#ReadThis - Un Servicio que lee las imagenes a través de Twitter

Mario Arias Escalona

Resumen: ....
....
....
....
....
....
....
....

Paraules clau—Paraules clau del projecte, màxim 2 línies. ....
....

Abstract: ....
....
....
....
....
....
....
....

Index Terms—Versió en anglès de les paraules clau. ....
....

1 INTRODUCCIÓN

EL proyecto es una iniciativa del Centro de Visión por Computador. Se trata de una institución pública líder en investigación y desarrollo en su campo. La Generalitat y la UAB lo fundaron en el año 1995 con la finalidad de hacer una investigación de excelencia mediante la generación de conocimiento de calidad y la transferencia de tecnología hacia la sociedad, ofreciendo también valor añadido a las empresas. Desde su creación, el CVC integra el Servicio de Tratamiento de Imágenes, servicio científico-técnico de la UAB.

Dado que se trata de un centro público de investigación y en consonancia con la comisión europea, la intención del CVC es la de seguir con la idea de "Open Science". Representa un enfoque del proceso científico basado en el trabajo cooperativo y nuevas formas de distribución del conocimiento mediante tecnologías digitales. Siguiendo en la línea anterior, desde el CVC se quieren activar procesos por los que se transfieran conocimientos científicos de esta institución a otras organizaciones, realizando así lo que se conoce como transferencia tecnológica.

Es por esto por lo que nace la necesidad de crear un servicio que sea accesible por cualquier persona u organización, y que estos puedan ver las posibilidades que ofrecen los sistemas y algoritmos desarrollados en el CVC. El sistema propuesto en este proyecto ofrece servicios de visión por computador a través de Twitter, de manera que con la interacción con la red social podamos solicitar un determinado servicio, simplemente enviando un tweet a una cuenta asociada al CVC.

[En la **Figura 1** se muestra una petición para un determinado servicio.]

El siguiente documento está organizado en las siguientes secciones:

- Objetivos
- Estado del arte
- Metodología seguida durante el desarrollo.
- Arquitectura
- Resultados
- Conclusiones
- Líneas de trabajo futuro.
- Agradecimientos
- Bibliografía

Figura 1 Ejemplo de petición de un servicio.

2 OBJECTIVOS

2.1 Objetivos generales

El objetivo del proyecto es el diseño y desarrollo de un sistema que sea capaz, a través de una tecnología concreta del CVC, de procesar imágenes a través de Twitter. Este sistema debe ser lo más intuitivo posible ya que debe ser capaz de abstraer al consumidor del servicio del funcionamiento del mismo, proporcionándole únicamente los resultados solicitados. Si bien en un futuro se añadirán nuevos servicios al sistema, en esta primera versión, se podrá utilizar el servicio de detección de texto en imágenes. El sistema ofrecerá mecanismos para poder añadir y eliminar nuevos servicios. De esta manera nos aseguramos de que el catálogo de servicios ofrecidos puede crecer.

2.2 Objetivos específicos

Los requisitos que debe cumplir el proyecto para considerarlo como satisfactorio son los siguientes.

- El sistema debe ser multiplataforma (Linux, Windows, Mac)

-Para cumplir este punto se ha decidido desarrollar todos los módulos del sistema en Java, utilizando JDK 1.8.0

- Las peticiones de servicio deben ser ágiles y simples. -Para ello la plataforma donde interactúan los usuarios deberá ser exclusivamente Twitter.

- Posibilidad de configurar servicios.
    - Creación
    - Actualización
    - Eliminación

- Preparar el sistema para la recepción de videos
-Los formatos serán los admitidos por Twitter (MP4 y MOV).

- Preparar el sistema para la recepción de fotos.
-Los formatos serán los admitidos por Twitter (JPG y PNG)

- Utilizar REST como protocolo de intercambio y manipulación de datos en los servicios del sistema.
-Los servicios deberán implementarse como web API's .

- Que los diferentes módulos del sistema sean integrables en un servidor del centro de visión por computador.

- Extensiblidad de la aplicación
-Facilidad para añadir nuevas funcionalidades al sistema.

- Performance
-El sistema debe responder prácticamente a tiempo real, salvo casos justificados (coste computacional alto de un determinado servicio)

3 ESTADO DEL ARTE

Al tener una API abierta, existen infinidad de aplicaciones que interactúan con Twitter. Para el tema que nos concierne que es el tratamiento de imágenes existen algunas aproximaciones de proyectos que aplican algoritmos OCR como, por ejemplo: **Reverse OCR [1]**, **Figura 2**. Esta

Figura 2 Reverse OCR.

cuenta bot dibuja líneas aleatorias hasta que el software de reconocimiento óptico de caracteres piense que parece una cierta palabra. También podemos encontrar bots que utilizan algoritmos de reconocimiento para cuestiones como saber cuánta gente cuelga fotos cuando tiene poca batería en el móvil. **Lowbatterymuch[2]**. Actualmente no existe ningún BOT que sea capaz de utilizar múltiples algoritmos de reconocimiento de imagen a partir de una misma cuenta. Si bien encontramos proyectos de muy diversa índole, muy pocos proyectos utilizan Twitter como una plataforma para solicitar un determinado servicio de visión por computador. Un bot que por la mecánica podría asemejarse al sistema diseñado en este proyecto podría ser **Quilt Bot [3],** cuenta que a partir de una imagen es capaz de hacer un tratamiento específico de esta. Hay algunos ejemplos de bots que prestan servicios como pueden ser **DearAssitant [4]** , un bot de Twitter que intentará responder a sus preguntas al igual que Siri, Google Now o Cortana.

4 METODOLOGÍA Y DESARROLLO

4.1 Introducción

Para el desarrollo del proyecto se ha seguido una metodología ágil, basada en sprints. Los sprints, con una duración de dos semanas cada uno, nos proporcionan una versión funcional en todo momento, a la que de forma iterativa le vamos añadiendo nuevas características. Para la realización de esta metodología se ha utilizado **GIT [5]** para el control de versiones y **Trello [6]** como tablero de tareas.

Para la obtención de los resultados esperados se ha realizado previamente una investigación de diferentes tecnologías y arquitecturas para desarrollar el proyecto. El sistema está diseñado bajo el patrón de arquitectura **Broker [7].** Para realizar este diseño han sido fundamentales los conocimientos adquiridos en la asignatura Arquitectura i tecnologías del software. El sistema cuenta con varias integraciones de tecnologías de diversa índole tales como **MongoDB [8],** las **API's de Twitter** [9] o las API's externas de tratamiento de imagen. Para la correcta integración entre todos los módulos del sistema se ha utilizado el protocolo de transferencia de hipertexto (HTTP), para la comunicación y transferencia de información.

4.2 Twitter Apps

El primer concepto desarrollado ha sido la creación de una cuenta BOT en la red social Twitter. Para realizar esta automatización necesitaremos crear aplicaciones asociadas a dicha cuenta, que nos permitan interactuar con la información que esta maneja. Se han creado 2 aplicaciones:

- **CVC_readThis:** Aplicación encargada de monitorizar la actividad de cvcBot17. Es capaz de capturar los cambios de estado de la cuenta asociada, permitiendo obtener en streaming (a tiempo real), mensajes recibidos, así como sus hastags o archivos adjuntos que contenga el tweet. La aplicación también es capaz de monitorizar la actividad de la cuenta origen hacia otras cuentas, pero esta información no nos interesa ya que la aplicación siempre esperara a una petición de servicio. No tenemos control sobre la información que nos proporciona dicha aplicación. Filtrar esta información a tiempo real será la tarea del Bróker.

- **CVC_postingResponse**: Aplicación encargada de escribir los tweets de respuesta sobre las cuentas que hayan solicitado un determinado servicio. Esta aplicación devolverá toda aquella información que el Bróker le envíe.

La decisión de trabajar sobre 2 aplicaciones en vez de con 1 es por el hecho de conseguir un mayor desacoplamiento del sistema. Una aplicación siempre debe estar monitorizando la actividad de la cuenta, por lo que utilizar la misma aplicación para enviar un determinado mensaje al cliente que ha solicitado el servicio puede provocar algunos errores en la aplicación y hace que el manejo de información por parte del Bróker sea más complejo.

Estas aplicaciones representan un punto de acceso a la API de Twitter desde aplicaciones externas. Para que el bróker pueda acceder a las funcionalidades de las aplicaciones de Twitter necesitara los tokens de acceso proporcionados por cada aplicación en el panel de configuración de cada una.

Para poder interactuar con la API de Twitter a través de las aplicaciones creadas anteriormente el sistema se apoyará en Twitter4j. Twitter4J es una biblioteca no oficial de Java para la API de Twitter. Con Twitter4J, podemos integrar fácilmente la aplicación Bróker Java con el servicio de Twitter. Nos abstrae de implementar las llamadas y se complementa perfectamente con la aplicación broker, al tratarse de una librería 100% para Java. Para cuestiones de seguridad utiliza el estándar abierto **OAuth [10]**. Es compatible con la versión 1.1 de la API de Twitter.

4.3 Bróker

El mediador o "Bróker" es responsable de coordinar la comunicación entre las diferentes partes del sistema. Se encarga de recibir las peticiones del cliente, hacer la solicitud a un servicio en concreto y devolver la respuesta al cliente. Ejerce de punto central del sistema. Siempre está escuchando peticiones, por lo que siempre se estará ejecutando. Una de las principales funciones del bróker, es la de proporcionar un determinado servicio a un cliente que lo solicite. El bróker no procesara en ningún momento la imagen enviada a través de Twitter. Este intermediario servirá para coordinar los diferentes módulos del sistema.
Uno de los elementos diferenciadores del bróker con el resto de elementos es que siempre se está ejecutando. Se trata de una aplicación Java, fácilmente ejecutable desde la

consola de comandos que se mantiene a la escucha de cualquier petición de servicio que se realice desde Twitter hacia la cuenta. La aplicación está construida bajo el concepto del patrón de diseño **Observer [11].** Utilizamos la idea de este patrón de diseño debido a que necesitamos que la aplicación llame a un determinado servicio cuando observe un cambio de estado en la cuenta de Twitter.
El patrón observer nos permite mantener desacopladas las aplicaciones de Twitter, que son las que emiten los eventos, de la aplicación broker. En la Figura 3 podemos ver como la cuenta @cvcBoT2017 recibe el tweet "Tweet para la monitorización de la cuenta" y en ese momento la aplicación broker recibe el tweet.

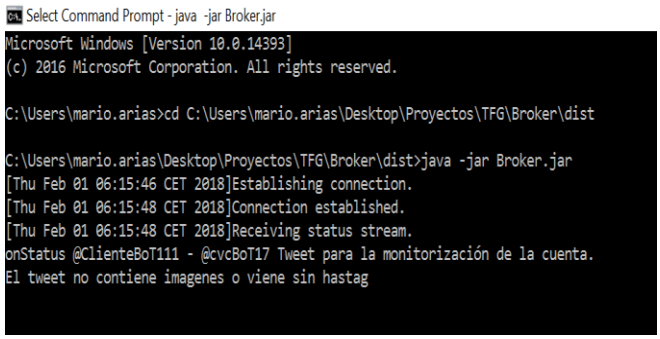

Figura 3 Salida de la aplicación Bróker.

serviceName: '{{serviceName()}}',
uri: '{{uri()}}',
status: '{{status()}}',
description: '{{description()}}',
input_parameters: '{{parameters()}}',
output: '{{output()}}' }
]

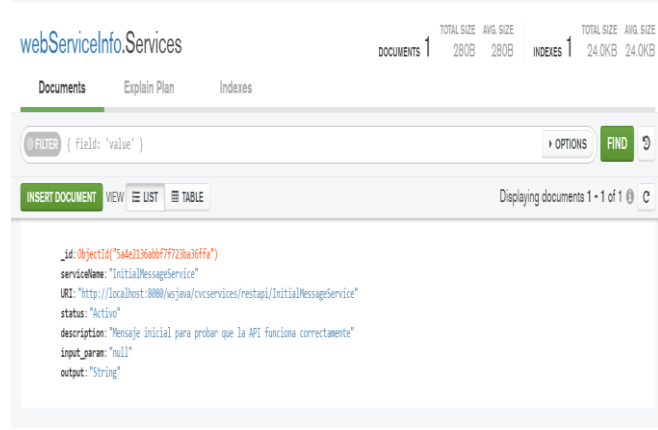

Figura 4 Documento con la información de un servicio en la base de datos MongoDB.

4.4 Servicios

Para el correcto funcionamiento del sistema toda la información de los servicios disponibles se almacenara en una base de datos no-relacional a la cual tendrá acceso el Bróker. Dado que solamente necesitaremos una tabla que contenga información, y esta no necesita relacionarse con ninguna otra, se ha tomado la decisión de utilizar una base de datos no relacional MongoDB. Este tipo de tecnología si bien no estructura los datos de la misma forma que las relacionales, el acceso es mucho más rápido y dado que el sistema debe responder de manera casi inmediata, se ha optado por este tipo de almacenamiento.

El controlador oficial MongoDB Java proporciona interacción síncrona y asíncrona con MongoDB. A través de esta librería y sus funciones interactuaremos con la base de datos MongoDB. Esta integración hay que hacerla para que el bróker, antes de pedir un servicio se asegure si existe o no

4.4.1 Información de servicios

Para almacenar la información en la base de datos los documentos insertados en la base de datos MongoDB deberán tener el siguiente formato:

[
{ **_id**: '{{ServiceID()}}',

MongoDB ha sido creado para brindar escalabilidad, rendimiento y gran disponibilidad, escalando de una implantación de servidor único a grandes arquitecturas complejas de centros multidatos. MongoDB brinda un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria. Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan, por lo que, en este proyecto, cumple con todos los requisitos necesarios para que se pueda extender la aplicación en un futuro.

```
private static void conectToMongoDB() throws UnknownHostException {

    Mongo mongo = new Mongo("localhost", 27017);
    db = mongo.getDB("webServiceInfo");
    tablaServicios = db.getCollection("Services");

}

public static String findDocumentByServiceName(String serviceName) {

    BasicDBObject query = new BasicDBObject();
    query.put("serviceName", serviceName);
    DBObject dbObj = tablaServicios.findOne(query);
    return dbObj.get("serviceName").toString();
}
```

Figura 5 Funciones Java para interactuar con MongoDB.

En la Figura 5 podemos ver las dos funciones que utilizaremos para trabajar con la base de datos MongoDB. Dado que los usuarios solicitaran el servicio a través de un **hastag [11],** buscaremos los servicios por nombre con la función **findDocumentByServiceName(serviceName).**

4.4.2 Configuración de servicios

Para añadir y eliminar nuevos servicios se utilizará una aplicación Java con interfaz gráfica. Esta aplicación tiene apariencia de formulario. Permitirá añadir un servicio añadiendo los campos especificados, ver los servicios que tenemos actualmente y eliminar un determinado servicio. Dado que la base de datos se configuró en la versión Alpha, seguiremos la misma estructura de datos que indica el apartado 4.4.1 para la creación del formulario. Las ids de los servicios de asignaran de manera dinámica, es decir, el configurador de servicios deberá introducir nombre, URI, estado del servicio, breve descripción, parámetros de entrada y salida del servicio. Para eliminar tendrá que indicarlo a través de la id.

Esta aplicación está conectada a la base de datos MongoDB y permite eliminar, insertar y consultar documentos de la misma.

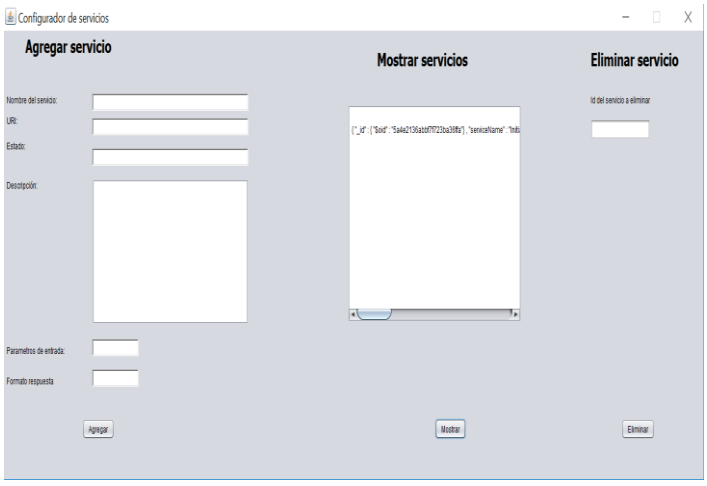

Figura 6 Aplicación de configuración de servicios

4.5 API REST

4.6 Servicio ReadThis

5 RESULTADOS

6 CONCLUSIONES

7 ARQUITECTURA

8 LÍNEAS FUTURAS

9 AGRADECIMIENTOS

10 BIBLIOGRAFÍA

APENDICE

A1. SECCIÓN APÉNDICE

....

A2. SECCIÓN APÉNDICE

....