#ReadThis - Un servicio que reconoce texto en imágenes a través de Twitter

Mario Arias Escalona

Resumen: En el presente documento se expone de manera detallada todo lo referente al diseño, implementación y finalización de la primera versión del sistema ReadThis. Por una parte, tenemos todo lo relacionado con la plataforma Twitter. Esta plataforma es la que nos permitirá la interacción con el cliente. En segundo lugar, tendremos la estructura bróker, intermediario que actúa como nodo central del sistema, por donde pasaran todas las peticiones de servicio, así como los resultados obtenidos por dichas peticiones. En una tercera parte nos encontraríamos con los servicios webs encargados del procesamiento de imágenes.

Paraules clau— Twitter, OCR, Broker, OAuth, MongoDB, Observer, API, REST, AWS, OpenCV, ReadThis

Abstract: In this document, everything related to the design, implementation and finalization of the first version of the ReadThis system is explained in detail. On the one hand, I have everything related to the Twitter platform. This platform is what allows us to interact with the client. Secondly, we will have the broker, intermediary structure that acts as the central node of the system, through which all service requests will pass, as well as the results obtained by these causes. In a third part we would find the web services in charge of image processing.

Index Terms— Twitter, OCR, Broker, OAuth, MongoDB, Observer, API, REST, AWS, OpenCV, ReadThis

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

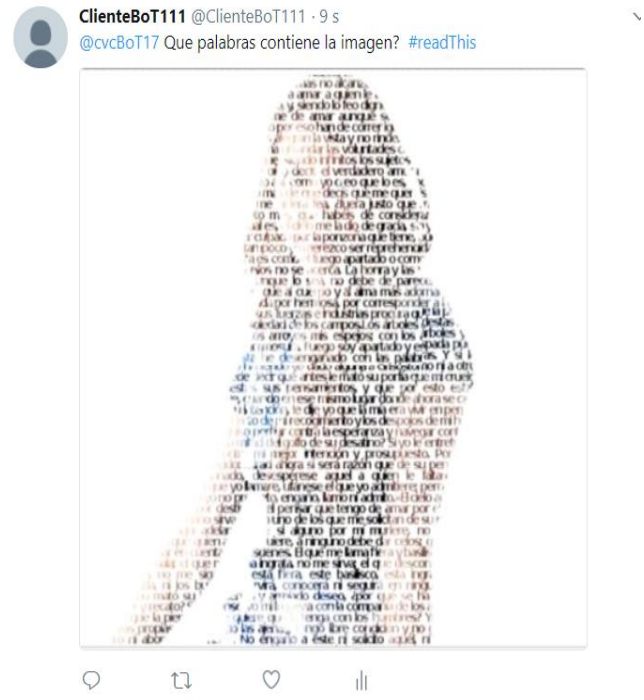

Figura 1 Ejemplo de petición de un servicio.

2 OBJETIVOS

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.
- Extensibilidad 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**.

Figura 2 Reverse OCR.

Esta 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.

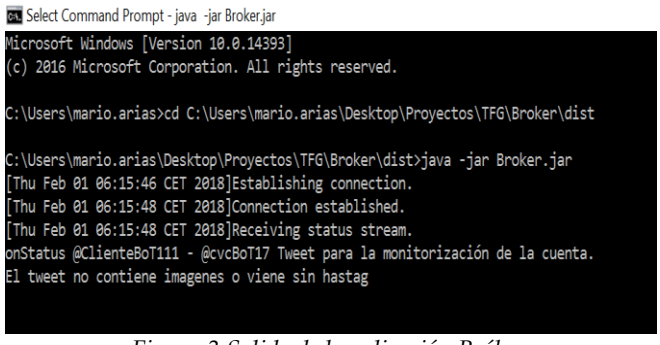

Figura 3 Salida de la aplicación Bróker.

4.4 Servicios

Para el correcto funcionamiento del sistema toda la información de los servicios disponibles se almacenará 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()}}',
        serviceName: '{{serviceName()}}',
        uri: '{{uri()}}',
        status: '{{status()}}',
        description: '{{description()}}',
        input_parameters: '{{parameters()}}',
        output: '{{output()}}' }
]
```

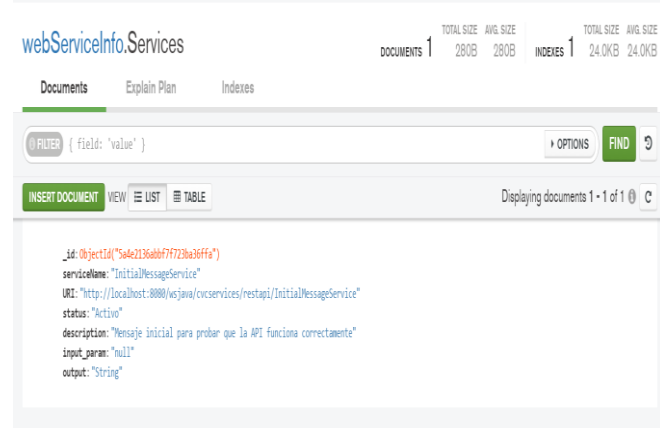

Figura 4 Documento con la información de un servicio en la base de datos MongoDB.

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. Seguiremos la misma estructura de datos que indica el apartado 4.4.1 para la creación del formulario. Las ids de los servicios se asignarán 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.

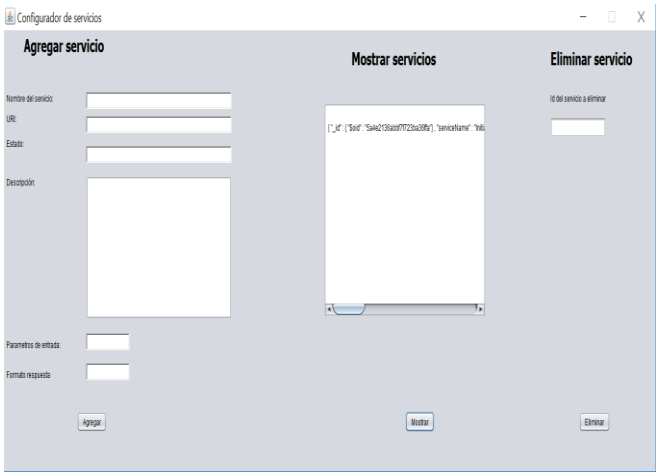

Figura 6 Aplicación de configuración de servicios

4.5 API REST

Los servicios de visión por computador deberán poder ser consumidos a través de diferentes API REST. Para que el servicio pueda estar incluido en el sistema deberá seguir las directrices del documento ***ServiciosyConfiguracion.pdf***.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Es una alternativa a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. En nuestro caso nos interesa una solución más sencilla de manipulación de datos como REST.

Cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar). Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación con nuestra base de datos MongoDB.

Utilizar API REST presenta unas ventajas que se adaptan a las características del sistema:

- Mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- La separación entre cliente y servidor tiene una ventaja y es que cualquier equipo de desarrollo puede escalar el producto. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta.
- La API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

En esta versión se ha creado una API en Java que únicamente implementa el método GET. Debido a las particularidades del algoritmo explicadas en el apartado 4.6, solo se puede ejecutar bajo un sistema operativo Linux, por lo que el ejecutable estará alojado en la misma máquina que la aplicación bróker del CVC.

Por cuestiones de seguridad y escalabilidad la API funciona sobre **Amazon Elastic Beanstalk.** Puede implementar y administrar aplicaciones rápidamente en la nube de AWS abstrayéndonos de conocer la infraestructura en la que se ejecutan. AWS Elastic Beanstalk reduce la complejidad de la administración. Cargando la aplicación en la plataforma, esta gestionará de manera automática los detalles de aprovisionamiento de capacidad, equilibrio de carga, escalado y monitorización del estado de la aplicación. Podremos controlar la cantidad de solicitudes que recibimos. Otro aspecto importante a tener en cuenta es que la capa de seguridad, así como su configuración ya están implementadas.

Esta API se conectará vía **SSH** al servidor donde está alojado el algoritmo. En un primer lugar transfiere la imagen que recibe del bróker al servidor. Una vez se ha transferido el archivo, se ejecutara un **script bash** , que se encargara de ejecutar el algoritmo con la imagen guardada anteriormente. La salida del algoritmo será guardada en un fichero

txt. La información de dicho fichero será transferida mediante un canal SFTP hacia la API, para que esta recoja el resultado y se lo envíe de vuelta al Bróker. El script se encargará de borrar la imagen recibida una vez haya enviado la información solicitada por la API.

```
public class SSHReadFile
    {
    public static void main(String args[])
    {
    String user = "xxxxxxx";
    String password = "xxxxxxxx";
    String host = "158.109.8.90";
    int port=22345;

    String remoteFile="/home/CVC_Server/textRecognitionResult.txt";

    try
        {
        JSch jsch = new JSch();
        Session session = jsch.getSession(user, host, port);
            session.setPassword(password);
            session.setConfig("StrictHostKeyChecking", "no");
        System.out.println("Establishing Connection...");
        session.connect();
            System.out.println("Connection established.");
        System.out.println("Crating SFTP Channel.");
        ChannelSftp sftpChannel = (ChannelSftp) session.openChannel("sftp");
        sftpChannel.connect();
        System.out.println("SFTP Channel created.");


        InputStream out= null;
        out= sftpChannel.get(remoteFile);
        BufferedReader br = new BufferedReader(new InputStreamReader(out));
        String line;
        List<String> results = new ArrayList<String>();
        while ((line = br.readLine()) != null)
            results.add(line);
        br.close();
        }
    catch(Exception e){System.err.print(e);}
    }
    }
```

Figura 7 Conexión SFTP y SH con Java

4.6 Servicio ReadThis

El servicio de visión por computador que se implementa en esta versión es el servicio que le da nombre al proyecto y no es otro que ReadThis. Este servicio está basado en una implementación hecha por LLuis Gomez i Bigorda el 3 de mayo del 2015. Podemos encontrar el algoritmo en el siguiente repositorio:

https://github.com/ComputerVisionCentre/RRC2015_Baseline_CV3Tess/blob/master/main.cpp

Para poder compilar el código y conseguir un ejecutable hay que seguir una serie de pasos. Si alguno de estos se configura mal o deriva en algún error, el algoritmo no funcionara. El código ha sido compilado en un sistema Operativo Ubuntu 16 ya que, pese a que todo el proyecto ha sido desarrollado en Windows 10, los componentes necesarios para la compilación daban muchos errores, por lo que se ha optado por trabajar con el algoritmo desde Linux.

Lo primero debe ser instalar correctamente Tesseract OCR. Se trata de un motor de OCR con soporte para Unicode y la capacidad de reconocer más de 100 idiomas de forma inmediata. Puede ser entrenado para reconocer otros idiomas.

En segundo lugar, deberemos instalar Leptonica. Se trata de una serie de herramientas, de código abierto que son muy útiles para el procesamiento de imágenes y aplicaciones de análisis de imágenes.

En tercer lugar, deberemos descargar la librería OpenCV. OpenCV es una biblioteca libre de visión por computador que fue diseñada para la eficiencia computacional y con un fuerte enfoque en aplicaciones en tiempo real.

Una vez tengamos todos estos componentes, será el momento de construir e instalar OpenCV. Para realizar dicha tarea utilizaremos la herramienta CMake. Debemos tener en cuenta que el algoritmo utiliza los módulos extra de la librería OpenCV, llamados **opencv_contrib** por lo que también deberemos descargarnos dichos módulos e incluirlos en el ensamblado. Es muy importante que para el empaquetado de la librería utilicemos correctamente los parámetros de entrada en CMake:

```
cmake -D BUILD_TIFF=ON -D WITH_CUDA=OFF -D ENABLE_AVX=OFF -D WITH_OPENGL=OFF -D WITH_OPENCL=OFF -D WITH_IPP=OFF -D WITH_TBB=ON -D BUILD_TBB=ON -D WITH_EIGEN=OFF -D WITH_V4L=OFF -D WITH_VTK=OFF -D BUILD_TESTS=OFF -D BUILD_PERF_TESTS=OFF -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D OPENCV_EXTRA_MODULES_PATH=/opt/opencv_contrib/modules /opt/opencv/
```

Una vez la librería ha sido instalada, deberemos compilar el algoritmo de la siguiente manera:

```
g++ main.cpp -o output `pkg-config --cflags --libs opencv`
```

En este momento, el algoritmo estará listo para ser ejecutado. Para ver que funciona correctamente se ha utilizado la imagen de la **Figura 8.**

Figura 8 Imagen para reconocimiento de texto

En la **Figura 9** podemos apreciar en el noveno parámetro de cada fila como el algoritmo ha sido capaz de reconocer el texto de la imagen.

```
270,130,619,130,619,214,270,214,GASOLINERA
646,153,819,153,819,216,646,216,SerVIcIo
836,128,939,128,939,217,836,217,24h
```

Figura 9 Salida del algoritmo ReadThis

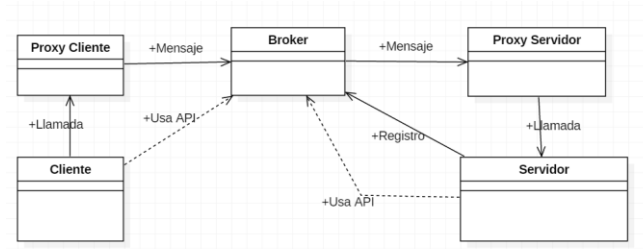

Figura 10 Diagrama patrón de diseño bróker

5 RESULTADOS

5.1 Twitter como plataforma de servicios

Gracias al sistema desarrollado en este proyecto, a partir de una plataforma como Twitter que comúnmente se utiliza para la interacción social, se ha conseguido crear un portal en el cual poder solicitar complejos servicios de visión por computador de manera totalmente accesible para cualquier usuario de esta conocidísima red social.

5.2 Bot de Twitter

La aplicación bróker actúa de nodo central del sistema, permitiéndonos integrar diferentes tecnologías para que todas puedan trabajar de manera independiente, siempre teniendo en común el mismo punto de unión. Esta versión deja entrever la potencia que puede llegar a tener el sistema. La automatización del bróker comportándose como un BOT de Twitter permite crear un sistema dinámico que es capaz de prestar complejos servicios a través de simples peticiones.

5.3 Cloud API

Los servicios son alojados en cloud APIs permitiendo al software solicitar los datos y los cálculos de uno o más servicios a través de una interfaz directa. La Cloud API más exponen sus características a través de REST.

6 ARQUITECTURA

El estilo arquitectónico seleccionado para la construcción de la solución es el patrón **Bróker**.

El patrón de diseño Bróker se utiliza para organizar sistemas distribuidos con componentes desacoplados que interaccionan realizando invocaciones remotas a servicios. El mediador o "Bróker" es responsable de coordinar la comunicación, tanto de enviar las peticiones, como de transmitir los resultados y las excepciones. En la **Figura 10**, se muestra de manera simplificada los diferentes elementos de un sistema bróker estándar.

El patrón bróker es útil para sistemas que evolucionan y crecen con el tiempo: el "bróker" se encarga de que podamos acceder a los distintos servicios del sistema sin necesidad de conocer su localización. En el caso que nos atañe, el sistema puede ir ampliándose ofreciendo cada vez más servicios, por lo que esta implementación nos permitiría que el sistema pudiera crecer de manera fácil y compacta.

Frente a una arquitectura clásica cliente-servidor el sistema nos ofrece más flexibilidad, mantenibilidad y adaptabilidad. Si alguno de los servicios fallase los demás podrían seguir funcionado de manera independiente.

El bróker se encargaría de abstraer al cliente donde están los servidores apropiados para un servicio en concreto. Esto facilitaría mucho la implementación del lado del cliente. Por la parte del servidor únicamente se debe encargar de realizar una determinada tarea y proporcionar el servicio solicitado por el bróker. Toda la lógica de comunicación queda centralizada en un mismo punto desacoplando en gran medida todos los componentes del sistema.

Aplicado el diseño a los diferentes módulos del sistema, a nivel general cada módulo quedaría descrito de la siguiente manera:

- **Cliente:** Aplicación de Twitter que accede a los servicios de al menos un servidor. Para invocar servicios remotos, el cliente enviará solicitudes al bróker. Después de que la operación se haya ejecutado, el cliente recibirá la respuesta o excepción del bróker. No necesita conocer la ubicación de los servidores a los cuales solicita un determinado servicio. Esto permite la agregación de nuevos servicios existentes a otras ubicaciones, aun cuando el sistema está siendo ejecutado.

- **Servidor:** Implementa objetos que exponen su funcionalidad a través de una interfaz que contiene operaciones y atributos. Es el lugar donde están alojados los servicios de tratamiento de imagen del sistema.

- **Bróker:** Es un mensajero, responsable de la

transmisión de solicitudes de clientes y servidores, así como de la transmisión de respuestas. Una vez iniciado el bróker, este se quedará a la escucha de cualquier solicitud que el cliente haga. El bróker siempre estará en ejecución.

- **Proxy-Cliente:** Representa una capa adicional entre los clientes y el bróker, para proveer de transparencia en el sentido que un objeto remoto aparece como local ante el cliente, es decir, esconde los detalles de implementación.

- **Proxy-Servidor:** Es el responsable de recibir solicitudes, desempaquetar los mensajes de entrada, llamar al servicio apropiado y de hacer el marshaling de resultados (transformar la representación en memoria de un objeto a un formato apropiado para almacenaje o transmisión).

7 Conclusiones

La arquitectura del software permite que el sistema sea escalable en un futuro. La aplicación Bróker brinda al sistema de una versatilidad y robustez que hacen que todo el contexto de la aplicación pase por un único nodo, siendo más fácil la detección de errores en alguno de los módulos.
Si alguno de los módulos falla, el sistema sabe responder gracias a su gran desacoplamiento. El bróker ordena y coordina los mensajes para un desarrollo de aplicaciones simplificado. La integración de bases de datos mejora el rendimiento de la aplicación y simplifica la administración.

Como ventaja a futuro, el diseño del sistema es aplicable a cualquier tipo de servicio y no únicamente a servicios de visión por computador. La estructura es fácilmente exportable a aplicaciones empresariales.

9 Agradecimientos

En primer lugar, agradecer a Dimosthenis Karatzas por ponerse en contacto conmigo para realizar este proyecto, así como agradecer su flexibilidad con mi horario laboral, ha sido de gran ayuda para poder realizar con éxito este proyecto de final de grado. Por último, agradecer también a mi pareja Carmen por apoyarme durante estos años de carrera, por aguantarme y por ayudarme, así como agradecer a mi familia todo el esfuerzo realizado y a los compañeros que me han acompañado a lo largo de la universidad.

10 Bibliografía

[1] MongoDB [En línia]. Blog MongoDB [Consultado: 15 de Noviembre de 2017] Disponible en Internet: https://www.mongodb.com/blog

[2] Broker Pattern [En línia]. openloop [Consultado: 17 de Noviembre de 2017]. Disponible en Internet:http://www.openloop.com/softwareEngineering/patterns/architecturePattern/arch_ Broker.htm

[3] MongoDB in Java[En línia]. MongoDB Java Driver [Consultado: 17 de Noviembre de 2017]. Disponible en Internet: http://mongodb.github.io/mongo-java-driver/3.6/

[4] Create API [En línia]. The little Manual of API Design [Consultado: 20 de Noviembre de 2017]. Disponible en Internet: http://www4.in.tum.de/~blanchet/api-design.pdf

[5] Twitter apps [En línia]. Twitter developer [Consultado: 25 de Noviembre de 2017]. Disponible en Internet: https://developer.twitter.com/en/docs

[6] Configurations GlassFish [En línia]. Oracle GlassFishServer [Consultado: 25 de Noviembre de 2017]. Disponible en Internet: https://docs.oracle.com/cd/E18930_01/html/821- 2426/abdhg.html

[7] OpenCV tutorial [En línia]. OpenCV 3.0.0 tutorial [Consultado: 3 de Diciembre de 2017]. Disponible en Internet: https://docs.opencv.org/3.0-rc1/d9/df8/tutorial_root.html

[8] Cpp to exe [En línia]. cprogramming [Consultado: 10 de Diciembre de 2017]. Disponible en Internet: https://cboard.cprogramming.com/cplusplus-programming/61887-changingprogram-cpp-exe.html

APENDICE

A1. SECCIÓN APÉNDICE

....
....
....
..... .

A2. SECCIÓN APÉNDICE

....
....
....
..... .