Universitat Autònoma de Barcelona

#ReadThis- Un servicio que lee el texto en les imágenes enviadas a través de Twitter- Informe de progreso I I

Mario Arias Escalona

Índice

DESARROLLO

1.Visión global

El sistema consta de 3 partes muy diferenciadas en cuanto a funcionamiento y desarrollo se refiere. A continuación, se muestra un esquema conceptual a nivel funcional de las diferentes partes de las que está compuesta la estructura del proyecto:

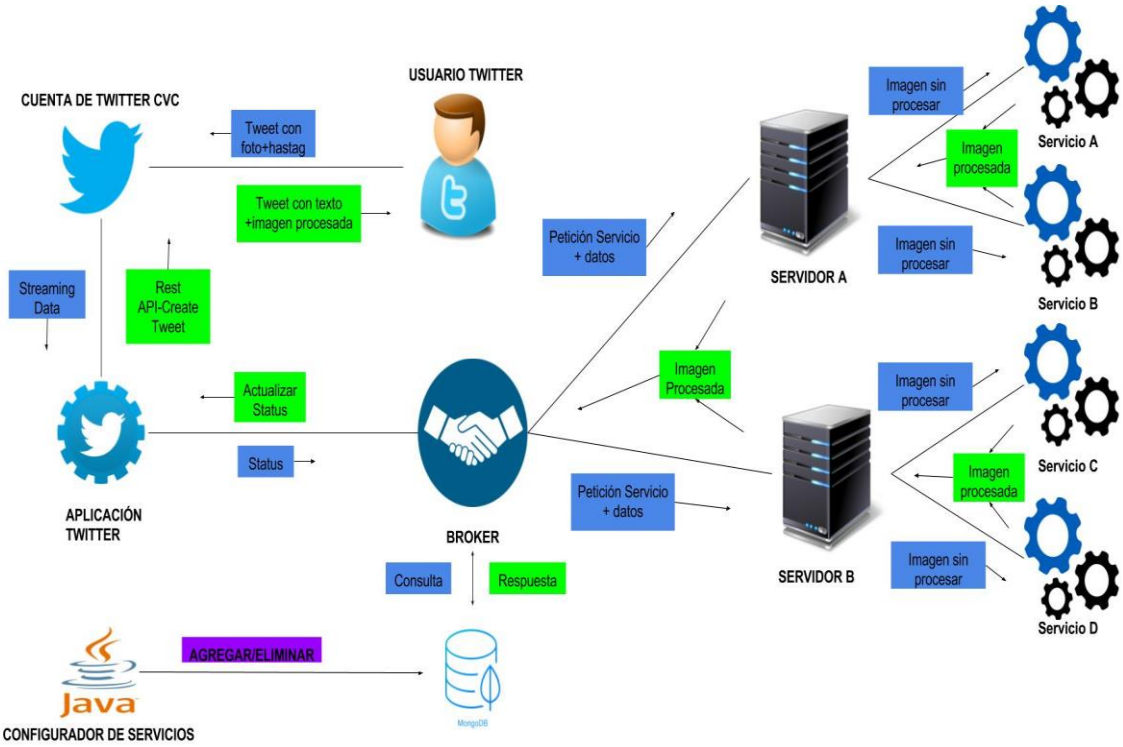

Figura 1: Diagrama funcional del sistema

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.

A nivel de desarrollo el programa se encuentra entre la versión Alpha y la versión Beta. En la versión Alpha, actualmente estable, se consiguió ejecutar el sistema haciendo que todos los módulos de este entran en juego consiguiendo así asegurar la interconectividad y la integración de las diferentes tecnologías en un mismo flujo de programa.

2.Versión Alfa

Esta es la primera versión del programa. Si bien no contiene todos los elementos, nos permite probar por primera vez un flujo completo del sistema.

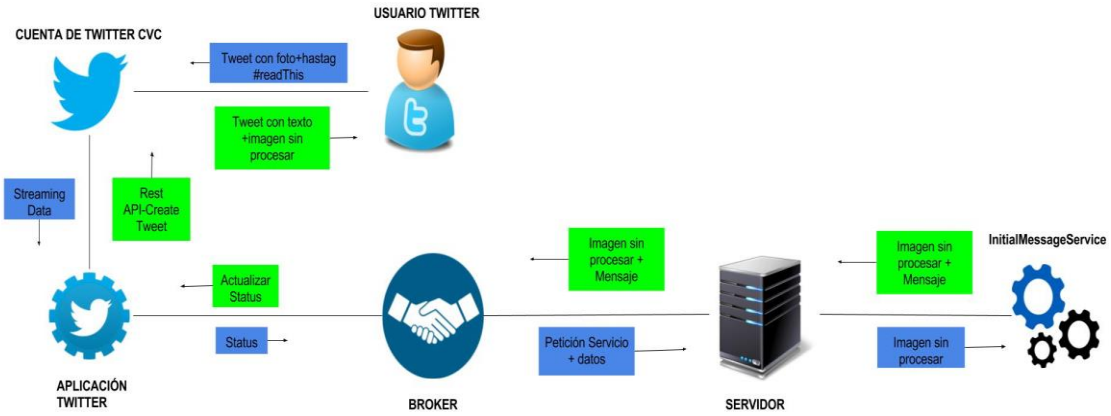

Figura 2: Diagrama funcional del sistema versión alfa

2.1 Twitter Apps

Esta primera versión se centra en la comunicación y automatización de los diferentes módulos del sistema. 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:

Figura 3: Aplicaciones de la cuenta cvcBot17

Como podemos apreciar en la *Figura 3* se han creado 2 aplicaciones asociadas a la cuenta cvcBot17. Cada una de ellas tiene una función específica:

- **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 compleja.

2.1.1 Configuración

Muchas redes sociales y Twitter no es una excepción, dotan a los programadores de una interfaz para que estos puedan trabajar sobre la plataforma. Estas interfaces se llaman API's (abreviatura de Application Programming Interface). Para interactuar con la API de Twitter la manera es a través de estas aplicaciones que se pueden crear desde el portal de desarrolladores de Twitter.

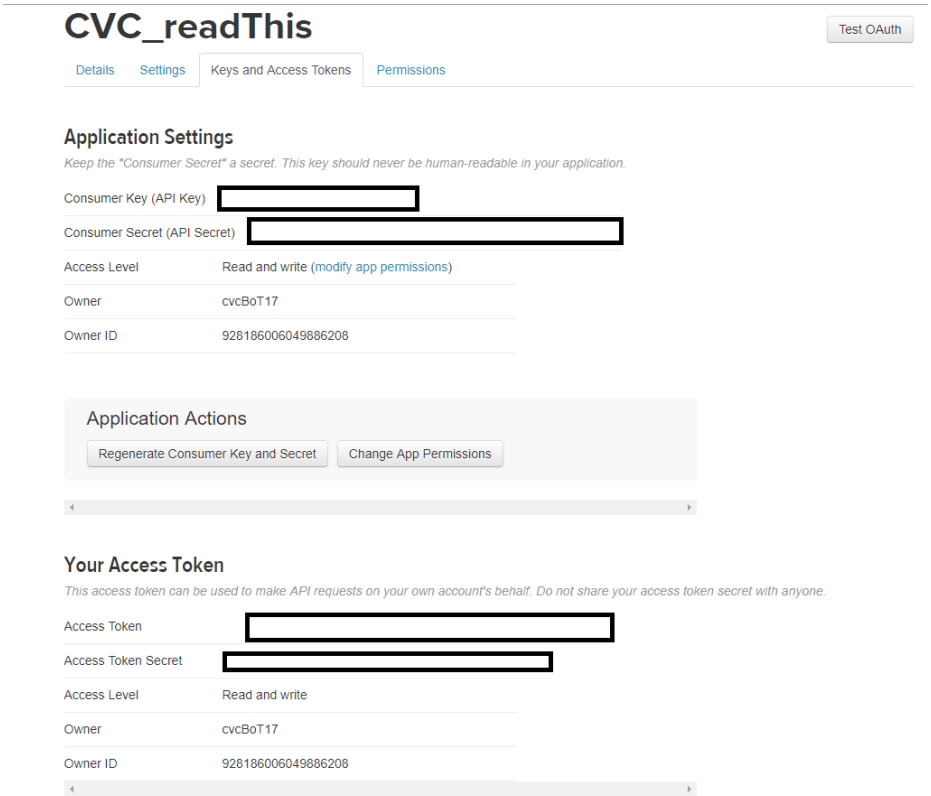

Figura 4: Panel de configuración aplicación de Twitter

En la Figura 4 podemos ver como tenemos varias pestañas que nos permiten configurar la aplicación de la manera que mejor nos venga. También podemos ver que existen unos tokens de acceso (tapados por temas de seguridad) que son los que nos permitirán el acceso a esta aplicación. Estas claves las debe conocer el Bróker para poder acceder a toda la información

que las aplicaciones le puedan proporcionar. También vemos que tipo de permisos tiene la aplicación y que estos pueden ser modificados.

2.1.2 Interacción con las aplicaciones

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. Podemos resumir las características de Twitter4J como:

- 100% para Java - funciona en cualquier plataforma Java versión 5 o posterior
- Plataforma Android y Google App Engine listo
- Cero dependencias: no se requieren JARS adicionales
- Soporte incorporado de OAuth
- Soporte de gzip listo para usar
- Compatible 100% con Twitter API 1.1

Fuente: http://twitter4j.org/en/index.html

2.2 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.

2.2.1 Petición de servicio

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. Para este caso tenemos 3 posibles escenarios:

Figura 5: Petición de servicio versión Alpha

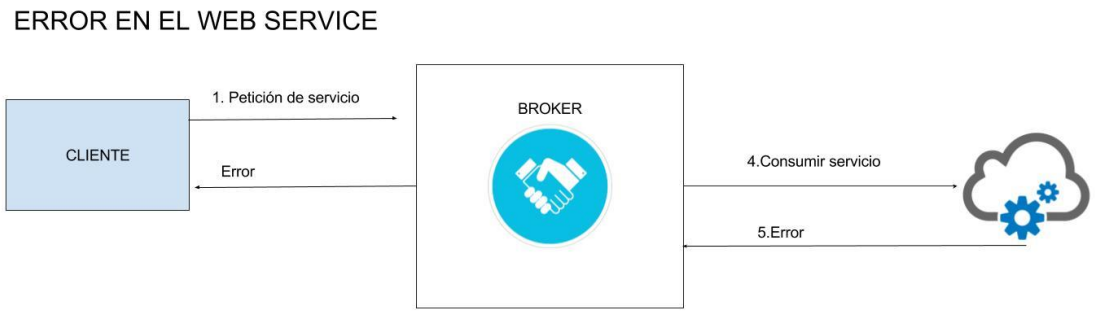

Figura 6: Error en el web service versión Alpha

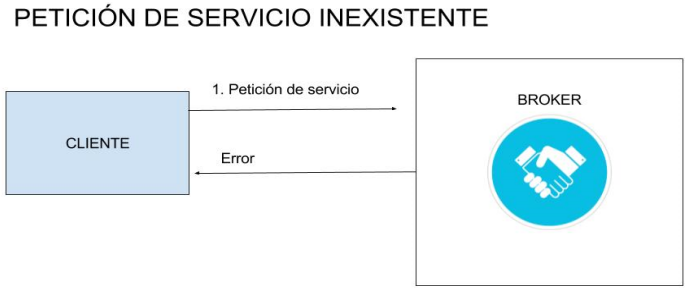

Figura 7: Petición de servicio inexistente versión Alpha

En el caso de la Figura 5, el cliente solicita correctamente el servicio. El bróker lo solicita al servicio web correspondiente y devolverá el resultado en forma de Tweet.

En la Figura 6, el servicio web tiene algún error o no está disponible para ser utilizado. En este caso se informará al cliente de que el servicio solicitado no está accesible.

Finalmente, en el caso de la Figura 7 si la petición del servicio no corresponde con los servicios disponibles o la petición carece de algún elemento importante como por ejemplo el envió de una imagen, el bróker informara a través de un Tweet con el error correspondiente en cada caso.

2.2.2 Escucha activa

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. A continuación, se muestra un ejemplo de la escucha activa del elemento broker.

1. Desde la cuenta **ClienteBoT1111** escribimos un tweet hacia la cuenta bot **cvcBot17**.

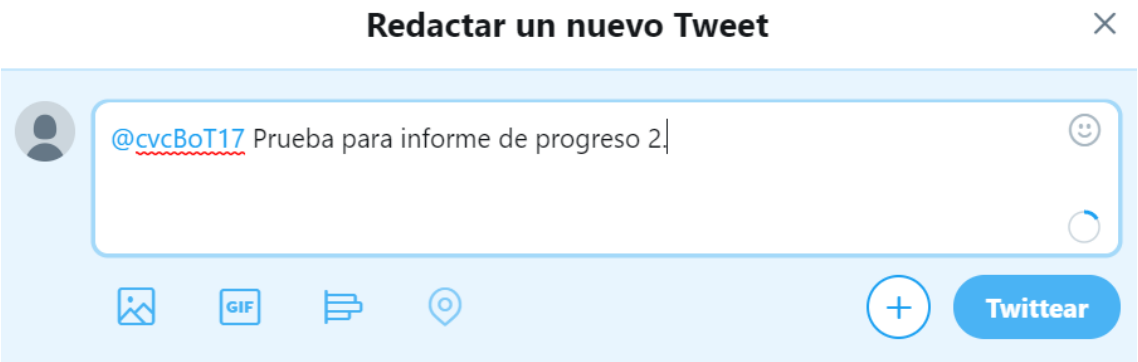

Figura 8: Tweet de ClienteBoT1111 hacia cvcBot17

2.El bróker recibe el mensaje prácticamente al momento. El mensaje se muestra por la salida del programa.

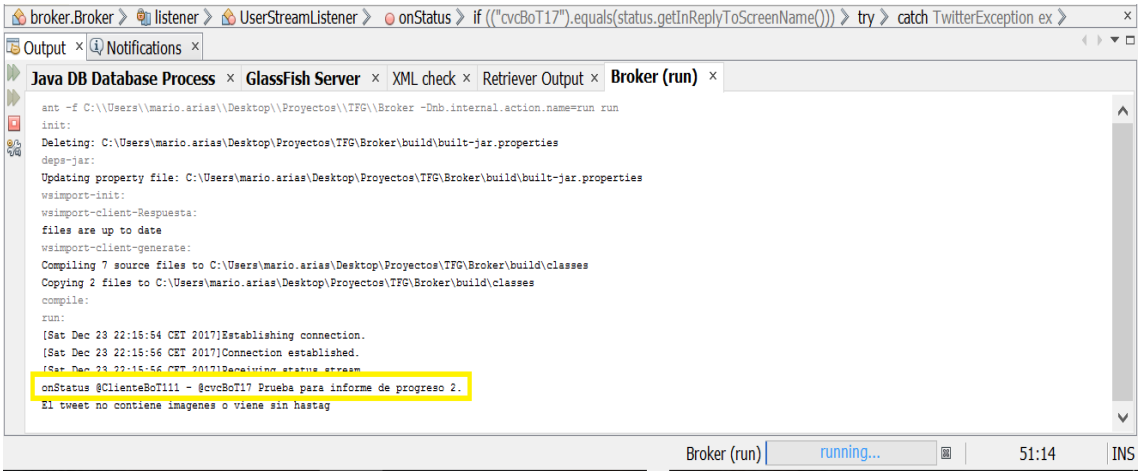

Figura 9: Consola de la aplicación bróker.

3. El bróker después de llamar al servicio de prueba, recibe la respuesta y envía automáticamente la respuesta en forma de tweet a la cuenta que había solicitado el servicio:

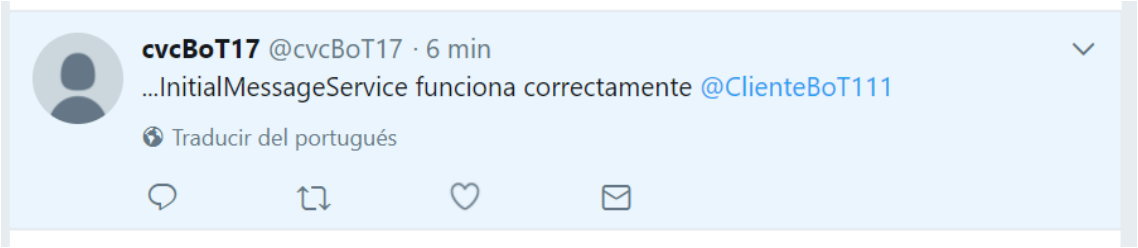

Figura 10: Tweet de respuesta de cvcBot17

2.3 Base de datos

Para el correcto funcionamiento del sistema todos los servicios que este pueda proporcionar serán almacenados en una base de datos no-relacional a la cual tendrá acceso tanto el bróker como el administrador del sistema. En la versión Alpha si bien se configurara la base de datos MongoDB, no se realizan consultas sobre ella, ya que solo tenemos un servicio.

- **Broker:** Realiza peticiones para recibir información de un determinado servicio. Si un determinado servicio se encuentra inactivo actualiza el estado de dicho servicio en la base de datos.
- **Administrador del sistema:** Puede realizar consultas, añadir y borrar servicios. En el caso de que un servicio vuelva a estar estable después de una caída, es el encargado de cambiar el estado de dicho servicio de manera manual.

2.3.1 Base de datos No relacional

Dado a 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. 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.

2.3.2 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, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta.

2.3.3 Esquema

Para almacenar la información en la base de datos seguiremos el siguiente esquema en formato JSON:

```
[
 {
  _id: '{{ServiceID()}}',
  serviceName: '{{serviceName()}}',
  uri: '{{uri()}}',
```

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

2.3.4 Aspectos relevantes

La base de datos estará alojada en el mismo servidor donde se alojará la aplicación Bróker. En esta versión Alpha únicamente existirá el servicio de prueba **InitialMessageService**. Al tratarse de un único servicio no se realizarán peticiones a la base de datos. Este desarrollo está enfocado a la versión Beta.

2.4 WebService SOAP

En esta versión Alpha únicamente existirá un servicio web que devolverá una cadena de texto. Esta implementación sirve únicamente para poder ver que el sistema es capaz de cerrar el circuito.

2.4.1 WSDL

```
 <?xml version="1.0" encoding="UTF-8" ?>
<definitions
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
    xmlns:wsp="http://www.w3.org/ns/ws-policy"
    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://InitialService/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://InitialService/" name="Respuesta">
    <types>
        <xsd:schema>
            <xsd:import namespace="http://InitialService/"
schemaLocation="http://win-
zlmdcdto5a:50852/MessageServiceInitial/Respuesta?xsd=1" />
        </xsd:schema>
    </types>
    <message name="MensajeInicial">
        <part name="parameters" element="tns:MensajeInicial" />
    </message>
    <message name="MensajeInicialResponse">
        <part name="parameters" element="tns:MensajeInicialResponse" />
```

```
    </message>
    <portType name="Respuesta">
        <operation name="MensajeInicial">
            <input
wsam:Action="http://InitialService/Respuesta/MensajeInicialRequest"
message="tns:MensajeInicial" />
            <output
wsam:Action="http://InitialService/Respuesta/MensajeInicialResponse"
message="tns:MensajeInicialResponse" />
        </operation>
    </portType>
    <binding name="RespuestaPortBinding" type="tns:Respuesta">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
        <operation name="MensajeInicial">
            <soap:operation soapAction="" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
    </binding>
    <service name="Respuesta">
        <port name="RespuestaPort" binding="tns:RespuestaPortBinding">
            <soap:address location="http://win-
zlmdcdto5a:50852/MessageServiceInitial/Respuesta" />
        </port>
    </service>
</definitions>
```

2.4.2 Test del Servicio

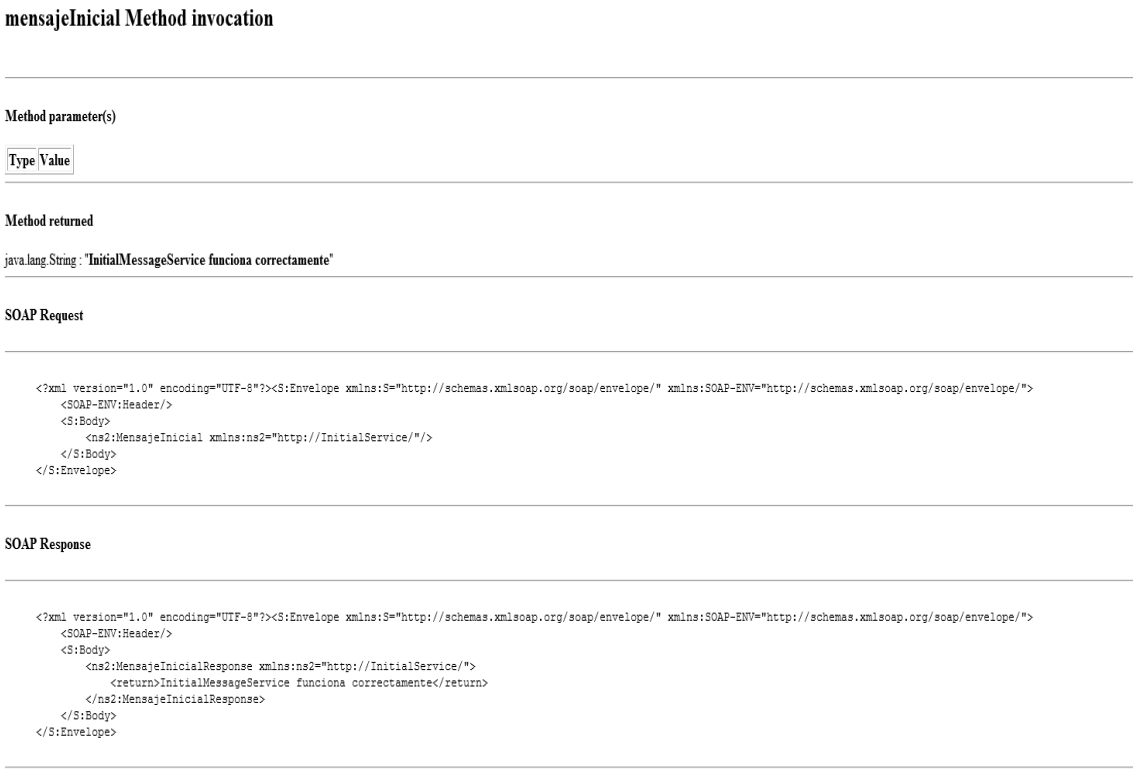

Figura 11: Test del servicio InitialMessageService

Vemos que el servicio está completamente operativo y listo para ser consumido. En la Figura 10 se puede ver el resultado de la respuesta del servicio una vez que el bróker ha cogido este mensaje y se lo ha enviado al cliente.

2.5 Estado de la versión

Estamos ante la primera versión del sistema. Al tratarse de una versión Alpha carece de algunas funcionalidades como por ejemplo la gestión de servicios (añadir y eliminar) y el consumo de un servicio de visión por computador. Usualmente los programas en estado Alpha no están preparados para ser utilizados por los usuarios. Por este motivo, el sistema se aloja en la maquina local donde se está desarrollando el software. En la versión Beta se prevé que los diferentes módulos del sistema estén alojados en una máquina del centro de visión por computador.

3. Versión Beta

Esta versión añade varias características importantes para la escalabilidad del sistema. Principalmente esta versión se centra en la configuración de los servicios para hacer que el sistema se pueda ampliar añadiendo nuevos servicios de tratamiento de imagen. Esta configuración deberá hacerse a nivel de base de datos para que el bróker sepa en todo momento los servicios disponibles.

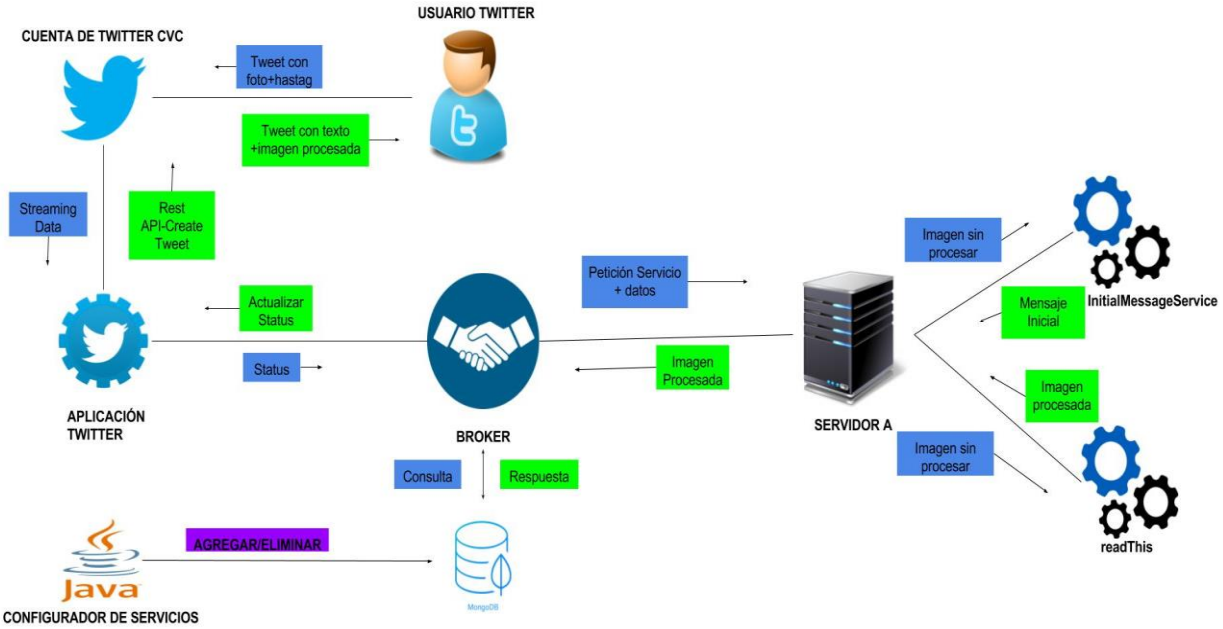

Figura 13: Diagrama funcional del sistema versión beta

3.1 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 2.3.3 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.

3.2 API REST de servicios

Para la creación de servicios de visión por computador, se ha creado una API REST alojada en la misma máquina que la aplicación bróker y la base de datos MongoDB. Para esta versión Beta prestara el servicio de **readThis** y el servicio migrado desde SOAP a la API REST que estaba en la versión Alpha **InitialMessageService**.

3.2.1 Justificación de cambio

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. Anteriormente en la versión Alpha al utilizar únicamente un servicio utilizaba SOAP. Viendo que la versión Beta debe ser dinámica y debe poder añadir y quitar servicios se ha optado por

esta alternativa, ya que para encontrar un servicio lo único que se necesita saber es la URI y nos abstrae de saber el servidor o el puerto, por ejemplo.

3.2.2 Características

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.

3.2.3 Ventajas

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

- 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.

3.3 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

La primera complicación que presenta este algoritmo es que está escrito en C++. Para que el web service sea independiente de lenguaje lo primero que se debe hacer es convertir este archivo CPP en un archivo ejecutable (.exe).

Como librería externa utiliza OpenCV 3.0 así como Tesseract OCR API. En un primer momento habrá que hacer que este módulo funcione de manera independiente. Una vez se consiga

ejecutar el programa, se deberá integrar como un Servicio API REST disponible para ser consumido por el bróker.

3.4 Bróker

Las funcionalidades del bróker en esta versión no se ven muy afectadas a excepción de algunas partes de su flujo normal de trabajo.

3.4.1 MongoDB Java Driver

El controlador oficial MongoDB Java proporciona interacción síncrona y asíncrona con MongoDB. Alimentando los controladores hay un nuevo núcleo de controlador y una biblioteca BSON. 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. Existen 3 posibles escenarios:

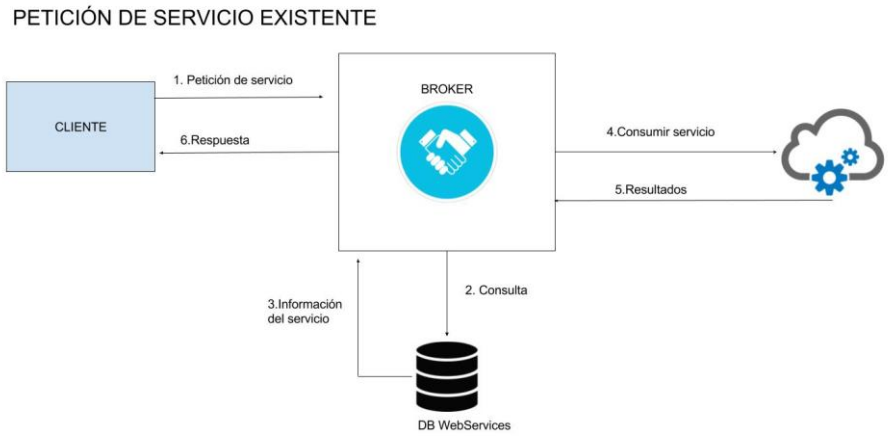

Figura 14: Diagrama funcional bróker, petición de servicio existente

En el flujo descrito en la Figura 14 podemos ver como el bróker, una vez recibe la petición de servicio junto con la posible imagen a tratar hace una petición a la base de datos que contiene la información de todos los servicios disponibles.

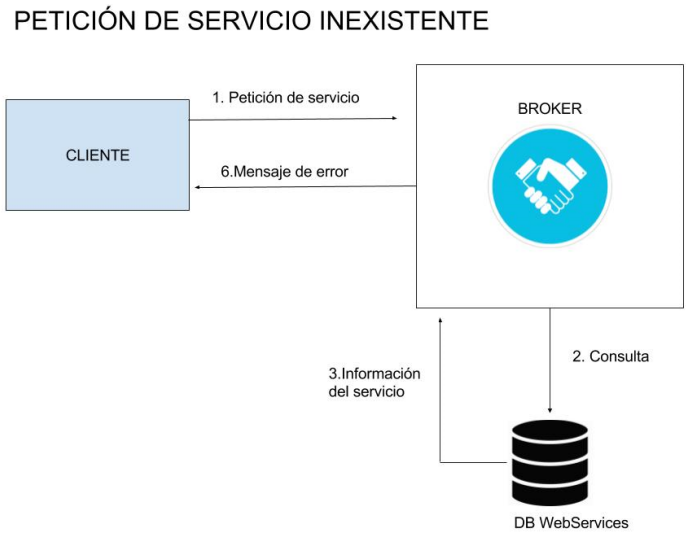

Figura 15: Diagrama funcional bróker, petición de servicio existente

En este segundo caso el servicio no existe. El bróker, al obtener una respuesta negativa por parte de la base de datos contestara al cliente que no ha podido realizar el servicio solicitado porque este no existe.

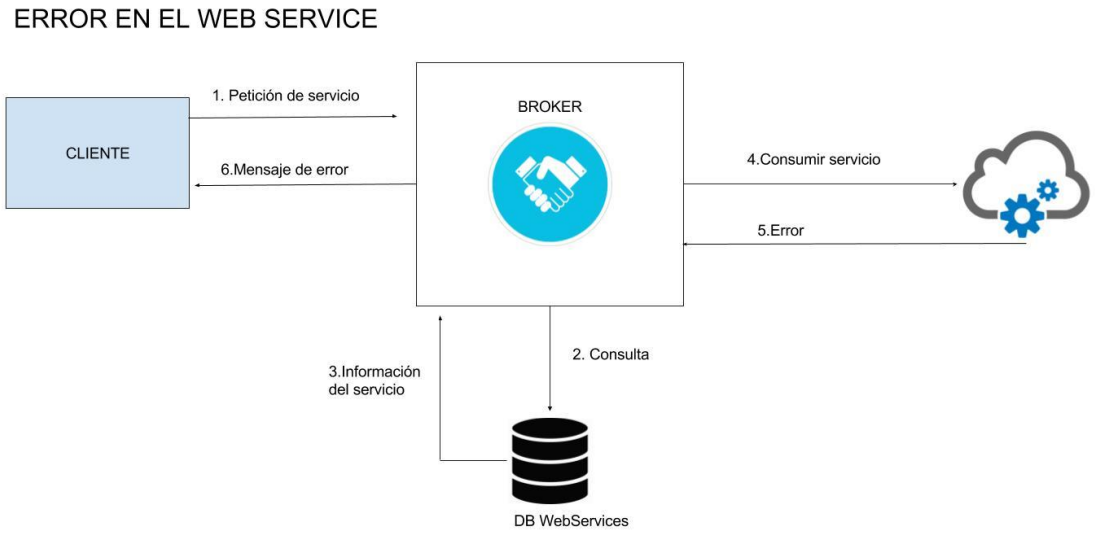

Figura 16: Diagrama funcional bróker, petición de servicio existente

En este caso el servicio se encuentra fuera de servicio. Para este caso el bróker deberá cambiar de estado activo a “fuera de servicio”, simultáneamente deberá enviar un mensaje al cliente informando de que el servicio solicitado se encuentra fuera de servicio.

3.5 Estado de la versión

En estos momentos esta versión se encuentra incompleta, si bien conceptualmente están abordados todos los módulos, falta el desarrollo total de alguno de ellos. Veámoslo con más detalle:

- **Servicio readThis**: Por problemas de disponibilidad y coordinación de calendario no ha sido posible recibir soporte con la puesta en marcha del algoritmo. Sin este primer paso no se ha podido proceder a la integración de este en el sistema.
- **Sistema en maquina del CVC**: La disponibilidad de las máquinas del CVC es algo que no depende directamente de los encargados de llevar a cabo el proyecto por lo que se sigue a la espera de una respuesta acerca de la maquina solicitada. Una vez se tenga, se procederá a realizar la migración de servidores y aplicaciones a la máquina del CVC.
- **Aplicación Java para la configuración de servicios**: La aplicación esta prácticamente terminada y resta realizar algunas pruebas para darla por finalizada.
- **API**: La API de servicios se encuentra finalizada y estable. Se ha realizado la migración del servicio **InitialMessageService** para que funcione con peticiones REST.

4 Discusión de resultados

4.1 Resultados Obtenidos

A fecha de hoy los resultados obtenidos se encuentran fuera de la planificación inicial ya que esta preveía tener finalizada la versión Beta para esta entrega. Esto ha requerido de un ajuste en los tiempos del proyecto, focalizando los esfuerzos en la finalización de la versión Beta. Por su parte la versión Alpha cumple con todos los requisitos establecidos y funciona correctamente.

- **Alpha:** Ha superado todas las pruebas, siendo capaz de realizar una comunicación de extremo a extremo del sistema. 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. Si bien todavía esta versión no nos ofrece la posibilidad del tratamiento de imágenes, sienta las bases de la arquitectura del sistema y consigue que la idea conceptual coja forma.
- **Beta:** No se han podido aplicar pruebas sobre ella porque todavía no se encuentra operativa.

4.2 Ajustes en la planificación

En previsión de que a la semana de vacacione de navidad, el desarrollo podía ir algo justo se dejó una semana de margen para posibles ajustes. Esto se indica en la planificación del informe inicial, y está marcado en rojo.

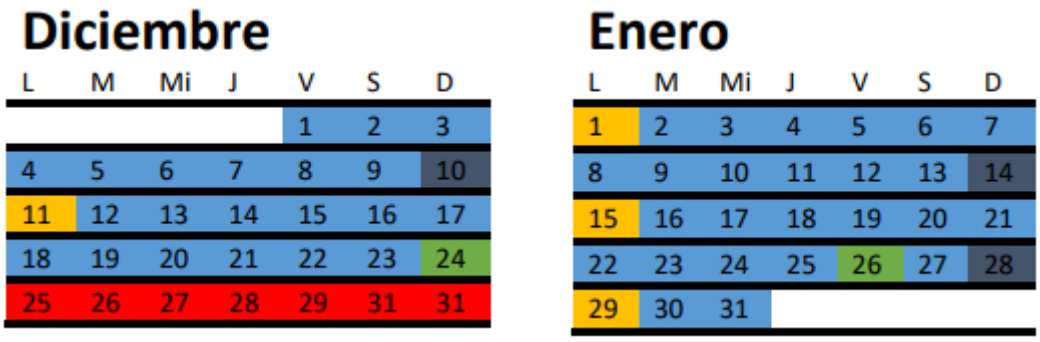

Figura 17: Planificación inicial meses de diciembre y enero

Los ajustes realizados para alcanzar los objetivos de una versión beta operativa se reflejan en la siguiente tabla:

Tarea	**Dias**
Servicio readThis	25,26,27,28
Aplicación Java	29,30
Integración aplicación Java	31,1
Integración servicio readThis	2,3
Migración de localhost a máquina del CVC	4,5
Testing	6,7

Tabla 1: Reajuste de la planificación inicial

5 Conclusiones provisionales

- La arquitectura del software permite que el sistema sea escalable en un futuro.
- La arquitectura 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 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.
- 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.

6 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-changing-program-cpp-exe.html