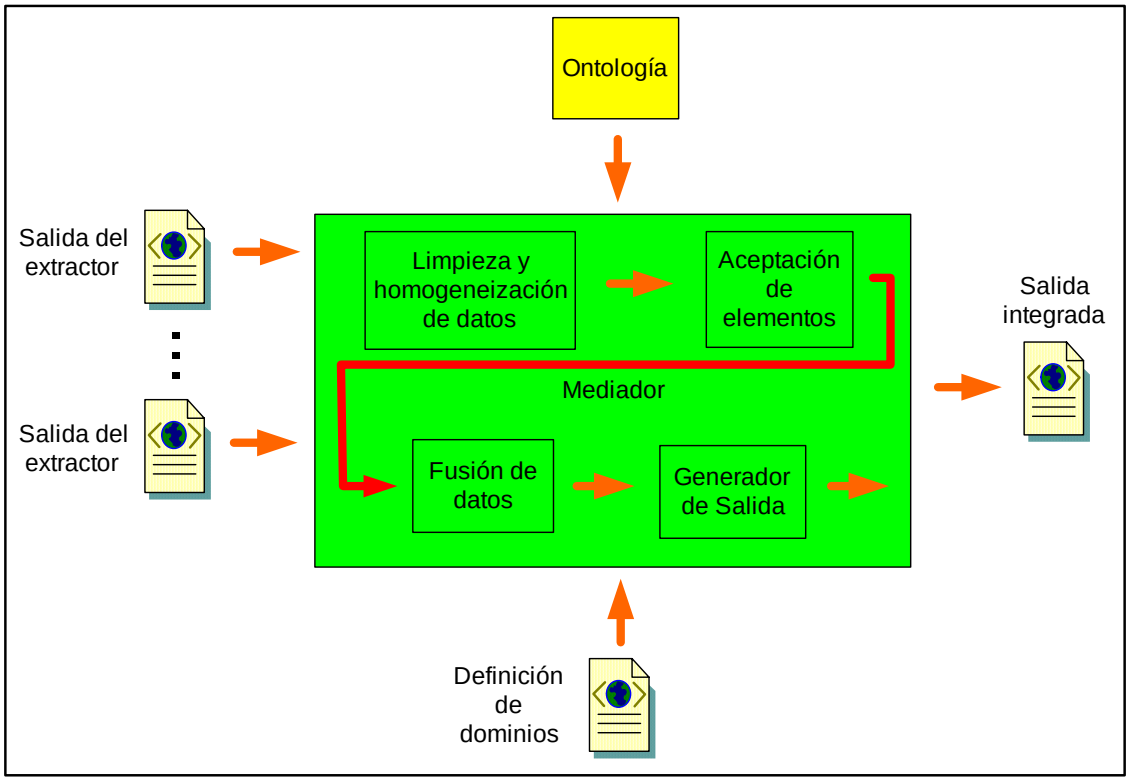


Figura 19 – Arquitectura del Mediador

El módulo de limpieza y homogeneización de datos se encarga de preparar la información extraída para su integración. El módulo de aceptación de elementos filtra elementos no válidos a consecuencia de la limpieza y homogeneización. El módulo de fusión de datos realiza la integración a nivel de datos, devolviendo como

resultado la información integrada con la metadata necesaria. Por último, el módulo generador de salida filtra la metadata, devolviendo un archivo sin metadata con el formato especificado en el esquema de solicitud de información.

5.2.1. Limpieza y homogeneización de datos

Este módulo tiene como entradas la salida proveniente del extractor y la definición de dominios y genera como salida un archivos con la información homogeneizada y la metadata necesaria (ver Figura 20)

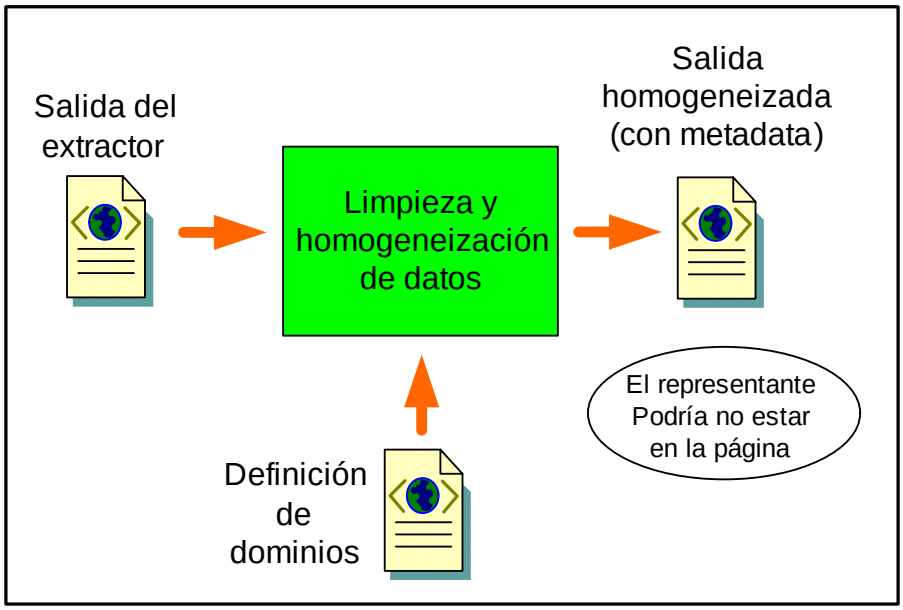


Figura 20 – Módulo de limpieza y homogeneización de datos

La definición del dominio debemos combinarla con la salida del extractor, o sea el XML con los datos requeridos, para poder realizar cierta limpieza de texto y chequeo de tipos, eliminando la información que no corresponda. Por ejemplo: si en el atributo país tenemos “el país es EE.UU.” querríamos pasar a tener simplemente “EE.UU”, dado que debemos buscar en la frase inicial una instancia de país. Pero a medida que realizamos esta limpieza podemos ya unificar los términos, esto significa que por ejemplo si tenemos “EE.UU.” como término para el atributo país, lo sustituiremos por el término “Estados Unidos” que será su representante. El representante se elige aleatoriamente dentro del grupo de términos que representan una misma instancia. O sea que, para cada término existirá uno que representa a la instancia y que lo va a sustituir, de esta forma a la hora de integrar podremos comparar directamente los strings.
En la metadata generada por este módulo se registra la porción de texto extraída de la página Web que origina la información que se integrará.
Este módulo devuelve un archivo XML que consiste de la información “homogeneizada” y metadata que indique como se realizó el proceso e indicando que limpieza ocurrió.

Metadata generada

El archivo XML de salida contiene para cada atributo de un elemento la información originalmente extraída para ese atributo (que denominamos **originalValue**), si es un atributo que tiene instancias en la ontología (o sea una enumeración de valores posibles) tiene sentido obtener cual es la instancia que se encuentra en la información original (que denominamos **instanceValue**), y de esa instancia

debemos conservar el representante que se utiliza (que denominamos **value**) para homogeneizar los datos. Para el caso de **originalValue** se taggea la ocurrencia de la instancia que se utiliza (con el símbolo # delimitándola), esto es útil para saber si un cambio afecta en la elección de la instancia y por lo tanto debe propagarse, o no.

Ejemplo:

Para el caso del atributo país ya mencionado debemos registrar la frase original: "el país es EE.UU", luego la instancia que se utilizó: "EE.UU." y finalmente el representante: "Estados Unidos". De esta forma mantenemos la trazabilidad del proceso que permite en caso de evolución detectar si un cambio debe reflejarse o no en el paso posterior del proceso. Por ejemplo: si hay un cambio dentro de la frase inicial pero no modificó al valor que se utilizó ("EE.UU."), entonces bastará con modificar la frase inicial pero no será necesario propagar el cambio.

```
<root>
   <element name="pelicula">
         <attribute name="género" value="aventura" valueInstance="aventuras" originalValue="#aventuras# "/>
         <attribute name="país" value="ee.uu" valueInstance="estados unidos" originalValue=" #estados unidos# "/>
         <attribute name="título" value="harry potter y el prisionero de azkaban" valueInstance=" harry potter y el
prisionero de azkaban " originalValue="# harry potter y el prisionero de azkaban #"/>
     </element>
</root>
```

5.2.2. Aceptación de elementos

En este punto, analizando la salida, notamos que es necesario realizar nuevamente una aceptación de los elementos según los valores de sus atributos, dado que es posible que en el paso de limpieza algunos atributos que tenían valor hayan quedado con valor vacío (""). Lo importante para la integración es que ciertos atributos requeridos (que fueron indicados así en el esquema original) no pueden quedar vacíos. Entonces, necesitamos de un módulo que chequee esto y por lo tanto necesita como entrada el esquema original (para revisar cuales son los atributos requeridos). La salida de este nuevo módulo que llamamos de "aceptación de elementos" consiste de un archivo XML que cumple con el esquema original y contiene los datos a integrar (salida homogeneizada sin metadata). (ver Figura 21)

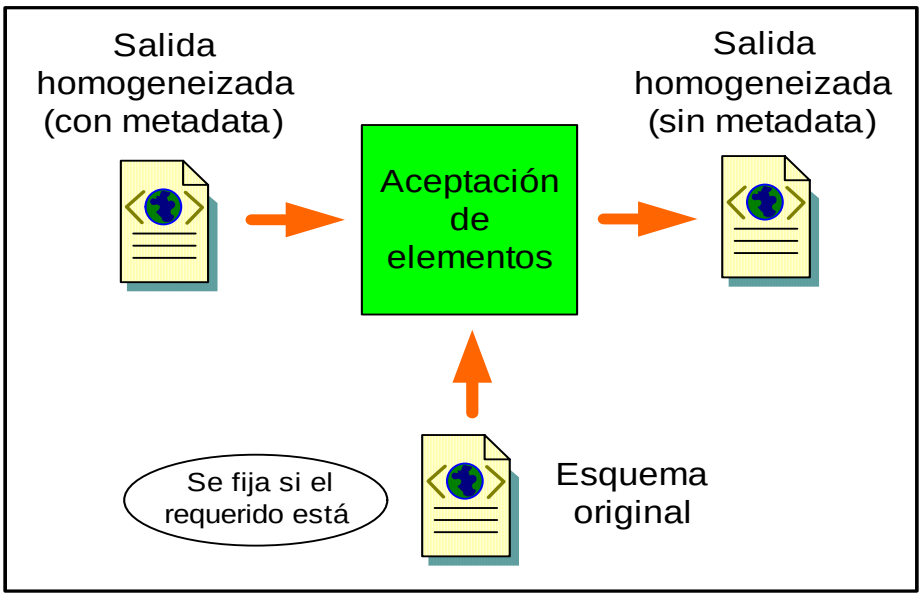


Figura 21 – Módulo de aceptación de elementos

Hasta este momento no hemos realizado aún la integración de la información de las distintas páginas, sino que fue necesaria una preparación previa para cada archivo.

5.2.3. Fusión de datos

Luego de la preparación, debemos integrar las diversas salidas homogeneizadas, instancias del XSD original y construir una única salida integrada. Por lo tanto, construimos un módulo que llamamos de “fusión de datos” (ver Figura 22).
En la etapa de fusión de datos también es necesaria la generación de metadata, dado que en este módulo seleccionamos la información que devolvemos para cada atributo. Esa selección la realizamos en base a opciones, o sea información correspondiente a ese atributo proveniente de diferentes fuentes (páginas Web).

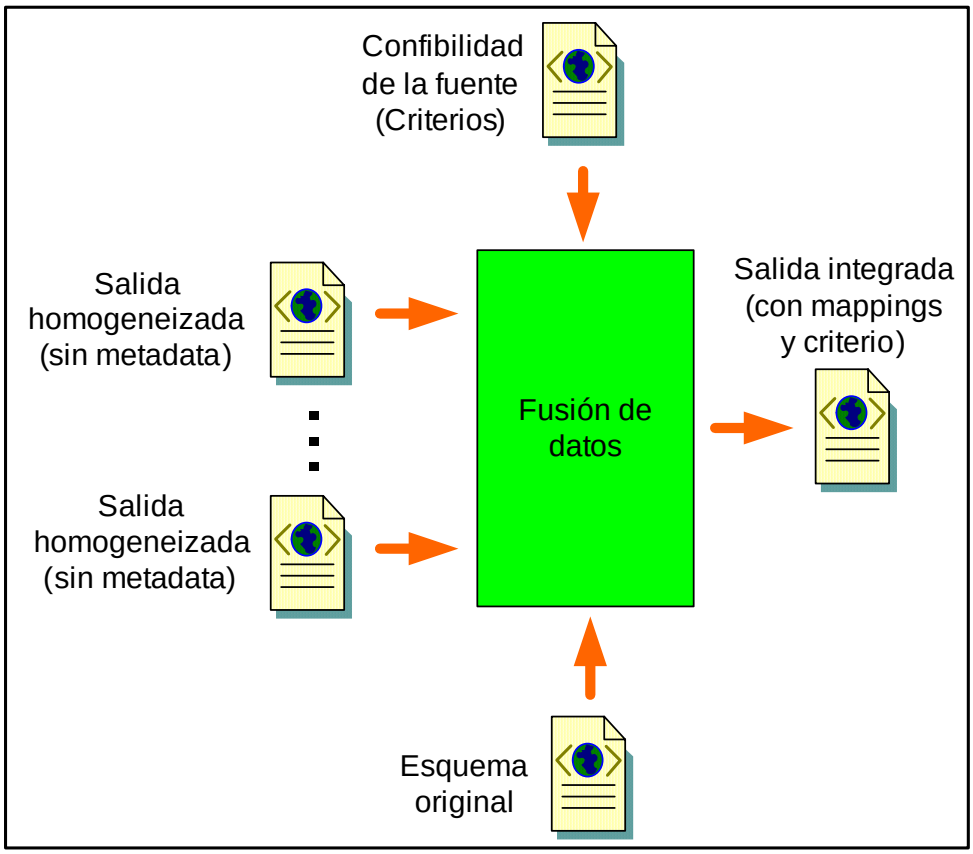


Figura 22 – Módulo de fusión de datos

Al realizar la integración es necesario algún mecanismo de prioridades que permita determinar la opción será elegida en base a un criterio de selección. En este caso utilizamos el criterio de confiabilidad las fuentes, que toma el dato proveniente de la fuente de mayor confiabilidad, utilizando como entrada un archivo con información de la confianza de cada fuente. De todas formas, se posibilita la extensión de los criterios permitiendo la creación de nuevos criterios personalizados a aplicar en la integración. Para crear un nuevo criterio es necesario crear una nueva clase Java [25] que implemente la interfaz ICriterion.
La integración se realiza entre elementos que tienen igual valor para el atributo requerido y los demás atributos del elemento se obtienen resolviendo los conflictos mediante el criterio de integración especificado.

Metadata generada:

A pesar de que la salida debe cumplir con el esquema original, consideramos necesario que se conserve metadata, que indique de qué fuente se obtuvo el dato (mappings) para cada atributo de los elementos y qué criterio se utilizó.
El archivo XML de salida contiene para cada atributo de un elemento, en primer lugar el valor que fue seleccionado, la fuente de la que proviene y su confianza, en segundo lugar conserva la lista de opciones que tuvo para realizar la elección, cada una de las cuales indica el valor que se había obtenido, la fuente de la que proviene y la confianza de esa fuente
Para el caso del criterio de confiabilidad de las fuentes elegido, utilizando la metadata podemos verificar que realmente se eligió el valor de mayor confianza.
Entonces, la salida de este módulo consiste de un archivo XML que es la salida integrada con metadata, esto posibilita la evolución considerando si el cambio en cierta página afecta el resultado final o no. Si la información de la página que

cambió fue la seleccionada debe propagarse el cambio, sino solamente actualizamos la opción en la metadata o la eliminamos si dejó de contener la información de interés, dado que si el dato que cambió no era utilizado, ya que no correspondía por ejemplo a la fuente más confiable, no debe propagarse hacia adelante el cambio.

Ejemplo:
Se observa para cada atributo del elemento película, el valor que se eligió (value), el nombre de la fuente de donde proviene (source) y la confiabilidad de esa fuente (trust). También, se registran las opciones que se tuvieron en el proceso de integración. Para cada una se guarda el valor que se extrajo, la fuente y su confiabilidad, pudiendo verificar la opción elegida.

```
<element name="pelicula">
- <attribute name="título" value="harry potter y el prisionero de azkaban" source="Cartelera Montevideo Shopping.htm"
trust="0.8">
  <options value="harry potter y el prisionero de azkaban" source="ADICTOSALCINE_COM - Harry Potter.htm" trust="0.1" />
  <options value="harry potter y el prisionero de azkaban" source="Cartelera Montevideo Shopping.htm" trust="0.8" />
  <options value="harry potter y el prisionero de azkaban" source="UruguayTotal Cartelera - Harry Potter y el prisionero de
Azkaban.htm" trust="0.7" />
  <options value="harry potter y el prisionero de azkaban" source="UruguayTotal Cartelera de Cine x Películas.htm" trust="0.6"
/>
  <options value="harry potter y el prisionero de azkaban" source="Yahoo! Cine - Harry Potter y el prisionero de Azkaban.htm"
trust="0.5" />
  </attribute>
- <attribute name="país" value="ee.uu" source="UruguayTotal Cartelera - Harry Potter y el prisionero de Azkaban.htm"
trust="0.7">
  <options value="ee.uu" source="ADICTOSALCINE_COM - Harry Potter.htm" trust="0.1" />
  <options value="ee.uu" source="UruguayTotal Cartelera - Harry Potter y el prisionero de Azkaban.htm" trust="0.7" />
  <options value="ee.uu" source="Yahoo! Cine - Harry Potter y el prisionero de Azkaban.htm" trust="0.5" />
  </attribute>
- <attribute name="género" value="aventura" source="Cartelera Montevideo Shopping.htm" trust="0.8">
  <options value="aventura" source="ADICTOSALCINE_COM - Harry Potter.htm" trust="0.1" />
  <options value="aventura" source="Cartelera Montevideo Shopping.htm" trust="0.8" />
  <options value="aventura" source="UruguayTotal Cartelera - Harry Potter y el prisionero de Azkaban.htm" trust="0.7" />
  <options value="aventura" source="UruguayTotal Cartelera de Cine x Películas.htm" trust="0.6" />
  <options value="aventura" source="Yahoo! Cine - Harry Potter y el prisionero de Azkaban.htm" trust="0.5" />
  </attribute>
</element>
```

5.2.4. Generador de salida

Ya hemos cumplido con la función del mediador pero debemos devolver como salida de la totalidad del proceso de integración un archivo XML que cumpla con el esquema original, por lo tanto debemos generar un nuevo archivo que no contenga metadata y contenga únicamente los datos solicitados. Para esto construimos un nuevo módulo pequeño llamado “generador de salida” que devuelve la salida integrada (ver Figura 23).

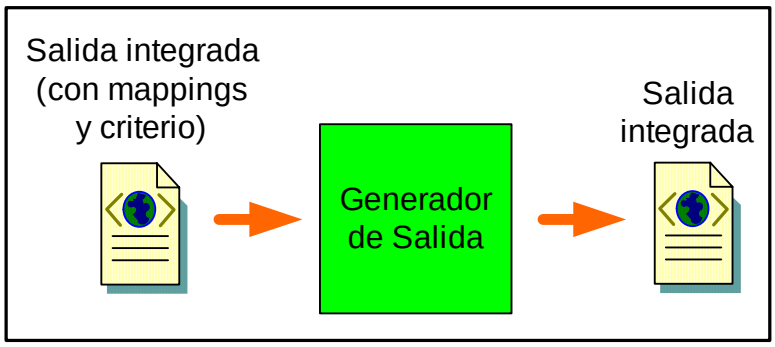


Figura 23 – Módulo generador de salida

Ejemplo:

```
<ResultSet>
  <pelicula género="aventura" país="ee.uu" título="harry potter y el prisionero de azkaban"/>
</ResultSet>
```

5.3. Arquitectura detallada

En la Figura 24 se presenta la arquitectura detallada, cuyas partes fueron explicadas en las secciones anteriores. En el anexo "Algoritmo de integración" se presenta el seudo código de cada uno de los módulos.

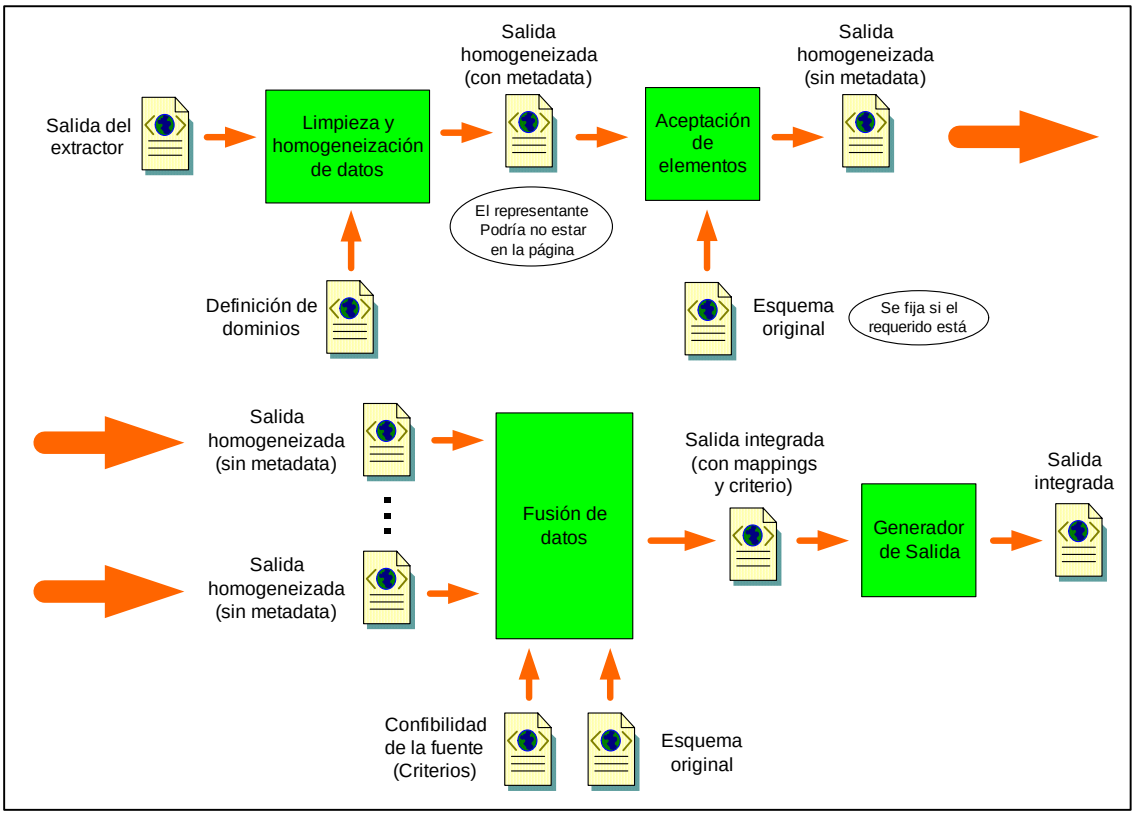


Figura 24 – Arquitectura detallada del mediador

Capítulo 6. Testing

6.1. Nivel de calidad

Describiremos aspectos importantes sobre el testeo que llevamos a cabo para alcanzar el nivel 3 de calidad según especificación en http://www.fing.edu.uy/~dvallesp/Tesis/webActividades/index.htm

Cumpliendo las actividades de este nivel debería conseguirse lo siguiente:

- Conocer las fallas del sistema y poder reproducirlas.
  Para aumentar la capacidad de corregir las faltas en el código, conviene automatizar las pruebas y así comprobar que el sistema ya no falla en los casos automatizados (modificabilidad, mantenibilidad)
- Se conoce la cantidad de fallas relacionada con la cantidad de casos de prueba funcionales ejecutados

En este nivel se conocen fallas del sistema y cuanta funcionalidad del mismo ha sido cubierto por los casos de prueba. De esta manera, se puede conocer que cosas el sistema no brinda (o de qué manera determinadas funcionalidades no se deben ejecutar) y preparar futuras versiones sabiendo de antemano que es lo que hay que corregir. También se tiene una idea de que falta testear. Este nivel permite tener confianza en la funcionalidad del sistema pudiendo hacer demos y teniendo conocimiento de algunas fallas del mismo, lo cuál permite "moverse" por el sistema con mayor flexibilidad.

6.2. Punto de partida del testing

De las actividades comprendidas en el nivel de calidad 3 nos enfocaremos en esta etapa a las de verificación y validación. Recordando la metodología de trabajo de dos iteraciones describiremos las actividades de testing en cada una de ellas. En la primer iteración desarrollamos en función de cumplir con los requerimientos básicos del sistema y teniendo en cuenta el test de aceptación. Este caso nos permite testear el sistema para verificar que cumple con la especificación de requerimientos (por más detalle, ver anexo “Cubrimiento de requerimientos”). Al final de la primera iteración se mostró el funcionamiento del sistema para el test de aceptación y fue aprobado en su funcionalidad y comportamiento.

En la segunda iteración se agregaron funcionalidades a las ya existentes (previstas desde la primera iteración) y se testeó el sistema con nuevos casos de prueba de forma de tener claro cómo se comporta frente a diferentes entradas. En el anexo “Mejoras realizadas” se enumeran las principales mejoras.

En conjunto con lo anterior, analizamos las salidas definiendo una medida del grado de acierto en cada caso.

La forma de trabajo que se siguió fue agregar algunas funcionalidades que se consideran importantes para cualquier caso que se podría ejecutar, corregir las fallas encontradas para esos casos, y buscar nuevos casos de prueba. Los nuevos casos de prueba fueron ejecutados con esa versión del programa y se analizaron las salidas para detectar cómo se comportaban y por qué. Luego se continuó agregando las funcionalidades pendientes, testeando paralelamente el caso de aceptación y los demás. Se dio la mayor importancia al caso de aceptación de forma de mejorar su comportamiento, pero a su vez se analizó que sucedía con los demás casos, si mejoraban entonces se continuaba, pero si empeoraban se analizaba la causa, de esta forma se pudieron detectar errores, particularidades del caso de aceptación o aspectos que el sistema no cubría.

6.3. Métricas

6.3.1. Indicadores de mejora de los casos

Construimos para cada caso la salida esperada de forma de automatizar el testeo, ya que analizar cada salida obtenida comparándola con la página Web del caso, insumiría un esfuerzo muy grande.

Desarrollamos, entonces, un módulo de testeo que se encarga de comparar la salida obtenida con la salida esperada.

Este módulo permite obtener archivos XML que indican para cada ítem de información solicitada, qué información debería haberse obtenido, qué información se encontró, un indicador de la precisión de la información y un indicador de resultado.

Para poder optimizar la detección de mejora en los casos se considerarán los dos indicadores:

- Accuracy (precisión): mide la precisión con que se obtuvo la salida.
- Result (resultado): indica si la se obtuvo información de más, o información de menos, o no se encontró, o si es información errónea, o es correcta.

El módulo desarrollado funciona de la siguiente forma:

- Si la información es correcta, o sea se obtuvo exactamente lo que se esperaba:
  Accuracy: es igual a 1.0
  Result: “OK”
- Si la información esperada está contenida en la información obtenida:
  Accuracy: se calcula como la relación entre la cantidad de caracteres de la información esperada y la cantidad de caracteres de la información obtenida
  Result: “EXTRA INFO”
- Si la información obtenida está contenida en la información esperada:
  Accuracy: se calcula como la relación entre la cantidad de caracteres de la información obtenida
  y la cantidad de caracteres de la información esperada
  Result: “MISSING INFO”
- Si la información no se encontró:
  Accuracy: es igual a 0.0
  Result: “NOT FOUND”
- Si la información es errónea, o sea se obtuvo algo pero no es lo que se esperaba:
  Accuracy: es igual a 0.0
  Result: “WRONG INFO”

Ejemplo:

```
<TestResult timeStamp="1100356298063" accuracy="0.8509804" result="NOT BAD">
  - <element name="pelicula" found="YES" accuracy="0.8509804" result="NOT BAD">
      <attribute name="título" expected="harry potter" found="harry potter" accuracy="1.0" result="OK" />
      <attribute name="país" expected="ee.uu" found="ee.uu de américa" accuracy="0.3125" result="EXTRA INFO" />
      <attribute name="actores" expected="D. radcliffe" found="radcliffe" accuracy="0.75" result="MISSING INFO" />
      <attribute name="año" expected="2004" found="" accuracy="0.0" result="NOT FOUND" />
      <attribute name="idioma" expected="" found="español" accuracy="0.0" result="WRONG INFO" />
    </element>
</TestResult>
```

De esta forma obtenemos indicadores para cada atributo de información solicitada, además podemos realizar un promedio para el elemento. Y finalmente calcular en otro archivo XML un resumen de cual fue el resultado para varias páginas (en caso de haber muchos elementos se promedia el accuracy).

6.3.2. Resultados obtenidos

La siguiente gráfica muestra la evolución de la métrica utilizada para evaluar la calidad de la salida del exactor durante el desarrollo.
Las mediciones comenzaron a ser realizadas cuando se contaba con un prototipo estable que daba buenos resultados para el caso de prueba de películas. Luego continuamos realizando pruebas sobre el dominio de actores, intentando mejorarlo. Pasamos a continuación al caso de prueba de hospitales, donde mejoramos los resultados sin perder de vista los dos casos anteriores. Finalmente, realizamos pruebas sobre el caso de médicos, el cual luego de empezar a observar mejoras decidimos no continuar ya que era necesario enriquecer la ontología. En los anexos "Casos de prueba: Películas" y "Casos de prueba: Actores" se detalla el comportamiento del sistema para cada página al final del desarrollo. Además, en el anexo "Resultados del clasificador" se presentan los resultados obtenidos en la utilización del clasificador que reutilizamos.

Podemos observar en la Figura 25, que en general una mejora en un caso de prueba mejora o mantiene constante la situación de los demás.

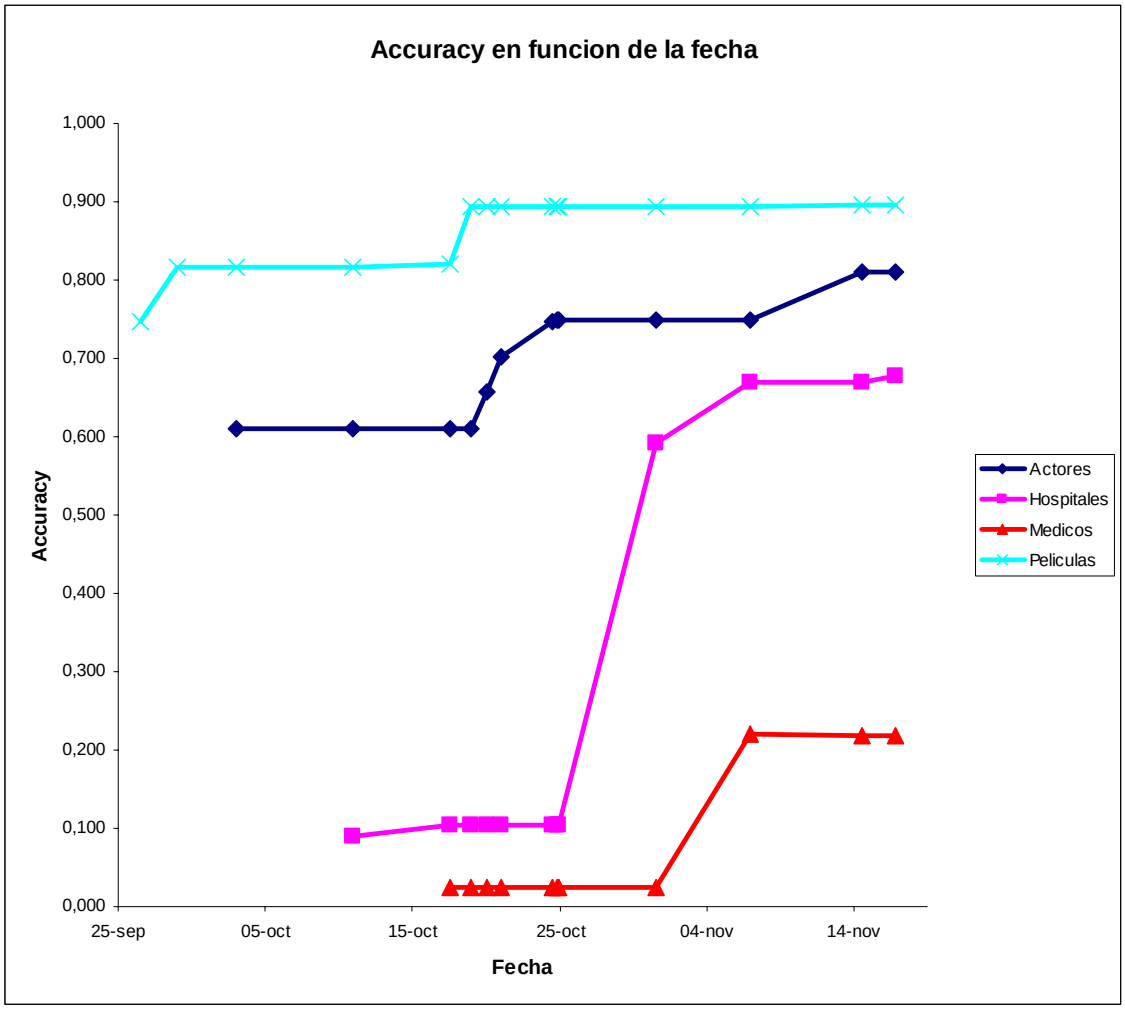


Figura 25 – Gráfico de precisión en función del avance en el desarrollo

A continuación, en la Figura 26 se muestra una gráfica del tiempo de ejecución en los distintos casos de prueba, en función de las mejoras que se hicieron al programa, el tiempo de ejecución aumentó para cada caso.

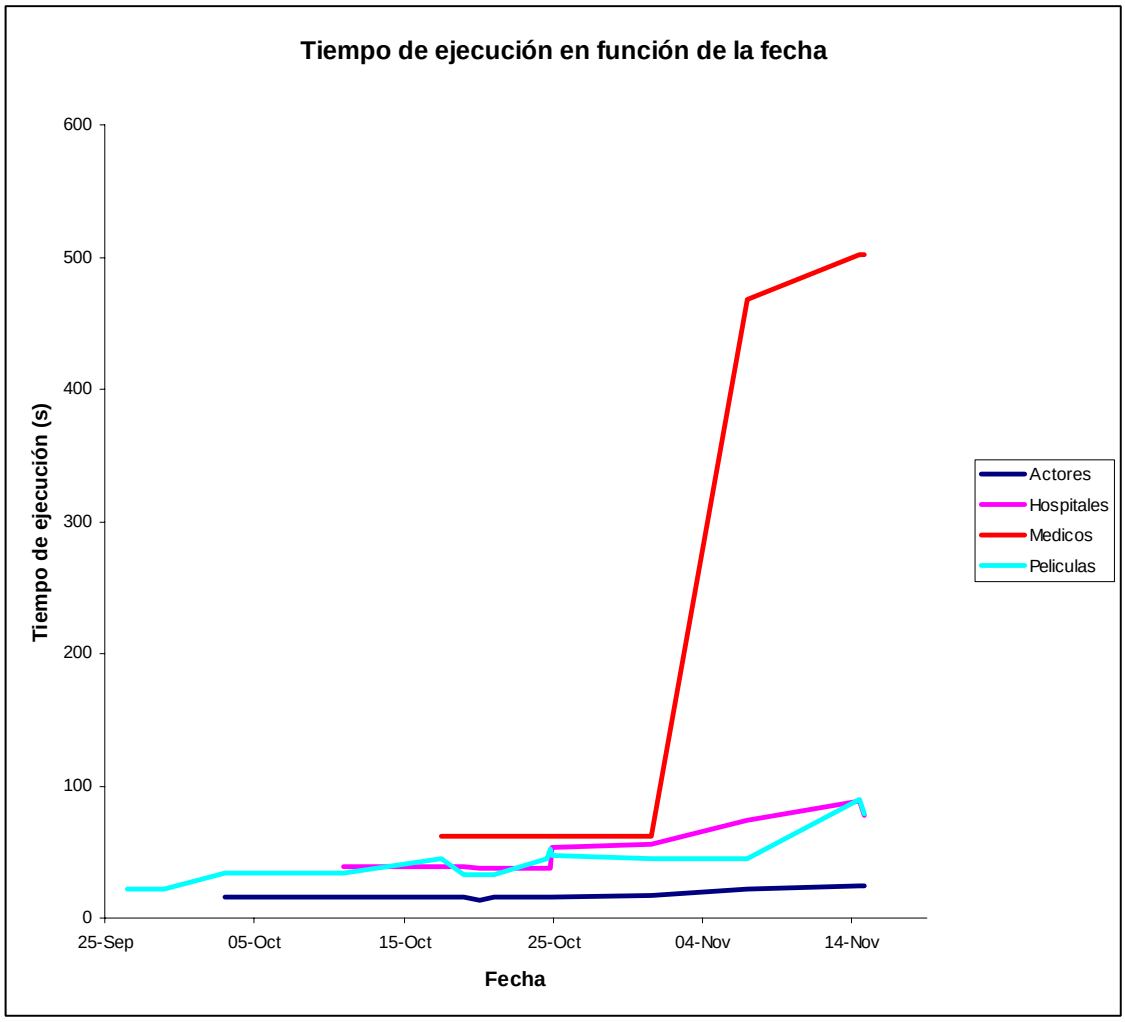


Figura 26 – Gráfico de tiempo de ejecución en función del avance en el desarrollo

Por otra parte, podemos observar que existen diferencias entre los tiempos de ejecución de los distintos casos, esto lo podemos atribuir a la influencia de diversos factores, entre los que se destacan:

- Cantidad de información en las páginas.
- Cantidad de información solicitada a través del esquema.
- Cantidad de información contenida en la ontología, en particular cantidad de sinónimos e instancias de la información solicitada en el esquema.
- Estructura en que se encuentra la información de las páginas, esto influye ya que las diferentes reglas que se aplican tienen diferentes tiempos de ejecución.

Dada la complejidad del análisis no se estudió el peso de cada uno de los factores en el tiempo de ejecución.

A modo de ejemplo, podemos suponer que la mayor demora del caso de prueba de médicos se debe a que contiene más cantidad de información (más páginas y con grandes listados de médicos) conjuntamente, la regla que más aplica es la que tiene mayor tiempo de ejecución.

Finalmente en la Figura 27 vemos los resultados obtenidos para los dominios de Películas, Actores y Hospitales.
El 53% de las páginas presentó estructuras que permitieron extraer la mayoría de la información requerida (75% a 100%), 40% de las páginas presentaba parte de la información de la forma esperada por el programa y otra parte presentada en otras formas, por tanto pudimos obtener entre 50% y 75% de la información requerida, 3% de las páginas contenía la información en formas similares a las comprendidas por el programa, por tanto, se pudo obtener poca información de ellas (entre 25% y 50%). Finalmente, en 5% de las páginas la información se encontraba en formas que el programa no soporta (como imágenes o texto sin estructura).
En conclusión, en más de un 90% de las páginas se encontró más del 50% de la información esperada.

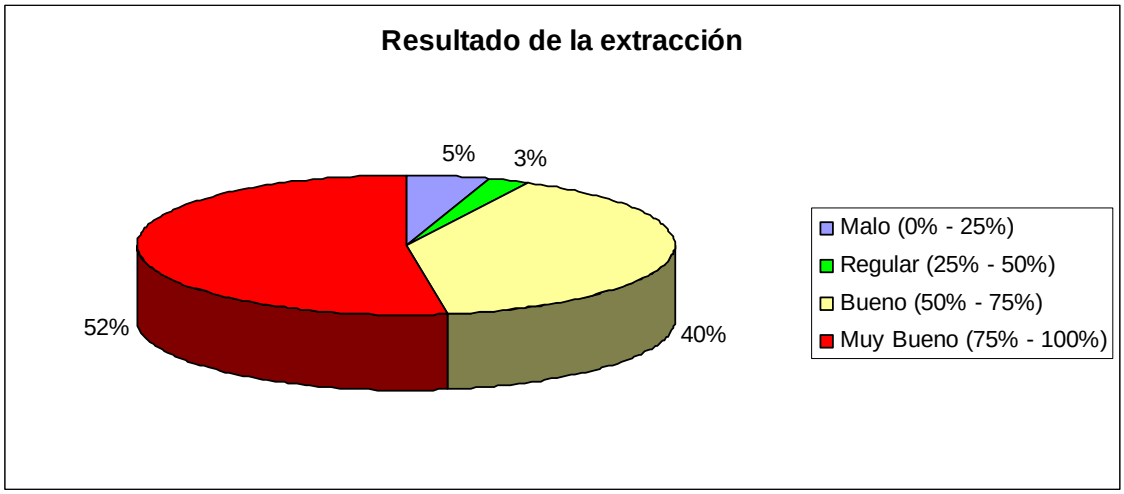


Figura 27 – Resultado de la extracción

Capítulo 7. Conclusiones

7.1. Conclusiones

El proyecto consistió del estudio de tecnologías y la implementación de un prototipo de la primer parte de la arquitectura de Web Warehouse propuesta en [1], centrándose en la extracción e integración información de la Web. Además, se analizó el impacto de cambios en las fuentes de información sobre la información obtenida, generando metadata para el soporte a la evolución.
La arquitectura de Web Warehouse es un enfoque que permitiría solucionar el problema de extracción e integración obtenida de la Web, así como el manejo de su evolución. Posibilitando así consultas sobre esa información que apoye a la toma de decisiones.
La solución propone:

- Mecanismos de extracción de información de páginas Web, abordado desde la perspectiva de la estructura de las páginas HTML, de forma de poder obtener la mayor cantidad de información posible dada su presentación en la página.
- Mecanismos de integración a nivel de datos de la información extraída. El problema de integración es atacado a través de un criterio que permite elegir el mejor valor, entre los valores provistos por diferentes fuentes.

Entre los aportes de la solución se destaca:

- El diseño de los módulos de extracción e integración de la arquitectura de Web Warehouse.
- La propuesta de un conjunto de reglas de extracción de información (basada en una serie de páginas Web representativas) y la utilización de XPath como lenguaje para expresar la ubicación de la información en la reglas de extracción.
- La propuesta de un mecanismo de integración basado en una clave y la confiabilidad de las fuentes.
- La generación de metadata que permite extensibilidad (por ejemplo crear nuevas reglas) y soporte para la evolución.

También, pudimos poner a prueba un mecanismo de clasificación implementado en el proyecto [5], lo cual permitió comprobar su portabilidad así como facilitar nuestro trabajo.

El proceso de extracción realizado mediante reglas que extraen información basándose en la estructura de las páginas Web, permitió obtener buenos resultados para el conjunto de páginas que se utilizó y posibilitó la integración posterior. Por otro lado, existen muchas formas de presentar información en Internet y nuevas formas de mostrar la información requerirían de nuevas reglas, algunas de las cuales podrían requerir ser muy específicas.

El proceso de integración resultó bueno para los casos planteados, aunque para que sea de utilidad al usuario se debe realizar un estudio de las fuentes a utilizar, para brindar al sistema la confiabilidad adecuada de cada una. Por otro lado, debe tenerse en cuenta que toda la información extraída de la página de mayor confiabilidad se encontrará en el resultado, con lo cual si la página cuenta con la mayoría de la información solicitada, se dependerá de ella (como fuente principal), lo cual podría ser un inconveniente si la página desaparece.

La metodología utilizada dio énfasis a la búsqueda de la calidad del desarrollo, en busca de una futura utilización en el proyecto global. Se destaca, el uso de una estrategia de verificación de aumento de casos de prueba junto con aumento de funcionalidades, realizándose testing automatizado.
De esta experiencia debemos destacar que las reglas de extracción, se crearon en base al primer dominio seleccionado, correspondiente al caso de prueba de aceptación planteado. Al agregar el segundo dominio éstas fueron parcialmente modificadas, pero al aplicarlas a los demás dominios ya no sufrieron mayores modificaciones.
La planificación se realizó al inicio del proyecto y la utilizamos como guía durante todo el desarrollo, la metodología de trabajo utilizada junto con el cronograma seguido puede consultarse en el anexo "Metodología de trabajo". Ajustamos el cronograma inicial, asignando más tiempo a la extracción de información, ya que al reutilizar el clasificador esa etapa insumió menos tiempo del planificado. Pudimos optimizar los tiempos de reuniones del grupo, planificando todas las semanas y dividiéndonos el trabajo.
El tiempo insumido de desarrollo lo estimamos en 20 horas durante 33 semanas por persona, lo que a pesar de una pequeña desviación respecto a las 15 horas semanales estipuladas como carga horaria de un Proyecto Grado, nos parece adecuado.

La finalización del proyecto nos permite analizar en retrospectiva los aspectos positivos pero también algunos aspectos a mejorar en futuros trabajos, entre los que se destacan los siguientes:

- La cantidad de documentación que realizamos en paralelo con el desarrollo del proyecto fue muy superior a lo que habíamos realizado en proyectos anteriores. Sin embargo, faltaron algunos documentos que tuvimos que terminar mientras se escribía el informe final.
- El seudo-código de los algoritmos de extracción fue refinado luego de su implementación con lo cual quedó de muy bajo nivel, dificultando su expresión y comprensión en lenguaje natural.
- El código no fue comentado mientras se implementaba, lo que llevó a realizar esta tarea hacia el final, por lo que los comentarios pueden no contener toda la información que se podría haber colocado si se hubiera hecho en el momento del desarrollo.

El análisis y prototipo realizado permiten demostrar la factibilidad técnica de la implementación de los módulos iniciales de la arquitectura de Web Warehouse de [1], abriendo camino a futuros desarrollos teóricos y prácticos que permitan la implementación de la solución completa.

7.2. Trabajo futuro

Como trabajo futuro, se propone:

- Aplicar el sistema a mayor cantidad de dominios o casos, para aportar mayor generalidad y en caso de ser necesario agregar nuevas reglas de extracción. Se decidió utilizar un mecanismo de extracción basado en la estructura de las páginas Web, el cual podría complementarse con un enfoque de reconocimiento de lenguaje natural, para obtener información de forma más precisa. Por ejemplo: obtener párrafos de la estructura de las páginas y luego procesarlos reconociendo frases con lenguaje natural y diferenciando los diferentes tipos de datos.
- Implementar un mecanismo que permita la inclusión de nuevas instancias en la ontología, alimentándola con nueva información. En el dominio de las

películas del cine podría utilizarse una página Web de base para enriquecerla con títulos de nuevas películas.

- Crear nuevos criterios de resolución de conflictos en la integración de datos, utilizando el mecanismo de extensión proporcionado por la herramienta. Por ejemplo: un criterio que devuelva como información resultante la que propone la mayoría de las fuentes (o sea, si la mayoría de las fuentes proponen un mismo dato para cierto atributo devolver ese como resultado).
- Implementar un mecanismo automático de actualización de la información obtenida como resultado, según los cambios que ocurran en las fuentes. Se planteó el análisis del manejo de la evolución de las páginas Web y su impacto en el sistema, soportado por la metadata generada con ese objetivo.
- Continuar el desarrollo de la arquitectura hasta llegar al Data Warehouse y de esta forma completar el sistema de Web Warehouse.

Referencias

[1] A. Marotta, R. Motz, R. Ruggia.Managing Source Schema Evolution in Web Warehouses. International Workshop on Information Integration on the Web (WIIW '2001).
Rio de Janeiro, Brazil. 2001

[2] Mark Levene and Alexandra Poulovassilis.
Web Dynamics.
Department of Computer Science and Information Systems Birkbeck College, University of London. January 2001

[3] Tim Berners-Lee
Realising the Full Potential of the Web Based on a talk presented at the W3C meeting, London, 1997/12/3

[4] Marcia Porteiro, Rodolfo Paiva
"Técnicas de estado finito para la extracción automática de metadata en la Web". Proyecto de Grado
Instituto de Computación - Facultad de Ingeniería - Universidad de la República 2003

[5] Álvaro Fernández.
Tesis de maestría: Un Generador de Instancias de Wrappers
Instituto de computación – Facultad de Ingeniería – Universidad de la República, Montevideo Uruguay, 2004.

[6] Laura Linzo, Marcela Pardo, Luis Rodríguez
"Web Classification Wrappers"
Universidad de la República Facultad de Ingeniería Instituto de computación
Marzo 2002

[7] Diego Vallespir
Tesis de maestría (en curso): Actividades para mejorar la calidad
Instituto de computación – Facultad de Ingeniería – Universidad de la República, Montevideo Uruguay
http://www.fing.edu.uy/~dvallesp/Tesis/webActividades/index.htm
Ultima visita: 21 de Noviembre de 2004

[8] HTML TIDY http://www.w3.org/People/Raggett/tidy/
Ultima visita: 21 de Noviembre de 2004

[9] Ontologies: http://www.daml.org/ontologies/keyword.html
Ultima visita: 21 de Noviembre de 2004

[10] Ontology Working Group: http://mged.sourceforge.net/ontologies/index.php
Ultima visita: 21 de Noviembre de 2004

[11] FOLDC: http://www.si.umich.edu/UMDL/glossary.html
Ultima visita: 04 de Diciembre de 2004

[12] Tom Gruber http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
(definición de ontología)
Ultima visita: 21 de Noviembre de 2004

[13] SemanticWorld: http://www.semanticworld.org/index.php
Ultima visita: 21 de Noviembre de 2004

[14] SchemaWeb: http://www.schemaweb.info/default.aspx
Ultima visita: 21 de Noviembre de 2004

[15] OWL Web Ontology Language Reference: http://www.w3.org/TR/owl-ref/
Ultima visita: 21 de Noviembre de 2004

[16] RDF Tutorial: http://www.w3schools.com/rdf/default.asp
Ultima visita: 21 de Noviembre de 2004

[17] XML Namespaces: http://www.w3schools.com/xml/xml_namespaces.asp
Ultima visita: 21 de Noviembre de 2004

[18] ASCII Table: http://www.asciitable.com/
Ultima visita: 21 de Noviembre de 2004

[19] OWL Web Ontology Language Overview: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1
Ultima visita: 21 de Noviembre de 2004

[20] Protegé: http://protege.stanford.edu/plugins/owl/owl-library/index.html
Ultima visita: 21 de Noviembre de 2004

[21] WordNet: http://www.cogsci.princeton.edu/~wn/
Ultima visita: 21 de Noviembre de 2004

[22] XPath: http://www.w3.org/TR/xpath
Ultima visita: 21 de Noviembre de 2004

[23] Tutorial Xpath:
http://www.zvon.org/xxl/XPathTutorial/General_spa/examples.html
Ultima visita: 21 de Noviembre de 2004

[24] Extensible Markup Language (XML): http://www.w3.org/XML/
Ultima visita: 21 de Noviembre de 2004

[25] Java Sun: http://java.sun.com/
Ultima visita: 21 de Noviembre de 2004

[26] Jena – A Semantic Web Framework for Java: http://jena.sourceforge.net/
Ultima visita: 21 de Noviembre de 2004

[27] Deborah L. McGuinness, Richard Fikes, Lynn Andrea Stein, James Hendler DAML-ONT: An Ontology Language for the Semantic Web. En Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editores. Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press, 2002

[28] Stefan Decker, Frank van Harmelen, Jeen Broekstra Michael Erdmann, Dieter Fensel, Ian Horrocks , Michel Klein, Sergey Melnik
The Semantic Web - on the respective Roles of XML and RDF, 2000, IEEE Internet Computing. http://www.ontoknowledge.org/oil/downl/

Utima visita: 14 de diciembre de 2004

[29] Regina Motz Adriana Marotta.
Transparencias del curso de Interoperabilidad: 4-integracion-esquemas.pdf
5-integracion-instancias.pdf
2004

[30] Regina Motz
Sistemas de Información Interoperables, curso de Interoperabilidad
Instituto de Computación, Universidad de la República. Uruguay 2004

[31] Andrea do Carmo
Tesis de Maestría "Aplicando Integración de Esquemas en un Contexto DW-Web", Facultad de Ingeniería, Universidad de la República, Uruguay. Febrero del 2000

[32] Pablo Castells y Francisco Saiz
Web Ontology Language (OWL)
Escuela Politécnica Superior Universidad Autónoma de Madrid
www.ii.uam.es/~castells/docencia/ semanticweb/apuntes/4-ontologias.pdf
Ultima visita: 21 de noviembre de 2004

[33] Lydia Silva Muñoz
Tutorial Básico de OWL DL http://www.fing.edu.uy/~lsilva/OWLBasico.html
Instituto de computación, Facultad de Ingeniería, Universidad de la República, Uruguay

[34] Yannis Papakonstantinou, Serge Abiteboul, Hector Garcia-Molina
Object Fusion in Mediator Systems, 1996
Proceedings of the Twenty-second International Conference on Very Large Databases

[35] Hector Garcia Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, Jennifer Widom
Integrating and Accessing Heterogeneous Information Sources in TSIMMIS
In Proceedings of the AAAI Symposium on Information Gathering,Stanford, California, March 1995.

[36] Ljiljana Stojanovic, Nenad Stojanovic, Siegfried Handschuh
Evolution of the Metadata in the Ontology-based Knowledge Management Systems
Research Center for Information Technologies at the University of Karlsruhe, Germany 2Institute AIFB - University of Karlsruhe, Germany