UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

UBUassistant

Presentado por Daniel Santidrián Alonso
en Universidad de Burgos — 19 de junio de 2017
Tutor: Pedro Renedo Fernández

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. Pedro Renedo Fernández, profesor del departamento de Ingeniería Civil, área de lenguajes y sistemas informáticos.

Expone:

Que el alumno D. Daniel Santidrián Alonso, con DNI 71310411S, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado UBUassistant.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 19 de junio de 2017

Vº. Bº. del Tutor:

D. Pedro Renedo Fernández

Resumen

La búsqueda de información específica en una página web puede ser un proceso largo y tedioso, y más aún cuando accedes a una página web por primera vez. Ciertos portales web, normalmente de instituciones públicas, disponen de abundante información que debería estar visible en un primer plano pero que, sin embargo, está encubierto por diversos menús o submenús.

Con estas premisas nace UBUassistant, un asistente virtual capaz de reconocer una entrada de texto por parte de un usuario y otorgar una respuesta a dicha entrada. De esta manera un usuario puede interactuar con un asistente para encontrar un apartado del portal web en lugar de navegar entre menús que pueden ser ambiguos.

Concretamente, UBUassistant es capaz de orientar a un usuario de la página web de la Universidad de Burgos sin tener que navegar sobre el portal, si existe una respuesta concreta ésta será mostrada de manera inmediata, de lo contrario se otorgará la dirección web donde se localiza la respuesta. Basta con preguntar al asistente sobre algún apartado del sitio web.

Descriptores

Asistente virtual, orientación en portales web, razonamiento basado en casos, aprendizaje tutelado, aplicación web.

Abstract

Searching for specific information in web sites can be a long and tedious process, and even more when you acess for first time a web page. Certain web portals, usually public institutions, provide abundant information that should be visible at first sight but, however, is covered by various menus or sub-menus.

With these premises UBUassistant is born, a virtual assistant that is able to recognize a text entry by a user and granting a response to that entry. This way a user can interact with an assistant to find a section of the web inestead of naviganting among menus that may be ambiguous.

Specifically, UBUassistant is able to guide a user of the website of the University of Burgos without having to navigate through the web, if there is a specific response, it will be shown immediately, otherwise the web adress where the response is located will be given. Just ask the assistant about some section of the website.

Keywords

Virtual assistant, orientation in web portals, case based reasoning, tutored learning, web application.

Índice general

Índice de figuras

Índice de tablas

Introducción

Encontrar información en las páginas de interés público es, a veces, una tarea complicada. Portales web que proporcionan información a un amplio sector de la ciudadanía deberían ser simples y estar estructurados correctamente.

La página web de la Universidad de Burgos es una de las mejores organizadas en comparación con otras universidades tanto nacionales como internacionales. Sin embargo, como denominador común, todas ellas tienen un defecto comprensible, disponen de demasiada información. En ocasiones no es posible encontrar de manera rápida y precisa un apartado concreto del sitio web y hay que sumergirse en ciertos menús que pueden llegar a ser abstractos. La Universidad de Burgos no es la excepción, a través de los menús de su página principal es posible acceder a todos los recursos, sin embargo esto genera un gran problema, son necesarios una cantidad importante de menús que en ocasiones no son todo lo concretos que desearíamos.

El diseño del portal web de la Universidad de Burgos está estructurado en cinco menús generales que contienen una cantidad de submenús que puede parecer excesiva. Además, cabe destacar la existencia de un problema de duplicación de apartados, es decir, a través de distintos submenús puedes llegar a un mismo recurso. Estas situaciones pueden ocasionar desorientación en un usuario.

Actualmente la página web de la Universidad de Burgos dispone de un apartado de búsqueda que permite a los usuarios introducir palabras clave para encontrar los apartados deseados.

Este buscador proporcionado es verdaderamente efectivo si el usuario proporciona palabras clave en el proceso de búsqueda, pero su efectividad baja al expresar con un lenguaje natural la búsqueda deseada.

El método propuesto para facilitar la navegación por la página de la Universidad de Burgos pretende otorgar al usuario una interfaz disponible en todo momento y con aspecto de chat, sobre la que poder realizar una búsqueda.

El asistente, llamado UBUassistant, se encargará de buscar una respuesta al texto introducido por el usuario mediante algoritmos de minería de datos. Con este asistente no será necesario expresarse de una manera artificial utilizando palabras clave, sino que será el propio asistente el encargado de analizar una frase expresada de manera natural para encontrar la respuesta adecuada.

La funcionalidad de UBUassistant se extiende hasta proporcionar sugerencias cuando encuentra múltiples respuestas, además de cuando no existe una contestación. La aplicación proporciona un log de uso, pudiendo obtener los casos más buscados e incluso realizar un aprendizaje de manera supervisada.

1.1. Estructura de la memoria

La memoria tiene la siguiente división en apartados:

- **Introducción:** En este apartado se realiza una descripción de una manera breve del problema que se intenta resolver y la solución otorgada. Además incluye subapartados con la estructura de la memoria y el listado de materiales adjuntos.
- **Objetivos del proyecto:** sección donde se explican los objetivos de desarrollar un proyecto de estas características.
- **Conceptos teóricos:** capítulo en el que se abordan los conceptos teóricos necesarios para comprender el resultado final del proyecto.
- **Técnicas y herramientas:** en esta sección se describen las herramientas y las técnicas que se han utilizado para el desarrollo y gestión del proceso del proyecto.
- **Aspectos relevantes del desarrollo:** apartado donde se tratan aquellos aspectos que se consideran destacados en el desarrollo del proyecto.
- **Trabajos relacionados:** capitulo que expone y describe aquellos trabajos que están relacionados con la temática de asistente virtual.
- **Conclusiones y líneas de trabajo futuras:** sección que explica las conclusiones obtenidas tras la realización del proyecto y la funcionalidad que es posible añadir en el futuro.

Además, se proporcionan los siguientes anexos:

- **Plan del proyecto software:** capítulo donde se expone planificación temporal del proyecto y su viabilidad.
- **Especificación de requisitos:** en este apartado se desarrollan los objetivos del software y la especificación de requisitos.
- **Especificación de diseño:** sección que describe el diseño de datos, el diseño procedimental y el diseño arquitectónico.

- **Documentación técnica de programación:** en este capítulo de explica todo lo relacionado con la programación, la estructura de directorios, el manual del programador y las pruebas realizadas.
- **Documentación de usuario:** apartado que realiza un explicación sobre los requisitos de usuarios, la instalación y proporciona un manual de usuario.

1.2. Materiales adjuntos

Los materiales que se adjuntan con la memoria son:

- Aplicación Java UBUassistant.
- JavaDoc.

Además, los siguientes recursos están accesibles a través de internet:

- Repositorio del proyecto. [32]

Objetivos del proyecto

En este apartado se detallan de forma precisa los distintos tipos de objetivos que se han perseguido con la realización del proyecto.

2.1. Objetivos software

- Desarrollar una aplicación web que permita a los usuarios de la página web de la Universidad de Burgos disponer de un asistente virtual con el que poder interactuar en busca de respuestas y apartados concretos del sitio web.
- Favorecer la búsqueda de información de forma simple y natural.
- Realizar un aprendizaje de nuevos casos en los que no existe respuesta de manera supervisada.
- Dotar a un administrador del log de uso del asistente virtual en busca de perfeccionar respuestas a conceptos más buscados, así como ofrecer la posibilidad de añadir nuevos casos, y editar o eliminar los existentes.

2.2. Objetivos técnicos

- Implementar un algoritmo para buscar respuestas a un texto introducido expresado de forma natural a través del análisis de palabras clave y un framework de razonamiento basado en casos.
- Llevar a cabo una web compatible con todos los navegadores web con el uso de HTML5 y CSS3.
- Realizar test unitarios, de integración y de interfaz.
- Aplicar Scrum, en la medida de lo posible, como metodología de desarrollo ágil.
- Realizar la aplicación siguiendo el concepto de Modelo Vista Controlador, separando la interfaz, el motor de la aplicación y los datos.
- Acceder a una base de datos MySQL mediante Hibernate y JDBC.

- Servirse de GitHub como sistema de control de versiones.
- Utilizar herramientas de control de calidad del software como Code Climate, RefactorIt o InCode.
- Publicar la aplicación resultante en un host accesible.

2.3. Objetivos personales

- Llevar a cabo una aplicación que aporte un valor añadido a la página web de la Universidad de Burgos.
- Intentar solucionar un problema común a la hora de buscar información sobre una página web con mucho contenido como es la de la Universidad de Burgos.
- Mejorar los conceptos adquiridos en la programación orientada a objetos y programación web distribuida.
- Utilizar el mayor número de conocimientos adquiridos durante la carrera.

Conceptos teóricos

Para la mejor compresión del funcionamiento del algoritmo de reconocimiento del texto y obtención de la respuesta mediante razonamiento basado en casos (*CBR, case base reasoning*) se procede a explicar el concepto teórico en el que se ha basado su implementación.

El razonamiento basado en casos es, desde el punto de la inteligencia artificial, un sistema experto que trata de emular el comportamiento humano a la hora de intentar resolver problemas.

3.1. Introducción al concepto

El razonamiento basado en casos basa su funcionamiento en experiencias anteriores del propio sistema o de la persona especializada para otorgar una solución o respuesta a un caso concreto [29].

Las experiencias en las que se basa este razonamiento son los llamados casos. Un caso es una pieza de conocimiento que representa una experiencia y que nos sirve para alcanzar una solución. Un conjunto de casos representarán la lógica con la que trabaja un sistema experto con este razonamiento.

UBUassistant, en un primer escenario, no considera una experiencia previa para definir los casos que utilizará, sino que, sus casos están basados en la experiencia del programador. La estructura considerada para los casos de la aplicación son cinco palabras clave o menos. Cada conjunción de estas cinco (como máximo) palabras clave forman un caso descriptivo. Cada caso descriptivo tiene asociado un caso solución. De esta manera es posible analizar un texto en busca del mayor número de palabras clave y devolver el caso solución correspondiente. El caso solución considerado para la aplicación es una cadena de texto.

Como ejemplo de caso descriptivo podemos tener la siguiente conjunción de palabras clave: [informática, ingeniería, presencial, grado, estudios]. Este

caso descriptivo estará asociado con un caso solución que como ejemplo puede tomar el valor [http://www.ubu.es/grado-en-ingenieria-informatica]. De esta manera conseguimos asociar un conjunto de palabras una solución concreta.

El framework utilizado para construir un sistema CBR ha sido jCOLIBRI2 [3].

3.2. Funcionamiento

El funcionamiento del razonamiento basado en casos consiste en un ciclo que contiene cuatro procesos diferenciados 3.1.

Anteriormente a este ciclo hay que realizar un paso previo, llamado preciclo (*precycle)* y posteriormente a la realización del ciclo se realiza un proceso final denominado postciclo (*postcycle*).

Preciclo

El preciclo realiza la carga de los casos desde sistema de persistencia.

En el caso del framework jCOLIBRI2 [36] disponemos de un método llamado preCycle() que carga nuestros casos desde nuestro sistema de persistencia MySQL.

Para poder llevar a cabo esta tarea es necesario definir el modo de acceso a los casos. Éstos pueden estar en diversos sistemas de persistencia, en nuestro caso debemos crear un conector de base de datos (*DataBaseConnector*).

Una vez definido el modo de acceso a los datos debemos indicar el modo de almacenar los datos cargados en la aplicación. En nuestro caso elegimos un almacenamiento de los casos leidos en una lista (*LinealCaseBase*).

La tecnología que usa internamente este framework para leer los casos de la base de datos es *Hibernate*.

Hibernate

Hibernate es una herramienta para Java que permite realizar un mapeo objeto/relacional (*ORM, Object Relacional Mapping*).

Con esta herramienta se consigue conectar los atributos de una clase orientada a objetos con los atributos de una base de datos relacional [10].

Hay dos formas de conseguir este mapeo, bien utilizando ficheros XML, bien usando las anotaciones de Hibernate en las clases Java. El modo de realizarlo por parte de jCOLIBRI2 es mediante ficheros XML.

Para ello debemos indicar al DataBaseConnector donde se encuentra el archivo XML que configura nuestra base de datos. En este fichero llamado

databaseconfig.xml se indican los nombres del resto de archivos que se encargarán de conectar con la base de datos y de establecer la estructura de los atributos de las clases que van a ser mapeadas.

En nuestro caso debemos tener una clase donde se almacenen las cinco palabras clave que forman nuestro caso de descripción, además de una clase donde se almacene el campo respuesta que forma nuestro caso solución.

Ciclo

Acabada la configuración y la carga de los casos, se procede a realizar el ciclo. Como ya se había comentado anteriomente un ciclo se compone de cuatro actividades.

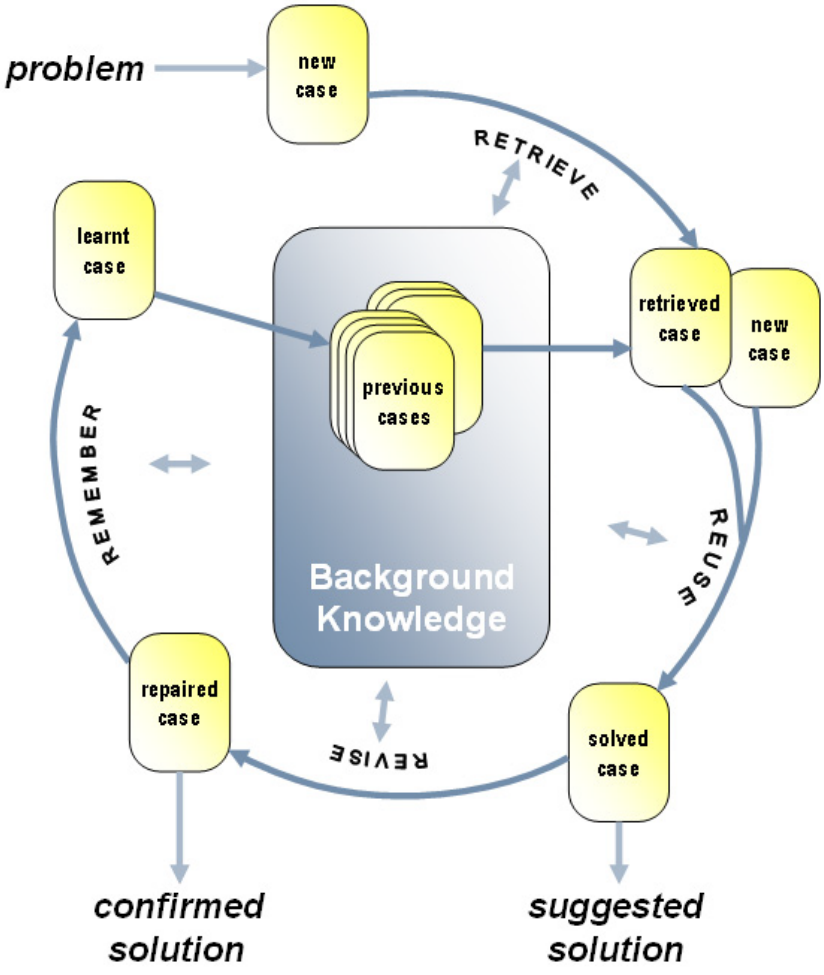

Figura 3.1: Ciclo del razonamiento basado en casos

Recordar

El proceso de recordar (*retrieval*) consiste en recordar los casos similares al que estamos analizando.

El sistema experto, a la hora de enfrentarse a un nuevo problema, deberá seleccionar, de entre los casos disponibles, aquellos que sean más similares al del nuevo problema. De esta manera el sistema lleva a cabo un proceso similar al de una persona humana, que ante un nuevo problema que tiene que resolver rescata de su memoria acontecimientos similares que le ayuden en esa tarea.

jCOLIBRI2 proporciona una serie de métodos para configurar y realizar el proceso de recordar. En un primer paso se define la forma de evaluar la similitud. Se configuran los siguientes apartados.

- **Similitud global:** con este parámetro se detalla la forma de obtener un valor a partir de varios. A partir de evaluar la similitud de cada atributo, cinco atributos en nuestro caso, se obtiene un valor global que puede calcularse de varias maneras. Entre las formas de calcularlo se puede encontrar la media, el máximo, el mínimo o la distancia euclídea. El método finalmente seleccionado fue la distancia euclídea, con el que se obtivieron mejores resultados.
- **Similitud de cada atributo:** define la forma de evaluar la semejanza de cada atributo. Al igual que en el anterior caso existen diferentes maneras de calcular este parámetro, seleccionando el método MaxString que se adecua a nuestra representación del caso como palabras clave.
- **Peso de cada atributo:** puede existir la situación donde no se desea que la evaluación de la similitud de los atributos de un caso tengan el mimo peso en el resultado final. Por ello, el framework nos permite configurar el peso de cada atributo, en nuestro caso las cinco palabras deberán tener el mismo peso. Se ha optado por un peso de 0.75 en todos los atributos al obtener resultados satisfactorios.

Una vez configurado el modo de obtener la similitud es posible utilizar el método *evaluateSimilarity* al que le pasaremos los casos, el caso obtenido del texto del usuario, y el objeto que almacena la configuración de similitud. Este método obtiene una colección, aplicando también el método de vecinos más cercanos, donde quedan almacenados los casos junto con el resultado de la comparación con el caso generado por el texto del usuario. La solución adoptada por la aplicación desarrollada no utiliza el método de vecinos más cercanos, en su lugar se eliminan todos los resultados cuya similitud sea inferior de 0.35, de esta manera se consiguen resultados más positivos.

Por cada palabra que introduce el usuario de crea una colección con los resultados de comparar esa palabra con las palabras existentes en los casos.

Reutilizar

El proceso de reutilización (*reuse*) consiste en utilizar la información obtenida de la etapa de recordar para resolver el problema.

El framework utilizado dispone de métodos propios básicos para esta etapa del ciclo. Sin embargo, la propia documentación deja este paso abierto a los desarrolladores argumentando que es una etapa muy dependiente del ámbito de la aplicación a desarrollar.

Siguiendo estos consejos se desarrolla un algoritmo propio para obtener los resultados deseados. El algoritmo podría resumirse en los siguientes pasos:

- **Construir resultados finales:** Como se ha mencionado con anterioridad, en la etapa de recordar analizamos cada palabra introducida por el usuario con las palabras de los distintos casos generando una colección con estos resultados (eliminando posteriormente los resultados con una evaluación peor que 0.35). Con esto conseguimos obtener las mejores soluciones o respuestas para cada palabra, si varias palabras comparten la misma solución (forman parte del mismo caso descriptivo) se combinan. Se expone un ejemplo para su mejor comprensión: Ante una búsqueda por parte del usuario de las palabras [ingeniería, informática, becas e internacionales] el algoritmo generaría los siguientes resultados finales, [ingeniería, informática] →[http://www.ubu.es/grado-en-ingenieria-informatica] y [becas, internacionales] →[http://www.ubu.es/becas-de-cooperacion]
- **Obtención de solución única:** Si en el proceso de construir resultados finales obtenemos solamente una respuesta, ésta es la que será mostrada como solución.
- **Obtención de solución múltiple:** En el caso de que el proceso de construir resultados finales obtenga varios resultados, como el ejemplo expuesto, la aplicación mostrará todas las opciones para que el usuario elija la que desee.
- **No existencia de resultados:** Se puede dar el caso más que probable de que el usuario realice una búsqueda con palabras que no existen en nuestros casos, en esta situación se mostrarán tres sugerencias donde se encontrarán los casos con mejor similitud aunque no lleguen a la anterior restricción de 0.35.

Revisar

El proceso de reutilizar nos otorga una solución, sin embargo, es necesario una etapa de revisión (*revise*).

Esta etapa consiste en determinar si una respuesta puede considerarse como válida en términos de solucionar el problema planteado.

De nuevo, como en ocasiones anteriores nos encontramos con una fase muy dependiente de la aplicación desarrollada, por lo que jCOLIBRI2 nos anima a utilizar nuestro propio sistema de revisión en aras de obtener mejores resultados.

El sistema de revisión propuesto no es automático sino que depende del usuario de la aplicación. Será el usuario el que decida si una respuesta otorgada se ajusta a sus pretensiones. Para facilitar el sistema de valoración de respuestas se ha decido otorgar al usuario un sistema basado en estrellas, siendo una estrella la valoración más baja y cinco estrellas el valor más alto.

Las valoraciones de los usuarios quedan almacenadas en una tabla de log de uso que puede ser visionada por el administrador de la aplicación desde la página destinada a la administración de la aplicación. Quedará a juicio del administrador cambiar la respuesta de un determinado caso que haya obtenido de manera continuada valoraciones negativas.

Retener

Retener (*retain*) es la etapa donde nuevos casos que pueden ser útiles en el futuro son almacenados.

Se opta por realizar un algoritmo independiente del framework que se adecúe más a nuestro problema.

La implementación de la etapa de retención llevada a cabo consiste en almacenar en una tabla de la base de datos una nueva palabra clave junto con una respuesta asociada. Cuando un usuario introduce una búsqueda para la que no existe una respuesta se muestran tres recomendaciones de búsqueda. El usuario puede obviar esas recomendaciones y proceder a realizar una nueva búsqueda, pero, en el caso de que acepte una de las recomendaciones, la respuesta de esta recomendación será asociada con la palabra buscada que ha generado esta recomendación.

Para aclarar el concepto se explica un ejemplo. Si un usuario busca una frase que contiene la palabra [mechanic] junto con otras palabras que tampoco tienen respuesta, se le otorgará una sugerencia de búsqueda que será [mecánica]. Si el usuario acepta la recomendación se almacenará en una tabla la palabra [mechanic] junto con la solución de la palabra [mecánica].

Este proceso de retener también requiere de un administrador que decida las palabras que finalmente se añadirán como un nuevo caso, ya que realizarlo de manera automática puede provocar un aprendizaje erróneo.

Postciclo

El postciclo es la última etapa después de realizar el ciclo. Se encargará de cerrar la conexión con la base de datos que carga los casos.

En nuestra aplicación podemos realizar el postciclo, mediante el método *postCycle*, en cualquier momento después de la etapa de reutilizar, ya que el algoritmo de las siguientes fases es independiente de la conexión con las tablas de casos de la base de datos.

Técnicas y herramientas

4.1. Metodología

Scrum

Scrum es una metodología ágil para el desarrollo software que consiste en realizar de manera iterativa pequeños incrementos sobre una primera aproximación del producto [24].

Existen distintos componentes humanos en esta metodología [23]

- **Product Owner:** se encarga de definir la funcionalidad y las prioridades en el product backlog junto con el scrum master.
- **Scrum master:** coordina y soluciona problemas del equipo de desarrollo.
- **Equipo de desarrollo:** encargado de la codificación.

Como puede verse en la imagen 4.2 se diferencian distintos periodos y reuniones:

- **Sprint:** periodo de trabajo para desarrollar un incremento. La duración puede variar entre una y cuatro semanas.
- **Reunión inicial:** se realiza al comienzo de cada sprint determinando el trabajo a realizar que se representa en el sprint backlog.
- **Reunión diaria:** donde el equipo se autogestiona.
- **Reunión final:** se realiza al final donde se revisa el producto obtenido.

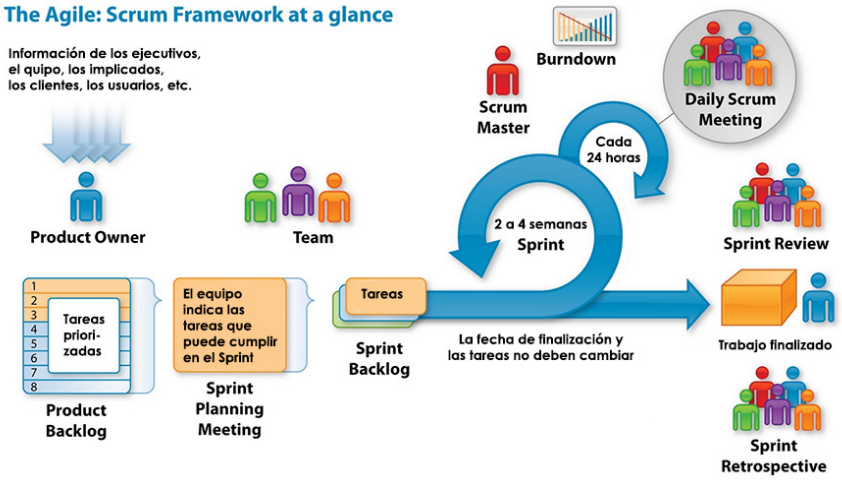

Figura 4.2: Ciclo de la metodología scrum

Programación extrema

Programación extrema (*eXtreme Programming*) es una metodología de desarrollo ágil de software. Su característica principal es la adaptabilidad al cambio de requisitos en cualquier momento de la vida del proceso. Además, esta metodología tiene más propiedades tales como [31]:

- Desarrollo iterativo e incremental.
- Integración frecuente del equipo de programación con el cliente.
- Entregas frecuentes.
- Corrección de errores antes de añadir nueva funcionalidad.
- Código simple.

4.2. Patrones de diseño

Modelo Vista Controlador

Modelo vista controlador (*MVC*) es un patrón arquitectónico que nos ayuda a separar los datos, la lógica de negocio y la interfaz de usuario [20].

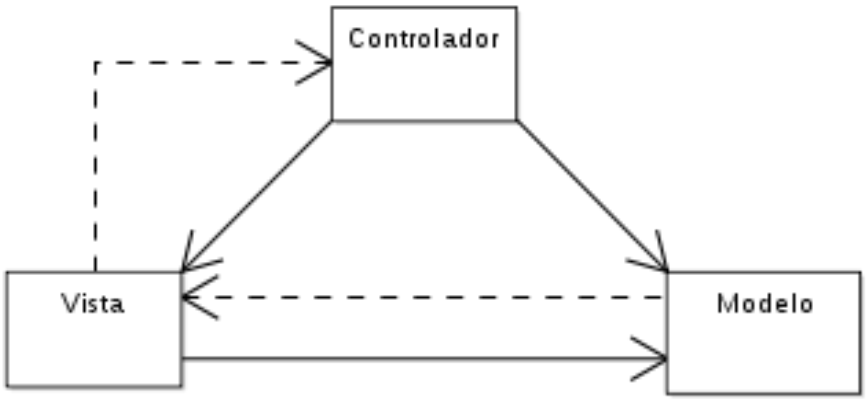

Figura 4.3: Diagrama de clases MVC

Sus componentes son los siguientes:

- **modelo:** representa los datos y la lógica de negocio.
- **vista:** presenta la información del modelo.
- **controlador:** controla las entradas del usuario y selecciona la vista.

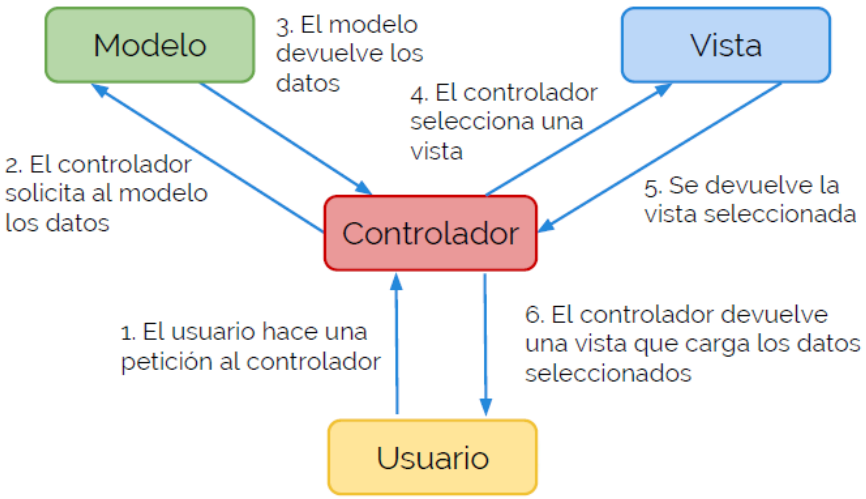

Figura 4.4: Diagrama de interacción MVC [37]

Aplicando estos conceptos a una aplicación web realizada mediante JSP con una base de datos como sistema de persistencia, podemos obtener el siguiente diagrama.

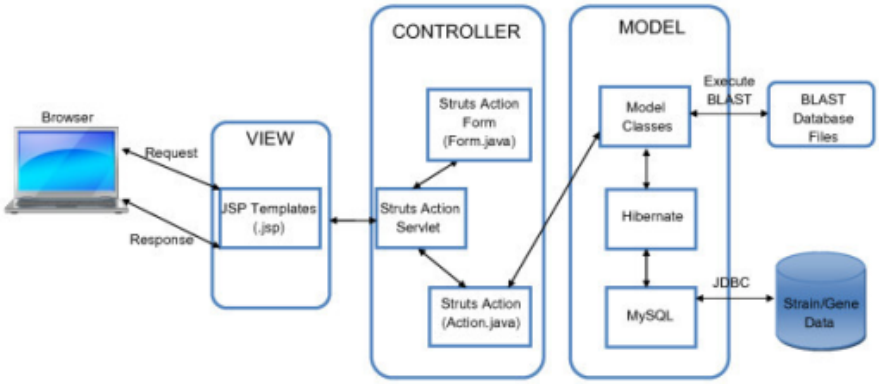

Figura 4.5: Diagrama de MVC específico de una aplicación web JSP

4.3. Control de versiones

- Herramientas consideradas: Git.
- Herramienta elegida: Git.

Git es un software que permite gestionar los cambios producidos sobre los componentes de un proyecto. Se dice que es un gestor de versiones, definiendo versión como el estado de un proyecto en un determinado momento de su desarrollo. Git tiene diferentes características que lo hacían muy atractivo para este proyecto, como la gestión distribuida trabajando con copias locales [7].

4.4. Hosting del repositorio

- Herramientas consideradas: Bitbucket, Github y XpDev.
- Herramienta elegida: Github.

GitHub es una plataforma donde poder alojar proyectos que utilizan Git como control de versiones. Si código es almacenado de forma pública la herramienta es gratuita.

GitHub dispone de una serie de características que favorecieron su elección en detrimentos de las demás herramientas consideradas. Alguna de estas características son la posibilidad de crear una página web del proyecto, ver gráficos, trabajo colaborativo e integración con herramientas de gestión de proyectos [8].

4.5. Gestión del proyecto

- Herramientas consideradas: Gestión manual.
- Herramienta elegida: Gestión manual.

La gestión del proyecto fue manual, organizando cada semana las tareas pendientes, en curso y las terminadas, junto con las horas estimadas y reales.

4.6. Comunicación

- Herramienta considerada: email.
- Herramienta elegida: email.

La comunicación entre los distintos integrantes de este proyecto ha sido realizada mediante email. El email proporciona una comunicación rápida y sencilla si los integrantes del proyecto son dos personas, como es el caso.

4.7. Lenguaje de programación

- Herramientas consideradas: Java y Python
- Herramienta elegida: Java.

Java es un lenguaje de programación orientado a objetos, que soporta concurrencia y permite ejecutar un código sin tener que recompilarlo. Es uno de los lenguajes más usados, incluidas las aplicaciones web cliente-servidor [11].

4.8. Entorno de desarrollo integrado (IDE)

- Herramientas consideradas: IntelliJ, NetBeans, ColibriStudio y Eclipse.
- Herramientas elegidas: ColibriStudio y Eclipse.

ColibriStudio es un IDE que integra todas las características del framework jCOLIBRI2. Este IDE nos permite crear aplicaciones basadas en el razonamiento basado en casos mediante asistentes visuales. En más alto nivel ColibriStudio es una versión modificada del popular Eclipse IDE [4].

Eclipse es un entorno de desarrollo integrado compuesto por una serie de herramientas que permite el desarrollo de aplicaciones Java. Su versión JEE nos provee de las características necesarias para el desarrollo de una aplicación

web JSP, pudiendo añadir software de terceros para ampliar su funcionalidad [6].

Se empezó con el uso de ColibriStudio, pero una vez creado el núcleo de la aplicación CBR se pasó a Eclipse.

4.9. Sistema de gestión de bases de datos

- Herramientas consideradas: PostgeSQL y MySQL.
- Herramienta elegida: MySQL.

MySQL es uno de los sistemas gestor de base de datos más populares y usados gracias a la alta compatibilidad de los lenguajes de programación y su facilidad de uso [21].

4.10. Documentación

- Herramientas consideradas: TexMaker y OpenOffice.
- Herramienta elegida: TexMaker.

TexMaker es un editor de textos que agrupa las herramientas necesarias para crear un documento LaTeX.

LaTeX es un sistema de creación de documentos que permite el uso de comandos y macros para realizar documentos con una alta calidad tipográfica [17].

4.11. Servicios de integración continua

Cobertura del código

- Herramienta considerada: EclEmma.
- Herramienta elegida: EclEmma.

EclEmma es una herramienta Java que permite integrarse en Eclipse para analizar la cobertura del código respecto a los test unitarios. Representa la cobertura mediante colores sobre el propio código además de presentar un porcentaje detallado por clase [5].

Calidad del código

- Herramientas consideradas: RefactorIt, InCode y SonarQube.
- Herramientas elegidas: RefactorIt, InCode y SonarQube.

RefactorIt es una herramienta que puede integrarse en el IDE de Eclipse o utilizarse en su versión independiente y que proporciona diferentes métricas medidas sobre el código Java permitiendo, además, refactorizar automáticamente el mismo [22].

InCode, por su parte, es una herramienta que analiza en código Java en busca de defectos del código.

SonarQube es una plataforma que evalúa código fuente en búsqueda de código duplicado, cobertura de código, bugs y defectos de código [30].

4.12. Sistemas de construcción automática del software

Maven

Maven es una herramienta que se utiliza para la gestión y construcción de proyectos Java. Sus tareas referentes a la construcción son la compilación y el empaquetado. Mediante un Project Object Model (POM) con estructura XML, Maven, describe el proyecto a construir, sus dependencias y componentes externos [18].

4.13. Sistemas de pruebas automáticas

Selenium

Selenium es una herramienta que proporciona un entorno para desarrollar pruebas en entornos web. Selenium nos provee de un IDE para realizar grabaciones sobre una página web las pruebas deseadas. Una vez realizada la grabación nos permite exportar las pruebas a un lenguajes de programación determinado estando disponibles algunos como Java, Python, PHP... [26]

4.14. Librerías

JUnit

JUnit lo forman un conjunto de bibliotecas utilizadas para realizar pruebas unitarias sobre el código [16].

jCOLIBRI2

jCOLIBRI es un framework Open Source que permite la creación de sistemas basados en el razonamiento basado en casos (CBR). Proporciona todos los métodos y mecanismos para la creación de un sistema experto de la manera más simplificada posible.

Es un framework desarrollado en Java por la Universidad Complutense de Madrid y se distribuye bajo licencia GNU Lesser General Public License (LGPL) [2].

Google Guava

Google Guava aes un conjunto de librerías donde se incluyen nuevos tipos de colecciones, colecciones inmutables, librerías de gráficos, y demás utilidades para aplicaciones Java [9].

Selenium

Selenium, como librería, otorga todas las herramientas para realizar pruebas automáticas sobre sitios web. Esta librería provee de todas las dependencias, como por ejemplo WebDriver, para poder realiza pruebas de interfaz en lenguaje Java [25].

4.15. Desarrollo web

Servidor Apache Tomcat

Apache Tomcat es un servidor que permite hospedar aplicaciones web desarrolladas en Java [28].

4.16. Otras herramientas

Mendeley

Medeley es una aplicación que permite compartir y gestionar referencias bibliográficas, documentos, etc. Con su versión web y su versión de escritorio podemos añadir tanto referencias a páginas web, como a documentos que tengamos descargados [19].

Una característica que la hizo interesante para su aplicabilidad es la capacidad de exportar las referencias guardadas en formato BibTex para poder utilizarse en LaTeX.

Aspectos relevantes del desarrollo del proyecto

Está sección tratará sobre los aspectos más interesantes del desarrollo del proyecto. Se incluyen las explicaciones de las distintas etapas del mismo, así como la exposición de las cuestiones más relevantes del diseño e implementación.

5.1. Inicio y fase de análisis

La idea del proyecto surge de un antiguo propósito de integrar un asistente virtual en la página web de la Universidad de Burgos que al final no pudo llevarse a cabo.

Tras comentar con el tutor una serie de propuestas, me transmitió la idea de implementar un asistente virtual con el que los usuarios de la Universidad de Burgos puedan interactuar. La propuesta me pareció interesante desde el punto de vista de un usuario de la web, ya que no siempre es fácil encontrar la información deseada.

5.2. Metodologías aplicadas

La metodología principal que se ha intentado aplicar en el desarrollo del proyecto ha sido Scrum.

No se siguieron las indicaciones de la metodología Scrum de manera estricta al tratarse de un proyecto pequeño, sin equipo de codificación grande, y sin poder realizar todas las reuniones necesarias. Sin embargo sí que se siguieron las siguientes pautas:

- Desarrollo iterativo e incremental del producto mediante sprints.

- Duración en la mayoría de los sprints de una semana, exceptuando alguna excepción por días festivos.
- Reuniones al final de cada sprint para evaluar el producto obtenido y planificar la siguiente iteración.
- Entrega del producto totalmente funcional al final de cada sprint.

Se considera también la aplicabilidad de la metodología *eXtreme Programming* al cambiar los requisitos en mitad de su desarrollo, característica principal de la programación extrema.

El desarrollo del algoritmo *CBR* fue realizado con técnicas de prueba y error para determinar los parámetros idóneos de la función que evalúa la similitud de los casos.

5.3. Formación

Durante el proceso de análisis y desarrollo del proyecto, se necesitaban conocimientos que todavía no habían sido adquiridos, en su mayor parte por estar cursando las asignaturas relacionadas de forma paralela. Por ello fueron necesarios algunos contenidos didácticos que ayudaron a realizar el producto.

Se necesitó la compresión de los sistemas expertos que utilizan razonamiento basado en casos. Para ello se leyeron los siguientes artículos:

- Razonamiento Basado en Casos: Ün visión general"(Laura Lozano y Javier Fernández) [34].
- Tutorial jCOLIBRI (J. Recio García, B. Díaz Agudo y P. González Calero) [36]

En la primera versión del proyecto se necesitó ampliar conocimientos sobre interfaces gráficas en Java Swing. Para ello se accedió a los siguientes recursos online:

- *Trail: Creating a GUI With JFC/Swing* (Oracle) [27].

Con el cambio de requisitos, la aplicación pasó a funcionar como un sistema distribuido cliente servidor gracias a JSP. Para poner en práctica este nuevo concepto se necesitaron conocimientos todavía no adquiridos. Esto se solventó consultando las siguientes referencias:

- *JSP Tutorial* (tutorialspoint) [14].
- Aplicaciones web/sistemas web (Juan Pavón Mestras) [35]

Además, durante todo el desarrollo se consultó con asiduidad la comunidad de *StackOverflow* y los tutoriales de *w3schools*.

5.4. Desarrollo del algoritmo

El algoritmo de búsqueda de respuesta al texto introducido por el usuario fue uno de los aspectos importantes en el desarrollo de la aplicación. Se debían tener en cuenta una serie de requisitos.

- La forma de representar los casos.
- El estilo de texto que se quiere reconocer.
- Proceso de aprendizaje supervisado por un administrador.
- Proceso de revisión supervisado por un administrador.

En un principio había que determinar el sistema de persistencia elegido, se optó por una base de datos MySQL dado que el framework ofrecía acceso a base de datos mediante Hibernate.

Acto seguido, se determinó la forma de representar los casos descriptivos en la base de datos, se empezó por una frase compacta, pero pronto nos dimos cuenta que la función de similitud no iba a trabajar de manera correcta sobre una frase. De este modo, se procedió a cambiar la representación de los casos por tres palabras clave, para posteriormente aumentarla a cinco palabras clave.

También había que especificar la representación de los casos solución. En este caso la elección fue más sencilla, una cadena de texto.

En este punto, una vez definidos todos los aspectos de los casos, nos encontramos el primer problema con el IDE seleccionado (ColibriStudio). No era posible conectar la base de datos mediante el asistente. Por lo tanto, fue necesario una tarea de comprensión más profunda de Hibernate para conectar la base de datos generando los ficheros necesarios de forma manual.

Solventado el problema, se necesitaron varios incrementos de la aplicación para conseguir el algoritmo exacto deseado. Con los casos cargados desde la base de datos el algoritmo consiste en un ciclo formado por cuatro partes:

- **Recordar:** en este punto del algoritmo se utiliza la función de similitud sobre cada palabra introducida por el usuario, generando una colección con el valor final obtenido.
- **Reutilizar:** fase que permite utilizar la colección anterior para extraer la respuesta. Este fue uno de los puntos más conflictivos, ya que el usuario puede escribir tantas palabras como quiera si relación alguna entre ellas, pertenecientes a distintos casos y en orden distintos al guardado en la base de datos.
  Por ello en el paso de recordar se analiza cada palabra del usuario de manera independiente y en este paso se fusionas los mejores resultados obtenidos que comparten caso, obteniendo así las respuestas deseadas.

- **Revisar:** consiste en realizar una valoración de la respuesta. Esta tarea es desempeñada por el usuario, que será el que decida si valora una respuesta o no. Por lo tanto esta fase puede no ser ejecutada siempre y es independiente del ciclo.
  Se le otorga al usuario una barra de estrellas que cuando es pulsada guarda en base de datos mediante JDBC una entrada en la tabla de log de uso con la puntuación otorgada asociada a las palabras del caso que han generado la respuesta.
  De esta manera, desde la página de administración, un encargado puede visualizar aquellos casos que necesitan ser revisados.
- **Retener:** consiste en realizar una asociación de una palabra buscada por el usuario a una respuesta de un caso ya almacenada.
  En este caso, este proceso es transparente pero también dependiente del usuario. Su funcionamiento se basa en las recomendaciones otorgadas por el sistema experto. Si una búsqueda no tiene resultados se generan sugerencias con los tres casos más próximos. Si se acepta una sugerencia se asocia la entrada del usuario con la respuesta del caso aceptado.

5.5. Arquitectura web

Como se ha comentado anteriormente, el proyecto cambió de requisitos durante el desarrollo del mismo. Esto fue debido a su intención inicial y los problemas que surgirían de continuar la anterior idea. Para ver más detalles ver la sección 5.7.

La arquitectura elegida para el desarrollo de la aplicación web fue JSP. Su elección se basó principalmente en poder reutilizar el algoritmo de la aplicación realizado bajo los anteriores requisitos.

La arquitectura JSP (*JavaServer Pages*) finalmente seleccionada permite la creación de páginas web dinámicas con llamadas al servidor, donde se alojan las clases Java [15].

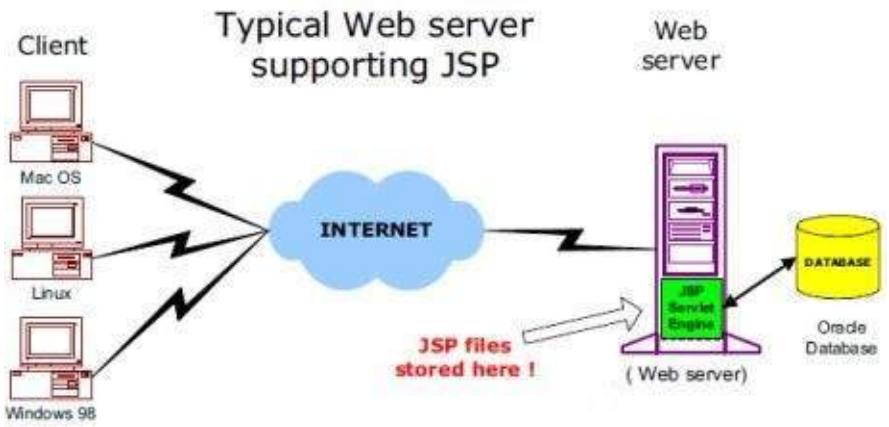

Figura 5.6: Arquitectura JSP [12]

Su aplicación en el proyecto ha consistido en realizar una página de inicio donde encontramos un icono en la parte inferior derecha. Este icono, al ser pulsado, abrirá una ventana flotante donde correrá nuestro asistente virtual.

La ventana flotante carga una página JSP, dentro de esta página se encontrará el mensaje de bienvenida, junto con lo elementos necesarios para que el usuario escriba un texto.

Cada vez que el usuario envía un texto, se realiza una llamada al servidor. Con esta llamada ejecutamos el algoritmo, concretamente la fase de reutilización y obtenemos las repuestas más acordes con las palabras introducidas por el usuario.

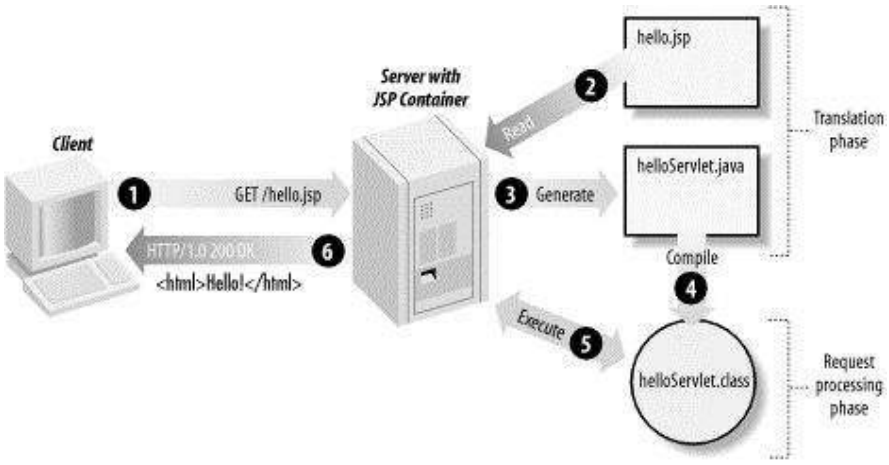

Figura 5.7: Pasos realizados por JSP en las llamadas al servidor

En este punto pueden producirse tres escenarios, existencia de una res-

puesta única, de una respuesta múltiple o la no existencia de una respuesta.

Si existe solamente una respuesta, ésta es mostrada y, acto seguido, el usuario es redirigido a una siguiente página JSP para mostrar el mecanismo de valoración de respuesta.

Si existen varias respuestas, el usuario es redirigido a otra página donde se le mostrarán las distintas respuestas, pudiendo elegir y valorar tantas como existan.

En el caso de que no exista una respuesta, se redirige al usuario a una página mostrando las sugerencias de búsqueda. En el caso de que el usuario acepte una sugerencia se realizará el proceso de aprendizaje.

Todas estas funcionalidades han sido desarrolladas tanto dentro de las propias páginas JSP, como con llamadas a las clases mediante código Java embebido en HTML.

El procesamiento de la entrada del usuario, así como el paso de elementos entre las distintas páginas se ha realizado mayoritariamente con formularios debido a su simplicidad, concretamente con el método *post* ocultando al usuario los parámetros pasados. En ciertos casos se han utilizado variables de sesión para poder utilizarlas en cualquier punto de la aplicación sin que sufran cambios, como es el caso del identificador único de usuario, que es creado al pulsar sobre el icono que abre la aplicación [13].

5.6. Base de datos

El gestor de base de datos elegido fue MySQL. Su elección se basó en su simplicidad y en su compatibilidad con las herramientas de las que se iba a hacer uso.

La base de datos está compuesta por siete tablas. Dos de ellas referentes a la representación de los casos.

La tabla *casedescription* contiene la descripción de los casos, cinco palabras clave junto con la categoría a la que pertenecen para almacenar el log de uso.

La tabla *casesolution* contiene la solución para cada caso, que está formada por una cadena de texto.

La tabla *saludos* incluye las palabras comunes que pueden ser introducidas por el usuario así como las contestación oportuna.

La tabla *frases* es la que posee tanto las frases de saludo como las frases de respuesta que otorgan a la aplicación un aspecto más humano y sociable.

La tabla *aprendizaje* recoge la palabra a aprender junto con la respuesta asociada.

En la tabla *logger* se almacena el log de uso de cada usuario. Se guardan por cada conjunto de palabras buscadas el identificador de usuario, la fecha, las palabras buscadas, la respuesta otorgada, la categoría de la búsqueda, el número de búsquedas, el número de votos y la votación total.

Por último la tabla *administradores* es la encargada de almacenar las credenciales para que los administradores puedan acceder a la página de mantenimiento.

Se han utilizado dos formas de acceder a las tablas almacenadas en la base de datos, *Hibernate* y *JDBC*.

El uso de Hibernate se ha debido al uso del propio framework jCOLIBRI, ya que es el método que utiliza para mapear las clases con las tablas de los casos almacenadas en la base de datos.

El asistente proporcionado por ColibriStudio permite configurar el método de persistencia con base de datos a través de un asistente, sin embargo, este asistente llegaba a un punto donde se producía algún tipo de error interno que no le dejaba avanzar. Por ello se tuvo que indagar en la forma de crear correctamente los archivos para que el framework funcionase de forma apropiada.

Los ficheros que forman la configuración de Hibernate son:

- **databaseconfig.xml:** contiene las referencias a los archivos de configuración de la base de datos (hibernate.cfg.xml) además de los nombres de las clases junto con los archivos que los mapean. **hibernate.cfg.xml:** archivo que posee la configuración de la base de datos (*username, password, driver, URL...*) **caseDescription.hbm.xml:** contiene la configuración para realizar el mapeo de cada atributo de la clase Java con la tabla almacenada referente a los casos descriptivos. **caseSolution.hbm.xml:** incluye la configuración necesaria para realizar el mapeo de cada atributo de la clase Java con la tabla almacenada referente a los casos solución.

El driver *JDBC* es utilizado por la clase Java *DatabaseConnection*. Esta engloba todo el acceso a la base de datos, y se encarga de la lectura y escritura en la base de datos de las tablas que no pertenecen a los casos.

Concretamente, realiza la lectura de las tablas que contienen las frases de saludo y las palabras comunes, y se encarga de la escritura del log y del aprendizaje.

5.7. Problemas

El propósito inicial del proyecto fue realizar una aplicación Java, con una interfaz gráfica que permitiera su ejecución embebida en una página web HTML.

A medida que avanzaba el desarrollo del proyecto nos dimos cuenta de la complicación de embeber una aplicación Java en una página web HTML. Principalmente, el detonante principal del cambio de concepto fue el poco soporte que los navegadores web ofrecen hoy en día a Java, siendo en algún caso inexistente como ocurre con Google Chrome.

Como problema añadido está la alta necesidad de Java para ejecutar un archivo *.jar*. Es necesario que la aplicación esté firmada,y aún así, sigue habiendo ciertos problemas para su ejecución por parte de un usuario inexperto.

Todo ello derivó en el cambio de requisito, de una aplicación Java a una aplicación web.

JSP fue la arquitectura elegida por el hecho de poder reutilizar el algoritmo CBR escrito en Java.

En un primer momento, traducir la aplicación desde la primera versión hacia JSP no parecía una tarea complicada, sin embargo, a la hora de proceder fue una tarea más larga y tediosa de lo esperado.

Fue necesaria una ampliación muy grande de los pocos conocimientos que se tenían sobre la programación web. La forma de interactuar con la interfaz HTML es radicalmente distinta a la experiencia con *Java Swing* por lo que se tuvieron que realizar grandes esfuerzos para obtener el mismo resultado que en la primera versión del proyecto.

La aplicación de los conceptos de redirección para realizar llamadas al servidor y su mezcla con la programación en el cliente mediante *JavaScript* fue una de las tareas más complicadas.

En cuanto al desarrollo de pruebas con Selenium, se encontraron varios problemas.

Uno de los problemas encontrados fue con la dependencia Guava. El framework jCOLIBRI2 utiliza una más versión antigua de la que necesitaba el WebDriver de Selenium para funcionar correctamente. Este problema fue más complicado de detectar y resolver de lo que puede parecer a primera vista ya que los errores que devolvía Selenium no eran muy descriptivos.

Otro de los problemas encontrados fue el uso de las variables de sesión el la aplicación. El WebDriver de Selenium al ejecutar las pruebas inicia una versión del navegador elegido que no es completa, esto ocasionaba que la sesión cambiase al realizar cualquier acción sobre la web, dejando al usuario

sin identificar. Además de con la herramienta de Selenium, este problema afectaba también al navegador Microsoft Edge que, aún teniendo las *cookies* activadas, no almacenaba la sesión correctamente. Este problema fue solucionado añadiendo la identificación de sesión a la URL mediante el método de *encodeURL + HTTPSession* en lugar de únicamente con *HTTPSession*. De esta forma el identificador de usuario es pasado en la URL impidiendo su variación.

5.8. Testing

La aplicación se ha testeado mediante test unitarios así como con test de interfaz.

Los test unitarios han sido los encargados de probar cada módulo de la aplicación por separado. Dentro de los test unitarios se han probado los métodos de las clases Java exceptuando el algoritmo de búsqueda de respuesta para lo que es necesario arrancar el servidor.

Por otra parte, se han realizado test de interfaz con Selenium. Se ha llevado a cabo un test de interfaz por cada requisito planteado, además de comprobar la funcionalidad básica de la web. Los test de interfaz se ejecutan sobre la página web y simulan el comportamiento de un usuario comprobando si el resultado obtenido de realizar una acción es la correcta. En este punto se comprueba también la integración de cada módulo del sistema, así como el correcto funcionamiento del algoritmo de búsqueda de respuesta. No es posible testear el algoritmo sin el servidor arrancado porque son necesarias dependencias que son cargadas dinámicamente desde el fichero *pom.xml* mediante Maven.

La realización de pruebas con la herramienta Selenium ha ocasionado varios problemas, ralentizando el desarrollo de los test de interfaz y por ente de la aplicación. Los problemas detallados pueden encontrase en la sección 5.7.

5.9. Estadísticas - Actualizar

A continuación se detallan algunas de las características más interesantes del proyecto. Para recoger las estadísticas no se han tenido en consideración los ficheros JSP en cuanto a estadísticas de código Java.

Las denominadas clases deseadas son aquellas que no forman parte del algoritmo de búsqueda de respuesta ya que, como se ha comentado, se necesitan dependencias dinámicas descargadas mediante Maven.

Los datos han sido obtenidos de las mediciones realizadas por RefactorIt y Eclipse.

Entidad	**Valor**
Número de clases Java	10
Número de archivos XML	8
Número de archivos JSP	16
Número de archivos HTML	6
Número de archivos CSS	10
Número de archivos JS	9
Total de líneas Java	1940
Líneas de código	54.7 %
Comentarios	25.5 %
Líneas en blanco	19.8 %
Número de test unitarios	15
Cobertura total test unitarios	36,6 %
Cobertura test unitarios clases deseadas	92.4 %
Número de test de interfaz	10

Tabla 5.1: Estadísticas del proyecto.

5.10. Documentación

Desde un primer momento se tuvo claro que a la hora de realizar la documentación la herramienta seleccionada sería LaTeX. Su elección se basó en la características que posee, como la alta calidad de texto formateado además de tener una plantilla disponible con todos los comandos especificados.

El editor de LaTeX seleccionado fue TexMaker, las razones fueron su extendido uso y su facilidad de uso, pudiendo trabajar con varios formatos como por ejemplo BibTex, extensión para definir la bibliografía.

5.11. Publicación - Por hacer

¿Hablar del servidor y la fase de producción?

Trabajos relacionados

Existen diferentes métodos de implementar un asistente virtual que permita el reconocimiento de texto. La amplia mayoría utilizan algoritmos de inteligencia artificial, siendo algunos de ellos los que se decantan por el razonamiento basado en casos.

6.1. Artículos de investigación

Comportamiento Adaptable de Chatbots Dependiente del Contexto

Artículo desarrollado por el grupo de investigación en sistemas de Información (GISI) de la Universidad Nacional de Lanús en Argentina, en el que se estudian los diferentes algoritmos utilizados en asistentes virtuales o *chatbots* centrándose en las mejoras que pueden introducirse en el reconocimiento del contexto. En el artículo se plantea el razonamiento basado en casos como una de las soluciones [38].

Un Sofbot de Charla Desarrollado con Técnicas de Razonamiento basado en Casos

En este artículo de la Revista de Investigación de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos se exponen las técnicas utilizadas para la construcción de un software de charla. El software construido utiliza la técnica de razonamiento basado en casos y se explican los distintos pasos realizados para su creación [33].

6.2. Proyectos

ChatBot

Es un proyecto *Open Source* que desarrolla una aplicación web donde se hospeda un chat construido mediante razonamiento basado en casos. Utiliza el framework de jCOLIBRI para llevarlo a cabo. Es un pequeño proyecto desarrollado por un estudiante de la Universidad de Auckland [1].

- Web del proyecto: https://github.com/agkphysics/ChatBot

SAMI

En el anterior artículo de investigación [33], se explican las técnicas seguidas para la realización de un software de charla basado en razonamiento basado en casos. El finalidad de este software consiste en proporcionar a los alumnos del aula virtual de la universidad información acerca de la misma. El software reconoce el diálogo escrito.

Este proyecto no dispone de página web y no ha sido encontrada más información a parte de la documentación.

6.3. Fortalezas y debilidades del proyecto

Las fortalezas de este proyecto radican en:

- La interfaz de usuario es muy sencilla.
- La interfaz es completa y se permite interactuar al usuario con ella.
- La aplicación es adaptable a cualquier ámbito solo con cambiar la base de datos.
- En cualquier momento es posible cambiar o retocar los casos, tanto descriptivos como respuesta.
- La aplicación dispone de una página de administración para consultar el log de uso y sugerencias de aprendizaje.

Por contra, existen ciertas debilidades como:

- La aplicación no se ha incluido en la página web de la Universidad de Burgos, hay que acceder a ella a través de otro sitio web.
- Actualmente solo está preparada para trabajar con casos que tratan sobre la Universidad de Burgos.
- No es todo lo precisa que un usuario podría desear, cubre los apartados más amplios de la Universidad.

Conclusiones y Líneas de trabajo futuras

Se procede a detallar las conclusiones que se derivan del desarrollo del proyecto y a realizar un informe crítico indicando cómo se puede mejorar el proyecto y cómo puede continuar su desarrollo.

7.1. Conclusiones

- Se ha conseguido realizar la idea propuesta al principio del desarrollo del proyecto. Los usuarios de la página web de la Universidad de Burgos pueden disponer, si lo desean, de un asistente que los ayude a orientarse dentro de la web, o a resolver ciertas dudas.
- El uso de un framework de inteligencia artificial como jCOLIBRI ha conseguido simplificar cierta parte del proceso de desarrollo, sin embargo, también a ocasionado otros problemas inesperados generando complejidad en aspectos que podrían no haberla tenido.
- Se han utilizado los conocimientos adquiridos durante la carrera, especialmente los adquiridos durante el último curso del grado, dificultando el desarrollo de la aplicación al cursar estas asignaturas de forma paralela.
- Se han ampliado conocimientos en lo relativo a la inteligencia artificial, desarrollo web, HTML, etc.
- Se han empleado ciertas herramientas que han aportado un valor añadido a la calidad del trabajo.
- El cambio de requisitos supuso un reto al tener que ampliar rápidamente conocimientos sobre técnicas que no eran conocidas en poco tiempo.
- Ha sido posible completar el proyecto en el plazo establecido.

7.2. Líneas de trabajo futuras

Este proyecto puede avanzar incluyendo distintas funcionalidades:

- Orientar la aplicación a más ámbitos dentro de la Universidad de Burgos, no solamente al guiado dentro de la propia web.
- Proporcionar reconocimiento de voz o comandos con los que poder interaccionar con la aplicación.
- Ofrecer una apariencia todavía más humana, realizando preguntas por ejemplo para refinar los resultados en caso de tener varios disponibles.
- Conseguir responder a preguntas de ámbito general aunque éstas no tengan relación con la Universidad. Por ejemplo otorgar la previsión meteorológica.
- Implementar la aplicación para que esté disponible en más plataformas.

Bibliografía

[1] A chat bot using CBR (case based reasoning) techniques. URL https://github.com/agkphysics/ChatBot. Accedido el 2017-05-07.

[2] jCOLIBRI — GAIA – Group of Artificial Intelligence Applications, . URL http://gaia.fdi.ucm.es/research/colibri/jcolibri. Accedido el 2017-05-06.

[3] COLIBRI — GAIA – Group of Artificial Intelligence Applications, . URL http://gaia.fdi.ucm.es/research/colibri. Accedido el 2017-05-05.

[4] COLIBRI Studio — GAIA – Group of Artificial Intelligence Applications, . URL http://gaia.fdi.ucm.es/research/colibri/colibristudio. Accedido el 2017-05-06.

[5] EclEmma - Java Code Coverage for Eclipse, . URL http://www.eclemma.org/. Accedido el 2017-05-06.

[6] Eclipse (software) - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/Eclipse{_}(software). Accedido el 2017-05-06.

[7] Git - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/Git. Accedido el 2017-05-06.

[8] GitHub - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/GitHub. Accedido el 2017-05-06.

[9] Google Core Libraries for Java. URL https://github.com/google/guava. Accedido el 2017-06-07.

[10] Hibernate ORM - Hibernate ORM. URL http://hibernate.org/orm/. Accedido el 2017-05-05.

[11] Java (lenguaje de programación) - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Java{_}(lenguaje{_}de{_}programaci{\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\penalty\@M\hskip\z@skip}n). Accedido el 2017-05-05.

[12] JSP arquitectura, . URL http://www.w3ii.com/es/jsp/jsp{_}architecture.html. Accedido el 2017-05-07.

[13] JSP procesamiento de formularios, . URL http://www.w3ii.com/es/jsp/jsp{_}form{_}processing.html. Accedido el 2017-05-07.

[14] JSP Tutorial, . URL https://www.tutorialspoint.com/jsp/. Accedido el 2017-05-06.

[15] JavaServer Pages - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/JavaServer{_}Pages. Accedido el 2017-05-06.

[16] JUnit - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/JUnit. Accedido el 2017-05-06.

[17] LaTeX - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/LaTeX. Accedido el 2017-05-06.

[18] Maven - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Maven. Accedido el 2017-05-06.

[19] Mendeley - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Mendeley. Accedido el 2017-05-06.

[20] Modelo–vista–controlador - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Modelo\OT1\textendashvista\OT1\textendashcontrolador. Accedido el 2017-05-05.

[21] MySQL - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/MySQL. Accedido el 2017-05-06.

[22] Refactorit download — SourceForge.net. URL https://sourceforge.net/projects/refactorit/. Accedido el 2017-05-06.

[23] Introducción — Scrum 1 documentation, . URL http://metodologiascrum.readthedocs.io/en/latest/Scrum.html. Accedido el 2017-05-05.

[24] What is Scrum?, . URL https://www.scrum.org/resources/what-is-scrum. Accedido el 2017-05-05.

[25] Selenium Library, . URL http://www.seleniumhq.org/download/. Accedido el 2017-06-07.

[26] Selenium - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/Selenium. Accedido el 2017-05-06.

[27] Trail: Creating a GUI With JFC/Swing (The Java™ Tutorials). URL http://docs.oracle.com/javase/tutorial/uiswing/. Accedido el 2017-05-06.

[28] Tomcat - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Tomcat. Accedido el 2017-05-06.

[29] Razonamiento basado en casos - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/Razonamiento{_}basado{_}en{_}casos. Accedido el 2017-05-05.

[30] SonarQube - Wikipedia, la enciclopedia libre, . URL https://es.wikipedia.org/wiki/SonarQube. Accedido el 2017-06-16.

[31] Programación extrema - Wikipedia, la enciclopedia libre. URL https://es.wikipedia.org/wiki/Programaci{\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{o\global\mathchardef\accent@spacefactor\spacefactor}\accent19o\egroup\spacefactor\accent@spacefactor\futurelet\@let@token\penalty\@M\hskip\z@skip}n{_}extrema. Accedido el 2017-05-05.

[32] Daniel Santidrián Alonso. Repositorio de UBUassistant en GitHub, 2017. URL https://github.com/DanielSantidrian/UBUassistant. [Internet; Accedido 02-mayo-2017].

[33] Shanir Majayra CAMACHO JARA. Un Sofbot de Charla Desarrollado con Técnicas de Razonamiento basado en Casos, 2006. URL http://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/4524/3604.

[34] Laura Lozano and Javier Fernández. Razonamiento Basado en Casos:"Una Visión General". pages 1–59, 2008. URL http://www.infor.uva.es/{~}calonso/IAI/TrabajoAlumnos/Razonamientobasadoencasos.pdf.

[35] Juan Pavón Mestras. Aplicaciones Web/Sistemas Web. URL https://www.fdi.ucm.es/profesor/jpavon/web/44-jsp.pdf.

[36] Juan a Recio-García, Belén Díaz-Agudo, and Pedro González-Calero. *jCOLIBRI2 Tutorial*. 2008. ISBN 9788469162040.

[37] Carlos López Nozal. Rodrigo Manjón Martín. *Introduccion a los patrones de diseño.pdf*. Burgos, 2017.

[38] Juan Manuel Rodríguez, Hernán Merlino, and Enrique Fernández. Comportamiento Adaptable de Chatbots Dependiente del Contexto. 2(2): 115–136, 2014. URL http://sistemas.unla.edu.ar/sistemas/gisi/papers/relais-v2-n2-115-136.pdf.