REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

MÉTODOS Y TÉCNICAS PARA EL CÁLCULO Y VISUALIZACIÓN DE MÉTRICAS DE HISTORIALES EN ARTÍCULOS DE WIKIS

TRABAJO ESPECIAL DE GRADO PARA OPTAR AL TÍTULO DE LICENCIADO EN COMPUTACION

PRESENTADO POR: D'APUZZO ROBERTO Y WORWA BENJAMÍN.

TUTOR: EUGENIO SCALISE.

CARACAS, MAYO 2012

RESUMEN

Este trabajo especial de grado titulado, “MÉTODOS Y TÉCNICAS PARA EL CÁLCULO Y VISUALIZACIÓN DE MÉTRICAS DE HISTORIALES EN ARTÍCULOS DE WIKIS”, trata acerca de la implementación de una aplicación cuyo objetivo es la recuperación del historial de artículos hospedado en cualquier wiki basado en MediaWiki (Web Scraper), para posteriormente almacenarlos, calcular métricas en base a los datos obtenidos del historial y proyectar dichas métricas a través de herramientas gráfica para la Web.

Dentro de este trabajo se puede identificar la parte teórica-practica que concierne al desarrollo de la aplicación, se utiliza el método ágil Extreme Programming (XP) así como las distintas fases que se deben cumplir para la realización de dicha aplicación.

Luego, se describe la estrategia de solución, la cual se consideró como cada una de las iteraciones correspondientes al desarrollo de la aplicación. Para este desarrollo se consideró dividir la aplicación en 2 partes, un back-end (de escritorio) y un front-end (web), comenzando con el back-end (de escritorio) y la creación de un Web Scraper de uso general con políticas de selección, re-visita, cortesía y paralización bien definidas y finalizando con el front-end Web y la visualización de las métricas obtenidas por el Web Scraper.

Finalmente se dan a conocer las limitaciones, conclusiones y recomendaciones para mejoras de dicha aplicación en cuanto a tiempos de respuestas, promoviendo el escalado horizontal de la base de datos.

Palabras clave: ingeniería Web, Web Crawling, Web Scraping, wiki, historial de wiki, visualización Web, métricas.

INTRODUCCIÓN

En la actualidad, con el crecimiento exponencial que ha tenido el uso de internet y la aparición del concepto de la Web 2.0, han surgido sitios Web que permiten a los usuarios crear y modificar, de manera colaborativa, su contenido. El ejemplo más exitoso y popular de este tipo de sitios Web colaborativos son los wikis. Un wiki puede definirse como un repositorio de artículos colaborativos los cuales pueden ser creados y editados por usuarios pertenecientes a la comunidad de dicho wiki.

Los wikis mantienen un registro de los cambios o ediciones que ha sufrido un artículo a través del tiempo, entre los cuales se detallan diversos atributos como: los usuarios que le han editado, las fechas y el tamaño de las ediciones, entre otros. A pesar que dicha información es útil, muchos de los wikis no aprovechan las propiedades presentes en la información. Por ejemplo, se puede tener una lista con los cambios realizados a un artículo, pero no existe manera de saber, a simple vista, qué usuario ha contribuido más en un artículo dado o incluso en qué fecha un artículo ha sufrido más ediciones. De igual manera, no se cuenta con visualizaciones gráficas que pueden resultar de gran utilidad a la hora de consultar y contrastar cambios variantes en el tiempo.

Es por esto que se propone la creación de una herramienta automatizada, que recupere la información presente en el historial de los artículos de un wiki de manera periódica, para el posterior cálculo de métricas que luego serán visualizadas de manera gráfica.

En el primer capítulo de este trabajo se mencionan algunos de los métodos ágiles para el desarrollo de software, así como una breve descripción de éstos. Se detalla sobre los Web Crawlers como mecanismos para la recuperación de información de Internet, así como las distintas técnicas para realizar Web Scraping (extraer de manera automatizada información de la Web). Aunque el tema se aborda de manera general, se hace énfasis en el Web Scraping del historial de Wikipedia.

En este capítulo también se discute el uso de bases de datos relacionales y bases de datos NoSQL para el procesamiento de grandes cantidades de datos distribuidos (replicados o fragmentados) en diversos servidores de bases de datos. De igual manera, se mencionan y describen brevemente los manejadores de bases de datos NoSQL MongoDB y CouchDB. Por último, se describen algunas herramientas, basadas en HTML5, para la visualización Web de gráficas.

En el segundo capítulo se identifican el problema y los objetivos del trabajo, se esboza una estrategia de solución (herramientas a desarrollar y tecnologías a utilizar) para alcanzar estos objetivos y se expone la documentación de las soluciones desarrolladas.

Finalmente, se presentan las conclusiones y recomendaciones.

CAPÍTULO I: MARCO TEÓRICO

En este capítulo se mencionan y describen brevemente los distintos métodos ágiles para el desarrollo de software, las distintas técnicas de recuperación de información de Internet, las bases de datos NoSQL y las ventajas de éstas sobre las bases de datos relacionales. Por último, se mencionan algunas tecnologías para la visualización Web de gráficas basadas en HTML5.

1.1. Métodos Ágiles para el desarrollo de software

No sólo con el uso de notaciones de modelado, varias herramientas, disponibilidad de tiempo e interacción grupal se logra desarrollar un software novedoso y exitoso, ya que es importante considerar que hay distintas maneras de hacer las cosas, por lo que para desarrollar un software exitoso también es necesario tomar en cuenta las diferentes metodologías y sus respectivos métodos de desarrollo. Cada metodología tiene una estructura de desarrollo, lo que puede o no repercutir en tiempo, costo, y otros factores durante el desarrollo de un software.

En todo método de desarrollo de software se establecen planificaciones; roles; actividades, las cuales que pueden ser grupales o no; modelado; documentación de lo que se desea desarrollar; entre otros. La metodología tradicional o “pesada”, a pesar de ser utilizada y efectiva en proyectos de gran envergadura, no suele ser la metodología más adecuada actualmente ya que los entornos de los sistemas varían constantemente. La metodología ligera o ágil tiene diferentes métodos que buscan reducir los tiempos de desarrollo manteniendo o incluso mejorando la calidad, estos métodos son orientados principalmente para desarrollo de proyectos pequeños.

Los métodos ágiles siguen una serie de principios expresados en el “Manifiesto Ágil” (1) publicado en febrero de 2001. Estos principios son:

- Satisfacer al cliente mediante la entrega temprana y continua de software con valor, es decir, software usable.
- Aceptar cambios, incluso en etapas tardías del desarrollo.
- La entrega de software funcional debe ser frecuente, semanas en vez de meses.
- Los responsables de negocio y los desarrolladores deben trabajar juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que proveerles el ambiente y el apoyo que necesiten, así como confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcional es la medida principal de progreso.
- Los métodos ágiles promueven el desarrollo sostenible. Los integrantes del grupo de trabajo deben ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

1.1.1. Métodos Ágiles

Ágil se refiere a la capacidad para proveer respuestas rápidas y adaptables al cambio. En el entorno de negocios de hoy en día, estas dos cualidades son indispensables, lo que conlleva a que el software también sea desarrollado de manera ágil.

Las necesidades de un cliente pueden variar desde el momento de contratación de un proyecto hasta el momento de su entrega. Esto requiere métodos de desarrollo de software diferentes, que en lugar de rechazar los cambios, sean capaces de adaptarse e incorporarlos. Estos métodos también deben:

- Producir versiones ejecutables en un lapso corto de tiempo con la finalidad de obtener por parte del cliente retroalimentación en cuanto al producto.
- Realizar soluciones sencillas con la finalidad de que el impacto de los cambios que puedan surgir se minimice.
- Mejorar la calidad del diseño para así lograr que las iteraciones posteriores requieran menos esfuerzos.
- Ejecutar pruebas continuas y desde temprano, para encontrar y solventar defectos antes que tengan un gran impacto.

En los últimos años han existido métodos que fomentan las prácticas antes mencionadas, estos son conocidos como “métodos agiles”.

Según Jim Highsmith y Alistair Cockburn, lo nuevo de estos métodos no son las prácticas que utilizan, sino su reconocimiento de la gente como principal factor de éxito de los proyectos. Adicionalmente, Cockburn define estos métodos como el uso de reglas “ligeras pero suficientes” y la aplicación de técnicas orientadas a las personas y su comunicación (2).

A continuación se listan algunos de los métodos ágiles más conocidos y aplicados durante el desarrollo de software hoy en día.

Adaptive Software Development (ASD)

Fue desarrollado a comienzos del año 1990 por James Highsmith y Sam Bayer con la intención de que el método se adapte al cambio en lugar de luchar contra él. Se basa en la adaptación continua a circunstancias cambiantes, por lo que es tolerante a los cambios. ASD sustituyó al tradicional proceso de diseño secuencial en cascada. En este método no existe un ciclo de planificación, diseño y construcción del software sino más bien un ciclo de especular, colaborar y aprender (3).

Agile Unified Process (AUP)

Su desarrollo fue propuesto y liderado por Scott Ambler, es un método de desarrollo de software que está basado en el Rational Process (RUP) de IBM. El ciclo de vida de AUP es serial en lo grande e iterativo en lo pequeño, con la finalidad de generar entregables incrementales en el tiempo (4). AUP tiene varias disciplinas y fases, las disciplinas son ejecutadas de forma iterativa mientras que las fases son implementadas de forma serial a lo largo de un proyecto dentro del cual el equipo de desarrollo define y ejecuta actividades para

construir, validar y liberar un software funcional, el cual debe cumplir con las necesidades de los involucrados en su desarrollo.

Feature-driven development (FDD)

Es un método ágil iterativo e incremental, adaptativo y orientado por rasgos (features). A diferencia de otros métodos ágiles no cubre todo el ciclo de vida sino sólo las fases de diseño y construcción (6). Se complementa con otros métodos ya que no precisa de un modelo específico, se enfoca en calidad, y entregas constantes funcionales y formas de evaluación del proyecto. Su ciclo de vida está compuesto por cinco procesos, desarrollar el modelo general, construir la lista de rasgos, planear por rasgos, diseñar por rasgos y construir por rasgos. Se rige por los siguientes principios: es mejor trabajar un proceso simple y bien definido, los pasos de un proceso deben ser lógicos, vanagloriarse del proceso puede impedir el trabajo real.

Extreme Programming (XP)

Fue desarrollada en el año 1999 por Kent Beck (7) y es uno de los métodos ágiles de desarrollo de software más destacados. Se diferencia de los métodos tradicionales en su enfoque hacia la adaptabilidad y previsión ante los cambios. Se considera que los cambios suelen ser frecuentes y aceptados durante el desarrollo de proyectos ya que por naturaleza es muy difícil especificar todos los requisitos al comienzo de un proyecto.

Se podría considerar XP como uno de los mejores métodos para el desarrollo de proyectos, ya que aparte de lo antes mencionado está centrada en potenciar las relaciones interpersonales promoviendo el trabajo en equipo y el éxito del mismo, preocupándose por el aprendizaje del personal y proporcionando un entorno cómodo para trabajo. XP se basa principalmente en la interacción entre el cliente y el equipo de desarrollo, simplicidad en las soluciones, comunicación constante y fluida entre equipos, y coraje para enfrentar cambios. XP suele ser muy útil para proyectos con requisitos imprecisos y muy variantes.

Con XP se realizan mejoras constantemente ya que el desarrollo es iterativo e incremental, también se hacen pruebas unitarias, continuas, repetitivas y frecuentemente automatizadas.

Scrum

Es un proceso iterativo e incremental para la administración de proyectos enfocado en la administración de proyectos de desarrollo de software (10).

Durante cada sprint, ciclo del proceso que dura por lo general entre dos y cuatro semanas, el equipo de desarrollo crea un producto probado y funcional. Durante este período de tiempo se mantiene un sprint backlog, el cual contiene un conjunto de tareas de alta prioridad a realizarse, las cuales son determinadas en una reunión previa a cada sprint. Durante esta reunión el cliente le indica al equipo de desarrollo qué tareas deberían incluirse en el sprint backlog para el próximo sprint, el equipo analiza la viabilidad (tiempo, carga de trabajo, etc.) de la petición y agrega la tarea al sprint backlog, concretando así los requerimientos para dicho sprint.

Todos los días durante el sprint se lleva a cabo una reunión de 15 minutos en promedio, en la cual se discuten puntos de planificación, objetivos logrados y objetivos por cumplir.

1.2. Recuperación de Información de Internet/Web

A continuación se mencionan y describen las distintas técnicas de recuperación de información de Internet.

1.2.1. Web Crawlers

Un Web Crawler es un programa automatizado el cual se encarga de visitar de manera sistematizada un conjunto de URLs con el fin de obtener información y/o realizar actividades de rutina o mantenimiento (12).

Este proceso es llamado Web Crawling y es usado por muchos sitios, por lo general motores de búsqueda, para proveer información actualizada siempre que un usuario lo requiera. Los Web Crawlers, como se mencionó anteriormente, tienen diversas funciones dependiendo del uso que se les quiera dar, una de ellas y la más importante, es tomar el código HTML de un documento (X)HTML de la Web con el fin de procesar esta información y recolectar datos importantes para ser procesados en pasos posteriores. Los Web Crawlers también pueden ser implementados para realizar actividades de rutina o mantenimiento como por ejemplo:

- Automatizar las tareas de mantenimiento en los sitios web.
- Comprobar enlaces o validar código (X)HTML.
- Recabar tipos de información específica de las páginas web.
- Realizar un estudio estadístico sobre el uso de un determinado término en página.
- Crear el índice de una máquina de búsqueda.
- Analizar enlaces de una página con la finalidad de verificar que los links apunten a páginas activas.
- Recolectar información específica como correos, precios, artículos, entre otros.

Un Web Crawler podría considerarse un robot cuya tarea se describe a continuación:

- Se tiene una lista de URLs iniciales llamadas semillas o pool de semillas. No necesariamente tienen que ser varias, por lo general y en la primera corrida de un Web Crawler se tiene una única URL, en corridas posteriores el pool de semillas puede incrementar o decrementar.
- El Web Crawler, siguiendo una serie de políticas de acceso, accede a cada una de las URL del pool de semillas. Adicionalmente, y dependiendo del comportamiento del Web Crawler, éste puede almacenar las URLs encontradas dentro del documento (X)HTML asociado a una URL e ir agregando dichas URLs a la frontera de rastreo. Este paso es opcional ya que si el Web Crawler tiene como objetivo URLs específicas y no necesita descubrir URLs adicionales, o simplemente no está trabajando con documentos (X)HTML sino con documentos en texto plano o XML, la frontera de rastreo será igual al pool de semillas.

- Se accede a cada URL de la frontera de rastreo y se realiza la tarea pertinente (realizar y almacenar una copia del (X)HTML del documento (X)HTML actual, verificar el estado de los enlaces dentro de un documento (X)HTML, etc.).

1.2.1.1. Políticas

El comportamiento de un Web Crawler podría resumirse como una combinación de las siguientes políticas:

- selección.
- re-visita.
- cortesía.
- paralelización.

Políticas de selección

Dado el gran tamaño de la Web y el hecho de que un Web Crawler siempre descarga una pequeña cantidad de páginas en un momento dado, es altamente deseado que dicha información sea relevante y no aleatoria o de poca importancia.

Por lo tanto, se requiere de métricas de importancia para priorizar la selección de las páginas, o documentos Web a visitar. La importancia de una página Web es un concepto abstracto que podría verse afectado por características difusas como la cantidad de visitas que recibe la página en un período de tiempo determinado o la cantidad de enlaces entrantes que ésta posea. Un intento de representar dichas características en un valor ponderable es el caso del PageRank de Google Inc., medida que toma en cuenta la relevancia, frescura, enlaces entrantes y velocidad de carga de una página, o documento, Web (12) .

La importancia de una página también podría verse representada por la demanda del usuario. En este caso el Web Crawler accedería a las URLs exigidas por el usuario en un momento dado, con un orden predeterminado o aleatorio, sin necesidad de preocuparse por priorización alguna.

Dentro de las políticas de selección existen una serie de técnicas que permiten no solo priorizar las URLs a visitar sino agilizar el proceso de selección de URLs. Estas son: Rastreo Concentrado, restricción de enlaces a seguir y normalización de URLs. A continuación se explica en detalle cada una de estas técnicas.

a) Rastreo Concentrado

La importancia de un documento Web también podría ser expresada en función a la similitud que éste tenga con otro documento Web previamente rastreado. La acción de rastrear páginas similares se conoce como Rastreo Concentrado. Los conceptos y tópicos acerca de esta técnica fueron introducidos por Menczer (13) (14) y por Chakrabarti (15) en los años 1997 y 1999.

El principal problema que presenta esta técnica radica en el hecho de tener que descargar de antemano un documento Web para verificar su similitud con otro, cuando lo ideal sería poder predecir la similitud, mediante el uso de elementos o atributos disponibles en los documentos Web, sin necesidad de descargar dicha página, la cual puede tener similitud o no.

Una posible solución presentada por Pinkerton (16) en 1994 fue la de predecir la similitud de un documento Web con otro en base a los enlaces, locales y externos, que esta tuviera. De manera tal que si un documento Web que trata sobre un tema dado enlaza en una única o múltiples ocasiones a otro documento Web, es probable que dicho documento Web contenga información de interés y puede que presente similitudes con el documento Web original.

Sin embargo, para que esto realmente funcione los desarrolladores Web deberían hacer uso completo y correcto de los atributos semánticos de los enlaces dentro de los documentos (X)HTML, como por ejemplo los atributos alt y title, en los cuales se describa de manera concreta la naturaleza y relación de la página enlazada.

b) Restricción de enlaces a seguir

La mayoría de los Web Crawlers que buscan información se enfocan mayoritariamente en el código (X)HTML de los documentos (X)HTML para extraer datos de interés y generar información en pasos posteriores. Una manera de asegurarse que el recurso que se está visitando va a devolver código (X)HTML sería realizar una petición HEAD al servidor para determinar el tipo MIME del recurso antes de realizar la respectiva petición GET.

Una petición GET es un método a través del cual un cliente puede solicitar a un servidor el contenido completo del recurso solicitado identificado por una URI. La petición GET implica una petición HEAD.

La petición HEAD es una petición de control que realiza un cliente a un servidor antes de realizar una petición GET. En esta petición se especifican distintos campos que le permiten al cliente decidir si utilizar una versión almacenada en caché o descargar el recurso por completo, ver el tipo MIME del recurso, a donde se han movido los recursos actualmente no disponibles, etc. Una petición HEAD no implica una petición GET.

Un ejemplo de una petición HEAD es el siguiente:

```
http://example.com --> 302 Found
http://www.iana.org/domains/example/ --> 200 OK
Connection: close
Date: Tue, 10 May 2011 21:31:26 GMT
Server: Apache/2.2.3 (CentOS)
Content-Length: 2945
Content-Type: text/html; charset=UTF-8
Last-Modified: Wed, 09 Feb 2011 17:13:15 GMT
Client-Date: Tue, 10 May 2011 21:31:25 GMT
```

```
Client-Peer: 192.0.32.8:80
Client-Response-Num: 1
```

Cuando el campo Content-Type es encontrado en la cabecera HTTP a la hora de solicitar un recurso, éste le indica al cliente qué hacer con el recurso. Por ejemplo, si el servidor especifica en la cabera HTTP el tipo MIME de un documento como text/html el cliente debería entender que el recurso es una página web posiblemente rastreable, mientras que un recurso con un tipo MIME image/jpeg es una imagen JPEG y debería ignorarse.

Como podemos observar las ventajas de realizar una petición HEAD antes de realizar una petición GET son múltiples ya que en ella podemos detallar:

- Las redirecciones que presenta la URL solicitada.
- El código de respuesta de estado HTTP de la URL solicitada.
- El valor hash MD5 del recurso solicitado (opcional).
- La fecha del servidor en la que se está accediendo a la URL.
- El tipo MIME y la codificación de caracteres del recurso solicitado.
- El lenguaje del recurso solicitado (opcional).
- El tamaño del recurso en bytes.
- La última fecha del servidor en la que el recurso fue modificado, entre otros.

Sin embargo, una de las ventajas más importante de realizar una petición HEAD antes de realizar una petición GET es el la velocidad y el tamaño de la respuesta por parte del servidor como podemos observar en la figura 2.1 y figura 2.2.

Figura 2.1: Tiempo y tamaño de respuesta al realizar una petición HEAD.

En la primera figura podemos observar que el tiempo de respuesta por parte del servidor al realizar una petición HEAD es menor que 0.33 segundos, mientras que en la segunda figura el tiempo de respuesta de la petición GET es más del triple que la anterior, 1.048 segundos.

Lo mismo ocurre con el tamaño de la respuesta, en la primera figura se obtienen menos de 706 bytes, mientras que en la segunda figura tenemos un tamaño de respuesta de 65.5 KB.

Figura 2.2: Tiempo y tamaño de respuesta al realizar una petición GET.

c) Normalización de URLs

La normalización de URLs permite minimizar el número posible de formas en que una URL es presentada al Web Crawler, de esta manera el Web Crawler no tendrá la necesidad de rastrear una misma página varias veces. El objetivo principal de la normalización de URLs es la de verificar si dos URL sintácticamente diferentes son equivalentes.

Existen varios tipos de normalizaciones, aquellas que preservan la semántica y las que no.

Entre las normalizaciones que preservan la semántica, descritas en el RFC 3986 (17), se tienen:

- Colocar en minúsculas el protocolo y el host del URL.
- Capitalizar las letras precedidas por un %.
- Decodificar los octetos, codificados con porcentajes, de caracteres no reservados.
- Agregar / al final del host y sub directorios.
- Remover el puerto 80 por defecto del URL.
- Remover los segmentos de puntos en el path de la URL.

Entre las normalizaciones que no preservan la semántica se tienen:

- Remover la referencia al archivo index por defecto.
- Remover los fragmentos (#fragmento) de la URL.
- Cambiar direcciones IP por nombres de dominio.
- Limitar los protocolos únicamente a HTTP.
- Remover “/” duplicados.
- Remover o agregar el prefijo de dominio WWW.
- Ordenar las variables del query string alfabéticamente.
- Remover las variables del query string que sean innecesarias.
- Remover el “?” si el query string es nulo.
- Estandarizar la codificación de caracteres.

Existen normalizaciones que son dependientes del sitio que se esté rastreando. Por ejemplo, si observamos el siguiente fragmento de la respuesta HEAD de la solicitud realizada a la siguiente URL:

```
>http://wikipedia.org/wiki/mount_sentinel

http://wikipedia.org/wiki/mount_Sentinel --> 301 Moved Permanently
http://www.wikipedia.org/wiki/mount_Sentinel --> 301 Moved Permanently
http://en.wikipedia.org/wiki/mount_Sentinel --> 301 Moved Permanently
http://en.wikipedia.org/wiki/Mount_Sentinel --> 200 OK
```

Podemos observar cuatro cosas importantes:

1. Wikipedia realiza una redirección de todas aquellas solicitudes de URLs sin prefijo de dominio WWW a una equivalente con prefijo de dominio WWW.
2. Se realiza una redirección de todas aquella solicitudes con prefijo de dominio WWW a una equivalente con un prefijo de dominio que describa el idioma seleccionado por el usuario en el navegador, reemplazando el prefijo WWW por EN en este caso.
3. Si la letra inicial del path de la URL no se encuentra en mayúscula, se realiza una redirección a una URL en la que la primera letra del path sea mayúscula.
4. Todas estas redirecciones son permanentes.

El overhead provocado por estos tres redireccionamientos podría evitarse simplemente normalizando la URL http://wikipedia.org/wiki/mount_sentinel al formato específico de Wikipedia, http://en.wikipedia.org/wiki/Mount_sentinel.

Los siguientes son resultados de dos pruebas realizadas haciendo una petición GET a la URL (figura 2.3):

```
http://wikipedia.org/wiki/mount_Sentinel
```

Y a la URL (figura 2.4):

```
http://en.wikipedia.org/wiki/Mount_Sentinel
```

En la figura 2.3 se observa que el tiempo de respuesta, por parte de Wikipedia, a la hora de acceder a la URL no normalizada de acuerdo a la estructura de URLs de Wikipedia es de 2.033 segundos, mientras que el tiempo de respuesta con una URL normalizada de acuerdo a la estructura de URLs de Wikipedia es de 1.133 segundos.

Se tiene un overhead de 0.900 segundos, casi un segundo, lo cual se traduce en 250 horas cuando hablamos de 1.000.000 de URLs mal estructuradas.

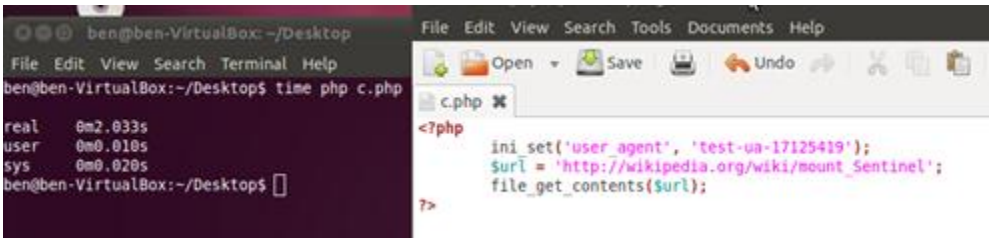

Figura 2.3: Tiempo de respuesta al realizar una petición GET con una URL no normalizada de acuerdo a la estructura de URLs de Wikipedia.

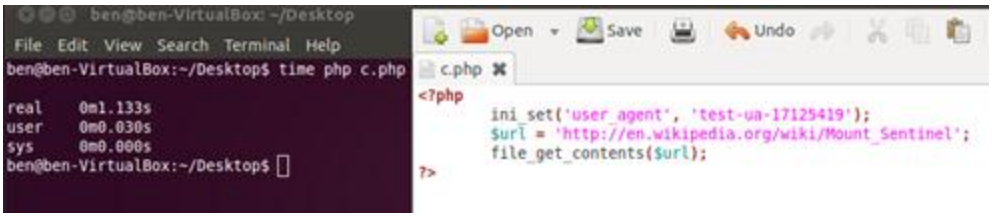

Figura 2.4: Tiempo de respuesta al realizar una petición GET con una URL normalizada de acuerdo a la estructura de URLs de Wikipedia.

Políticas de re-visita

Dada la naturaleza dinámica de la Web, no basta con solo definir políticas de selección y visitar una página en una única ocasión. Muchas de las veces, encontraremos escenarios en donde las páginas que hemos visitado han sido actualizadas días, horas e incluso segundos después de nuestra primera visita, por lo que los datos recolectados podrían ser datos obsoletos mucho antes de haber sido procesados.

Esto genera un nuevo problema, si se tiene una frontera de rastreo de 1.000.000 de URLs, es muy probable que para el momento en el que se rastree la mitad de ellas (500.000) la información de la primera URL ya sea obsoleta y necesite volver a rastrearse. Sin políticas de re-visita claras se corre el riesgo de tener en un bucle infinito de rastreo.

Existen básicamente tres maneras de evitar este problema:

1. Usando la información de caducidad de un recurso suministrada por los sitemaps del dominio, siempre y cuando estén disponibles.
2. Usando la información del campo Last-Modified de la respuesta HEAD de la URL en cuestión.
3. Visitar todas las URL de la frontera de rastreo y luego repetir el proceso.

a) Sitemaps

Los sitemaps son archivos XML o de texto plano que describen cada una de las URLs que contiene un sitio Web. Su principal uso es el de informar a los Web Crawlers aquellas páginas disponibles para ser rastreadas.

El protocolo de los sitemaps se cita textualmente (18) a continuación:

- Comenzar con una etiqueta de apertura <urlset> y terminar con una de cierre </urlset>.
- Especificar el espacio de nombres (protocolo estándar) en la etiqueta urlset.
- Incluir una entrada <url> para cada dirección URL como una etiqueta XML principal.
- Incluir una entrada secundaria <loc> para cada etiqueta principal <url>.
- Las demás etiquetas son opcionales.
- Todas las URLs deben proceder de un único host.

Adicionalmente se emplea el uso de atributos opcionales para cada URL que son de nuestro interés, éstos son:

- <lastmod>, similar al campo Last-Modified de las cabeceras HTTP. Utiliza un formato AAAA-MM-DD.
- <changefreq>, indica la frecuencia con la que cambia una URL en el tiempo, puede tomar los valores: always, hourly, daily, weekly, monthly, yearly, never. El valor always es utilizado para aquellas páginas que cambian en cada acceso (dinámicas), el valor never es utilizado para aquellas páginas que nunca cambian.
- <priority>, indica la prioridad de la URL. Su valor es un float con rango desde 0.0 hasta 1.0, siendo 0.5 el valor por defecto.

Estos últimos atributos son de suma importancia a la hora de establecer una política de re-visita correcta y efectiva. Una página con un atributo <changefreq>never</changefreq> no tendría por qué re-visitarse posteriormente a su primer rastreo, de igual manera una página que cambie cada mes o cada año tampoco tendría por qué re-visitarse cada semana o incluso cada día.

También se podrían filtrar páginas por el atributo <lastmod>, ya que si una página ha sido visitada con anterioridad y dicha fecha ha sido almacenada en un repositorio de datos, no se tendría por qué volver a visitar una página cuyo atributo <lastmod> sea menor a dicha fecha.

Cabe destacar que el sitemap no es de uso obligatorio y su existencia en un servidor es opcional. Incluso si el sitemap estuviese presente, como se mencionó anteriormente, los atributos que realmente nos interesan son opcionales.

b) Last-Modified

Este atributo de la cabecera HTTP ya ha sido mencionado anteriormente y uso es similar a aquel definido por el atributo <lastmod> del punto anterior. Sin embargo este campo tiene la ventaja de estar presente en un mayor número de ocasiones que el atributo <lastmod> de un sitemap o incluso que un mismo sitemap.

c) Re-visitar todas las URLs

Esta política es básicamente la última opción, siempre y cuando las dos opciones anteriores no se encuentren disponibles. En este caso se rastrean primero todas las URLs de la frontera de rastreo y una vez que se finalice este proceso y luego de un tiempo predeterminado se inicia el proceso de re-visita.

Este último proceso vuelve a visitar todas las URLs indistintamente de su prioridad o de su frecuencia de cambio utilizando una cola FIFO.

Políticas de cortesía

Los Web Crawlers son programas automatizados que se encargan de visitar un conjunto de URLs de uno o varios sitios Web en específico. Debido a este hecho, la cantidad

de peticiones que puede generar un Web Crawler supera con creces a aquellas peticiones que podría realizar una persona manualmente, esta gran cantidad de peticiones en un corto período de tiempo puede tener un impacto negativo en el desempeño del sitio Web que se esté rastreado.

Si un Web Crawler realiza 200 peticiones por segundo a un servidor que puede manejar alrededor de 1.500 peticiones por segundo, esto tendría dos implicaciones importantes:

1. En un momento dado, 200 usuarios no tendrán acceso al servidor y se les negará el servicio.
2. Si tuviésemos diez Web Crawlers, en lugar de uno, rastreando la página en un momento dado, 1500 usuarios no tendrían acceso al servidor y se les negaría el servicio.

Cabe destacar que desde el punto de vista de los Administradores de Sistemas, un Web Crawler es un intruso y éste no representa ganancia alguna como lo haría un usuario regular. Debido a esto sería lógico pensar que un Web Crawler no debería utilizar más recursos por segundo que la cantidad de recursos que un usuario regular utilizaría (19).

Koster (20) menciona que el uso de los Web Crawlers es útil para un gran número de tareas, sin embargo presentan diversas desventajas como:

- Alto consumo de recursos de red del servidor, debido al nivel de paralelismo que los Web Crawlers pueden llegar a manejar y la velocidad con la que pueden realizar peticiones a un servidor.
- Recarga del servidor, ligada a la desventaja anterior y como se mencionó al principio de este punto, la frecuencia de acceso a un servidor puede derivar en un DoS sin intención.

Uno de los puntos más importantes en la política de cortesía es la frecuencia de acceso al servidor. Cho (19) propone un tiempo entre peticiones de 10 segundos, lo que equivaldría a 6 accesos por minuto, lo equivalente a la velocidad humana.

Otro punto importante sería tomar en consideración el uso del archivo robots.txt siempre que se encuentre disponible. Este archivo indica qué directorios y recursos del servidor son rastreables y cuáles no, restringe el acceso total o parcial al servidor por parte de los Web Crawlers e indica, de manera opcional, la ubicación del archivo sitemap.xml mencionado en un punto anterior.

En el archivo robots.txt también se encuentra la directiva Crawl-delay, la cual indica, en segundos, el intervalo de tiempo que el Administrador de Sistemas desea que el(los) Web Crawler(s) utilicen a la hora de rastrear su(s) sitio(s) Web.

A estas alturas se puede definir un conjunto de políticas de cortesía que todo Web Crawler debería cumplir, estas son:

1. De existir el archivo robots.txt, verificar la directiva Crawl-delay y realizar peticiones al servidor con dicho intervalo de tiempo.
2. De no existir el archivo robots.txt, realizar peticiones al servidor con un intervalo de tiempo de 10 segundos entre peticiones. Evitar a toda costa el bombardeo de peticiones que puede derivar en un DoS sin intención.
3. De existir el archivo robots.txt, verificar si tenemos acceso al servidor.
4. De existir el archivo robots.txt, verificar si tenemos acceso al directorio en el que se encuentra el recurso. De esta manera podríamos filtrar aquellas URLs de nuestra frontera de rastreo que se encuentren en un directorio inaccesible.
5. De existir el archivo robots.txt, verificar si tenemos acceso al recurso.
6. No realizar más de 6 peticiones por minuto sin importar el grado de paralelismo. Un Web Crawler debe representar a un único usuario, no a dos, ni a cien.

Políticas de paralelización

Un Web Crawler puede ser single-threading o multi-threading. Sin embargo, dadas las políticas de cortesía anteriormente mencionadas ¿valdría la pena tener un Web Crawler multi-threading si vamos a realizar una petición al servidor cada 10 segundos? Sin duda alguna la descarga de una página tardaría mucho menos de 10 segundos, sin embargo, se podrían utilizar 6 threads en el que cada uno:

1. Realice la petición al servidor.
2. Procese el documento (X)HTML recibido (Web Scraping).
3. Almacene el resultado del paso anterior.
4. Espere el tiempo necesario hasta que se cumplan los 10 segundos desde el paso número 1.

Esto aseguraría dos cosas importantes:

1. Un thread siempre tendrá el 100% del ancho de banda en un momento dado (siempre que no hayan procesos externos que se encuentren utilizando la red y asumiendo que el tiempo de descarga de una página siempre será menor a 10 segundos).
2. Si un thread tarda más de 10 segundos en completar un ciclo de trabajo (pasos del 1 al 3), los demás threads no se verán afectado por esta demora.

1.2.1.2. Arquitectura de un Web Crawler

La arquitectura de un Web Crawler es bastante simple y podría graficarse como se ve en la figura 2.5.

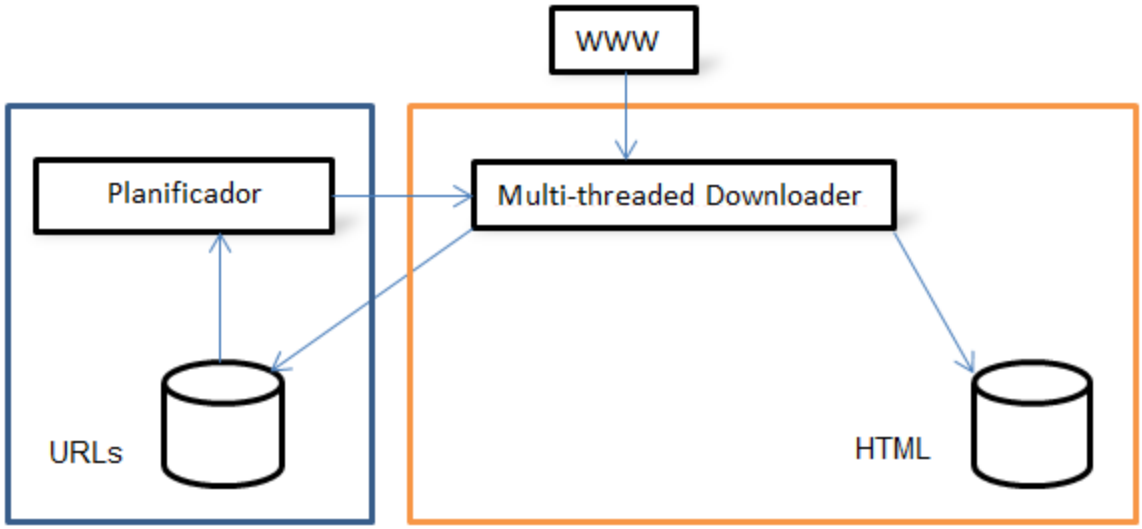

Figura 2.5: Arquitectura de un Web Crawler multi-threaded

El planificador toma las URLs aún no visitadas del repositorio de URLs y se crea el pool de semillas. Luego el planificador dispara una cantidad específica de threads con URLs a visitar del pool de semillas y las marca como visitadas.

Cada thread del downloader se encarga de visitar una URL provista por el planificador, descargar el documento (X)HTML o recurso, procesar el código (X)HTML obtenido extrayendo los enlaces y datos de interés y almacenándolos en sus respectivos repositorios. Pareciera mucho trabajo para el downloader, sin embargo debemos recordar que cada thread cuenta con 10 segundos de ejecución, tiempo más que suficiente para completar dicho conjunto de tareas. Además, si asumimos, como mencionamos anteriormente, el uso de 6 threads, cada thread tendría un tiempo de ejecución de 60 segundos antes de volver a empezar su ciclo de vida.

Este proceso se repite de manera indefinida ya que cada URL tiene un tiempo de vida y aunque se hayan visitado todas las URLs existentes en un servidor, todavía se tendría que volver a re-visitar las URLs para actualizar su contenido.

1.2.2. Web Scraping

Web Scraping es una técnica utilizada para extraer información de sitios Web (22). La principal idea del Web Scraping es tomar contenido no estructurado disponible en la Web y convertirlo en contenido estructurado que sirva como fuente de datos para que otras herramientas puedan generar información.

Hoy en día muchos sitios Web ofrecen estos tipos de datos, comúnmente obtenidos a través del Web Scraping, a través de Servicios Web. Sin embargo esto no es una práctica común y por lo general la cantidad de información que puede obtener un usuario navegando una página Web supera a la cantidad de información que podría obtener utilizando dichos

servicios. Adicionalmente algunos sitios pequeños o no comerciales podrían contener información relevante y valiosa pero no tener los recursos o tecnologías para prestar dichos servicios.

1.2.2.1. Técnicas

Existen diversas técnicas para extraer de manera automatizada información de la Web. A continuación se describen algunas de ellas.

Programación HTTP

El contenido completo de una página Web puede ser obtenido realizando una petición HTTP al servidor que la contenga. Para esto se pueden utilizar diversos métodos nativos de los lenguajes de programación actuales. En C, por ejemplo, se utilizarían sockets mientras que en lenguajes con un mayor nivel de abstracción como Python simplemente utilizaríamos la función urlopen de la librería nativa urllib para extraer el HTML del recurso solicitado.

Expresiones regulares

Muchas veces no se necesita extraer el contenido completo de una página Web, por lo que combinando expresiones regulares junto con el método anterior se podrían extraer secciones de nuestro interés que coincidan con un patrón predefinido, obviando aquellas partes irrelevantes.

Analizadores sintácticos de (X)HTML

Un analizador sintáctico, a diferencia de las expresiones regulares, está en la capacidad de analizar un documento (X)HTML y representarlo mediante una estructura de árbol similar al DOM. Uno de los grandes problemas que presenta el uso de expresiones regulares para analizar sintácticamente un documento (X)HTML es la ambigüedad y la dependencia a la estructura de una página en particular.

Por ejemplo, si tenemos el siguiente elemento <h1 class=”title”>Título</h1> y se utilizan expresiones regulares para extraer el título de esta página, se tendrían que buscar todas las etiquetas <h1.*class=”title”.*>, pero ¿qué pasaría si el desarrollador de dicha página decide que quiere cambiar el nombre de la clase a “titulo” en vez de “title”? Se tendría que cambiar la expresión regular siempre que los atributos de los elementos y la estructura del código (X)HTML cambien.

Ahora, si utilizamos un analizador sintáctico de (X)HTML y tenemos la siguiente estructura de página <header><img src=”logo.ext”><h1>Titulo</h1></header>, simplemente se puede extraer el primer hijo del nodo <header> que sea <h1>. Por lo tanto, siempre que cambie la clase, el id, se agreguen o eliminen elementos dentro del header y después del elemento <h1> el resultado no debería verse afectado.

Por supuesto, siempre va a existir cierta dependencia entre la estructura del documento (X)HTML y el método que se utilice. No existe una “bala de plata” para solucionar este

problema. Si el desarrollador Web decide que el título ya no será un <h1> sino un o un <p>, o que el título ya no será el primer hijo del nodo <header> sino el segundo, afectaría el resultado y deberían realizarse los cambios necesarios. Sin embargo, es más probable que se cambien atributos de elementos con más frecuencia a la estructura del documento como tal.

Este tema se profundizará en la sección 2.4. Análisis sintáctico de (X)HTML.

1.2.2.2. Limitaciones

Los Web Scrapers simplemente se encargan de extraer información de la Web, la cual está compuesta en su mayoría de documentos (X)HTML. Extraer el código (X)HTML de una página en particular no representa ningún problema, basta con realizar una petición al servidor solicitando el recurso en cuestión y aplicar alguna de las técnicas previamente mencionadas para extraer los segmentos de nuestro interés.

Sin embargo existen páginas cuyo contenido no es devuelto como tal al realizar una petición al servidor, sino es cargado de manera asíncrona posterior a la carga del recurso solicitado. Un Web Scraper no es un Navegador Web, ni pretende serlo dada su sencillez, por lo que este no descargará de manera automática recursos apuntados por las etiquetas <img>, <link>, <script>, etc. Tampoco ejecutará scripts de JavaScript de manera automática, renderizará el código (X)HTML ni analizará sintáctica ni semánticamente el código (X)HTML descargado.

En conclusión, un Web Scraper no descarga contenido de manera asíncrona, por lo que el contenido del documento, bien sea estático o dinámico, debe ser servido de manera síncrona y sin el uso de scripts de JavaScript.

1.2.2.3. Aspectos legales

Es importante mencionar que los Web Scrapers pueden ir en contra de los términos de uso de los sitios en donde se utilicen ya que la reproducción total o parcial de cierta información puede ser ilegal en países como Estados Unidos.

Sin embargo, los términos de servicio de algunos servicios como Wikipedia expresan claramente que cualquier contenido de la enciclopedia, incluido el contenido del historial de los artículos, puede ser utilizado libremente (21).

1.2.3. Web Scraping del historial de Wikipedia

El historial de un artículo en Wikipedia es aquel en donde se ven reflejados todos los cambios que ha sufrido dicho artículo desde su creación. Por lo tanto, todo artículo en la enciclopedia tiene un historial en el que se detallan, por ejemplo, el número de revisiones que ha tenido el artículo, la fecha en la que se ha realizado dicha revisión, el usuario que ha realizado la revisión, etc.

A continuación se describe el historial de un artículo de la Wikipedia en español tomando en cuenta los elementos de interés para este trabajo:

- El título del artículo se puede localizar en la etiqueta h1 con el id firstHeading entre «» o en la etiqueta title del documento HTML.
- Los enlaces a las páginas con las revisiones del artículo poseen la clase mw-numlink y dentro del atributo title poseen el nombre del artículo.
- El listado de las revisiones se encuentra dentro de un formulario con id mw-history-compare en forma de lista desordenada (ul) con el id pagehistory.
- Cada elemento de la lista posee la siguiente estructura:
    a. Una etiqueta span con la clase mw-history-histlinks la cual contiene dos enlaces, act y ant, los cuales detallaremos más adelante.
    b. Una etiqueta (a) al artículo sin comparación de ediciones el cual expresa la fecha y hora de edición del artículo.
    c. Un span con la clase history-user el cual contiene:
        i. Una etiqueta (a) a con la clase mw-userlink el cual contiene el nombre del usuario que realizó la edición como texto del enlace. Si el usuario no es un usuario registrado en vez del nombre se coloca la dirección IP del usuario.
        ii. Un span con la clase mw-usertoollinks el cual contiene un enlace (a) a las contribuciones que ha realizado dicho usuario en diversos artículos.
    d. Una etiqueta opcional abbr cuya presencia indica si la revisión del artículo ha sido menor.
    e. Un span con la clase historysize la cual contiene el tamaño en bytes del artículo luego de la revisión.

En la figura 1.6 podemos observar el código HTML de un elemento de la lista de revisiones.

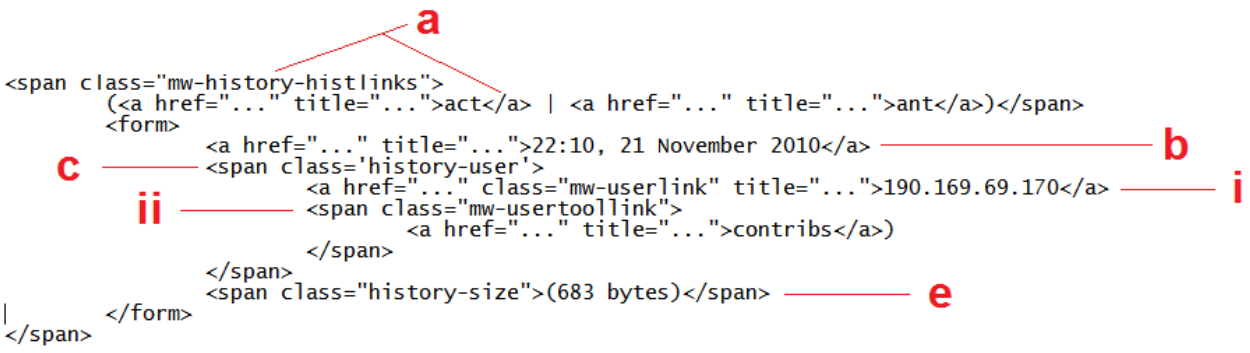

Figura 2.6: Código HTML de un elemento de la lista de revisiones

1.2.3.1. Enlaces ant y act

Como se mencionó anteriormente, los dos enlaces (act y ant) los cuales se encuentran en el span con clase mw-history-histlinks apuntan a dos versiones distintas de un artículo en cuestión. Lo interesante de estos enlaces es que no muestran únicamente el artículo como lo haría el enlace siguiente a estos dos enlaces, muestra además una comparación entre las revisiones realizadas, seguida del artículo como tal.

Ant muestra los cambios realizados con respecto a la versión anterior del artículo. Act por otro lado, muestra la los cambios realizados posteriormente a la revisión, los artículos más recientes no tendrán una versión act.

Por ejemplo, si se tiene el siguiente texto:

“Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas quis arcu vitae quam consectetur fermentum. Quisque et magna quam. Duis nec dui at massa lobortis posuere. Nullam egestas tortor in lorem cursus tristique.”

Y se cambia la primera palabra por Ipsum, quedando:

“Ipsum ipsum dolor sit amet, consectetur adipiscing elit. Maecenas quis arcu vitae quam consectetur fermentum. Quisque et magna quam. Duis nec dui at massa lobortis posuere. Nullam egestas tortor in lorem cursus tristique.”

La versión ant muestra el cambio de la palabra que se acaba de realizar, Lorem por Ipsum. Sin embargo, como nuestra revisión es la más actual no dispondremos de una versión act. Tiempo después alguien se tomará la molestia de corregir lo que hemos alterado y revertir nuestros cambios, quedando el artículo como estaba originalmente. Nuestra revisión ya no será la más actual, el artículo con nuestros cambios revertidos lo será, sin embargo la revisión ahora cuenta con una versión act la cual refleja el cambio de la palabra Ipsum por Lorem.

La estructura comparativa de las versiones anteriormente mencionadas es la siguiente (figura 2.7):

- Se tiene una tabla con clase diff la cual contiene dos columas.
- Ambas columnas están divididas en filas las cuales contienen las secciones editadas en forma de un cuadro comparativo. En estas columnas no se muestra todo el contenido del artículo, solo aquellas secciones las cuales han sido editadas.
- Las secciones editadas se dividen en párrafos.
- La columna de la izquierda muestra la versión anterior a los cambios, mientras que la columna de la derecha muestra la versión luego de los cambios, marcando en negritas aquellas palabras modificadas, agregadas o eliminadas. Las palabras que contengan en común las secciones correspondientes de ambas columnas no se encuentran en negritas.
- Las celdas de la columna de la izquierda (td) tienen la clase diff-deletedline, mientras que las de la derecha tienen la clase diff-addedline.
- Todo aquello que ha sido modificado, agregado o eliminado se encuentra dentro de un span con clase diffchange.

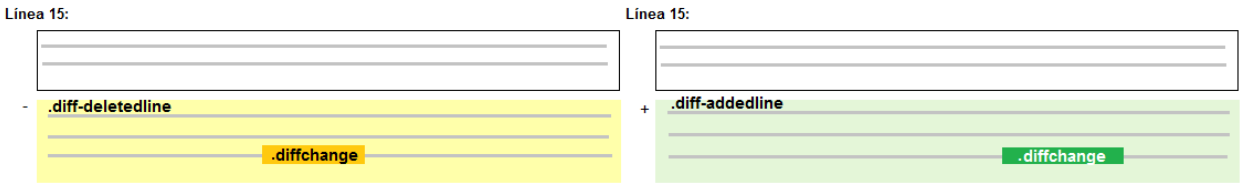

Figura 2.7: Columnas de comparación encontradas en las versiones ant y act.

1.2.3.2 Posibles problemas y soluciones

Actualmente existen dos problemas principales, la obtención de una lista completa de las revisiones de un artículo y el acceso a los historiales de los artículos de manera automatizada utilizando lenguajes de programación.

El primer problema surge debido a que Wikipedia muestra un número fijo de revisiones en el historial, por defecto 50, lo crea la necesidad de extraer los enlaces de las páginas de manera recursiva al momento de realizar el Web Scraping. Este problema se puede solucionar cambiando el valor de la variable limit del query string de la URL del historial a un valor lo suficientemente grande como para ser prácticamente inalcanzable, es decir, que se pueda asegurar que el número de revisiones de dicho un artículo no supere dicho número.

Este método es poco elegante pero presenta dos ventajas importantes:

1. Se realiza una única petición GET al servidor en vez de varias.
2. Se obtienen todas las revisiones en un único documento HTML, simplificando la estructura e implementación del Web Scraper.

Sin embargo y por razones desconocidas, este método sirve únicamente con la versión en español de los artículos. La versión en inglés tiene una cota superior de 5.000 revisiones por página.

Teniendo esto en cuenta y además el hecho de que una petición que obligue al servidor a realizar una consulta a una base de datos que devuelva más de 1.000.000 de tuplas puede causar un error del tipo connection timed-out, la solución más sensata sería cambiar el valor de la variable limit del query string a 5.000 y obtener los enlaces a páginas subsiguientes de manera recursiva al hacer el Web Scraping. De esta manera se obtiene una solución intermedia, no se obtiene la totalidad de las revisiones en una sola petición pero disminuimos la cantidad de páginas y por consiguiente el número de peticiones y el tiempo que esto representa.

El segundo problema surge debido a una restricción de acceso por parte de Wikipedia a programas con una identificación de user-agent no definida o nula como se puede apreciar en la figura 2.8. Este problema puede solucionarse especificando una identificación de user-agent cualquiera para nuestro programa como se ve en la figura 2.9.

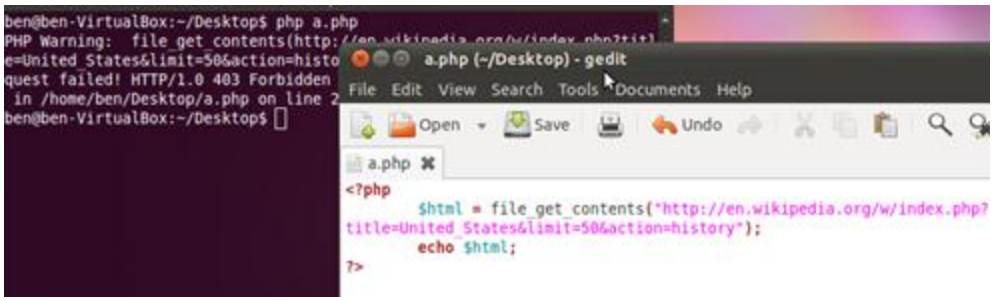

Figura 2.8: Respuesta 403 Forbidden al realizar una petición sin una identificación de user-agent definida.

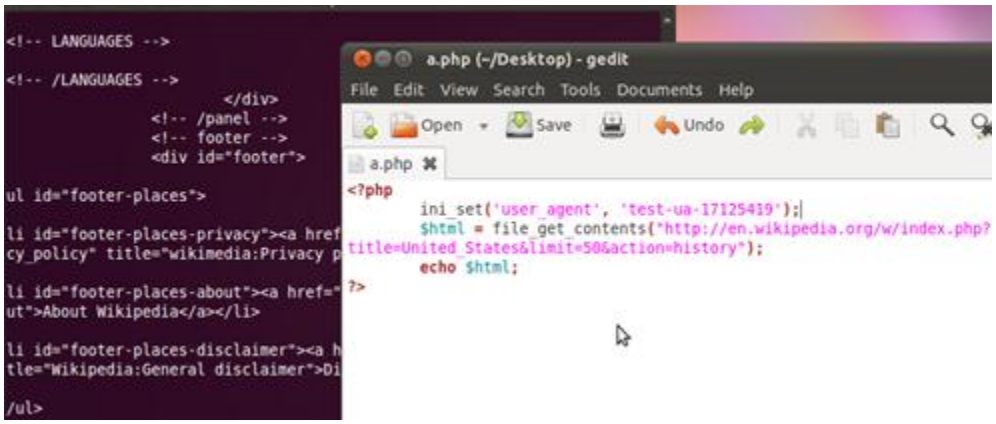

Figura 2.9: Respuesta 200 OK al realizar una petición con una identificación de user-agent definida.

De igual manera existe un tercer problema el cual se considera como secundario, la codificación de caracteres o el charset. Se considera un problema secundario ya que este no afecta la obtención de los datos sino su presentación.

La solución de este problema es relativamente sencilla ya que muchos lenguajes de programación de alto nivel proveen funciones nativas o librerías que se encargan de la detección, conversión y corrección de la codificación de caracteres de una cadena de caracteres, evitando la implementación de las mismas.

1.2.4. Análisis sintáctico de (X)HTML

Una vez obtenido el código (X)HTML de un documento (X)HTML en la Web y luego de este haber sido almacenado en un repositorio de datos, surge la necesidad de manipular dicha información.

El análisis sintáctico de (X)HTML, o parsing, se resume básicamente como el proceso de separar la data relevante de la irrelevante dentro de un documento (X)HTML. Esto es, detectar, extraer y separar imágenes, enlaces y cualquier otra información que sea de interés.

Los analizadores sintácticos de (X)HTML analizan código (X)HTML, el cual está compuesto por una serie de tokens y determina la estructura gramatical del documento con respecto al DTD especificado en la declaración doctype del documento. En base a esto y dada la estructura actual de árbol del DOM (22), un analizador sintáctico de (X)HTML puede tomar un documento (X)HTML, o un fragmento del mismo, y representarlo mediante una estructura de árbol. De esta manera la búsqueda y el acceso a elementos del documento es fácil, segura y resistente, dentro de lo posible, al cambio.

Usualmente el código (X)HTML que se obtiene de la Web no es del todo semántica ni sintácticamente correcto, dada la naturaleza humana de cometer errores. Muchos Navegadores Web hoy en día son capaces de mostrar de manera correcta documentos (X)HTML pobremente escritos, sin embargo, un Web Scraper no es un Navegador Web, ni pretende serlo. Los documentos (X)HTML pobremente escritos pueden causar un mal funcionamiento de los analizadores sintácticos de (X)HTML y generar resultados no deseados o incorrectos. Es por esto que es de suma importancia que todo código (X)HTML que se obtenga de la Web mediante Web Scraping, o cualquier otro método, sea corregido antes de analizarlo.

1.2.4.1. Expresiones Regulares

Las expresiones regulares no son analizadores sintácticos como tal, sin embargo mediante su uso se pueden extraer y separar datos de interés, tal cual lo haría un analizador sintáctico. El uso de expresiones regulares está asociado más que todo al análisis por comparación de patrones definidos por el usuario más que el análisis y estructuración del (X)HTML de forma estándar basados en DTDs.

Sin embargo la construcción de analizador sintáctico de (X)HTML que utilice expresiones regulares para construir un árbol que represente de manera exitosa un documento (X)HTML sería posible, al fin y al cabo los DTDs son conjuntos de patrones que describen la estructura sintáctica de un documento (X)HTML. No obstante, al hacerlo surge un problema inmediato.

Realizar un análisis sintáctico entero de un documento (X)HTML utilizando expresiones regulares es prácticamente imposible ya que éstas solo pueden analizar lenguajes regulares mientras que (X)HTML es un lenguaje libre de contexto, siempre y cuando esté bien escrito, por lo que para analizar sintácticamente un documento (X)HTML se necesitaría una expresión regular infinitamente larga. Es posible tener un documento (X)HTML que cause que una expresión regular genere un resultado completamente erróneo.

Esto no quiere decir que no se puedan utilizar expresiones regulares para extraer información concreta de un documento (X)HTML relativamente simple, sin embargo la obtención de información mediante su uso se basa netamente en heurística, lo que las hace ser una no muy buena opción para el análisis sintáctico de documentos (X)HTML. Simplemente no se puede asegurar al 100% que lo que encontremos dentro de un documento (X)HTML sea lo que se esté buscando.

En la figura 2.10 se puede observar lo antes mencionado. El código PHP mostrado en la figura (a.php) busca extraer todas las etiquetas (X)HTML de un documento dado, en este caso b.html. Como se puede observar en el documento HTML, la primera etiqueta <link> es sintácticamente correcta, sin embargo utilizando expresiones regulares, el resultado obtenido es completamente incorrecto. Lo mismo ocurre con el resto de los casos ya que el código (X)HTML dentro de comentarios o scripts de JavaScript no deben ser catalogados como etiquetas (X)HTML a la hora de analizar sintácticamente un documento (X)HTML.

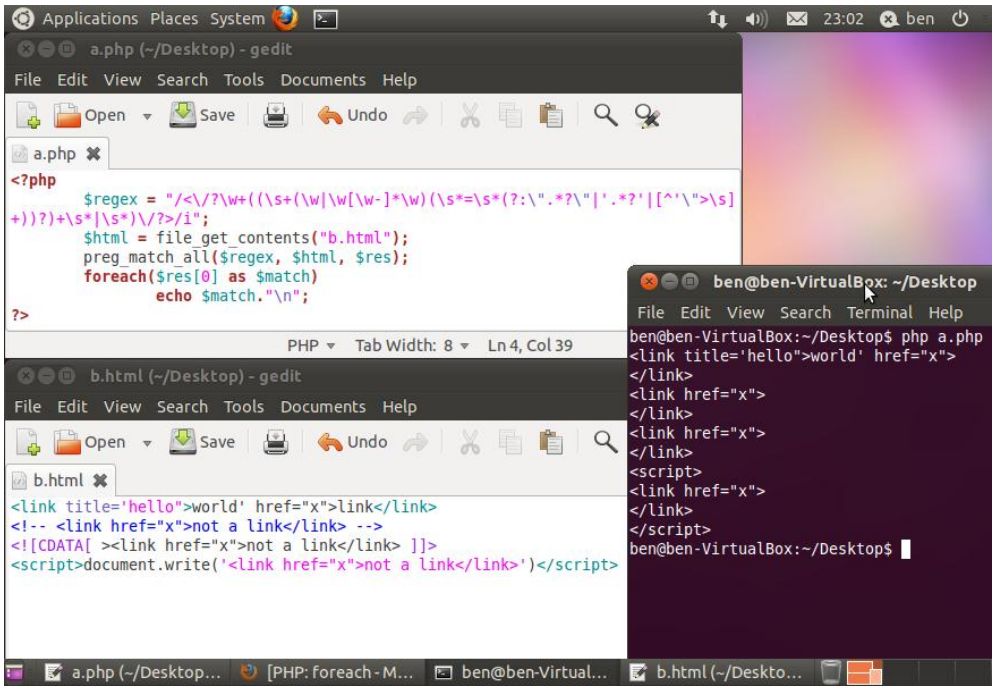

Figura 2.10: Análisis incorrecto de un documento HTML utilizando expresiones regulares.

1.2.4.2. XPath

XPATH es un lenguaje cuyo objetivo principal es el de encontrar y/o seleccionar nodos de un documento XML, representando dicho documento en una estructura de árbol de nodos (23).

Por lo anterior, es posible navegar a través de los nodos de un documento para encontrar información de nuestro interés, de manera sencilla, rápida y resistente al cambio.

XPATH define siete tipos de nodos: elemento, atributo, texto, namespace, instrucciones de proceso, comentario y nodos de documento. Todos los nodos de tipo elemento y atributo tienen un padre. Los demás nodos pueden tener o no padres, hijos, hermanos, ancestros y/o descendientes.

La selección de uno o más nodos de un documento se realiza utilizando una expresión de camino. Las expresiones de camino más útiles se listan a continuación:

- Nombredelnodo, selecciona todos los nodos hijos del nodo elementos identificado con dicho nombre.
- /, selecciona los nodos partiendo desde el nodo raíz.
- //, selecciona los nodos del documento desde el nodo actual.
- ., selecciona el nodo actual.
- .., selecciona el nodo padre del nodo actual.
- @, selecciona atributos.

El siguiente ejemplo pretende clarificar el uso de las expresiones anteriores. Dado el siguiente documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book>
<title lang="eng">Harry Potter</title>
<price>29.99</price>
</book>
<book>
<title lang="eng">Learning XML</title>
<price>39.95</price>
</book>
</bookstore>
```

- bookstore, selecciona los hijos del elemento raíz bookstore.
- /bookstore, selecciona el nodo raíz bookstore.
- bookstore/book, selecciona todos los elementos book que son hijos de bookstore.
- //book, selecciona todos los elementos book sin importar donde se encuentren dentro del documento.
- bookstore//book, selecciona todos los elementos book que sean descendiente de bookstore.
- //@lang, selecciona todos los atributos lang sin importar donde se encuentren.

En XPATH también se pueden usar predicados para encontrar nodos con valores específicos mediante el uso de predicados. Estos predicados se especifican dentro de corchetes dentro de una expresión de camino, algunos de ellos son:

- [1], selecciona el primer elemento encontrado.
- [n], selecciona el n-ésimo elemento encontrado.
- [last()], selecciona el último elemento encontrado.
- [last()-1], selecciona el penúltimo elemento encontrado.
- [position()<3], selecciona los dos primeros elementos.
- [@atributo], selecciona los elementos que contengan dicho atributo.
- [@atributo=”valor”], selecciona los elementos que contengan el atributo con dicho valor.
- Nodo1[nodo2>19.50], selecciona todos los elementos nodo1 cuyo elemento hijo nodo2 tenga un contenido mayor a 19.50.

Existen diversas maneras de seleccionar nodos (24) y diversas funciones que permiten la manipulación de datos, muy extensas para ser listadas en este documento. Sin embargo y teniendo en cuenta la definición de XPath, mencionada al principio de este punto, surge la duda, ¿cómo utilizar XPath para analizar y manipular documentos HTML?

1.2.4.3. HTML Tidy

Para solucionar el problema anterior y el problema de la corrección de códigos (X)HTML pobremente escritos extraídos de la Web existe la herramienta HTML Tidy.

HTML Tidy es una de las utilidades más usadas para reparar documentos (X)HTML y/o CSS pobremente escritos. Fue desarrollado por Dave Ragget del W3C, el cual estuvo involucrado en el desarrollo de HTML2, 3 y 4 (25). HTML tidy está disponible bajo la licencia W3C, una licencia permisiva compatible con la licencia GNU General Public License (26).

Con HTML Tidy se puede no solo generar un documento (X)HTML sintácticamente correcto a partir de uno incorrecto, en la mayoría de los casos, sino que también podemos convertir un documento HTML en un documento XHTML (27), una versión XML de HTML que presenta las mismas funcionalidades pero cumpliendo con las especificaciones sintácticas estrictas de XML (28). De esta manera se puede utilizar XPath con un documento HTML previamente convertido a XHTML y reparado utilizando HTML Tidy.

1.2.4.4. XQuery

XQuery, como lo define la W3C (29), es a XML lo que SQL es a las base de datos. XQuery y XPath comparten el mismo modelo de datos, soportan los mismos operadores y comparten las mismas librerías de funciones.

XQuery utiliza la misma sintaxis de expresiones para hacer referencia a partes específicas de documentos XML que XPath con la adición de expresiones FLWOR, un acrónimo para: FOR, LET, WHERE, ORDER BY, RETURN. FLWOR es vagamente análogo al SELECT, FROM, WHERE de SQL y permite realizar operaciones análogas al Join de SQL sobre documentos XML.

Las cláusulas de FLWOR se descomponen de la siguiente manera (30):

- FOR, asocia a cada elemento retornado por la expresión in una variable. La cláusula for resulta en iteración. Pueden haber múltiples cláusulas for en la misma expresión FLWOR.
- LET, permite asignaciones de variable y evita repetir la misma expresión varias veces. La cláusula let no resulta en iteración.
- WHERE, filtra los elementos en base a una expresión booleana. Esta Cláusula es opcional.
- ORDER BY, ordena los elementos filtrados por la cláusula WHERE por el valor del elemento o atributo especificado. Por defecto el ordenamiento es ascendente y se pueden ordenar elementos por más de un atributo o valor.

- RETURN, es evaluado solo una vez por cada elemento filtrado por la cláusula WHERE y especifica qué es lo que se va a retornar.

El siguiente ejemplo muestra las ventajas de las expresiones FLWOR sobre las expresiones de camino al momento de trabajar con más de un documento XML relacionados:

Si se tienen dos documentos XML: estudiantes.xml y facultad.xml de donde se quiere listar todas las facultades cuyo promedio del promedio de sus estudiantes sean mayor o igual que 14.0 y mostrar el nombre de la facultad, ordenados de manera descendente por promedio de promedios. Sin embargo el archivo estudiantes.xml no contiene el nombre de la facultad a la que pertenece el estudiante sino el ID de la misma. Una expresión FLWOR válida para este problema sería:

```
for $f in doc("facultad.xml")/facultad
let $e := doc("estudiantes.xml")/estudiante[fid = $f/id]
where avg($e/promedio) >= 14.0
order by avg($e/promedio) descending
return
<resultado>
{
    $f/nombre,
    <avg>{avg($e/promedio)}</avg>
}
</resultado>
```

Cuyo resultado sería:

```
<resultado>
      <nombre>Ciencias</nombre>
      <avg>14.20</avg>
</resultado>
<resultado>
      <nombre>Humanidades</nombre>
      <avg>16.30</avg>
</resultado> …
```

Si se quiere resolver el problema anterior utilizando expresiones de camino únicamente, se tendría que primero obtener el conjunto de nodos de las facultades /facultad y el conjunto de nodos de estudiantes /estudiantes y asociar dichos conjuntos con la ayuda de un lenguaje de

programación, para luego calcular el promedio de promedios nuevamente con la ayuda de un lenguaje de programación.

Las expresiones condicionales if-then-else también son válidas en las expresiones FLWOR, por ejemplo:

```
for $f in doc("facultad.xml")/facultad
return
    if($f/nombre=”Ciencias”)
        then $f/nombre
```

Lo que generaría:

```
<nombre>Ciencias</nombre>
```

También se puede apreciar la ventaja del RETURN en FLWOR, ya que nos permite generar salidas estructuradas de acuerdo al formato que se desee, bien sea XML, HTML, JSON, texto plano, etc.

XQuery también permite la definición de funciones por parte de los usuarios. En el siguiente ejemplo se declara la función minPrice, la cual retorna el valor de un libro dado el precio y el descuento del mismo (31):

```
declare function local:minPrice($p as xs:decimal?,$d as xs:decimal?) AS
xs:decimal?
{
    let $disc := ($p * $d) div 100
        return ($p - $disc)
}
```

La cual se invocaría de la siguiente manera:

```
<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>
```

Como se puede observar, la estructura de la declaración es similar a la de otros lenguajes de programación:

- Los tokens declare y function que permiten identificar el inicio de la declaración de una función.
- El nombre de la función.
- Los parámetros de la función separados por coma, especificando explícitamente sus tipos.
- El tipo del retorno.
- El token return seguido de una expresión, el cual finaliza la función y retorna el valor de la expresión la cual debe ser del tipo del retorno especificado anteriormente.

1.3. Almacenamiento de la información recuperada

Luego de haber obtenido los datos de interés, mediante las técnicas vistas en la sección 1.2, se debe tener algún mecanismo que permita asegurar su persistencia. Dada la naturaleza automatizada de los Web Crawlers y Web Scrapers, la cantidad de datos e información recopilada puede llegar a ser demasiado extensa como para ser almacenada en archivos de texto cuya lectura y manejo secuencial es ineficiente a gran escala.

Otra alternativa al problema previamente planteado es el uso de una base de datos (BD), un repositorio centralizado o distribuido de información, el cual permite la manipulación (agregación, remoción y edición) de los datos de manera eficaz, eficiente y relativamente sencilla.

Toda base de datos tiene un sistema manejador de base de datos (SMBD), un software que sirve de intermediario entre los usuarios y la base de datos física. El SMBD se encarga no sólo de ofrecer mecanismos que permita la creación, el mantenimiento y el uso de una base de datos, sino además se encarga de ofrecer un nivel de abstracción a los usuarios que les permita olvidarse de problemas que abundan en los sistemas de bases de datos, como por ejemplo: seguridad, concurrencia, bloqueos, recuperación, replicación, etc.

Al final, el usuario ve a una base de datos como una estructura lógica, independientemente del modelo que se utilice para representar los datos, la cual puede ser manipulada a través de directivas o lenguajes específicos que entiendan dichos SMBD, sin tener que involucrarse con manipulaciones de archivos a bajo nivel y todo lo que esto conllevaría.

1.3.1. Bases de datos relacionales

Las bases de datos relacionales (BDR) nacen en el año 1970, cuando Edgar Codd de IBM describe su proceso de creación (32). Desde entonces, las bases de datos relacionales han crecido en popularidad hasta convertirse en el estándar en el mercado de bases de datos.

Las bases de datos relacionales pueden describirse como una colección de relaciones, usualmente denominadas tablas. Cada tabla contiene una colección de tuplas las cuales comparten los mismos atributos. De esta manera, una tupla representa a una entidad y a la información asociada a dicha entidad y es almacenada en una relación o tabla (33).

La mayoría de los escenarios cotidianos, reales o conceptuales, pueden representarse como relaciones entre objetos. Por ejemplo, si tratásemos de describir la Facultad de Ciencias de la Universidad Central de Venezuela, podríamos hacerlo en base a las relaciones de sus entidades o componentes: estudiantes, profesores, escuelas, etc. y de esta misma manera sería almacenado en una base de datos relacional.

Las aplicaciones o usuarios se comunican con las bases de datos relacionales a través de consultas en lenguaje SQL, interpretados por los sistemas manejadores de bases de datos relacionales (SMBDR).

SQL, Structured Query Language, fue desarrollado en 1970 por Donald Chamberlin y Raymond Boyce de IBM y fue el primer lenguaje comercial de las bases de datos relacionales propuestas por Codd ese mismo año. SQL se ha convertido en el lenguaje estándar de las bases de datos relacionales, sin embargo, los distintos manejadores de bases de datos relacionales que existen actualmente en el mercado utilizan versiones distintas de SQL con leves diferencias en su sintaxis (34).

La manipulación de datos en SQL se realiza a través del DML, Data Manipulation Language, el cual es un subconjunto de SQL utilizado para manipular, agregar, actualizar o eliminar data dentro de las bases de datos relacionales. La definición de los datos se realiza a través del DDL, Data Definition Language, un segundo subconjunto de SQL que permite crear, modificar o eliminar bases de datos y tablas, restricciones, índices, etc (35).

1.3.1.1. Problemas

Aunque las bases de datos relacionales, en conjunto con sus respectivos manejadores de bases de datos relacionales, proveen a los usuarios robustez, simplicidad, flexibilidad y rendimiento, existe un factor que ha se ha convertido en un elemento crítico de las bases de datos hoy en día, este factor es la escalabilidad.

A medida que pasa el tiempo, los costos de los dispositivos de almacenamiento son cada vez más bajos y su capacidad mucho mayor. De igual manera y con el advenimiento de las Web 2.0, en la cual la cantidad de información generada por parte de los usuarios es cada vez mayor, comienzan a surgir problemas de rendimiento con los modelos de bases de datos relacionales a gran escala (36).

Las bases de datos relacionales pasan por un proceso de normalización durante su diseño. La normalización de una base de datos relacional implica un aumento en el número de relaciones o tablas y por lo tanto un aumento en el número de uniones o JOINs que se deben realizar entre las relaciones para recuperar y reconstruir los datos.

Como es sabido, los JOINs son las operaciones más costosas del SQL ya que su uso sobre grandes relaciones puede derivar en un producto cartesiano y a su vez en un alto consumo de CPU y constantes de lecturas de disco, así como en un impacto negativo en el desempeño general del sistema. Lo anterior lo podemos apreciar en la figura 3.1.

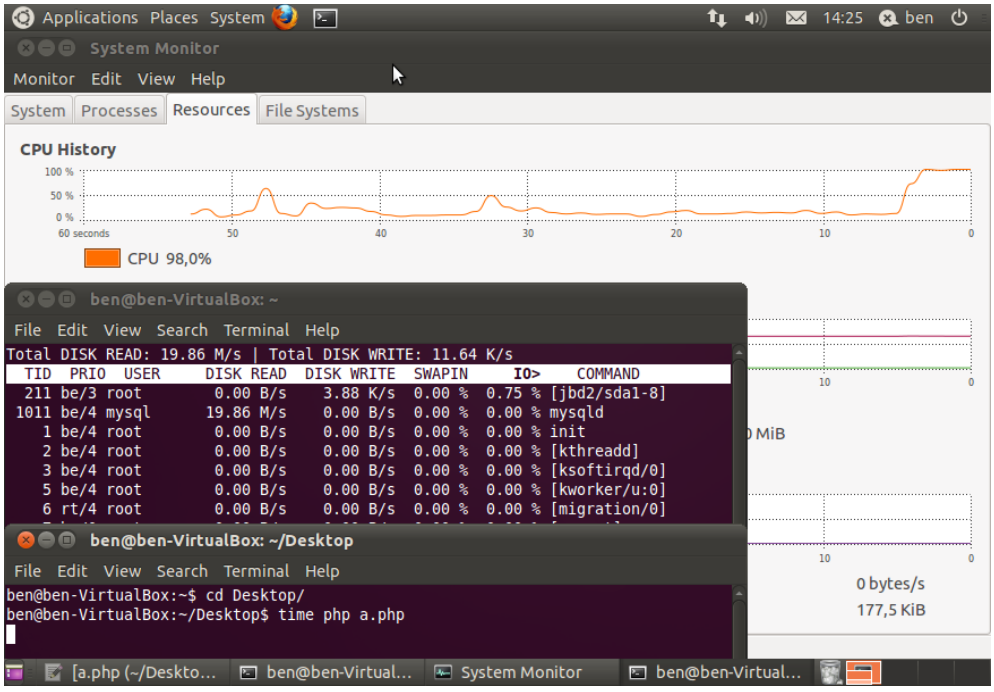

Figura 3.1: Costo del CPU y E/S al realizar un JOIN entre dos tablas de 5.000.000 de registros cada una (12 seg. aprox.).

Si se toma el problema específico de Wikipedia, ésta tiene más de 3 millones de artículos sólo en inglés y en español alrededor de 700.000 artículos. Esto aunado al hecho de que Wikipedia cuenta con alrededor de 30 lenguajes.

Es cierto que los manejadores de bases de datos relacionales tratan siempre de optimizar las consultas, eligiendo el método y la vía más óptima para devolver el resultado esperado en el menor tiempo posible. Sin embargo y citando a Urs Hölzle, primer Vicepresidente de Ingeniería de Google, “Todo a gran escala tiende a dejar de funcionar” (37).

Una consulta sobre las mismas tablas de la figura 3.1 en la que se utilicen claves primarias o atributos indexados como criterio para filtrar los resultados, puede llegar a ejecutarse de manera casi inmediata como se puede observar en la figura 3.2 (a). El problema surge a la hora de utilizar atributos no indexados como criterio para filtrar los resultados, como se puede observar en la figura 3.2 (b).

Sin embargo, surge la pregunta ¿por qué no se indexan todas las columnas de una tabla si a larga esto optimizaría el desempeño de las consultas? Si bien esto es cierto, como se puede observar en la segunda consulta de la figura 3.2 (a) con respecto a la consulta de la figura 3.2 (b), el costo de actualizar millones de índices a la hora de insertar, eliminar o actualizar datos es demasiado elevado como para siquiera pensar en implementar dicha solución. En la figura 3.3 (a) se pueden observar observar los tiempos de actualización de 5.000.000 de registros y 1.000.000 de registros no indexados. En la figura 3.3 (b) podemos observar los tiempos de actualización de 5.000.000 de registro y 1.000.000 de registros, esta vez indexados.

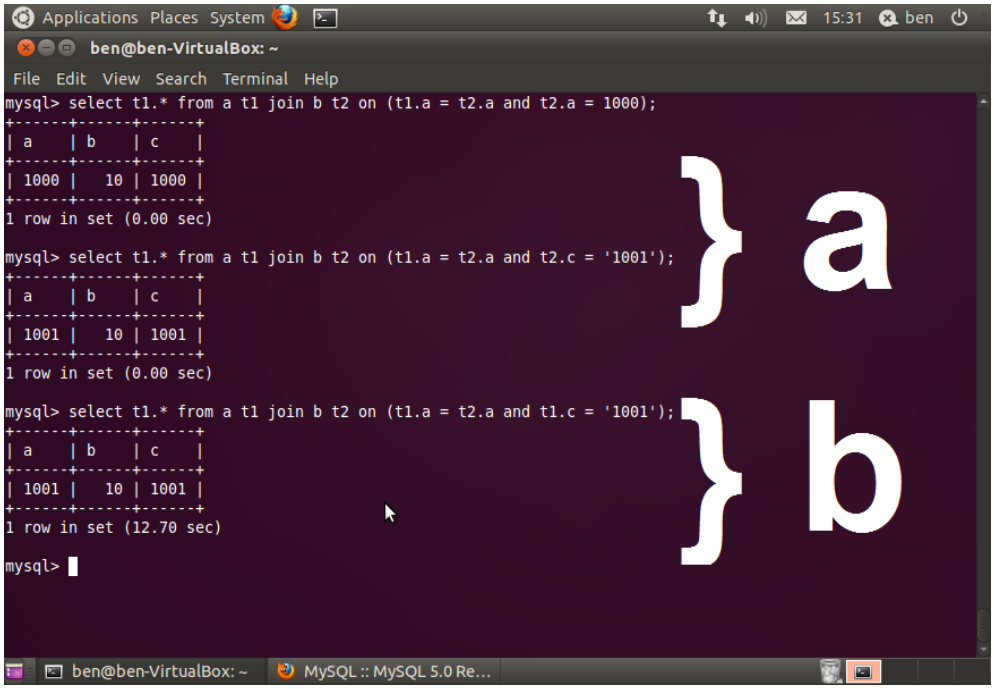

Figura 3.2: (a) Cuando se filtran los resultados a través de claves primarias o indexadas el tiempo de ejecución de una consulta es bajo. (b) Cuando se filtran los resultados a través de claves no indexadas el tiempo de ejecución aumenta de manera considerable.

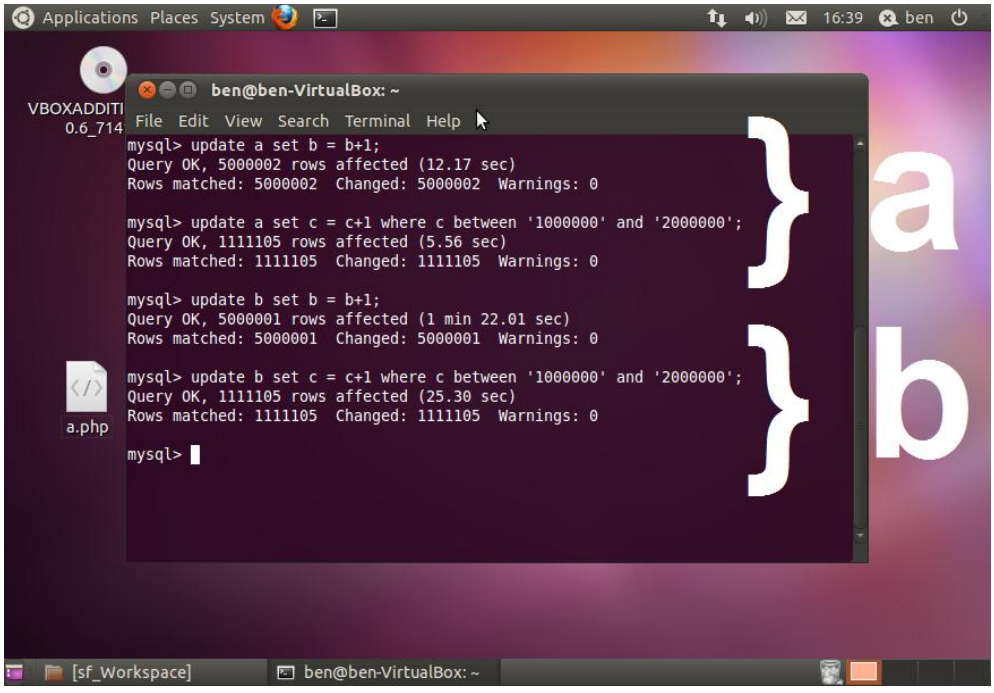

Figura 3.3: (a) Costo de actualizar alrededor de 5.000.000 de registros y 1.000.000 de registros no indexados. (b) Costo de actualizar los mismos registros, esta vez indexados.

El problema existente entre los grandes sistemas y las bases de datos relacionales es la gran cantidad de datos e información que estos sistemas generan y el decremento significativo del rendimiento que presentan las bases de datos relacionales a la hora de manipular estas grandes cantidades de información generada. Muy probablemente los tiempos obtenidos en las figuras 3.3 (b) y 3.2 (b) puedan disminuir si se decide escalar verticalmente el servidor, esto es, reemplazar o agregar los componentes actuales del servidor por otros mucho más potentes. No obstante, escalar verticalmente puede resultar muy costoso ya que a mayor potencia de los componentes, mayor será el costo. Incluso, llegará el momento en el que un servidor ya no se pueda escalar verticalmente debido a las limitaciones físicas del mismo y se necesite aumentar aún más el desempeño del sistema.

En este caso, en vez de escalar verticalmente se puede escalar horizontalmente, esto es, tener varios servidores de base de datos, poco costosos, conectados a través de una red, cada uno con una réplica o fragmento de la base de datos, los cuales distribuirían la carga de trabajo. En el caso de la replicación se compartiría la carga de lecturas mientras que en el caso de la fragmentación se compartiría no solo la carga de lecturas sino de escrituras, lo que conllevaría a un aumento casi lineal en el desempeño del sistema.

La distribución física de una base de datos relacional, aunque posible, no es nada trivial y usualmente el costo de las herramientas que permiten la distribución, fragmentación y replicación de una base de datos relacional sobre una red suele ser muy alto, como por ejemplo la herramienta RAC de Oracle que puede llegar a costar $26.000 por procesador (38).

1.3.2. NoSQL

NoSQL, Not only SQL, es una amplia clase de sistemas de base de datos que se definen como no relacionales y cuyo objetivo es el de tratar de evitar el uso de las funcionalidades de las bases de datos relacionales (36). Las bases de datos NoSQL deben ser (39):

- No relacionales, es decir, las bases datos NoSQL no poseen ni manipulan relaciones o tablas, en cambio representan los datos a través de documentos, grafos, objetos, entre otros.
- Escalables horizontalmente, lo cual implica la posibilidad de tener varios servidores de base de datos compartiendo recursos y carga de trabajo, aumentando casi linealmente el desempeño del sistema por cada servidor en la red (36).
- Distribuidas, lo cual implica que los datos pueden ser replicados y/o fragmentados en múltiples nodos. Esto es consecuencia de la escalabilidad horizontal.

A su vez, las bases de datos NoSQL pueden, o no, cumplir con las siguientes características (36):

- Esquema de datos libre, lo cual implica la posibilidad de agregar o eliminar nuevos atributos sobre los datos sin la necesidad de modificar la estructura de la base de datos y todo lo que esto conllevaría.

- Independencia de los nodos, ningún nodo depende de otro, esto quiere decir que los datos son replicados y/o fragmentados en cada uno de los nodos, aumentando la disponibilidad, redundancia de datos, distribución de la carga y nivel de paralelización.

Usualmente las bases de datos NoSQL no usan JOINs ni constraints sobre los datos como lo harían sus contrapartes relacionales, debido a que las bases de datos NoSQL no pasan por un proceso de normalización durante su creación y, por lo tanto, los datos no son descompuestos en múltiples relaciones. A su vez, el modelo de consistencia deja de ser ACID (atomicidad, consistencia, aislamiento, durabilidad), esto ocurre debido a la dificultad de garantizar la consistencia en sistemas de bases de datos distribuidos. Si un dato es agregado, modificado o eliminado en cualquiera de los nodos, todos los nodos deberían actualizarse inmediatamente para poder garantizar la consistencia general del sistema. Esto último no es tarea sencilla, mucho menos eficaz o eficiente (36).

Las bases de datos NoSQL usan el modelo de consistencia BASE (básicamente disponible, estado suave, consistencia eventual). En base a esto, la consistencia de una base de datos NoSQL no se alcanza inmediatamente luego de una transacción, se alcanza eventualmente, ya no se tendrá un estado fuerte sino un estado suave. Esto quiere decir que si una transacción es ejecutada exitosamente en un nodo, eventualmente el resultado de dicha transacción se propagará al resto de los nodos. Lo importante de este modelo es la disponibilidad permanente de los datos al tener un sistema distribuido en n nodos, cada uno con una réplica de los datos (36).

Esto último asumiendo que una base de datos NoSQL sea distribuida y tenga al menos dos servidores o nodos. Muchas bases de datos NoSQL permiten replicar los datos, más no es obligatorio. Si una base de datos NoSQL trabaja en un único nodo o servidor, sin replicación, ésta sigue trabajando bajo el modelo de consistencia ACID.

Es importante acotar el hecho de que aunque NoSQL sea un movimiento novedoso y una alternativa a las bases de datos relacionales, esto no implica que su implementación sea necesaria en el 100% de los escenarios existentes en la actualidad. Estas bases de datos surgen para solucionar problemas muy específicos de rendimiento, escalabilidad y resistencia al cambio que son característicos de grandes sistemas que generan grandes cantidades de información. También cabe destacar que las bases de datos NoSQL no fueron concebidas con la intención de soportar consultas complejas sobre múltiples modelos de datos, para esto las bases de datos relacionales siguen siendo la mejor opción (36).

Incluso existen casos de grandes sistemas con grandes cantidades de información en los que las bases de datos NoSQL no son una opción viable. Un ejemplo de esto son los sistemas bancarios. En este tipo de sistemas no podemos garantizar una consistencia eventual ya que podría darse el caso de tener dos balances de cuenta distintos en nodos diferentes al mismo tiempo. Si tomamos el ejemplo del sistema de métricas de Wikipedia, una consistencia eventual no representaría problema alguno, el hecho de que una métrica en un nodo tenga un valor y en otro nodo un valor complemente distinto en un momento dado, no representaría

ningún problema, eventualmente el valor más reciente de dicha métrica se propagará al resto de los nodos y podrá ser visualizado.

Como se mencionó anteriormente, las bases de datos NoSQL escalan horizontalmente y por lo tanto son distribuidas. Es por esto que a su vez las bases de datos NoSQL cumplen con el teorema CAP, también conocido como el Teorema de Brewer, el cual menciona la imposibilidad de un sistema distribuido de garantizar simultáneamente consistencia, disponibilidad de los datos y tolerancia al particionamiento. Según el teorema un sistema distribuido, sólo puede garantizar dos de las tres características anteriores (40).

También se mencionó el hecho de que las bases de datos NoSQL utilizan un esquema de datos libre. La ventaja principal de este hecho es la flexibilidad que esto ofrece a la hora de actualizar la estructura de los datos dentro de las bases de datos.

Muchas veces una base de datos relacional es concebida bajo cierto esquema, se tienen una serie de relaciones o tablas derivadas del proceso de normalización, las cuales definirían lo que sería nuestro esquema de datos. Por ejemplo, si se retoma el ejemplo anterior de la Facultad de Ciencias, se tendría una relación estudiantes con los atributos cédula, nombre, apellido, etc. Sin embargo, los sistemas no son inmutables y es probable que se desee alterar el esquema de datos de dicho sistema. Quizás en la Facultad de Ciencias requieran agregar un nuevo campo a la tabla o relación estudiante, esto implicaría costosas reescrituras de tablas o relaciones y el posible y tedioso rediseño de todo el modelo relacional, o por lo menos gran parte de él.

1.3.2.1. Categorías de las bases de datos NoSQL

Las bases de datos NoSQL se categorizan de acuerdo al tipo de esquema de datos que utilizan. Ya hemos mencionado que no usan un esquema de datos relacional, en cambio emplean esquemas de datos libres como: clave-valor, documentos, columnares, grafos, XML, entre otros (36).

A continuación se describirán algunos de estos esquemas de datos.

Clave-valor

Una base de datos clave valor, como su nombre lo indica, asocia un valor a una clave. Cada clave es única, lo cual se logra utilizando funciones hash. Los valores almacenados pueden ser tipos de datos bien definidos, como enteros, cadenas de caracteres, arreglos de bytes, etc (36).

Muchas bases de datos clave-valor no restringen el tipo de dato almacenado, simplemente almacenan lo que se les indique almacenar. Estos datos son almacenados como BLOBS (Objetos Binarios Grandes) los cuales no necesitan ser interpretados ni manipulados por la base datos, simplemente almacenados.

Un ejemplo de un par clave-valor podría verse en la figura 3.4.

Clave	Valor
3df47d19e2c6aa	Foo Bar

Figura 3.4: Ejemplo de un par clave-valor.

La ventaja de este modelo es su simplicidad, sin relaciones ni estructuras. Fueron concebidas para un manejo rápido y eficiente de datos en sistemas distribuidos.

Algunos ejemplos de este tipo de base de datos son: Dynamo, Cassandra y Redis.

Documentos

Este tipo de base de datos utiliza distintos tipos de documentos independientes para representar los datos. Algunos ejemplos de los tipos de documentos son: XML, JSON, YAML, entre otros. Este tipo de base de datos podría considerarse como un caso particular de las bases de datos clave-valor, sin embargo el sistema manejador de base de datos debe conocer con anterioridad qué tipo de documento se va a utiliza para poder interpretarlo. Esta es una de las principales diferencias entre las bases de datos de documentos y las bases de datos clave-valor, simplemente no podemos almacenar distintos tipos de documentos de manera arbitraria (36).

La principal ventaja de este tipo de base de datos es que el esquema de los datos ya no es fijo, es libre, los documentos pueden tener estructuras y atributos arbitrarios asociados a ellos. Por ejemplo, supongamos que tenemos un documento JSON de una persona cuyos atributos son los siguientes:

```
{
    “nombre” : “A”,
    “teléfonos” : “123-123-4567”
}
```

Ahora, si tenemos un segundo documento JSON con los atributos:

```
{
    “nombre” : “B”,
    “email” : “C”
}
```

En un sistema de base de datos relacional se tendría la relación persona con los atributos nombre, email y teléfono, en donde el teléfono del segundo documento y el email del primero serían campos vacíos, o NULL. En una base de datos de documentos sólo los datos que realmente son usados son almacenados, evitando así los gastos generales de recursos que implicaría tener campos vacíos y teniendo la libertad de almacenar, sin restricción alguna, atributos de nuestro interés. Como se mencionó anteriormente, los sistemas no son inmutables

y este tipo de base de datos nos permite tener una gran flexibilidad ante los cambios. Si se quisiera agregar una dirección o algún otro atributo al documento persona se podría hacer sin ningún problema, de la misma manera en la que agregamos el campo email en el segundo documento.

Es inconcebible siquiera pensar en una base de datos relacional que sus tuplas tengan distintos atributos, por lo que ante un cambio de la estructura del sistema habría que rediseñar el modelo relacional subyacente, proceso nada fácil y engorroso.

Algunos ejemplos de este tipo de base de datos: MongoDB, CouchDB, entre otros.

Grafos

Este tipo de bases de datos tienen su origen en la teoría de grafos, en la cual se tienen nodos y aristas que conectan los nodos. En estas bases de datos, los nodos son entidades, como por ejemplo personas, mientras que las aristas representan las relaciones entre dichas entidades. Estas relaciones son similares a las relaciones encontradas en las bases de datos relacionales (36).

Los grafos utilizados pueden ser grafos regulares o grafos dirigidos. En los grafos dirigidos la dirección de las aristas es de suma importancia ya que la relación entre dos entidades puede ser no recíproca.

La ventaja de este tipo de base de datos radica en la consulta de los datos. Las bases de datos de grafos pueden utilizar recorridos en lugar de JOINs a la hora de navegar a través de un conjunto de relaciones y entidades, lo cual es más eficiente y simple.

Las bases de datos de grafos son fáciles de escalar horizontalmente, proveyendo replicación de vértices y aristas. Para un mayor desempeño, la teoría de grafos provee el grafo de partición. Para particionar un grafo se divide el grafo en pedazos, tal que cada pedazo sea del mismo tamaño que los demás y hayan pocas conexiones entre dichos pedazos (41).

Algunos ejemplos de este tipo de base de datos son: FlockDB, Pregel y Trinity.

1.3.2.2. Sistemas manejadores de bases de datos NoSQL

De igual manera que las bases de datos relacionales, las bases de datos NoSQL necesitan de sistemas manejadores de bases de datos NoSQL que permitan la interacción entre ellas y los distintos actores que las manipulen. A continuación se muestra un cuadro comparativo entre MongoDB, CouchDB y el manejador de bases de datos relacionales MySQL (44).

	MongoDB	CouchDB	MySQL
Modelo de datos	Orientado a documentos (JSON)	Orientado a documentos (JSON)	Relacional
Tipos de datos	String, number, boolean, array, object	String, int, double, boolean, date, bytearray, object, array, entre otros	Bigint, int, decimal, date, datetime, char, varchar, string, entre otros
Objetos grandes (Archivos)	Archivos adjuntos	GridFS	BLOBS
Particionamiento horizontal	CouchDB Lounge (aplicación de terceros)	Auto-sharding	
Replicación	Mestro-maestro	Maestro-esclavo y conjuntos de réplicas	Maestro-esclavo
Almacenamiento de objetos (Filas)	Un único gran repositorio	Basado en colecciones	Basado en tablas
Métodos de consulta	Funciones map/reduce en JavaScript	Lenguaje basado en objetos	SQL
Índices secundarios	Sí	Sí	Sí
Atomicidad	Único documento	Único documento	Sí, avanzada
Interfaz	REST	Drivers nativos, complementos para REST	Drivers nativos
Manipulación de datos por lotes por parte del servidor		Map/reduce, JavaScript del lado del servidor	Sí, SQL
Creado en	Erlang	C++	C++
Consistencia distribuida	Eventualmente consistente	Fuertemente consistente	Fuertemente consistente

Figura 3.5: Cuadro comparativo entre los manejadores de bases de datos MongoDB, CouchDB y MySQL (44).

1.4. Tecnologías para visualización Web.

Una vez obtenidos los datos de interés, y luego de haberlos almacenado en una base de datos, es necesario contar con herramientas que permitan visualizar dichas métricas de manera gráfica, ya que por medio de éstas se pueden apreciar rasgos claves, patrones o tendencias de una o más variables y también permiten reconocer la existencia de relación entre variables. Estas gráficas aportan la evidencia necesaria para fundamentar conclusiones y permiten una mejor y más rápida comprensión de los resultados de las métricas, debido a la gran cantidad de datos que un único artículo puede llegar a generar.

Para este trabajo se ha decidido utilizar un front-end web y no una aplicación de escritorio como tal. Esto presenta una serie de ventajas como:

- Portabilidad, para acceder a los datos simplemente se necesitará un Navegador Web que soporte JavaScript.
- Independencia, no se dependerá de las arquitecturas ni de los sistemas operativos de los computadores desde los que se acceda a los datos.
- Actualizaciones transparentes para el usuario final, si se desea actualizar el front-end, el usuario final no se verá obligado a descargar nuevas versiones, siempre accederá a la versión más reciente a través de un Navegador Web.

Para poder visualizar información de manera gráfica a través de un Navegador Web, se tienen básicamente dos opciones: utilizar JavaScript o utilizar Adobe Flash. Ésta última opción no será considerada ya que los resultados que pudiesen obtenerse al utilizar Adobe Flash también pudiesen obtenerse utilizando Canvas, una herramienta introducida en la quinta revisión de HTML que detallaremos más adelante, en conjunto con JavaScript, sin forzar ni atar al usuario a depender de un plugin para acceder y visualizar la información.

1.4.1. HTML5

HTML5 es la quinta revisión del Lenguaje de Marcado de Hipertexto, mejor conocido como HTML. Esta revisión fue desarrollada por el grupo WHATWG (Web Hypertext Application Technology Working Group), fundado por Apple, la Fundación Mozilla y Opera Software en el 2004, luego de que la W3C decidiera abandonar HTML, posteriormente a la publicación de la revisión 4.01de HTML, para enfocarse en el desarrollo de XHTML1.0 y XHTML2.0. Posteriormente, en el 2007, el grupo WHATWG decide que el nuevo grupo encargado del desarrollo de HTML en el W3C adopte el proyecto HTML5 y así sucede.

En esta versión se introducen nuevas características, así como nuevos elementos, en base a las necesidades más comunes de los desarrolladores Web (45).

Cabe destacar que HTML5 no es un todo, sino más bien un conjunto de características y elementos individuales que lo componen. Por lo tanto, el soporte de HTML5 por parte de los Navegadores Web consiste en el soporte individual de dichas características y nuevos elementos.

HTML5 es completamente compatible con las versiones anteriores de HTML. Si se tiene una aplicación Web desarrollada en HTML4.01, por ejemplo, o una versión anterior, es posible convertir dicha a aplicación a una aplicación HTML5 tan sólo con cambiar el doctype de los documentos Web de la aplicación. Más aún, trabajando sobre una versión anterior de HTML, es posible hacer uso de los nuevos elementos y nuevas características de HTML5 sin necesidad de realizar algún tipo de conversión o reescritura y restructuración de los documentos Web.

Algunos de los nuevos elementos introducidos en esta versión de HTML son:

- Video: el cual nos permite insertar un reproductor de video básico en un documento Web, sin la necesidad de utilizar herramientas desarrolladas con Adobe Flash o desarrollarlas desde cero. Soporta los formatos Ogg, H.264 y WebM VP8.
- Audio: similar al elemento anterior pero en este caso se inserta un reproductor de sonido. Soporta los formatos Ogg Vorbis, WAV PCM, MP3, AAC y Speex.
- Campos especiales: los cuales se apoyan en el Navegador Web para realizar las validaciones de dichos campos en vez de apoyarse en soluciones realizadas en JavaScript. Esto no sólo reduce los tiempos de desarrollo sino que asegura una correcta validación de los mismos. Algunos de estos campos especiales son: email, url, phone, number, range, date, time, search, ente otros.

HTML incluye, adicionalmente, elementos semánticos que le permiten a los Navegadores Web y Web Crawlers en general, entender e interpretar las aplicaciones Web que están tratando y por ende generar resultados o información más relevantes y mejor definidos para el usuario final. No detallaremos los elementos semánticos ni hablaremos de la Web 3.0, ya que esto escapa al alcance de este trabajo.

Así como HTML5 introduce nuevos elementos, también provee una serie de nuevos métodos, atributos y funcionalidades para poder interactuar con dichos elementos utilizando JavaScript. Por ejemplo, el elemento Video tiene los métodos play() y pause(), así como el atributo volume.

Anteriormente se hizo mención del elemento Canvas, elemento que nos permitía visualizar información de manera gráfica en un Navegador Web sin la necesidad de instalar plugins adicionales ni de desarrollar soluciones utilizando Adobe Flash, a continuación explicaremos más en detalles este elemento.

1.4.1.1. Canvas

El elemento Canvas, como lo define el WHATWG, es “un mapa de bits dependiente de la resolución el cual puede ser utilizado para generar gráficos, gráficos de juegos u otras imágenes visuales sobre la marcha.” (46). Esencialmente, el elemento Canvas es un rectángulo en el cuál se puede utilizar JavaScript para dibujar lo que se quiera (47).

Es posible tener más de un elemento Canvas dentro de un documento o página Web. Cada elemento Canvas será visible en el DOM y mantendrá su propio estado.

Todo elemento Canvas tiene un contexto de dibujo, el cual puede ser obtenido a través del método getContext() de dicho elemento. Dicho método acepta la dimensión del mapa de bits como parámetro. Actualmente los Navegadores Web solo soportan, de manera estándar, mapas de bit de dos dimensiones (47).

Todos los métodos y propiedades de dibujo se encuentran definidos en dicho contexto de dibujo.

Los elementos Canvas pueden volver a su estado original, es decir, quedar en blanco, tan solo con cambiar el atributo height o width de dichos elementos (47). Esto puede resultar útil a la hora de realizar animaciones o incluso juegos.

Teniendo en cuenta todo lo anterior, un elemento Canvas, con un contexto de dibujo de dos dimensiones, podría definirse como una cuadrícula bidimensional, siendo la coordenada (0, 0) el punto ubicado en la esquina superior izquierda de dicha cuadrícula. En la figura 4.1 podemos observar una representación de un elemento Canvas con dimensiones 500x375 (ancho, alto).

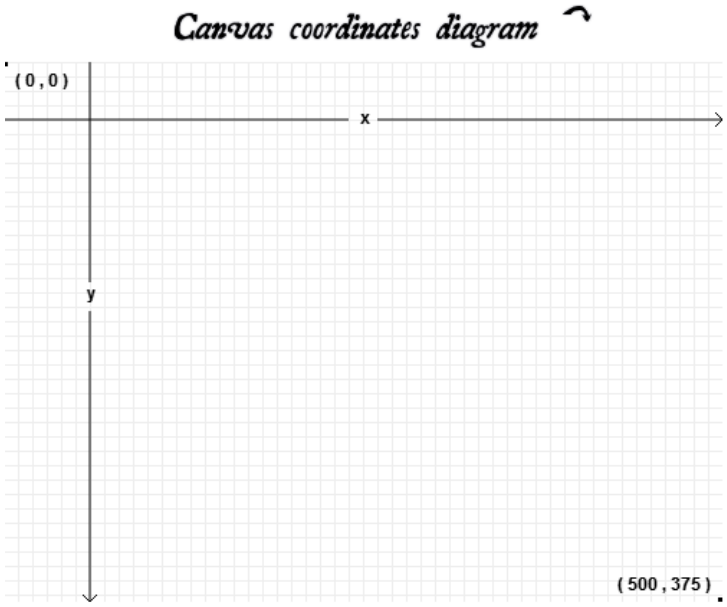

Figura 4.1: Representación de un elemento Canvas (47).

Con este nuevo elemento se podrían generar gráficos que pueden ir desde lo más sencillo, como podemos observar en la figura 4.2.

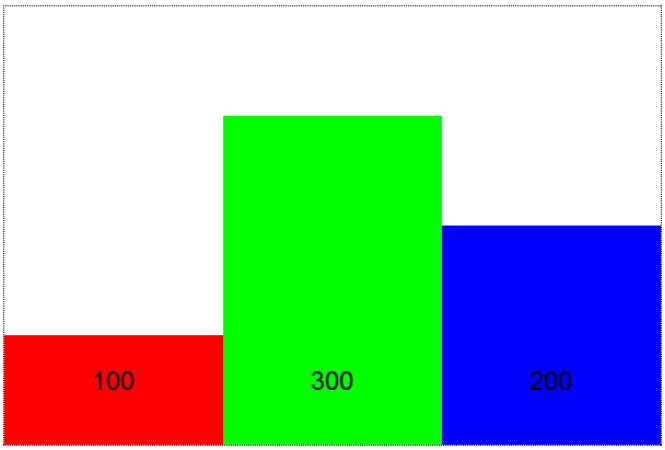

Figura 4.2: Sencillo gráfico de barras creado utilizando el elemento Canvas.

Hasta lo más complejo como se puede observar en la figura 4.3.

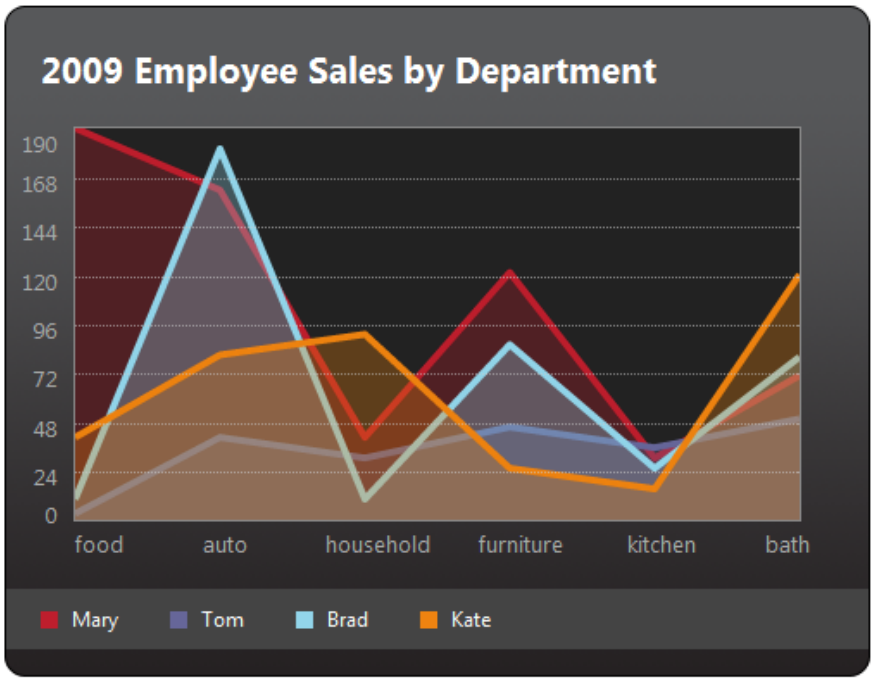

Figura 4.3: Gráfico complejo creado utilizando el elemento Canvas (48).

Sin embargo, crear una librería o un API que nos permita crear gráficos desde cero no es la opción más adecuada, sobre todo cuando existen decenas de soluciones libres disponibles.

En la próxima sección se mencionarán y describirán distintas herramientas para crear gráficos utilizando JavaScript, así como posibles elementos de HTML5, disponibles en la actualidad.

1.4.2. Herramientas para crear gráficos utilizando JavaScript

A continuación se listan algunas herramientas libres que permiten crear gráficos utilizando únicamente JavaScript y/o Canvas.

1.4.2.1. Google Chart Tools

Google Chart Tool es una herramienta que permite visualizar datos de manera gráfica en una página o documento Web. Permite visualizar distintos tipos de gráficos como por ejemplo: de línea, jerárquicos, de torta, de barra, de área, dispersos, geográficos, de tabla, entre otros (49).

Los gráficos son tratados como clases de JavaScript y los datos son cargados a través de clases DataTable. Estas clases representan una tabla de valores multidimensional y mutable. A cada columna de dicha tabla se le asigna un tipo de dato, además de varias propiedades opcionales como ID, título, entre otros. De igual manera, se pueden agregar propiedades definidas por el usuario (50). En la figura 4.4 podemos observar la instanciación de una clase DataTable con las columnas Task y Hours per Day, cada una con su respectivo ID, título y tipo de dato. A su vez, en la figura 4.4 se puede observar el llenado de dicha instancia de la clase DataTable mediante la inserción de filas con sus respectivos datos.

```
var dt = new google.visualization.DataTable(
     {
       cols: [{id: 'task', label: 'Task', type: 'string'},
                {id: 'hours', label: 'Hours per Day', type: 'number'}],
       rows: [{c:[{v: 'Work'}, {v: 11}]},
              {c:[{v: 'Eat'}, {v: 2}]},
              {c:[{v: 'Commute'}, {v: 2}]},
              {c:[{v: 'Watch TV'}, {v:2}]},
              {c:[{v: 'Sleep'}, {v:7, f:'7.000'}]}
             ]
     },
   0.6
)
```

Figura 4.4: Instanciación y llenado de la clase DataTable (50).

En la figura 4.5 podemos observar el resultado de graficar el DataTable anterior.

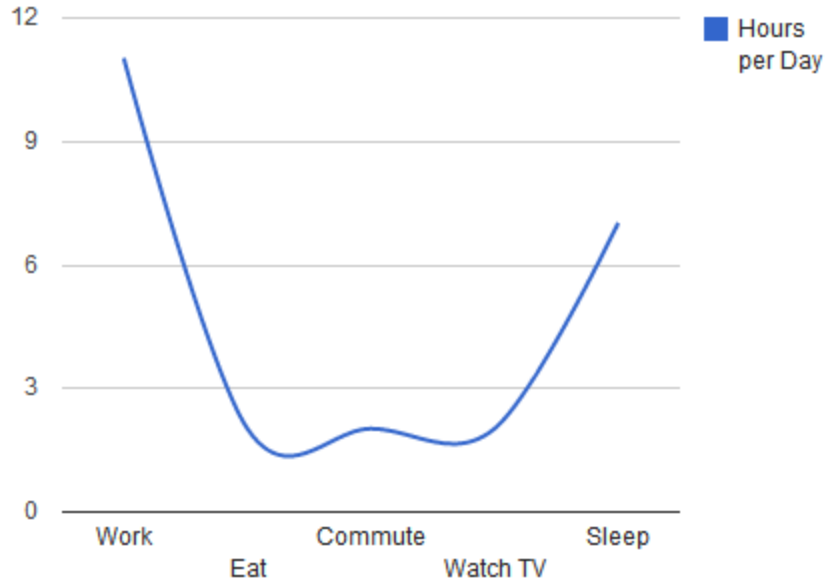

Figura 4.5: Gráfico realizado utilizando Google Chart Tools.

Lo interesante de esta herramienta es que permite consultar sitios Web que implementen el protocolo Chart Tools Datasource. Éste protocolo incluye un lenguaje de consulta al estilo de SQL. El protocolo puede incluso implementarse en páginas y/o aplicaciones Web creadas por usuarios y convertir dichas soluciones en proveedores de datos.

1.4.2.2. Flotr

Flotr es una librería que permite graficar conjuntos de datos arbitrarios utilizando JavaScript, basada en el framework de JavaScript Prototype 1.6+ y Canvas de HTML5 (51).

Flotr permite crear gráficos de histogramas, de líneas, de puntos, de torta, de radar, entre otros.

Para crear un gráfico utilizando Flotr primero se debe crear un elemento de tipo div que contendrá el gráfico. Flotr inserta dos elementos de tipo Canvas dentro de dicho div. El primer Canvas es utilizado para dibujar las líneas de fondo, los ejes del plano y el gráfico como tal. El segundo elemento es usado como una capa que permite mostrar información basada en las acciones del usuario.

Luego, se invoca el método draw propio de la librería de Flotr. Dicho método posee tres parámetros: el div contenedor, un arreglo de datos o series de datos y un objeto opcional de opciones.

Los datos son representados en arreglos compuestos de coordenadas x, y. Un ejemplo de un posible arreglo de datos es el siguiente: [{ x1, y1}, {x2, y2}].

El objeto de opciones es un objeto que tiene como atributos las opciones del gráfico.

Flotr permite graficar distintos arreglos de datos dentro de un mismo gráfico. En la figura 4.6 podemos apreciar un gráfico basado en el siguiente conjunto de datos: [{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}] y [{0, 2.5}, {1, 5.5}, {2, 8.5}, {3, 11.5}, {4, 14.5}].

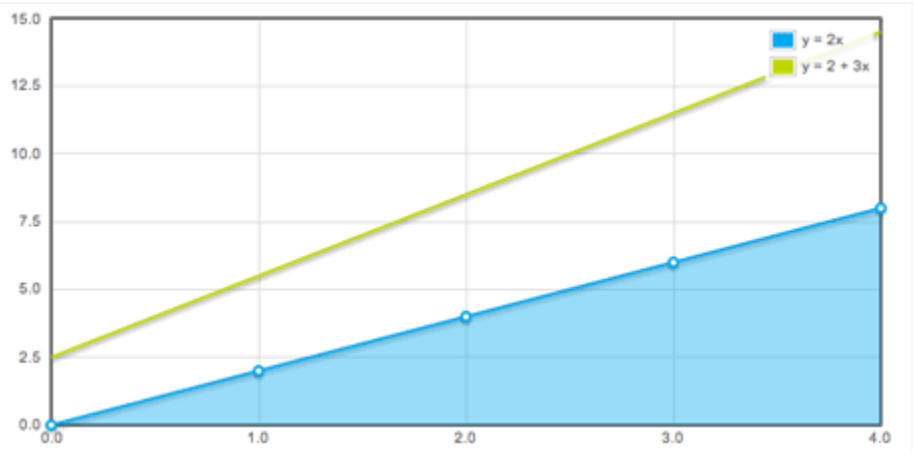

Figura 4.6: Gráfico creado utilizando Flotr

1.4.2.3. jqPlot

jqPlot es un plugin de jQuery, muy similar a Flotr, que permite crear gráficos sobre conjuntos de datos arbitrarios en documentos Web. Requiere jQuery 1.4 o superior y utiliza Canvas de HTML5. jqPlot permite crear gráficos de líneas, dispersos, de torta, de barras, de burbujas, entre otros (52).

Para crear un gráfico se debe primero crear un elemento contenedor, preferiblemente un div. Seguidamente se debe llamar al plugin pasándole tres parámetros, equivalentes a los tres parámetros vistos en Flotr. El primer parámetro es el contenedor del gráfico, el segundo es el conjunto de datos a graficar y el tercer parámetro son las opciones del gráfico.

Los datos son representados por un arreglo de arreglos de coordenadas x, y. Esto permite graficar varios conjuntos de datos en un mismo gráfico.

En lafigura 4.7 podemos apreciar un gráfico basado en el siguiente conjunto de datos: [[[1, 2], [3,5.12], [5,13.1], [7,33.6], [9,85.9], [11,219.9]]].

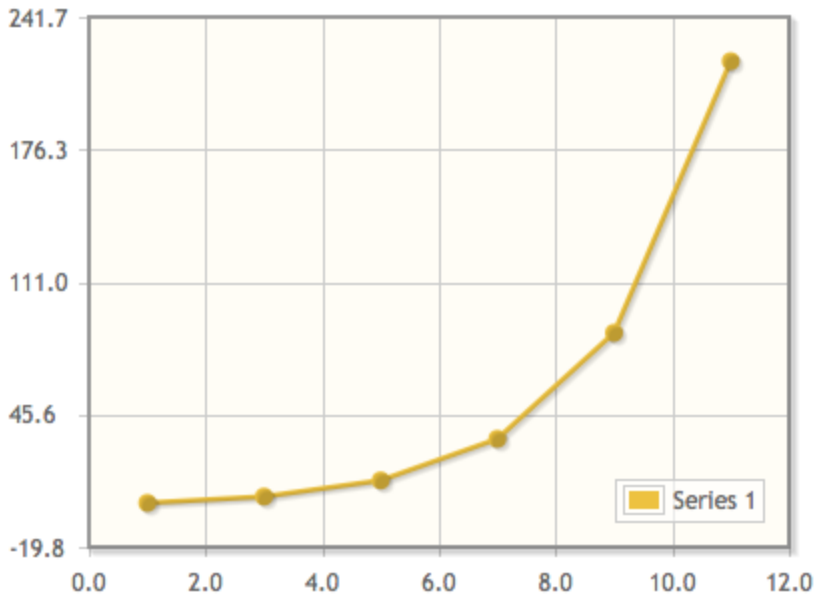

Figura 4.7: Gráfico creado utilizando jqPlot

1.4.2.4. Highcharts

Highcharts es una librería de graficación escrita puramente en JavaScript. Permite crear gráficos interactivos de líneas, de área, splines, splines de áreas, gráficos de columnas, de torta y dispersos (53).

Utiliza el elemento SVG de HTML5 para generar los gráficos. Su uso es gratuito siempre y cuando se utilice sin fines comerciales.

Los datos, como en los dos ejemplos anteriores, son representados a través de arreglos de datos. En la figura 4.8 podemos apreciar un gráfico realizado utilizando Highcharts.

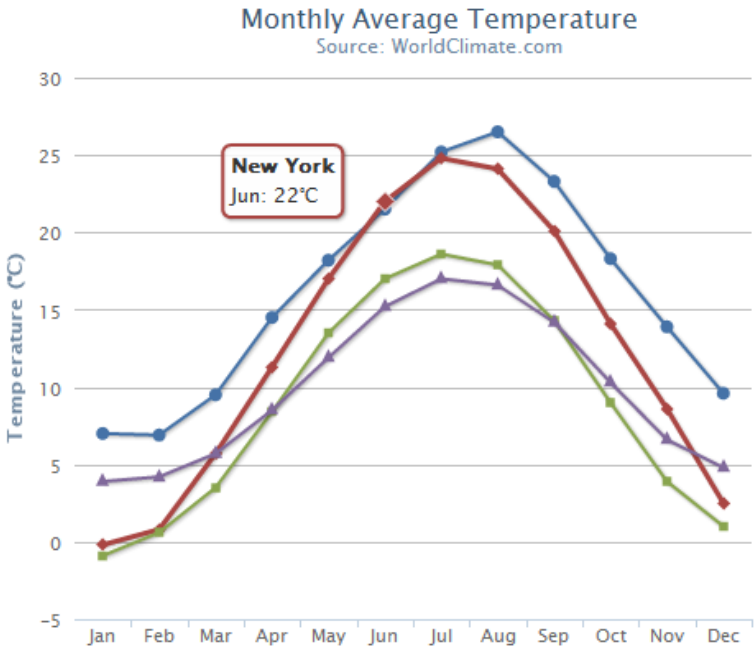

Figura 4.9: Gráfico creado utilizando Highcharts

1.4.2.5. D3

D3 permite asociar un conjunto arbitrario de datos a un Modelo de Objetos del Documentos (DOM) de un documento (X)HTML y luego aplicar transformaciones guiadas por los datos a dicho documento. D3 utiliza CSS3 y elementos de HTML5 como Canvas y SVG y una sintaxis muy similar la utilizada por jQuery y Prototype (54).

D3 no es un framework de visualización tradicional, en vez de ofrecer un sistema monolítico tradicional con todas las características que el usuario podría necesitar en un momento dado, D3 se enfoca en resolver el problema esencial, manipulación eficiente de documentos basados en datos.

D3 permite crear representaciones de datos poco convencionales, como por ejemplo: vistas de calendario, diagramas de acordes, mapas de coropletas, pirámides de población, cartogramas no contiguos, diagramas voronoi, entre otros.

Los datos son representados en arreglos de datos arbitrarios. En la figura 4.10 podemos apreciar un gráfico realizado utilizando D3.

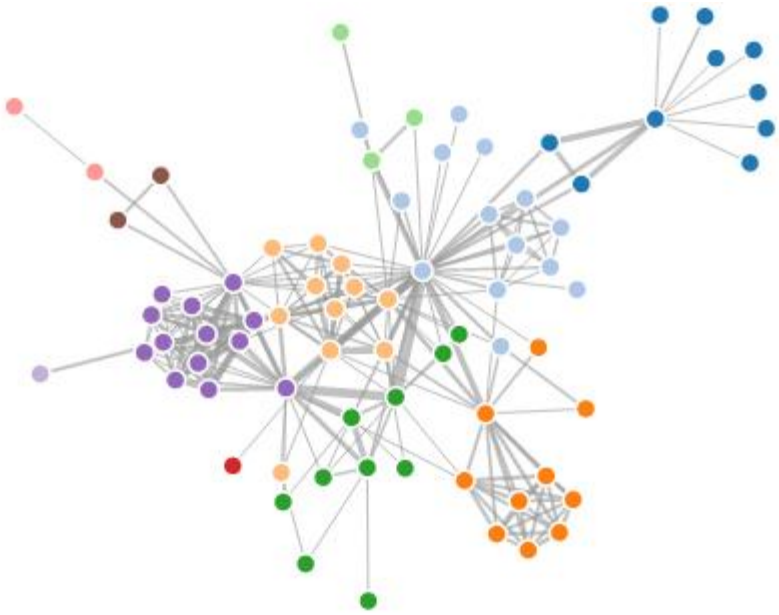

Figura 4.10: Gráfico creado utilizando D3.

CAPÍTULO II: MARCO METODOLÓGICO

A continuación se identifica el problema que da lugar a este trabajo especial de grado, se propone un objetivo general y varios objetivos específicos que nos permiten dar una solución al problema y se pone en práctica el método ágil Extreme Programming (XP), en su totalidad, para llevar a cabo dicha solución.

2.1. Identificación del problema

Un wiki se define como un repositorio web de artículos colaborativos los cuales pueden ser creados y editados por usuarios pertenecientes a la comunidad de dicho wiki.

Los wikis mantienen un registro de los cambios o ediciones que ha sufrido un artículo a través del tiempo dentro de su historial, en donde podemos encontrar información como: las distintas versiones del artículo, los distintos usuarios que han participado en la edición del artículo, las distintas fechas de ediciones de los artículos, entre otras.

Aunque esta información puede llegar a ser útil y permite responder preguntas como ¿cuál fue la última fecha de modificación de un artículo?, no nos permite responder preguntas como ¿en qué fecha sufrió más modificaciones un artículo?

Ciertamente se puede navegar a través de las páginas del historial del artículo e ir contando las modificaciones que éste ha sufrido a través del tiempo de manera manual, sin embargo esto no sería lo ideal.

Lo ideal sería poder ver las distintas métricas de interés que pudiesen derivar de la información presente en el historial de los artículos de un wiki en general. Esto de manera gráfica, para una comprensión rápida, clara y sencilla de los datos que se presenten.

Por lo tanto, se define el problema como: la falta de información de interés, derivada de los datos encontrados en el historial de los artículos de un wiki, que nos permita seguir, de manera gráfica, las métricas y propiedades relacionadas a los cambios que ha sufrido un artículo a través del tiempo.

2.2. Objetivos del trabajo

En base al problema planteado, el objetivo general consiste en la implementación de una aplicación que permita la recuperación del historial de los artículos de cualquier wiki basado en MediaWiki, un software para wikis libre desarrollado por la Fundación Wikimedia, para luego calcular las métricas y propiedades de interés que deriven de la información obtenida en dichos historiales.

Los objetivos específicos necesarios para cumplir con el objetivo general antes planteado son los siguientes:

- Aplicar un método ágil a utilizar para la elaboración de la aplicación, en este caso Extreme Programming (XP).

- Crear un Web Scraper, con políticas bien definidas, que se encargue de recuperar, de manera periódica, el historial de los artículos de cualquier wiki basado en MediaWiki.
- Definir qué manejador de base de datos se va a utilizar, teniendo en cuenta que éste debe permitir manipular grandes cantidades de información de manera eficaz y eficiente.
- Desarrollar una aplicación que permita agregar al sistema las URLs a visitar e implemente una política de recuperación periódica.
- Seleccionar un conjunto de métricas a ser calculadas con los datos obtenidos del historial de los artículos.
- Desarrollar un front-end Web que permita la visualización de las métricas obtenidas a través de un Navegador Web que soporte JavaScript.

2.3. Estrategia de solución

Teniendo en cuenta los objetivos del trabajo, la estrategia de solución consiste en diseñar un Web Scraper, preferiblemente de uso general, con políticas de selección, re-visita, cortesía y paralelización bien definidas. Se planifica el desarrollo de una herramienta para ingresar las URLs al sistema que sirven de entrada para el Web Scraper, así como el desarrollo del daemon de prioridades, el cual se encargaría de actualizar las prioridades de las URLs para definir así la política de re-visita del Web Scraper. Por último, se seleccionan las tecnologías a utilizar para el desarrollo del proyecto, entre ellas el manejador de bases de datos NoSQL y el lenguaje de programación, se diseñó la estructura de los documentos a almacenar en la base de datos y se dividió en el desarrollo en un back-end (de escritorio) y un fron-end (Web).

A continuación se describen cada uno de los puntos de la estrategia de solución.

2.3.1. Web Scraper

El back-end del sistema cuenta un Web Scraper, el cual se guía por las siguientes políticas.

2.3.1.1. Políticas de selección

Las URLs a visitar son provistas por el usuario final; una a la vez, o por lote de URLs a través de un archivo de texto en el cuál se especifica una URL por línea.

Dada la gran cantidad de artículos que posee Wikipedia, extraer URLs de forma recursiva a partir de una URL inicial puede derivar en bucles que bloqueen el proceso de extracción de URLs y se necesite de la intervención del usuario final de todas maneras.

La URL provista es la URL de un artículo y a partir de ésta se deduce, normaliza y almacena la URL del historial del artículo. De igual manera el usuario podrá ingresar la URL del historial del artículo de manera directa, la cual se normaliza y almacena de manera similar a la anterior. Por defecto se extraen las primeras 5.000 ediciones del historial de un artículo. En caso de que el artículo posea más 5.000 ediciones, la URL de la página siguiente del historial del artículo se extrae de manera automática. Posteriormente, al visitar dicha URL extraída de

manera automática, se extrae la página siguiente, de existir, y así sucesivamente hasta obtener y visitar todas las URLs de dicho artículo.

2.3.1.2. Políticas de re-visita

La re-visita de las URLs se basa en prioridades de re-visita. Cuando una URL es almacenada en el sistema como una nueva URL, su prioridad tiene un valor igual a 0.5. Al visitar una URL y luego de haber extraído la información requerida, la prioridad de dicha URL será actualizada con un valor igual a 0. Esta probabilidad aumentará en el tiempo de acuerdo a la siguiente fórmula:

$$1 - \left(\frac{T_0}{T_i}\right)$$

Dónde:

- T_0 es la marca de tiempo (timestamp) de la última fecha de actualización de la URL, por defecto será 0.
- T_i es la marca de tiempo (timestamp) de la fecha en la que se está calculando la prioridad de re-visita.

Esta fórmula presenta las siguientes ventajas:

- Una nueva URL siempre tiene una prioridad superior a cualquier URL que haya sido visitada previamente. Esto permite visitar cualquier URL nueva tan pronto como sea posible.
- El incremento de la prioridad es muy lento, asegurando una prioridad igual a 0.5 en 30 años y una prioridad igual a 1.0 en 150 años.

Las URLs serán re-visitadas siempre y cuándo éstas tengan una prioridad mayor que 0, y de acuerdo a su prioridad. Adicionalmente, se verifica la cabecera HTTP last-update devuelta por el servidor al solicitar un artículo y el hash MD5 de la primera página del historial del artículo para determinar si el recurso ha sido modificado desde nuestra última visita.

2.3.1.3. Políticas de cortesía

Luego de cada petición se interrumpirá la ejecución del Web Scraper durante un (1) segundo.

2.3.1.4. Políticas de paralelización

Luego de realizar varias pruebas, se llega a la conclusión de que tener varios threads, realizando varias peticiones de manera simultánea a historiales de gran magnitud, duplicaría e incluso triplicaría el tiempo de respuesta por parte del servidor en cada petición realizada. Si un Web Scraper single-threaded procesa una URL en 60 segundos, un Web Scraper multi-threaded procesaría esa misma URL en 120 o 180 segundos aproximadamente.

2.3.2. Herramienta de URLs

En el back-end también cuenta con la herramienta para agregar URLs al sistema. Esta herramienta es una pequeña aplicación que recibe como parámetro una URL o la bandera –f seguida de la ubicación de un archivo de texto. Dicho archivo tiene una lista de URLs, una por línea, sin límite de URLs.

Esta herramienta verifica la validez de las URLs y las agrega al sistema. Aquellas URLs inválidas no son agregadas al sistema, se le indica al usuario y se crea la entrada respectiva dentro del reporte correspondiente.

2.3.3. Daemon de prioridades

De igual manera, en el back-end encontraremos un daemon de prioridades el cual se ejecuta diariamente, exactamente a las 00:00:00 A.M. (hora del servidor). Este daemon simplemente se encarga de re-calcular y actualizar las prioridades de las URLs.

2.3.4. Arquitectura

En base a todo lo discutido anteriormente, la arquitectura de la aplicación es la siguiente (figura 3.1):

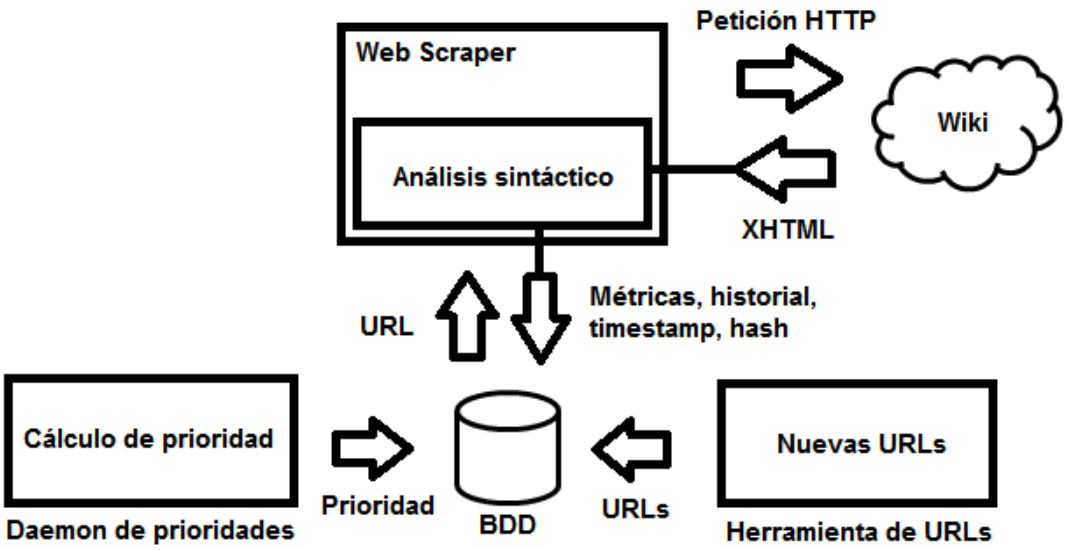

Figura 3.1: Arquitectura de la aplicación a desarrollar.

2.4. Tecnologías a utilizar

Para llevar a cabo la estrategia de solución antes planteada, la cual permite alcanzar el objetivo general propuesto, se utilizan las siguientes tecnologías.

2.4.1. Lenguaje de programación

Ya que las aplicaciones a desarrollar no son CPU-Bound sino I/O-Bound y dependen más que todo del tiempo de respuesta por parte del servidor, el cual hemos determinado que puede llegar a ser mayor a 60 segundos (dependiendo de la cantidad de revisiones que un artículo pueda llegar a tener), podemos entonces prescindir de lenguajes como ANSI C/C++ y optar por un lenguaje con un mayor nivel de abstracción que permite un desarrollo rápido de las aplicaciones y brinde un conjunto de librerías, preferiblemente nativas, que faciliten las tareas a ejecutar.

Python cumple con las características antes mencionadas. No sólo permite un desarrollo eficiente de aplicaciones sino que brinda un conjunto de librerías nativas para realizar peticiones HTTP, manipular sitemaps, cadenas de caracteres, archivos XML, fechas, entre otros. Es por esto que se propone el uso de Python para el desarrollo de las aplicaciones.

Si bien Python no tiene librerías nativas para trabajar con XPath, existe la librería py-dom-xpath que permite implementar soluciones con XPath sin ningún problema. De igual manera, existe la librería pytidylib, un wrapper para limpiar documentos XHTML utilizando HTML Tidy y el driver oficial de MongoDB, pymongo, para trabajar con MongoDB desde Python.

2.4.2. Base de datos

Para almacenar los datos utilizaremos MongoDB, por su relativa facilidad y velocidad a la hora de procesar consultas sobre grandes cantidades de información. Incluso podría, en un futuro, escalarse horizontalmente el sistema sin la necesidad de realizar algún cambio extraordinario. MongoDB permite fragmentar los datos de manera natural y seguir trabajando con las aplicaciones superiores (Web Scraper, herramienta de URLs y daemon de prioridades) como si se estuviera trabajando con una base de datos sin fragmentar y en un único servidor.

2.4.3. Front-end Web

El front-end Web puede es desarrollado en PHP.

Para visualizar las métricas utilizaremos Google Chart Tool, una herramienta madura, con diversidad de gráficos útiles y usables. Google Chart Tools también tiene la ventaja de poder convertir al servidor en un proveedor de datos, utilizando el protocolo Chart Tools Datasource, lo que se considera útil ya que se está trabajando con datos de interés para diversas aplicaciones que pueden simplemente consultar las métricas sin necesidad de crear un Web Scraper para obtener información del front-end Web y repetir todo este proceso nuevamente.

2.5. Planificación inicial del proyecto

Para el desarrollo de este proyecto se escoge el método ágil XP, debido a su gran flexibilidad y tolerancia a cambios, a los pocos roles y fases que este método establece y a la importancia que se le da al trabajo en grupo e intercambio de conocimiento.

A continuación se muestra la planificación de historias inicial luego de reunirse, por vez primera, con el cliente y recopilar los requerimientos funcionales del sistema.

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 1</td><td colspan="2">Usuario: Final</td></tr>
<tr><td colspan="3">Nombre historia: Web Scraper</td></tr>
<tr><td colspan="2">Prioridad en negocio:

Alta</td><td>Riesgo en desarrollo:

Medio</td></tr>
<tr><td colspan="3">Programador responsable: Benjamin Worwa – Roberto D’Apuzzo</td></tr>
<tr><td colspan="3">Descripción:

Dada una URL del historial de un artículo en Wikipedia (tomada de una base de datos), es necesario extraer toda la información referente a las revisiones encontradas en dicho historial y almacenarlas en una base de datos. Es necesario tener en cuenta que el Web Scraper debe ser lo suficientemente flexible y general como para poder ser utilizado en cualquier wiki basado en MediaWiki. Se deben crear políticas de cortesía y paralelización adecuadas.

A su vez es necesario encontrar y medir los límites y capacidades del Web Scraper en cuento a la cantidad de URLs máxima que éste puede procesar por hora y la cantidad de revisiones que éste puede procesar por artículo sin problema alguno (base teórica: 51 URLs/hora de 5000 revisiones c/u).</td></tr>
<tr><td colspan="3">Observaciones:

La política de selección y re-visita no pertenecen a esta historia.

De los historiales se deben calcular ciertas métricas como:

• Número de versiones del historial.
• Número de ediciones menores.
• Proporción de ediciones menores.
• Cantidad de usuarios participantes en la edición del artículo.
• Proporción de número de usuarios editores vs. Número de versiones.
• Número de modificaciones por usuario.</td></tr>
</table>

- Proporción de modificaciones por usuario.
- Número de modificaciones menores y no menores por usuario.
- Entre otras.

Estas métricas pueden ser calculadas sobre la marcha a la hora de consultar los datos o previamente, a la hora de extraer los datos del historial. Cuál de las dos opciones se va a utilizar se definirá en un futuro.

Historia de Usuario	
Número: 2	**Usuario:** Final
Nombre historia: Daemon de prioridades	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: La política de re-visita del Web Scraper se basa en prioridades calculadas a partir de la fecha de la última visita de una URL en particular y la fecha actual del servidor. Toda nueva URL ingresará al sistema con una prioridad de 0.5. Al momento de visitar una URL en particular, la prioridad de ésta es actualizada con una prioridad de 0. Para calcular las nuevas prioridades se utilizará la siguiente fórmula: 1 – (timestamp de la fecha de la última visita / timestamp de la fecha actual) Lo que permite calcular prioridades no acumulativas y de suave crecimiento, superando la marca de 0.5 en 30 años y 1 en 150 años.	
Observaciones: Quedan por definir los intervalos de tiempo adecuados en los que el daemon de prioridades se ejecutará (cada 24 horas, 12 horas, etc.).	

Historia de Usuario	
Número: 3	**Usuario:** Final
Nombre historia: Herramienta de URLs	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Dada una URL de un artículo de Wikipedia (tomada de la entrada estándar o de un archivo de texto), se debe generar la URL correspondiente al historial de dicho artículo, normalizar esta nueva URL y almacenarla en una base de datos. Esto corresponde a la política de selección del Web Scraper.	
Observaciones: Las URLs son utilizadas por el Web Scraper y Daemon de prioridades, por lo que la normalización es de suma importancia.	

Historia de Usuario	
Número: 4	**Usuario:** Final
Nombre historia: Front-end Web	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Se debe crear una aplicación Web que permita la consulta y visualización gráfica de las métricas recopiladas y almacenadas por el Web Scraper.	

Observaciones:
Se tiene previsto el uso de Google Chart Tools para la representación gráfica de las métricas.

Luego de levantar las historias de usuario iniciales se procedió con el plan de entrega del proyecto.

2.5.1. Plan de entrega

Se acuerda un tiempo de entrega de cuatro (4) a cinco (5) meses. En base a esto se procedió a la planificación de la primera iteración del proyecto.

2.6. Primera Iteración

A continuación se lista la documentación de la planificación, diseño, codificación y pruebas que tuvieron lugar durante la primera iteración del proyecto.

2.6.1. Planificación

2.6.1.1. Historias de usuario

Historia de Usuario	
Número: 1	**Usuario:** Final
Nombre historia: Web Scraper	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Dada una URL del historial de un artículo en Wikipedia (tomada de una base de datos), es necesario extraer toda la información referente a las revisiones encontradas en dicho historial y almacenarlas en una base de datos. Es necesario tener en cuenta que el Web Scraper debe ser lo suficientemente flexible y general como para poder ser utilizado en cualquier wiki basado en MediaWiki. Se deben crear políticas de cortesía y paralelización adecuadas. A su vez es necesario encontrar y medir los límites y capacidades del Web Scraper en cuento a la cantidad de URLs máxima que éste puede procesar por hora y la cantidad de	

revisiones que éste puede procesar por artículo sin problema alguno (base teórica: 51 URLs/hora de 5000 revisiones c/u).
Observaciones: La política de selección y re-visita no pertenecen a esta historia. De los historiales se deben calcular ciertas métricas como: • Número de versiones del historial. • Número de ediciones menores. • Proporción de ediciones menores. • Cantidad de usuarios participantes en la edición del artículo. • Proporción de número de usuarios editores vs. Número de versiones. • Número de modificaciones por usuario. • Proporción de modificaciones por usuario. • Número de modificaciones menores y no menores por usuario. • Entre otras. Las cuales pueden ser calculadas sobre la marcha a la hora de consultar los datos o previamente, a la hora de extraer los datos del historial. Cuál de las dos opciones se va a utilizar se definirá en un futuro.

2.6.1.2. Plan de entrega

Luego de reunirse todos los actores del proyecto, se determina que para la primera entrega se concretaría la historia de usuario número 1 (Web Scraper).

Se decide entonces dividir la historia de usuario número 1 en dos historias, las cuales se muestran a continuación.

	Historia de Usuario
Número: 1.1	**Usuario:** Web Scraper de MediaWiki
Nombre historia: Web Scraper de uso general	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta

Programador responsable: Benjamin Worwa – Roberto D'Apuzzo
Descripción: Dada una URL cualquiera que referencie a un recurso HTML/XML (válido o no), es necesario crear una clase que permita extraer toda la información especificada por el usuario a través de expresiones de XPath. Esta información debe ser pública y accesible para todas aquellas clases que implementen a esta clase.
Observaciones: • Es necesario crear un sistema de configuración lo más flexible posible, que permita especificar expresiones de XPath y asociarlas a un host. • Únicamente se tratarán URLs cuyo recurso sea HTML/XML. • Es necesario poder determinar la última fecha de actualización del recurso referenciado para así evitar descargas innecesarias. • Es necesario limpiar (tidy) el código HTML/XML obtenido y codificarlo (charset) de acuerdo a las necesidades del usuario. La codificación de caracteres puede especificarse en el archivo de configuración. • Dado el hecho de que el Web Scraper se ejecuta de manera continua e indefinida en el servidor, es necesario implementar un sistema de registro de mensajes de la aplicación (errores, advertencias, mensajes) que permita ver la fecha y hora de cualquier evento ocurrido. • Es necesario, siempre que sea posible, apegarse a las directivas del archivo /robots.txt.

Historia de Usuario		
Número: 1.2	**Usuario:** Final	
Nombre historia: Web Scraper de MediaWiki		
Prioridad en negocio: Alta		**Riesgo en desarrollo:** Media
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo		

Descripción:
Es necesario crear una clase que implemente la clase “Web Scraper de uso general” (historia de usuario 1.1); que trate, normalice y guarde toda la información extraída, de una URL de un historial de un artículo de Wikipedia (o cualquier otro wiki basado en MediaWiki), en una base de datos. Esta clase también se encargará de decidir cual URL (historial de un artículo) será visitada en un momento dado (tomada de la base de datos). A su vez, esta clase será la encargada de la recuperación de posibles errores que arroje la clase “Web Scraper de uso general” (peticiones fallidas, timeouts, etc.).
Observaciones: • La base de datos a utilizar es MongoDB. • Toda la información obtenida, tratada y normalizada debe estar claramente asociada a una URL a la hora de almacenarla en la base de datos. • Es necesario poder recorrer las distintas páginas de un historial de un artículo de manera automática. • Es necesario crear mecanismos de recuperación de errores coherentes y eficaces.

A su vez, se decide que la historia 1.1 sería realizada durante la primera iteración y la historia 1.2 sería realizada en una segunda iteración.

2.6.1.3. Velocidad del proyecto

Se estima un tiempo de desarrollo de un mes y medio a dos meses para la realización de la historia de usuario número 1.1, aun cuando el cliente especificó no tener una fecha límite específica.

2.6.1.4. Plan de iteraciones

Se divide la historia 1.1 en tareas específicas, las cuales se listan y explican a continuación.

Tarea	
Número tarea: 1	**Número historia:** 1.1
Nombre tarea: Configuración del Web Scraper	
Tipo de tarea : Desarrollo	
Fecha inicio: 22 de Octubre de 2011	**Fecha fin:** 24 de Octubre de 2011

Programador responsable: Benjamin Worwa – Roberto D'Apuzzo

Descripción:

Diseñar e implementar el sistema de configuración del Web Scraper, el cual debe ofrecer la flexibilidad y generalidad buscada. Este sistema de configuración estará compuesto un archivo XML, a través del cual se especican el user agent, codificación de caracteres y expresiones de XPath a ser evaluadas por el Web Scraper (agrupadas por host).

El user agent es utilizado a la hora de realizar peticiones HTTP para identificar al Web Scraper. La codificación de caracteres indica en qué formato será codificado el contenido obtenido de la petición HTTP, mientras que las expresiones de XPath, agrupadas por host, indicarn los elementos que serán buscados y obtenidos para luego ser procesados. Cada expresión de XPath debe tener asociada:

- Un host (P.ej. es.wikipedia.org), el cual permite evaluar distintas expresiones de XPath para cada host.
- Un nombre (P.ej. usuario), el cual permite identificar el resultado de evaluar la expresión de XPath para su uso posterior por parte de otras clases.
- Un contexto (P.ej. historia), el cual permite definir si la expresión de XPath se evaluará sobre el recurso HTML/XML o sobre el resultado de otra expresión de XPath.
- Una variable que indique si se desea obtener el elemento (node) o el contenido del elemento (TextNode).

Este archivo de configuración es cargado en el diccionario de configuración de la clase "Web Scraper de uso general", llamada Scraper de ahora en adelante, cada vez que se instancie dicha clase, indicando inconsistencias y errores en el archivo.

Pruebas de aceptación:

- Se debe obtener un mensaje de error y detener la ejecución de la aplicación siempre y cuando se encuentre una inconsistencia o error en el archivo XML de configuración.
- De no haber ninguna inconsistencia o error en el archivo XML de configuración, se debe obtener en la clase Scraper una variable de tipo diccionario, arreglo asociativo o cualquier estructura de datos similar, que refleje el contenido del archivo.

Tarea	
Número tarea: 2	**Número historia:** 1.1
Nombre tarea: Clase de mensajes	
Tipo de tarea : Desarrollo	
Fecha inicio: 17 de Octubre de 2011	**Fecha fin:** 20 de Octubre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

Diseñar e implementar el sistema de mensajes del Web Scraper, el cual permite al usuario final agregar o modificar los mensajes que muestra el Web Scraper a la hora de arrojar un error, advertencia o mensajes de información. Este sistema de mensajes esta compuesto por un archivo XML, a través del cual se especifican los mensajes y los nombres de dichos mensajes, así como una clase Messages que hace uso de este archivo XML.

El archivo XML de mensajes es cargado cada vez que se instancia la clase Messages, indicando inconsistencias y errores en el archivo.

La clase Messages debe contar con métodos para levantar errores (stderr), informar advertencias (stderr) y mensajes (stdout).

Pruebas de aceptación:

- Se debe obtener un mensaje de error y detener la ejecución de la aplicación siempre y cuando se encuentre una inconsistencia o error en el archivo de XML de mensajes.
- De no haber ninguna inconsistencia o error en el archivo XML de mensajes, se deben crear variables de clase públicas identificadas con los nombres de los mensajes previamente especificados en el archivo XML de mensajes. Estas variables deberán contener el mensaje correspondiente.
- Los métodos para levantar errores e informar advertencias y mensajes deben comportarse como es esperado.

	Tarea
Número tarea: 3	**Número historia:** 1.1
Nombre tarea: Clase de registro de mensajes	
Tipo de tarea : Desarrollo	
Fecha inicio: 20 de Octubre de 2011	**Fecha fin:** 22 de Octubre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

Diseñar e implementar la clase de registro de mensajes del Web Scraper, la cual permite registrar los mensajes arrojados por el sistema (tarea número 2) en los registros del sistema.

Estos registros son divididos en dos categorías, Internal y Scraper. Internal hace referencia a los errores de configuración de la clase Scraper, así como a cualquier otro error interno que surja durante la ejecución de esta clase (errores de E/S, errores de socket, etc.). Scraper hace referencia a los errores que puedan surgir a la hora de realizar una petición HTTP a un servidor (tiempo de conexión agotado, recurso no disponible, etc.). Estos registros serán guardados en los directorios /log/internal y /log/scraper respectivamente y tienen por nombre el dia, mes y año actual del servidor con extensión .log (P.ej. /log/internal/31-12-2011.log).

Cada registro del sistema cuenta con un conjunto de mensajes que tendrán la hora, minutos y segundos seguidos del mensaje a registrar (P.ej. [13:59:59] Ha ocurrido un error).

Esta clase es utilizada por la clase Messages (tarea número 2).

Pruebas de aceptación:

- Al arrojarse un mensaje por parte de la clase Scraper, éste debe verse reflejado en los registros del sistema. Los mensajes deben clasificarse de manera correcta de acuerdo a su categoría.
- La ubicación de los directorios de registros del sistema (/log/internal y /log/Scraper), así como la de los registros del sistema (/log/internal/foo.log y /log/Scraper/bar.log), deben ser relativas y siempre crearse/modificarse en el mismo directorio, sin importar la ubicación actual del usuario.
- La hora y fecha en la que se registran los mensajes debe ser correcta.

	Tarea
Número tarea: 4	**Número historia:** 1.1
Nombre tarea: Clase de validaciones	
Tipo de tarea : Desarrollo	
Fecha inicio: 26 de Octubre de 2011	**Fecha fin:** 26 de Octubre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase de validaciones del Web Scraper, la cual permite validar las entradas por parte del usuario (URLs, identificadores de mensajes, identificadores de expresiones de XPath, etc.). Esta clase es utilizada por la clase Messages y Scraper y debe contar con métodos para validar URLs e identificadores de Python.	
Pruebas de aceptación: • Dada una URL se debe verificar que ésta sea una URL válida o inválida. • Dado un identificador de Python (nombre de variable) se debe verificar que éste sea un identificador válido o inválido.	

	Tarea
Número tarea: 5	**Número historia:** 1.1
Nombre tarea: Clase de peticiones	
Tipo de tarea : Desarrollo	
Fecha inicio: 26 de Octubre de 2011	**Fecha fin:** 2 de Noviembre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase de peticiones del Web Scraper, la cual se encarga de realizar los distintos tipos de peticiones (HEAD, GET), así como de solicitar el archivo robots.txt y analizarlo. También se encarga de analizar el código de respuesta HTTP de la	

petición.

Esta clase contará con los siguiente métodos:

- Knock, el cual verifica, a través del archivo robots.txt, si el Web Scraper tiene acceso al recurso que se está solicitando. De igual manera, verifica si existe la directiva crawl-delay. De existir la directiva crawl-delay se utiliza el tiempo especificado en ella como demora entre peticiones, de lo contrario se utiliza un (1) segundo (negociable). Si la petición del archivo robots.txt falla, se debe reintentar un máximo de tres veces antes de ignorar el archivo robots.txt.
- Make, la cual realizará la petición (HEAD o GET) correspondiente al servidor. De ser una petición GET y el contenido devuelto un archivo HTML/XML, se debe codificar el contenido de la respuesta utilizando el formato de codificación especificado en el archivo de configuración (tarea número 1) y luego limpiarse (tidy) para obtener un código HTML/XML válido. Para ambos tipos de peticiones se crea un diccionario con las cabeceras HTTP devueltas. Si se encuentra un código de respuesta HTTP mayor que 299 y menor que 600 se levanta el error respectivo.

Esta clase es utilizada por la clase Scraper.

Pruebas de aceptación:

- El método Knock debe retornar False en caso de que el archivo /robots.txt especifique de manera explícita que el Web Scraper no tiene permiso para solicitar el recurso en cuestión. Debe retornar True en caso contrario.
- Los tres intentos recursivos en caso de error del método Knock deben completarse de manera exitosa.
- Se debe verificar que de existir la directiva crawl-delay ésta sea acatada.
- Al realizar una petición (HEAD o GET) por parte del método Make, se debe verificar que las cabeceras HTTP devueltas por parte del servidor sean almacenadas en un diccionario, arreglo asociativo o cualquier estructura de datos similar.
- Al realizar una petición HEAD se debe verificar que el cuerpo de la respuesta por parte del servidor sea vacía.
- Al realizar una petición GET por parte del método Make, se debe verificar que el código de respuesta HTTP se válido (> 200, < 299), de lo contrario se debe levantar la excepción correspondiente.
- Al realizar una petición GET por parte del método Make se debe verificar que el tipo MIME del recurso en cuestión sea un tipo MIME de HTML/XML válido, de lo contrario se debe descartar el recurso.
- Al realizar una petición GET por parte del método Make, se debe verificar la codificación de caracteres del recurso y decodificarla/codificarla según sea necesario.
- Al realizar una petición GET por parte del método Make, se debe limpiar el código

HTML/XML (tidy), independientemente de su validez.

	Tarea
Número tarea: 6	**Número historia:** 1.1
Nombre tarea: Clase XPath	
Tipo de tarea : Desarrollo	
Fecha inicio: 3 de Noviembre de 2011	**Fecha fin:** 18 de Noviembre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase XPath para "modularizar" la evaluación de las expresiones de XPath y tener un mayor control sobre futuras versiones. Esta clase tiene un método find, el cual se encarga de evaluar la expresión de XPath en un contexto dado, retornando el resultado en caso de haberlo o Empty en caso contrario. Esta clase es utilizada por la clase Scraper.	
Pruebas de aceptación: • Verificar que la evaluación de las expresiones de XPath en su contexto correspondiente sea correcta. • Verificar que a la hora de no encontrar resultados se retorne "None".	

	Tarea
Número tarea: 7	**Número historia:** 1.1
Nombre tarea: Clase Scraper	
Tipo de tarea : Desarrollo	
Fecha inicio: 17 de Octubre de 2011	**Fecha fin:** 18 de Noviembre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

Diseñar e implementar la clase principal del Web Scraper, la cual, dada una URL cualquiera, se encarga de extraer el contenido HTML/XML del mismo (tarea número 5) y de resolver las expresiones de XPath especificadas en el archivo de configuración del Web Scraper (tarea número 6).

Por cada URL se realizan de dos (2) a tres (3) peticiones HTTP; una petición al archivo robots.txt (a través de la tarea número 5), la cual permite saber si el Web Scraper tiene permiso para solicitar el recurso en cuestión; una petición HEAD, la cual indica si el recurso ha cambiado con respecto a la última visita y una petición GET que se realiza únicamente si el recurso ha cambiado desde la última visita y es HTML/XML.

Una vez obtenido el recurso y éste haya sido codificado y limpiado (tarea número 5), se procede a evaluar las expresiones de XPath en el contexto correspondiente (tarea número 6), almacenando los resultados en variables conocidas para el usuario y con los nombres respectivos de las expresiones de XPath encontradas en el archivo XML de configuración (tarea número 1) y retornando el código HTTP devuelto por el servidor.

Esta clase cuenta con un único método run, el cual da inicio a todo lo antes mencionado, retornando False en caso de ocurrir un error interno durante la ejecución del programa o retornando el código HTTP devuelto por el servidor como se mencionó anteriormente.

Esta clase utiliza a todas las demás clases previamente mencionadas.

Observaciones:

La respuesta del archivo robots.txt debe asociarse a un host, almacenarse y refrescarse periódicamente, de esta manera se evitará realizar peticiones innecesarias al visitar URLs de un mismo host de manera consecutiva o de manera intermitente.

Esta clase no se encarga de la recuperación de errores, de esto se encarga la clase que implemente a la clase Scraper en base al código HTTP devuelto, cabeceras, etc.

Pruebas de aceptación:

- Todas las pruebas de aceptación antes mencionadas.
- Verificar que si un recurso tiene una fecha de actualización previa a la fecha de la última visita al recurso (provista por el usuario) éste sea ignorado.
- Verificar que si un recurso no es un recurso HTML/XML éste sea ignorado.
- Verificar que se creen las variables que contienen los resultados de las evaluaciones de las expresiones de XPath.
- Verificar que el código HTTP devuelto por el método run sea el esperado, tanto en caso de éxito como en caso de fallo.

2.6.1.5. Rotaciones

Dado el número de programadores en el equipo, se decidió no programar rotaciones durante el desarrollo de las tareas. Ambos programadores trabajaron de manera conjunta.

2.6.1.6. Reuniones

Durante esta primera iteración se llevaron a cabo reuniones cortas y de manera regular, antes de comenzar con cada una de las tareas de desarrollo, en las que se discutían las tareas a desarrollar, problemas que se podrían presentar, o que se hayan presentado, y sus posibles soluciones.

La duración de estas reuniones estuvo comprendida entre diez (10) y quince (15) minutos.

2.6.2. Diseño

Como primer punto durante esta etapa de diseño se acordó la utilización de Python como lenguaje de programación.

Para la implementación de la historia de usuario número 1.1 se optó por estructurar el proyecto subdirectorios, donde el subdirectorio /config contendría todos los archivos XML de configuración mientras que el subdirectorio /src contendría el código fuente y binario del proyecto.

El subdirectorio /src, a su vez, estaría compuesto por el subdirectorio core, el cual contendría todas las clases a implementar durante esta primera iteración.

Se propuso la realización de seis (6) clases, producto de la planificación de las tareas de desarrollo. Estas clases son: Validation, Log, Request, XPath, Messages y Scraper.

Las clases Validation, Log, Request y XPath fueron ubicadas en el subdirectorio helpers, ubicado a su vez en el subdirectorio /src/core. Las clases Messages y Scraper fueron ubicadas en el subdirectorio /src/core.

Se decide que todo aquello contenido en el subdirectorio /src/core sería totalmente independiente y no sería influenciado por la implementación de clases futuras en próximas iteraciones.

A continuación se muestra un esquema general del diseño del funcionamiento de éste Web Scraper de uso general (figura 6.1).

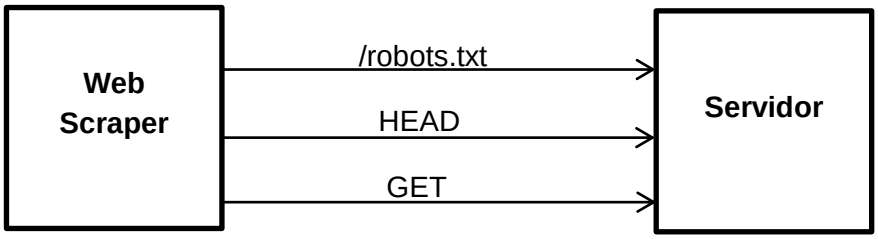

Figura 6.1: Esquema general del diseño del funcionamiento del Web Scraper de uso general.

2.6.2.1. Metáfora del sistema

Se decide escoger “Web Scraper de HTML/XML de uso general utilizando XPath” como metáfora para esta primera iteración.

2.6.2.2. Diagrama de clases

A continuación se muestran las distintas clases que se desarrollarán durante esta primera iteración.

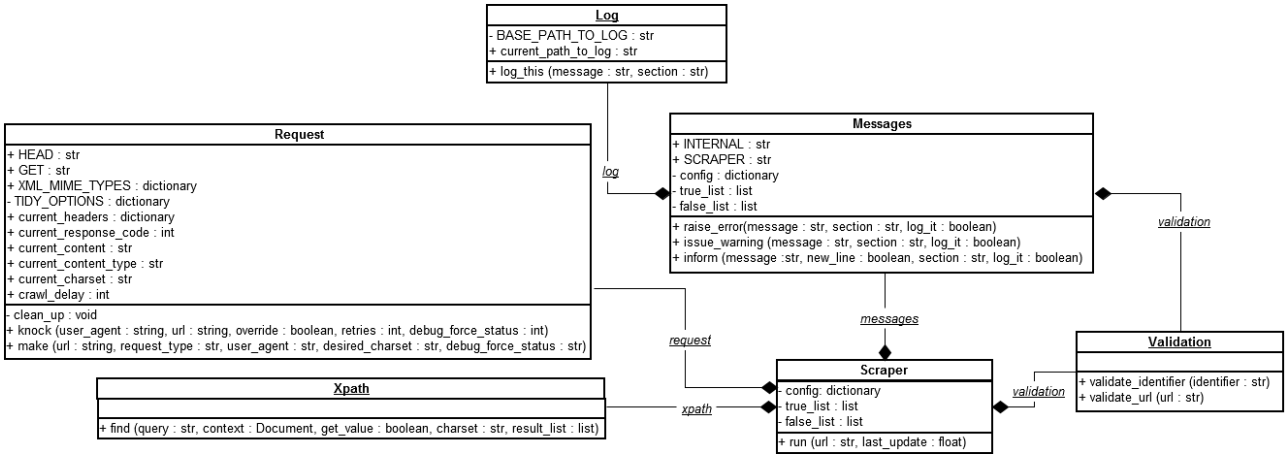

Figura 6.2: Diagrama de clases de la primera iteración.

Glosario de términos:

- **Log:** clase encargada de registrar los mensajes proporcionados por la clase Messages en los registros del sistema.
- **Messages:** clase encargada de cargar los mensajes del sistema del archivo XML de mensajes, mostrarlos y ordenar su registro a la clase Log.
- **Request:** clase encargada de realizar peticiones HTTP y de verificar el archivo /robots.txt para determinar si se cuenta con los permisos necesarios.
- **Scraper:** clase principal del Web Scraper de uso general.
- **Validation:** clase encargada de validar URLs e identificadores de Python.

- **XPath:** clase encargada de resolver las expresiones de XPath en un contexto dado.

2.6.2.3 Soluciones Spike

Durante esta etapa se desarrollaron una serie de scripts con el fin de familiarizarse con el uso de las distintas librerías a utilizar durante el proyecto (httplib, pymongo, py-dom-xpath, uTidyLib).

Estos scripts fueron creados a través de la consola de Python por lo que fueron descartados una vez obtenidos los conocimientos deseados.

2.6.3. Codificación

Previo a la etapa de codificación se acordó la utilización de un repositorio público, en este caso GitHub, el cual contiene todo lo desarrollado durante el proyecto. Los commit se realizaron al finalizar cada sesión de programación, siempre y cuando se aprobaran todas las pruebas unitarias y se tuviera una entrega totalmente funcional.

Se acordó seguir los estándares de programación "pytónica" (http://www.python.org/dev/peps/pep-0008/) hasta cierto punto, utilizando nombres de variables informativos (sin importar su longitud) y separados por "_". También se acordó el uso de listas por definición, evitar el uso innecesario de whiles y fors y romper aquellas líneas que fuesen extremadamente largas utilizando "()" así como una indentación apropiada.

La codificación se llevó a cabo en pareja, partiendo de la tarea número 2, siguiendo con la tarea número 3 y posteriormente implementando las demás tareas por orden de numeración.

Para el archivo de XML de mensajes se utilizó la siguiente estructura:

```
<messages>
    <message name=”mensaje”>Hola mundo</message>
    …

</message>
```

Donde name es el nombre de la variable de clase de la clase Messages cuyo valor sería "Hola Mundo". Adicionalmente se agregó soporte para el formato de cadenas de caracteres utilizando el carácter "%".

Para el archivo de configuración del Web Scraper de uso general se utilizó la siguiente estructura:

```
<scraper>
    <general>
```

```
        <user_agent>Prueba</user_agent>
        <charset>utf8</charset>
    </general>
    <xpath>
        <queries hosts="es.wikipedia.org, en.wikipedia.org">
            <query name="prueba" context="none">/html</query>
            <query name="prueba2" context=" prueba"
get_value=”false”>/ul/li</query>
        </queries>
    </xpath>
</scraper>
```

Donde queries representa el conjunto de consultas de XPath y queries.hosts representa los dominios (hosts) en los que dichas consultas serán evaluados. Esto nos permite tener consultas independientes para dominios distintos alcanzando la “generalización” buscada.

Query representa una expresión de XPath, donde name es el nombre de la variable que contendrá el resultado de evaluar la expresión de XPath en el contexto determinado por context, es importante destacar que name debe ser un identificador de Python válido.

Context debe ser un name previamente definido, none o un atributo no existente. Si context es none o un atributo no existente, la expresión de XPath se evalúa en todo el documento HTML/XML obtenido, de lo contrario se evalúa en el resultado de evaluar el query identificado por name en su contexto determinado.

Por ejemplo, asumamos que tenemos dos query “prueba” y “prueba2”, como se muestra a continuación:

```
<query name="prueba" context="none">/html</query>
<query name="prueba2" context=" prueba" get_value=”false”>/ul/li</query>
```

Donde “prueba” generará la variable de clase “prueba” de la clase Scraper (Scraper.prueba) y su contenido será el elemento “html” de un documento HTML cualquiera. “prueba2”, por su parte, generará la variable de clase “prueba2” de la clase Scraper (Scraper.prueba2) y su contenido serán todos los elementos “li” encontrados en el elemento “html” de “prueba”.

Get_value indica si se quiere obtener el valor del nodo de texto del elemento HTML/XML o el elemento como tal.

Una vez especificada la estructura de los documentos XML de configuración y de mensajes, así como determinados sus roles dentro de la aplicación, se procedió al desarrollo de las clases. Para el almacenamiento de la información obtenida de los archivos de XML de

configuración y de mensajes se optó por el uso de diccionarios, estructura de datos similar a los objetos en JavaScript (JSON).

Se decidió trabajar con la librería xml.dom.minidom para tratar los archivos XML de configuración y de mensajes, esta librería también sirvió para tratar los documentos HTML/XML obtenidos de la Web, así como para trabajar con los contextos de los querys de XPath.

Para la clase Messages se crearon los métodos raise_error, issue_warning e inform. El método raise_error es utilizado para levantar errores, éste informa al usuario acerca del error, ordena su registro y detiene la ejecución del Web Scraper. Los métodos issue_warning e inform son similares a raise_error, con la diferencia de que éstos no detienen la ejecución del Web Scraper. Raise_error e issue_warning escriben los mensajes a través de stderr, mientras que inform lo hace a través de stdout.

Para realizar las peticiones HTTP se utilizó la librería httplib, la cual ofrece la posibilidad de realizar peticiones HEAD reales, a diferencia de las librerías urllib o pycurl. Para verificar los permisos y directivas del archivo /robots.txt se optó por utilizar la librería nativa de Python RobotFileParser.

A la hora de solicitar el recurso /robots.txt y en caso de que ocurra algún error recuperable (408, 500 y 503), se reintenta la petición un máximo de tres veces antes de ignorar dicho recurso. Si el error no es recuperable simplemente se ignora el recurso, mientras que si el error es un error de DNS, se cancela la petición y se finaliza la ejecución del Web Scraper ya que muy probablemente las peticiones subsiguientes fallarán. De obtener de manera satisfactoria el recurso, se resuelven los permisos y directivas, ignorando la URL o continuando con la ejecución del Web Scraper según sea el caso.

El tiempo entre peticiones es de un (1) segundo, a menos que se encuentre la directiva crawl-delay en el recurso /robots.txt y se indique el tiempo entre peticiones deseado.

Al realizar una petición HEAD se verifica el código de respuesta HTTP obtenido, levantando una excepción a la hora de encontrar un código de respuesta HTTP mayor que 299 y menor que 600. Si se obtiene un código de respuesta HTTP mayor que 200 y menor que 299 se considera una respuesta libre de errores y se procede a analizar las cabeceras HTTP devueltas por el servidor. Para verificar la última fecha de modificación de los recursos solicitados se utilizó la fecha devuelta por la cabecera HTTP last-modified (en formato RFC 2822), la cual luego fue convertida a UNIX timestamp para su comparación con una fecha en particular proporcionada por el usuario. De ser un recurso “fresco” se continúa con la ejecución del Web Scraper, de lo contrario se ignora la URL.

Al realizar una petición GET, se verifica el tipo MIME del recurso y se verifica que sea un tipo MIME de XML válido, ignorando la URL o continuando con la ejecución del Web Scraper según sea el caso.

Para resolver las expresiones de XPath se utilizó la librería py-dom-xpath, compatible con la librería xml.dom.minidom.

Al finalizar la ejecución del Web Scraper, éste creará las variables de clase especificadas en el archivo XML de configuración y retornará el código de respuesta HTTP obtenido por parte del servidor. En caso de haber ocurrido algún error durante la ejecución retornará False y no se creará ninguna variable de clase.

2.6.4. Prueba

Durante esta etapa se crearon una serie de pruebas unitarias, utilizando el framework unittest de Python, basadas en las pruebas de aceptación especificadas en las tareas de desarrollo. A continuación se puede observar la aprobación de todas las pruebas unitarias (un total de siete (7) pruebas) al finalizar esta primera iteración.

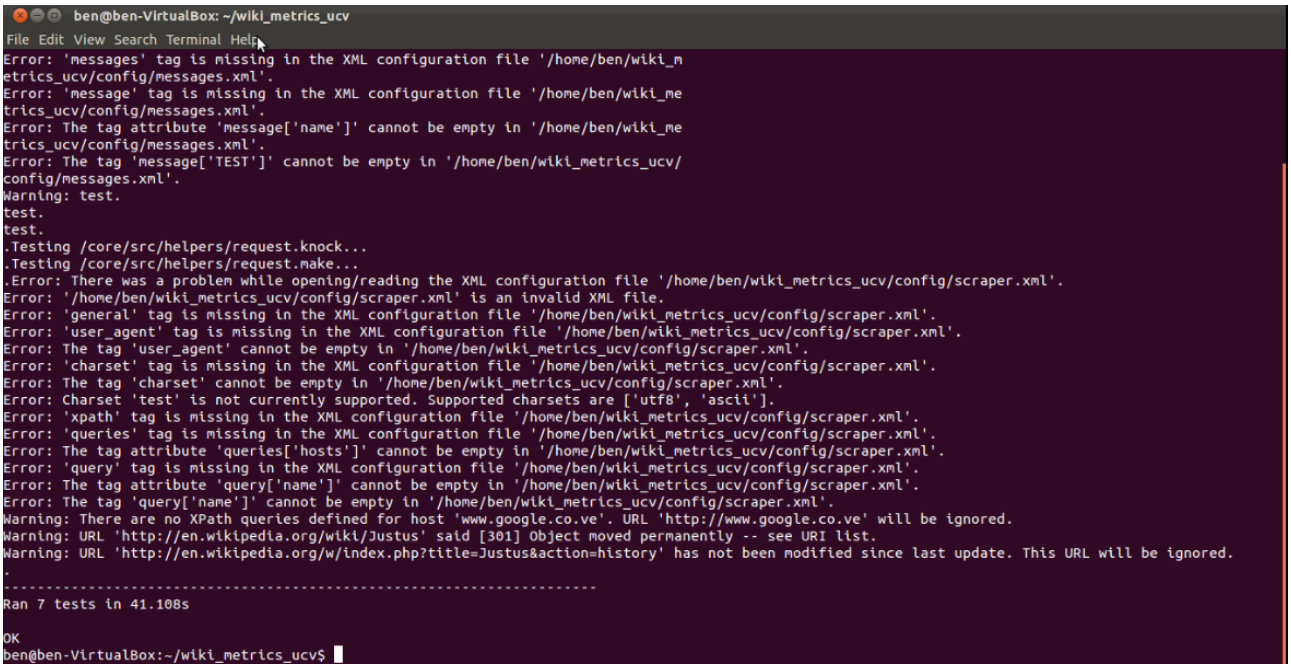

Figura 6.3: Resultado de las pruebas unitarias de la primera iteración (los errores mostrados son esperados)..

2.7. Segunda Iteración

Durante la segunda iteración se adaptó el Web Scraper de uso general desarrollado en la primera iteración, denominado XCraper de ahora en adelante, a la solución general del problema, es decir, la obtención de datos de las revisiones de los historiales de artículos hospedados en wikis basados en MediaWiki.

2.7.1. Planificación

2.7.1.1. Historias de usuario

<table>
<tr><th colspan="3">Historia de Usuario</th></tr>
<tr><td>Número: 1.2</td><td colspan="2">Usuario: Final</td></tr>
<tr><td colspan="3">Nombre historia: Web Scraper de MediaWiki</td></tr>
<tr><td colspan="2">Prioridad en negocio:

Alta</td><td>Riesgo en desarrollo:

Media</td></tr>
<tr><td colspan="3">Programador responsable: Benjamin Worwa – Roberto D’Apuzzo</td></tr>
<tr><td colspan="3">Descripción:

Es necesario crear una clase que implemente la clase “Web Scraper de uso general” (historia de usuario 1.1); que trate, normalice y guarde toda la información extraída, de una URL de un historial de un artículo de Wikipedia (o cualquier otro wiki basado en MediaWiki), en una base de datos. Esta clase también se debe encargar de decidir cual URL (historial de un artículo) es visitada en un momento dado (tomada de la base de datos).

A su vez, esta clase es la encarga de la recuperación de posibles errores que arroje la clase “Web Scraper de uso general” (peticiones fallidas, timeouts, etc.).</td></tr>
<tr><td colspan="3">Observaciones:

• La base de datos a utilizar será MongoDB.
• Toda la información obtenida, tratada y normalizada debe estar claramente asociada a una URL a la hora de almacenarla en la base de datos.
• Es necesario poder recorrer las distintas páginas de un historial de un artículo de manera automática.
• Es necesario crear mecanismos de recuperación de errores coherentes y eficaces.</td></tr>
</table>

2.7.1.2. Plan de entrega

Para esta segunda iteración se acordó el desarrollo de la historia de usuario número 1.2 y realizar una primera entrega funcional de la historia de usuario número 1.

2.7.1.3. Velocidad del proyecto

Se acordó por el grupo de desarrollo que esta iteración tendría un tiempo de entrega estimado de tres (3) semanas. Esto basado en el tiempo que tomó finalizar la primera iteración.

El cliente indicó que no tenía una fecha de entrega determinada y que podíamos tomarnos todo el tiempo que fuese necesario.

2.7.1.4. Plan de iteraciones

A continuación se muestra el conjunto de tareas de desarrollo en las que se dividió la historia de usuario número 1.2.

<table>
<tr><th colspan="2">Tarea</th></tr>
<tr><td>Número tarea: 1</td><td>Número historia: 1.2</td></tr>
<tr><td colspan="2">Nombre tarea: Clase Mongo</td></tr>
<tr><td>Tipo de tarea : Desarrollo</td><td></td></tr>
<tr><td>Fecha inicio: 18 de Noviembre de 2011</td><td>Fecha fin: 23 de Noviembre de 2011</td></tr>
<tr><td colspan="2">Programador responsable: Benjamin Worwa – Roberto D’Apuzzo</td></tr>
<tr><td colspan="2">Descripción:

Diseñar e implementar la clase Mongo, la cual permite la interacción entre el Scraper y la base de datos MongoDB. Al instanciar esta clase se debe cargar un archivo XML de configuración, similar al archivo de configuración de XCraper, en el cual se especificará toda la información necesaria para conectarse a la base de datos (host, puerto, etc.).

Esta clase tiene una serie de métodos que permiten la inserción, actualización y eliminación tanto de artículos como de historiales almacenados en la base de datos.</td></tr>
<tr><td colspan="2">Pruebas de aceptación:

• Se debe verificar la integridad y coherencia del archivo XML de configuración.
• Los métodos de inserción, actualización y eliminación de datos deben generar resultados coherentes y correctos.</td></tr>
</table>

Tarea	
Número tarea: 2	**Número historia:** 1.2
Nombre tarea: Clase Wikimetrics	
Tipo de tarea : Desarrollo	
Fecha inicio: 18 de Noviembre de 2011	**Fecha fin:** 27 de Noviembre de 2011
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase Wikimetrics, clase que hace uso de la clase Scraper (XCraper) y procesa los datos obtenidos por ésta. Una vez que la clase Scraper evalúa las expresiones de XPath en el historial de un artículo y obtiene los datos de interés, la clase Wikimetrics se encarga de normalizar estos datos y almacenarlos en la base de datos. A su vez, la clase Wikimetrics se encarga de verificar si existe más de una página de revisiones en el historial de un artículo y de visitarlas todas. A su vez, se encarga de la recuperación de errores que puedan ocurrir durante las peticiones HTTP (timeouts, recursos no encontrados, etc.) y toma las decisiones pertinentes según sea el caso. De igual manera que con las clases principales del proyecto, la clase Wikimetrics carga un archivo XML de configuración en el que se especifican la cantidad de revisiones por historial a solicitar, así como cualquier otro parámetro necesario.	
Pruebas de aceptación: • Se debe verificar la integridad y coherencia del archivo XML de configuración. • Se debe verificar que los datos obtenidos de los historiales sean correctos.	

2.7.1.5. Rotaciones

Dado el número de programadores en el equipo, se decidió no programar rotaciones durante el desarrollo de las tareas. Ambos programadores trabajaron de manera conjunta.

2.7.1.6. Reuniones

Se mantuvo el hábito de las reuniones de la primera iteración.

2.7.2. Diseño

Para mantener la integridad de la estructura de directorios con la que se trabajó durante todo el proyecto, se decide que todo lo que fuese a ser desarrollado durante esta segunda

iteración sería ubicado en e subdirectorio /src/usr, mientras que los archivos de configuración serían ubicados en el subdirectorio común de configuración /config.

Se propone el desarrollo de dos (2) clases, la clase Mongo y la clase Wikimetrics, cuyas funcionalidades se encuentran descritas en la planificación de las tareas de desarrollo.

Para la base de datos, se propone la siguiente estructura para los documentos de los artículos con la ayuda del cliente:

```
Articulos =  {
    _id : URL,
    priority : 0.5,
    timestamp : 1234567890,
    md5: abc123abc123abc123abc123abc123ab
}
```

Donde priority será la piedra angular de la política de re-visita de los historiales, timestamp permite saber la última fecha de visita a un historial y md5 nos permitirá saber si un historial ha cambiado si contenido (revisiones) a lo largo del tiempo.

Para los historiales se propuso:

```
Historiales = {
    _id : ObjetcId,
    mediawiki_id : 456987,
    history_id: URL,
    date : dd-mm-yyyy hh:mm,
    user : name,
    minor : true,
    size : 12345, //bytes
    comment : Reverted edits by Maunus to last version by 134.50.208.167
}
```

A través del cual se almacenan todos los posibles datos que se pueden encontrar en las revisiones de un historial.

Ya que las métricas solicitadas por el cliente son conteos, promedios, máximos y mínimos y proporciones, se consideró innecesario el almacenamiento de éstos en los documentos de los artículos.

2.7.2.1. Metáfora del sistema

Se decide escoger “Herramienta de recuperación y almacenamiento de información de los historiales de artículos hospedados en wikis basados en MediaWiki” como metáfora para esta primera iteración.

2.7.2.2. Diagrama de clases

A continuación se muestran las distintas clases que se desarrollarán durante esta segunda iteración.

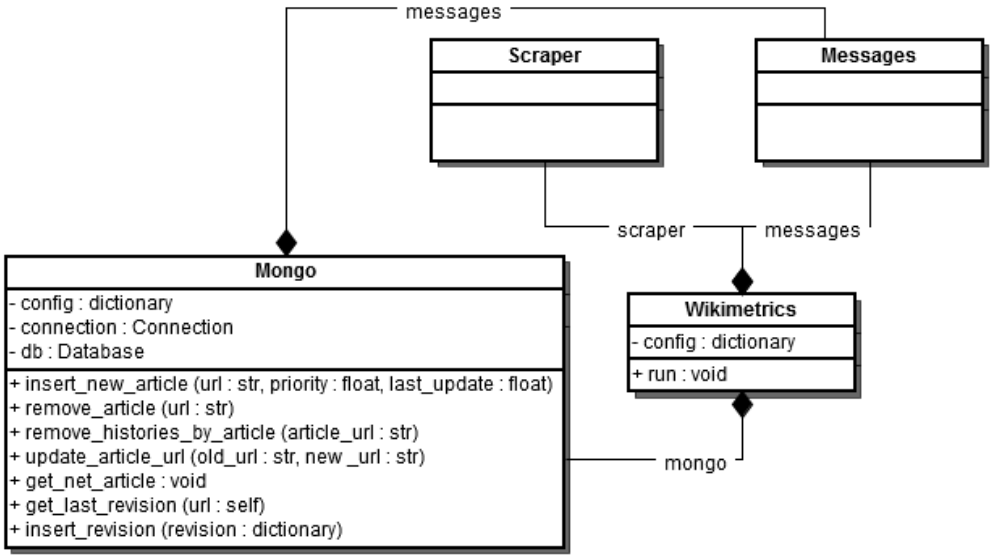

Figura 7.1: Diagrama de clases de la segunda iteración.

Glosario de términos:

- **Mongo:** clase encargada de interactuar con la base de datos.
- **Wikimetrics:** clase encargada de normalizar y almacenar los datos recuperados de la Web, ordenar la visita a las distintas páginas de revisiones de un historial en particular y de la recuperación de errores que puedan ocurrir durante las peticiones HTTP.

2.7.2.3. Soluciones Spike

Se llevaron a cabo un conjunto de soluciones spike para familiarizarse con el uso de la librería pymongo a ser utilizada por la clase Mongo. Éstas fueron descartadas una vez obtenido el conocimiento buscado.

2.7.3. Codificación

La implementación de la clase Mongo fue relativamente sencilla dada la experiencia obtenida durante la primera iteración. Se hizo uso de la librería pymongo para trabajar con MongoDB.

Se crearon una serie de métodos para interactuar con la base de datos como:

- insert_new_article y remove_article, los cuales permiten insertar y eliminar, respectivamente, un artículo a la colección “artículos”.
- remove_article, el cual permite eliminar un artículo en particular.
- remove_histories_by_article, el cual permite eliminar todas las revisiones asociadas a un artículo en particular.
- update_article_url, el actualiza la URL de un artículo en particular y actualiza todas las referencias a dicha URL en la colección “historiales”.
- get_next_article, el cual retorna la URL de un artículo con la mayor prioridad.
- get_last_revision, el cual retorna el id de la última revisión de un artículo en particular.
- insert_revision, el cual inserta una nueva revisión en la colección “historiales”.

La clase Mongo fue desarrollada en paralelo con la clase Wikimetrics.

La clase Wikimetrics posee un único método recursivo run, similar al método run de la clase Scraper (no recursivo), el cual:

- Dada una URL de un artículo, se obtiene el ID de su última revisión. En caso de no existir una última revisión (primera visita) el ID es 0.
- Se llama al método run de la clase Scraper, con la URL del artículo y su timestamp (última visita). De ser la primera vez que se visita el timestamp es cero (0).
- Se obtiene el código de respuesta HTTP por parte del método run de la clase Scraper y se evalúa como se indica a continuación:
    - Si el código es 301 (Moved permanently), se actualiza la URL del artículo (update_article_url) con el contenido de la cabecera HTTP Location y se actualizan a su vez todas sus referencias en la colección “historiales”. Se realiza una llamada recursiva con la nueva URL y se comienza el proceso nuevamente.
    - Si el código es 302, 303 o 307 (Found, See other), se realiza una llamada recursiva con la URL provista por la cabecera HTTP Location y con la URL canónica original, comenzando el proceso nuevamente. La URL canónica nos permite asociar cualquier URL a una URL de un artículo, útil para las páginas de las revisiones de los historiales y redirecciones temporales.
    - Si el código es 408, 500, 503 (Request timed out, Internal server error, Service unavailable), se reintenta la petición un máximo de tres veces antes de ignorar la URL.
    - Si el código es 410 (Gone), se elimina el artículo y sus revisiones.
    - Si el código es 200, se almacenan en la base de datos aquellas revisiones con un ID mayor que el ID de la última revisión obtenido en el primer paso y se van contando, creando así tres posible escenarios:
        - Si la cuenta final es 0, se le informa al usuario que no ha habido cambio alguno.

- Si la cuenta final es menor que la cantidad de revisiones por historial a solicitar, se le indica al usuario cuántas nuevas revisiones se han encontrado.
- Si la cuenta final es igual que la cantidad de revisiones por historial a solicitar, se verifica si el artículo tiene una página siguiente. De no tener una página siguiente ocurre lo mismo que en el punto anterior, en caso contrario se realiza una llamada recursiva con la URL de la página siguiente y la URL canónica original.

A su vez se propuso cambiar la estructura de los documentos de los historiales por la siguiente:

```
Historiales = {
    _id : 456987,
    article: URL,
    date : timestamp,
    user : name,
    minor : true,
    size : 12345, //bytes
    comment : Reverted edits by Maunus to last version by 134.50.208.167
}
```

Esta modificación se debe a que se determinó que el “mediawiki_id” es único para cada revisión y trabajar fechas como marcas de tiempo es mucho más ventajoso.

2.7.4. Prueba

De igual manera que en la primera iteración, se crearon una serie de prueba unitarias utilizando el framework unittest de Python para la clase Mongo. Estas pruebas corresponden a las pruebas de aceptación encontradas en la planificación de las tareas de desarrollo.

Sin embargo, debido a que los resultados devueltos por la clase Wikimetrics son dinámicos y desconocidos (dependen del contenido de un historial en un momento dado), se decidió no utilizar pruebas automatizadas y se recurrió a pruebas manuales.

A continuación se puede observar la aprobación de todas las pruebas unitarias, tanto las de la primera iteración como las de la segunda, por parte del sistema (un total de nueve (9) pruebas).

```
.Testing /core/src/helpers/request.make...
.Error: There was a problem while opening/reading the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: '/home/ben/wiki_metrics_ucv/config/scraper.xml' is an invalid XML file.
Error: 'scraper' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: 'general' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: 'user_agent' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: The tag 'user_agent' cannot be empty in '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: 'charset' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: The tag 'charset' cannot be empty in '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: Charset 'test' is not currently supported. Supported charsets are ['utf8', 'ascii'].
Error: 'xpath' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: 'queries' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: The tag attribute 'queries['hosts']' cannot be empty in '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: 'query' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: The tag attribute 'query['name']' cannot be empty in '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Error: The tag 'query['name']' cannot be empty in '/home/ben/wiki_metrics_ucv/config/scraper.xml'.
Warning: There are no XPath queries defined for host 'www.google.co.ve'. URL 'http://www.google.co.ve' will be ignored.
Warning: URL 'http://en.wikipedia.org/wiki/Justus' said [301] Object moved permanently -- see URI list.
Warning: URL 'http://en.wikipedia.org/w/index.php?title=Justus&action=history' has not been modified since last update. This URL will be ignored.
.Testing /src/usr/wikimetrics
Error: There was a problem while opening/reading the XML configuration file '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml'.
Error: '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml' is an invalid XML file.
Error: 'wikimetrics' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml'.
Error: 'revisions_limit' tag is missing in the XML configuration file '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml'.
Error: The tag 'revisions_limit' cannot be empty in '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml'.
Error: Revisions limit must be an integer value greater than 0 in '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml.
Error: Revisions limit must be an integer value greater than 0 in '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml.
Error: Revisions limit must be an integer value greater than 0 in '/home/ben/wiki_metrics_ucv/config/wikimetrics.xml.
.
----------------------------------------------------------------------
Ran 9 tests in 30.301s

OK
ben@ben-VirtualBox:~/wiki_metrics_ucv$
```

Figura 7.2 (a): Resultado de las pruebas unitarias de la primera iteración (los errores mostrados son esperados).

De igual manera, podemos observar la ejecución exitosa de la clase Wikimetrics...

```
ben@ben-VirtualBox:~/wiki_metrics_ucv$ python src/main.py
Visiting URL 'http://en.wikipedia.org/w/index.php?title=Computer_science&action=history'. Checkpoint set at revision '465178212'.
18 new revision(s) found at URL 'http://en.wikipedia.org/w/index.php?title=Computer_science&action=history'.
ben@ben-VirtualBox:~/wiki_metrics_ucv$




ben@ben-VirtualBox:~/wiki_metrics_ucv$ python src/main.py
Visiting URL 'http://en.wikipedia.org/w/index.php?title=Computer_science&action=history'. Checkpoint set at revision '469926640'.
No new revisions found since revision '469926640' at URL 'http://en.wikipedia.org/w/index.php?title=Computer_science&action=history'.
ben@ben-VirtualBox:~/wiki_metrics_ucv$
```

Figura 7.2 (b): Resultado de las pruebas unitarias de la primera iteración.

… verificando con la base de datos y el historial.

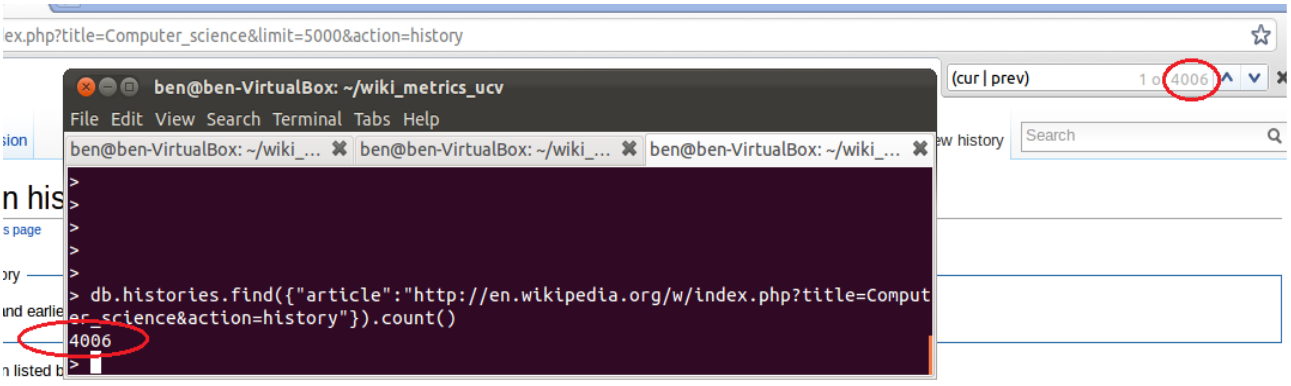

Figura 7.2 (c): Resultado de las pruebas unitarias de la primera iteración.

2.8. Tercera Iteración

Luego de finalizar la segunda iteración y durante la demostración al cliente, se encontró el siguiente problema:

- Al ocurrir un timeout, no es suficiente con sólo reintentar la petición un número finito de veces, es necesaria la implementación de un mecanismo que permita determinar un “punto pendiente” en el historial de revisiones de un artículo y el cual pueda ser resuelto en una futura re-visita.

De igual manera, era necesario culminar la segunda iteración implementando lo siguiente:

- Actualizar la prioridad de un artículo con un valor de cero (0) luego de ser visitado y “procesado”.
- Almacenar el hash MD5 de la primera página del historial de revisiones de un artículo para comprobar futuros cambios en el mismo y ahorrar tiempo en el procesamiento innecesario de éste.
- Actualizar la fecha de la última visita al artículo.

Posteriormente se lleva a cabo una segunda reunión con el cliente en la cual se propuso la creación de una aplicación que incorporase todos los elementos del back-end (Web Scraper, Daemon de prioridades y de URLs) y permitiera el control de los mismos a través de comandos por consola, lo cual aceptó. El cliente también especificó la necesidad, de haber tiempo, de crear una versión del Web Scraper y de la herramienta de prioridades a modo de daemon, lo cual facilitaría su tarea como administrador de servidor.

Es por esto que se deja de lado el concepto de Daemon de prioridades, que se venía tratando a lo largo de las iteraciones, y se remplaza por el concepto de Herramienta de prioridades. Esto último no representa contratiempo alguno ya que las prioridades calculadas, como se explicó durante la primera iteración, no son acumulativas (no dependen de una prioridad previa) sino absolutas y por lo tanto no se depende de un proceso continuo para cálculo.

2.8.1. Planificación

2.8.1.1. Historias de usuario

	Historia de Usuario
Número: 1.2.1	**Usuario:** Final
Nombre historia: Web Scraper de MediaWiki	
Prioridad en negocio:	**Riesgo en desarrollo:**

Alta	Medio
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: (Refactorización) • Al ocurrir un timeout, no es suficiente con sólo reintentar la petición un número finito de veces, es necesaria la implementación de un mecanismo que permita determinar un "punto pendiente" en el historial de revisiones de un artículo y el cual pueda ser resuelto en una futura re-visita. • Actualizar la prioridad de un artículo con un valor de cero (0) luego de ser visitado y "procesado". • Almacenar el hash MD5 de la primera página del historial de revisiones de un artículo para comprobar futuros cambios en el mismo y ahorrar tiempo en el procesamiento innecesario de éste. • Actualizar la fecha de la última visita al artículo.	

Historia de Usuario	
Número: 2	**Usuario:** Final
Nombre historia: Herramienta de prioridades	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Dada una URL (tomada de una base de datos) cuya prioridad es menor que 0.5, se debe calcular y actualizar la prioridad de dicha URL utilizando la siguiente fórmula: 1 – (timestamp de la fecha de la última visita / timestamp de la fecha actual)	
Observaciones: Quedan por definir los intervalos de tiempo adecuados en los que el daemon de prioridades se ejecutará (cada 24 horas, 12 horas, 10 minutos, etc.).	

Historia de Usuario	
Número: 3	**Usuario:** Final
Nombre historia: Herramienta de URLs	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Dada una URL de un artículo de un wiki basado en MediaWiki (tomada de la entrada estándar o por lotes a través de un archivo de texto), se debe generar la URL correspondiente al historial de dicho artículo, normalizar esta nueva URL y almacenarla en la base de datos. Esto corresponde a la política de selección del Web Scraper.	
Observaciones: • La URL puede ser la URL de un artículo o la URL del historial de un artículo. En ambos casos la URL generada debe ser exactamente la misma. • Las URLs son utilizadas por el Web Scraper y la Herramienta de prioridades, por lo que la normalización de éstas es de suma importancia.	

2.8.1.2. Plan de entrega

Para esta tercera iteración se acuerda el desarrollo de la historia de usuario número 1.2.1, 2 y 3.

2.8.1.3. Velocidad del proyecto

Se acuerda por el grupo de desarrollo que esta iteración va a tener un tiempo de entrega estimado de una (1) semana. Esto basado en el tiempo que tomó finalizar la primera iteración y segunda.

El cliente no indicó una fecha de entrega determinada.

2.8.1.4. Plan de iteraciones

A continuación se muestran las tareas a completar durante esta tercera iteración.

Tarea	
Número tarea: 1	**Número historia:** 1.2.1
Nombre tarea: Refactorización de la clase Wikimetrics	
Tipo de tarea : Desarrollo (Refactorización)	
Fecha inicio: 15 de enero de 2012	**Fecha fin:** 18 de enero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

- Al ocurrir un timeout, no es suficiente con sólo reintentar la petición un número finito de veces, es necesaria la implementación de un mecanismo que permita determinar un "punto pendiente" en el historial de revisiones de un artículo y el cual pueda ser resuelto en una futura re-visita.
- Actualizar la prioridad de un artículo con un valor de cero (0) luego de ser visitado y "procesado".
- Almacenar el hash MD5 de la primera página del historial de revisiones de un artículo para comprobar futuros cambios en el mismo y ahorrar tiempo en el procesamiento innecesario de éste.
- Actualizar la fecha de la última visita al artículo.

Pruebas de aceptación:

- Se debe verificar el correcto funcionamiento del mecanismo de recuperación de timeouts.
- Se debe verificar que la prioridad de un artículo luego de su visita y "procesamiento" sea 0.
- Se debe verificar que se almacene el hash MD5 de la primera página del historial de revisiones y la fecha de la última visita al artículo.
- Se debe verificar que los mecanismos que involucran al hash MD5 y la fecha de última visita al artículo funcionen correctamente.

	Tarea
Número tarea: 2	**Número historia:** 2
Nombre tarea: Clase Priority	
Tipo de tarea : Desarrollo	
Fecha inicio: 18 de enero de 2012	**Fecha fin:** 19 de enero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

Diseñar e implementar la clase Priority, la cual se encarga de actualizar las prioridades de los artículos en la base de datos utilizando la siguiente fórmula:

1 – (timestamp de la fecha de la última visita / timestamp de la fecha actual)

Esta actualización se hace de manera periódica y por lotes, con un tiempo entre actualizaciones determinado por el usuario en el archivo XML de configuración de la clase.

Pruebas de aceptación:

- Se debe verificar la integridad y coherencia del archivo XML de configuración.
- Dada una marca de tiempo de última visita y una marca de tiempo actual, se debe verificar que la prioridad obtenida sea correcta.

	Tarea
Número tarea: 3	**Número historia:** 3
Nombre tarea: Clase Urls	
Tipo de tarea : Desarrollo	
Fecha inicio: 19 de enero de 2012	**Fecha fin:** 20 de enero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	

Descripción:

Diseñar e implementar la clase Urls, la cual se encarga de insertar y eliminar URLs,

especificadas por el usuario en la base de datos. Estas URLs son especificadas a través de la entrada estándar (stdin) o a través de un archivo de texto (por lotes). En este último caso, el archivo debe contener una URL por línea.
Pruebas de aceptación: • Se debe verificar que las URLs normalizadas sean insertadas/eliminadas en la base de datos.

	Tarea
Número tarea: 4	**Número historia:** 3
Nombre tarea: Clase Normalization	
Tipo de tarea : Desarrollo	
Fecha inicio: 20 de enero de 2012	**Fecha fin:** 20 de enero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase Normalization, la cual se encarga de normalizar las URLs de artículos hospedados en wikis basados en MediaWiki. Esto es, generar una URL de historial de un artículo dada una URL de un artículo o de historial de un artículo. Independientemente de cuál de las dos URLs antes mencionadas haya sido utilizada el resultado debe ser exactamente el mismo.	
Pruebas de aceptación: • Se debe verificar que las URLs sean normalizadas de manera correcta.	

	Tarea
Número tarea: 5	**Número historia:** 1, 2, 3
Nombre tarea: Clase Console	
Tipo de tarea : Desarrollo	
Fecha inicio: 20 de enero de 2012	**Fecha fin:** 26 de enero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase Console, la cual se encarga de recibir comandos por parte del usuario para interactuar con el Web Scraper, la Herramienta de prioridades y la Herramienta de URLs. El Web Scraper y la Herramienta de prioridades pasan a ser hilos independientes de la aplicación, mientras que la Herramienta de URLs es parte del hilo principal de la aplicación.	
Pruebas de aceptación: • Se debe verificar el correcto funcionamiento de los comandos.	

	Tarea
Número tarea: 4	**Número historia:** 3
Nombre tarea: Clase Threads	
Tipo de tarea : Desarrollo	
Fecha inicio: 30 de enero de 2012	**Fecha fin:** 2 de febrero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: Diseñar e implementar la clase Threads, la cual se encarga de administrar (iniciar, detener) los hilos de ejecución de la aplicación (Web Scraper, Herramienta de prioridades).	

2.8.1.5. Rotaciones

Dado el número de programadores en el equipo, se decide no programar rotaciones durante el desarrollo de las tareas.

2.8.1.6. Reuniones

Se mantuvo el hábito de las reuniones de la primera y segunda iteración.

2.8.2. Diseño

A continuación se presentan la metáfora del sistema para esta tercera iteración, así como el diagrama de clases correspondiente.

2.8.2.1. Metáfora del sistema

Para esta tercera iteración se decidió escoger “Consola administrativa del back-end” como metáfora del sistema.

2.8.2.2 Diagrama de clases

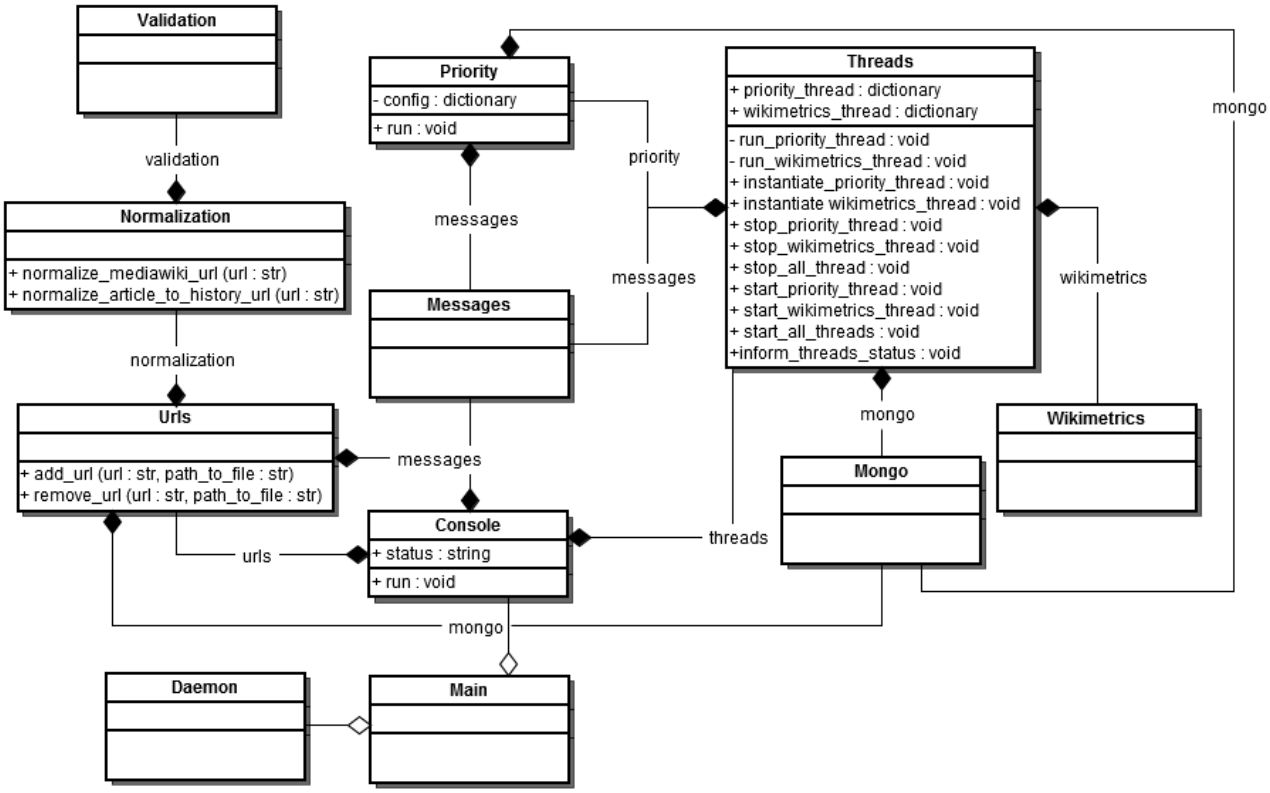

Figura 8.1: Diagrama de clases de la tercera iteración.

Glosario de términos:

- **Priority:** clase encargada actualizar las prioridades de los distintos artículos almacenados en la base de datos.
- **Urls:** clase encargada de insertar y eliminar URLs de artículos en la base de datos.
- **Normalization:** clase encargada de transformar y normalizar una URL de un artículo hospedado en un wiki basado en MediaWiki a la URL de historial de artículo correspondiente.
- **Console:** clase encargada de recibir comandos por parte del usuario y realizar las acciones correspondientes.
- **Threads:** clase encargada de administrar el hilo de ejecución del Web Scraper y de la Herramienta de prioridades.
- **Daemon:** clase ya existente utilizada para ejecutar procesos como demonios/deamons.
- **Main:** clase principal del sistema.

2.8.3 Codificación

Para solucionar el problema del timeout, se implementó un mecanismo que permite crear un punto de control en cualquier página del historial de un artículo. Cada página del historial de un artículo, hospedado en un wiki basado en MediaWiki, tiene asociada una variable offset en su query string. Esta variable offset representa una fecha en particular e indica que todas las revisiones encontradas en la página de dicho historial serán anteriores a dicha fecha. Esto presenta la ventaja de tener puntos de control inmutables.

Al ocurrir un timeout se procede a guardar la URL normalizada con su respectivo offset, denominada como URL pendiente. Al visitar de nuevo dicho artículo se verifica si existe una URL pendiente, de ser así se procede a extraer todas las revisiones faltantes y luego las nuevas revisiones que puedan existir. Al tratar de insertar una revisión ya existente se asume que todas las revisiones anteriores a ésta ya han sido procesadas y se detiene el proceso de obtención de revisiones para dicha URL.

Nunca se obtendrán nuevas revisiones hasta haber obtenido todas las revisiones pendientes.

Cada vez que un artículo es visitado su prioridad cambia a cero (0), sin importar si ha ocurrido algún error durante su visita.

El hash MD5 de la primera página de un historial es almacenado para verificar algún cambio en el mismo, evitando una pérdida innecesaria de tiempo procesando un historial que no ha sido modificado (probablemente algún cambio en el diseño de la página y no en el historial del artículo actualizó la cabecera HTTP last-update).

Para normalizar las URLs éstas son separadas en sus distintos componentes (scheme, host, path, query string, etc.) y la reconstruimos de la siguiente manera:

```
url = scheme + "://" + host + path
```

De existir un query string en la URL, colocamos siempre el título del articulo de primero, seguido por la variable action y, de existir, la variable offset (P. ej. ?title=titulo&action=history[&offset=123456789])

Para generar la URL del historial de un artículo a partir de la URL del artículo, obtenemos el título del artículo del path de la URL y armamos una URL similar a la anterior, sin agregar la variable offset y cambiando el path de la URL por "/w/index.php" (path del historial por defecto en la mayoría de los wikis basados en MediaWiki).

Para agregar/eliminar URLs a la base de datos se especifica la URL o el archivo que la contenga, en caso de especificar la URL ésta es agregada/eliminada. De especificarse un archivo, el cual debe contener una URL por línea, se itera sobre la líneas del archivo, agregando/eliminando las respectivas URLs.

Para actualizar las prioridades de los artículos se toman de la base de datos todos los artículos con prioridad menor que 0.5 y luego se recalculan sus prioridades utilizando la fórmula mencionada en la tarea número 2. Luego de la actualización de las prioridades se procede a suspender la actualización por un período definido de tiempo (especificado por el usuario en el archivo de configuración de la clase Priority).

La clase Threads es la clase encargada de administrar los hilos de ejecución del Web Scraper y la Herramienta de prioridades. Para esto se cuenta con los siguientes métodos:

- run_priority_thread, ciclo infinito que llama al método para actualizar las prioridades de la clase Priority antes mencionado.
- run_wikimetrics_thread, ciclo infinito que obtiene un artículo de la base datos y su fecha de última actualización e invoca al método run de la clase Wikimetrics.
- instantiate_priority_thread, instantiate_wikimetrics_thread, métodos que instancian la clase nativa Thread de Python apuntando los métodos antes descritos respectivamente.
- stop_priority_thread, método que detiene la ejecución del hilo de la Herramienta de prioridades, esperando que finalice su tarea de manera segura.
- stop_wikimetrics_thread, método que detiene la ejecución del hilo del Web Scraper, esperando que finalice su tarea de manera segura.
- stop_all_threads, wrapper para los dos métodos antes descritos.
- start_priority_thread, start_wikimetrics_thread, métodos que invocan al método start de las instancias de Thread antes descritas.
- start_all_threads, wrapper para wrapper para los dos métodos antes descritos.
- inform_threads_status, método que informa acerca del estado de los hilos (en ejecución o detenido).

La consola es un ciclo infinito que espera por un comando del usuario y toma la acción correspondiente. La consola actualmente soporta los siguientes comandos:

- **clear:** similar al comando clear en cualquier distribución de GNU/Linux.

- **exit:** invoca al método stop_all_threads de la clase Threads y culmina la ejecución de la aplicación.
- **start [-a | -p | -w]:** inicia los hilos del Web Scraper y la Herramienta de prioridades, Herramienta de prioridades o Web Scraper de manera correspondiente.
- **stop [-a | -p | -w]:** detiene los hilos del Web Scraper y la Herramienta de prioridades, Herramienta de prioridades o Web Scraper de manera correspondiente.
- **tstatus:** informa acerca del estado de los hilos del Web Scraper y la Herramienta de prioridades (en ejecución o detenido).
- **url [(-add | -rm) (URL | -f /path/to/file.txt)]:** agrega o elimina una URL, o un conjunto de URLs especificadas a través de un archivo de texto (identificado por /path/to/file.txt), en la base de datos.
- **help:** muestra los comandos soportados y su explicación.

La clase Daemon es una clase creada por Sander Macheral, la cual permite ejecutar procesos como demonios/deamons en Python. Esta clase daemoniza los hilos del Web Scraper y de la Herramienta de prioridades, creando un archivo temporal con el PID del proceso asociado a cada nuevo proceso creado para cada hilo. Este PID es usado luego para detener los daemons.

La clase principal toma los siguientes argumentos:

- **-c:** inicia la consola.
- **-d (start | stop) (a | p | w):** inicia/detiene los hilos del Web Scraper y la Herramienta de prioridades, Herramienta de prioridades o Web Scraper de manera correspondiente.

Es importante destacar que no se podrá iniciar la aplicación como daemon siempre que se esté ejecutando la aplicación como consola, ya que no es deseable el solapamiento de los hilos del Web Scraper y Herramienta de prioridades con los daemon equivalentes. De igual manera, no se podrán iniciar los hilos del Web Scraper y Herramienta de prioridades a través de la consola cuando daemons equivalentes se estén ejecutando. Para esto se utilizan archivos de control y los archivos temporales con los PID de los procesos daemonizados.

También se agregaron métodos para capturar las señales SIGTERM, Ctrl+c y Ctrl+d para evitar la finalización abrupta del Web Scraper y la pérdida de datos durante el proceso.

2.8.4. Prueba

A continuación se muestra la aprobación de las catorce (14) pruebas unitarias (nueve de la segunda iteración y cinco de la tercera) implementadas utilizando el framework unittest de Python.

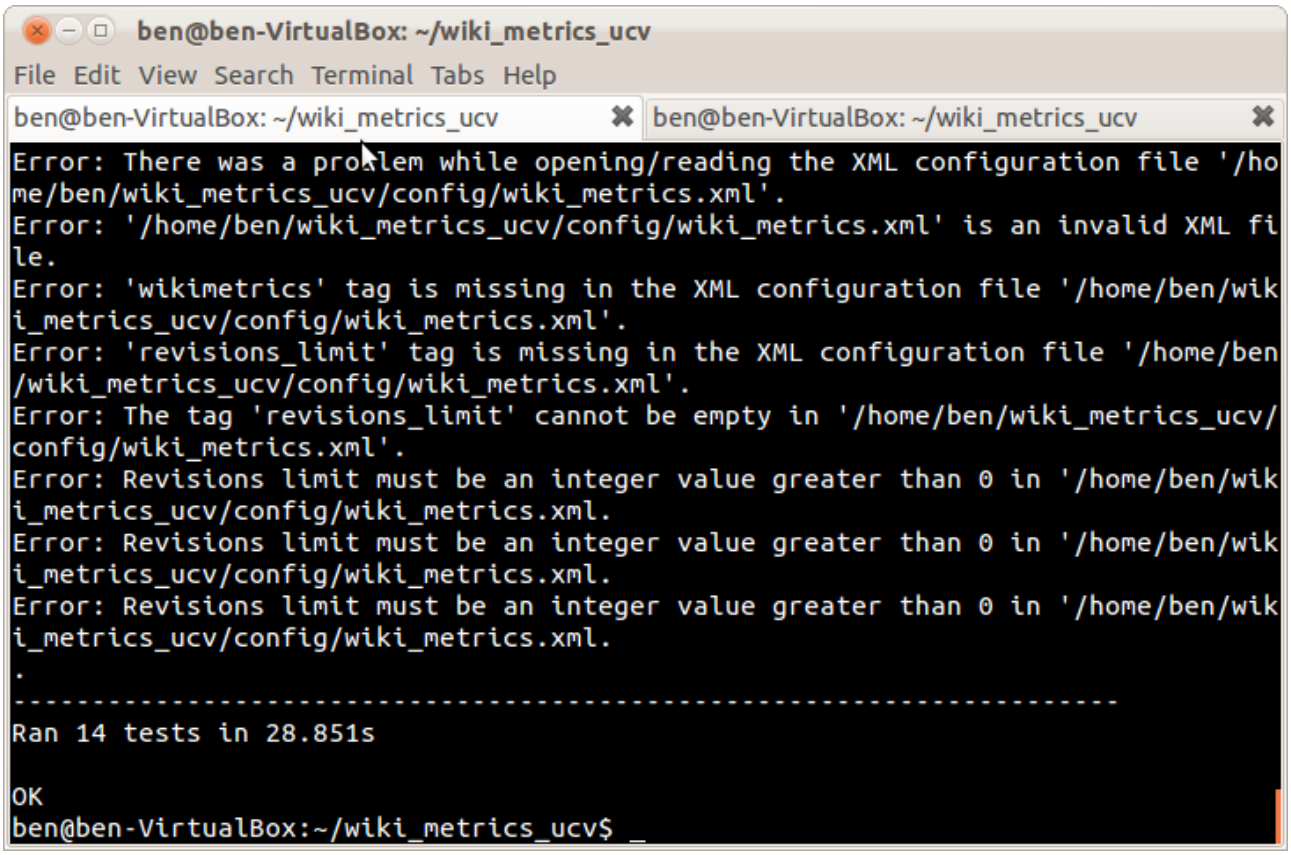

Figura 8.2: Resultado de las pruebas unitarias de la tercera iteración (los errores mostrados son esperados).

2.9. Cuarta Iteración

2.9.1. Planificación

2.9.1.1. Historias de usuarios

		Historia de Usuario
Número: 4	**Usuario:** Final	
Nombre historia: Front-end Web		
Prioridad en negocio: Alta		**Riesgo en desarrollo:** Baja
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo		

Descripción: Diseñar y desarrollar un front-end Web que permita la visualización, de manera gráfica, de las distintas métricas, especificadas en la historia de usuario número 1, de los distintos artículos especificados por el usuario a través de la Herramienta de URLs descrita en la iteración anterior.
Observaciones: • Se debe tener una opción de autocompletado a la hora de buscar artículos (por título). • Se debe permitir la visualización de las métricas a través de distintos tipos de gráficas. • Se debe permitir la visualización de todos los artículos existentes en la base de datos (“mostrar todos”). • De las métricas, se debe mostrar una pequeña descripción de la misma.

2.9.1.2. Plan de entrega

Para esta tercera iteración se acuerda el desarrollo de la historia de usuario número 4.

2.9.1.3. Velocidad del proyecto

Se acuerda por el grupo de desarrollo que esta iteración va a tener un tiempo de entrega estimado de un (1) mes.

El cliente no indicó una fecha de entrega determinada pero mostró interés en la pronta culminación del proyecto.

2.9.1.4. Plan de iteraciones

A continuación se muestran las tareas a completar durante esta cuarta iteración.

	Tarea
Número tarea: 1	**Número historia:** 4
Nombre tarea: Desarrollo de las vistas	
Tipo de tarea : Desarrollo	
Fecha inicio: 6 de febrero de 2012	**Fecha fin:** 29 de febrero de 2012

Programador responsable: Benjamin Worwa – Roberto D'Apuzzo
Descripción: • Diseñar las distintas vistas del front-end Web. • Se le debe proveer al usuario un mecanismo que le permita buscar artículos o listar todos los artículos existentes en la base de datos. • Se le debe ofrecer al usuario un mecanismo de autocompletado a la hora de buscar un artículo (por título). • Se debe utilizar Google Chart Tools para visualizar las gráficas de las métricas. • Se debe mostrar una pequeña descripción de las métricas. • Se debe visualizar, siempre que sea posible, el host de los artículos. • Se debe implementar, de manera opcional, un mecanismo que permita recordar búsquedas recientes.

	Tarea
Número tarea: 2	**Número historia:** 4
Nombre tarea: Desarrollo de los controladores	
Tipo de tarea : Desarrollo	
Fecha inicio: 15 de febrero de 2012	**Fecha fin:** 29 de febrero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: • Diseñar el controlador de bienvenida (Welcome). Este controlador carga la vista de bienvenida que contiene una caja de búsqueda con opción de autocompletado, un enlace para visualizar todos los artículos almacenados en la base de datos y una lista con los artículos recientemente buscados. De igual manera, carga el modelo que permite obtener los artículos recientemente buscados. • Diseñar el controlador de métricas (Metrics). Este controlador es el encargado de cargar la vista de las métricas, el modelo de métricas (Metrics_Model) y el modelo común (Common_Model). • Diseñar el controlador de AJAX (Ajax). Este controlador permite la implementación del mecanismo de autocompletado.	

Tarea	
Número tarea: 3	**Número historia:** 4
Nombre tarea: Desarrollo de los modelos	
Tipo de tarea : Desarrollo	
Fecha inicio: 17 de febrero de 2012	**Fecha fin:** 29 de febrero de 2012
Programador responsable: Benjamin Worwa – Roberto D'Apuzzo	
Descripción: • Desarrollar el modelo común (Common). Este modelo, disponible para todos los controladores, contiene un conjunto de funcionalidades comunes como: normalización de URLs, obtener artículos de la base de datos, etc. • Desarrollar el modelo de métricas (Metrics). Este modelo, disponible únicamente para el controlador Metrics, es el encargado de resolver las métricas de los artículos bajo demanda (en tiempo real).	

2.9.1.5. Rotaciones

Dado el número de programadores en el equipo, se decide no programar rotaciones durante el desarrollo de las tareas.

2.9.1.6. Reuniones

Se mantuvo el hábito de las reuniones de la primera, segunda y tercera iteración.

2.9.2. Diseño

Para esta cuarta iteración se acuerda el uso de PHP como lenguaje de scripting, CodeIgniter como framework MVC de PHP, jQuery como framework de JavaScript y Google Chart Tools como herramienta para la visualización de gráficas.

2.9.2.1. Metáfora del sistema

Para esta tercera iteración se decide escoger "Front-end Web para la visualización de métricas" como metáfora del sistema.

2.9.2.2. Diagrama de clases

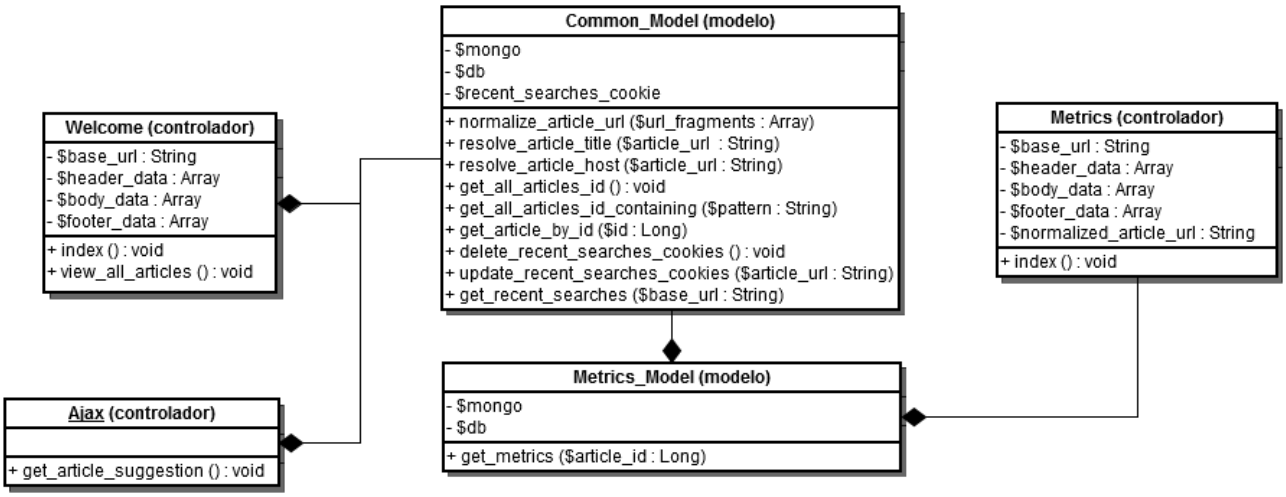

Figura 9.1: Diagrama de clases de la cuarta iteración.

Glosario de términos:

- **Welcome:** clase encargada de cargar la vista de bienvenida y el modelo común que permitirá obtener los artículos recientemente buscados.
- **Ajax:** clase encargada de cargar el modelo común que permitirá obtener los artículos con un título similar al buscado por el usuario.
- **Metrics:** clase encargada de cargar el modelo de métricas (Metrics_Model) para la obtención de las métricas de los artículos.
- **Common_Model:** clase que recopila un conjunto de funciones comunes para todos los controladores.
- **Metrics_Model:** clase encargada de calcular las métricas bajo demanda (en tiempo real) de los artículos.

2.9.3. Codificación

El front-end Web se desarrolló utilizando el patrón de arquitectura de software MVC.

Para la visualización de las métricas se optó utilizar un modelo de pestañas, en donde cada pestaña permite la visualización de una métrica del artículo en particular. Estas pestañas fueron desarrolladas utilizando jQuery.

Para recordar los artículos recientemente buscados se utiliza una cookie en la cual se almacenan las URLs de los últimos cuatro (4) artículos buscados. Al visitar un artículo esta cookie es actualizada con la URL del artículo actual.

Para el autocompletado se utilizó, de igual manera, jQuery. Cada vez que el usuario ingresa un carácter en la caja de búsqueda, se activa un temporizador que nos permite saber cuándo el usuario ha terminado de escribir la consulta, esto último para evitar hacer una petición al servidor por cada carácter ingresado. Acto seguido se envía la consulta ingresada

por el usuario utilizando el método GET al controlador de AJAX, llamando al método get_article_suggestion(). El controlador de AJAX carga el modelo común y éste obtiene de la base de datos las sugerencias de los artículos, utilizando expresiones regulares.

Para el cálculo de las métricas se utilizó el método group de MongoDB, el cual permite agrupar documentos teniendo en cuenta ciertos criterios. Este método group toma como parámetro un método recursivo reduce, el cual itera sobre la totalidad de los documentos de una colección almacenando la información calculada en cada iteración en el segundo parámetro del método (prev) y haciendo referencia al documento actual a través del primer parámetro del método (obj). Por ejemplo:

```
function(obj, prev) { prev.sum += obj.value; }
```

En este caso, la función anónima, iteraría sobre cada uno de los documentos de la colección actual y almacenaría en la variable sum del objeto prev la suma de todos los valores de cada uno de los documentos de la colección.

De esta manera se calculan las métricas de un artículo en particular, iterando sobre cada una de las revisiones asociadas a un artículo y calculando sus valores en tiempo real. Por engorroso y lento que parezca, en realidad se realiza una única iteración sobre cada una de las revisiones asociadas a un artículo, evitando tener que hacer múltiples consultas para cada una de las métricas.

Para las gráficas ´delas métricas se utilizó el API de Google Chart Tools. Para esto se generaron las estructuras de datos necesarias (DataTable) de manera dinámica utilizando PHP y los daos obtenidos del cálculo de las métricas.

2.9.4. Prueba

Durante esta iteración todas las pruebas fueron manuales, tanto para los mecanismos implementados utilizando JavaScript como para los resultados de las métricas. En las figuras 9.2, 9.3, 9.4, 9.5, 9.6 y 9.7 se muestran algunas capturas de pantalla de la interfaz gráfica y algunas métricas.

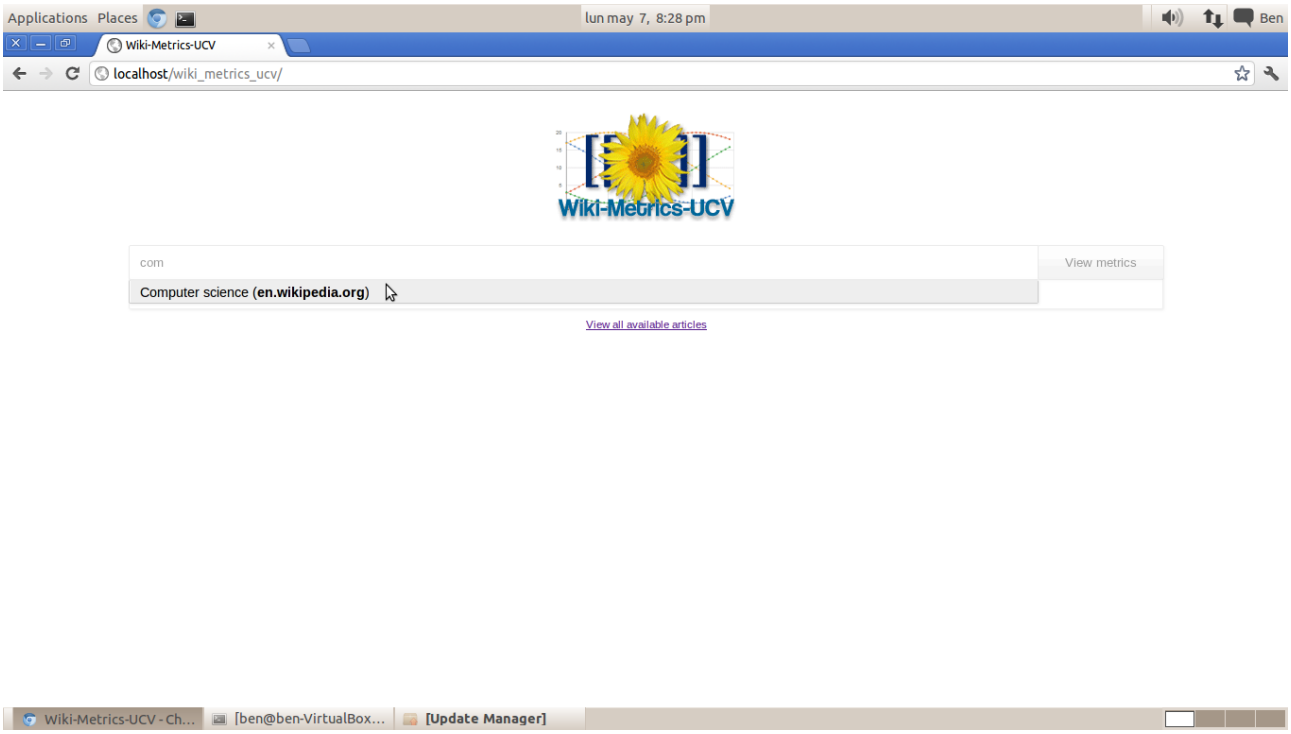

Figura 9.2: Página principal de la aplicación; barra de búsqueda con autocompletado.

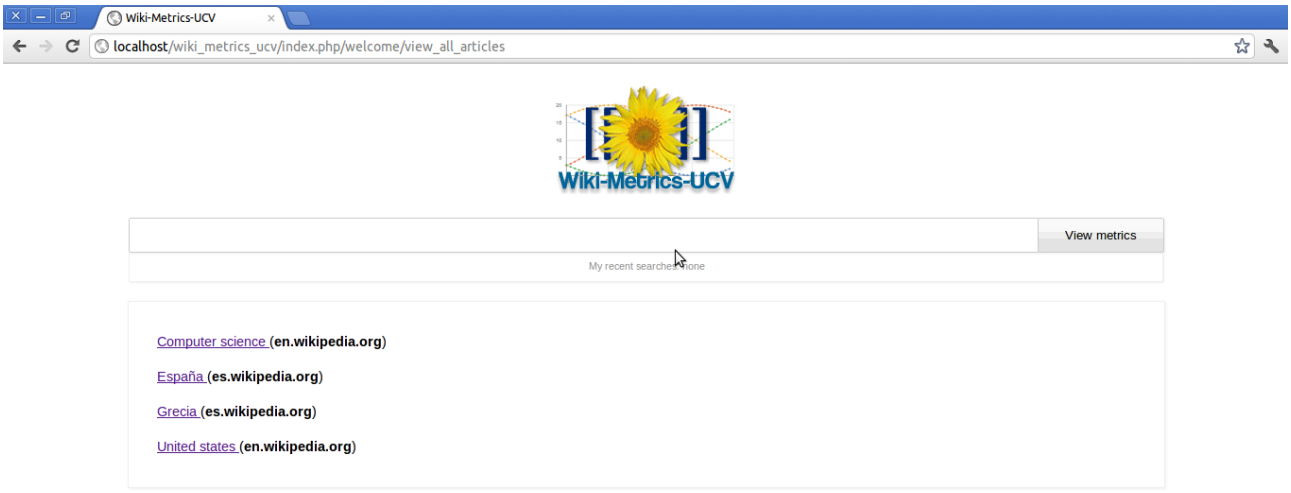

Figura 9.3: Página en la que se listan todos los artículos agregados a la base de datos.

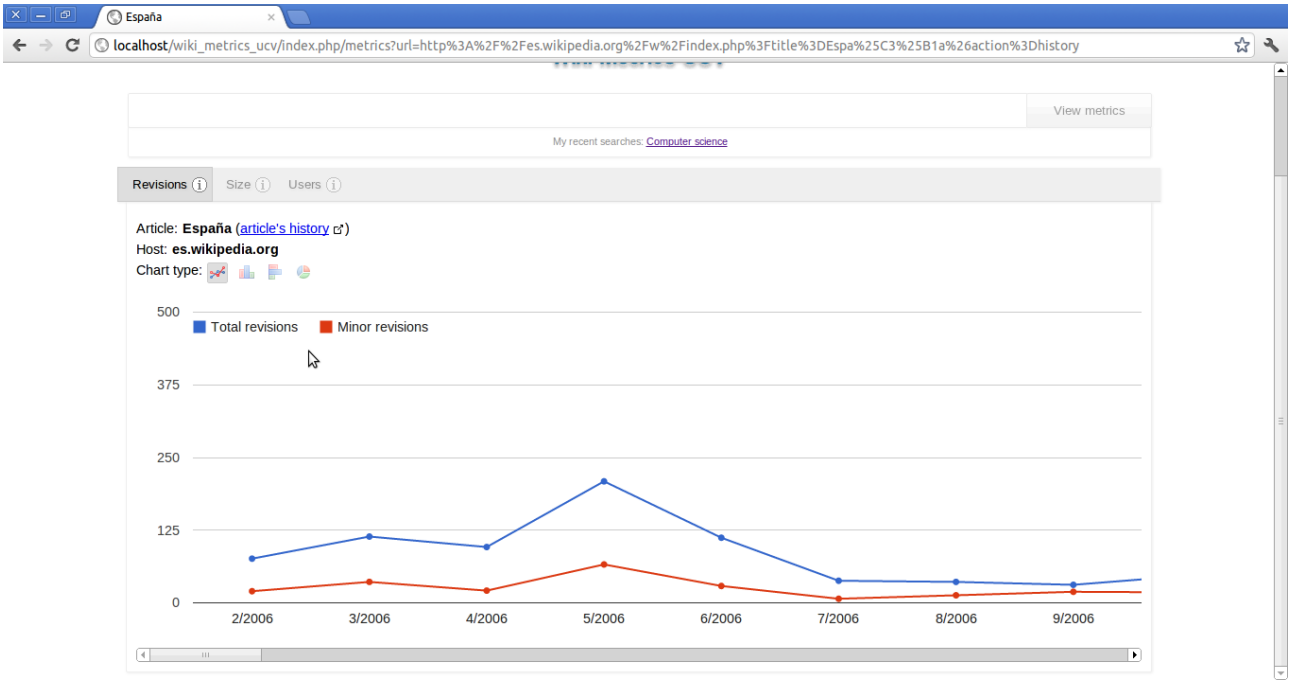

Figura 9.4: Página de métricas de los artículos; métrica de revisiones/mes; gráfica de línea.

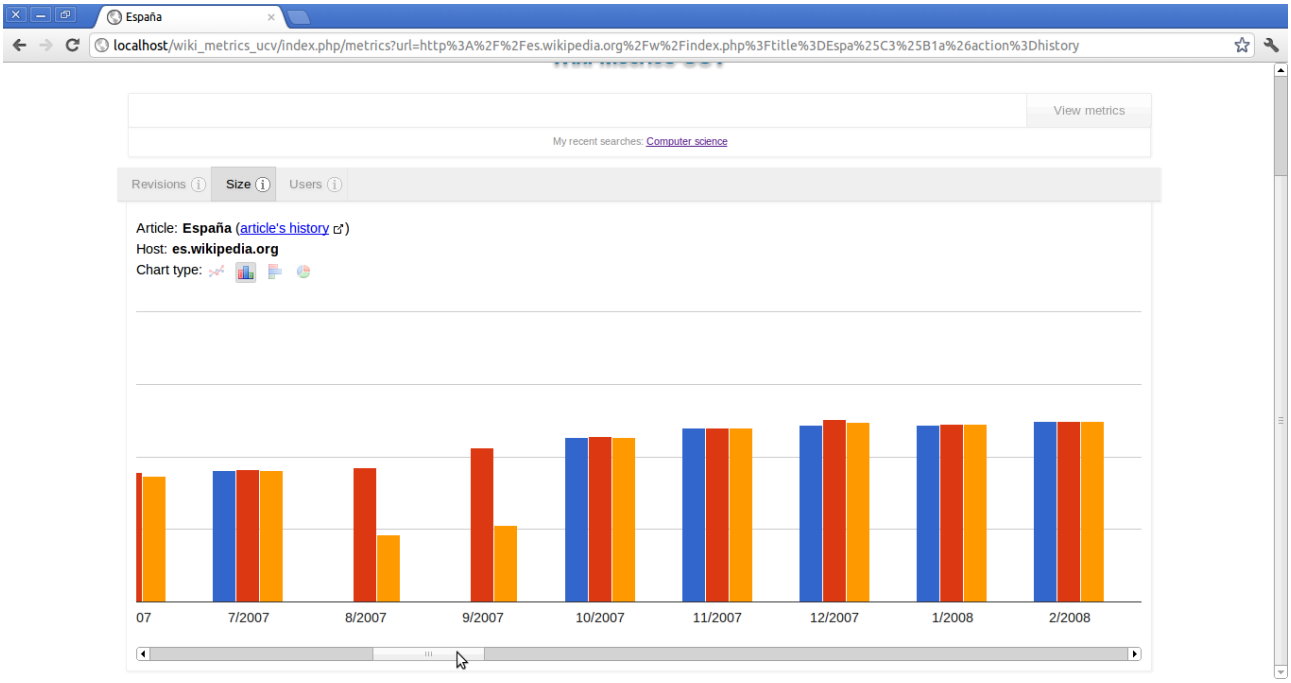

Figura 9.5: Página de métricas de los artículos; métrica de tamaño/mes; gráfica de columnas.

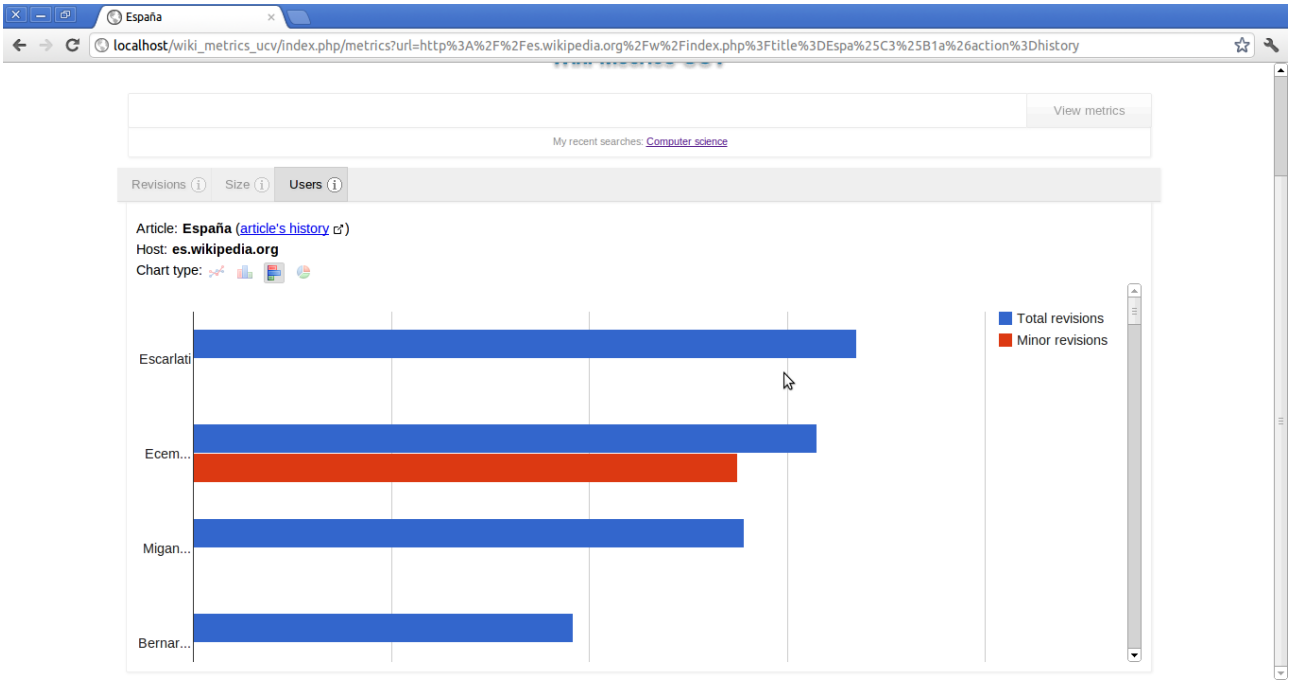

Figura 9.6: Página de métricas de los artículos; métrica de revisiones/usuario; gráfica de barras.

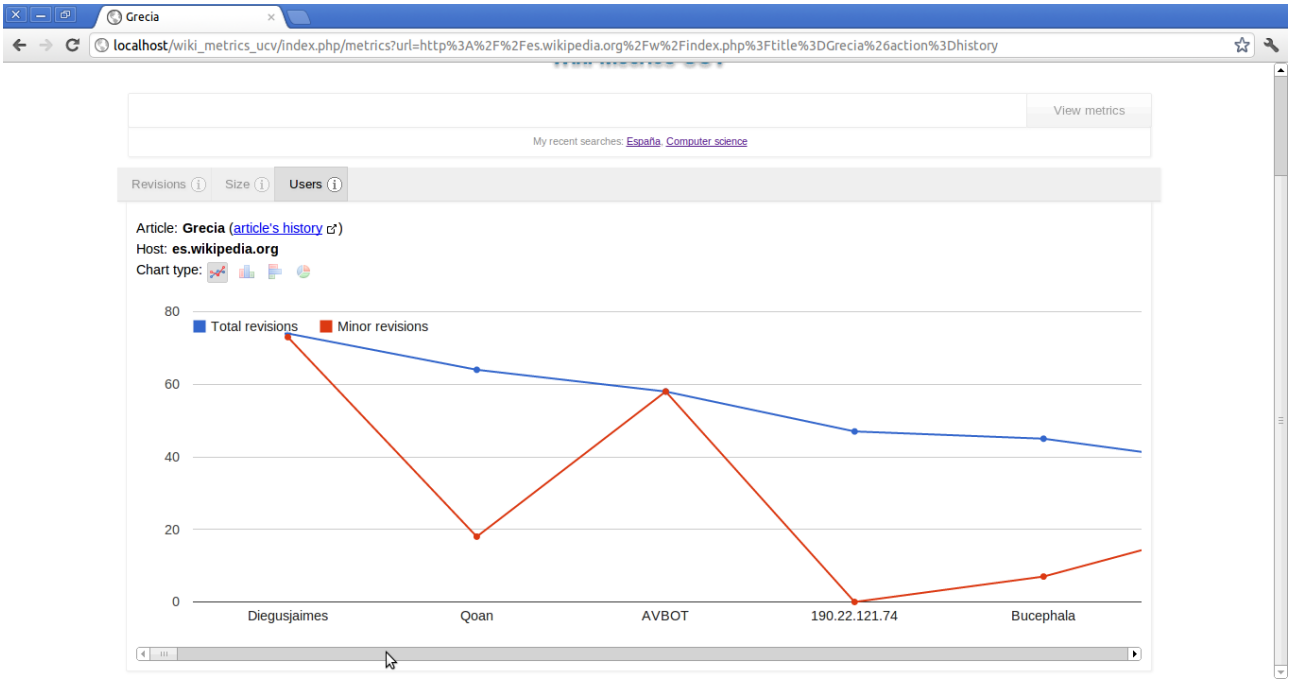

Figura 9.7: Página de métricas de los artículos; métrica de revisiones/usuario; gráfica de línea.

2.1. Enlaces de Interés

Este proyecto esta hospedado en GitHub, que es un controlador de versiones gratuito para proyectos de código libre. Durante el desarrollo del proyecto se realizaron 2 módulos que se enuncian a continuación.

Web Scraper de uso general

- XCraper (https://github.com/bworwa/xcraper)

Web Scraper adaptado al proyecto en cuestión con el front-end y las herramientas adicionales.

- wiki_metrics_ucv (https://github.com/bworwa/wiki_metrics_ucv)

CONCLUSIONES Y RECOMENDACIONES

En este trabajo se demostró la viabilidad del desarrollo de un Web Scraper de uso general que puede ser adaptado para recuperar cualquier tipo de información de cualquier página o aplicación Web al desarrollar el Web Scraper de uso general, Scraper.

Para el desarrollo de esta aplicación se consideró el uso del método ágil Extreme Programming (XP), considerando uno de los más importantes de sus principios, en donde hay constante comunicación entre el grupo de desarrolladores con la finalidad de compartir conocimiento e intercambiar ideas.

De igual manera se pudo comprobar mediante pruebas y evidencias las ventajas de las base de datos NoSQL sobre las base de datos relacionales a la hora de manejar grandes cantidades de información y su superioridad a la hora de tratar con replicaciones y fragmentaciones de los datos a través de distintos servidores de bases de datos. En particular, pudimos apreciar la superioridad en cuanto a la velocidad de escritura de MongoDB por encima de MySQL y la superioridad en cuanto a la velocidad de lectura al hablar de más de 5.000.000 de elementos almacenados.

Se decidió que las URLs a ser visitadas por la aplicación debían ser suministradas por el usuario final, bien sea por una entrada estándar o mediante un archivo de texto que permitiría especificar múltiples URLs, una por línea, con la finalidad de que la aplicación se enfoque en URLs de interés (y no URLs aleatorias sin importancia para el usuario) y, así mismo, evitar posibles bucles que pudieran bloquear la obtención de nuevas URLs.

Las métricas a calcular fueron seleccionadas en base a las necesidades del cliente. Estas métricas se calculan en tiempo real, ya que debido a la estructura de las consultas por asociación de MongoDB éstas pueden ser calculadas en una única pasada y no presenta mayor inconveniente con los tiempos de ejecución.

Google Chart Tools permitió la visualización de las métricas ya que es una herramienta robusta y flexible, como se pudo observar durante la cuarta iteración. También presenta la ventaja de ser desarrollada netamente en HTML5 y no depender del plugin de Adobe Flash para su visualización.

De tener un conjunto de artículos a visitar muy extenso, es probable que el Web Scraper no pueda procesar todas las actualizaciones y visitas a nuevas URLs. Por ejemplo, si se tienen 100.000 URLs obsoletas, a visitar, en la base de datos y se insertan 100.000 nuevas URLs, las URLs obsoletas serán desplazadas por las nuevas URLs (prioridad 0.5). Si calculamos un promedio de un (1) minuto por URL (tiempo promedio para historiales con más de 5.000 revisiones con procesamiento de resultados), se tiene que esperar setenta (70) días para volver a visitar las URLs obsoletas, luego de haber visitado la totalidad de las nuevas URLs.

Existe otra limitación surge de la cantidad de URLs almacenadas en la base de datos. Si se retoma el ejemplo anterior y se asume que el tiempo promedio de visita por URL es de veinte (20) segundos (tiempo promedio para historiales con más de 5.000 revisiones sin procesamiento de resultados) y que tenemos 100.000 URLs en la base de datos, se tiene que

esperar 23 días para actualizar cada URL, esto sin contar las nuevas URLs que pudiesen agregarse durante estos 23 días.

Ya que las limitaciones no dependen del poder de cómputo o de la capacidad de almacenamiento del servidor, se propone como solución el escalamiento horizontal del sistema.

Actualmente las métricas se muestran agrupadas por mes, ya que se consideró que agrupar las métricas por día generaría graficas demasiado extensas para artículos muy antiguos y agrupándolas por año no se mostrarían los detalles y tendencias buscados. Sin embargo, en ocasiones, el usuario podrías considerar útil la agrupación por día y por año, independientemente de lo engorroso o poco preciso que esto sea. Es por ello que es recomendable implementar estas opciones durante la vista de las métricas en el front-end Web.

Finalmente y en base a las limitaciones, la recomendación sería la distribución del sistema a través de diversos servidores (escalamiento horizontal). De esta manera, si se reparten las 100.000 URLs a través de seis (6) servidores, se disminuira el tiempo entre actualizaciones de cada una de las URLs de 23 días a 4 días. Es importante destacar que estos tiempos dependerán de la velocidad de descarga del servidor y de la capacidad de procesamiento del mismo. Actualmente se trabaja con un promedio de tamaño descarga por artículo de 3 MB y de un tiempo de procesamiento de esta cadena de caracteres de 3 MB de 40 segundos.

REFERENCIAS

1. **Beck, Kent, et al.** *Principios detrás del Manifiesto Ágil.* s.l. : Agile Alliance, 2001.

2. **Guru, Software.** *http://ldc.usb.ve.* [En línea] http://ldc.usb.ve/~abianc/materias/ci4713/SG-200603-Agil-RevistaMexicana.pdf.

3. ASD Ingeniería de Software. *ingenieriadesoftware.mex.tl.* [En línea] http://www.ingenieriadesoftware.mex.tl/61154_ASD.html.

4. The Agile Unified Process. *una.ac.cr.* [En línea] http://cgi.una.ac.cr/AUP/index.html.

5. **Wikipedia.** Herramienta CASE. *Wikipedia.org.* [En línea] http://es.wikipedia.org/wiki/Herramienta_CASE.

6. —. Feature Driven Development. *Wikipedia.org.* [En línea] http://en.wikipedia.org/wiki/Feature_Driven_Development.

7. Extreme Programming: A gentle introduction. *extremeprogramming.org.* [En línea] http://www.extremeprogramming.org/.

8. **José H. Canós, Patricio Letelier y Mª Carmen Penadés.** Metodologías Ágiles para el Desarrollo de Software. s.l. : Universidad Politécnica de Valencia.

9. **Carlos., Reynoso Billy.** Metodos Agiles en Desarrollo de Software, Introducción a la Arquitectura de Software. s.l. : Universidad de Buenos Aires.

10. Scrum. *Scrum.org.* [En línea] http://www.scrum.org/storage/scrumguides/Scrum%20Guide%20-%20ES.pdf#view=fit.

11. Scrum Process. *cprime.com.* [En línea] http://www.cprime.com/about/scrum_faq.html.

12. The PageRank Citation Ranking: Bringing Order to the Web. *stanford.edu.* [En línea] http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf.

13. **Menczer, F.** *ARACHNID: Adaptative Retrieval agents Choosing Heuristic Neighborhoods for Information Discovery. In D. Fisher, ed., Machine Learning: Proceedings of the 14th International Conference (ICML97). Morgan Kaufmann.* 1997.

14. **Menczer, F y Belew, R.K.** *Adaptive Information Agents in Distributed Textual Environments. In K. Sycara and M. Wooldridge (eds.) Proc. 2nd Intl. Conf. on Autonomous Agents (Agents '98).* s.l. : ACM Press, 1998.

15. **Chakrabarti, S, Van Den Berg, M y Dom, B.** *Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery.* s.l. : Computer, 1999.

16. **Pinkerton, B.** *Finding what people want: Experiences with the WebCrawler. In Proceedings of the First World Wide Web Conference.* Geneva, Switzerland : s.n., 1994.

17. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax. *ietf.org.* [En línea] http://tools.ietf.org/html/rfc3986.

18. Protocol. *sitemaps.org.* [En línea] http://www.sitemaps.org/protocol.php.

19. **Cho, J y Garcia-Molina, H.** *Effective page refresh policies for web crawlers. ACM Transactions on Database Systems, 28(4).* 2003.

20. **Koster, M.** *Robots in the web: threat or treat?* s.l. : ConnecXions, 9(4), 1995.

21. Terms of use. *http://wikimediafoundation.org.* [En línea] Wikimedia Foundation. http://wikimediafoundation.org/wiki/Terms_of_Use.

22. What is the Document Object Model? *w3.org.* [En línea] W3C. http://www.w3.org/TR/DOM-Level-3-Core/introduction.html.

23. XPath Introduction. *w3.org.* [En línea] W3C. http://www.w3schools.com/xpath/xpath_intro.asp.

24. XML Path Language (XPath) - 2 Location Paths. *w3.org.* [En línea] W3C. http://www.w3.org/TR/xpath/#location-paths.

25. Dave Raggett. *w3.org.* [En línea] W3C. http://www.w3.org/People/Raggett/.

26. W3C SOFTWARE NOTICE AND LICENSE. *w3.org.* [En línea] W3C. http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231.

27. HTML Tidy Configuration Options - output-xhtml. *sourceforge.net.* [En línea] http://tidy.sourceforge.net/docs/quickref.html#output-xhtml.

28. Extensible Markup Language (XML) 1.0 (Fifth Edition) - 2.1 Well-Formed XML Documents. *w3.org.* [En línea] W3C. http://www.w3.org/TR/xml/#sec-well-formed.

29. Introduction to XQuery. *w3.org.* [En línea] W3C. http://www.w3schools.com/xquery/xquery_intro.asp.

30. XQuery Selecting and Filtering. *w3.org.* [En línea] W3C. http://www.w3schools.com/xquery/xquery_select.asp.

31. XQuery Functions. *w3.org.* [En línea] W3C. http://www.w3schools.com/xquery/xquery_functions.asp.

32. *A Relational Model of Data for Large Shared Data Banks.* **Codd, Edgar F.** 1970, Communications of the ACM, págs. 13 (6): 377–87.

33. **Wikipedia.** Relational database. *Wikipedia.org.* [En línea] http://en.wikipedia.org/wiki/Relational_database.

34. **Chamberlin, Donald D y Boyce, Raymond F.** SEQUEL: A Structured English Query Language. 1974.

35. **Vegas, Jesús.** Introducción al SQL. *infor.uva.es.* [En línea] Universidad de Valladolid, 1998. http://www.infor.uva.es/~jvegas/cursos/bd/sqlplus/sqlplus.html.

36. *NoSQL Databases.* **Weber, Silvan.** University of Applied Sciences HTW Chur, Switzerland : s.n.

37. Google: 'At scale, everything breaks'. *zdnet.co.uk.* [En línea] ZDNet. http://www.zdnet.co.uk/news/cloud/2011/06/22/google-at-scale-everything-breaks-40093061/.

38. Product Details - Oracle Real Application Clusters. *shop.oracle.com.* [En línea] Oracle. https://shop.oracle.com/pls/ostore/f?p=700:product:1388611079819463::NO:RP,3:P3_LPI,P3_PROD_HIER_ID:4509920279431805720007,4509953293451805720010&tz=-4:0-30.

39. NOSQL Databases. [En línea] http://nosql-database.org/.

40. **Browne, Julian.** Brewer's CAP theorem. *julianbrowne.com.* [En línea] 2009. http://www.julianbrowne.com/article/viewer/brewers-cap-theorem.

41. **Wikipedia.** Graph Partition. *Wikipedia.org.* [En línea] http://en.wikipedia.org/wiki/Graph_partition.

42. MongoDB, CouchDB, MySQL Compare Grid. *mongodb.org.* [En línea] http://www.mongodb.org/display/DOCS/MongoDB,+CouchDB,+MySQL+Compare+Grid.

43. **Tilkov, Stefan.** InfoQ: A Brief Introduction to REST. *infoq.com.* [En línea] http://www.infoq.com/articles/rest-introduction.

44. **Wikipedia.** Representation State Transfer. *Wikipedia.org.* [En línea] http://en.wikipedia.org/wiki/Representational_State_Transfer.

45. HTML5. *W3.org.* [En línea] W3C. http://www.w3.org/TR/html5/.

46. 4.8.11 The canvas element. *whatwg.org.* [En línea] WHATWG. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element.

47. **Pilgrim, Mark.** Canvas - Dive Into HTML5. *diveintohtml5.org.* [En línea] http://diveintohtml5.org/canvas.html.

48. Update to jQuery Visualize: Accessible Charts with HTML5 from Designing with Progressive Enhancement. *filamentgroup.com.* [En línea] http://www.filamentgroup.com/lab/update_to_jquery_visualize_accessible_charts_with_html5_from_designing_with/.

49. Introduction to Using Chart Tools - Google Chart Tools - Google Code. *code.google.com.* [En línea] Google Inc. http://code.google.com/intl/es-ES/apis/chart/interactive/docs/.

50. Google Visualization API Reference - Google Chart Tools - Google Code. *code.google.com.* [En línea] Google Inc. http://code.google.com/intl/es-ES/apis/chart/interactive/docs/reference.html#DataTable.

51. http://flot.googlecode.com/svn/trunk/README.txt. *flot.googlecode.com.* [En línea] http://flot.googlecode.com/svn/trunk/README.txt.

52. jqPlot Usage. *jqplot.com.* [En línea] http://www.jqplot.com/docs/files/usage-txt.html.

53. Highcharts - Highcharts products. *http://highcharts.com.* [En línea] http://highcharts.com/products/highcharts.

54. d3.js. *mbostock.github.com.* [En línea] http://mbostock.github.com/d3/.

55. **Wikipedia.** Programación Extrema. *Wikipedia.org.* [En línea] http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

56. **S. Edlich, A. Friedland, J. Hampe, B. Brauer.** *NoSQL; Einstieg in die Welt nichtrelationaler.* 2010.

57. **Anderson, Chris , Lehnardt , Jan y Slater, Noah.** *CouchDB: The Definitive Guide.* s.l. : O'Reilly Media, 2009.

58. XML Path Language (XPath). *w3.org.* [En línea] W3C. http://www.w3.org/TR/xpath/#section-Introduction.

59. HTTP/1.1: Method Definitions. *w3.org.* [En línea] W3C. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.3.

60. The XML C parser and toolkit of Gnome. *xmlsoft.org.* [En línea] http://xmlsoft.org/.

61. countergram/pytidylib - GitHub. *github.com.* [En línea] https://github.com/countergram/pytidylib/.

62. PyMongo 2.0 Documentation. *mongodb.org.* [En línea] http://api.mongodb.org/python/current/.