UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TRABAJO DE FIN DE GRADO EN
INGENIERÍA DEL SOFTWARE

Asistente Virtual (Chatbot) para la web de la Facultad de Informática

Autor: Ismael Vallejo Ruiz

Director: Juan Pavón Mestras

Septiembre 2015

I. Agradecimientos

En primer lugar quiero agradecer a mis padres, Jesús y Mari Paz, por lo exigentes que han sido conmigo desde que cursaba primaria. Siempre me han transmitido que todo esfuerzo tiene en el futuro sus resultadas. Quizás este proyecto es el resultado de esa exigencia.

De la misma forma a mis yayos, Julio y Julia, que han actuado como mis segundos padres, y siempre han estado preocupándose por mis estudios.

No puedo olvidarme de mi novia Sonia, quien ha tenido que aguantar días y semanas sin apenas vernos debido a exámenes, prácticas, y especialmente estos últimos meses, debido a la realización de este proyecto.

También quería mencionar a mi hermano Jesús, que aunque optó por no elegir la vida de la universidad, y posiblemente sea de las personas que más me haya alterado en toda mi vida, siempre ha sido el primero en ofrecerse cuando he necesitado un favor.

En cuanto a la realización del proyecto, me gustaría hacer un agradecimiento especial el director de este proyecto, Juan Pavón, quien me ha mostrado su ayuda incluso estando de vacaciones, para poder entregar a tiempo este proyecto. Hubo momentos en los hasta yo mismo dudaba que no estuviese a tiempo. Por suerte, finalmente puedo decir que aquí está terminado.

Por último, quería acordarme de todos los compañeros y compañeras que han pasado por mis años de vida como universitario. Han sido muchos momentos buenos vividos, y muchas risas echadas desde que llegamos el primer día y todo nos venía grande. Espero que todos ellos tengan un gran futuro.

II. Índice de contenidos

III. Índice de figuras

IV. Resumen

El proyecto tiene como objetivo implementar un asistente virtual o chatbot que permita facilitar la búsqueda de direcciones para un usuario que se conecta a la web de la facultad de informática de la Universidad Complutense de Madrid.

Un chatbot es un derivado de un buscador tradicional, que no sólo es capaz de encontrar una url a partir de las palabras de entrada, sino que es capaz de responder como si de una persona física se tratase, así como de reconocer oraciones completas, extraer la información necesaria y poder proporcionar una respuesta también en lenguaje natural. Para ello, se hace uso del lenguaje de programación AIML, así como de una base de datos, que son capaces de reconocer patrones de entrada, para mostrar una salida coherente en lenguaje humano, y en caso de no encontrar un patrón de entrada, utilizar un repositorio de urls indexadas para devolver las que más se ajusten.

V. Abstract

The project aims to implement a virtual assistant or chatbot which can facilitate address search for a user connects to the web of the Faculty of Information Technology at the Complutense University in Madrid.

A chatbot is a derivative of a traditional search engine, which is not only able to find a url from the input words, it is able to respond as if it were a natural person, and to recognize complete sentences, extract the necessary information and also to provide an answer in natural language. To do this, use the programming language AIML is made, as well as a database, you are able to recognize input patterns to show a consistent output in human language, and if not find a pattern of input, use a repository of urls indexed to return the most suited.

VI. Palabras clave

- Asistente virtual
- Chatbot
- Araña web
- Indexar
- Buscador
- Lucene
- Solr
- Nutch
- A.L.I.C.E.
- Scrum

VII. Keywords

- Virtual assistant
- Chatbot
- Crawler
- Index
- Search
- Lucene
- Solr
- Nutch
- A.L.I.C.E.
- Scrum

1. Introducción

1.1. Motivación

Cuando se decide desarrollar este proyecto, la primera pregunta que debe hacerse es: ¿De dónde partimos? ¿Hay algo que exista ya, y por tanto, se pueda mejorar?

En este caso, la idea era crear un asistente virtual, que como ya se explicó, es una evolución de los buscadores tradicionales, pero que son capaces de reconocer palabras clave ante una oración dada y responder como si una persona física estuviese detrás. Esto hace que el usuario pueda realizar consultas en lenguaje natural, con oraciones completas, y no sólo con palabras sueltas. Y curiosamente, un buscador tradicional es algo que ya existe en la web de la facultad, por tanto es momento de estudiar cómo funciona, que aporta, y cómo podemos mejorarlo.

Figura 1.1.1: Buscador actual de la web de la facultad

En cuanto a la interfaz, podemos decir que está en perfecta sintonía con el entorno de la web. Tiene buena ubicación (barra de título), pero sin embargo, podemos observar que sólo aparece en la página principal de la facultad. En cuanto intentamos navegar por alguna de las páginas, el buscador desaparece. Esto podría ser mejorable, haciendo que nuestro chatbot esté disponible desde cualquier punto dentro del dominio. Para esto, se desarrollará de la siguiente manera:

- Se creará una imagen con un link, que nos abra un pop-up (ventana emergente), que nos dé acceso a la conversación con nuestro chatbot. Esta imagen debería pertenecer a la plantilla de la web, de forma que automáticamente aparezca en todo el dominio. Sin embargo, esta parte se escapa del ámbito del proyecto, ya que no tenemos acceso a modificar código original de la web.

- En cuanto al pop-up, este deberá estar visible mientras el usuario lo considere oportuno y mientras no sea cerrado manualmente. Es decir, el usuario podrá realizar tantas búsquedas como desee, estando en cualquier página del dominio, aunque cambie de ésta, y deberán estar conectados de manera que desde la ventana emergente, se pueda modificar la url de la ventana principal.

Dejando a un lado la parte de interfaz e integración, pasamos a hacer un test al buscador actual. En mi caso, he optado por lanzar 10 preguntas, de dificultad ascendente, y observar el resultado obtenido.

- Localización -> Respuesta coherente
- Biblioteca -> Respuesta coherente
- Decano -> Respuesta coherente
- Campus virtual -> Respuesta coherente
- Adaptación a grado -> Respuesta coherente
- Licenciatura -> Sin resultados
- Geaportal -> Sin resultados
- Junta de gobierno -> Sin resultados
- Proyecto de fin de carrera -> Sin resultados
- Días festivos -> Sin resultados

Como se puede observar, ciertas páginas sí están indexadas en la web, pero sin embargo, el resultado no es del todo bueno. Posiblemente sea porque no se hace una búsqueda dinámica del contenido de la página en el momento de la búsqueda, sino que estas están previamente almacenadas (quizás en una base de datos) Esto nos deja un gran margen de mejora para este buscador, ya que debería existir una continua actualización del buscador para nuevos contenidos.

En segundo lugar, se ha probado a realizar búsquedas de oraciones con sentido en el buscador, y no ha habido ninguna pregunta con resultados. Esta será la segunda pieza clave de nuestro proyecto, poder buscar una página utilizando lenguaje coherente, no sólo palabras sueltas.

En resumen, podemos destacar las siguientes mejoras:

- Mejora de las búsquedas por parte de los usuarios.
- Posibilidad de realizar búsquedas en lenguaje natural, no sólo por palabras concretas.
- Actualización continua de las páginas indexadas, y por tanto de los resultados obtenidos.
- Interfaz mejorada.

1.2. Objetivos

Podemos destacar los siguientes objetivos principales para dicho proyecto:

- El objetivo principal del proyecto consiste en reducir el tiempo de búsqueda que un usuario tarda en encontrar una página dentro de la web de la facultad de informática, ya que, especialmente para un usuario nuevo en la web, le puede llevar tiempo encontrar exactamente lo que busca.
- La diferencia es que esto se realizará de forma que se pueda producir una interacción usuario-chatbot en lenguaje natural, de manera que el usuario pueda preguntar con oraciones completas, y el chatbot pueda responder con links, pero camuflados en lenguaje natural, para dar una breve descripción de dónde le conducirá el enlace, siempre que se refiera a preguntas relacionadas con la facultad.
- Para otro tipo de preguntas, el objetivo es que el chatbot sea capaz de redirigir al usuario hacia el contexto de la facultad, ya que no está diseñado para ser un chatbot con el que conversar de otros temas no relacionados.
- Esto será además una evolución del buscador ya existente en la facultad, que se ha comprobado que muchas de las páginas no están registradas.

En segundo lugar, y como objetivos necesarios del proyecto, podemos destacar:

- Familiarización con lenguaje de programación AIML para el procesamiento de lenguaje natural.
- Familiarización con herramientas de crawleo e indexación web (Nutch y Solr), para su uso como buscador de urls.
- Aprendizaje de la gestión del proyecto software.

1.3. Metodología y plan de trabajo

El proyecto se ha realizado utilizando un marco de desarrollo ágil, tipo scrum. Esta forma de desarrollo se caracteriza por:

- Entregas parciales y regulares del producto final. De esta forma se observan resultados de manera temprana, y de esa forma se van adoptando medidas para mejorar dichos prototipos hasta llegar a un resultado final.
- Cada una de estas iteraciones recibe el nombre de sprint. En nuestro caso, el proyecto está compuesto por 11 sprints, que podemos clasificar de esta forma:
    - Sprints de análisis y búsqueda de información: 1.
    - Sprints de desarrollo: 2, 5, 6, 8, 9 y 10.
    - Sprints de creación y modificación de interfaz: 3.
    - Sprints de corrección: 3.
    - Sprints de integración/implantación: 7, 11.

  Para más detalle acerca de cada uno de los sprints, se puede consultar el apartado *6: Evolución del proyecto*.

2. Asistentes virtuales (Chatbots)

En primer lugar, y a modo de introducción, es necesario ponerse en situación y saber dónde nos situamos para conocer el entorno que rodea al mundo de los chatbots.

Un chatbot no es más que un sistema que emula tener una conversación con una persona. Esto puede resultar trivial en un principio pero la realidad es que es un sistema de mucha complejidad, ya que lo que se intenta es que razones como si una persona real estuviese sentada detrás de la máquina, es decir, que sea capaz de responder oraciones coherentes, con sentido, y como si de una forma inteligente se tratase.

No se puede hablar de chatbots, o Inteligencia Artificial, sin nombrar también a Alan Turing. Como muchos ya conocerán, se dedicó en gran medida a la Inteligencia Artificial, y una prueba de ello, es el test que inventó (Test de Turing) para probar la habilidad de una máquina para hacerse pasar por una persona real, como si tuviese inteligencia propia (el juego de la imitación).

A grandes rasgos el test se considera por superado si más de un 30% de un jurado no es capaz de diferenciar si está hablando con una persona o una máquina en una conversación de 5 minutos. Turing predijo que hasta el año 2000 no se conseguiría superar este test; sin embargo, ha sido necesario llegar hasta el 2014, para que un chatbot, Eugene Goostman, consiguiese hacer creer al 33% del jurado que era un niño ucraniano de 13 años cuando era realmente un ordenador.

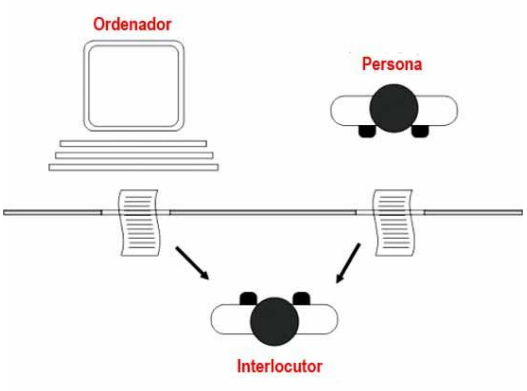

Figura 1.3.1: Test de Turing

Está claro que no todo el mundo comparte la misma opinión. Muchas personas creen que superar este test no da inteligencia a la máquina, sino simplemente es un algoritmo muy complejo que proporciona respuestas automáticas en función de la entrada que recibe, y eso no es inteligencia. Lo que está claro, es que la Inteligencia Artificial, es una rama de la informática que ha avanzado mucho en los últimos años, y uno de esos casos son los chatbots.

Ahora bien, ¿qué ventajas puede proporcionar un chatbot frente a una persona humana? En la actualidad, son muchas las empresas que están invirtiendo una cantidad importante de capital para adaptar chatbots a sus necesidades.

Llevándolo todavía más al terreno del proyecto, un chatbot por sí mismo, es capaz de atender una gran cantidad de peticiones simultáneamente, algo que una persona no puede hacer, por ejemplo, en un Call Center.

Esto conlleva ventajas económicas por parte de las empresas, ya que se lo ahorran en personal, además de que el número de peticiones resueltas por unidad de tiempo sería mucho mayor que si de una persona se tratase.

Este es el caso de nuestro proyecto. La idea no es crear un sistema que sea capaz de superar el Test de Turing. Eso sería un proyecto muy ambicioso que se escapa del objetivo de este proyecto. La idea es crear un sistema que pueda ayudar a todas las personas que necesitan encontrar información en la página de la facultad de informática, y no engañar de si lo que hay detrás es o no es un humano, ya que está claro que no lo será. Con esto se podría, por ejemplo, atender a más estudiantes en menos tiempo, con respuestas más actualizadas, y evitando llamadas masivas, por ejemplo, en épocas de matriculación.

Por último, y a modo de ejemplo, me gustaría nombrar 2 chatbots que actualmente están implantados en sus respectivos dominios y que ayudan en las tareas de los usuarios que se conectan. Obviamente hay muchos más, en una gran diversidad de idiomas, pero me voy a centrar en 2 que son en español, y en los que en cierta manera me he basado para desarrollar el proyecto.

El primero es el chatbot de Renfe, Irene, que nos ayuda en la tarea de encontrar reservas, billetes, e información relevante sobre esta empresa ferroviaria.

Irene me dio la idea de realizar el módulo de conversación en una ventana emergente, de forma que pudiese ser más sencillo usarlo para otro tipo de aplicaciones, ya que es un módulo totalmente independiente del código de la propia web.

Además de ello, la forma que tiene de responder Irene también es muy parecida a la de mi chatbot, ya que hace un breve comentario sobre el contenido de la página destino, para después proporcionar un enlace camuflado en dicha respuesta.

Figura 1.3.2: Irene: ejemplo del chatbot utilizado por Renfe

El segundo que os voy a presentar es debido a que la temática está íntimamente relacionada con el proyecto, ya que también es un "chatbot universitario", concretamente el de la web de la Universidad de Granada. Su nombre es Elvira.

Este no está creado en un pop-up, sino en un widget movible creado en Flash, dentro de la propia web, que particularmente en mi opinión, puede molestar en ciertos momentos de la navecación, de ahí mi idea de usar el anterior ejemplo. Este chatbot me sirivió para contrastar las respuestas que proporcionaba ante ciertas preguntas y compararlas con las que daba el mío en un contexto parecido.

Figura 1.3.3: Elvira: ejemplo del chatbot utilizado por la UGR

3. Definición del proyecto

Como ya se ha comentado más arriba de forma más general, el cometido del sistema es el de procesar una entrada escrita por un usuario que se conecte a la web de la facultad y necesite ayuda de navegación. Esta entrada debe ser analizada por el chatbot, para proporcionar una salida coherente y que se adapte a la necesidad del usuario.

Por ello, a continuación se especifica una definición más concreta del alcance del proyecto, que ayuda a entender los diferentes casos de uso especificados, diferentes escenarios de prueba creados y en general, una descripción más detallada de la funcionalidad del proyecto.

3.1. Descripción de la funcionalidad del sistema

En este apartado, se pretende describir, en un primer acercamiento en alto nivel, los diferentes escenarios que se pueden presentar dentro de la aplicación, en función de la entrada especificada por el usuario. Esto hará que la aplicación tome un recorrido u otro para llegar a la respuesta que más se adapte a dicha entrada. Gracias a este primer análisis de escenarios, será más sencillo identificar unos casos de uso concretos, así como establecer la arquitectura más adecuada, para una correcta cohesión de los módulos necesarios, y generar los diagramas de secuencia más óptimos para el sistema.

Por tanto, debido a la organización de los módulos elegida, he decido crear un posible escenario de uso para cada camino que puedan seguir los datos, abarcando de esta manera, el mayor número de casos de uso diferentes.

- **Escenario 1: Abriendo la aplicación**

  (Mensaje automático)

  Lucy: ¿Qué tal? Soy Lucy, experta en asuntos de la facultad de informática. Dime sobre qué aspectos te gustaría buscar información.

- **Escenario 2: Saludo**

  Pregunta: ¡*Hola*!
  Lucy: Hola, ¿tienes alguna pregunta acerca de la facultad de informática?

- **Escenario 3: Conversación de temas básicos**

  Pregunta: Si no es mucha indiscreción, ¿me puedes decir la edad que tienes?
  Lucy: Soy de julio del 2015, o sea que tengo poco más de 1 mes.

- **Escenario 4: Palabra clave reconocida**

  Pregunta: junta directiva
  Lucy: Puedes hacer click aquí para encontrar información sobre la junta de facultad.

- **Escenario 5: Oración con palabras clave reconocidas**

  Pregunta: ¿Quién es el actual decano?
  Lucy: Desde este enlace podrás encontrar información referente al decano de la facultad: Daniel Mozos Muñoz.

- **Escenario 6: Palabras claves no reconocidas, pero sí encontradas con Solr**

  Pregunta: ¿En qué consiste el PGP Party?
  Lucy: He encontrado algunas urls para tu búsqueda:
  http://informatica.ucm.es/facultad
  http://informatica.ucm.es/pgp-party-2014-12-04
  http://informatica.ucm.es/eventos

- **Escenario 7: Oración desconocida**

  Pregunta: ¿Qué sabes acerca de la filología hebrea?
  Lucy: Lo siento, no he encontrado ninguna url que se adapte a tus necesidades.
  Revisa si no has cometido ninguna falta de ortografía o intenta hacer una búsqueda más precisa.

- **Escenario 8: Insulto**

  Pregunta: G1L1P0LL4S
  Lucy: Creo que estás siendo un obsceno. No quiero ayudar a personas así. Me marcho. (Se cierra la ventana)

- **Escenario 9: Despedida**

  Pregunta: *Adiós*
  Lucy: Hasta luego. (Se cierra la ventana)

3.2. Especificación de requisitos

3.2.1. Requisitos funcionales

- **RF-01 – Responder directamente ante patrón encontrado**

  Se proporcionará una respuesta inmediata si la entrada es resuelta como patrón AIML.

- **RF-02 – Reconocer palabra clave / oración desde la base de datos**

  Se proporcionará una breve descripción, así como un link de redirección cuando la entrada coincida con alguna entrada de la base de datos de palabras clave.

- **RF-03 – Analizar oración y extraer palabras clave**

  Se analizará una oración de entrada, extrayendo todas aquellas palabras que cuya categoría gramatical sea de "sustantivo/nombre" para desechar palabras comunes.

- **RF-04 – Utilizar buscador para devolver urls recomendadas**

  Se hará uso de un buscador adicional para resultados no registrados que guarde todo el contenido de la web y pueda hacer una búsqueda más completa.

- **RF-05 – Detectar y gestionar insultos y palabras malsonantes**

  Si el chatbot detecta una palabra malsonante, expulsará al usuario de la conversación.

- **RF-06 – Gestionar uso de acentos en búsquedas**

  Se realizará un parseo de la entrada para evitar problemas con acentos en las búsquedas.

- **RF-07 – Gestionar transformaciones de palabras en lexemas básicos**

  Se realizarán las transformaciones oportunas para analizar los sustantivos de entrada en masculino singular.

- **RF-08 – Redireccionar url de ventana principal al elegir respuesta**

  Se deberá redireccionar la venta principal, y no la ventana del chatbot, para permitir al usuario trabajar con ambas ventanas simultáneamente.

- **RF-09 - Detectar despedida para cerrar ventana**

  Se detectará una despedida por parte del usuario, para cerrar la ventana del chatbot pasados unos segundos.

3.2.2. Requisitos no funcionales

- **RNF-01 – Disponibilidad total**

  Se estima que el sistema tendrá un porcentaje de disponibilidad del 99%, lo que significa que se destinarán unas 100 anuales para labores de mantenimiento, actualización de repositorio, o solución de incidencias encontradas.

- **RNF-02 – Usabilidad**

  Se deberá asegurar que cualquier tipo de usuario pueda utilizar la aplicación, indiferentemente de si es un usuario con o sin conocimientos del contenido de la web. Para ello, se dispondrá de una interfaz gráfica durante la conversación, disponible con un sólo click, y que no presente dudas de utilización.

- **RNF-03 – Concurrencia entre peticiones simultáneas**

  El sistema deberá ser capaz de atender peticiones simultáneas sin mezclar respuestas entre usuarios diferentes. El sistema de por sí no contempla límites impuesto para el número máximo de conexiones, siempre dependerá de la capacidad del servidor donde sea instalado.

- **RNF-04 – Interfaz acorde con la estética actual**

  El módulo deberá ser instalado sin alterar la estética actual de la web de la facultad. Para ello, se deberán utilizar los mismos estilos, fuentes, colores etc…

- **RNF-05 – Portabilidad a otros entornos**

  Se deberá asegurar la posible portabilidad en otros entornos de ejecución, tales como cambios en el servidor, sistema operativo, navegador etc…

- **RNF-06 – Escalabilidad ante crecimiento de la información**

  El sistema debe estar preparado para albergar más cantidad de información sin que esta reduzca su calidad en las respuestas obtenidas.

- **RNF-07 – Mantenibilidad del sistema**

  Se debe asegurar una sencilla resolución de cualquier problema que pueda surgir en el funcionamiento del sistema.

- **RNF-08 – Rendimiento en tiempos medios de respuesta**

  Se debe asegurar un tiempo medio de respuesta entre todas las entradas, establecido en 3 segundos. Este resultado es una aproximación media compuesto por los bajos tiempos de respuesta cuando se usa como buscador (1 seg aprox.) y los más altos tiempos de respuesta cuando se utilizan oraciones completas y es necesario analizarlas (6 seg aprox.).

- **RNF-09 – Rendimiento en tiempos máximos de respuesta**

  No se debe superar un tiempo máximo de respuesta por entrada, establecido en 6 segundos.

3.3. Casos de uso

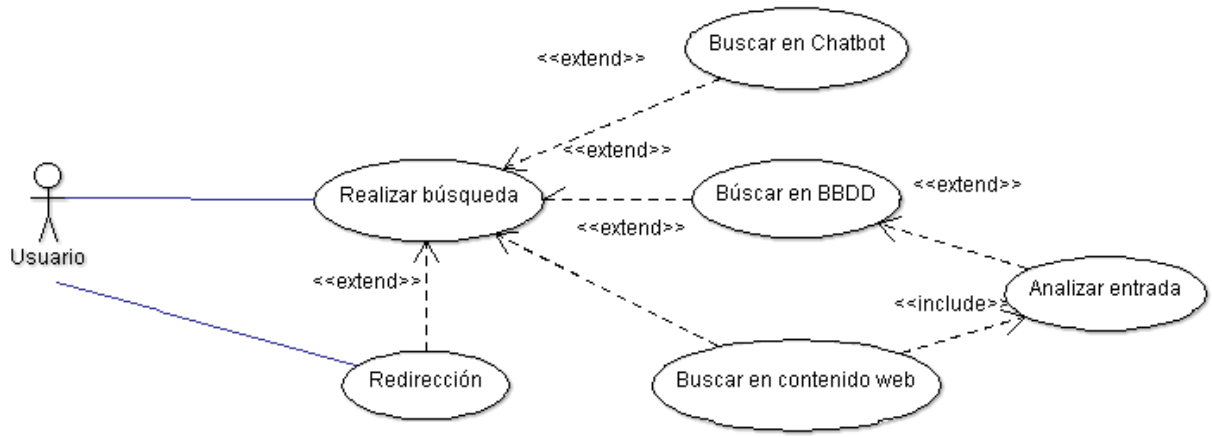

Figura 3.3.1: Diagrama de casos de uso del actor Usuario

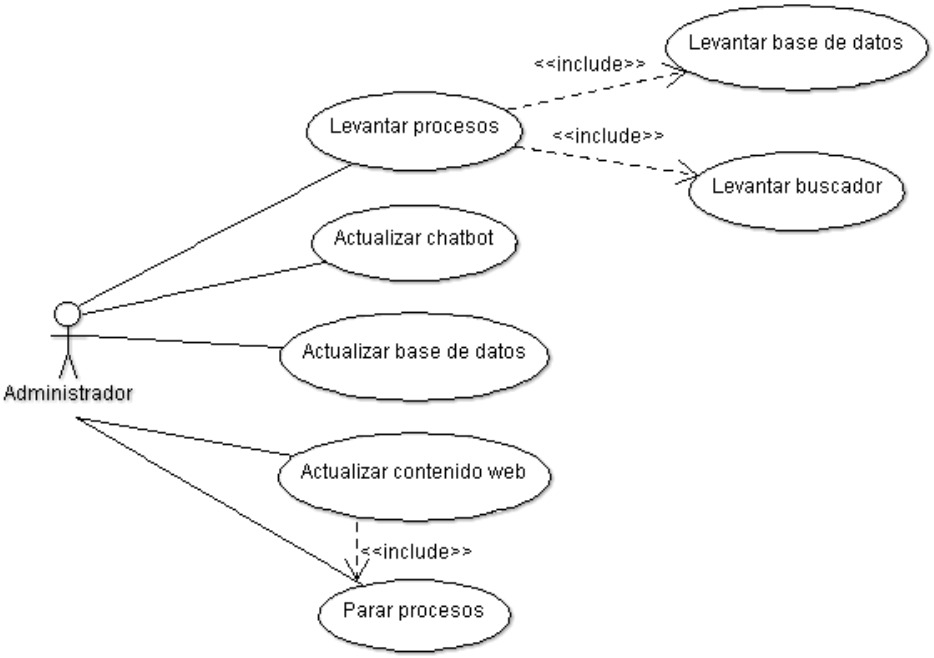

Figura 3.3.2: Diagrama de casos de uso del actor Administrador

CU-01	Realizar búsqueda	
Dependencias		
Precondición	Sistema abierto	
Descripción	El usuario realiza una búsqueda en el sistema	
Secuencia normal	**Paso**	**Acción**
	1	Click en imagen que abre el pop-up
	2	Escribir entrada en cuadro de texto
	3	Presionar enter o hacer click en botón de buscar
Postcondición	El sistema devolverá los resultados encontrados	
Excepciones	**Paso**	**Acción**
	3a	Algún proceso no está levantado

CU-02	Buscar en chatbot	
Dependencias		
Precondición	CU-01: Realizar búsqueda	
Descripción	El sistema utiliza el motor del chatbot para encontrar patrones que se adapten a la entrada.	
Secuencia normal	**Paso**	**Acción**
	1	El controlador envía la entrada al motor del chatbot
	2	Se reconoce un patrón y se devuelve una respuesta
Postcondición	Se muestra por pantalla la respuesta del chatbot	
Excepciones	**Paso**	**Acción**
	1a	Error en motor del chatbot. Revisar log.
	2a	No se reconoce ningún patrón. CU-03: Buscar en base de datos.

U-03	Buscar en base de datos	
Dependencias		
Precondición	Base de datos abierta. CU-02: Buscar en chatbot o CU-05: Analizar entrada	
Descripción	El sistema utiliza la base de datos para encontrar entradas similares a la entrada.	
Secuencia normal	**Paso**	**Acción**
	1	Se hace un SELECT LIKE de la entrada.
	2	Se devuelve la respuesta.
Postcondición	Se muestra por pantalla la respuesta de la base de datos y un link para el usuario.	
Excepciones	**Paso**	**Acción**
	1a	Base de datos no abierta. Levantar base de datos.
	1b	Entrada no encontrada. Devolver que no hay respuesta para gestionar llamada a “Analizador” o “Buscador”.

CU-04	Buscar en contenido web	
Dependencias	CU-05: Analizar entrada	
Precondición	CU-01: Realizar búsqueda	
Descripción	Busca urls cuyo contenido se adapte a la entrada.	
Secuencia normal	**Paso**	**Acción**
	1	Manda entrada a buscador.
	2	Devuelve links de urls que se adapten.
Postcondición	Se muestran por pantalla los links de las url que se hayan encontrado.	
Excepciones	**Paso**	**Acción**
	1a	Buscador no levantado. Levantar buscador.
	2a	No encuentra resultados.
	2b	Mostrar por pantalla que se haga otro tipo de búsqueda.

CU-05	Analizar entrada	
Dependencias	Servidor GRAMPAL funcionando	
Precondición		
Descripción	Se extraen los sustantivos de una oración.	
Secuencia normal	**Paso**	**Acción**
	1	Se pasa por HTTP GET la oración a analizar.
	2	Se genera una tabla del análisis de la oración.
	3	Se extraer por DOM los campos de la tabla necesarios.
	4	Se almacenan los datos en una estructura.
Postcondición	Array de sustantivos y array de lexema de sustantivos.	
Excepciones	**Paso**	**Acción**
	1a	GRAMPAL caído. Esperar que vuelva a estar operativo.

CU-06	Redirección	
Dependencias		
Precondición	Url encontrada	
Descripción	El usuario elige una url para ser redirigido.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click sobra la url a la que quiera ser redirigido.
	2	La ventana principal cambia de url
Postcondición	El usuario se encuentra en la nueva url sin perder el pop-up del chatbot.	
Excepciones	**Paso**	**Acción**
	1a	No hay ninguna url. Realizar otra búsqueda.
	2a	La url ya no existe. Hablar con administrador.

CU-07	Levantar procesos	
Dependencias	Servidor funcionando	
Precondición		
Descripción	Levantar procesos necesarios para que los usuarios puedan utilizar el chatbot.	
Secuencia normal	**Paso**	**Acción**
	1	CU-08: Levantar base de datos
	2	CU-09: Levantar buscador
Postcondición	Los procesos quedan levantando y el chatbot funcionando.	
Excepciones	**Paso**	**Acción**
	1a	La base de datos no levanta. Revisar logs. Hablar con el DBA.
	2a	El buscador no levanta. Revisar logs.

CU-08	Levantar base de datos	
Dependencias		
Precondición		
Descripción	Poner en funcionamiento la base de datos para realizar consultas.	
Secuencia normal	**Paso**	**Acción**
	1	Asegurar datos de conexión.
	2	Levantar proceso de base de datos.
Postcondición		
Excepciones	**Paso**	**Acción**
	1a	Datos incorrectos. Revisar datos de conexión.
	2a	El proceso no levanta. Revisar log (mysql_error.log).

CU-09	Levantar buscador	
Dependencias		
Precondición		
Descripción	Levanta el proceso del servidor SOLR	
Secuencia normal	**Paso**	**Acción**
	1	Ejecutar el proceso. java –jar start.jar&
	2	SOLR se ejecuta en segundo plano
Postcondición	Buscador listo para recibir peticiones GET.	
Excepciones	**Paso**	**Acción**
	1a	Aparece un error y no levanta. Revisar error. Revisar logs.

CU-10	Actualizar chatbot	
Dependencias		
Precondición		
Descripción	Actualizar respuestas del chatbot para conversaciones básicas.	
Secuencia normal	**Paso**	**Acción**
	1	Abrir administrador del chatbot.
	2	Loguearse.
	3	Realizar actualizaciones oportunas en archivos AIML.
Postcondición	Chatbot actualizado con éxito.	
Excepciones	**Paso**	**Acción**
	1a	No se llega al administrador. Revisar ruta.
	2a	Login incorrecto. Revisar credenciales.

CU-11	Actualizar base de datos	
Dependencias		
Precondición		
Descripción	Actualiza palabras, direcciones y respuestas frecuentes.	
Secuencia normal	**Paso**	**Acción**
	1	Entrar en administración de base de datos.
	2	Loguearse.
	3	Añadir, modificar o eliminar datos de las tablas “palabras_clave”, “urls_comunes” y/o “relacion”.
Postcondición	La base de datos queda actualizada.	
Excepciones	**Paso**	**Acción**
	1a	Imposible acceder al administrador. Revisar acceso y login.
	2a	Hablar con superusuario de la base de datos para restaurar credenciales.
	3a	Error al insertar. Registro ya existente. Revisar registro y actualizar si procede.

CU-12	Actualizar contenido web	
Dependencias		
Precondición	CU-13: Parar procesos	
Descripción	Actualiza el repositorio de urls y contenido web.	
Secuencia normal	**Paso**	**Acción**
	1a	Ejecutar o programar un nuevo crawleo.
Secuencia alternativa	**Paso**	**Acción**
	1b	Abrir administración de Solr
	2b	Insertar, actualizar o eliminar documentos necesarios.
Postcondición	Repositorio de urls actualizado.	
Excepciones	**Paso**	**Acción**
	1a	Fallo en crawleo. Revisar logs de Nutch y de Solr.
	1b	Imposible entrar en administración de Solr. Revisar url y logs.

CU-13	Parar procesos	
Dependencias	Procesos abiertos	
Precondición		
Descripción	Baja los procesos	
Secuencia normal	**Paso**	**Acción**
	1	CU-07 a la inversa.
Postcondición	Procesos bajados. Chatbot no funciona correctamente aunque haya acceso desde la interfaz.	
Excepciones	**Paso**	**Acción**
	1	Mismas excepciones de CU-07.

4. Tecnología utilizada

4.1. Servidor local

Un servidor local es una aplicación que se instala en un ordenador cualquiera, y actúa como servidor de páginas web, entre otras funcionalidades.

En mi caso, he utilizado XAMPP por varias razones.

En primer lugar porque llevo años utilizándolo y me ha ido muy bien. Sencillo de instalar, y con resultados muy buenos. Las siglas de XAMPP describen perfectamente su funcionalidad.

- X: Válido para prácticamente todos los sistemas operativos. En mi caso utilicé Linux, en su distribución CentOS 5.
- A: Apache. Servidor web puro. Encargado de servir las páginas web.
- M: MySQL. Gestor de Base de Datos. Lo utilizo como almacén de datos tanto de ficheros AIML (gestionados por Program-O), como para almacén de búsquedas típicas.
- P: PHP. Crucial para este proyecto. El 80% de código escrito ha sido en PHP para la gestión de los diferentes clientes implementados. El 20% restante ha sido HTML, CSS, SQL y JavaScript.
- P: Perl. En mi caso no lo he utilizado, o al menos no explícitamente, pero viene instalado por defecto.

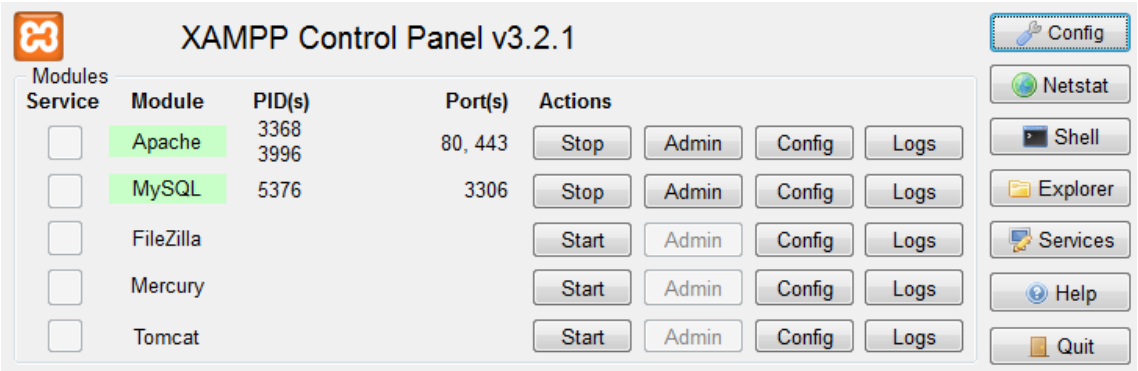

Figura 4.1.1: Interfaz de XAMPP

4.2. Araña web

La araña web, o crawler, es otro de los pilares fundamentales de este proyecto. Es el encargo de recorrer un dominio web, en mi caso la web de la facultad, e ir indexando todas las url que vaya encontrando, además de ir entrando en dichas direcciones para realizar la misma indexación de manera recursiva.

De esta forma, una vez ha finalizado, tendremos un índice de palabras clave que apuntarán a aquellas direcciones donde encuentre resultados.

En mi caso, por recomendación del profesor, utilicé Apache Nutch, un motor de búsqueda y crawler basado en Lucene. Se puede usar tanto el Linux como en Windows (mediante la tecnología cygwin). En mi caso, como ya he dicho, utilicé Linux.

La versión que utilicé fue la 1.9 que admite su uso integrado con la herramienta Solr, de manera que todas las páginas que indexa en la base de datos, lo lanza directamente a Solr, y así después poder usarse como buscador.

Como dato, una de las formas de tener siempre el repositorio actualizado, es programar una tarea (en mi caso con Cron de linux), para que por ejemplo, por las noches que hay menos carga en la web de la facultad, actualice el repositorio indexando nuevamente la web.

Uno de los parámetros ajustables es el número de iteraciones. Cuanto más haya, más tiempo tarda en indexar, pero más cantidad de urls recuperará. En mi caso, después de algunas pruebas, elegí 5 como valor ideal, ya que por debajo recuperaba muchas menos urls, y a partir de 5 apenas había diferencia de resultados, pero sí de aumento del tiempo, que era de varias horas. Finalmente se recuperaron más de 1000 urls, tardando más de 2 horas.

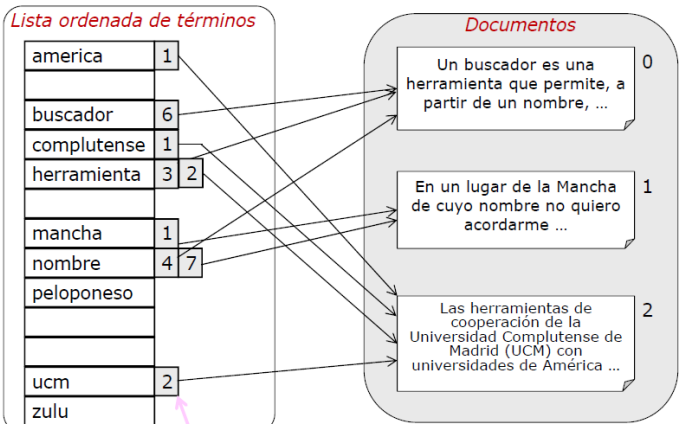

Figura 4.2.1: Metodología de indexación de Nutch

4.3. Motor de búsqueda

Junto al crawler, forman el pilar fundamental de este proyecto. Su misión es la de almacenar toda la información recogida por la araña web (Nutch), y hacer las funciones de servidor de los resultados al cliente que lo necesite.

El software que se ha elegido para esta misión, también por recomendación del profesor, ha sido Solr.

En concreto, yo sólo he utilizado la parte de buscador, pero Solr da la opción de insertar, actualizar y eliminar contenido. Su funcionamiento, aunque no es SQL, tiene una base en común. En el caso de las búsquedas, hay que definir ciertos campos, que en mi caso han sido los siguientes:

- qt (Query Type): tipo de consulta -> SELECT
- q (Query): query a realizar -> CONTENT:<palabra clave>
- start, rows: número de campos -> 0, Nº REGISTROS
- fl (Field List): campos a devolver -> url
- wt (Writer Type): formato de salida -> PHP

Como se observa en la imagen, esto nos devuelve un Array PHP, que podemos interpretar con este lenguaje de programación.

En el ejemplo, solicitamos que nos devuelva las urls en cuyo contenido web aparezca la palabra “english”, en formato PHP y con 10 resultados como máximo.

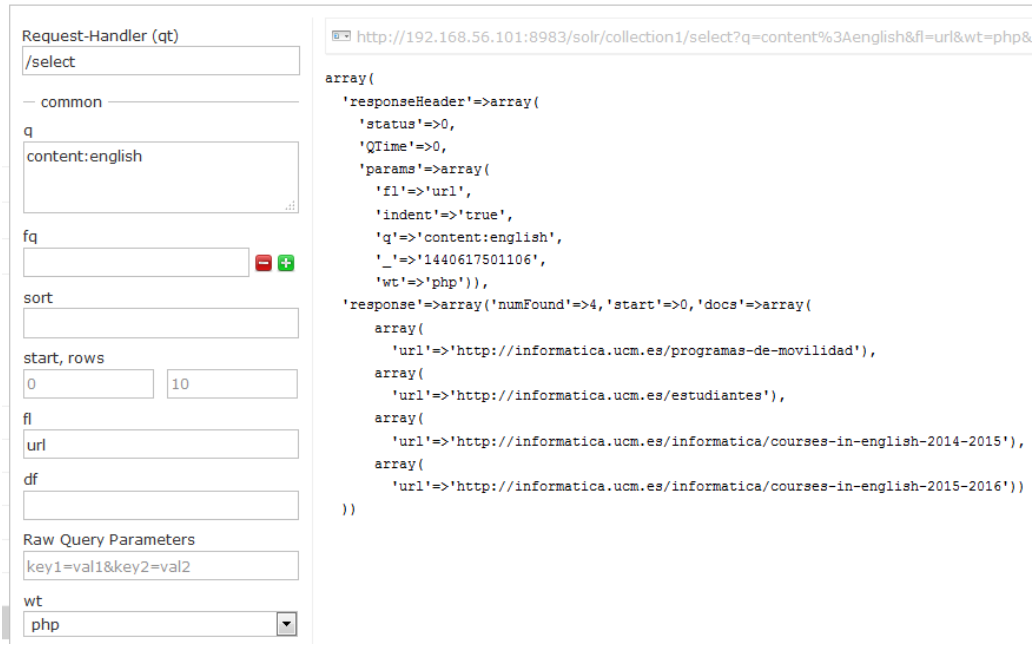

Figura 4.3.1: Ejemplo de búsqueda en Solr

4.4. Analizador morfológico

Este apartado, es precisamente el que marcará la diferencia entre un buscador tradicional, y un verdadero chatbot, y es la capacidad de procesar oraciones, extraer la información específica y usarla en el buscador.

En nuestro caso, la herramienta utilizada, o mejor dicho, la web utilizada, ha sido GRAMPAL, una herramienta creada por la Universidad Autónoma de Madrid como analizador morfológico.

Esta tecnología ha sido la única, en todo el proyecto, que se ha dejado en manos de una herramienta de terceros; y tiene sus desventajas claras. Es verdad que el funcionamiento, una vez entendidas sus limitaciones, es más o menos el esperado, pero también es cierto, que si hubiese que elegir un punto a mejorar en el futuro, sería este.

Una vez que dejas una herramienta en manos de terceros, tienes que ver las limitaciones que se presentan. Estas han sido algunas de ellas.

- Dependes a un 100% de su servidor. Si este se cae, no hay posibilidad de usarse.
- Se ha comprobado que es una web un tanto lenta. Con esto quiero decir, que desde que le mandas una oración, hasta que obtienes la información, tiene una media de entre 5 y 10 segundos, lo cual es bastante. En el caso de una búsqueda puntual, puedo no llegar a notarse, pero si una persona mantuviese una serie de consultas, podría llegar a ser bastante pesado.
- Muchas veces, el resultado no es del todo esperado debido a las reglas gramaticales que posee. Esto ha llevado a tener que hacer un análisis previo de la consulta realizada por el usuario antes de mandarlo a esta web.

Su funcionamiento es sencillo: le mandas una oración, en mi caso por el método GET, lo analiza, y te genera una tabla con la categoría gramatical, y su lema. Después, ya es tarea de nuestro cliente, utilizar esos datos, que en mi caso, lo he hecho mediante DOM.

Cuándo	*categoría*	***P***	*lema*	***CUÁNDO***			
son	*categoría*	***V***	*lema*	***SER***			
los	*categoría*	***ART***	*lema*	***EL***	*rasgos*	***masculino plural***	
exámenes	*categoría*	***N***	*lema*	***EXAMEN***	*rasgos*	***masculino plural***	

Figura 3.2.2.1: Ejemplo de tabla devuelta por Grampal

4.5. Chatbot

La esencia del proyecto, la herramienta principal. Esta parte se ocuparía de las conversaciones, no de las búsquedas. Y era importante matizarlo bien, porque el chatbot debía diferenciar una oración del tipo: "¿Cómo te llamas?" de "¿Cuándo son los exámenes?". Y aquí entraba en juego el mundo de los chatbots y del lenguaje AIML.

Tras analizar varios motores de procesamiento de AIML, así como algunos chatbots online, finalmente me decanté por un motor propio, y concretamente fue Program-O, un proyecto que contiene un completo panel de administración de chatbots, y además, implementado en PHP, que era justo el lenguaje utilizado durante todo el proyecto.

Desde aquí, se podían configurar todas las opciones del chatbot: archivos AIML importados, modificación de los mismos, palabras malsonantes, palabras mal escritas, pruebas etc…

Figura 4.5.1: Interfaz de Program-O

4.6. Servidor de producción

Una vez llegado al punto de la implantación final, se debe proveer de un servidor capaz de soportar ciertas características, entre las que podemos enumerar:

- PHP.
- Base de Datos MySQL.
- Servidor Solr.

La gran mayoría de los alojamientos web actuales, tienen cabida para los 2 primeros, sin embargo, posiblemente sea necesario adaptar, o incluso migrar, el actual servidor de la web de la facultad, para asegurar que se pueda instalar un módulo que gestione las peticiones a Solr, si el actual no dispone de él.

4.7. Navegador web

Finalmente, para que los usuarios finales puedan utilizar la aplicación, simplemente necesitarán el uso de un navegador web, exactamente igual que cuando entran a la web de la facultad, ya que el sistema no tiene ningún módulo incompatible con ningún navegador actual.

Aun así, se recomienda el uso de los navegadores Mozilla Firefox (con el que se han realizado la gran mayoría de las pruebas), Google Chrome o Internet Explorer.

4.8. Lenguajes de programación web

Para este proyecto, como ya comenté más arriba, el 80% del código utilizado ha sido escrito en PHP, ya que es un lenguaje con el que tenía más experiencia que otros, y permite hacer páginas webs dinámicas, con contenido a base de datos, de una forma relativamente sencilla. El resto de lenguajes usados han sido HTML5, CSS3, y JavaScript.

También merece especial atención, aunque no forma parte de la estructura de la aplicación, el lenguaje de programación AIML, esencial para los chatbots. En este caso, no lo incluyo como código como tal, ya que lo considero más un lenguaje de almacén de contenidos (datos).

Figura 4.8.1: Lenguajes de programación web utilizados

4.8.1. HTML5

Prácticamente todo el mundo conoce o ha escuchado hablar del lenguaje HTML (HyperText Markup Language). Es la base de la programación web, y sobre éste debería centrarse todo la estructura de la página.

En mi caso, como se explicará más adelante, no lo he usado demasiado, porque mi intención no trabajar sobre una web escrita desde cero, sino sobre una que ya existe, la web de la facultad, por tanto, su uso ha sido casi en exclusiva, para el pop-up que se crea al utilizar el chatbot, así como para crear el iframe con el contenido de la web.

4.8.2. CSS 3

En segundo lugar, y como recomiendan los estándares, para todo lo referente a estilos, se debe usar CSS (Cascading Style Sheets).

Al igual que ocurre con HTML, no he usado este lenguaje en abundancia, excepto para 2 puntos:

- Adaptar el estilo de la web de la facultad al pop-up del chatbot.
- Crear una imagen, también del estilo de formato de la web, para usarse como link al pop-up.

Como se puede observar, su uso no ha sido de muchas líneas, pero sin embargo, lo considero vital a la hora de conseguir una armonía entre mi módulo de chatbot y el estilo ya existente.

4.8.3. PHP 5.6.12

Llegamos al lenguaje sobre el que se sustenta el funcionamiento de la aplicación. Desde la página principal del pop-up, se llama a un archivo PHP, llamado controlador.php, que se encargará de conducir los datos, en función de la entrada, al cliente que considere oportuno según las respuestas obtenidas.

Como se explicará más adelante, se ha decidido modular cada función del sistema, de manera que sea un código mucho más asequible de leer, estudiar o modificar, de la siguiente manera:

- controlador.php: gestiona el recorrido de los datos por cada cliente.
- analizador.php: encargado de transformar oraciones completas, en un array de palabras clave.
- client-bbdd: cliente encargado de conectarse a la base de datos y obtener, de manera rápida, respuesta a preguntas típicas.
- client-bot: cliente necesario para la parte conversacional del usuario con el bot. Gestiona entradas no relacionadas con búsquedas de la web.
- client-solr: cliente encargado de conectar con el servidor Solr, para que devuelva urls a partir del contenido de las webs.

Además de estos ficheros, se han utilizado APIs ya creadas de diferentes fuentes, donde siempre se ha buscado que estuviesen escritos en PHP, para tener que evitar la parte de parseo de código desde otro código. Las APIs utilizadas han sido las siguientes:

- Simple HTML DOM: para la extracción del código de Grampal.
- Program-O: como motor de reconocimiento de ficheros programados con AIML.
- Solr PHP Client: como cliente de conexión al servidor Solr.

4.8.4. JavaScript

JavaScript no ha sido utilizado nada más que en contadas ocasiones.Fue el lenguaje recomendado por el profesor para desarrollar el cliente, ya que era una forma sencilla de implementar una ventana emergente que se pudiera abrir o cerrar según la conveniencia del usuario. Sin embargo, para el resto de la aplicación, debido a que era un lenguaje bastante desconocido para mí, opté por hacerlo en PHP.

Sin embargo, JavaScript, al ser un lenguaje ejecutado desde la máquina del cliente, nos permite crear acciones dinámicas, sin necesidad de hacer una petición al servidor, lo cual resultar muy beneficioso en muchas ocasiones.

En el caso particular del proyecto, JavaScript ha sido utilizado en las siguientes ocasiones:

- Abrir la ventana emergente donde se produce la conversación entre el usuario y el chatbot.
- Especialmente en la integración del chatbot con la interfaz de la web de la facultad. Debía ser capaz de poder controlar, desde el pop-up abierto en una ventana, la ruta de un iframe contenido dentro de la ventana padre, para que cuando se hiciese click sobre un enlace del pop-up, se modificase dicho iframe en vez de la ventana. Esto sólo fue posible hacerse con este lenguaje.

4.8.5. AIML

Como ya dije anteriormente, AIML (Artificial Intelligence Mark-up Language), no ha sido un lenguaje que haya sido utilizado en la creación de la estructura, ni para el control de la aplicación; sin embargo, dado que es el lenguaje principal de los chatbots, creo que merece atención especial en este punto, así como hacer un repaso de su funcionamiento básico.

AIML es un lenguaje de programación basado en la estructura XML, y además es bastante actual, ya que tiene aproximadamente 15-20 años. Desde que se creó, ha sido utilizado como herramienta principal para la creación de diferentes

chatbot, debido por una parte, a su facilidad de uso, y por otra parte, a la diversidad de comunidades que han ido surgiendo, mejorando los primeros chatbots y adaptándolos a diferentes ámbitos.

En cuanto al lenguaje AIML, podemos decir que está basado en XML. Por tanto, tiene una estructura muy determinada compuesta por ciertas etiquetas que servirán para definir las respuestas que nos dará el chatbot ante ciertos patrones de entrada.

A modo de resumen, podemos encontrar las siguientes etiquetas.

- <category>: podemos definirlo como una unidad que contiene al menos una pregunta y una respuesta. Se utilizará una categoría para organizar un módulo entrada-salida.
- <pattern>: es la unidad de entrada al chatbot, o lo que es lo mismo, el patrón que debe reconocer el sistema como pregunta del usuario. Puede contener 3 tipos de contenido:
    - Palabras en lenguaje natural sin importar mayúsculas/minúsculas.
    - Signo "*" (estrella). Sustituye a cualquier secuencia de palabras.
    - Signo "_" (guión bajo). Sustituye una secuencia de una o más palabras, y tiene preferencia sobre los dos anteriores.

    Por tanto, no es el mismo patrón "HOLA", "HOLA*" y "_HOLA*". Esto es necesario tenerlo en cuenta para hacer sinónimos (<srai>), es decir, indicar en el fichero que la segunda y tercera entradas apunten a la primera.

- <template>: cuando un patrón es reconocido, se leerá el contenido de estas etiquetas para proporcionar una respuesta. Puede ser desde una simple cadena, como una respuesta más elaborada, aleatoria o crear una respuesta compuesta o incluso personalizada.
- <random>: esta etiqueta almacena un conjunto de posibles respuestas ante un mismo patrón, de esta forma la respuesta es aleatoria, dando sensación de realismo.
- <that> y <topic>: no han sido utilizadas en el proyecto, pero estas etiquetas dotan de un contexto específico a la categoría. Esto permite diferentes respuestas para una misma pregunta en función del contexto que se especifique.
- <srai>: es utilizado para la recursión, de forma que la respuesta estará formada por la respuesta de varias categorías. Realmente tiene muchos usos, ya que la capacidad de recursividad que tiene, se puede utilizar de

diferente manera para obtener resultados mucho más completos y personalizados. Algunos ejemplos son:

- Reducir formas gramaticales complejas en otras más sencillas.
- Romper una entrada en dos independientes y combinar respuestas.
- Crear sinónimos, para usos como el de "*" y "_", o entre palabras diferentes.
- Correcciones de palabras mal escritas.
- Respuestas condicionales, al estilo de una sentencia "if".

Hay que tener especial cuidado al crear etiquetas <srai>, ya que podemos estar creando inconscientemente un bucle infinito que nos lleve de una categoría a otra sin control.

- <star>: esta etiqueta guarda el contenido de la cadena que sustituye a "*" en una entrada, para poder usarse como respuesta. Para hacer referencia a ella, también se hace a través de <srai><star></srai>. Debido a su uso tan común, se ha sustituido este conjunto de etiquetas por <sr/>.

Estas son las etiquetas más utilizadas. AIML permite otras muchas, que otorgan nuevas funciones al chatbot, como variables que guardan fecha, nombres…

4.9. Resumen

A modo de recopilación de todas las herramientas utilizadas para el desarrollo del proyecto, junto a sus versiones son las siguientes:

- XAMPP 5.6.12
- Nutch 1.9
- Solr 4.10.4
- Program-O 2.4.7
- Lenguajes de programación web
  - HTML5
  - CSS3
  - PHP 5.6.12
  - JavaScript
  - AIML 1.0

La integración de las diferentes aplicaciones ha sido implementada mediante PHP, ya que se ha buscado este lenguaje para las APIS que los manejan, y así evitar problemas de conversión entre ellos.

5. Arquitectura utilizada

En este apartado se describirá la conexión entre los módulos que forman el sistema desde dos perspectivas diferentes. En un primer momento, desde un punto de vista estático, con sus respectivos módulos, elementos y diagramas de clase; y en un segundo plano, la relación dinámica entre sus componentes cuando estos interaccionan.

5.1. Versión estática

5.1.1. Visión general del sistema

A continuación se muestra la arquitectura seguida para el correcto desarrollo de la aplicación.

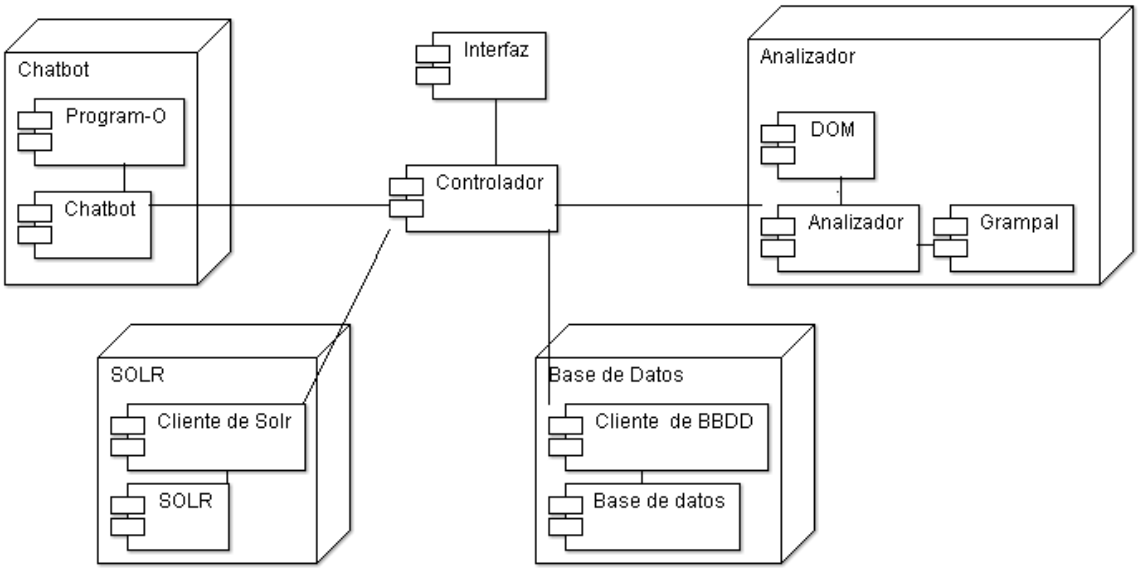

Figura 5.1.1: Diagrama de componentes general de la aplicación

5.1.2. Elementos del sistema

- **Controlador:** módulo central encargado de conectar al resto de los componentes del sistema. Todos los datos pasan por él y éste los va reenviado según es necesario en cada paso.

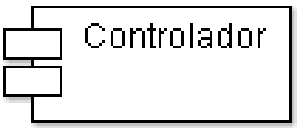

Figura 5.1.2: Diagrama de componentes del controlador

El controlador es el único elemento que está conectado a la intefaz, ya que actúa de puente entre la propia ventana, y el resto de los componentes. Es invocado desde la interfaz (mediante una función PHP) y mediante la llamada a funciones de los otros módulos, irá creando una respuesta que mostrar al usuario.

Su forma de actuar es bastante simple.

- Es invocado
- Llama a una función, y almacena el resultado en una variable $respuesta.
- Según el valor de la variable $respuesta, irá tomando unos caminos u otros, hasta dar con una repuesta final.

El orden de llamada a los módulos es el siguiente, lo que no significa que siempre tenga que realizar todos:

- Llamada al cliente que gestiona el tipo de pregunta (chatbot).
  - client-bot.analizarChatbot();
- Llamada al cliente que conecta con la BBDD (modo buscador).
  - client-bbdd.primeraBusqueda();
- Llamada al cliente que analiza la oración completa.
  - analizador.analizarOracion();
- Llamada al cliente que conecta con la BBDD (modo oración).
  - client-bbdd.ultimaBusqueda();
- Llamada al cliente que conecta con Solr.
  - client-solr.escribirRespuestaSOLR();

- **Analizador**
  - Analizador: se encarga de parsear las oraciones que recibe y extraer los sustantivos que encuentre.
  - DOM: módulo encargado de transformar el código HTML en una estructura de nodos DOM, desde el que obtener las palabras claves, desde la tabla proporcionada por Grampal.
  - Grampal: herramienta de la UAM utilizada como analizador morfológico de oraciones.

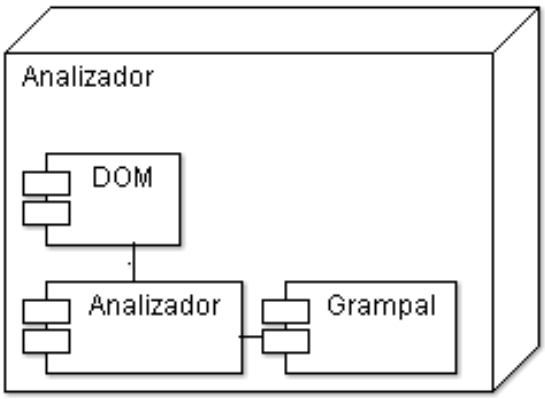

Figura 5.1.3: Diagrama de componentes del analizador

Su funcionamiento requiere de los servicios de una herramienta externa (Grampal), a la que se llama desde el analizador, extrayendo el código HTML, y parseándolo con DOM para extraer los datos necesarios.

Su funcionamiento es el siguiente:

1. Adaptamos entrada del usuario (cambiamos espacios por “+”, quitamos tildes…)
2. Petición get desde el analizador (file_get_contents(url)).
3. Cargamos en DOM el contenido de la web (realmente sólo nos interesa la tabla)
4. Procesamos los nodos que necesitamos (celdas de columnas 0, 1 y 2), y los guardamos en un array.

- **Base de Datos**
  - Base de datos: repositorio que almacena tanto las búsquedas clave como el repositorio de archivos AIML.
  - Cliente de base de datos: encargado de conectarse a la base de datos y obtener respuesta a partir de palabras clave.

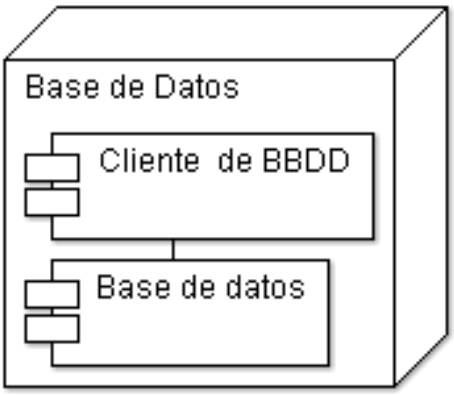

Figura 5.1.4: Diagrama de componentes del módulo de base de datos

La base de datos consta de 3 tablas:

- palabras_clave: guarda las palabras clave que se deben identificar.
- urls_comunes: guarda el conjunto de las urls más comunes de la web.
- relacion: tabla que relaciona las dos tablas anteriores, ya que son del tipo M:N (muchos a muchos)
  - Una palabra clave puede redirigir a varias urls.
  - Una misma url puede tener distintas palabras de origen.

El diagrama quedaría de esta manera.

- **Buscador**
  - Cliente de Solr: cliente encargado de conectar con el servidor Solr, para que devuelva urls a partir del contenido de las webs.
  - Solr: actúa como repositorio de urls y contenido directamente extraído de la web de la facultad. Entrará en juego cuando no se haya podido detectar el patrón de entrada ni se haya detectado una palabra clave registrada.

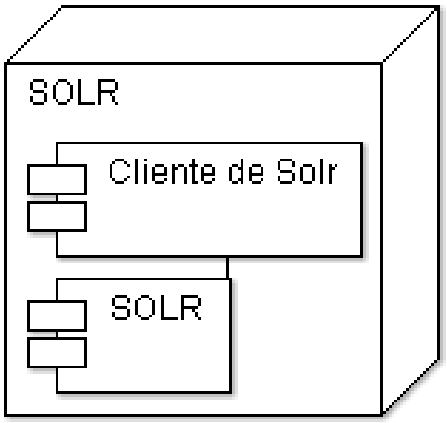

Figura 5.1.5: Diagrama de componentes del módulo de Solr

El funcionamiento del buscador consiste en realizar una petición get al servidor Solr para que nos devuelva el resultado de una consulta SELECT, donde le pasamos como parámetro la palabra clave para que campo “content” (contenido) y nos devuelve una o más urls que tengan dicho contenido en su página.

```
$client = new SolrClient("<ip>", "<puerto>");
$client->setUriPath("<ruta de colección de Solr>");
$client->addParameter("start", <primera fila a devolver>);
$client->addParameter("rows", <filas a devolver>);
$client->addParameter("q", "content:".<palabra clave>));
```

Esto forma una petición del tipo:

http://<ip>:<puerto>/solr/collection1/select?q=content%<palabra_clave>df=url&wt=php&indent=true

http://localhost:8983/solr/collection1/select?q=content%3Aprueba&fl=url&wt=php&indent=true

```
array(
  'responseHeader'=>array(
    'status'=>0,
    'QTime'=>0,
    'params'=>array(
      'fl'=>'url',
      'indent'=>'true',
      'q'=>'content:prueba',
      '_'=>'1441611791352',
      'wt'=>'php')),
  'response'=>array('numFound'=>40,'start'=>0,'docs'=>array(
      array(
        'url'=>'http://informatica.ucm.es/estudiantes'),
      array(
        'url'=>'http://informatica.ucm.es/secretaria'),
      array(
        'url'=>'http://informatica.ucm.es/informatica/convalidaciones-reconocimientos-adaptaciones-y-cambios-de-titulacion'
      array(
        'url'=>'http://informatica.ucm.es/informatica/conferencias-de-posgrado'),
      array(
        'url'=>'http://informatica.ucm.es/estudios/2014-15/licenciatura-ingenieriainformatica-plan-106134'),
      array(
        'url'=>'http://informatica.ucm.es/estudios/2014-15/master-ingenieriadesistemas-plan-604450'),
```

Chatbot

- Chatbot: gestiona entradas no relacionadas con búsquedas de la web, es decir, la parte conversacional del chatbot.
- Program-O (Motor AIML): módulo encargado de la parte conversacional. Lee patrones de entrada y genera salidas configuradas.

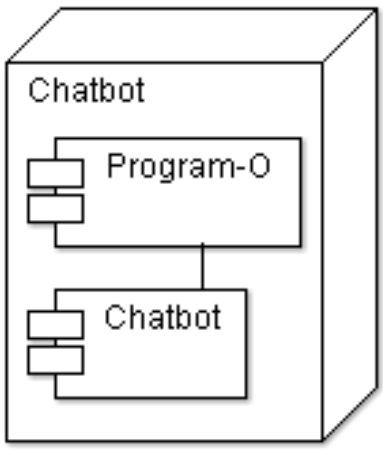

Figura 5.1.6: Diagrama de componentes del chatbot

El módulo del chatbot es un motor de reconocimiento de patrones mediante archivos AIML.

Cuando el usuario lanza su búsqueda, este es el primer módulo en ser llamado, para tener una idea clara de qué tipo de patrón lanza el usuario. El sistema reconoce entre:

- Patrón de conversación: si tiene un patrón reconocido, por ejemplo, un saludo, una pregunta personal, temática etc…
- Patrón de insulto: si reconoce una palabra malsonante, automáticamente devuelve un aviso de responder y cerrar la ventana.
- Patrón de despedida: si reconoce una palabra reconocida como despedida, actuará de la misma forma que el patrón de insulto, pero respondiendo educadamente.
- Por último, si el patrón no es reconocido, se enviará al siguiente módulo para analizar si corresponde con una palabra clave etc…

- **Interfaz**
  - Interfaz: será el módulo que interactuará con el usuario que se conecte a la web y necesite el uso del chatbot.

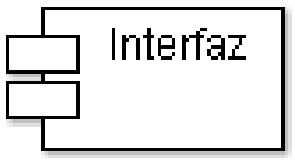

Figura 5.1.7: Diagrama de componentes de la interfaz

Está compuesto por una ventana emergente, llamada al hacer click sobre la imagen que anuncia la existencia del chatbot.

El usuario interactuará con el formulario de la ventana emergente, donde irá escribiendo sus dudas en un input, y el chatbot le irá proporcionando las respuestas que considere más válidas, así como los links que llevarán a dichas urls.

5.2. Visión dinámica

A continuación se muestra el diagrama de actividad que relaciona de forma dinámica todos los módulos, mostrándose de forma esquematizada, los diferentes caminos que pueden tomar los datos, desde que el usuario envía los datos desde el formulario, hasta que finaliza en una de las 4 posibles salidas.

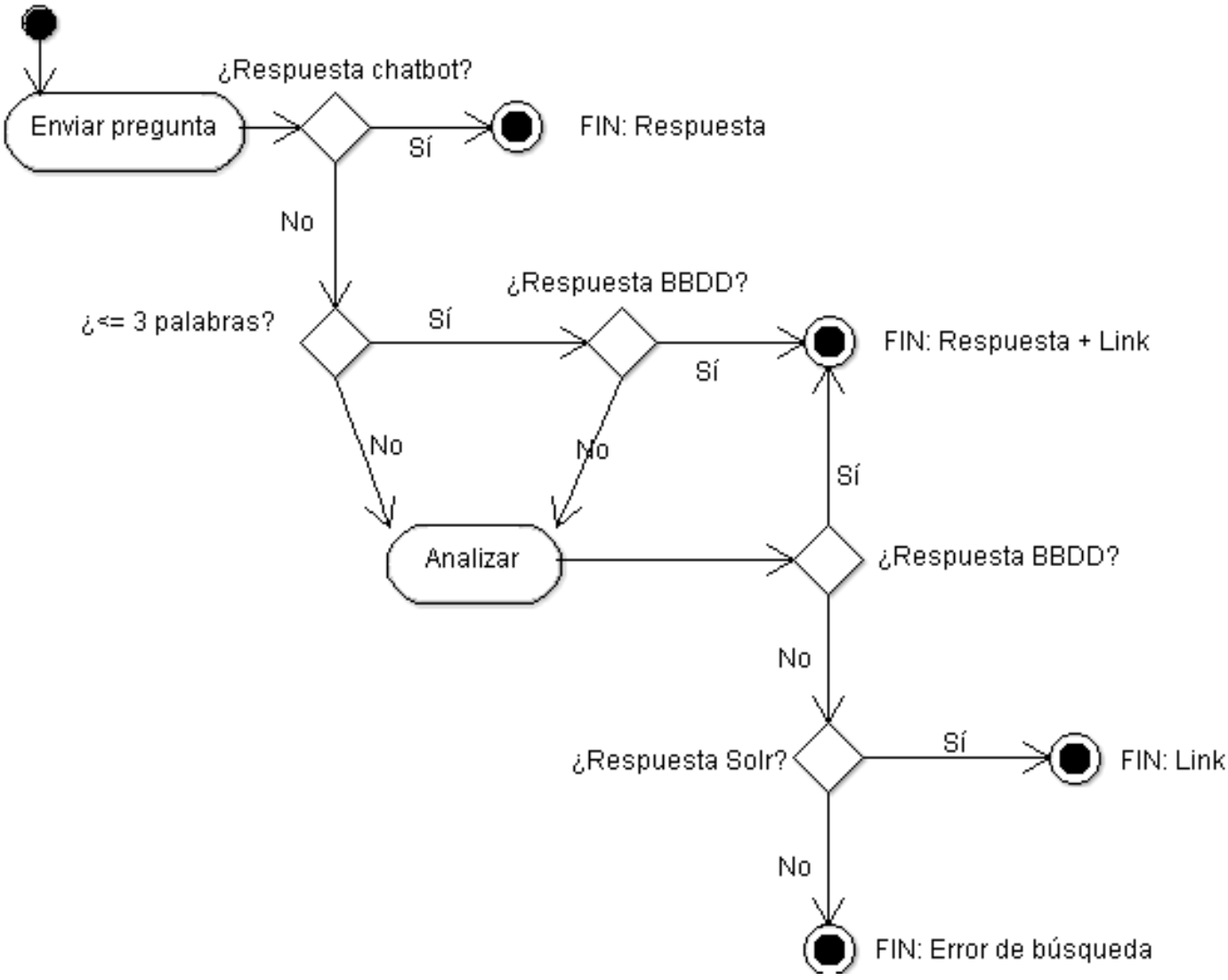

Figura 5.2.1: Diagrama de flujo general

El esquema anterior muestra el siguiente flujo de actividad:

1. Se analiza la palabra y se establece la variable $respuesta a true o false según si el Chatbot (AIML) tiene respuesta.

2a. Si se obtiene respuesta, se muestra por pantalla. **FIN.**

2b. Si no se obtiene respuesta, se analiza el número de palabras de la entrada.

3a. Si tiene 3 palabras o menos, se manda a la BBDD en busca de respuesta.

4a. Si se obtiene respuesta de la BBDD, se muestra por pantalla. **FIN.**
4b. Si no hay respuesta, se envía al analizador para extraer sustantivos.

3b. Si tiene más de 3 palabras, se envía al analizador para extraer sustantivos.

4a. Se manda de nuevo a la BBDD en busca de respuesta.

5a. Si se obtiene respuesta de la BBDD, se muestra por pantalla. **FIN.**

5b. Si no hay respuesta, se manda a Solr.

6a. Si Solr obtiene respuesta, se muestra por pantalla. **FIN**

6b. Si Solr no obtiene respuesta, se muestra el error por pantalla. **FIN.**

Como ya se puntualizó en el apartado 3.1, donde describíamos la funcionalidad del sistema, ahora usaremos algunos de esos escenarios para especificar, con más detalle, el flujo de información mediante diagramas de secuencia.

En concreto se han escogido 4 de estos escenarios, que abarcan prácticamente la totalidad de los supuestos mencionados más arriba.

- **Escenario 3: Conversación de temas básicos**

Pregunta: Si no es mucha indiscreción, ¿me puedes decir la *edad* que tienes?
Lucy: Soy de julio del 2015, o sea que tengo poco más de 1 mes.

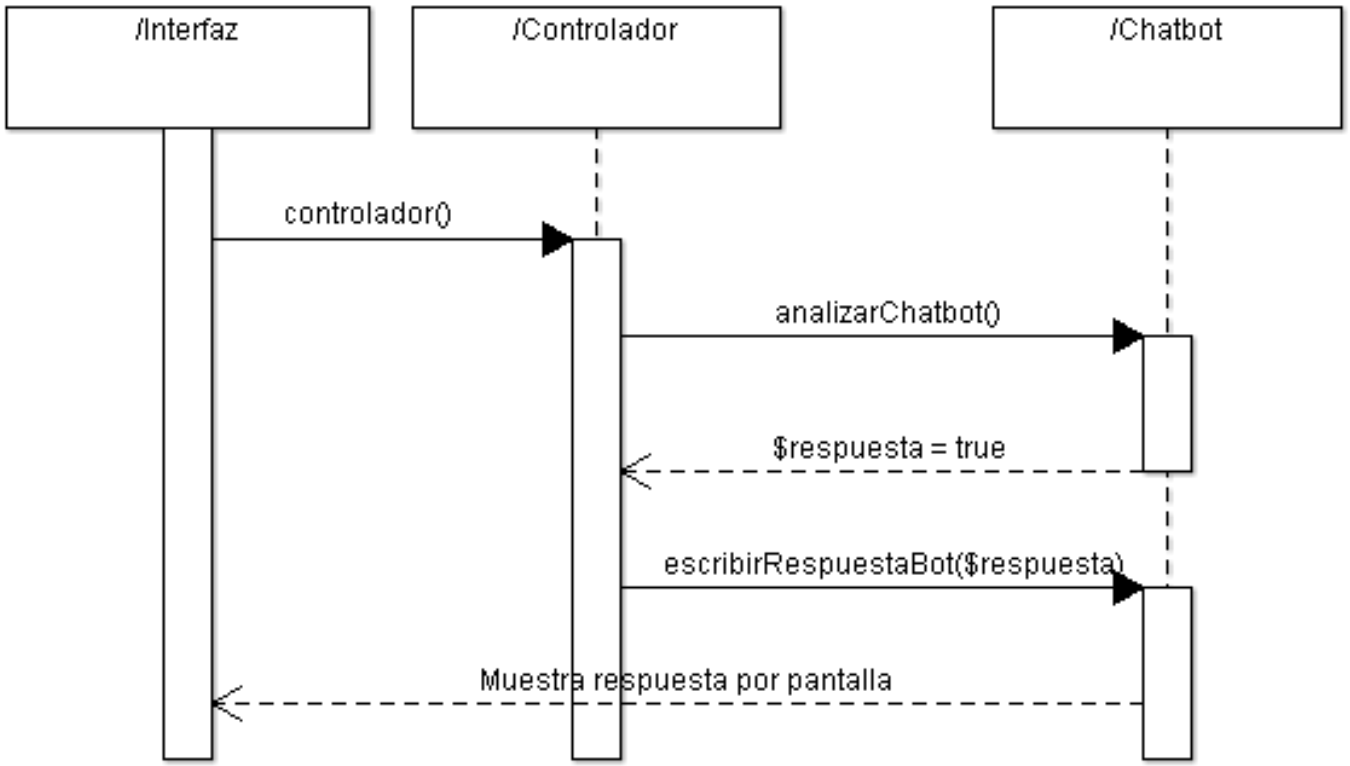

Figura 5.2.2: Diagrama de secuencia "Escenario 3"

En este caso, una vez que el Chatbot reconoce el patrón entre los archivos AIML (_ EDAD *), devuelve $respuesta = true. Esto hace que se llame directamente al procedimiento encargado de mostrarlo por pantalla, finalizando el recorrido.

- **Escenario 4: Palabra clave reconocida**

Pregunta: junta directiva
Lucy: Puedes hacer click aquí para encontrar información sobre la junta de facultad.

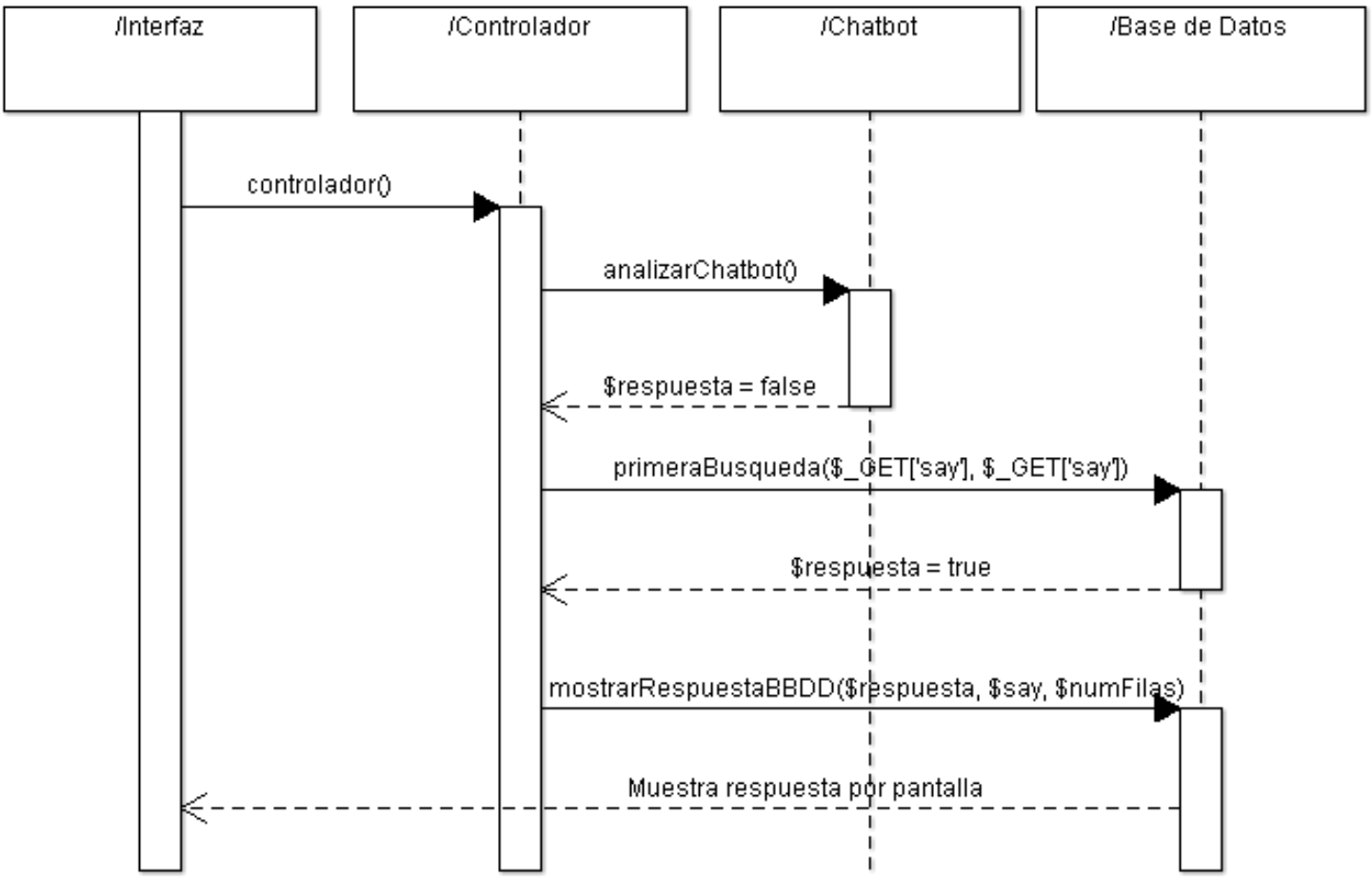

Figura 5.2.3: Diagrama de secuencia "Escenario 4"

En este caso, "junta directiva", no está reconocido como patrón de AIML, por lo que se devuelve $respuesta = false y el controlador pasa a consultar en la base de datos si dicha entrada existe en la tabla "palabras_clave", ya que dicha entrada tiene 3 palabras o menos. Como en este caso es afirmativo, devuelve $respuesta = true, pasándose a llamar al procedimiento que mostrará por pantalla la respuesta almacenada en la tabla "urls_comunes", así como el link que conducirá a ella.

- **Escenario 5: Oración con palabras clave reconocidas**
  Pregunta: ¿Quién es el actual decano?
  Lucy: Desde este enlace podrás encontrar información referente al decano de la facultad: Daniel Mozos Muñoz.

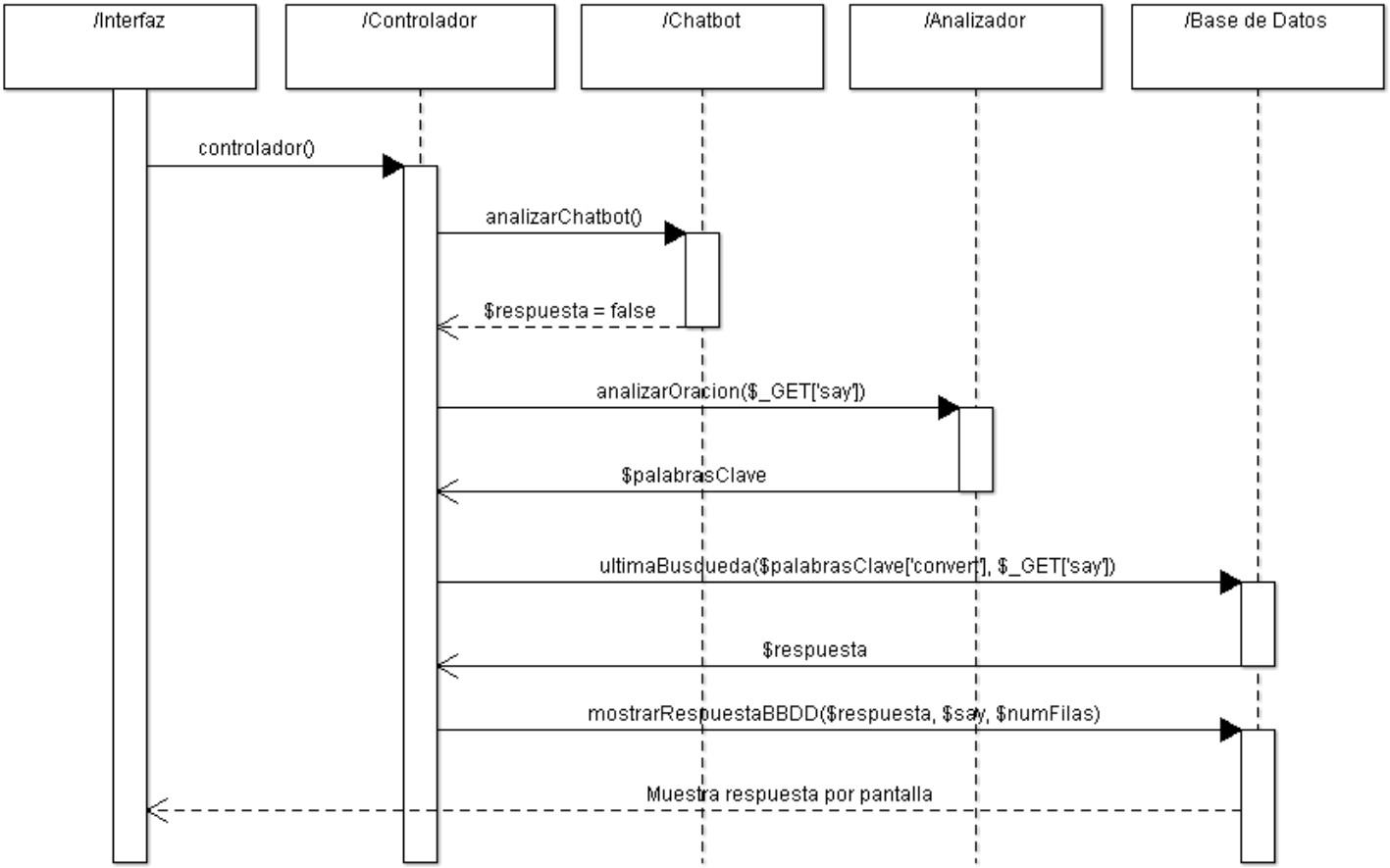

Figura 5.2.4: Diagrama de secuencia "Escenario 5"

Parecido al caso anterior. Sin embargo, en este caso, la entrada tiene más de 3 palabras, por lo que ni siquiera se consulta en la base de datos (no hay palabras clave de más de 3 palabras). Esta entrada pasa a enviarse al analizador para que extraiga los sustantivos que encuentre: "decano". Ahora, esta palabra es enviada de nuevo a la base de datos, donde esta vez sí es reconocida como palabra clave, proporcionándose la misma salida que en el caso anterior.

- **Escenario 6: Palabras claves no reconocidas, pero sí encontradas con Solr**
- **Pregunta:** ¿En qué consiste el PGP Party?
  Lucy: He encontrado algunas urls para tu búsqueda:
  http://informatica.ucm.es/facultad
  http://informatica.ucm.es/pgp-party-2014-12-04
  http://informatica.ucm.es/eventos

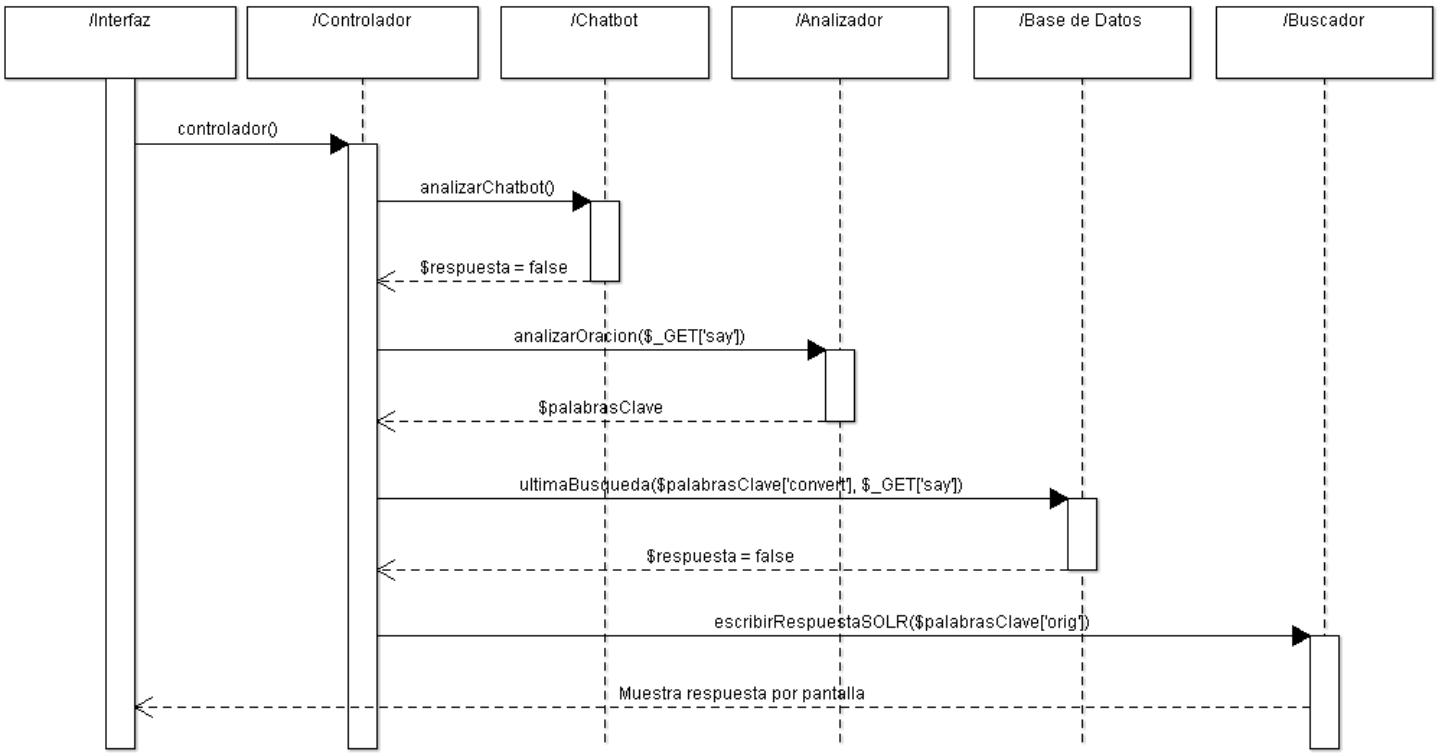

Figura 5.2.5: Diagrama de secuencia "Escenario 6"

En el último caso, obtenemos el recorrido más largo, ya que en la segunda iteración a la base de datos sigue sin encontrar ninguna palabra clave que se adapte. Por tanto, se manda el resultado a Solr para que analice el contenido indexado de todas las web almacenadas en el repositorio, y muestre las que más respuestas relacionadas encuentre.

6. Evolución del proyecto

6.1. Sprint 1: Análisis y asentamiento (15/06/2015)

El primer paso fue un primer análisis del proyecto propuesto en general, así como el asentamiento en un entorno de trabajo que ayudase a su realización.

Este sprint fue uno de los más largos, y aunque no se considera un prototipo acabado como tal, merece la pena nombrarlo como hito conseguido en el principio del proyecto.

Por ello, debido al desconocimiento de las herramientas a utilizar (en un principio Lucene, Nutch y Solr), hubo que recopilar la mayor cantidad posible de documentación y ejemplos de internet, así como la ayuda proporcionada por el profesor.

Otro punto que se analizó e investigó desde un principio fue el del funcionamiento de los Chatbot, especialmente desde las webs de ALICE y Pandorabots, desde donde se recopiló información sobre cómo piensa un chatbot y especialmente sobre el tipo de archivos AIML, basados en XML. Una vez acabada esta fase de análisis, se empezó realmente a construir el proyecto.

La primera decisión fue la del Sistema Operativo a utilizar. Todas las herramientas eran válidas tanto para Windows como para Linux, pero dado que como tanto la idea que se planeó, que era la de un servidor, como por propia recomendación del profesor, finalmente se optó por utilizar Linux, en su distribución CentOS 5 de 64 bits.

Esto se realizó sobre una máquina virtual utilizando el software VirtualBox 4.3, para simular que era un servidor en una máquina diferente independiente del cliente que lo utilizase.

Antes de instalar software específico del proyecto, se procedió a realizar labores preliminares, como la configuración de la red para acceder desde fuera por IP o la propia salida de la máquina a internet, ya que sería necesario para poder indexar la web de la facultad. Otro software que se previó necesario, fue el servidor XAMPP, necesario para servir páginas web mediante Apache.

6.2. Sprint 2: Indexación y primer buscador (01/07/2015)

Dejando ya análisis y acciones preliminares, el objetivo marcado era indexar la web de la facultad mediante el crawler (Nutch) y almacenar el contenido recogido en Solr.

Para ello, se procedió a la instalación del software necesario para la indexación:

- Instalación de Apache Ant (para construir Nutch).
- Instalación de Apache Nutch 1.1.
- Configuración de variables de entorno (JAVA_HOME y ANT_HOME).
- Instalación de Apache Solr.
- Arranque de Solr y prueba de funcionamiento: http://localhost:8983/solr/

Después de esto, la configuración de Nutch era más o menos sencilla:

- Archivo nutch-site.xml, donde se debían configurar algunos parámetros.
- Archivo seed.txt: donde se incluía la url base a indexar.

A la hora de lanzar el crawleo, llegó el primer problema: era una versión (1.1) demasiado antigua para utilizarla conjuntamente con Solr. Por tanto, hubo que actualizar a la versión 1.9 que sí admitía esta funcionalidad.

Una vez actualizada la versión, lo que conllevó realizar toda la configuración de nuevo, no parecía una tarea demasiado difícil, pero sin embargo por alguna razón desconocida no indexaba nada. Simplemente devolvía que había indexado 0 webs.

Esto supuso bastante tiempo para solucionar el error, hasta el punto de que al no encontrarse, se optó por reinstalar todo desde el principio, incluido el Sistema Operativo por si el problema venía de otro lugar. No hubo suerte.

Finalmente, y tras seguir buscando el error, resultó que todo estaba debido a que el proyecto se realizó desde dos entornos diferentes, y casualmente siempre se lanzaba el crawleo desde uno de ellos, que estaba detrás de un proxy, y que bloqueaba la correcta actuación de Nutch.

Solventado el error, y haciendo un primer crawleo de prueba que resultó finalmente exitoso, se procedió a realizar en total 2 intentos más, en cada uno ajustando un poco más las opciones de Nutch:

- Primer intento: una iteración sin filtros.
- Segundo intento: una iteración con filtro para url propias de la facultad: +^http://informática.ucm.es/([a-z0-9]*\.)*
- Indexado definitivo: cinco iteraciones con el mismo filtro del apartado anterior.

Al realizar las cinco iteraciones el tiempo fue considerablemente mayor, pero finalmente acabó correctamente con un total de 1055 urls indexadas.

Llegó la hora de lanzar Solr y realizar diferentes búsquedas de prueba.

En el ejemplo de abajo con los siguientes parámetros:

- Buscar registros con url = facultad.
- Devolver sólo las url (direcciones web).
- Mostrar sólo 10 resultados.

A modo ilustrativo, sería similar a una sentencia SELECT contra una Base de Datos del estilo:

SELECT url FROM tabla WHERE url = 'facultad' AND rownum <= 10;

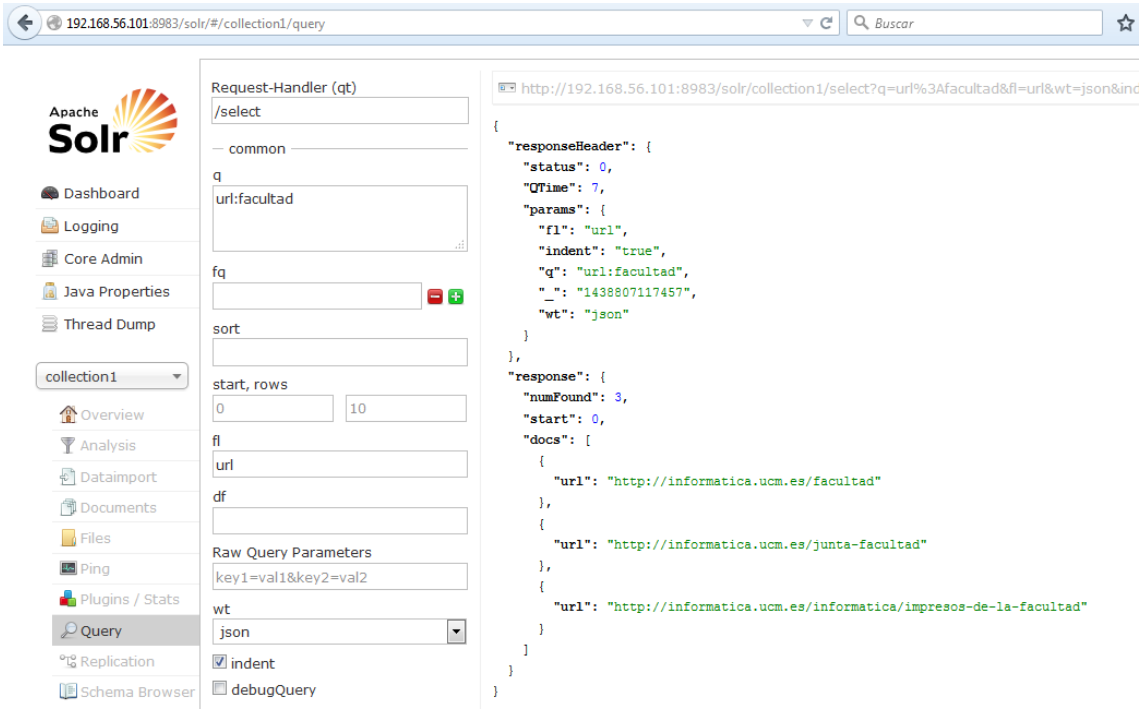

Figura 6.2.1: Ejemplo SELECT desde Solr

6.3. Sprint 3: Buscador en JavaScript (22/07/2015)

Teniendo ya la web capturada, el siguiente objetivo consistía en realizar un formulario básico en HTML que se conectase con Solr mediante un cliente que mandase peticiones.

Para ello, en internet hay multitud de APIs disponibles dependiendo del lenguaje que queramos utilizar.

Por conocimiento, mi primera idea fue Java, y esto suponía la creación de un applet que integraríamos en la web. Sin embargo, el profesor me recomendó que no eligiese esta opción, especialmente debido a problemas para ejecutar programas Java en navegadores.

Su opción recomendada fue JavaScript, que por desgracia no era demasiado familiar para mí, y esto me llevó a probar varias APIs, que a su vez requerían otras librerías, como es el caso de JQuery o Node.js, también con las que no tenía casi ninguna experiencia.

Finalmente conseguí realizar búsquedas en Solr a través de Node.js, pero sólo mediante consola de comandos mostrando objetos JSON, pero por más que lo intenté, no conseguí llevarlo a una página web que pudiese ser mostrada por un navegador.

Llegado a este punto, tenía 2 opciones:

- Aprender JavaScript y Node.js lo suficiente para poder seguir desde este punto hasta el final del proyecto con esta tecnología, e ir aprendiendo según se planteasen problemas.
- Redirigir el proyecto hacia otro lenguaje.

Haciendo un análisis de la segunda posibilidad, tenía que encontrar un lenguaje que no tuviese los problemas de Java, es decir, que no fuese un lenguaje externo a la programación web, y que no fuese tan desconocido para mí como JavaScript; esto me llevó a PHP.

6.4. Sprint 4: Buscador en PHP (27/07/2015)

Al igual que ocurría con Java o JavaScript, en internet había diversas APIs disponibles como cliente de Solr, pero dado que la única utilidad que se buscaba era la de hacer consultas tipo SELECT sobre las url indexadas, la idea era no buscar nada más complejo que lo necesario, y tras probar varias de ellas, finalmente me decanté por un cliente muy simple que consistía en un único archivo PHP con una clase que implementaba justo lo que necesitaba y que no requería de más herramientas.

El proceso fue bastante simple, y de esta forma se llegó a conseguir el primer prototipo funcional, que consistía en un cuadro de texto con un botón de búsqueda.

Esto conectaba con el cliente, que llamaba a Solr, y éste devolvía un array PHP con las urls encontradas cuando el contenido (CONTENT) de la página encontraba dicha palabra.

Figura 6.4.1: Ejemplo respuesta proporcionada por buscador PHP

6.5. Sprint 5: Mejora de interfaz. Nace Lucy (28/07/2015)

Este sprint no era estrictamente necesario hacerlo en este punto, ya que no era indispensable para que la aplicación funcionase, pero dado el avance que se tenía hasta ahora, y por otra parte debido a que era una parte del proyecto más ligera y dinámica, se optó por darle una nueva imagen a la interfaz del usuario.

Para ello se diseñó un avatar desde la web "http://avachara.com", con aspecto de mujer. Después se añadió dicha imagen a la interfaz que ya había creada de los puntos anteriores. Y así nació Lucy.

Otra mejora que se implementó fue la de dotarla de movimiento, creando una animación que hacía que moviese la boca cada vez que se mostraba por pantalla la respuesta dada.

Figura 6.5.1: Interfaz de Lucy

6.6. Sprint 6: Primera conversación (01/07/2015)

Hasta ahora sólo se había conseguido que el chatbot se limitase a mostrar los resultados de Solr tal cual se recuperaban del servidor, pero una de las características de un chatbot, es precisamente que parezca que detrás hay una persona real, que pueda dar respuestas coherentes, y que se pueda entablar una mínima conversación.

Aquí entró de nuevo la parte de análisis del Sprint 1, donde se había recopilado bastante información sobre el funcionamiento de los chatbot, y de cómo leen los archivos AIML, que en esencia, consiste en reconocer un patrón (palabra o conjunto de palabras), y buscar una asociación que corresponderá con la respuesta o respuestas (a modo aleatorio) que nos dará.

Para ello, en un principio se pensó en la idea de utilizar un servidor de chatbot online, donde se creaba el robot desde la web (se usó Pandorabots), y después mediante una clave de aplicación y de chatbot, se le llamaba desde la aplicación web.

Esto dio lugar a varios problemas. La web se cayó al día siguiente, y eso podría pasar en cualquier otro momento, por tanto, había una dependencia absoluta de un servidor ajeno a nosotros, aparte de que si algún día, esa web dejara de existir, podríamos perder todos los datos, y habría que re-implementar de nuevo parte de la aplicación.

Esto dio lugar a investigar más a fondo la web de ALICE, una de las más importantes dentro del mundo de los chatbots, y desde donde encontré diversos proyectos para la creación de un motor que leyese archivos AIML y que me permitiese tener mi propio chatbot alojado en mi servidor.

Volviendo a buscar por lenguaje PHP, di con Program-O, un motor que utilizaba PHP y MySQL para leer y procesar archivos AIML. Además contaba con una instalación muy intuitiva, con un panel de administración también muy sencillo y que me permitía tener el Chatbot en mi servidor.

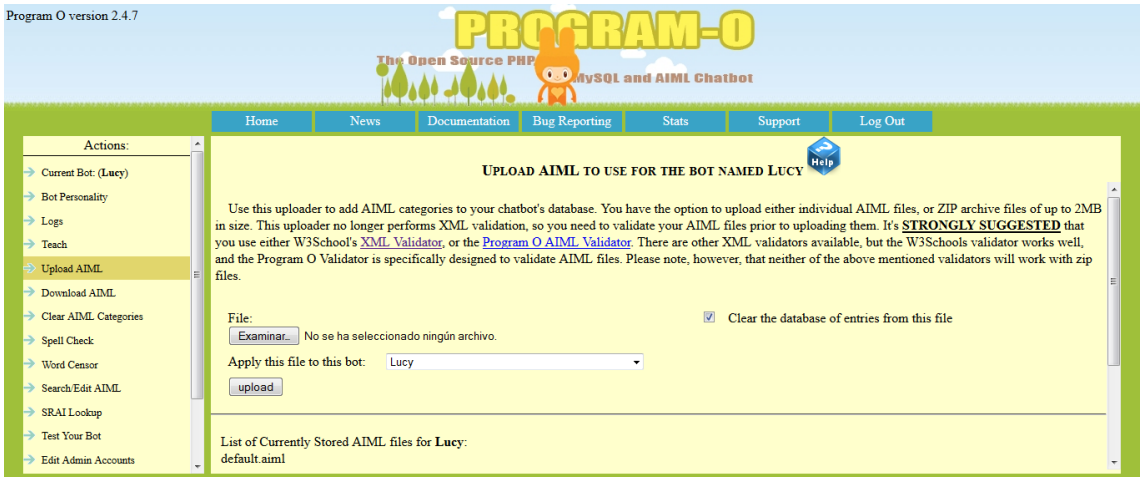

Figura 6.6.1: Ejemplo cargar archivos AIML desde Program-O

Por suerte, también llevaba incluido un cliente que leía desde el motor las peticiones que le mandabas, por tanto, fue tarea sencilla reutilizar dicho código para adaptarlo a mi página web, y que desde el formulario se llamase a dicho cliente.

Como base para la creación de archivos AIML, encontré también en la web de A.L.I.C.E. ejemplos de conjuntos de ficheros para simular el chatbot. Uno de ellos era Sara, que estaba implementado en español. Después, simplemente tuve que subir los archivos al motor Program-O, y Lucy ya los reconocía.

Figura 6.6.2: Primera integración web Lucy

6.7. Sprint 7: Cohesión de clientes SOLR y Chatbot (02/08/2015)

En este punto se había conseguido que Lucy supiera realizar 2 acciones, pero por separado, es decir, para implementar la parte de Solr, había que dejar en comentario la parte de chatbot y viceversa, ya que aún no era capaz de diferenciar cuándo debía responder con oraciones con sentido y cuándo devolver urls.

En este sprint se solucionó este problema. La idea era que el chatbot recibiese una entrada; si ésta era reconocida como patrón, significaba que tenía una respuesta apropiada, y respondía según lo que le enviaba el motor; si por el contrario no era reconocido, se pasaba a invocar al cliente de Solr, para que devolviese las urls que cumpliesen dicha búsqueda. Eso sí, hasta aquí, sólo se podía procesar una palabra por búsqueda.

Al llegar a este punto se decidió hacer una mejora debido a que las búsquedas en Solr no eran demasiado precisas.

Hasta ahora se buscaba por el contenido, es decir, si buscabas “grado”, te devolvía las páginas en las que en alguna parte de la página pusiese “grado”. Eso significaba que podría haber multitud de páginas que contuviesen dicha palabra, pero quizás, la más importante, o la más probable que el usuario quisiese buscar, eran precisamente “los grados ofertados por la facultad”, y quizás podía darse la casualidad de que esa página no apareciese entre las 10 que Solr devolvía.

Para implementar esta mejora, se diseñó un sistema doble llamada a Solr:

- La primera búsqueda era sobre la propia url, lo que filtraba muchas páginas, ya que solo se mostrarían, en el ejemplo de antes, las que contuviesen la palabra “grado” en su dirección web.
- Si el número de páginas respondidas era menor a 10, entonces se hacía una segunda búsqueda por contenido, como estaba hecho al principio, y se devolvía las que encontrase hasta 10 o menos, según el número de respuestas.

Una vez comprobada esta mejora, se observó que se devolvían páginas más coherentes debido a que los programadores de una página web, suelen llamar a la página según un nombre que la identifique, y que tiene más valor que una palabra que aparezca en el contenido, sin embargo, más tarde ya no sería necesaria, gracias a la mejora del sprint 9.

6.8. Sprint 8: Lucy analiza oraciones completas (06/08/2015)

El siguiente objetivo era que no sólo fuese capaz de analizar palabras sueltas. Lo normal es que un usuario que utilice el buscador escriba algo como "Quiero conocer los horarios", y el chatbot debe ser capaz de extraer sólo la información relevante.

Por tanto era necesario eliminar palabras que no fuesen necesarias para realizar búsquedas, y en general, los núcleos de las oraciones están compuestos por sustantivos y verbos.

Si analizamos sustantivos y verbos de oraciones, se puede apreciar que realmente es más significativo un sustantivo que un verbo, pero en un principio, la idea era extraer ambos, y más tarde, estudiar la relevancia de los verbos.

Para esta tarea, era necesario un analizador morfológico, que nos devolviese la categoría gramatical de la palabra, y así poder eliminar el resto. Por ello, se buscaron webs en internet que ofrecieran estos servicios, y finalmente, debido en gran medida a que se podía extraer información de ella gracias a peticiones GET, se decidió utilizar GRAMPAL.

Figura 6.8.1: Ejemplo de la tabla devuelta por Grampal

Para recuperar la información, dado que la página muestra los resultados en una tabla, se utilizó DOM. De esta forma, se recuperaba el código HTML, se almacenaba en una estructura DOM, y se localizaba el nodo "table". Luego simplemente había que seguir sus nodos hijos "td" para acceder a las celdas, y

según si la celda mostraba "N" (Nombre) o uno diferente, se crearon 2 arrays que almacenase las palabras que cumplían esto, para después poder lanzarse al cliente que lo necesitase (Solr o Chatbot):

- En el primer array, se almacenaba la palabra tal cual el usuario la había escrito. Esto era útil para llamadas a Solr, donde es importante que se mantenga la palabra original, ya que se buscará en el contenido de la web.
- En el segundo array, que fue creado tiempo después, se almacenaba la palabra en masculino singular (tal como la transforma Grampal). Esto es debido a la mejora que se hizo sobre búsquedas en la base de datos, que viene explicado más adelante. Era mucho más sencillo almacenar en la base de datos "decano", que no "decano", "decana", "decanos" y "decanas".

Después, cada cliente sabría cómo tratar dicho array. A rasgos generales, recorrían el array de palabras buscando resultados. En el caso de las búsquedas en base de datos, se mostraban todos los resultados obtenidos (ya que suelen ser pocos y concretos), y en el caso del cliente Solr, se hacía una ordenación de las urls encontradas por cada palabra clave, y se devolvía la que mayor moda obtuviese.

6.9. Sprint 9: Almacén de búsquedas típicas (15/08/2015)

Esta mejor vino como resultado de un principio que había aprendido hacía un tiempo en otra asignatura, y era la del principio de Pareto (regla del 80-20).

¿En qué consistía? Bueno, partiendo que el principio de Pareto consiste en el hecho de pensar, que el 80% de las búsquedas que se realizarían en el chatbot estaría englobado dentro de sólo un 20% de las búsquedas totales que se pueden realizar.

Esto significa que hay un cierto número de búsquedas (20%), que se repetirán la mayoría de las veces (un 80% aproximadamente). La idea era recoger este 20%, almacenarlo en una tabla de nuestra BBDD y si se encuentra la palabra apropiada, directamente devolver la url y una respuesta. Con esto se conseguiría, especialmente, más eficiencia, ya que evita tener que hacer una búsqueda mucho más general, pudiendo acotarse esta en un gran porcentaje de los casos.

Para ello el método utilizado fue el siguiente.

- Se realizó en Solr una búsqueda de toda las url indexadas. Esto dio un total de 1055 direcciones diferentes. La idea a partir de ahora, era buscar el 20% de direcciones más frecuentes, lo que suponía unas 200-250.
- El primer filtro aplicado fue quitar todas las url que perteneciesen al curso anterior 2014/2015, ya que se encontraban duplicadas con la misma url, pero en el curso actual 2015/2016. Esto eliminó unos 400 resultados.
- El segundo filtró consistió en quitar el detalle de cada estudio ofertado y dejar sólo el primer nivel, ya que por cada uno de los estudios se indexaban alrededor de 30 urls (información, estructura, horarios etc…). La idea era dejar sólo una url por cada uno. En total se eliminaron otros 400 resultados, quedando solamente unos 250.
- Con esto ya habíamos entrado en el porcentaje requerido, pero observando todas las urls resultantes, se observó que había otras que eran menos probables (por ejemplo todas las direcciones que partían de /grupos/).
- Tras aplicar este último filtro, el resultado obtenido de direcciones más importantes resultó ser de unas 175, lo cual era más eficiente de lo buscado.
- Por último, se fueron eliminando algunas que en el desarrollo del proyecto se observó que no tenía demasiada trascendencia. En el momento de finalización del proyecto, el número de urls almacenadas en la base de datos fue de 88.

Cabe destacar que tras aplicar este filtro no significaba que el resto de urls dejasen de estar indexadas; simplemente no estarían entre las más probables.

Llegados a este punto quedaría la segunda parte de este sprint, consistente en crear una tabla en la BBDD en la que se almacenase la url, las palabras que deberían conducir a dicha url, y la respuesta que daría el chatbot. Este paso era especialmente largo ya que había que había que almacenar en la BBDD todos los registros y para cada uno buscar las palabras más relevantes. Eso significaba que el número de registros en la tabla sería superior a las 88 urls.

Finalmente se implementó con 3 tablas: una de palabras, una de urls, y una que relacionaba ambas, ya que se trataba de una relación M:N., es decir, una palabra puede generar varias url y una url puede venir de varias palabras.

6.10. Sprint 10: Enseñando a Lucy (28/08/2015)

Este sprint podía durar días, meses o incluso años, ya que el mantenimiento típico de un chatbot consiste, entre otros puntos, precisamente en que se vaya actualizando, y aprendiendo para que nos dé la impresión de que estamos hablando realmente con una persona.

Por ello, dado que el final del proyecto estaba previsto para el día 31 de agosto, se decidió llevar este sprint hasta el día 28, para dejar los 3 últimos días para implementar la integración con la web, solucionar errores que se observasen, y en general, cualquier otra situación que pudiese darse.

Por tanto, de aquí en adelante el objetivo era ampliar todo lo posible los archivos AIML, simular conversaciones y búsquedas y, en general, realizar todos los casos posibles que diese tiempo, para llegar a la fecha con una base que pudiese atender la mayoría de las búsquedas realizadas sobre la web de la facultad. Dicho de otra forma, la idea era que Lucy aprendiese todo lo posible y fuese capaz de defenderse en todas las situaciones posibles.

6.11. Sprint 11: Integración con web de la facultad (31/08/2015)

Llegados a este punto, quedaba el último paso, y era el de realizar una correcta integración entre la interfaz de nuestro chatbot y la web de la facultad.

Esta tarea se realizó usando un iframe, un pop-up y código JavaScript.

Primero, se creó una página html en blanco que tenía un único iframe que ocupaba todo su contenido. Este iframe llamaba a la web de la facultad. Sobre este iframe, se creó una imagen por encima, basándose en los estilos de la facultad, que tapaba el antiguo buscador, y dejaba un link que, si pinchábamos sobre él, nos habría un pop-up donde podríamos realizar nuestras búsquedas.

Figura 6.11.1: Ejemplo integración total de Lucy con la web de la facultad

Figura 6.11.2: Ejemplo del lanzamiento del pop-up de Lucy

Aquí se presentó una dificultad que se pudo resolver utilizando lenguaje JavaScript. La idea era que, cuando escribiésemos una consulta que nos devolviese un link, pudiésemos hacer click sobre él, de manera que nos modificase el contenido del iframe de la ventana padre. La dificultad fue primero crear la referencia a la ventana padre, y en segundo lugar, no queríamos que nos modificase la web de su padre, sino del iframe, ya que la página apuntaba a localhost, y era el iframe el que debía apuntar a la nueva url linkada. El resultado fue el esperado.

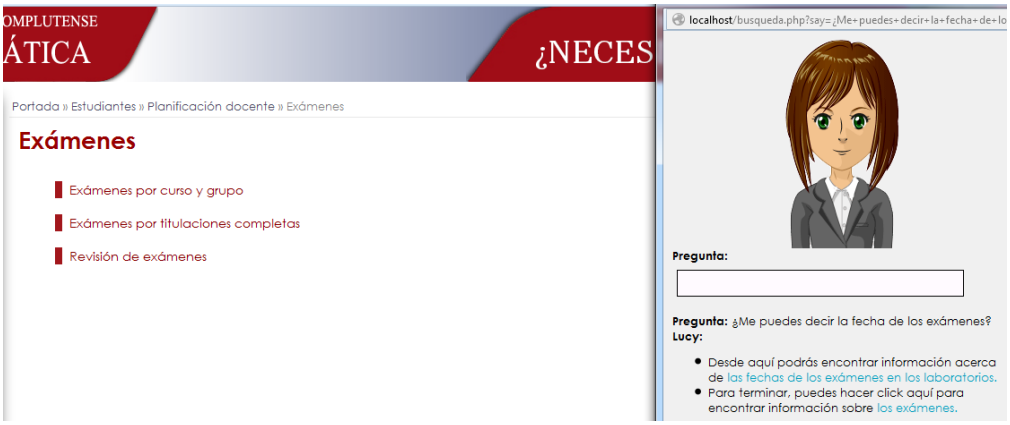

Figura 6.11.3: Ejemplo del funcionamiento final de Lucy ante una búsqueda

7. Instalación y ejecución

Para proceder a la implantación en el sistema de explotación, procederemos de la siguiente manera:

7.1. Instalación de software necesario

Debemos desplegar las siguientes aplicaciones:

- Nutch: para futuras indexaciones.
- Solr: indispensable para utilizar el chatbot.

En el caso de Solr, deberá estar siempre ejecutándose en background, ya que debe estar escuchando peticiones en el puerto que deseemos, en nuestro caso hemos utilizado el puerto por defecto: 8983.

El resto de funcionalidades no necesitan de instalación previa, ya que todas las clases necesarias, así como la interfaz y los clientes, vienen implementados dentro del directorio /lucy/. Para desplegarlo simplemente es necesario llevarnos este directorio al servidor y hacer referencia desde la propia interfaz de la web de la facultad. En mi caso lo he hecho mediante la línea:

7.2. Ejecución de la aplicación

Por último, voy a proceder a crear un ejemplo de una búsqueda que, a modo de ejemplo, podría ser el que yo mismo realizase ahora mismo para buscar información acerca de las fechas de entra del Trabajo de Fin de Grado.

El ejemplo incluye un caso completo de ejecución.

1. El usuario entra a la web de la facultad.
2. Decide utilizar a Lucy para encontrar más rápidamente la información.
3. Lucy le da la respuesta de 2 urls, de las cuales la segunda es la que se busca.
4. El usuario hace click y la página principal se redirige.
5. El usuario agradece a Lucy su ayuda.
6. Lucy se despide, y automáticamente cierra la ventana emergente para dejar al usuario seguir trabajando.

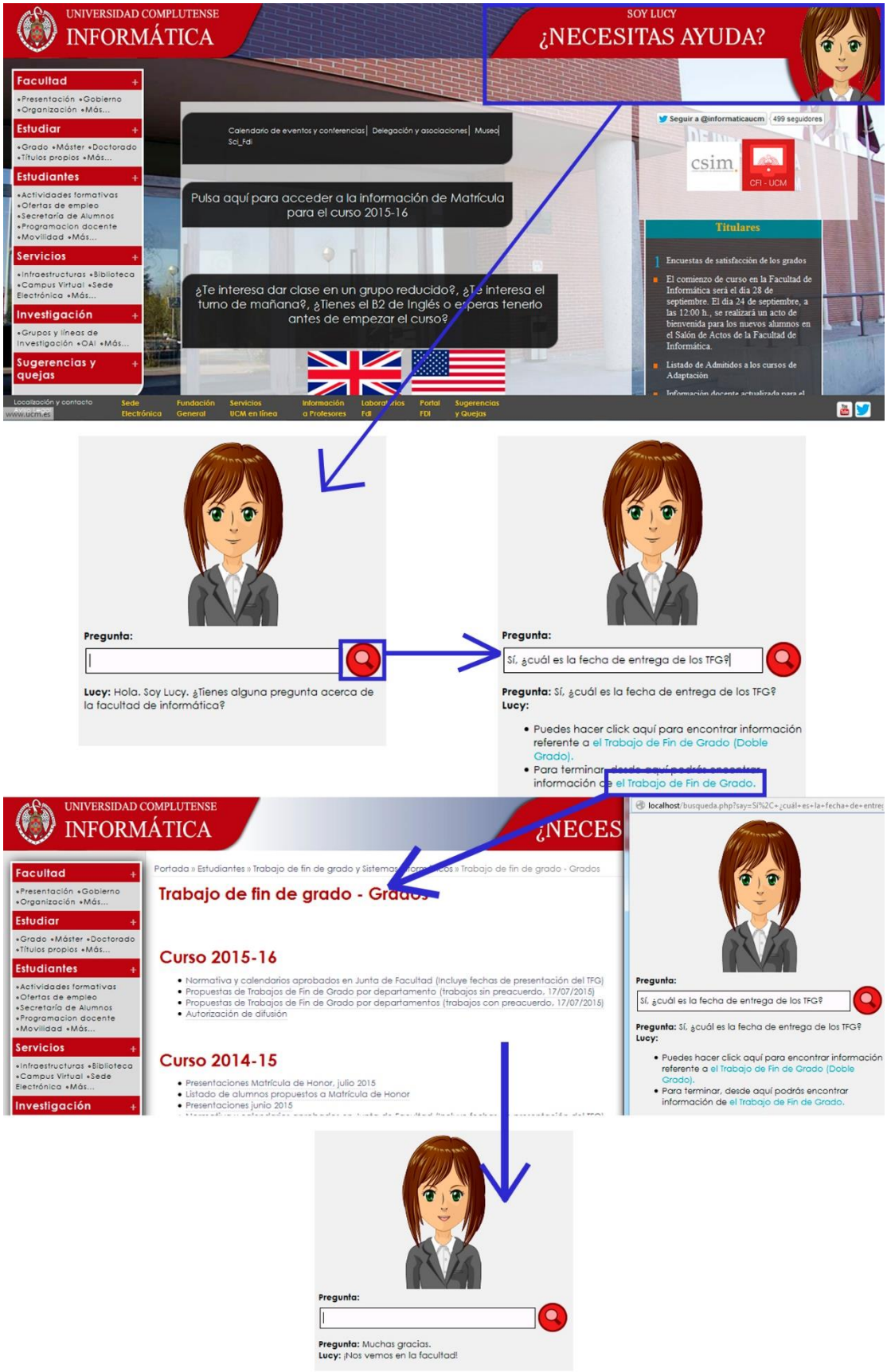

Figura 5.1.2.1: Ejecución completa del chatbot

8. Conclusiones

Ya para finalizar, me gustaría dejar, a modo personal, lo que ha significado realizar este proyecto, ya que puede servir de motivación para otros estudiantes que quieran investigar con más profundidad en el mundo de los chatbots o, sin tener por qué ser estudiante, que tengan una primera guía de por dónde empezar a desarrollar un chatbot.

Bien es cierto que documentación hay mucha en internet, pero también es verdad que una gran mayoría está en inglés y quizás haya quienes no tengan tanto dominio de este idioma. Por ello, creo que esta memoria puede resultar de mucha ayuda para cualquier hispanohablante que necesite consejos o ayudas para resolver problemas que me han surgido a mí.

En mi caso, los problemas han sido bastantes y muy diversos, debido en gran medida a que era una tecnología totalmente nueva para mí. Es verdad que en este último año de carrera (Ingeniería del Software), he cursado casualmente la asignatura de Ingeniería del Conocimiento (Inteligencia Artificial), pero ha sido todo desde un punto de vista más teórico, sin entrar en demasiado detalle en este mundo de los chatbots.

Además de esta tecnología, ha sido necesario hacer un recordatorio de todo lo que viene siendo programación web, ya que ha sido necesario realizarlo todo en un entorno que, de implantarse, sería sobre la página web de la facultad, lo que implica que es necesario conocer estos lenguajes.

Uno de los puntos donde más satisfecho estoy, es en el hecho de que finalmente el sistema se ha creado como un módulo independiente, es decir, para poder llevarlo a la web de la facultad, simplemente haría falta adaptarlo al servidor, y añadir un par de líneas desde la interfaz principal que apunte al módulo. El resto funciona automáticamente. Esto podría ser muy útil especialmente para llevarse a otros sitios, ya que simplemente habría que modificar los archivos de la base de datos, y hacer un reindexado del nuevo sitio.

Por otro lado, tengo que decir, que personalmente todo esto me ha servido mucho en mi crecimiento como ingeniero, ya que es interesante crear un proyecto desde cero, y realizar desde su planificación en el tiempo que se tiene (en mi caso desde la finalización de los exámenes de junio, hasta los exámenes

de septiembre), hasta el propio desarrollo, pruebas e implantación, así como la documentación que acompaña a este proyecto.

Finalmente se ha conseguido implementar un chatbot que responde perfectamente ante una gran mayoría de peticiones acerca de la facultad de informática ante usuario que se conectan en busca de ayuda.

Bien es cierto, que no está especialmente diseñado para mantener conversaciones con el usuario, ya que no es un chatbot diseñado para eso. Como ya se explicó más arriba, el objetivo principal era tener una interacción usuario-chatbot donde se puedan preguntar oraciones completas, y el chatbot responda con una oración con sentido y no una simple url.

Está claro que algunas de las mejoras que se podrían planificar son las siguientes:

- Que el chatbot pudiese distinguir entre diversos contextos para intentar averiguar con más detalle exactamente qué quiere el usuario, y en función de eso proporcionase diferentes respuestas mucho más exactas.
- Añadir voz al chatbot, gracias al Text-to-Speech, que se encargase de reproducir las respuestas del chatbot por los altavoces.
- En un caso más avanzado, permitir la mejora anterior de forma bidireccional, es decir, que también el usuario pudiese comunicarse por voz (Speech-to-Text).
- Una interfaz propia donde actualizar la base de datos de palabras claves frecuentes y donde la actualización de tablas fuese transparente para el adminsitrador.
- Una forma de aprendizaje del chatbot, donde los usuario pudiesen enseñarle cuando diese respuestas muy equivocadas, pasándole lo que buscaban y la url que les hubiese gustado obtener.

Ya para terminar, simplemente deciros que espero que esta memoria me sirva también a mí en un futuro, para recordar mi última hazaña para finalizar lo que es un ciclo de 6 años, que me han permitido acabar con la Ingeniería Técnica en Informática de Gestión, y ahora poder realizar este último curso de adaptación.

Espero os pueda servir a muchas personas de ayuda.

9. Bibliografía

9.1. Chatbots

- http://alice.pandorabots.com/
- http://www.alicebot.org
- https://www.chatbots.org
- http://www.pandorabots.com/
- www.renfe.com/asistente.html
- http://avachara.com/avatar/

9.2. Nutch

- Diapositivas Juan Pavón. Herramientas de recuperación de información en la web. Desarrollo de aplicaciones y servicios inteligentes.
- http://nutch.apache.org/
- http://www.cs.upc.edu/~caim/lab/session4.html
- http://lucidworks.com/blog/nutch-solr/
- http://jcgpz.blogspot.com.es/2011/02/primeros-pasos-con-nutch.html
- http://www.slis.tsukuba.ac.jp/~joho.hideo.gb/doku.php?id=development/how_to_crawl_the_web_with_nutch_and_search_with_solr
- http://blog.building-blocks.com/technical-tips/building-a-search-engine-with-nutch-and-solr-in-10-minutes

9.3. Solr

- http://lucene.apache.org/solr/
- http://lucidworks.com/blog/nutch-solr/
- http://www.slis.tsukuba.ac.jp/~joho.hideo.gb/doku.php?id=development/how_to_crawl_the_web_with_nutch_and_search_with_solr
- http://blog.building-blocks.com/technical-tips/building-a-search-engine-with-nutch-and-solr-in-10-minutes

- http://wiki.apache.org/solr/IntegratingSolr#JavaScript_.2F_JSON_.2F_Node.js
- https://code.google.com/p/solr-php-client/downloads/list

9.4. Lenguajes de programación (PHP, JavaScript, AIML…)

- https://secure.php.net/
- https://validator.w3.org/
- https://books.google.es/books?id=Np7RAgAAQBAJ&pg=PA107&lpg=PA107&dq=aiml+recursion&source=bl&ots=KHJsyQ3RQa&sig=XvMCf8xFt6YRTvH1_mXQTlt2VC0&hl=es&sa=X&ved=0CCsQ6AEwA2oVChMIi7H9rMe0xwIVxwMaCh3jWwFY#v=onepage&q=aiml%20recursion&f=false

9.5. Descarga de software/herramientas

- Nutch: http://archive.apache.org/dist/nutch/1.9/apache-nutch-1.9-src.zip
- Solr: http://archive.apache.org/dist/lucene/solr/4.10.4/solr-4.10.4.zip
- Cliente Solr: https://solr-php-client.googlecode.com/files/SolrPhpClient.r60.2011-05-04.zip
- Program-O: https://github.com/Program-O/Program-O/archive/master.zip
- Simple HTML DOM: http://downloads.sourceforge.net/project/simplehtmldom/simple_html_dom.php?r=&ts=1440697508&use_mirror=netix
- XAMPP: https://www.apachefriends.org/xampp-files/5.6.12/xampp-linux-x64-5.6.12-0-installer.run

9.6. Imágenes

- http://informatica.ucm.es/
- http://secretariageneral.ugr.es/pages/gabcom2012/elviranuevo/!/
- https://www.ucm.es/data/cont/media/www/pag-32082/UCM.jpg
- http://images.eldiario.es/economia/Avatar-ayuda-renfecom_EDIIMA20130219_0412_13.jpg

- http://cartago.lllf.uam.es/grampal/grampal.cgi?m=etiqueta&e=Quiero+conocer+los+horarios
- http://cienciagandia.webs.upv.es/wp-content/uploads/2015/03/p00v2b0b.jpg

10. Anexo I: Código de la aplicación

En este apartado se incluye el código necesario para desarrollar la aplicación. Como se puede observar, no es un código demasiado extenso ni demasiado complicado de entender. Por claridad y espacio, se ha decidido no incluir aquí los comentarios, sin embargo, en los ficheros originales sí aparecen.

10.1. index.html

```
<!DOCTYPE html>
<head>
   <meta charset="UTF-8">
   <meta name="Keywords" content="chatbot,chatbots,Asistente Virtual,Asistentes Virtuales,Inteligencia Artificial,IA,lenguaje natural">
   <title>Facultad de Informática. Universidad Complutense de Madrid</title>
   <link rel="icon" href="lucy/img/logoucm.ico" />
   <link href="lucy/css/integracion.css" rel="stylesheet" type="text/css" media="all"/>
</head>
<body>
   <iframe id="facultad" src="http://informatica.ucm.es"  name="facultad" style="border:0px;"></iframe>
   <a id="lucy" href="busqueda.php?say=openlucy" onclick="window.open(this.href, 'popupwindow', 'scrollbars, resizable, height=600, width=400, left=5000'); return false;">
   <img src="lucy/img/lucyFondo.png" alt="Link a pop-up de Lucy">
   </a>
</body>
</html>
```

10.2. busqueda.php

```
<?php
    session_start();
    require_once ('lucy\controlador.php');
?>

<!doctype html>
<head>
    <meta name="Keywords" content="chatbot,chatbots,Asistente Virtual,Asistentes
Virtuales,Inteligencia Artificial,IA,lenguaje natural">
    <title>Facultad de Informática. Universidad Complutense de Madrid</title>
    <link rel="icon" href="lucy/img/logoucm.ico" />
    <link href="lucy/css/lucy.css" rel="stylesheet" type="text/css" media="all" />
</head>
<body onload="document.getElementById('say').focus()">
    <div id="lucy">
        <center>
            <img src="lucy/img/lucyHablando.gif?<?php echo date("Ymdgis");?>">
        </center>
            <form action="busqueda.php" method="get"
onsubmit="if(document.getElementById('say').value == '') return false;">
                <label for="say"><b>Pregunta:</b></label>
        <center>
                <input type="text" name="say" id="say"/>
                <input type="submit" name="submit" id="btn_say" value="">
        </center>
            </form>

    </div>
    <div id="responses">
        <?php
            controlador();
        ?>
    </div>

</body>
</html>
```

10.3. controlador.php

```
<?php

    require_once ('client-solr.php');
    require_once ('analizador.php');
    require_once ('client-bot.php');
    require_once ('client-bbdd.php');
    header("Content-Type: text/html; charset=utf-8");

    function controlador(){

        $respuesta = analizarChatbot();
        if ($respuesta['response'] == 'insulto'){
            escribirRespuestaInsulto($respuesta);
        } else if ($respuesta['response'] == 'despedida'){
            escribirRespuestaDespedida($respuesta);
        } else if ($respuesta['response'] != 'desconocido'){
                    escribirRespuestaBot($respuesta);
                } else {
                    if (count($palabraClave, $_GET['say']) <= 3){
                        $respuesta = primeraBusqueda($_GET['say'], $_GET['say']);
                    }
                    if ((count($palabraClave, $_GET['say']) > 3) || ($respuesta ==
false)){
                        $palabrasClave = null;
                        $palabrasClave = analizarOracion($_GET['say']);

                        $respuesta = ultimaBusqueda($palabrasClave, $_GET['say']);
                        if ($respuesta == false)
                            escribirRespuestaSOLR($palabrasClave['orig']);
                    }
                }
    }
?>
```

10.4. analizador.php

```
<?php

    require("resources/dom/simple_html_dom.php");

    function quitarTildesURL($string){
        $string = str_replace('á','%C3%A1',$string);
        $string = str_replace('é','%C3%A9',$string);
        $string = str_replace('í','%C3%AD',$string);
        $string = str_replace('ó','%C3%B3',$string);
        $string = str_replace('ú','%C3%BA',$string);
        $string = str_replace('Á','%C3%81',$string);
        $string = str_replace('É','%C3%89',$string);
        $string = str_replace('Í','%C3%8D',$string);
        $string = str_replace('Ó','%C3%93',$string);
        $string = str_replace('Ú','%C3%9A',$string);
        $string = str_replace('ñ','%C3%B1',$string);
        $string = str_replace('Ñ','%C3%91',$string);
        return $string;
    }

    function quitarContracciones($string){
        $string = str_replace(' al ',' a+el ',$string);
        $string = str_replace(' del ',' de+el ',$string);
        return $string;
    }

    function analizarOracion($busqueda){

        $busqueda = str_replace(' ','+',$busqueda);
        $busqueda = quitarTildesURL($busqueda);
        $busqueda = quitarContracciones($busqueda);
        $busqueda = strtolower($busqueda);


        $url = 'http://cartago.lllf.uam.es/grampal/grampal.cgi?m=etiqueta&e=' .
$busqueda;

        $html=file_get_contents($url);
        $dom = new domDocument;
        @$dom->loadHTML($html);

        $dom->preserveWhiteSpace = false;
        $tds = $dom->getElementsByTagName('td');
        $array = array();
```

```
        $i = 0;
        $j = 0;
        foreach ($tds as $td){
            $fila = $i;
            $columna = $j % 4;
            if ($columna == 0)
                $array[$fila][$columna] = $td->nodeValue;
            elseif ($columna == 1 || $columna == 2)
                $array[$fila][$columna] = $td->lastChild->nodeValue;
            $j++;
            if ($columna == 2)
                $i++;
        }

        for ($j = 0; $j < count($array); $j++){
            if ($array[$j][1] == 'N' || $array[$j][1] == 'NPR'){
                $palabrasClave['orig'][] = utf8_decode($array[$j][0]);
                $palabrasClave['convert'][] = utf8_decode($array[$j][2]);
            }
        }

        return $palabrasClave;
    }

?>
```

10.5. client-bbdd.php

```
<?php

    function abrirBBDD(){

        $dbuser = 'program_o';
        $dbpass = '00000';
        $dbhost = 'localhost';
        $dbname = 'program_o';

        $dblink = mysql_connect($dbhost,$dbuser,$dbpass);
        if ($dblink){
            mysql_select_db($dbname,$dblink);
            mysql_query("SET NAMES 'UTF8'");
        }
    }

    function cerrarBBDD(){
        mysql_close($dblink);
    }

    function quitarTildes($string){
         $string = str_replace('á','a',$string);
         $string = str_replace('é','e',$string);
         $string = str_replace('í','i',$string);
         $string = str_replace('ó','o',$string);
         $string = str_replace('ú','u',$string);
         $string = str_replace('Á','A',$string);
         $string = str_replace('É','E',$string);
         $string = str_replace('Í','I',$string);
         $string = str_replace('Ó','O',$string);
         $string = str_replace('Ú','U',$string);
        return $string;
    }

    function primeraBusqueda($palabraClave, $say){
        abrirBBDD();
        $palabraClave = quitarTildes($palabraClave);

        $arrayResultados = Array();

        $query = "select u.respuesta, u.url
                  from urls_comunes u, palabras_clave p, relacion r
```

```
                    where p.palabra like '%".$palabraClave."%' and p.id = r.id_palabra and
r.id_url = u.id;";

        $result = mysql_query($query);

        $numFilas = mysql_num_rows($result);

        if ($result != null){
            for ($j = 0; $j < $numFilas; $j++){
                $respuesta = mysql_fetch_assoc($result);
                $arrayResultados[] = $respuesta; ;
            }
        }

        if ($arrayResultados != null && $numFilas > 0){
            mostrarRespuestaBBDD($arrayResultados, $say, $numFilas);
            cerrarBBDD();
            return true;
        } else{
            return false;
            cerrarBBDD();
        }

    }

    function ultimaBusqueda($palabrasClave, $say){
        abrirBBDD();

        $numPalabras = count($palabrasClave['convert']);

        $arrayResultados = Array();
        for ($i = 0; $i < $numPalabras; $i++){
            if ($palabrasClave['convert'][$i] != 'UNKN')
                $palabraClave = quitarTildes($palabrasClave['convert'][$i]);
            else
                $palabraClave = $palabrasClave['orig'][$i];
            $query = "select u.respuesta, u.url
                      from urls_comunes u, palabras_clave p, relacion r
                      where p.palabra like '%".$palabraClave."%' and p.id = r.id_palabra
and r.id_url = u.id;";

            $result = mysql_query($query);
            $numFilas = mysql_num_rows($result);

            if (($result != null) && ($numFilas > 0)){
                for ($j = 0; $j < $numFilas; $j++){
                    $respuesta = mysql_fetch_assoc($result);
```

```
                    $arrayResultados[] = $respuesta;
                }
            }
        }
        if ($arrayResultados != null){
            $arrayResultados = array_unique($arrayResultados, SORT_REGULAR);
            $arrayResultados = array_values($arrayResultados);
            $numFilas = count($arrayResultados);
            mostrarRespuestaBBDD($arrayResultados, $say, $numFilas);
            cerrarBBDD();
            return true;
        } else{
            return false;
            cerrarBBDD();
        }
    }

    function mostrarRespuestaBBDD($respuesta, $say, $numFilas){
        echo '<b>Pregunta: </b>' . $say . '<br>';

        for ($i = 0; $i < $numFilas; $i++){
            if ($i == 0){
                echo '<b>Lucy: </b>';
                if($numFilas > 1)
                    echo '<br><ul><li>';
                echo nexo(1) . '<a href="#"
onclick="parent.opener.document.getElementById(\'facultad\').src = \''.
$respuesta[$i]['url'] .'\'; return false;">'. $respuesta[$i]['respuesta'] . '</a>';
                if($numFilas > 1)
                    echo '</li>';
                else
                    echo '<br>';
            }else if ($i == ($numFilas - 1))
                echo '<li>' . nexo(3) . '<a href="#"
onclick="parent.opener.document.getElementById(\'facultad\').src = \''.
$respuesta[$i]['url'] .'\'; return false;">'. $respuesta[$i]['respuesta'] . '</a></li>';
            else if ($i < $numFilas)
                echo '<li>' . nexo(2) . '<a href="#"
onclick="parent.opener.document.getElementById(\'facultad\').src = \''.
$respuesta[$i]['url'] .'\'; return false;">'. $respuesta[$i]['respuesta'] . '</a></li>';
        }
        if($numFilas > 1)
            echo '</ul>';
    }


    function nexo($tipo){
```

```
        $nexosContinuacion = array("Además ", "También ", "Por otra parte ", "Igualmente
", "Asimismo ");
        $nexosTerminacion = array("Por último, ", "Para terminar, ", "En último lugar,
");

        $introduccion = array(
            "desde aquí podrás encontrar información acerca de ",
            "puedes hacer click aquí para encontrar información acerca de ",
            "desde aquí podrás encontrar información sobre ",
            "puedes hacer click aquí para encontrar información sobre ",
            "desde aquí podrás encontrar información referente a ",
            "puedes hacer click aquí para encontrar información referente a ",
            "desde aquí podrás encontrar información de ",
            "puedes hacer click aquí para encontrar información de ",
            "desde este enlace podrás encontrar información acerca de ",
            "puedes hacer click en este enlace para encontrar información acerca de ",
            "desde este enlace podrás encontrar información sobre ",
            "puedes hacer click en este enlace para encontrar información sobre ",
            "desde este enlace podrás encontrar información referente a ",
            "puedes hacer click en este enlace para encontrar información referente a ",
            "desde este enlace podrás encontrar información de ",
            "puedes hacer click en este enlace para encontrar información de "
        );

        switch ($tipo){
            case 1: return (ucfirst($introduccion[array_rand($introduccion,1)]));
                break;
            case 2: return($nexosContinuacion[array_rand($nexosContinuacion,1)] .
$introduccion[array_rand($introduccion)]);
                break;
            case 3: return ($nexosTerminacion[array_rand($nexosTerminacion,1)] .
$introduccion[array_rand($introduccion)]);
                break;
        }
    }
?>
```

10.6. client-bot.php

```
<?php
    $display = "";
    $thisFile = __FILE__;
    require('resources/engine-bot/config/global_config.php');
    require('resources/engine-bot/chatbot/conversation_start.php');
    function analizarChatbot(){
        $get_vars = (!empty($_GET)) ? filter_input_array(INPUT_GET) : array();
        $post_vars = (!empty($_POST)) ? filter_input_array(INPUT_POST) : array();
        $form_vars = array_merge($post_vars, $get_vars);
        $bot_id = (!empty($form_vars['bot_id'])) ? $form_vars['bot_id'] : 1;
        $say = (!empty($form_vars['say'])) ? $form_vars['say'] : '';
        $convo_id = session_id();
        $format = (!empty($form_vars['format'])) ? $form_vars['format'] : 'html';
        global $respuesta;
        return $respuesta;

    }
    function escribirRespuestaBot($respuesta){
        if ($respuesta['input'] != 'openlucy')
            echo '<b>Pregunta: </b>' . $respuesta['input'] . '</br>';
        echo '<b>Lucy: </b>' . $respuesta['response'];
    }
    function escribirRespuestaInsulto($respuesta){
        echo '<b>Pregunta: </b>' . $respuesta['input'] . '</br>';
        echo '<b>Lucy: </b> Creo que estás siendo un obsceno. Personas así no necesitan
mi ayuda. Me marcho.';

        cerrarVentana();
    }

    function escribirRespuestaDespedida($respuesta){
        echo '<b>Pregunta: </b>' . $respuesta['input'] . '</br>';
        echo '<b>Lucy: </b> ¡Nos vemos en la facultad!';

        cerrarVentana();
    }

    function cerrarVentana(){
        echo "<script languaje='javascript' type='text/javascript'>
        setTimeout(\"window.close()\",3000);
        </script>";
    }
?>
```

10.7. client-solr.php

```
<?php

    function escribirRespuestaSOLR($palabrasClave){
        global $say;
        define("MAX_RESULTS", 5000);

        require_once("resources/engine-solr/SolrPhpClient.php");
        $numPalabras = count($palabrasClave);

        for ($i = 0; $i < $numPalabras; $i++){

            $client = new SolrClient("192.168.56.101", "8983");
            $client->setUriPath("/solr/collection1/");
            $client->addParameter("start", 0);
            $client->addParameter("rows", MAX_RESULTS);
            $client->addParameter("q", "content:".quitarTildes($palabrasClave[$i]));
            try {
                $client->doQuery();
            } catch (Exception $e) {
                echo "Caught Exception: " . $e->getMessage();
            }


            if ($client->getResults() != null){
                $docs[$i] = $client->getResults();

                for ($j = 0; $j < count($docs[$i]); $j++){
                    $urls[] = $docs[$i][$j]['url'];
                }
            }
        }

            $arrayModa = array_count_values($urls);
            $moda = max($arrayModa);

            $urlsModa = array_keys($arrayModa, $moda);

            $numResultados = count($urlsModa);

            if ($numResultados > 0){
                echo '<b>Pregunta: </b>' . $say . '</br>
                    <b>Lucy: </b>He encontrado algunas urls para tu búsqueda:</br><ul>';
                for ($k = 0; $k < $numResultados; $k++){
```

```
                    $url = $urlsModa[$k];
                    echo '<li><a href="#"
onclick="parent.opener.document.getElementById(\'facultad\').src = \'' .$url.'\'; return
false;">'.$url. '</a></li>';
                }
                echo "</ul>";
            } else
                echo '<b>Pregunta: </b>' . $say . '</br>
                    <b>Lucy: </b>Lo siento, no he encontrado ninguna url que se adapte a
tus necesidades.<br>
                     Revisa si no has cometido ninguna falta de ortografía o intenta
hacer una búsqueda más precisa.';
    }
?>
```

10.8. integracion.css

```
html, body{
    overflow:hidden;
    width:100%;
    height:100%;
    margin:0 ;
    padding:0;
}
#facultad{
    width:100%;
    height:100%;
    left:0px;
    top:0px;
    position:absolute;
    frame-border:0px;
}

#lucy{
    position:fixed;
    right:17px;
    outline:0px;
}
```

10.9. lucy.css

```
body{
    background:rgb(240,240,240);
    margin:0px;
    font-family: 'Century Gothic', Arial, sans-serif;
    font-size:0.8em;
}
#lucy, #responses{
    margin:10px;
}
#say{
    border:solid 1px black;
    z-index:1;
    width:320px;
    height: 30px;
    margin: 0px 0px 0px 0px;
    padding: 0px 0px 0px 4px;
    background-color: #FFFAFF;
    font-family: 'Century Gothic', Arial, sans-serif;
    font-size:1em;
}
#btn_say{
    height: 44px;
    width:44px;
    cursor:pointer;
    z-index:2;
    background:rgba(0, 0, 0, 0) url("../img/lupa.png") no-repeat scroll 0px 0px;
    background-size: 44px 44px;
    border: 0px;
    box-shadow: none;
}

a, a:link, a:visited{
    color: #009FBF;
    text-decoration: none;
}
a:hover, a:active{
    color: #009FBF;
    text-decoration: underline;
}
```

11. Anexo II: Test de Lucy

Cuando se planificó el proyecto al principio, para probar la valía del chatbot se decidió formular 20 preguntas que podrían ser típicas que un usuario de la facultad buscase. En el punto 1.1, ya se puso a prueba el buscador de la facultad, haciendo una búsqueda de 10 palabras clave, en el que se obtuvo un resultado de 5/10. Para este caso, el test pasa de ser de 10 a 33 preguntas (22 comunes y 11 menos comunes), y ya no se introducen palabras, sino oraciones completas.

Este test ser realizó 2 veces. En primer momento tras la primera versión funcional de la aplicación (al finalizar el Sprint 9). El segundo fue tras la versión final del sistema, previo a su entrega final.

Las respuestas se valoran con una puntuación de 0 a 3, siendo:

0 – Sin respuesta
1 – Respuesta encontrada pero muy poco precisa
2 – Respuesta medianamente precisa o encontrada entre otras muchas incorrectas
3 – Respuesta perfecta o muy precisa

El resultado mostrado con el código de colores, corresponden a la segunda vez que se realizó el test (versión final).

Preguntas comunes:

- ¿Me puedes decir las fechas de los exámenes de septiembre? 2/3
- ¿Dónde está situada la facultad? 3/3
- ¿Cuándo son las vacaciones de Navidad? 3/3
- ¿Qué grados se ofertan en esta facultad? 3/3
- ¿Dónde hay información sobre el proyecto de fin de grado? 2/3
- ¿Hay máster o doctorados en la facultad? 1/3
- ¿En qué planta está el museo? 3/3
- ¿Hasta cuándo se cursa la antigua Ingeniería Informática? 3/3
- ¿Cuál es la web de la biblioteca? 2/3
- ¿Me puedes dar acceso al Campus Virtual? 3/3
- ¿Cómo puedo matricularme de forma online? 1/3
- ¿Cuál es el horario de secretaría? 3/3
- ¿Tenéis estudios con Erasmus? 3/3

- ¿Cuál es el despacho del profesor Juan Pavón? 2/3
- ¿Cuál es la programación docente de Ingeniería del Software? 2/3
- ¿Se ofrecen cursos de inglés? 3/3
- ¿Se ofrece bolsa de trabajo para estudiantes? 3/3
- ¿Me puedes dar acceso al correo de estudiantes? 3/3
- ¿Cuándo es la convocatoria extraordinaria de febrero? 3/3
- ¿Tenéis programas para estudiantes discapacitados? 1/3
- ¿Hay posibilidad de convalidar el First Certificate? 1/3
- ¿Puedes darme acceso a Geaportal? 3/3

Preguntas poco comunes:

- ¿Cómo puedo homologar un título sacado en el extranjero? 3/3
- ¿Qué es el Centro Superior de Idiomas Modernos? 3/3
- ¿Dónde puedo obtener el carnet de estudiante? 3/3
- ¿Cuál ha sido la nota de corte de selectividad este año? 3/3
- ¿Dispone la facultad de salón de actos? 1/3
- ¿Ofrece la universidad cursos de verano? 1/3
- ¿Puedo convalidar créditos de libre configuración con deportes? 2/3
- ¿Dónde puedo solicitar la convocatoria de gracia? 3/3
- ¿En qué puedo trabajar cuando finalice la carrera? 0/3
- ¿En qué consiste el PGP Party? 3/3
- ¿Cuál es el código de la asignatura “Ingeniería del Conocimiento”? 1/3

Resultado final tras primera versión

	Puntuación máxima posible	**Puntuación obtenida**	**Relación**	**PORCENTAJE DE CALIDAD**
Preguntas comunes	66	43	43 / 66	0.65
Preguntas poco comunes	33	13	13 / 33	0.39
TOTAL	99	53	56 / 99	**0.56**

Resultado final tras versión final

	Puntuación máxima posible	Puntuación obtenida	Relación	PORCENTAJE DE CALIDAD
Preguntas comunes	66	53	53 / 66	0.84
Preguntas poco comunes	33	23	23 / 33	0.69
TOTAL	99	76	76 / 99	**0.76**

Como conclusión, y tras haber realizado el test por segunda vez, podemos afirmar que el chatbot tiene una calidad del 76%, lo que podemos catalogar de: **NOTABLE.**

La misión del administrador del chatbot consistiría en monitorizar las consultas que realizan los usuarios que utilizan este módulo, para conocer si los resultados que se proporcionan son los esperados y resultan de utilidad. A partir de este punto, sería necesario hacer modificaciones en la base de datos, actualizar los ficheros AIML o modificar el repositorio de Solr.