Desarrollo de módulos de Karira ECM

Autor:
Aarón Martín Bermejo

Director:
César Ferri Ramírez

Tutor:
Modesto San Juan Alvárez

Septiembre de 2011

PFC para la titulación ITIS

karira etsinf UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

1. Introducción

La siguiente memoria condensa y resume el trabajo realizado en la empresa Karira en el desarrollo de ciertos módulos del gestor de contenidos Karira ECM, que se comentará a continuación.

Como curiosidad, todo el trabajo se ha desarrollado en forma de tele-trabajo, es decir, cada uno de los trabajadores de Karira nos encontramos en un lugar geográfico distinto, sin que ello haya sido en ningún momento un impedimento.

La estructura seguida en la memoria ha sido la de introducir el producto global (que actualmente continúa en desarrollo) que es Karira ECM y posteriormente explicar cada uno de los módulos desarrollados.

La estructura para definir cada uno de ellos ha sido:

- En primer lugar, exponer el problema a resolver o función a proporcionar y los motivos para que merezca la pena afrontar el desarrollo. Si es necesario, una breve introducción al ámbito o al contexto del desarrollo.

- En segundo lugar, cuál ha sido la solución o la forma que se ha ideado para solucionar el problema o aportar esa funcionalidad.

- En tercer lugar, cómo se ha implementado esta solución

- En cuarto y último, si es necesario, se explicarán las pruebas realizadas sobre la solución para verificar su corrección y adecuación al problema y su implantación en el sistema

Como se puede observar, esta estructura básicamente es la de Análisis – Diseño – Implementación - Pruebas.

1.1 Definición ECM

Un gestor de contenidos empresariales o Enterprise Content Management (ECM) es un programa que desarrolla todo un software de soporte (un framework) cuya principal funcionalidad es la de la creación y la gestión de contenidos. Suelen utilizarse principalmente para la creación, administración y mantenimiento de páginas y sitios web.

Un ECM facilita la tarea del mantenimiento de una página web separando el diseño de los contenidos, de forma que la modificación del diseño de la página web no afecte en absoluto a los contenidos que se mostraban hasta la modificación.

Con el increíble auge de internet desde los años 90 y, sobre todo, con el auge de la web 2.0 y la interacción de los usuarios con las páginas web, un ECM se ha convertido en esencial para cualquier organización que pretenda triunfar en el mundo web ya que la gestión de contenidos, la personalización de la apariencia y la información que se muestra a cada tipo de usuario ha incrementado sustancialmente su complejidad.

Esto es debido a que, tanto por la actividad muy intensa en internet así como por la cantidad y tamaño de la información que se maneja en una página web (incrementándose día a día) es tal, que necesitamos de herramientas que la hagan manejable.

Además, otra de las características de un ECM es que suele estar orientado a usuarios sin conocimientos de programación y/o bajos conocimientos en informática, de forma que sean capaces de construir, ampliar y gestionar una página web sin que la complejidad real de esta tarea sea ningún impedimento.

Esto acerca los gestores de contenidos cada día a un público más amplio que quiere tener su espacio en la red, pero que carece de los conocimientos necesarios para enfrentarse a ello.

Dentro del mundo de los gestores de contenidos, Karira ECM abarca la totalidad de los procesos vinculados a los contenidos, desde la definición, el desarrollo, la explotación, la gestión y el mantenimiento. Además, para garantizar su conectividad con otros productos es compatible con el estándar Content Management Interoperability Services (CMIS).

En cuanto a las características propias de Karira ECM:

- Basado en .Net Framework 3.5.
- Multiplataforma Windows, Linux y MacOs a través de Mono.
- Multi-idioma.

- Arquitectura orientada a servicios.
- Arquitectura de persistencia orientada a objetos compatible con múltiples soportes de almacenamiento.
- Sistema de publicación integrado con Microsoft Workflow Foundation.
- Sistema WSIWYG de gestión de páginas , plantillas, y módulos.
- Soporte nativo de archivos binarios.
- Integración con Visual Studio 2008.
- Integración con Office.
- Completo sistema de gestión de ficheros integrado con el motor de gestión de contenidos.
- Sistema de búsqueda e indexación compatible con LinQ.

Además, Karira ECM tiene una arquitectura orientada a servicios, por lo que cualquiera de los servicios puede ser reutilizado por el desarrollador para adaptarlos a los procesos de negocio propios.

1.2 Tecnologías utilizadas

Las tecnologías sobre las que se basa Karira ECM son las siguientes:

- **.NET Framework 4.0** como plataforma básica de desarrollo y **C#** como lenguaje de programación.

- **Mono Project**[1], para aportar compatibilidad con Linux. Mono básicamente es un proyecto cuyo objetivo es llevar las librerías de .NET a Linux en código opensource de forma que cualquier desarrollador pueda tener una aplicación en .NET funcionando en Linux.

- **Json** [2] [3]**,** acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos alternativo al uso de XML. Los objetos se serializan en strings de forma que cada valor está entre corchetes, en pares nombre/valor:

[1] http://www.mono-project.com/Main_Page
[2] http://www.json.org/
[3] http://es.wikipedia.org/wiki/JSON

{“persona”: {“dni”, 44527821” }, {“telefono”: “961234567” } }

- **MongoDB**[4], base de datos opensource, de alto rendimiento, libre de schema y documental. Al ser documental, está orientada al trabajo con Json y, con ello, a una base de datos más cercana a la orientación a objetos. Entre algunos de los usuarios, están Sourceforge o Electronic Arts [5].

- **ActiveMQ** [6], tecnología de encolado de mensajes asíncrona de software libre sobre Java.

- **IKVM**[7], implementación de Java y .NET para Mono para aportar interoperabilidad entre ambas plataformas. Básicamente, permite “compilar” .jar en librerías y ejecutables de .NET y a la inversa, reduciendo la restricción de uso de librerías en .NET de Java.

- **Dotnetlog** [8] y **Nlog** [9] como tecnologías de logging de mensajes y errores.

- **Nunit** [10], como framework para el desarrollo de las pruebas unitarias necesarias para la verificación de un correcto funcionamiento del software desarrollado.

- **Karira Framework**, framework sobre el que se trabaja para interactuar con MongoDB, ActiveMQ y gran parte de las tareas que se desarrollan en Karira ECM.

- Mención especial merece **Plastic SCM 11**[11], software de gestión de código y versiones que ha facilitado en una medida inmensa la tarea de gestión de código y recuperación de borrados accidentales.

Para tecnologías utilizadas en casos más concretos, se comentarán de forma puntual donde corresponda.

[4] http://www.mongodb.org/
[5] http://www.mongodb.org/display/DOCS/Production+Deployments
[6] http://activemq.apache.org/
[7] http://www.ikvm.net/
[8] http://www.theobjectguy.com/dotnetlog/
[9] http://nlog-project.org/
[10] http://www.nunit.org/
[11] http://www.plasticscm.com/

1.3 Objetivo del proyecto

Debido a la enorme tarea que es el desarrollo de un ECM sencillo, sin hablar de Karira ECM que tiene una cantidad de características y módulos propios que implican incluso una complicación considerablemente mayor, el objetivo de mi proyecto y de mis tareas en la empresa Karira ha sido el desarrollo de diversos módulos de Karira ECM.

Es por ello que mi tarea se centró en el desarrollo de los siguientes módulos:

- Módulo de **Crawling** (recorrido, indexación y búsqueda): recorrerá las páginas que se le indiquen, las indexará y permitirá realizar búsquedas sobre ellas. Básicamente, un motor de búsqueda.

- Módulo de **alojamiento**, **balanceo** y **adaptación** a picos de carga en el servidor: alojamiento en máquinas virtuales adaptativas a los picos de carga.

- Módulo de **auto-categorización** mediante Inteligencia Artificial: categorización automática de artículos mediante técnicas de minería de datos, aprendizaje automático e inteligencia artificial. Uso de Weka.

- **Compatibilidad** con **Linux**, bajo Mono: adaptación del proyecto Karira ECM a Mono para lograr compatibilidad entre plataformas

2. WebCrawlerService

Karira ECM, como gestor de contenidos, necesita de un servicio de búsqueda que permita realizar búsquedas muy rápidamente sobre el contenido, incluso aunque éste tenga un tamaño considerable. Este requisito es básico en casi cualquier sitio web creado con Karira ECM, puesto que las búsquedas serán una funcionalidad importante.

Para ello, antes habrá recorrer cada página del sitio, extraer el contenido e indexarlo para posiblitar una búsqueda veloz, de modo que se evite el realizar un recorrido sobre el contenido del sitio en cada una de las búsquedas.

Esta indexación consiste en recopilar los contenidos que se vayan extrayendo en el recorrido de las páginas y generar un índice con estos contenidos de forma ordenada. Este enfoque es necesario para una búsqueda veloz y es el que utilizan los buscadores más utilizados como google o yahoo.

De hecho, el algoritmo de indexación en estos buscadores es motivo de intenso estudio por parte de aquellos profesionales que quieren lograr una posición alta en el índice de los mismos. Sin embargo en nuestro caso no es la publicidad exterior la que se busca, si no sencillamente búsquedas internas.

En resumen, este servicio tendrá como tareas a realizar:

- Recorrido sobre el contenido del sitio
- Indexación del contenido
- Búsquedas sobre el contenido indexado

Este servicio deberá poder recorrer un sitio web y todas las páginas del mismo, de modo que obtenga el contenido de cada página y lo indexe para acelerar las búsquedas.

El contenido a indexar y sobre el que realizar las búsquedas será textual. Indexará tanto el texto plano de las páginas (eliminando los tags HTML y obteniendo únicamente el texto) como cualquier tipo de archivo textual al que enlace la página.

Los formatos posibles de estos archivos de texto enlazados serán .txt, formatos de Microsoft Office (.doc, .docx, .ppt, .pttx, .xls), PDF's y rtf (formato de texto enriquecido).

Una vez indexado, se deberán poder realizar búsquedas de forma que devuelva en qué página, páginas o archivo se encuentra el término buscado, así como el contenido/s de la/s misma/s.

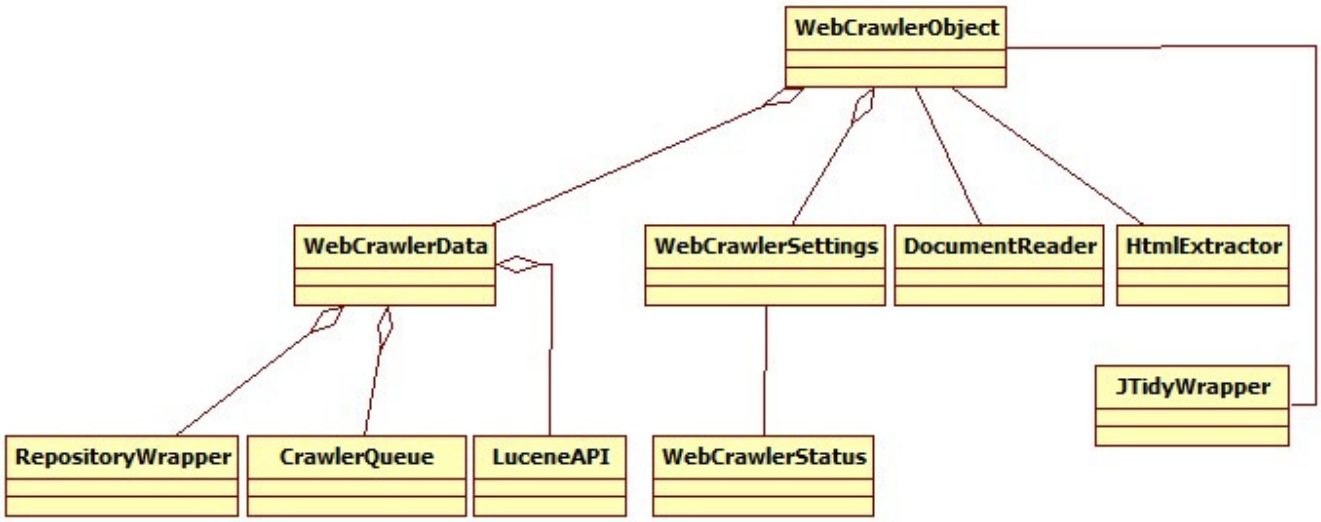
Además, es muy interesante devolver el “score” o porcentaje de probabilidad de que sea una determinada página la que necesita el usuario al realizar la búsqueda por el término deseado, dotando al sistema de cierta “inteligencia”.

Este ciclo recorrido – indexación deberá automatizarse, de forma que el usuario especifique cada cuánto tiempo desea que se repita la indexación del sitio.

Esta solución busca no saturar el sistema con un crawling continuo, ya que si el sitio es excesivamente grande este proceso será costoso tanto en tiempo como en carga del equipo/s que lo realicen.

2.1 Diseño

Para este cometido se optó por la siguiente arquitectura:



2.2 Tecnologías utilizadas

Lucene

Lucene[12] es una **librería** open source de alto rendimiento que proporciona todas las herramientas necesarias para crear un motor de búsqueda. Además de sus altas capacidades de rendimiento (alrededor de 95 GB/hora indexados en un hardware habitual) también proporciona una alta escalabilidad, de forma que no es necesario un hardware muy potente (tan sólo necesita 1MB de RAM y el tamaño del índice suele ser de 20% - 30% del texto indexado)[13].

Es por ello que se escogió como librería a utilizar para las tareas de indexado y búsqueda, ya que porporciona unos resultados muy positivos.

Algunos de sus usuarios más famosos pueden ser Apple o Eclipse IDE (ambos para las búsquedas en su documentación).

Es ampliamente utilizada para la implementación de motores de búsqueda por lo que en muchas ocasiones es confundida con una librería o un software que proporciona esta funcionalidad. Sin embargo, como se verá a continuación, no es así y el resto de aspectos del motor han sido implementados desde cero.

Está escrita en Java, por lo que fue necesario el uso de IKVM para "recompilar" Lucene a la plataforma .Net para poder utilizarla.

Apache POI

Apache POI[14] es un proyecto bajo licencia Apache que proporciona toda una serie de librerías escritas en Java para el trabajo con documentos en formatos de Microsoft Office, como Word, Excel o Powerpoint.

Al ser un proyecto opensource, al contrario que las librerías que proporciona Microsoft, proporciona ya una ventaja considerable frente a éstas en el plano económico. Además, la complejidad del trabajo con Apache POI frente a las librerías proporcionadas por Microsoft para trabajar con documentos de office sorprendentemente es menor.

---

[12] http://lucene.apache.org/java/docs/index.html
[13] http://lucene.apache.org/java/docs/features.html
[14] http://poi.apache.org/

Por ello se escogió para la lectura de los textos en formatos Office encontrados en las páginas al realizar el crawling. Como viene siendo habitual, se utilizó IKVM para ser utilizado desde .Net.

Open XML SDK

Sin embargo, Apache POI no cubre todos los formatos, si no que actualmente cubre los formatos Office bajo el estándar Microsoft Office XML[15]. En cambio, los formatos bajo la especificación Office Open XML[16] no son compatibles con Apache POI.

Por ello, se decidió descargar el SDK de Microsoft para este estándar[17], que pese a no ser opensource, proporcionaba las herramientas que eran necesarias para trabajar con aquellos archivos fuera de Microsoft Office XML.

Estos tipos de archivos son aquellos más recientes en el mundo Office, como pueden ser .docx o .pptx de Office 2007 o 2010.

PDFBox

PDFBox[18] es una librería bajo licencia Apache que proporciona las herramientas necesarias para el trabajo con documentos en PDF (como su propio nombre indica). Se escogió esta tecnología para la lectura de textos en formato PDF durante el crawling debido a su característica opensource, su facilidad de uso y sus buenos resultados.

Html Agility Pack

Html Agility Pack[19]**,** es una librería opensource escrita en .Net que facilita el trabajo con documentos HTML ya que proporciona toda una serie de funcionalidades para "parsearlo", trabajar con nodos al estilo XML...

Se utiliza para tomar las páginas HTML y no tratar con ellas como simple texto plano, si no con las características propias de un documento de este tipo.

[15] http://es.wikipedia.org/wiki/Microsoft_Office_XML
[16] http://es.wikipedia.org/wiki/Office_Open_XML
[17] http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=5124
[18] http://pdfbox.apache.org/
[19] http://htmlagilitypack.codeplex.com/

Los motivos para escoger esta librería concreta fueron que es opensource y tiene una buena tolerancia ante el código html mal redactado, algo que resulta bastante habitual.

Este código HTML mal redactado no se refiere únicamente a código sin tabulación o sin espaciados, si no más bien a tags sin cerrar o sin abrir que, en la práctica en muchas ocasiones pasan desapercibidos.

Sin embargo, a la hora de un análisis exhaustivo como realiza la extracción de enlaces del crawling, puede resultar un serio problema para un buen funcionamiento.

JTidy

JTidy[20] es una librería opensource escrita en Java diseñada para verificar la corrección de código Html y, en la medida de lo posible, ordenarlo y repararlo.

La necesidad de utilizar una librería de este tipo surgió debido a que al realizar las pruebas del crawler se detectó que, en casos en los que el código html de la página visitada estaba mal redactado, la extracción del texto plano a partir de las páginas web no se realizaba correctamente. Esto implicaba que en el texto que se indexaba se colaban etiquetas de marcado que adulteraban las posteriores búsquedas.

Por ello se escogió esta librería ya que tiene una buena tolerancia frente al código html mal redactado, algo bastante habitual. Además tiene una buena capacidad para "repararlo", de forma que se obtiene un código html "limpio" que evita que se cuele entre el texto plano que se extraiga posteriormente.

Expresiones regulares

Se ha utilizado una expresión regular para extraer el texto plano (sin etiquetas HTML) de las páginas recorridas, ya que inicialmente se realizaba un filtrado de aquellos nodos (del documento HTML de HTML Agility Pack) que correspondiesen con nodos de texto.

Sin embargo, es obvio y se comprobó que esta aproximación resultaba algo deficiente, con lo que se optó por el uso de Regex[21].

Una expresión regular básicamente es un patrón que es capaz de definir un conjunto de cadenas sin enumerar sus elementos. Esta simple afirmación esconde toda una compleja teoría matemática que, para nuestros propósitos, no será necesaria explicar.

[20] http://jtidy.sourceforge.net/
[21] http://es.wikipedia.org/wiki/Expresi%C3%B3n_regular

Las expresiones regulares son muy útiles en nuestro caso, puesto que se definió una expresión regular que encontrase todo el contenido que estuviese entre “<” o “>” (básicamente) y se sustituyó por la cadena vacía. Concretamente, la expresión regular definida es la siguiente:

```
string withoutHtml = new Regex("<\\S[^><]*>", RegexOptions.IgnoreCase | RegexOptions.Singleline | RegexOptions.Multiline | RegexOptions.CultureInvariant | RegexOptions.Compiled).Replace( extractFrom.DocumentNode.InnerHtml, String.Empty);
```

De esta forma se obtiene el texto plano de una página HTML.

2.3 Implementación

La necesidad de un motor de búsqueda para Karira ECM se ha satisfecho mediante la siguiente implementación, viéndolo de abajo hacia arriba:

WebCrawlerObject

Se optó por la construcción de un webcrawler[22] [23].

La definición de un WebCrawler es la de un programa o software que recorre “Internet” de una forma metódica y automatizada.

Esta tarea la realiza visitando una o varias páginas inicialmente, llamadas semillas, de las que extraerá todos los hipervínculos que encuentre en ellas, guardándolos como próximas páginas a visitar. Este ciclo se irá repitiendo indefinidamente o hasta que se cumpla alguna condición de fin especificada.

Este proceso tiene problemas a sortear, que son los siguientes:

1. Las URL’s posibles a recorrer pueden llegar a ser infinitas, en gran medida por el enorme tamaño de Internet. Esto hace que en la práctica sea totalmente necesario un o unos criterios de filtrado de las páginas a recorrer.
2. Estas páginas pueden estar bajo una constante actualización, sobre todo hoy en día con la web 2.0[24]. Es por ello que también habrá que definir un criterio que indique cuándo se debe volver a recorrer las páginas.

[22] http://es.wikipedia.org/wiki/Ara%C3%B1a_web
[23] http://en.wikipedia.org/wiki/Web_crawler
[24] http://es.wikipedia.org/wiki/Web_2.0

3. Alguna política para no sobrecargar excesivamente los sitios web.

4. Políticas de almacenamiento frente a la posibilidad de crawlers paralelos, básicamente para que no se recorra una y otra vez la misma página por cada crawler, si no que esta información esté compartida.

En resumen, el objetivo de un crawler o araña web es el de recorrer y crear una "copia" de las páginas webs deseadas para utilizarlas posteriormente en un motor de búsqueda.

El webcrawler tendrá como cometido el recorrido de las páginas de un sitio o de las páginas hijas de una especificada.

Sin embargo, dado que nuestras necesidades son un poco más amplias, también se ha dotado al crawler de una implementación que, cara al exterior, también realiza la indexación de los contenidos que recorre y, una vez indexados, el mismo crawler será capaz de funcionar como motor de búsqueda sin ningún problema.

Sin embargo, respecto a la indexación de las páginas, el WebCrawler internamente trasladará la tarea al objeto LuceneAPI (explicado más adelante) así como también la tarea de la búsqueda sobre el contenido indexado, debido en gran medida a la facilidad y rapidez de esta librería para este cometido.

Los datos necesarios para la inicialización serán:

- La url de la página padre; tan sólo se recorrerán las páginas hijas de ésta (página semilla).

- El tamaño máximo del contenido a indexar, para no saturar la memoria del equipo si el sitio web es excesivamente grande.

- El tiempo en segundos que debe pasar para actualizar el contenido indexado, de forma que automáticamente recorra las páginas una vez pasado este tiempo y actualice el índice, si es que éste ha sido modificado (solucionando el problema número 2).

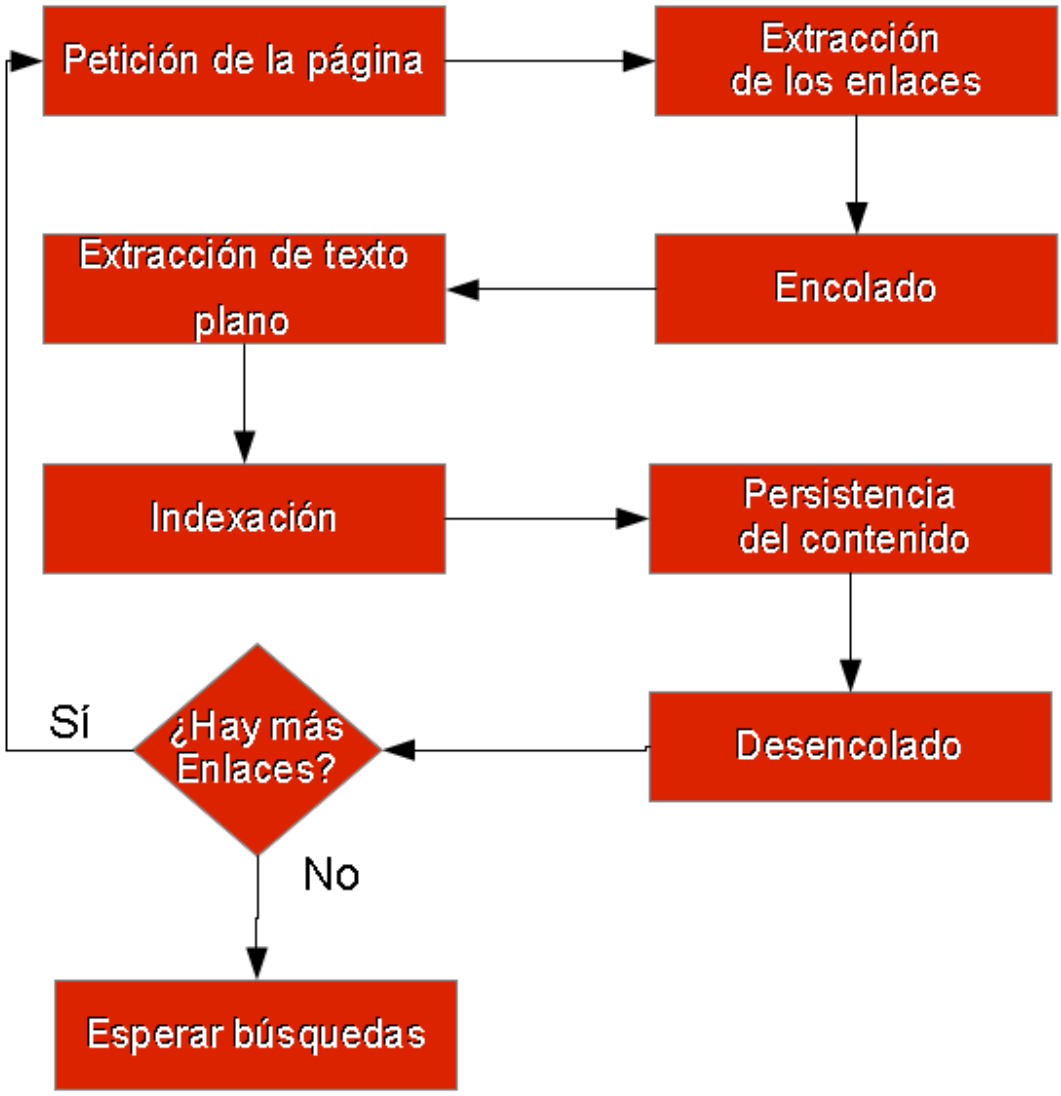
Estos datos de inicialización irán encapsulados en la clase WebCrawlerSettings (explicada a continuación), de forma que se podrán guardar en una base de datos de Mongo. De esta forma, podremos inicializar el WebCrawlerObject en cualquier momento y éste tendrá acceso al contenido que haya recorrido e indexado mediante Lucene.

Como criterio de filtrado, el WebCrawler únicamente recorrerá e indizará aquellas páginas que sean “hijas” de la página padre o semilla. Esto quiere decir que si, por ejemplo, nuestra página padre es www.ejemplo.com/ejemplo , el crawling sólo se realizará sobre aquellas páginas cuya url comience por la del padre, por ejemplo www.ejemplo.com/ejemplo/probando .

De esta forma, solucionamos también el problema número 1.

Ciclo de vida

Para realizar las tareas necesarias, el WebCrawlerObject seguirá el siguiente ciclo de vida:



1. **Petición de la página**: se obtiene la página al servidor que corresponda.
2. **Extracción de los enlaces**: se extraen todos los "links" o enlaces que contenga la página.
3. **Encolado**: se toman todos los enlaces, se filtran aquellos que sean hijos de la página padre a analizar y se encolan (encolado explicado más adelante).
4. **Extracción del texto plano**[25]: eliminación del código HTML, de forma que se indexe sólo el texto que ésta contenga.

[25] En el caso de trabajar sobre un archivo de texto, se obvia el paso número 4.

5. **Indexación:** se indexa, mediante Lucene, el texto plano extraído (explicado más adelante).
6. **Persistencia del contenido:** se guarda en Mongo el contenido en sí, para dotarlo de persistencia
7. **Desencolado:** se desencola el siguiente enlace y, si hay alguno, se repite el ciclo
8. **Esperar búsquedas:** si ya se han extraído todos los enlaces de todas las páginas hijas y se han recorrido todas ellas, el crawler pasa a esperar peticiones de búsqueda sobre el índice recién creado

WebCrawlerSettings

Es el objeto diseñado mediante el patrón memento[26] que permite la inicialización del WebCrawlerObject, ya que encapsula los datos que necesita. De esta forma permite guardarlos en una base de datos de Mongo e inicializar siempre que se necesite un WebCrawlerObject que tenga las referencias para conocer el contenido indexado anteriormente.

WebCrawlerStatus

Una clase que sencillamente encapsula el estado del WebCrawler, de forma que en caso de tener más estados en un futuro aparte de Running y Stopped, puedan ser añadidos sin ningún problema.

WebCrawlerData

WebCrawlerData es una clase que actúa de fachada[27] entre el objeto WebCrawler y sus correspondientes repositorio, colas e índice.

Está construida siguiendo el patrón singleton[28], de forma que en memoria haya una sola referencia para cada WebCrawler.

Así obtenemos un acceso transparente entre la lógica del WebCrawlerObject y sus datos, obteniendo un acoplamiento entre capas débil.

Para ello, proporciona los métodos que necesita el WebCrawlerObject para interactuar con sus datos, llamando directamente al objeto correspondiente (el repositorio, la cola o el índice).

[26] http://www.dofactory.com/Patterns/PatternMemento.aspx
[27] http://www.dofactory.com/Patterns/PatternFacade.aspx
[28] http://www.dofactory.com/Patterns/PatternSingleton.aspx

LuceneAPI

LuceneAPI es el objeto que realmente se encarga de las tareas de indexación y búsqueda. Como su nombre indica, es un API construido siguiendo el patrón de diseño fachada[7] de forma que adapta y nos da una serie de métodos que serán los que necesitaremos tanto para indexar el contenido como para realizar búsquedas sobre el mismo.

Para inicializarlo, se le pasará el path del disco donde crearemos el índice y el id del crawler que lo va a utilizar.

Como su nombre indica, es un API que internamente utiliza la librería Lucene, anteriormente comentada.

Este API permite:

- **Indexar páginas web**, teniendo el documento indexado como parámetros: la url de la página, el texto extraído (sin html), la fecha de indexación y, en caso de tener la página metadatos[29], el lenguaje, las "keywords" y la descripción de la página.

- **Indexar textos**, teniendo el documento indexado como parámetros: la url de donde descargar el texto, el texto en sí y la fecha de indexación.

- **Realizar búsquedas sobre el índice**, indicándole el término a buscar, en qué sitio web ha de ser buscado, el número de resultados por página y el número de página (estos dos últimos parámetros son para la paginación de resultados).

Esta búsqueda no será literal, si no por aproximación, ya que Lucene realiza de esta forma las búsquedas. Así, si buscamos pesca nos devolverá los resultados de pesca, pescador, pescadería... cada uno con su "Score"[30] o puntación correspondiente. Esta puntuación nos indica el porcentaje o la puntuación que tienen los resultados de ser correctos, es decir, un número estimado por Lucene que nos indica la probabilidad de que el resultado sea el correcto.

La búsqueda devolverá una lista de objetos Content, clase que se explicará a continuación.

Content

Content es una clase que encapsula los datos que se guardarán en el indexado y que se devolverán en las búsquedas.

---

[29] http://www.w3schools.com/html/html_meta.asp
[30] http://lucene.apache.org/java/2_3_2/scoring.html

Cada objeto de esta clase tendrá: un Id (para identificarlo al guardarlo en Mongo), UrlFrom (la url de donde viene), ContentText (el contenido en sí), Score (la "puntuación" que le da Lucene de que sea esta la búsqueda correcta), IndexedAt (la fecha en la que se indexó).

RepositoryWrapper

RepositoryWrapper es una clase construida siguiendo el patrón wrapper o adaptador[31] y que adapta la clase MongoDBRepository de Karira Framework haciendo que muestre una serie de métodos adaptados a las necesidades del WebCrawlerObject.

Esta clase permite guardar los contenidos indexados mediante Lucene en Mongo, ya que es más sencillo distribuir un repositorio con Karira Framework que un índice de Lucene.

Además, recuperar objetos de Mongo es incluso más veloz que desde Lucene, ya que Mongo no está en disco si no en memoria y, aunque no fuera así, la eficiencia de Mongo en estos aspectos es mayor.

De esta forma, tenemos el índice en Lucene y el contenido en sí en un repositorio de Mongo.

CrawlerQueue

CrawlerQueue es otra clase adaptadora [8] , que adapta la clase MessageQueue de Karira Framework para las tareas de encolado de mensajes del WebCrawlerObject. Internamente, MessageQueue utiliza ActiveMQ, tecnología comentada anteriormente.

La necesidad del uso de una cola surgió debido a que el Crawler en muchos casos encontraba un número alto de enlaces en las páginas que recorría. Se decidió por dos motivos:

1. Para hacerlo más ligero, evitando que lleve la carga de todos los enlaces que requieren de una futura visita.
2. Sobre todo, para permitir que el crawler pueda ser parado en medio de su ciclo de vida y que pueda reanudarlo sin ningún problema.

Así, el crawler envía una lista de enlaces pendientes de visita a ActiveMQ que los guarda en la cola del crawler y, aunque el crawler fuera parado y destruido, la cola seguiría persistiendo. En cualquier momento que se reconstruya el crawler con las WebCrawlerSettings, desencolará el primer enlace que haya en la cola y continuará su trabajo por donde lo dejó.

[31] http://www.dofactory.com/Patterns/PatternAdapter.aspx

Una vez realiza cada pasada en el bucle de su ciclo de vida, desencola el siguiente enlace que haya en la cola de ActiveMQ y vuelve a iniciar el ciclo (visita, extracción, indexación...) hasta que no haya más enlaces encolados.

Cada cola tendrá su nombre, que, gracias al Id de las CrawlerSettings, será única. De esta forma cada Crawler tendrá su propia cola.

Las capacidades de esta clase serán encolar y desencolar mensajes, que serán Url's a visitar.

DocumentReader

DocumentReader es una clase construida también siguiendo el patrón adaptador, cuyo objetivo es leer cualquier tipo de documento textual y extraer su contenido.

Para ello, internamente utiliza las librerías Apache POI, Open XML SDK y PDFBox, comentadas anteriormente. Gracias a estas librerías, es capaz de extraer el contenido de los siguientes formatos de archivo: .pdf, .docx, .pptx, .txt, .rtf, .xls, .doc, .ppt .

HTMLExtractor

HtmlExtractor es una clase cuyo objetivo es el de extraer el texto plano de un código Html. La forma en que realiza esta tarea es mediante expresiones regulares, ya que se probaron diversas librerías cuyos resultados no fueron especialmente satisfactorio.

Finalmente, se optó por las expresiones regulares ya que su rendimiento era aceptable y sus resultados hasta el momento, inmejorables.

JTidyWrapper

JTidyWrapper es un adaptador de la librería JTidy (comentada anteriormente), cuyo único objetivo es el de recibir un código html y devolverlo organizado y reparado (si es que es necesario).

WebCrawlerService

En cuanto al servicio en sí, se ha implementado siguiendo el modelo MVC[32].

[32] http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

El servicio es capaz de manejar diversos crawlers, gracias a la sencillez de su almacenado e inicialización mediante las WebCrawlerSettings que funcionan como memento o recuerdo de sus estado y que permiten la reconstrucción de un crawler sin ninguna dificultad.

Esta capacidad se utiliza para poder manejar una cantidad casi infinita de crawlers y, con ellos, sus respectivas colas, índices y repositorios, todos ellos con la posibilidad de estar distribuidos sin que ello suponga tampoco ningún problema.

De esta forma, tenemos una capacidad de crawling muy considerable, gracias a su distribución.

Este uso de múltiples crawlers puede ser totalmente transparente para el usuario, ya que el servicio ofrece una interfaz de métodos que simplemente ofrece la posibilidad de recorrer e indizar y de buscar.

Aunque para usuarios más avanzados en el uso de Karira ECM se ofrece una serie de métodos en los que se especifica el webCrawlerId para evitar el tener que buscar qué crawler concreto ha indizado una página (por ejemplo).

Para que resulte más claro, se muestra a continuación la interfaz **IWebCrawlerService**:

```
[OperationContract]
WebCrawlerSettings AddOrUpdateWebCrawler(WebCrawlerSettings crawler);

[OperationContract]
bool DeleteWebCrawler(string crawlerId);

[OperationContract]
bool ContainsWebCrawler(string crawlerId);

[OperationContract]
List<WebCrawlerSettings> GetWebCrawlers();

[OperationContract]
WebCrawlerSettings GetWebCrawler(string webCrawlerId);

[OperationContract]
bool StartCrawling(string webCrawlerId);

[OperationContract]
bool StopCrawling(string webCrawlerId);

[OperationContract]
string GetWebCrawlerStatus(string webCrawlerId);

[OperationContract]
List<Content> GetContentIndexed(string webCrawlerId, int itemsPerPage, int pageNumber);

[OperationContract]
Content GetContentIndexedFrom(string website);
```

```
[OperationContract]
Content GetContentIndexedFrom(string website, string webCrawlerId);
[OperationContract]
bool DeleteContentIndexed(string webCrawlerId);

[OperationContract]
string GetTextIndexedFrom(string website);

[OperationContract]
string GetTextIndexedFrom(string website, string webCrawlerId);

[OperationContract]
bool DeleteContentIndexedFrom(string website);

[OperationContract]
List<Content> Search(string website, string termToSearch, int itemsPerPage, int pageNumber);


[OperationContract]
bool Start();

[OperationContract]
bool Stop();
```

AddOrUpdateWebCrawler: permite añadir o actualizar un crawler concreto, de forma que, si no existe, se crea su repositorio, su cola y su índice

DeleteWebCrawler: permite borrar un crawler concreto, junto con su repositorio (y el contenido que haya en él), su cola de mensajes y su índice.

ContainsWebCrawler: permite saber si en los crawlers que tenemos almacenados, existe el crawler que estamos buscando

GetWebCrawlers: obtiene una lista de todos los webcrawlers almacenados

StartCrawling: inicia el ciclo recorrido - indexación por parte del crawler especificado. En este crawler ya estará indicada la página objetivo. Devuelve true en caso de que se inicie sin problemas.

StopCrawling: detiene el ciclo recorrido - indexación de un crawler concreto, pudiendo reiniciarlo mediante StartCrawling en cualquier momento. Devuelve true en caso de que se haya detenido correctamente.

GetWebCrawlerStatus: devuelve el estado del webCrawler indicado (running, stopped...).

GetContentIndexed: devuelve una lista de los contenidos indexados por el crawler especificado; esta lista estará paginada.

GetContentIndexedFrom: se devuelve el contenido indexado de la página especificada; si se especifica qué crawler la indexó, se ahorra tiempo ya que el servicio no necesita buscarlo.

GetTextIndexedFrom: devuelve el texto plano que se haya indexado de la URL especificada; en realidad, hace la misma llamada que GetContentIndexedFrom pero devolviendo únicamente el ContentText. Igual que en GetContentIndexedFrom, si se especifica el crawlerId, se ahorra tiempo.

DeleteContentIndexedFrom: permite borrar el contenido indexado de una página, tanto del índice como del repositorio.

Search: permite realizar búsquedas. Buscará en las páginas hijas del website especificado (incluyéndolo) el término especificado y devolverá un resultado paginado.

Start y **Stop:** arrancan y paran el servicio, respectivamente.

WebCrawlerController

Además del servicio, se implementa una API para proporcionar el acceso transparente al WebCrawlerService. En el modelo MVC, es el controlador, como su propio nombre indica.

WebCrawlerController tiene prácticamente los mismos métodos que WebCrawlerService, ya que simplemente se limita a llamar a éste cuando se realiza una petición y a devolver el resultado que el servicio devuelve. Es, básicamente, una interfaz que llama al servicio.

Se podría decir que es un controlador "tonto", ya que toda la lógica se encuentra en el servicio exceptuando las más simples verificaciones (valores no nulos, enteros mayores que 0...).

2.4 Pruebas

Como es habitual en Karira, se diseñaron una serie de pruebas unitarias y automatizadas que comprueben que el código está libre de errores, tanto al nivel de detalles de implementación como a nivel e funcionalidad.

Para ello se utiliza el framework Nunit que se ha comentado anteriormente, junto con Resharper[33] que, aparte de toda la funcionalidad que aporta para el desarrollo con Visual Studio, aporta un módulo para integrar en Visual Studio la posibilidad de lanzar las pruebas de frameworks alternativos a los de .Net.

[33] http://www.jetbrains.com/resharper/

Gracias a este plugin, lanzar las pruebas es tan sencillo como pulsar el “play”, además de ofrecer también la posibilidad de hacer un Debug.

En cuanto al WebCrawlerService, se diseñaron pruebas tanto para comprobar el correcto funcionamiento de la indexación, las búsquedas, del webcraling y el funcionamiento del servicio en sí.

Se separaron en tres clases, de forma que las pruebas referentes a WebCrawlerData están en la clase WebCrawlerDataTests, la referente al Crawling en sí se encuentra en ebCrawlerTests y las referentes al servicio y al API MVC se encuentran en WebCrawlerServiceTests.

CrawlerDataTests

Estos tests son relativamente cortos, ya que la indexación y búsqueda es segura gracias al uso de Lucene y la persistencia está garantizada gracias a Karira Framework y su uso de MongoDB.

Solamente se realizan dos pruebas, una en la que se indexa una página web, se comprueba que se ha indexado y se realiza una búsqueda por aproximación comprobando que nos devuelve la página indexada.

La otra consiste en guardar una página en Mongo y comprobar que efectivamente se guarda a través de WebCrawlerData sin ningún problema.

WebCrawlerTests

Para las pruebas de WebCrawlerTests se utilizó Microsoft Webmatrix para “montar” páginas web en local, de forma muy sencilla. Así se evitaron los primeros errores iniciales fruto de la conectividad y no del crawling, que era lo que se buscaba probar.

En las pruebas no se realiza el crawling de las páginas completas, si no que se realiza durante 30 segundos, se detiene y se realiza la búsqueda sobre lo indizado hasta el momento.

Esto se hace así para no ralentizar el resto de pruebas de Karira ECM, puesto que la página en local sobre la que se realizan las pruebas tiene un tamaño muy considerable y realizar el crawling completo cuesta alrededor de media hora.

Básicamente, del crawling se realizan 3 pruebas:

- Primera, en la que se realiza el crawling de una página repleta de textos (de alrededor de 20GB de textos) y se busca entre las páginas y los textos una palabra, esperando resultados.

- Segunda, en la que se realiza el crawling de www.karira.com, para comprobar el funcionamiento sobre una página real. La prueba consiste en buscar la palabra “karira” y obtener resultados. También se realizaron con las palabras clave “.Net” o “I+D”.

- Tercera, en la que de nuevo se realiza una búsqueda sobre una página en local, pero que no será la página padre típica si no una hija.

  Se comprueba que no se indexe ninguna que no corresponda con el criterio de filtrado y después se realiza una búsqueda relacionada.

2.5 Integración

La integración del WebCrawlerService y el WebCrawlerController con el resto de servicios y controladores que conforman el ecosistema de servicios de Karira ECM no tuvo ninguna complicación, puesto que:

- Los servicios están integrados en la clase Environment, a partir de la cual se accede a todos ellos y en la que son inicializados y arrancados

- Los controladores están integrados en Karira.ECM.API, constituyendo todos ellos una aplicación web a la que se accede mediante llamadas http habituales.

Por ello, para integrar el servicio había que incluir el código de inicialización en Environment y para integrar el controlador simplemente incluir la clase WebCrawlerController en el proyecto Karira.ECM.API y listo.

3. Balanceo y adaptación a picos de carga

3.1 Análisis

El alojamiento web en el mundo de internet es siempre una cuestión importante que plantearse a la hora de construir un sitio o una aplicación web, ya que tiene una gran repercusión en la difusión y la satisfacción del producto.

Esto es debido a que un servidor caro suele implicar una eficiencia mayor y por tanto, soportará una carga de usuarios mayor. Sin embargo, también repercutirá de forma importante en el precio del producto, puesto que el alojamiento se suele pagar mensual o anualmente y un alojamiento excesivamente caro para un producto pequeño, puede causar serios problemas a su éxito.

Lo mismo ocurre en el caso contrario, un alojamiento barato, suele implicar una carga menor de usuarios y de datos aunque se reducen costes. Sin embargo, frente a momentos de gran carga, este tipo de servidores reaccionan reduciendo enormemente la velocidad de respuesta o incluso "colgándose", una situación muy indeseable si queremos lograr una buena satisfacción con el producto.

Por ello, desde el punto de vista de Karira ECM se optó por utilizar los servicios de Amazon EC2[34].

Amazon, aparte de la famosa página web de compra y venta que empezó vendiendo libros, actualmente ha extendido su mercado al campo del cloud computing, ofreciendo toda una serie de servicios web que van desde el alojamiento, bases de datos en la nube, comercio electrónico...

Todos estos servicios están relacionados con un concepto tan en boga como es el cloud computing [35] o computación en la nube, que, básicamente, es un paradigma software que busca ofrecer servicios de computación a través de internet de forma transparente al usuario.

Entre estos servicios, destacó Amazon EC2, ya que se solucionaba de una forma elegante y sencilla el problema de rendimiento frente a coste.

Amazon EC2 es un servicio web que permite lanzar máquinas virtuales en la nube[36] de amazon. Por cada máquina virtual lanzada, amazon cobra por horas en función de si utilizamos

[34] http://aws.amazon.com/es/ec2/
[35] http://es.wikipedia.org/wiki/Computaci%C3%B3n_en_nube
[36] http://es.wikipedia.org/wiki/Computaci%C3%B3n_en_nube

una máquina Windows o Linux y si es de mayores capacidades o de menos. Estas máquinas virtuales pueden llegar a ser clústers de 23 Gb de memoria RAM, 33,5 unidades de cálculo (cada una con 2 x Intel Xeon X5570, arquitectura "Nehalem" de cuatro núcleos).

También se hicieron pruebas de conectividad una vez se tenían lanzadas máquinas y con cualquier test de velocidad de conexión disponible en internet se comprobó que la máquina tenía alrededor de unos 15 - 20 mb/s de descarga y más o menos la misma velocidad de subida, ppara un tipo de máquina "micro".

Además del hardware, que encarece la máquina virtual, ésta puede venir con un software preinstalado. Este software se especifica escogiendo una "Amazon Machine Image". Una amazon machine image es un tipo especial de sistema operativo predefinido que incluye también un software predefinido que se utiliza para arrancar máquinas virtuales en los servicios de Amazon.

Se puede escoger entre una gran cantidad de AMI's predefinidas[37] de diversos sistemas operativos y con la casi totalidad de configuraciones posibles de software para la implantación de un sistema web. También se puede crear una AMI propia, de forma que lancemos máquinas virtuales con nuestro propio software, en este caso concreto, con Karira ECM.

Pero lo más interesante de este modelo de alojamiento es que proporciona un sencillo servicio web que balancea la carga, el Elastic Load Balancing[38]. Este servicio se sitúa como "puerta" a través de la cual acceden las conexiones a la/s máquina/s virtual/es que tengamos lanzadas en Amazon EC2. Cuando detecta que se está produciendo demasiada carga es capaz de balancear la carga de las conexiones de datos entre las distintas máquinas virtuales.

Además, la facturación de este servicio se realiza en función del tráfico que soporta, por lo que sólo se factura exactamente el uso que realicemos y no un necesitaremos un sobre coste para soportar sobregargas.

Además, como es capaz de devolvernos el estado de las máquinas virtuales (si están sobrecargadas o incluso si se han colgado), nos ofrece la posibilidad de que, mediante el servicio EC2, lancemos otra máquina virtual y prevenir o recuperarnos rápidamente de un pico de carga.

Con este modelo, el alojamiento de una página web es adaptativo a las necesidades del momento. El siguiente ejemplo será bastante ilustrativo.

Con el 11 de septiembre o el 11 de marzo en España, todas las páginas webs y sistemas de la mayor parte de los medios de comunicación se sobrecargaron o incluso permanecieron caídos durante la avalancha de usuarios en busca de información. Esta situación, que podría haber

[37] http://aws.amazon.com/amis
[38] http://aws.amazon.com/es/elasticloadbalancing/

sido muy beneficiosa para cualquier periódico a pesar de la enorme tragedia, sólo repercutió en gastos para reparar los servidores sobrecargados.

La solución más extendida a este problema ha sido siempre la de tener un servidor enorme contratado o incluso propio, de forma que frente a una avalancha de conexiones sea capaz de soportarlas. Sin embargo, el resto de días que no sean ni 11 de septiembre ni 11 de marzo, la capacidad del servidor se estará utilizando en un porcentaje muy bajo, con lo que estaremos despilfarrando recursos la mayor parte del tiempo.

Con el modelo de Amazon EC2 y Elastic Load Balancing, somos capaces de tener un número reducido de máquinas (incluso con una sola de capacidad pequeña - mediana) y responder sin problemas de retardos a las conexiones habituales y, en caso de detectar una sobrecarga, lanzar todas las máquinas virtuales que necesitemos y responder sin problemas a este pico, "apagándolas" una vez finalizado.

De esta forma, se paga sólo por el servidor alojamiento que necesitamos en cada momento, siendo incluso capaces de responder a grandes cargas de trabajo.

Es por este motivo por el que se escogieron los servicios de Amazon como servidores de alojamiento.

En caso de querer consultar precios o máquinas consultar la documentación de Amazon EC2: http://aws.amazon.com/es/ec2/

3.2 Tecnologías utilizadas

AWS SDK

Para el desarrollo de software bajo la plataforma .Net Amazon nos proporciona un SDK que proporciona toda una serie de librerías que facilitan en una gran medida la tarea del desarrollo de software que utilice o que integre cualquiera de los servicios que ofrece Amazon.[39]

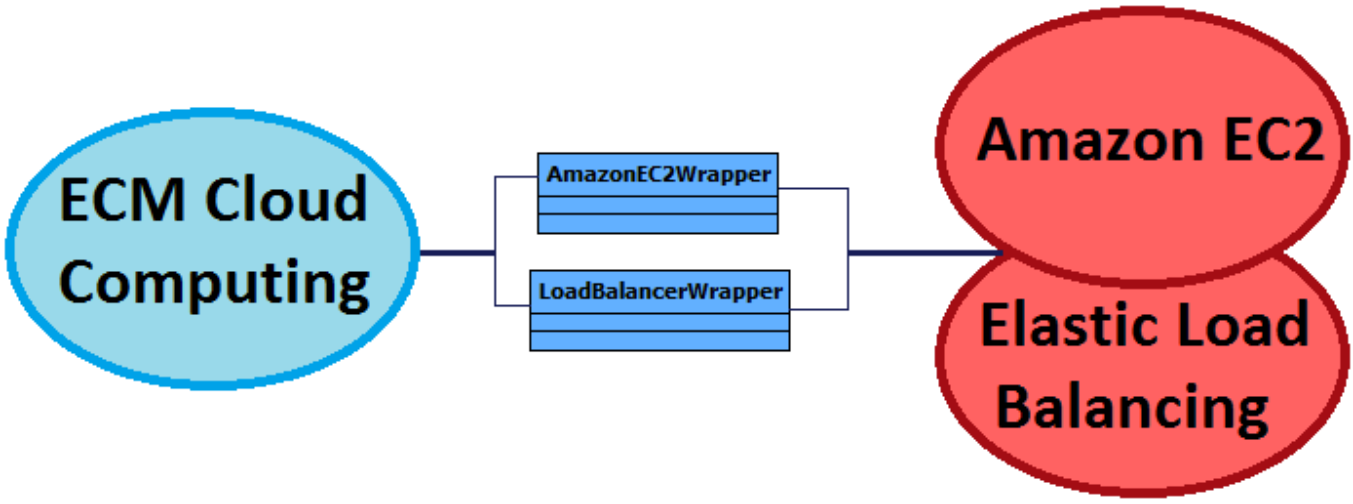
Además, incluye toda una serie de ejemplos de código que incluso facilitan más la tarea.

Sin embargo, a pesar de todo en un comienzo el adaptarse al AWS SDK resulta una tarea un poco complicada, puesto que tiene una forma un tanto curiosa de hacer las cosas, pero, desde luego, frente al desarrollo de toda una API que se adaptase a los servicios de Amazon, es muy preferible este enfoque.

[39] http://aws.amazon.com/es/

3.3 Diseño

Para el uso de cloud computing en Karira ECM se ha seguido la siguiente arquitectura:



Como se puede observar, se hace uso de los dos servicios de Amazon, Amazon EC2 y Elastic Load Balancing desde unas clases adaptadoras (que se explicarán en el siguiente apartado), aportando así la funcionalidad de estos servicios para la integración del Cloud Computing en Karira ECM.

3.4 Implementación

Para la integración con los servicios de Amazon, se optó por escribir el código en Karira Framework, de forma que el framework sea capaz de ofrecer esta posibilidad de integración a cualquier proyecto de Karira incluyendo, por supuesto, a Karira ECM.

Para la integración de los servicios Amazon EC2 y Elastic Load Balancing con Karira Framework y que pueda ser utilizado por Karira ECM sin ningún problema, se optó por la construcción de dos clases adptadoras: AmazonEC2Wrapper, LoadBalancerWrapper.

AmazonEC2Wrapper

AmazonEC2Wrapper es una clase construida mediante el diseño Adapter, reseñado anteriormente, que busca dar toda la funcionalidad necesaria para lanzar, detener, reiniciar y conocer el estado de una máquina virtual de Amazon.

Estas funcionalidades son las más básicas que debe aportar la integración con Amazon EC2.

Sin embargo, según fue avanzando el desarrollo se detectó que, al lanzar la máquina todos los puertos se encuentran cerrados al exterior y sólo se puede contactar con la máquina desde el interior de la nube de Amazon (desde otra máquina en EC2, desde la AWS Management Console[40]...)

Investigando, se descubrió que el problema residía en que el firewall de una máquina de Amazon se basan en "Security Groups"[41].

Un security group en amazon es un conjunto de reglas de aceptación de conexiones entrantes en las que se especifica el puerto, el protocolo y el rango de direcciones IP permitidas. Este rango se especificará mediante una direccion IP y su máscara[42], de forma que se especificará una subred de internet. Por ejemplo, si se define la subred como 0.0.0.0/0 se permitirán conexiones entrantes desde cualquier dirección de Internet.

Además, permite la configuración de máquinas en roles (de hecho, es lo recomendado) de forma que se puede asignar diversas máquinas a un rol llamado "Mongo Servers" que tenga abierto el puerto 27017 (por defecto en MongoDB) para el protocolo tcp para cualquier dirección IP.

De ahí su nombre, "security groups", ya que son reglas de seguridad definidas para un conjunto de máquinas que se asignen al grupo; los protocolos permitidos son tcp, udp e icmp.

De esta forma, la seguridad de nuestras máquinas se incrementa enormemente, puesto que no tenemos unas máquinas abiertas o cerradas completamente a Internet, si no un conjunto de grupos de seguridad configurables.

Además, Amazon ofrece la posibilidad de obtener una IP estática, algo también bastante interesante para un servidor web.

Otro concepto importante de Amazon EC2 es la zona de disponibilidad. Una máquina se puede situar en diversas zonas de disponibilidad, que a su vez se agrupan en regiones.

Estos dos conceptos definen la localización geográfica de la máquina y además garantiza que si se produce un fallo en otra zona o incluso en otra región, no afecte en absoluto a la nuestra. Además, será interesante situar la/s máquinas en zonas cercanas a nuestro mayor número de usuarios.

---

[40] http://aws.amazon.com/es/console/

[41] http://www.somic.org/2009/09/21/security-groups-most-underappreciated-feature-of-amazon-ec2/

[42] http://en.wikipedia.org/wiki/Subnetwork

Las regiones actuales son: EE.UU. Este (Norte de Virginia), EE.UU. Oeste (Norte de California), UE (Irlanda) y Asia Pacífico (Singapur).

Las zonas de disponibilidad disponibles (valga la redundancia) van en función de nuestra cuenta en los servicios de Amazon, por lo que lo más interesante es consultarlo en la consola de amazon.

Aún así, AmazonEC2Wrapper ofrece un método para consultarlas.

En vista de la interfaz de IAmazonEC2Wrapper será mucho más comprensible:

```
interface IAmazonEC2Wrapper
{
string CreateInstance(string instanceType, string ImageId, string AvailabilityZone);
string CreateInstance(string instanceType, string imageId, string availabilityZone, string
securityGroupName);

void StartInstance(string instanceId);
void ShutDownInstance(string instanceId);

void TerminateInstance(string instanceId);
void TerminateAllInstances();

string GetInstanceState(string instanceId);
bool IsRunningInstances();
List<RunningInstance> GetRunningInstances();

void OpenPort(string instanceId, int port, string protocol);
void OpenPort(string instanceId, int port, string protocol, string ipMask);

string GetAssignedGroup(string instanceId);

string GetInstanceIp(string instanceId);
void AssociateAddress(string instanceId);
bool ReleaseAddress(string ip);

string[] GetAvailabilityZones();
}
```

Constructor: a pesar de que no aparece en la interfaz, es interesante comentarlo.

El constructor necesita como parámetros la clave pública y privada de nuestra cuenta de Amazon para poder conectarse a los servicios que se ofrecen, concretamente a Amazon EC2.

CreateInstance (sobrecargado): el método CreateInstance devuelve el Id de la máquina virtual de AmazonEC2 que crea, indicándole:

- instanceType: el tipo de instancia de máquina virtual que deseamos lanzar (pequeña, grande, mediana...). Los tipos posibles están indicados en la clase EC2InstanceType.

  Sin embargo, en función de la imagen que incluyamos en la máquina, estaremos supeditados a un/os tipos de instancias, ya que si queremos una máquina con Windows Server 2008, IIS, ASP, etc es muy posible que necesitemos una máquina de tamaño medio..

- imageId: el Id de la AMI (imagen de la máquina) que queremos incluir como software en la máquina virtual. Hay algunos ejemplos en EC2Amis, pero una lista más completa la podemos encontrar en la página oficial de AMIS de amazon, aquí.

  También en The Cloud Market, una página que recopila un catálogo mucho más completo que el presente en la página oficial de las AMI's de amazon.

- avalaibilityZone: zona de disponibilidad en la que queremos que se sitúe nuestra máquina. Hay algunos ejemplos de zonas en la clase EC2AvailabilityZone, aunque para consultar las que podemos utilizar, se debe llamar al método GetAvailabilityZones().

- securityGroupName: si se indica, se asocia inmediatamente la máquina virtual que se está lanzando a este grupo de seguridad, con todas las características que éste tenga.

  En caso de que no se indique, se crea un grupo de seguridad único, puesto que su identificador será un Guid, y cuyo único miembro será la máquina creada.

StartInstance: en caso de que hayamos apagado la máquina, se puede encender llamando este método e indicándole el Id que queremos encender.

ShutDownInstance: en caso de querer apagar la máquina, se llamará a este método indicándole el Id.

Hay que tener en cuenta que una máquina apagada no implica que Amazon deje de cobrarnos por ella, aunque sí se reduce el consumo, algo que también se refleja en la factura. La única forma de que finalice la facturación por el servicio, es que la máquina termine, como se explica a continuación.

TerminateInstance: se realizará esta terminación de la máquina cuyo Id le sea indicado.

Terminar una instancia implica que se borrará completamente, por lo que perderemos todo el trabajo que hayamos realizado en ella. Viene a ser una especie de disolución del espacio

virtual que nos reserva amazon en la gran masa de la nube. Por ello, hay que ser muy cuidadoso.

Eso sí, una vez que se realiza esta llamada, la facturación de Amazon por esta máquina termina.

Hay que ser muy cuidadosos con este tipo de cosas, puesto que debemos evitar a toda costa tener una máquina corriendo porque la hayamos apagado en lugar de terminado o porque la llamada haya tenido algún error y no lo hayamos detectado.

TerminateAllInstances: esta llamada es más peligrosa incluso, puesto que termina todas las instancias de máquina que estén asociadas a nuestra cuenta de amazon. Es una especie de parada de emergencia, por lo que sólo en casos de extrema necesidad de destruir todas nuestras máquinas se debe utilizar.

GetInstanceState: devolverá el estado actual de la máquina que le indiquemos en instanceId. Algunos de los estados posibles son “running”, “stopped”, “terminating”...

IsRuninngInstances: indicará con un true si hay alguna máquina lanzada asociada a nuestra cuenta.

GetRunningInstances: devolverá una lista de RunningInstance que contendrá todas aquellas máquinas que estén en nuestro espacio reservado de Amazon (asociadas a nuestra cuenta) y cuyo estado sea “running”.

RunningInstance es una clase del AWS SDK que contiene una cantidad bastante importante de información acerca de la máquina virtual, que puede ir desde su estado o su Ip hasta el tipo de Ram, o el tipo de disco duro.

OpenPort: este método abre un puerto de la máquina que se le indique para el protocolo especificado. Estos 3 parámetros son obligatorios.

También puede especificarse la máscara Ip a las que permitirá el acceso, por ejemplo 0.0.0.0/0 para cualquier máquina.

Es importante saber que este método busca el grupo de seguridad al que esté asignada la máquina y abre el puerto de ese grupo. Por ello, hay que andarse con pies de plomo si esta máquina está incluida en un grupo de seguridad al que estén adscritas otras máquinas puesto que se abrirá el puerto para todas ellas.

Es por ello que es preferible que cada máquina tenga su propio grupo de seguridad, aunque haya que abrir después una por una sus puertos.

GetAssignedGroup: devolverá el identificador del grupo al que esté asignada la máquina

GetInstanceIp: devolverá la Ip de la máquina. Hay que tener en cuenta que pasa cierto tiempo hasta que la máquina tiene una Ip asignada desde que es creada, no desesperar, no suele superar los 5 minutos desde el arranque.

AssociateAddress: este método hace una petición a Amazon para que la máquina que se le indica pase a tener una Ip fija. Hay que tener en cuenta que una Ip fija tiene un precio de facturación, alrededor de 0,01 $ por hora.

ReleaseAddress: se libera la Ip de la máquina, reasignándosele otra. Este método libera tanto una Ip estática pedida mediante el método AssociateAddress como una Ip dinámica, de forma que cambiamos de Ip en cualquiera de los dos casos.

GetAvailabilityZones: devuelve un array de strings que contienen laz zonas de disponibilidad que podemos utilizar con la cuenta con la que nos hayamos logueado en los servicios de Amazon.

LoadBalancerWrapper

LoadBalancer es una clase diseñada mediante el patrón adaptador y que busca ofrecer una serie de métodos dirigidos a crear un balanceador de carga y a asignar máquinas al mismo.

Esta clase internamente realizará llamadas al servicio de Amazon Elastic Load Balancing[43], cuyas funcionalidades básicas son: detectar instancias de máquinas virtuales de Amazon EC2 en mal estado (sobrecargadas o con errores) y redirigir el tráfico hacia aquellas que se encuentren en buen estado.

De esta forma, en caso de "caerse" alguna de nuestras máquinas este hecho resultaría prácticamente transparente para el usuario.

De nuevo, en vista de la interfaz será sencillo captar la funcionalidad que aporta LoadBalancerWrapper:

```
void CreateLoadBalancer(string name, string protocol, Dictionary<int, int>
balancerPortInstancePort, List<string> ec2AvailabilityZone);

void CreateLoadBalancer(string name, string protocol, int loadBalancerPort, int
instancePort, string ec2AvailabilityZone);

void CreateUSALoadBalancer(string name, string protocol, int instancePort, int
loadBalancerPort);

void AsssignInstance(string instanceId, string loadBalancerName);
void DeassignInstance(string instanceId, string loadBalancerName);

void DeleteLoadBalancer(string loadBalancerName);
void DeleteAllLoadBalancers();

List<Instance> GetAllAssignedInstances();
bool IsAssigned(string instanceId);
bool IsAssigned(string instanceId, string loadBalancerName);
```

Constructor: en el constructor de la clase LoadBalancerWrapper se ha de indicar la clave púlibca y secreta de acceso a los servicios de Amazon de la cuenta que deseemos utilizar, para poder conectarse a Elastic Load Balancing con ellas.

CreateLoadBalancer: este método crea un balanceador de carga, llamando al servicio de Amazon Elastic Load Balancing. Los parámetros que se le pasan son:

[43] http://aws.amazon.com/es/elasticloadbalancing/

- name: nombre e identificador del balanceador que se creará. Esto quiere decir que el nombre debe ser único entre los balanceadores que tengamos asociados a nuestra cuenta.

- protocol: protocolo por el que el balanceador de carga estará escuchando. Por la especificación del servicio, este parámetro no podrá ser modificado durante el ciclo de vida del balanceador [44].

  La lista de protocolos disponibles para un balanceador de carga está disponible en la clase **LoadBalancerProtocol**.

- LoadBalancerPort - InstancePort: estos pares de puertos indican en qué puerto escuchará el balanceador de carga en las conexiones entrantes (las que provengan del exterior) y en qué puerto espera la instancia de la máquina que le lleguen las conexiones.

  Como se puede ver, se puede especificar un puerto de balanceador y uno de instancia o varios pares, encapsulados en un diccionario de pares de puertos.

- AvailabilityZone: este parámetro es bastante importante, puesto que el balanceador de carga la distribuirá entre las instancias que tenga asignadas **dentro** de las zonas de disponibilidad que se hayan definido al crearse.

  Por ello, habrá que tener en cuenta en qué zonas se encuentran las máquinas que queremos asignar al balanceador, para que éste contenga estas zonas.

  Como parámetro, al estar sobrecargado el método, se le podrá pasar una zona o una lista de zonas.

De esta forma, el método CreateLoadBalancer creará un balanceador que escuchará a un protocolo por uno o varios puertos, que estarán enlazados con sus correspondientes puertos en las máquinas.

Además, el balanceador podrá utilizarse para las zonas de disponibilidad que se definan en su construcción.

CreateUSALoadBalancer: la única diferencia de este método es que crea un balanceador en una zona de disponibilidad de Estados Unidos. Es un método de referencia, para en un futuro

[44] http://docs.amazonwebservices.com/ElasticLoadBalancing/latest/APIReference/index.html?API_CreateLoadBalancerListeners.html

tener un método para cada zona que deseemos, simplificando la tarea a la hora de hacer el servicio de Karira Framework referente a los servicios de Amazon.

AssignInstance: este método asigna una instancia de máquina al balanceador, para que la incluya en el balanceo que realiza entre todas las máquinas que tenga asignadas.

Se le pasan como parámetros el Id de la máquina y el nombre del balanceador.

DeassignInstance: desasigna una instancia de una máquina de un balanceador. Se le pasan el Id de la máquina y el nombre del balanceador del cual se quiere desasignar.

DeleteLoadBalancer: borra un balanceador de carga, indicándole el nombre del mismo. Se desasignan automáticamente todas las instancias asignadas, aunque ello no implica que dejen de funcionar ni mucho menos, simplemente no habrá un balanceo en caso de errores o sobrecargas.

DeleteAllLoadBalancers: se borran todos los balanceadores de carga asociados a la cuenta especificada en el constructor de la clase.

GetAllAssignedInstances: este método devuelve todas las instancias de máquina que hayan sido asignadas a **algún** balanceador.

IsAssigned: si sólo se especifica el Id de la instancia, comprueba si está asignada a **algún** balanceador. En caso de especificarlo, comprueba si está asignada al balanceador **concreto**.

Como se puede observar, únicamente con estas dos clases logramos tener casi toda la funcionalidad que el servicio necesita, puesto que somos capaces de lanzar máquinas, apagarlas, terminarlas y balancear la carga entre ellas de forma transparente.

Así, las conexiones siempre pasarían a través del balanceador, de forma que analice hacia qué máquina redirigirla en función del estado de todas ellas.

Internamente, tendríamos un conjunto de máquinas capaces de albergar uno o varios de los servicios de Karira ECM que, en caso de caer alguna de ellas, seríamos capaces de relanzarla detectando periódicamente su estado.

3.5 Pruebas

En cuanto a las pruebas automatizadas diseñadas para comprobar el correcto funcionamiento de AmazonEC2Wrapper y AmazonLoadBalancer, se diseñaron las siguientes pruebas en el proyecto Karira.Framework.AmazonTests en la clase AmazonEC2Tests:

- Primero, se lanza una máquina virtual de tipo “micro” y con el sistema operativo OpenSuse (Linux) para que pasar las pruebas no resulte prácticamente un coste a la empresa.

- Se verifica que la máquina está lanzada y se espera hasta que pasa al estado “Running”

- Una vez ya está corriendo, se abren todos los puertos bajo el protocolo icmp (para AmazonEC2, indicarle el puerto -1 es abrirlos todos) se pide la dirección Ip de la máquina y se realiza una petición de ping. En caso de realizarse con éxito, esto implica que se pueden abrir puertos sin problema y que la máquina tiene una Ip válida

- Después de ello, se crea un balanceador de carga y se asigna la máquina a éste. Se comprueba que esté asignada.

- Si es así, se abre un puerto de la máquina y se abre otro distinto del balanceador, pero indicándole a qué puerto de la máquina debe redirigir las conexiones.

- Se intenta hacer una conexión tcp a la máquina y si se logra, es porque el balanceador está redirigiendo correctamente las conexiones.

De momento no se ha realizado ninguna prueba de esfuerzo, debido a que la facturación del balanceador de carga se calcula en función de la cantidad de datos que pasen por él y resultaría costoso.

Paralelamente a las pruebas unitarias y automatizadas, también se realizaron otro tipo de pruebas más prácticas. Éstas consistían en intentar conectarse remotamente a la máquina virtual lanzada y lograr manejarla.

Estas pruebas se realizaron utilizando una máquina virtual lanzada con SUSE Linux Enterprise Server 11, concretamente esta : http://thecloudmarket.com/image/ami-e4a3578d--amazon-public-images-suse-linux-enterprise-server-11-sp1-64-bit

Se escogió SUSE por su íntima relación con Mono Project, que se explicará más adelante.

Para ello, a base de búsquedas, pruebas, errores y sudor, se definieron los siguientes pasos a seguir (estos pasos se definieron en febrero - marzo de 2011, no se garantiza que funcionen a la perfección a fecha de hoy):

1. Crear un KeyPair, si no se tiene uno, desde la AWS Management Console de Amazon
   1.1. Guardar el KeyPair público (.pem) en nuestro ordenador para usarlo más tarde

   2. Lanzar una máquina virtual de OpenSuse en AmazonEC2 con el anterior KeyPair
   2.1. Asociar una IP a la máquina (para que no nos la puedan quitar)
   2.2. Comprobar que tiene los puertos 80 y 22 abiertos para el protocolo tcp; en caso contrario, abrirlos

   3. Descargar Bitvise Tunnelier (http://www.bitvise.com/download-area)
   3.1. Instalarlo con las opciones por defecto
   3.2. Abrir BitVise Tunnelier
   3.3. En Server, poner la dirección de la máquina (dejar el puerto por defecto)
   3.4. En authentication, poner **root** y como Initial Method, **public key – slot 1**
   3.5. Pulsar User keypair Manager
   3.6. Pulsar Import…
   3.7. Seleccionar el KeyPair que hemos guardado antes
   3.8. Volver a la ventana principal y pulsar sobre Login
   3.9. Se nos abrirá una ventana de ssh con la consola y otra con la lista de archivos de la máquina

A partir de aquí, si no se dice lo contrario, las acciones son en la máquina remota

   4. Introducir en la consola:
   4.1. zypper install –t pattern gnome (instala gnome para ver el escritorio con nx)

   5. Instalar nxserver introduciendo los siguientes comandos en la consola:
   5.1. zypper install http://64.34.161.181/download/3.4.0/Linux/nxclient-3.4.0-7.i386.rpm
   5.2. zypper install http://64.34.161.181/download/3.4.0/Linux/nxnode-3.4.0-16.i386.rpm
   5.3. zypper install http://64.34.161.181/download/3.4.0/Linux/FE/nxserver-3.4.0-17.i386.rpm

   6. Realizar los siguientes pasos para iniciar sesión con nx-client:
   6.1. En la consola remota:
      6.1.1. Dar una contraseña a root: passwd root
      6.1.2. Introducir, por ejemplo, 1234
      6.1.3. Crear un usuario nuevo: useradd -p 1234 -d /home/karirauser1 -m karirauser1
      6.1.4. Asegurar que tiene contraseña: passwd karirauser1
      6.1.5. Si no tiene, introducir 1234 (por ejemplo)
   6.2. En **LOCAL**, Ir a la ventana principal de Bitvise Tunnelier
      6.2.1. Open new SFTP Window…
      6.2.2. Acceder a la carpeta /usr/NX/share/keys
      6.2.3. Copiar a nuestra máquina local el archivo server.id_dsa.key

   6.3. En **LOCAL**:
      6.3.1. Descargar nxclient (http://www.nomachine.com/download-client-windows.php).
      6.3.2. Instalarlo con las opciones por defecto.
      6.3.3. Configurar una sesión con el nombre que se le quiera dar
      6.3.4. Host: dirección ip de la máquina con suse de Amazon
      6.3.5. Escoger Unix – **Gnome**

6.3.6. Pulsar Configure…
6.3.7. Comprobar que el escritorio Unix sea Gnome
6.3.8. Pulsar sobre Key…
6.3.9. Pulsar Import…
6.3.10. Abrir el archivo **server.id_dsa.key** que nos hemos descargado anteriormente y pulsar **Save**
6.3.11. Cerrar la configuración y poner como usuario de conexión karirauser1 y contraseña 1234
6.3.11. Pulsar Connect

Una vez completados estos pasos, nos habremos conectado a nuestra máquina en Amazon bajo OpenSuse y seremos capaces de instalar cualquier programa, descargar cualquier software o hacer cualquier cosa remotamente.

Una opción para configurar una máquina a nuestras necesidades podría ser perfectamente mediante estos pasos. Sin embargo, en el caso de querer lanzar varias máquinas con una configuración determinada, lo más interesante es crearse una AMI (Amazon Machine Image) adaptada a las necesidades concretas.

Este proceso de creación de una AMI se encuentra detallado en este tutorial de Amazon:
http://docs.amazonwebservices.com/AmazonEC2/dg/2006-06-26/creating-an-ami.html

4. Compatibilidad bajo Mono

Como es natural en casi cualquier proyecto informático, un factor muy importante para el éxito del producto es su capacidad de adaptarse a cualquier plataforma.

Sin embargo, a pesar de las grandes ventajas y la comodidad del trabajo con la plataforma .Net, esta plataforma es propiedad de Microsoft y las políticas de Microsoft en cuanto a la compatibilidad con otras plataformas distintas de Windows son de sobra conocidas.

Esta actitud es también algo relativamente natural, puesto que si tienes un proyecto tan sumamente costoso como es Windows no vas a desarrollar para otra plataforma que no sea la tuya, tanto por "amor propio" como por desarrollar ampliaciones y software que dé motivos para su uso.

Sin embargo, es muy interesante lograr esta compatibilidad si se quiere abarcar un mercado mayor, además de que a nivel económico hay una gran diferencia entre desplegar una aplicación en un sistema Linux que en un sistema Windows. Sólo la diferencia entre la licencia de Windows y la licencia de la distribución más cara de Linux ya sería importante, sin contar con Internet Information Server[45] o cualquier otra licencia.

Es por ello que sería importante lograr esta compatibilidad.

A pesar de que en un primer momento esto parezca un tanto imposible, gracias al proyecto Mono[46] dirigido por Miguel de Icaza[47] se posibilita esta compatibilidad.

El proyecto Mono, como se explicó al comienzo de esta memoria, es una implementación Open Source de los estándares ECMA[48] para Microsoft C#.

Este proyecto de código libre implementa toda la infraestructura de Microsoft .Net, además de diversas clases orientadas únicamente a las funcionalidades de Linux. Por ejemplo, clases relacionadas con el uso de LDAP, Gtk+...

Básicamente, está orientado al desarrollo de aplicaciones bajo la plataforma .Net compatibles con Linux, de forma que se obtenga una mayor difusión tanto de Linux como de la plataforma .Net.

---

[45] http://es.wikipedia.org/wiki/Internet_Information_Services
[46] http://es.wikipedia.org/wiki/Proyecto_Mono
[47] http://es.wikipedia.org/wiki/Miguel_de_Icaza
[48] http://www.ecma-international.org/

Mono Project incluye una máquina virtual, un compilador “Just in time”, un biblioteca de clases que funciona para cualquier lenguaje CLR[49], un compilador de C#...

Gracias a todo ello, Mono soporta tanto casi cualquier sistema operativo (Windows, Linux, Mac OS X, BSD, Nintendo WII, Playstation 3, Apple IPhone) como casi cualquier arquitectura (x86, x64, sparc 32...) dotando de una compatibilidad envidiable.

Por todo ello, se tomó la decisión de hacer compatible Karira Framework y Karira ECM bajo Mono, para lograr la mayor capacidad multiplataforma posible.

4.1 Adaptación a Mono

En un primer momento se pensó que Karira ECM y Karira Framework necesitarían de una seria remodelación para lograr funcionar bajo Mono.

Sin embargo, pronto quedó patente que no sería necesaria ninguna remodelación, si no sencillamente ser capaces de lanzar las pruebas unitarias y automatizadas de ambos para comprobar que funcionaban correctamente.

Para comprobar este, se optó por utilizar una máquina virtual (local) con Open Suse de 64 bits. Esta elección de Open Suse se realizó debido a que la compañía Novell [50] es la propietaria o al menos el principal sponsor del proyecto Mono y de Open Suse.

Debido a ello, Open Suse en sus últimas versiones incluye las últimas distribuciones de Mono y diversas herramientas relacionadas, por lo que facilita en gran medida la tarea.

En la máquina se instaló Mono Develop[51], un IDE diseñado para construir aplicaciones bajo Mono y que, en nuestro caso se utilizará para comprobar el funcionamiento de las pruebas.

Otra opción habría sido la interfaz gráfica de Nunit, pero se quería investigar la posibilidad de desarrollar directamente bajo Linux.

De hecho, la elección de Nunit como framework de pruebas fue precisamente porque Mono Develop y Mono proporcionan la integración de Nunit para el desarrollo de pruebas unitarias.

---

[49] http://es.wikipedia.org/wiki/Common_Language_Runtime
[50] http://www.novell.com/home/
[51] http://monodevelop.com/

Sin embargo, el framework de Visual Studio para las pruebas no está soportado, por lo que sí que hubo que hacer una pequeña migración de las pruebas que existían en el framework de Visual Studio a Nunit, que realmente no fue nada traumático puesto que la sintaxis es prácticamente la misma.

Una vez instalado Mono Develop, se abrió la solución de Karira ECM y, debido a problemas de sintaxis de Visual Studio, precisamente el proyecto de pruebas no lograba abrirlo.

Por ello, se tuvo que crear un nuevo proyecto de pruebas llamado KariraECM.Tests.Mono en el que se incluyeron todas las clases y referencias preexistentes y se lanzaron todas las pruebas, con un resultado inicialmente erróneo.

Algunos eran debido a ligeros errores en referencias a librerías de .Net pero el más grave era que el framework de logging DotnetLog[52] que se utilizaba en Karira Framework daba errores bajo Mono.

Por ello se tuvo que hacer un cambio y Karira Framework pasó a utilizar NLog[53] en caso de estar bajo Mono y DotNetLog bajo Windows.

Además de esto, al tener Karira ECM una arquitectura orientada a servicios, estos servicios han de ser desplegados en algún servidor web para poder realizar pruebas también sobre los servicios.

En Windows, se utiliza Internet Information Server algo que, por motivos obvios, es imposible en Linux.

Para ello se exploraron 4 servidores:

- **XSP[54]:** en este caso Mono proporciona un servidor muy sencillo llamado XSP[55] que permite desplegar aplicaciones ASP.Net en Mono y Windows sin prácticamente ningún problema.

  Sin embargo, no es especialmente eficiente.

- **Apache Server[56]:** Mono proporciona un módulo para que Apache Server funcione con aplicaciones ASP .Net llamado Mod mono[57]. De esta forma, el despliegue con Apache

[52] http://www.theobjectguy.com/dotnetlog/
[53] http://nlog-project.org/
[54] http://www.mono-project.com/ASP.NET#ASP.NET_hosting_with_XSP
[55] http://www.mono-project.com/ASP.NET#ASP.NET_hosting_with_XSP
[56] http://httpd.apache.org/
[57] http://www.mono-project.com/Mod_mono

Server resulta un poco más complejo que la forma habitual, pero con las ventajas de eficiencia y resultados de Apache

- **Nginx**[58]**:** siendo uno de los servidores más utilizados en la red y con un rendimiento de soportar 500 millones de peticiones al día sin problema, la pregunta es ¿cómo no probarlo?.

  Se utiliza Nginx junto con fastcgi-monoserver de forma que ambos coexisten y Nginx redirige las peticiones a monoserver y las respuestas de éste hacia el exterior.

  De entre los servidores estudiados, era el que mejor resultados ofrecía.

- **Cherokee**[59]**:** al ser un servidor bastante liviano y moderno, sobre todo frente a Apache Server (con 15 años de antigüedad) era muy interesante probar sus resultados.

  Sin embargo, después de varios días intentando lograr su funcionamiento con Karira ECM no se consiguió, por lo que aún está pendiente el segundo asalto para lograr saber si resultaría interesante este servidor para el despliegue en Linux.

Finalmente, se lanzaron las pruebas de Karira ECM una vez bajo cada servidor, obteniendo en todos los casos (excepto bajo Cherokee) un resultado satisfactorio, observando que bajo Nginx el resultado era a simple vista mejor.

Sin embargo también es cierto que descubrir como realizar el despliegue bajo Nginx fue considerablemente más complejo que bajo Apache, sin tener en cuenta XSP (el más sencillo de todos, con mucha diferencia).

Si se quiere conocer los pasos necesarios para el despliegue de Karira ECM, consultar el anexo ***MVC under OpenSuse.***

[58] http://nginx.org/
[59] http://www.cherokee-project.com/

5. ContentCategorizationService

5.1 Análisis

Karira ECM, como sistema de gestión de contenidos, ofrece como característica diferenciadora la capacidad de categorizar contenidos de forma automática. El ContentCategorizationService o "Servicio de Categorización de Contenidos" permite discernir a qué categoria o lenguaje pertenece un contenido, definiendo en qué categorías se podrá clasificar.

Este servicio es especialmente útil cuando se tiene un gran abanico de contenidos preexistentes, de forma que su categorización manual es demasiado costosa en tiempo y/o dinero. En estos casos suele ser interesante una categorización rápida y sin intervención humana aunque no sea completamente precisa, de forma que se facilite la búsqueda de elementos.

Además, a la hora de redactar un artículo o incluso una nueva página en el sitio, será muy interesante que el servicio sea capaz de determinar con precisión a qué categoría conceptual pertenece, puesto que dotará al sistema de una considerable "inteligencia".

Otra de las capacidades del servicio será la categorización de textos en cualquier idioma, de forma que el servicio deberá ser capaz de discernir en qué idioma y además, dentro de este idioma, a qué categoría pertenece el texto.

En resumen, el servicio de categorización deberá ser capaz de **categorizar automáticamente** textos, independientemente del idioma.

5.2 Aprendizaje, categorización automáticos y minería de datos

Sin embargo, antes de entrar en materia, para comprender plenamente la complejidad de la categorización automática e, íntimamente relacionado, el aprendizaje automático, a continuación se realiza una breve introducción a estos dos temas tan densos e interesantes.

Minería de datos

Con la sociedad de la información actual, la cantidad de datos que nos rodean y a los que tenemos acceso actualmente no es abrumadora si no prácticamente infinita.

Los ordenadores están presentes en prácticamente todos los aspectos de nuestra vida, incluyendo cada vez en más ámbitos a los dispositivos móviles y otros tipos de dispositivos.

Todas nuestras interacciones con ellos generan una cantidad enorme de datos de ámbitos que pueden ser radicalmente distintos o incluso inimaginables.

Con la extensión impresionante de Internet, también inimaginable hace algunos años, todos estos datos se han convertido en accesibles para cualquier usuario. Además esta extensión ha creado otra fuente de datos que parece ser inagotable: las redes sociales y la web 2.0.

Es una cuestión fuera de duda que la cantidad de información disponible a través de internet es inabarcable para una persona. Sería incluso cuestión de amplio debate si sería posible abarcarla por la humanidad.

Es por ello que día a día es más necesaria alguna forma de dotar a esta enorme cantidad de información un sentido, un significado.

Esta tarea tan sencilla para un ser humano (tan sencillo como leer un artículo sobre el último triunfo del equipo de fútbol local y tener claro que trata sobre deportes, que está relacionado con balones, equipos, trofeos, competiciones...) es de una complejidad bastante importante en el caso de que queramos automatizarla.

Puesto que esta tarea requiere de “inteligencia” y sentido común y, ¿cómo programar una máquina para que tenga estas cualidades?

Básicamente sobre este tema versa la minería de datos, de “entender” unos datos en bruto. Es decir, pasar de un conjunto de datos en bruto a información entendible por un ser humano.

A pesar de que en sí el concepto de entender requiere de una discusión filosófica muy profunda y que resulta interesante para este campo, definiremos que, básicamente, la minería de datos “entiende” los datos extrayendo patrones de comportamiento presentes en los datos.

Con ello, definiendo una serie de patrones de comportamiento, la minería de datos es capaz de extraer un modelo que dota de un sentido a los datos.

Además, será capaz de predecir futuras situaciones basándose en estos modelos de comportamiento.

Por todo ello la minería de datos es una disciplina muy importante actualmente, especialmente para la industria. Un ejemplo de ello podría ser uno típico para explicar la utilidad de la minería de datos:

En una cadena de supermercados resulta muy importante realizar estudios sobre hábitos de compra, por lo que se contabilizan los artículos comprados cada día e incluso de qué zona postal es el cliente que la compra.

Mediante minería de datos y sus técnicas estadísticas se pueden extraer conclusiones como por ejemplo que los clientes de la zona centro de la ciudad compran más artículos de cierto tipo cuando se acercan los meses de verano. Siguiendo esta conclusión se podrían hacer

campañas publicitarias, situar este tipo de productos al fondo del supermercado (para que tengan que recorrerlo al completo y ver el resto de productos), etc.

Asimismo y fuera del ámbito mercantilista (omnipresente), en un ámbito más científico la minería de datos se utiliza ampliamente en el campo de la genética, como por ejemplo para detectar la tendencia a padecer ciertas enfermedades con ciertas mutaciones genéticas.

Gracias a las técnicas de minería de datos se obtiene el aprendizaje antes mencionado, puesto que programando un software que aplique este tipo de técnicas a un conjunto de datos es capaz de generar un patrón de comportamiento con lo cual, está siendo capaz de "entenderlo".

Este patrón puede ser actualizado constantemente según los datos que el software vaya recibiendo, por lo que será capaz de ir desarrollando un "entendimiento" en función de los datos que vaya recibiendo por lo que, en función de éstos aprenderá a reconocer ciertos patrones y a predecirlos.

Al ser capaz de predecir estos comportamientos, esto implica que si enseñamos al software las características de unas categorías (cosas rojas, objetos cuyo color sea de un tono rojizo, por ejemplo) a fuerza de mostrar qué datos entran en cada una de estas categorías más tarde será capaz de observar las características de estos datos y clasificarlos por si mismo.

A pesar de que esta afirmación parece de ciencia ficción, actualmente es un hecho.

Sin embargo estas técnicas, pese a unos resultados muy aceptables, no alcanzan el nivel de la ciencia ficción de crear una inteligencia artificial. Aún se encuentra en estadios prematuros en los que es capaz de imitar los procesos cognitivos de un ser humano en la infancia: reconocer patrones simples, actuar en función de ellos y esperar que se produzcan en futuras situaciones.

Aún queda pendiente el siguiente paso, la madurez de este reconocimiento "sencillo".

5.4 Tecnología utilizada: WEKA

Para cumplir con las necesidades planteadas, se optó por la utilización de Weka.

Weka es una plataforma de software para el aprendizaje automático de sistemas software, así como la minería de datos (dos campos íntimamente ligados).

Básicamente, Weka es una instántanea del estado del arte actual en el campo de la minería de datos y del aprendizaje automático en el que se incluyen los algoritmos y técnicas más importantes en este campo (no por ello implica que sean pocas).

Proporciona toda una serie de capacidades, como puede ser el clustering[60], el análisis de regresión[61] e incluso su propia interfaz gráfica de forma que puede ser utilizada directamente.

También incluye herramientas para preprocesar conjuntos de datos, como por ejemplo el procesar un texto y añadir una serie de datos para indicarel número de ocurrencias de determinadas palabras o incluso de carácteres.

Sin embargo, la característica importante de Weka es que en sus librerías incluye toda una serie de capacidades que ofrecen soporte a la categorización de texto. Por ello fue escogida para este servicio, además su rendimiento y su facilidad de uso (transparencia total en cuanto a los complejos algoritmos que conllevan estos aspectos).

Además Weka es OpenSource y escrita en Java, algo que no resulta un impedimento gracias a IKVM.

Clasificadores de Weka

Weka implementa toda una serie de clasificadores basados en algoritmos cuya finalidad es la de clasificar instancias y aprender incrementalmente.

En función del algoritmo que se escoja para realizar este proceso, el rendimiento y los resultados del software variarán considerablemente, por lo que es importante conocer mínimamente los algoritmos que implementa y sus características básicas.

Estos clasificadores están divididos en distintas categorías o, mejor dicho, familias.

[60] http://en.wikipedia.org/wiki/Data_clustering
[61] http://es.wikipedia.org/wiki/An%C3%A1lisis_de_la_regresi%C3%B3n

Clasificadores Bayesianos

Los clasificadores bayesianos, como su propio nombre indica, están basados en el teorema de Bayes. Este teorema tiene una repercusión en el campo de la probabilidad y la estadística matemáticas muy importantes, algo que se refleja también en los campos de la minería de datos y la inteligencia artificial.

De hecho, como se verá más adelante son este tipo de algoritmos los que mejores resultados han arrojado en las pruebas.

En esta familia de casificadores tenemos:

- Naïve Bayes o Bayes Ingenuo[62]: aplica el teorema de Bayes utilizando una fuerte dependencia de atributos en ello. Esto implica que considera que la ausencia o presencia de un atributo no influye en la presencia o ausencia de otros.

  Esto quiere decir por ejemplo, que si para ser un balón de fútbol las características a cumplir con ser esférico, con cuadros blancos y negros y tener un cierto diámetro, se observarán únicamente estas características y de forma independiente unas de otras con lo que con tener un círculo del diámetro adecuado y a cuadros blancos y negros estaríamos muy cerca de que sea un balón de fútbol (algo poco realista).

  A pesar de ello, se ha comprobado que Naïve Bayes arroja unos resultados muy positivos, paradójicamente.

- NaïveBayesUpdateable: es una versión incremental que procesa una instancia a la vez; puede usar un estimador kernel, pero no discretización.

- NaïveBayesMultinomial: en este algoritmo se tiene en cuenta tanto las palabras que aparecen en un documento como el número de ocurrencias de las mismas. En general, funciona mejor que el estándar particularmente para diccionarios de gran tamaño.

- NaïveBayesMultinomialUpdateable: como indica su nombre, una implementación conjunta de los dos algoritmos.

- AODE[63] : busca solucionar el problema de la independencia de atributos del algoritmo NaïveBayes. Puede ser capaz de arrojar mejores resultados que este último en conjuntos de datos con atributos no independientes.

- BayesNet: es capaz de aprender redes Bayesianas[64]. Para definir las tablas de la red se pueden utilizar algoritmos basados en “Hill Climbing”[65] o incluso algoritmos genéticos.

[62] http://en.wikipedia.org/wiki/Naive_Bayes_classifier
[63] http://en.wikipedia.org/wiki/AODE
[64] http://es.wikipedia.org/wiki/Red_bayesiana
[65] http://en.wikipedia.org/wiki/Hill_climbing

Árboles

Como el nombre indica, estos clasificadores se basan en la construcción de un árbol de decisión que se irá adaptando en función de los cambios que se produzcan.

Dentro de la familia de clasificadores basados en árboles tenemos:

- **ADTree**: construye árboles de decisión alternantes. Además, utiliza Boosting[66] y está optimizado para problemas de "dos clases" (o una u otra).
- **DecisionStump**: construye árboles de decisión de un sólo nivel para conjuntos de datos con clases numéricas o categóricas. Se enfrenta a valores "perdidos" tratándolos como un valor separado y extendiendo una tercera rama desde la principal.
- **Id3**: algoritmo básico "divide y vencerás" de árboles de decisión.
- **J48**: implementa el algoritmo de aprendizaje en árboles C4.5[67], el cual es una actualización del algoritmo Id3. La útlima actualización de este algoritmo, C5.0, ha resultado ser el caballo de batalla de la industria de las máquinas de aprendizaje.
- **LMT**: contruye árboles de modelo logístico.
- **M5P**[68]: modelo de árbol de aprendizaje que combina un árbol de decisión convencional con la posibilidad de utilizar funciones de regresión lineal en sus nodos.
- **NBTree**: construye un árbol de decisión con clasificadores NaïveBayes en las hojas.
- **RandomForest**: construye bosques aleatorios utilizando conjuntos de árboles aleatorios.
- **RandomTree**: construye un árbol que considera un número dado de características aleatorias en cada nodo.
- **REPTree**: un veloz árbol de aprendizaje que utiliza un "limpiados" con errores reducidos. Es similar al algoritmo C4.5, implementado con el clasificador J48.
- **UserClassifier**: permite a los usuarios construir su propio árbol de decisión.

[66] http://en.wikipedia.org/wiki/Boosting
[67] http://en.wikipedia.org/wiki/C4.5_algorithm
[68] http://www.opentox.org/dev/documentation/components/m5p

Reglas

Los clasificadores basados en reglas buscan construir una serie de reglas que les permitan ser capaces de dirimir las categorías en las que situar las instancias. En muchos aspectos, son muy similares a los basados en árboles de decisión e incluso hay híbridos entre ellos.

Dentro de la familia de clasificadores basados en reglas tenemos:

- **ConjunctiveRule**: clasificador simple de reglas conjuntivas. Aprende una única regla que predice tanto valores numéricos como nominales.

- **DecisionTable**: construye una tabla simple de decisión por mayoría.Evalúa subconjuntos de categorías usando la primera mejor búsqueda y puede utilizar validación cruzada para la evaluación.

- **JRip**: es un algoritmo RIPPER para una rápida y efectiva inducción de reglas. Un algoritmo RIPPER está basado en reglas y construye un conjunto de éstas que identifican las clases o categorías minimizando la cantidad de errore. El error está definido por el número de instancias de entrenamiento mal clasificadas por las reglas.

- **M5Rules**: obtiene reglas de modelos basados en árboles utilizando árboles del tipo M5.

- **Nnge:** este algoritmo utiliza el método de “el vecino más cercano” para la generación de reglas usando ejemplos generalizados no anidados.

- **OneR:** implementación del algoritmo 1R de obtención de reglas simples, con buenos resultados.

- **Part:** obtiene reglas de árboles de decisión parciales construidos con el algoritmo el tipo de árboles J4.8.

- **Prism**: implementa la cobertura más básica de la inducción de reglas.

- **Ridor**: aprende reglas con excepciones generando una regla por defecto, utilizando “limpieza” con error reducido para encontrar las excepciones con el menor índice de error, encontrando las mejores excepciones de cada excepción e iterando.

- **ZeroR**: predice la clase o categoría mayoritaria o el valor medio en caso de valores numéricos. Así de sencillo.

Funciones

En la familia de funciones tenemos la implementación de clasificadores que utilizan directamente funciones estadísticas y probabilísiticas.

Dentro de la familia de funciones tenemos:

- **LeastMedSq**: regresión robusto utilizando cuyo método es ir reduciendo la mediana (en lugar de la media) de los cuadrados de las divergencias de la línea de regresión.

- **LinearRegression:** implementación de la regresión lineal estándar.

- **Logistic**: una implementación alternativa a SimpleLogisitic para construir y utilizar un modelo logístico multinomial de regresión con un estimador de crestas para proteger de la sobreestimación penalizando los grandes coeficientes, basado en el trabajo de le Cessie y van Houwelingen[69].

- **MultilayerPerceptron**: retropropagación de redes neuronales[70].

- **PaceRegression**: construye modelos de regresión lineal utilizando regresión por pasos[71].

- **RBFNetwork:** implementa una función Gaussian radial básica de red.

- **SimpleLinearRegression:** desarrolla un modelo de regresión lineal basado en un único atributo.

- **SimpleLogistic:** construye modelos de regresión lineal con selección de atributos incorporada.

- **SMO:** implementa una optimización secuencial mínima para entrenar un clasificador de vector de soporte, utilizando en el núcleo de su implementación polinomios o funciones Gaussianas.

- **SMOreg:** algoritmo de optimización secuencial para problemas de regresión.

- **VotedPerceptron**[72]**:** algoritmo de redes neuronales “votadas”.

- **Winnow:** red neuronal dirigida por errores con actualizaciones multiplicativas.

---

[69] http://www.ncbi.nlm.nih.gov/pubmed/2277880
[70] http://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s
[71] http://researchcommons.waikato.ac.nz/handle/10289/1041
[72] http://www.sebastian-kirsch.org/moebius/blog/index.php/archives/2005/01/11/46/struggling-with-the-voted-perceptron/

Vagos

Estos clasificadores son llamados así debido a que únicamente almacenan las instancias que se les va pasando y no realizan ningún trabajo hasta que se llega al momento de la clasificación.

En la familia de los clasificadores “vagos” tenemos:

- **IB1:** este clasificador encuentra la instancia de entrenamiento más cercana en distancia Euclidiana a la instancia a clasificar dada y toma la clase o categoría de la instancia de entrenamiento como la que debe predecir para la que está categorizando.

- **IBk**: el mismo tipo de clasificador que IB1 pero no toma únicamente el “vecino más cercano”, si no que toma los k vecinos más cercanos.

- **KStar:** es del mismo tipo que los anteriores (vecino más cercano) pero con un método de función de distancia generalizada basada en transformaciones.

- **LBR:** es un clasificador Bayesiano que delega todo el trabajo a realizar al momento de la clasificación. Funciona con buenos resultados en pequeños conjuntos de datos.

- **LWL:** es un algoritmo genérico para el aprendizaje localizado con peso. Asigna pesos usando un método basado en instancias y construye un clasificador a partir de las instancias con peso.

Varios

Entre los algoritmo que no encajan adecuadamente en ninguna familia tenemos:

- **Hyperpipes:** se utiliza para problemas de clasificación discreta. Almacena el rango de valores observados en los datos de entrenamiento para cada atributo y categoría y extrae los rangos que contengan los valores de los atributos de una instancia a categorizar, escogiendo la categoría con un mayor número de rangos correctos.

- **VFI** (Voting feature intervals): construye intervalos alrededor de cada clase discretizando los atributos numéricos y utilizando intervalos de puntos para los nominales, guardando el conteo de categorías para cada intervalo de cada atributo y clasificando las instancias por “voto”. En teoría, es más veloz que Naïve Bayes pero menos que Hyperpipes.

5.5 Implementación

En cuanto a cómo se han cumplido las necesidades, se ha optado como se ha comentado anteriormente, por la construcción de un servicio web.

El servicio tendrá dos tareas fundamentales: entrenamiento y categorización.

Entrenamiento

El entrenamiento será transparente para el usuario final del sitio web. Sin embargo, es una tarea vital, puesto que sin entrenamiento la clasificación no funciona. Incluso con poco entrenamiento el porcentaje de error no es demasiado aceptable para el desarrollador, no hablemos del usuario final.

Por ello debe ser entrenado hasta un nivel aceptable, de forma que tenga los suficientes datos como para ser capaz de discernir tanto entre textos de cualquier idioma como en las categorías conceptuales que estén definidas. Sin embargo, mientras mayor sea la cantidad de datos que se manejen en el entrenamiento, mayor será el porcentaje de acierto a la hora de categorizar automáticamente.

Previamente al entrenamiento, los textos serán preprocesados por lo que en Weka son llamados “Filter”.

Estos Filter’s procesan los datos para que puedan ser procesados a su vez por los clasificadores. Hay una gran cantidad de filtros disponibles, aunque en Karira Categorization Service se utiliza únicamente uno: StringToWordVector.

Este filtro convierte un “String” a un vector de palabras a las cuales se les añade un número que indica el número de ocurrencias de la misma.

Además, a este filtro se le pueden añadir delimitadores o palabras a “esquivar” de forma que no se tomen en cuenta palabras como “el”, “de”, “la”, etc.

Este preprocesado es obligatorio, ya que se trabajará con textos o lo que es lo mismo, con String’s y ningún clasificador de Weka es capaz de trabajar con String’s directamente. Además, el filtro StringToWordVector proporciona el número de ocurrencias de la palabra, algo que resulta muy útil a la hora del entrenamiento.

Una vez realizado el preprocesado, quiénes realmente serán entrenados serán los clasificadores, ya que serán los encargados de las tareas de decisión. Será a estos clasificadores quienes tendrán como parámetros el conjunto de categorías y/o idiomas posibles y quienes al final de todo el proceso de entrenamiento devolverán los resultados.

El entrenamiento de los clasificadores es un proceso no bloqueante, de forma que se puede ordenar la construcción de los clasificadores y que en el momento en el que este proceso termine sean puestos a disposición de los usuarios de forma automática.

Clasificación

Una vez entrenados los clasificadores, el servicio será capaz de solicitarles la clasificación de textos. Esta tarea no conlleva practicamente coste, puesto que solicitar un resultado (una categoría en este caso) a un clasificador es algo muy veloz (sobre todo comparado con el tiempo necesario para el entrenamiento) con lo que la carga realmente estará en el proceso de entrenamiento.

El funcionamiento del proceso es el siguiente:

1. Se proporcionan contenidos de entrenamiento en bruto al servicio. Estos contenidos se componen de un texto, un conjunto de categorias a las que pertenece y un idioma en el que está escrito. El servicio en primer lugar purifica el texto eliminando las etiquetas HTML que pudieran existir, formatos y otros elementos similares. A continuación construye un objeto que contiene el texto, las categorias y el idioma y lo guarda dentro de un repositorio de instancias de entrenamiento.

2. Una vez se han procesado suficientes instancias, se puede construir un clasificador. Utilizando información acerca de las categorias o idioma en los que se quiere entrenar al clasificador, el servicio selecciona un conjunto de instancias de entrenamiento. A continuación construye un clasificador y lo entrena utilizando esas instancias.

3. Cuando el entrenamiento ha terminado, el servicio permite que se realicen preguntas al clasificador, de forma que se le proporciona una instancia de un texto y el clasificador indica a que categoria o lenguaje pertenece, indicando con qué porcentaje de confianza lo afirma.

5.6 Pruebas

En las pruebas iniciales con Weka, cuando todavía se estaba probando si sería interesante su uso o no en vista de sus resultados, se tomaron una serie de textos extraídos de noticias de periódicos online y se dividieron en dos, textos para entrenamiento y textos para categorizar.

Hay que tener en cuenta que, a pesar de ser unos 15 - 20 textos, este número es muy bajo para obtener unos resultados sólidos pero sí para unos resultados iniciales.

Las categorías que se definieron fueron:

- Deportes: los textos tenían que ver con fútbol, baloncesto o fórmula 1
- Rosa: relacionados con la boda de Kate Middleton (noticia del momento) o cotilleos varios sobre bodas
- Política: asuntos sobre gobiernos, EEUU, Obama...
- Tecnología: artículos sobre Iphone, Ipad y/o ordenadores en general.

El número y el tamaño de los textos en cada sección estaba equilibrado en el entrenamiento y la categorización, para evitar los típicos falsos positivos o negativos (aunque son inevitables en la inteligencia artificial, al igual que los errores en la inteligencia humana).

Estas pruebas se diseñaron para comprobar los resultados que Weka ofrecía, utilizando cada uno de los distintos y numerosos algoritmos que incluye.

Inicialmente se probó a entrenar el clasificador con los textos destinados al entrenamiento e introducir una frase, por ejemplo “La boda del famoso se realizará con anillos de oro y caballos” y que el clasificador devolviese la categoría que mejor puntuación sacase como posible resultado.

También devolvía la distribución entre las categorías posibles, que viene a ser una matriz de double que indica la “puntuación” que tiene cada categoría de ser la indicada.

Una vez hechas estas pruebas y comprobado que, efectivamente, Weka devolvía unos resultados que denotaban cierta inteligencia, se hizo una prueba un poco más exhaustiva utilizando los textos mentados.

La prueba consistía en entrenar el clasificador e ir pasándole los textos que no estuvieran entre los de entrenamiento para que los clasificase.

Como era conocida la categoría de estos textos, si el clasificador fallaba, se sumaba un error y cuando acabase con todos los textos se sacaba un porcentaje de aciertos.

Para conocer qué textos se utilizaron para realizar las pruebas, se encuentran en el archivo comprimido anexo Textos-Weka.

Esto se hizo con todos los posibles clasificadores de Weka (cada uno de ellos utiliza un algoritmo distinto) y se obtuvieron los siguientes resultados:

Name	Success %
bayes.BayesNet	20
bayes.NaiveBayes	90
bayes.NaiveBayesMultinomial	70
bayes.NaiveBayesMultinomialU...	70
bayes.NaiveBayesUpdateable	90
functions.GaussianProcesses	-1
functions.LinearRegression	-1
functions.Logistic	-1
functions.SGD	-1
functions.SimpleLinearRegressi...	-1
functions.SimpleLogistic	10
functions.SMO	60
functions.SMOreg	-1
functions.VotedPerceptron	-1
lazy.IBk	30
lazy.LWL	40
meta.AdaBoostM1	20
meta.AdditiveRegression	-1
meta.AttributeSelectedClassifier	30
meta.Bagging	40
meta.ClassificationViaRegression	30
meta.CostSensitiveClassifier	-1
meta.CVParameterSelection	30
meta.FilteredClassifier	30
meta.LogitBoost	40
meta.MultiClassClassifier	80
meta.MultiScheme	30
meta.RandomCommittee	50
meta.RandomSubSpace	30
meta.RegressionByDiscretization	-1
meta.Stacking	30
meta.Vote	30
misc.InputMappedClassifier	30
misc.SerializedClassifier	-1
rules.DecisionTable	20
rules.JRip	20
rules.M5Rules	-1
rules.OneR	20
rules.PART	20
rules.ZeroR	30
trees.DecisionStump	20
trees.J48	30
trees.M5P	-1
trees.RandomForest	60
trees.RandomTree	20
trees.REPTree	30
bayes.net.EditableBayesNet	20
bayes.net.BayesNetGenerator	20

Antes que nada, los resultados con un porcentaje de acierto de -1 son de aquellos algoritmos que no se pueden utilizar para categorizar texto.

Aparte de esto, se puede ver que los dos algoritmos con mejores resultados en este caso son NaiveBayes y NaiveBayesUpdateable, seguidos de cerca por meta.MultiClassClassifier.

Sin embargo en pruebas posteriores este resultado se desveló como un falso positivo, al contrario que los NaiveBayes.

De hecho, en pruebas posteriores todos los clasificadores de tipo NaiveBayes dieron unos resultados mejores que los del resto de algoritmos.

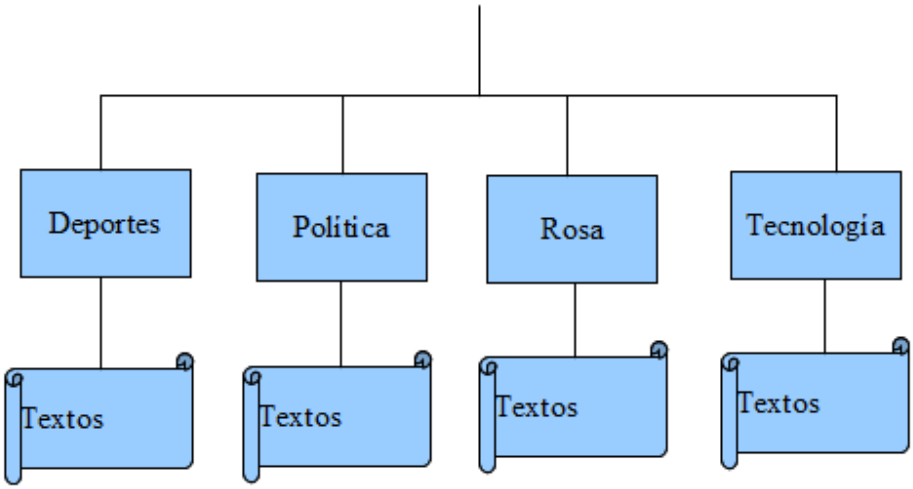
Esto es contrastable con otros proyectos de clasificación y categorización como por ejemplo Autonomy, cuya base algorítmica principal es el algoritmo de Bayes.

De hecho, en sus reuniones su primera diapositiva suele ser la fórmula del algoritmo de bayes.

En caso de querer recrear estos resultados, probar Weka o tomar muestras de uso de ella, se puede utilizar el proyecto de Visual Studio adjunto “WekaTests”.

Para los textos, tanto de entrenamiento como de prueba, comentar que para poder utilizarlos deben estar bajo una estructura de ficheros muy determinada.

Esto quiere decir que los directorios de los textos deben tener una estructura del siguiente tipo:



El nombre del directorio será también el de la categoría y los textos que estén dentro del mismo serán con los que se entrene al clasificador. Por supuesto, el nombre de los directorios podrá ser cualquiera.

Teniendo esto en cuenta, se pueden utilizar las pruebas sin problema.

6. Conclusiones

Con el desarrollo de este proyecto he trabajado en campos bastante distintos del desarrollo de software, ya que no tiene nada que ver el desarrollo de un motor de búsqueda con el desarrollo de un módulo dedicado al balanceo de carga en el servidor de alojamiento.

Sin siquiera nombrar el módulo de auto-categorización de artículos, con toda la teoría de inteligencia artificial y minería de datos que conlleva.

Es por ello que este proyecto me ha permitido ampliar en una medida que no imaginaba mis horizontes de conocimientos a diversos niveles:

- A nivel de desarrollo software, puramente, el desarrollo con arquitecturas orientadas a servicios, el desarrollo de toda una batería de pruebas unitarias y automatizadas, el uso de patrones de diseño en casos reales, refactorizaciones... Todo ello ha provocado que mis habilidades en el desarrollo de un software de calidad y mi experiencia en ello haya aumentado muy considerablemente, además de que mi interés por ello ha crecido con mis nuevos conocimientos.

- A nivel de conocimientos, el uso de tecnologías muy distintas de las utilizadas hasta el momento del desarrollo de Karira ECM, el estudio en profundidad necesario de las técnicas de inteligencia artificial y minería de datos y los continuos estudios e investigaciones de tecnologías a aplicar, ha provocado que hoy por hoy sea capaz de investigar y ser capaz de deducir si utilizar una tecnología o no en un proyecto, además de conocer toda una serie de tecnologías muy interesantes de aplicar en nuevos proyectos.

Aparte de estos nuevos conocimientos, el trabajo en un primer proyecto en un ambiente de aprendizaje tan rico y en un ambiente de trabajo tan fluido ha resultado una experiencia tremendamente satisfactoria.

En cuanto al trabajo a distancia, en ningún momento ha resultado ningún impedimento, puesto que las tareas que se me han ido asignando en el desarrollo y las “reuniones” para dudas, exposición de objetivos o posibles ayudas que necesitase, se han realizado a través de Skype.

Además estas ayudas y reuniones tengo la sensación de que se han realizado incluso con más asiduidad que en el caso de estar físicamente cercanos, debido a que por la sensación de alejamiento y de desconocimiento del trabajo que se va realizando, nos comunicábamos diariamente e incluso varias veces al día.

Personalmente, ha sido una experiencia muy enriquecedora a nivel personal y profesional.

7. Futuras mejoras

Como prácticamente en todos los proyectos software, en Karira ECM siempre queda trabajo por hacer, tanto en lo que concierne a mejoras del software ya desarrollado como en desarrollo de más módulos que se adapten a las necesidades del mercado y de los clientes.

En cuanto a las mejoras, a pesar de haber realizado un trabajo más intensivo en el apartado de auto-categorización, al ser un asunto tremendamente complejo y bastante experimental siempre quedan por realizar más estudios acerca de la eficiencia y resultados de este servicio.

También sería muy interesante realizar un entrenamiento manual intensivo, para que los clasificadores tengan una base de entrenamiento ya sea para presentaciones o incluso para una primera versión de despliegue.

También en el apartado del Crawling sería muy interesante realizar pruebas intensivas de rendimiento para mejorarlo, puesto que en un motor de búsqueda e indizador que deberá tratar con grandes cantidades de datos deberá ofrecer una respuesta rápida.

En cuanto al módulo de balanceo de carga, sería interesante realizar pruebas de estrés para comprobar el resultado que aportan, aunque ello acarrearía quizás un cargo económico extra a la empresa.

A pesar de todo, los resultados que aporta el software desarrollado son plenamente satisfactorios y están listos para un despliegue en el mundo empresarial sin ningún problema.

8. Agradecimientos

En primer lugar agradecer a mi tutor en la empresa Modesto San Juan Alvárez su guía sobre todo en los primeros meses de mi incorporación en Karira y en el proyecto Karira ECM, ya que en el campo del desarrollo software real mis conocimientos eran tremendamente limitados y gracias a su ayuda, he logrado conseguir un nivel en el que me siento cómodo para afrontar cualquier proyecto de desarrollo. También a mi compañero Pablo Torrejón su ayuda en los primeros meses (que también fueron los suyos) en los momentos en los que la inspiración divina no acudía.

Asimismo, a mi tutor en la Universidad Politécnica de Valencia César Ferri Ramírez por su guía y ayuda y el aprendizaje que en su momento tuve en la asignatura Metodologías de la Programación, bastante básica para este proyecto.

Agradecer también la existencia de páginas como www.stackoverflow.com , www.dofactory.com , los foros de ayuda de los servicios de amazon, Mono Project, todos los ejemplos de código sobre Crawling y sobre todo las enormes ayudas de google para obtener información.

Y, por último, pero no menos importante, a Carmen por soportar mis nervios al experimentar con estas tecnologías que tantos quebraderos de cabeza me han dado.

9. Bibliografía

- Data Mining - Practical Machine Learning Tools and Techinques.
  Ian H. Witten y Eibe Frank
  ISBN: 0-12-088407-0

- ActiveMQ in Action
  Bruce Snyderis
  ISBN-13: 978-1933988948

- The Definitive Guide to MongoDB
  Peter Membrey , Eelco Plugge , Tim Hawkins
  ISBN-13: 978-1430230519