Universidad Zaragoza

Trabajo Fin de Grado

Desarrollo de un sistema recomendador de cervezas basado en ontologías y lógica difusa

Development of an ontology and fuzzy logic based beer recommendation system

Autor

Fernando Alegre Martínez

Director

Fernando Bobillo Ortega

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Fernando Alegre Martínez,

con nº de DNI 73029510-W en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado de ingeniería informática, (Título del Trabajo) Desarrollo de un sistema recomendador de cervezas basado en ontologías y lógica difusa,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 31 de octubre de 2017

Fdo: Fernando Alegre Martínez

Desarrollo de un sistema recomendador de cervezas basado en ontologías y lógica difusa

RESUMEN

Hoy en día, en el que todo el mundo lleva un smartphone en el bolsillo y está de moda el consumo de diferentes y exóticos tipos de cerveza, florecen las apps que permiten al usuario obtener información acerca de todas y cada una de ellas. Estas apps ofrecen muchas posibilidades, como el poder reconocer una cerveza con solo hacerle una foto a la etiqueta, compartir con otros usuarios las puntuaciones o qué cervezas se han probado recientemente, pero ninguna permite filtrar según los deseos del usuario.

Este trabajo está orientado a esta funcionalidad en concreto, basándose en una ontología que emplea lógica difusa para tratar con las imprecisas preferencias de los usuarios y un razonador semántico para extraer la información necesaria de la ontología.

Con dicho objetivo, a lo largo de este trabajo se han intentando utilizar diferentes razonadores portados para ser utilizados en dispositivos móviles, poniendo de manifiesto el punto de inmadurez en el que se encuentra esta tecnología, siendo imposible su uso en este tipo de dispositivos.

Como resultado de este trabajo, queda la ontología de cervezas, con una jerarquía de clases bastante trabajada y muy fiel a la realidad y con una gran cantidad de individuos (15.317), además de la app propiamente dicha, que utiliza técnicas de lógica difusa como elemento innovador y que se diferencia de manera importante de las demás apps del mercado.

Índice

1. Introducción

Actualmente, con el auge de la inteligencia artificial y el uso de las preferencias del usuario para ofrecerle una experiencia lo más cercana y personalizada posible, parece el entorno perfecto para el desarrollo de aplicaciones basadas en ontologías y lógica difusa. Las ontologías permiten concretar y resumir el área de conocimiento para que los razonadores puedan explotar todas sus ventajas, pero aunque en muchos aspectos sí que se están utilizando, no parece tener tirón en el tan amplio e interesante mundo de la cerveza. Tanto es así, que mientras que sí que hay aplicaciones para obtener información acerca de una determinada cerveza, no hay nada en el mercado, y menos en el sector de la computación móvil, capaz de aplicar todos estos conceptos como sí ocurre en otros ámbitos como el del vino.

Así pues, a lo largo del presente trabajo se plantea resolver este problema cubriendo un nicho de mercado todavía inexplorado. Para ello, es necesario experimentar con los razonadores actuales, ya que estos no están pensados, ni mucho menos optimizados, para su uso en móviles. La resolución de este problema, ofrecería resultados interesantes sobre el uso de razonadores semánticos en un entorno en el que no estamos acostumbrados a verlos, abriendo así una nueva vía de posibilidades para la computación móvil. Desafortunadamente, los resultados obtenidos ponen de manifiesto que, o la tecnología móvil no está preparada para soportar el nivel de computación necesario, o que aún queda mucho trabajo por delante con los razonadores para que estos puedan ser portados a otras plataformas más ligeras.

La memoria del trabajo está organizada de la siguiente manera:

- Un primer apartado de preliminares, en el que se introducen y explican algunos conceptos que pueden resultar desconocidos y que son utilizados a lo largo del documento.
- Una vez explicados estos conceptos, se pasan a detallar los distintos aspectos que se han desarrollado durante el proyecto, como son:
    - La ontología: sección dividida en la jerarquía de clases establecida con lo estilos de cerveza, las propiedades seleccionadas como relevantes, la obtención de los distintos individuos que la pueblan y el uso que se le da a la lógica difusa en ella.
    - Los razonadores utilizados, los resultados obtenidos con cada uno de ellos y las implicaciones que estos resultados han tenido en relación al proyecto.
    - Y, por último, el desarrollo de la aplicación móvil, en el que se plasma todo lo investigado y diseñado sobre la ontología y los razonadores a usar.

Este apartado está dividido según la arquitectura del sistema, detallando a continuación la parte del cliente y la del servidor por separado para terminar explicando el papel que juega la lógica difusa en la app y sus requisitos.

- Una vez explicadas las aportaciones del proyecto, estas son contextualizadas en una sección llamada "Trabajo relacionado", en la que se habla de cómo se sitúa este trabajo respecto a otros trabajos similares, tanto en el ámbito de las cervezas como en el mercado del vino, sector en el que se encuentran las mayores similitudes.

- Tras haber expuesto tanto el trabajo realizado como el contexto en el que se lleva a cabo, comento las conclusiones que extraigo valorando los resultados obtenidos a nivel personal y a nivel técnico.

- A continuación aparecen las referencias bibliográficas que son citadas a lo largo del texto.

- Cerrando el documento se encuentran los anexos, divididos en 4 apartados, que son: Ingeniería del software (diagramas que describen la app), Manual de usuario de la app, Temporización y planificación del proyecto y Pseudocódigo, que ilustra el funcionamiento de algunas partes del sistema.

2. Preliminares

A continuación se explican algunos conceptos relacionados con el proyecto. Estos están ordenados siguiendo el orden de aparición en el documento.

Ontología. Una ontología es un sistema de representación del conocimiento que resulta de seleccionar un dominio o ámbito del conocimiento, y aplicar sobre él un método con el fin de obtener una representación formal de los conceptos que contiene y de las relaciones que existen entre dichos conceptos. Las ontologías introducen un mayor nivel de profundización semántica y proporcionan una descripción lógica y formal que puede ser interpretada tanto por las personas, como por las máquinas. [1]

Dentro de una ontología, hay elementos importantes que también aparecen en el texto.

- **Clases:** Conjuntos, colecciones o tipos de objetos o cosas existentes en la ontología.
- **Individuos:** Cada uno de los objetos, instancias o cosas que pueblan la ontología y forman las clases.
- **Data properties:** Propiedad de los individuos cuyo valor es un datatype, como por ejemplo: enteros, reales, strings...
- **Object properties:** Propiedad de los individuos cuyo valor es otro individuo de la ontología.
- **Axioma:** Condición formal sobre elementos de la ontología que debe verificarse siempre para preservar la semántica.

Razonador semántico. Un razonador semántico es un software capaz de obtener información tanto explícita como implícita de una ontología, haciendo así su capacidad deductiva mucho mayor que la de las bases de datos convencionales.

ABV. Las siglas ABV vienen de las palabras en inglés *"Alcohol By Volume* e indican el porcentaje de alcohol que contiene un líquido.

IBU. Las siglas IBU también vienen del inglés, en concreto de *"International Bitterness Unit"* y mide el valor de amargura de una bebida. En el caso de la cerveza, estos valores van desde casi el 0 hasta rozar los 120 en algunos casos.

API. La interfaz de programación de aplicaciones, abreviada como API del inglés: *Application Programming Interface*, es un conjunto de subrutinas, funciones y

procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. [2]

Ale. Ale es un nombre que abarca a todas las cervezas de fermentación alta. Esto quiere decir que en las ales, el proceso de fermentación ocurre en la superficie del líquido. Fermentan rápidamente a temperaturas entre 15 y 25℃ y se sirven, por lo general a una temperatura de 12℃ o más. Estas suelen tener una mayor graduación, un color más oscuro y ser algo más densas. [3]

Lager. Lager es un tipo de cerveza que se sirve fría, caracterizada por fermentar en condiciones más lentas utilizando levaduras de fermentación baja y que en las últimas partes del proceso son almacenadas en bodegas a baja temperatura. Son las cervezas más claras y con una menor graduación alcohólica. [4]

Lógica difusa. La lógica difusa es una técnica de la inteligencia computacional que permite trabajar con información con un alto grado de imprecisión, vaga, ambigua o incompleta. Es una lógica multivaluada que permite valores intermedios para poder definir evaluaciones entre sí/no, verdadero/falso/, negro/blanco, etc. [5]

La lógica difusa se caracteriza por la inclusión del concepto de conjunto difuso, conjunto que puede contener elementos de forma parcial, es decir, que la propiedad de que un elemento pertenezca al conjunto puede ser cierta con un grado parcial de verdad. La función de pertenencia a este conjunto puede ser calculada de diversas formas.

Funciones *shoulder*. Una función *left-shouder* es una función cuyos valores comienzan en 1 hasta que llegan a un punto en el que comienzan a decrecer de forma constante hasta llegar a 0. A partir de este punto, todos los valores son nulos. Esta función también puede ser simétrica (*right-shoulder*), comenzando en 0 hasta cierto punto en el que empieza a subir y, una vez que llega al valor máximo, lo mantiene indefinidamente.

Función triangular. Una función triangular es una función cuyos valores comienzan en 0 hasta llegar a un punto en el que aumentan de forma constante hasta llegar a un máximo en el que comienzan a decrecer de nuevo de forma constante hasta llegar de nuevo a 0.

Clustering. Un algoritmo de agrupamiento (en inglés, *clustering*) es un procedimiento de agrupación de una serie de vectores de acuerdo con un criterio. Esos criterios son por lo general distancia o similitud. La cercanía se define en términos de una determinada función de distancia, como la euclídea, aunque existen otras

más robustas o que permiten extenderla a variables discretas. Generalmente, los vectores de un mismo grupo (o clústers) comparten propiedades comunes. [6]

K-means. K-means es un método de agrupamiento, que tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo centroide es más cercano. Es un método utilizado en minería de datos. [7]

Dado un conjunto de observaciones $(x_1, x_2, \ldots, x_n)$, donde cada observación es un vector real de d dimensiones, k-means construye una partición de las observaciones en k conjuntos ($k \leq n$) a fin de minimizar la suma de los cuadrados dentro de cada grupo (WCSS): S = $S_1, S_2, \ldots, S_k$

$$\arg\min_{s} \sum_{i=1}^{k} \parallel x_j - \mu_i \parallel^2$$

donde μ_i es el centroide de S_i.

OWL 2. OWL 2 es una evolución de OWL con mayores prestaciones semánticas que lo han convertido en uno de los estándares a la hora de trabajar con ontologías. OWL es el acrónimo del inglés Web Ontology Language, un lenguaje de marcado para publicar y compartir datos usando ontologías en la WWW. OWL tiene como objetivo facilitar un modelo de marcado construido sobre RDF, pudiendo estar condificado con diferentes sintaxis como, por ejemplo, XML. [8]

ART. Android Runtime (ART) es un entorno de ejecución de aplicaciones utilizado por el sistema operativo móvil Android. ART reemplaza a Dalvik, que es la máquina virtual utilizada originalmente por Android, y lleva a cabo la transformación de la aplicación en instrucciones de máquina, que luego son ejecutadas por el entorno de ejecución nativo del dispositivo. [9]

Threads. Los *threads* o hilos, en español, son los procesos en los que se ejecuta una aplicación en Java. Todos los programas en Java constan de al menos un thread principal creado por la máquina virtual de Java en el que se ejecuta el código, sin embargo, este comportamiento puede ser alterado para que existan múltiples threads que comparten los recursos de la máquina sobre la que están corriendo.

Socket. Socket designa un concepto abstracto por el cual dos programas pueden interactuar cualquier flujo de datos, generalmente de manera fiable y ordenada. [10]

Protocolo TCP. Protocolo de control de transmisión (en inglés *Transmission Control Protocol*) que garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron. También proporciona un mecanismo para distinguir distintas aplicaciones dentro de una misma máquina, a través del concepto de puerto. [11]

Operador OWA. En matemáticas aplicadas, especialmente en lógica difusa, los operadores OWA (*ordered weighted averaging*) son una clase de operadores de agregación de tipo media parametrizados. Esto permite asignar pesos distintos a los valores α_i que se pretenden agregar.

- Sea σ una permutación de $\{1, \ldots, n\}$ que ordena los α_i, es decir: $\alpha_{\sigma(1)} \geq \alpha_{\sigma(2)} \geq \ldots \geq \alpha_{\sigma(n)}$
- Sea un vector de pesos $W = [w_1, \ldots, w_n]$ con $w_i \in [0, 1]$ y $\sum_{i=1}^{n} = 1$
- Un operador OWA se define como:

$$@_{OWA}^{W}(\alpha_1, \ldots, \alpha_n) = \sum_{i=1}^{n} w_i \alpha_{\sigma(i)}$$

- Cada peso w_i no se asigna a un argumento concreto, sino al que ocupe un orden determinado. [12]

3. Ontología

La ontología es la piedra angular sobre la que gira todo el trabajo el aspecto más novedoso del mismo. Novedoso porque no hay, o al menos no he podido encontrar, una ontología sobre cervezas que recoja la jerarquía completa de estilos adecuadamente confeccionada así como una muestra representativa de individuos con datos sobre cada uno de ellos. Además, esta es una ontología que emplea lógica difusa para catalogar las cervezas en diferentes etiquetas lingüísticas relacionadas con características propias de las cervezas, como son el grado de alcohol y el grado de amargura.

3.1. Jerarquía de clases

La jerarquía de clases final ha sido el resultado de ampliar y mejorar jerarquías y clasificaciones existentes a lo largo de Internet. En particular, de un borrador realizado por la universidad de Maryland [13] primero, y de la página *RateBeer* [14] después. Del borrador extraje una primera idea de cómo podía ser la distribución final de los estilos salvando importantes diferencias. De la página web *RateBeer* obtuve todos los estilos que han acabado formando la jerarquía completa, aunque también sujetos a cambios, ya que estos no estaban completamente clasificados y distribuidos, simplemente se encontraban agrupados respecto a algunas etiquetas, sin ningún tipo de jerarquía ni orden entre ellos. Una vez tenía un primer borrador y todos los estilos que iban a ser incluidos en la ontología, fue cuestión de ir buscando información acerca de todos ellos hasta poder clasificarlos de la manera más coherente posible. El proceso de modificación manual de las clases fue llevado a cabo con *Protegé* [15].

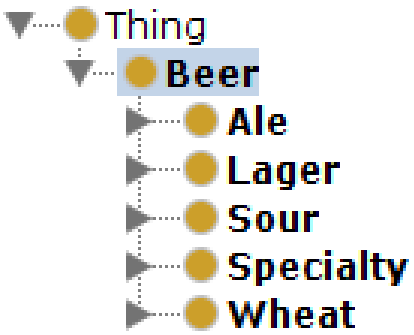


Figura 1: Principales clases de la jerarquía

Así pues, el resultado final de la distribución de los estilos (Figura 1) es el siguiente: todas las cervezas están clasificados en 5 grandes grupos, que agrupan el resto de las 88 clases que componen la ontología, sumando un total de 93 clases diferentes organizadas hasta en 5 niveles de profundidad. Los dos primeros grandes grupos, disjuntos entre sí, son los formados por las cervezas *Ale* y *Lager*. Estas dos clases están claramente

identificadas en todas las clasificaciones de cervezas debido a sus diferencias en el proceso de fermentación que tienen una repercusión directa tanto en sus sabores como en sus apariencias. Además de estas dos grandes familias, he considerado adecuado señalar otros 3 grandes grupos. Si bien no tan obvios como los dos primeros, pero con importancia suficiente como para hacer esta distinción. Los otros tres grandes grupos que quedan son los conformados por las cervezas de trigo (*Wheat*), las cervezas cuya característica principal es su sabor ácido y agrio (*Sour*) y, por último, todas aquellas un poco más inclasificables por estar hechas, por ejemplo, con frutas, con especias exóticas o por haber sido ahumadas (*Specialty*). Las cervezas de trigo suelen ser del tipo Ale por su fermentación alta, e incluso las hay que están catalogadas dentro de la categoría *Sour*, como es el caso del estilo *Berliner Weisse*. En ambos casos no se considera conveniente usar herencia múltiple no porque no se pueda, sino porque los propios expertos hacen la distinción entre *Wheat* y *Ale*. Como ejemplo de estilo de cerveza con herencia múltiple, están las *Weizen Bock*, cervezas de trigo de fermentación baja pertenecientes a la familia de las *Wheat*, pero también catalogadas como cervezas de tipo *Bock*, subclase de las *Lager*.

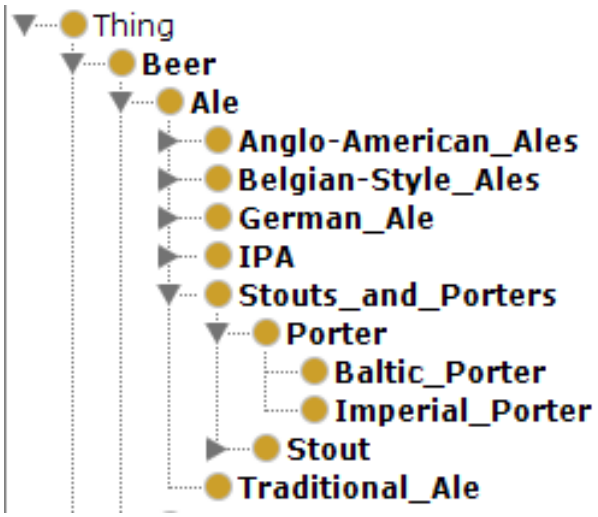


Figura 2: Ejemplo de organización de la clase *Ale*

3.2. Propiedades

Una vez establecida una jerarquía de clases consistente, el siguiente paso para seguir con la creación de la ontología era definir las propiedades disponibles. Desde el punto de vista del análisis, hay multitud de propiedades que pueden resultar interesantes a la hora de tratar con una ontología sobre cervezas y que aportan información de valor sobre ellas. Las características que fueron tenidas en cuenta en un principio fueron: grado de alcohol (ABV), grado de amargura (IBU), aroma, cervecería, color,

tipo de fermentación, sabor, cantidad de espuma, si es industrial o artesana y el país de procedencia de cada cerveza. Como vemos en la Figura 3, todas ellas figuran como data properties menos el país de procedencia, que se pensó que sería mejor representarlo mediante una object property. Otra posibilidad, habría sido definir el aroma, el color y el sabor como object properties también, de manera que el vocabulario utilizado estuviera más acotado y controlado, pero al no saber qué iba a encontrar durante la fase de diseño, se quedó como una idea.

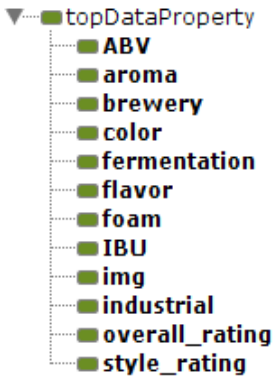


Figura 3: Data properties incluidas en la ontología

Desgraciadamente, no todas estas propiedades son fáciles de definir para una amplia muestra de cervezas, por lo tanto, a la hora de poblar la ontología no se consiguieron todos los datos deseados.

Finalmente, las propiedades empleadas estuvieron condicionadas por *RateBeer* [14], portal del que fueron extraídos prácticamente el total de los individuos que componen la ontología. De esta manera, me aproveché de la información que guardaba dicho portal sobre cada cerveza para utilizar las propiedades de ABV, IBU y cervecería que ya tenía planteadas durante la fase de diseño y, además, añadir las propiedades de imagen(una url con una imagen de la cerveza), puntuación global de la cerveza y puntuación de dicha cerveza entre los demás individuos de su mismo estilo.

Las propiedades de IBU, tipo de cervecería y país de procedencia no están definidas para todos los individuos, mientras que ABV, cervecería y url de la imagen sí.

```
Individual: <http://beerOntology.es/Alhambra_Reserva_Roja>

    Types:
        <http://beerOntology.es/Doppelbock>

    Facts:
        <http://beerOntology.es/ABV>  7.2,
        <http://beerOntology.es/IBU>  19,
        <http://beerOntology.es/brewery>  "Alhambra (Grupo Mahou-San Miguel)",
        <http://beerOntology.es/industrial>  true,
        <http://beerOntology.es/overall_rating>  13,
        <http://beerOntology.es/country>  <http://beerOntology.es/Spain>,
        <http://beerOntology.es/style_rating>  4,
        <http://beerOntology.es/img>  "https://res.cloudinary.com/ratebeer/image/uploa
```

Figura 4: Ejemplo de un individuo de la ontología

3.3. Individuos

Como ya he comentado anteriormente, los individuos que pueblan la ontología, así como los valores de sus propiedades están sacados del portal *RateBeer* [14]. Para ello, ya que la API del sitio estaba en construcción hasta hace unas pocas semanas, fue necesario el uso de unos scripts [16] en Python. Estos scripts me permitieron crear mi propio código, también en Python, para descargar la información sobre cada una de las cervezas y volcarla en un fichero con el formato adecuado para pasar a formar parte de la ontología utilizando la sintaxis Manchester [17] como muestra la Figura 4. Más concretamente utilicé los métodos:

- *beer_style_list*: Devuelve los nombres de cada uno de los estilos de cerveza existentes en *Ratebeer* junto con su número identificador correspondiente.

- *beer_style*: Con los identificadores de los estilos obtenidos previamente, devuelve una lista con los urls de las cervezas del estilo indicado ordenado ya sea por puntuación total, por el número de valoraciones por parte de los usuarios o por ABV.

- *get_beer*: Una vez obtenidas las urls de todas las cervezas, este método devuelve un objeto "Cerveza" con todos los atributos con los que cuenta en RateBeer.

Con todos los datos de las cervezas descargados y formateados de manera que pudiera trabajar con ellos desde el editor Protegé, fue necesaria una fase de limpieza o *cleaning*, puesto que había muchos caracteres extraños que el editor no reconocía. Entre estos caracteres se encontraban, por ejemplo, los acentos eslavos, como son ý, č, ö... e incluso otros caracteres que, por su codificación, ni siquiera podían ser mostrados y cuya búsqueda, de forma manual, resultó bastante tediosa y frustrante.

A lo largo del proyecto, he trabajado con 2 versiones de la ontología. Una versión ligera, con 4.068 cervezas durante las versiones más tempranas y una completa, con un total de 15.317 cervezas y 79.159 asertos del tipo data property. De todas estas cervezas, el 99 % ha sido obtenido directamente de *RateBeer* mediante los scripts citados anteriormente. El 1 % restante provienen de una colección privada de cervezas más accesibles desde nuestro país que, al no ser relevantes en Estados Unidos en ninguno de los aspectos comentados anteriormente (puntuación, número de puntuaciones y ABV), no fueron devueltos durante la búsqueda inicial. Para la inclusión de estas cervezas en la ontología, que estaban en formato html, hice otro script que obtenía los nombres de los individuos, extraía de *RateBeer* sus datos y los escribía en un fichero adecuadamente formateado, listos para ser añadidos a la ontología.

En la recta final del proyecto, con la intención de añadir más cervezas españolas a la ontología, trabajé también con la API de *RateBeer* [18], ya que me restringieron el acceso a sus datos con los scripts con los que trabajé desde un principio. Esta API utiliza tecnología GraphQL [19], un lenguaje de consultas orientado a ser usado por API's, que devuelve la información en formato JSON. De esta manera, conseguí extraer cerca de 2.000 cervezas elaboradas por las cervecerías españolas más importantes. Aunque esta API es bastante sencilla de usar, el acceso a los datos también está limitado. No permiten más de 5.000 peticiones mensuales ni permiten realizar consultas demasiado amplias sobre su base de datos, de manera que recuperar un gran número de individuos no puede hacerse de forma totalmente automatizada.

Como apunte final, cabe decir que, al estar sacadas la inmensa mayoría de las cervezas de un portal de origen estadounidense, no todas son comunes o fáciles de localizar en España. Las cervezas cuya producción ha sido retirada no están incluidas en el catálogo por la dificultad de conseguirlas.

3.4. Lógica difusa

La lógica difusa es aplicada a la ontología dividiendo los posibles valores de alcohol(ABV) y amargura(IBU) en 5 etiquetas lingüísticas, incluidas en la ontología como "datatypes difusos" con el plugin *FuzzyOWL* [20] de Protegé [15]. En concreto, como datatypes clásicos con anotaciones que codifican la parte difusa, de modo que las herramientas clásicas como los razonadores ignoran esa parte y la tratan como una ontología clásica más. Estas etiquetas son, en ambos casos, *"Muy poco"*, *"Poco"*, *"Moderado"*, *"Mucho"* y *"Muchísimo"*. Las dos etiquetas extremas, *"Muy poco"* y *"Muchísimo"* son sendas funciones *shoulder*, mientras que las etiquetas centrales son funciones triangulares como muestra la Figura 5.

Para obtener los valores de corte de las etiquetas para los dos atributos, escribí un script en Python utilizando librerías de clasificación y dibujo de gráficos propias del lenguaje. Concretamente, utilicé el algoritmo de clustering k-means para localizar cinco clústers entre el total de valores de ABV e IBU, cuyos centroides son los parámetros que caracterizan las funciones que aparecen en la Figura 5. La razón de que fueran 5 etiquetas lingüísticas, es que es un número de valores natural para los humanos y usual en lógica difusa. Además, el algoritmo k-means para obtener los centroides fue ejecutado varias veces para observar que no hubiera grandes diferencias entre ejecuciones y, salvo casos excepcionales en los que el resultado de una ejecución podía alejarse bastante de las demás, el resultado de las ejecuciones era bastante homogéneo.

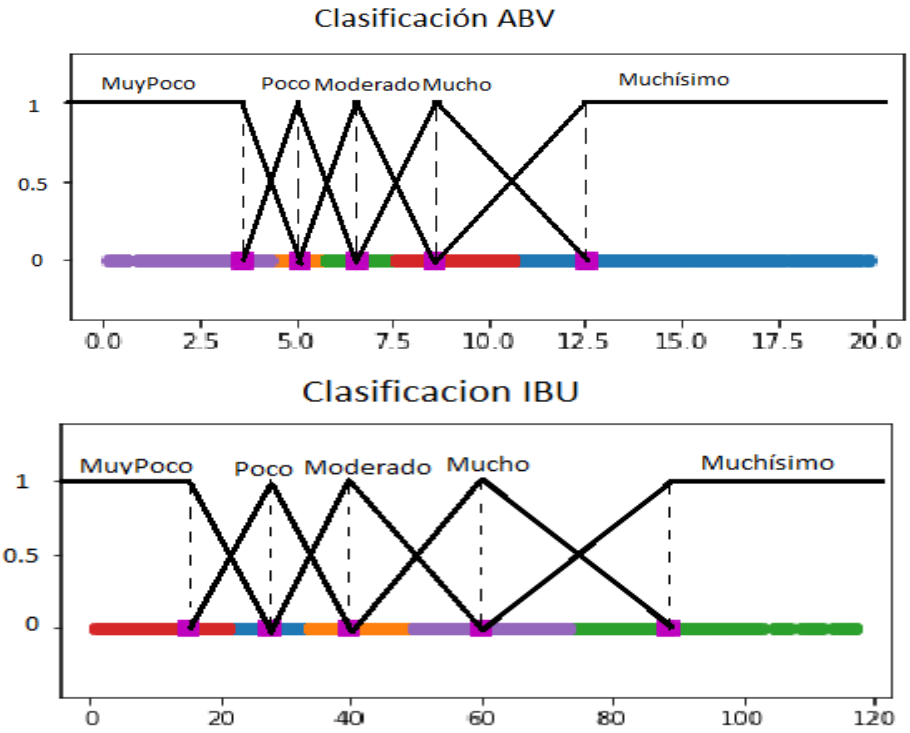
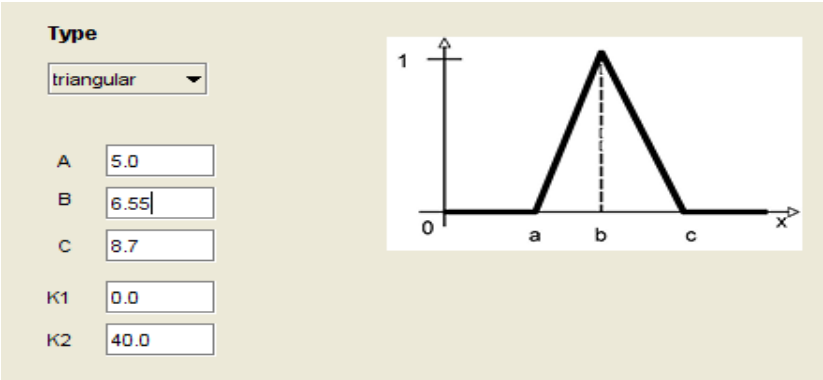


Figura 5: Clasificaciones de ABV e IBU



(a) Representación del datatype en el plugin *FuzzyOWL*

```
Datatype: <http://beerOntology.es/#ModeradoABV>

    Annotations:
        <http://beerOntology.es/#fuzzyLabel> "<fuzzyOwl2 fuzzyType=\"datatype\">
<Datatype type=\"triangular\" a=\"5.0\" b=\"6.55\" c=\"8.7\" />
</fuzzyOwl2>"

    EquivalentTo:
        (xsd:double[>= 0.0] and xsd:double[<= 40.0])
```

(b) Representación del datatype en la ontología

Figura 6: Representaciones de los datatypes difusos

4. Razonadores

Si bien la construcción de la ontología fue un proceso largo y vital para la consecución de los objetivos del proyecto, esta no tendría ningún sentido sin la existencia de un razonador capaz de extraer información de los datos almacenados en ella. El principal problema a este respecto es que no hay todavía ningún razonador diseñado específicamente para trabajar en Android disponible, así que la única posibilidad que queda es utilizar los razonadores que han sido portados de manera no oficial. Así pues, comencé a probar con los razonadores portados por el grupo de Sistemas de Información Distribuidos de la Universidad de Zaragoza [21] [22].

El uso del razonador amplía la potencia deductiva de que dispone la app a la hora de obtener información de la ontología. Por ejemplo, si queremos hacer una búsqueda sobre las cervezas de tipo *lager*, porque nos gusta la cerveza *Ámbar*, la *Ámbar* normal (*Ámbar Especial*), está catalogada como una *Premium Lager*, sin embargo, al ser este estilo una subclase de *Lager*, el razonador es capaz de devolver tanto la *Ámbar Especial* como el resto de cervezas que pertenecen a todos los estilos que forman el conjunto de las *Lager*.

Un diagrama de secuencias de la comunicación que se realiza entre el servidor y el cliente está disponible en el apartado A.4 de los apéndices. En él está representado el caso de una búsqueda simple, que es exactamente igual en el caso de los otros dos tipos de búsqueda en cuanto al esquema se refiere.

4.1. *HermiT*

El razonador *HermiT* fue el primer razonador de los que están portados con el que probé, por ser uno de los más famosos y reputados y por estar incluido en *Protegé* [15]. *HermiT* además, soporta la versión completa de OWL 2 y fue el primer razonador capaz de clasificar grandes ontologías, así que parecía ser el más indicado para el trabajo de clasificar todos los individuos de la ontología de cervezas.

Tras investigar un poco y hacer algunas pruebas con la ontología reducida citada en el apartado anterior, conseguí obtener los resultados esperados para las consultas que le estaba haciendo, como verificar que la ontología era consistente o comprobar si una clase era subclase de otra clase dada.

El principal y único problema que presentó el rendimiento de este razonador fue el tiempo que requería para realizar cualquier cosa. Ya con la versión reducida de únicamente 4.000 cervezas, tardaba en cargar la ontología alrededor de 7 segundos y para terminar de clasificar el total de individuos requería entre 42 y 45 segundos. El terminal con el que realicé estas pruebas se trata de un Xiaomi Redmi 3, lanzado

en enero de 2016. Cuenta con 2GB de memoria RAM y un procesador *Qualcomm MSM8939v2 Snapdragon 616* octacore, con 4 núcleos funcionando a 1.5GHz y los otros 4 a 1.2GHz. Con la versión completa de la ontología, era necesario esperar entre 4.5 y 5 minutos para que terminara de clasificar todos los individuos. Obviamente, para el uso que se le pretendía dar en una app móvil, este coste de tiempo era algo completamente imposible de asumir, por mucho que las consultas fueran luego respondidas en unos pocos segundos (alrededor de 8-9).

También realicé pruebas utilizando el emulador de Android en el ordenador para comprobar si era un problema de falta de potencia en el terminal. Las especificaciones del ordenador en cuestión, son un procesador *Intel i7-4510U* de dos núcleos físicos y 4 lógicos, capaz de correr a 2.6GHz y 12GB de memoria RAM. El dispositivo emulado fue un Google Pixel. Móvil de alta gama lanzado al mercado en Octubre de 2016. Cuenta con un procesador *Qualcomm Snapdragon 821 (MSM8996 Pro)* de 4 núcleos, con dos corriendo a 2.15GHz y los otros dos a 1.6Ghz. Los resultados obtenidos fueron incluso algo peores que los obtenidos en el dispositivo físico mencionando antes, lo que confirmaba la imposibilidad de usar el razonador *HermiT* en el móvil.

4.2. *Pellet*

Pellet, era el otro razonador prometedor junto con *HermiT*. También está implementado en Java y soporta la versión completa de OWL 2. Fue el primer razonador de lógica descriptiva capaz de soportar OWL.

El problema con este razonador fue que, aunque en su día estuviera portado con éxito, con las nuevas versiones de Android no funciona. Intenté hacerlo funcionar corrigiendo algunos errores manejables que me fueron surgiendo, como por ejemplo un método que compartía nombre con un nuevo método de Java pero no lo sobrescribía o que al importarlo desde el proyecto de AndroidStudio [23], excedía el número de métodos permitidos por la máquina *ART*. Por desgracia, llegué a un punto en el que no sabía corregir el error que obtenía al ejecutarse la primera línea de código referente al razonador, así que el uso del razonador *Pellet* quedó también descartado.

4.3. Otros razonadores

Puesto que la ontología cumple los requisitos necesarios para ser considerada como parte del perfil OWL 2 EL [24], subconjunto más restrictivo de OWL que sacrifica potencia expresiva a cambio de un mayor rendimiento computacional, se planteó la posibilidad de utilizar algún razonador específico de este perfil. Sin embargo, esta idea tuvo que ser rechazada dado que los principales razonadores disponibles, como son *ELK*

[25], *TrOWL* [26] y *jcel* [27], no soportan el tipo de axiomas *data property assertions*, vitales para este proyecto y reflejadas en las Figuras 3 y 4 (tras la palabra reservada *"Facts"*), puesto que todas las propiedades de las cervezas están representadas de esta manera.

4.4. *HermiT* en el ordenador

Finalmente, y tras el fracaso, en mayor o menor medida, de los razonadores anteriores en la plataforma móvil, decidí probar con el razonador *HermiT* de nuevo, pero esta vez ejecutándolo desde el ordenador. Los resultados en esta ocasión fueron mucho más alentadores. Con la versión ligera, el tiempo de carga de la ontología y clasificación del razonador conjunto es de unos 4,5 segundos, con una respuesta a las consultas recibidas de pocas décimas de segundo. De esta manera, viendo los buenos resultados obtenidos con la versión ligera, probé con la versión completa para comprobar que tardaba en cargar y clasificar la ontología alrededor de 40 segundos.

Viendo estos resultados, y dado que la app tenía que girar en torno al razonador, decidí que lo mejor era una estructura cliente-servidor en la que el servidor era lanzado una única vez en el ordenador, ahorrando así el tiempo de las sucesivas cargas y clasificaciones que supondría tener que hacer si se lanzara directamente desde el móvil. Así pues el móvil recogería sencillamente las preferencias del usuario y mandaría la petición al servidor para recibir y mostrar los resultados de nuevo al usuario.

	Ontología ligera			Ontología completa		
Razonador	Carga	Clasif	Query	Carga	Clasif	Query
Razonador en móvil físico	7s	40-42s	8s	1.2m	4.5m	9s
Razonador en móvil simulado	7s	42-45s	8s	1.3m	5m	9s
Razonador en PC	0.1s	4.5s	0.2-1s	0.1s	40s	0.2-1.8s

Cuadro 1: Tabla comparativa de tiempos según las distintas arquitecturas.

Los tiempos a los que se refiere la tabla de carga y clasificación, es lo que le cuesta cargar primero el fichero que contiene la ontología y, clasificar el razonador después para dejarlo listo para resolver consultas. Por lo tanto, ambos tiempos habría que sumarlos para obtener el tiempo completo que pasa desde que se inicia el proceso hasta que el razonador está listo para resolver la primera consulta.

5. App

Con una primera versión de la ontología ya cerrada, y definida la estructura inicial de la aplicación, dictada por el rendimiento de los razonadores en la plataforma móvil, quedaba comenzar a implementar la propia app. Durante su desarrollo, y siguiendo una estrategia centrífuga, ha sido necesaria la adaptación de algunos aspectos tanto del diseño de la app como de la propia ontología. Esta fue una de las fases más costosas en cuanto a tiempo se refiere, debido a mi falta de experiencia con el entorno de trabajo y a los problemas derivados de los razonadores comentados en el apartado anterior. Todo el desarrollo del código de la app ha sido realizado con el entorno de trabajo Android Studio [23].

5.1. Arquitectura

La arquitectura de la aplicación, durante la fase de diseño, estaba prevista para que fuera de una sola capa. Es decir, que todo el procesado iba a estar concentrado en el propio dispositivo móvil, pero como ha quedado reflejado en el apartado de los razonadores, esto suponía un coste de tiempo completamente inasumible para una aplicación móvil.

De este modo, reciclando y modificando parte del código desarrollado en Java durante las prácticas de la asignatura de Programación de Sistemas Concurrentes y Distribuidos de la Universidad de Zaragoza y añadiéndole toda la parte del razonador, implementé un servidor que, mediante el uso de *threads*, es capaz de atender varias consultas de forma simultánea. La ontología es cargada y clasificada una vez se inicia el servidor, y cada *thread* recibe una consulta y una instancia del razonador ya clasificado, por lo que los resultados son procesados al momento, sin necesidad de ningún tipo de espera adicional. Un esquema de la arquitectura del sistema se encuentra en el apartado A.2 de los apéndices, en la Figura 9.

La comunicación entre el cliente y el servidor se realiza mediante sockets TCP. El cliente le manda un mensaje con los parámetros de la petición. El servidor resuelve esa petición y devuelve la respuesta por el mismo canal de comunicación en forma de una simple cadena de caracteres. El cliente toma esa cadena y la procesa para formatearla adecuadamente y mostrarla al usuario en la interfaz. Debido a que la comunicación es realizada a través de Internet, es necesario que la app tenga los permisos necesarios para ello. Cada consulta equivale a unos 30 Bytes enviados por parte del cliente al servidor y a unos 52 KB enviados desde el servidor al cliente como máximo.

5.2. Cliente

El cliente, o la app propiamente dicha (llamada *GimmeHop*, como juego de palabras entre *hop*, lúpulo, y *hope*, esperanza), permite distinguir entre tres tipos de búsquedas sobre la colección de cervezas. Las tres opciones te devuelven, en el caso de que haya individuos que satisfagan los términos requeridos, una lista con las cervezas encontradas ordenadas según cuáles se ajustan mejor a los criterios de la búsqueda. Desde estas listas, se puede acceder a las cervezas que aparezcan en ellas y consultar información detallada sobre cada una de ellas. A la primera opción la he llamado "búsqueda simple" y permite buscar por el nombre de las cervezas o por el nombre de las cervecerías que las hayan elaborado. La segunda opción, "búsqueda por similitud", devuelve cervezas de características lo más similares posibles a la cerveza seleccionada. Para estos dos primeros tipos de búsquedas, el razonador recupera todos los individuos de la ontología para su posterior filtrado en función de la búsqueda y los parámetros.

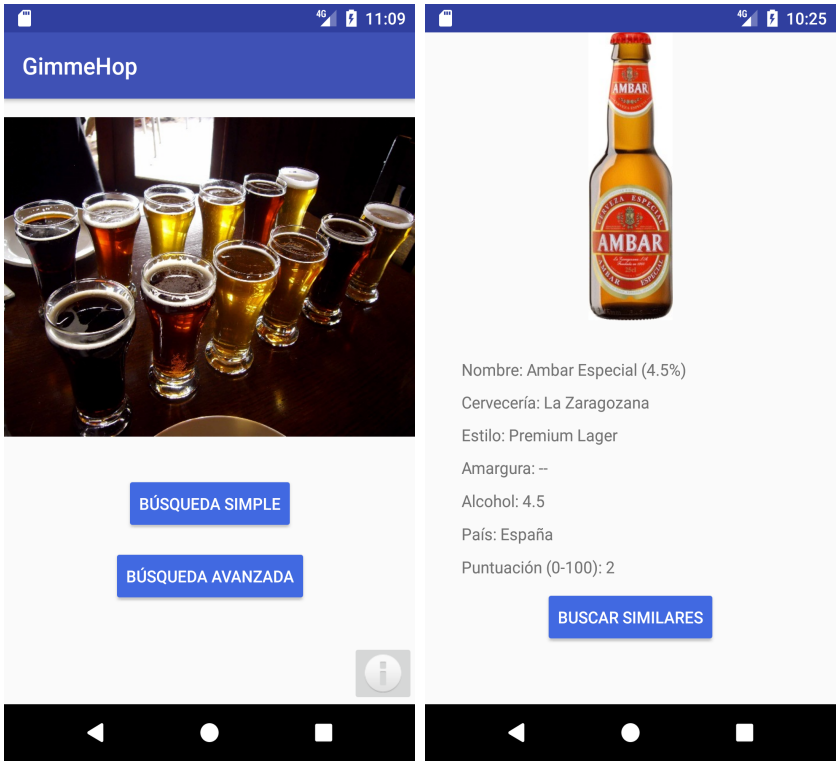


Figura 7: Pantalla principal e información de una cerveza

La tercera opción es la "búsqueda avanzada". Esta opción permite elegir entre las etiquetas lingüísticas definidas para los valores de alcohol y amargura, pudiendo elegir entre una de las cinco o la opción de "indiferente" para las características que no sean relevantes para la búsqueda, el estilo de cerveza y qué característica queremos que tenga más peso a la hora de buscar si se desea. En este tipo de búsqueda, el razonador recupera únicamente los individuos pertenecientes al estilo seleccionado. En el caso

de que se trate de una familia de estilos, recupera las cervezas pertenecientes a las subclases.

En los tres tipos de búsqueda se tiene en cuenta el contexto del usuario, dándole más importancia a las cervezas de producción nacional. Para conseguir esto, el cliente envía, además de los parámetros de búsqueda necesarios, el país con el que el dispositivo ha sido configurado. Se ha elegido esta manera por ser la más rápida y que menos afecta al rendimiento de la app, pues no hace uso del gps ni de ningún otro servicio externo al ser una mera comprobación.

Para ello, se utilizan tanto actividades como tareas asíncronas. Actividades para representar las diferentes funcionalidades que ofrece la app, permitiendo así la navegación adelante y atrás entre la pila de actividades de Android durante la ejecución de la aplicación. Las tareas asíncronas, que es un elemento de Android que permite trabajar fácilmente con diferentes hilos de ejecución y ejecuciones en segundo plano, son utilizadas para elementos secundarios de la ejecución, como son las comunicaciones con el servidor, la gestión y la preparación de la respuesta y la descarga de las imágenes de las cervezas de Internet.

5.3. Servidor

En la parte del servidor es donde se realiza todo el trabajo de razonamiento, clasificación y cálculo, liberando así al cliente de estas tareas. Distingue entre los tres tipos de consultas aunque para todas utiliza el razonador. En la "búsqueda simple" lo utiliza para recuperar todas las cervezas e ir comprobando una por una si contiene todas las palabras indicadas por el usuario, ya sea en el nombre de la cerveza o en el nombre de la cervecería que la haya fabricado (este diagrama de secuencias aparece en el apartado A.4 de los apéndices). Exceptuando la "búsqueda simple", en la que por su naturaleza los resultados no pueden ser ordenados (o coinciden los términos o no), tanto en la "búsqueda por similitud" como en la "búsqueda avanzada", los resultados están ordenados en función de cuánto se asemejan a los parámetros de búsqueda tomando como factor adicional el contexto del usuario. De esta manera, si un usuario situado en España manda una petición, las cervezas españolas tendrán una mayor prioridad a la hora de ser seleccionadas como relevantes para su búsqueda. Además, para los tres tipos de búsqueda, los resultados devueltos al cliente están limitados a 200 individuos, ya que con grandes cantidades de datos la app deja de funcionar correctamente.

No es hasta que no le llega una petición de que realice una "búsqueda avanzada" o una "búsqueda por similitud" cuando utiliza lógica difusa para calcular su respuesta. Para la búsqueda de cervezas por similitud a una dada, se recuperan todas las cervezas que pueblan la ontología y se calcula mediante lógica difusa cuán parecidas son a

la cerveza seleccionada. La función utilizada, detallada en el último apartado de los apéndices, tiene en cuenta cuánto se acercan los valores de alcohol y amargura y qué clase de cerveza es. De esta manera, las cervezas de la misma clase tendrán el valor máximo de similitud en este apartado, las que comparten superclase o uno de los estilos es superclase del otro tienen la mitad y valor nulo en cualquier otro caso. Esta manera de buscar cervezas similares a una dada sustituye al algoritmo original que estaba planteado, en el que simplemente se extraían las etiquetas lingüísticas de la cerveza seleccionada y se trataba como una "búsqueda avanzada", en la que se pedían al servidor cervezas de su mismo estilo y etiquetas lingüísticas. De esta manera, la "búsqueda por similitud" se acerca más a la realidad de lo que sugiere su nombre obteniendo resultados mucho más exactos y parecidos a la cerveza seleccionada.

Para las búsquedas avanzadas, el servidor recupera las cervezas del estilo que haya sido seleccionado por el usuario y calcula la función de pertenencia a las etiquetas lingüísticas indicadas tanto del alcohol como de la amargura (en el caso de que estas no hayan sido marcadas como indiferentes para la búsqueda). Una vez hecho esto, coge también la puntuación de las cervezas dentro de su propio estilo, normalizada entre los valores [0, 1] y junto con las otras dos funciones de pertenencia obtenidas anteriormente calcula el interés de cada cerveza para la consulta dada. El cálculo de esta medida se realiza de dos maneras diferentes. La opción por defecto (cuando el usuario indica que le es indiferente qué característica debe tener más peso), es mediante un OWA que otorga un peso del 40 % al valor más bajo de pertenencia a los subconjuntos difusos y 30 % a los otros 2. Es una visión pesimista, pues trata de minimizar que el individuo tenga un valor muy bajo en alguno de sus atributos, favoreciendo a aquellas cervezas que cumplen los requisitos en todos sus atributos por encima de aquellas que solo se acercan mucho en alguno de ellos. La segunda manera de realizar este cálculo se utiliza cuando el usuario elige alguna característica por encima de las otras dos. En este caso el cálculo se realiza mediante una media ponderada, en la que la característica elegida tiene el 50 % del peso y las otras 2 el 25 %.

Otra posibilidad para realizar el cálculo que subyace en las búsquedas avanzadas, sería delegar este cálculo al cliente. De esta forma, el servidor solo se encargaría de recuperar todas las cervezas del estilo correspondiente y enviarlas al dispositivo móvil para que este calculara cuáles son relevantes. Haciendo esto, existiría la posibilidad de que cada usuario marcara los límites de sus propias etiquetas lingüísticas, y el cálculo fuera realizado respecto a sus propios valores. El mayor problema que esto presenta es que aumentaría en gran medida el tráfico de datos entre el servidor y el cliente, haciendo el proceso mucho más ineficiente.

Como medida adicional, hay dos máquinas gemelas en Amazon Web Services

(AWS), con una copia de la ontología en local configuradas para poder ejecutar el servidor. Ambas máquinas están localizadas en Europa occidental, una en Londres y la otra en Frankfurt. La segunda máquina es una réplica por si la primera fallara, pero éstas no trabajan en paralelo.

5.4. Lógica difusa

Para ilustrar un poco cuál sería el papel que juega la lógica difusa en contraposición a la lógica binaria en el funcionamiento de la aplicación usaré un ejemplo. Cojamos dos cervezas muy conocidas en España, como son la *Ambar Especial*, la cerveza más popular de *La Zaragozana* y la *Mahou 5 Estrellas*, su contrapartida del *Grupo Mahou-San Miguel*. La *Ambar Especial* cuenta con un porcentaje de alcohol por volumen de 4.5, mientras que la *Mahou 5 Estrellas* tiene 5.5. Recordando los valores de las etiquetas lingüísticas establecidas vistos en la sección 3.4, más concretamente en la Figura 5, ambas pertenecerían probablemente a la de "Poco" alcohol aunque no tengan el mismo ABV.

Sin embargo, aplicando la función de pertenencia correspondiente propia de la lógica difusa, los resultados son algo más complejos.

- *Ambar Especial:* Obtiene un valor de pertenencia a la categoría de "Poco" alcohol de 0.64, 0.36 a la categoría de "Muy poco" y 0 para la categoría de "Moderado".
- *Mahou 5 Estrellas:* Su valor de pertenencia a la categoría "Poco" alcohol es 0.68, para la categoría "Muy poco" es 0 y para la categoría "Moderado" es 0.32.

Como se ve, la *Mahou 5 Estrellas* sería ligeramente más relevante para la búsqueda de cervezas con "Poco" alcohol por pertenecer a dicha categoría en un mayor grado, además de ser también tomada en cuenta para las búsquedas sobre cervezas con un nivel de alcohol "Moderado". Por el otro lado, aunque la *Ambar Especial* sea menos relevante en búsquedas sobre la categoría "Poco", será considerada también como parte de la categoría de cervezas con "Muy poco" alcohol.

5.5. Requisitos

La app está construida pensando en el nivel 25 del API de Android, que se corresponde con la versión 7.1.2 *Nougat* lanzada en junio de 2016, aunque el nivel mínimo requerido para funcionar es el 18, que es el existente en la versión 4.3.1 *Jelly Bean* lanzada en junio del 2012. El único requisito adicional para que la aplicación pueda funcionar correctamente es el acceso a Internet y el permiso correspondiente por parte del usuario para que pueda hacer uso de dicha conexión.

6. Trabajo relacionado

Una de las mayores ventajas que ofrece esta aplicación es la novedad que introduce en un mercado ya muy trillado como es el de la cerveza. Si bien es cierto que hay multitud de aplicaciones que te permiten conocer los detalles de una cerveza ya sea buscándola por su nombre o haciéndole una foto a la etiqueta, creo que no hay ninguna otra que utilice un sistema de recomendación basado en ontologías, con lógica difusa y un razonador semántico por detrás. Las apps más importantes de cerveza que hay actualmente son: RateBeer [14], Untappd [28] y Beer Citizen [29]. El funcionamiento de todas ellas es más o menos similar, centrándose más en el concepto de red social en el que poder compartir y ver las valoraciones de otros usuarios del sistema y ofreciendo información más o menos detallada de las cervezas.

En cuanto a las ontologías sobre cervezas disponibles en Internet, hay realmente pocas disponibles o al menos fácilmente accesibles. Entre ellas se encuentran la mencionada anteriormente *Beer Ontology Draft* [13] y la *beer-ontology* [30]. La primera mucho más parca, pues solo ofrece un boceto simple de una jerarquía de clases entre estilos de cerveza y, la segunda, mucho más detallada, ofreciendo una jerarquía mucho más elaborada y tratando de definir características propias de cada estilo, como pueden ser la apariencia, el sabor o el olor, además de definir algunos rangos de alcohol y amargura para los subtipos. Sin embargo, ambas ontologías están lejos de estar completas, ya que no solo les faltan estilos, si no que también (y más en el caso de la primera, pues la segunda al menos cita algunos ejemplos) les faltan multitud de individuos con sus propios detalles.

Lo más parecido a esta aplicación que hay en el mercado podría ser en el apartado de los vinos. Para empezar, en este caso sí que existe constancia de la existencia de ontologías, ya que incluso el W3C ha publicado la suya propia [31]. Partiendo de esta base, es fácil encontrar aplicaciones que recomiendan vinos en base a preferencias del usuario, como pueden ser el precio, el tipo de vino, el tipo de uva o el maridaje, como es el caso del recomendador *Consum* [32] o la app de origen estadounidense *Hello Vino* [33]. Sin embargo, y aunque no se aportan detalles de implementación sobre estas herramientas, no consta que utilicen ni ontologías, ni razonadores semánticos ni lógica difusa.

Como contrapunto, cabe destacar la existencia de una app para el Nokia N900 que sí que utiliza ontologías y lógica difusa pero no un razonador semántico, ya que durante su desarrollo estos todavía no existían. [34]

7. Conclusiones

Como resultado de este proyecto, caben destacar distintas contribuciones: Primero y más obvio, una app que incluye elementos innovadores sobre un tema relevante que ha despertado el interés de alguna empresa privada y de alguna institución pública. Una ontología bastante completa y trabajada, que mejora en algunos aspectos a otras ontologías en Internet, como es el hecho de una buena cantidad de individuos con información relevante sobre cada uno de ellos o una jerarquía de clases contrastada. Además, esta ontología contiene una gran cantidad de datos en formato semántico que podrían ser reutilizados por otras aplicaciones inteligentes y ser integrados en sistemas más fácilmente. Es una aplicación capaz de adaptarse a las preferencias del usuario expresadas de una manera cómoda para este y a su contexto mediante su localización geográfica. Y, por último, constituye una prueba de concepto a temas de investigación que no habían pasado del nivel teórico hasta ahora.

Por otro lado, ha quedado reflejado que aún falta mucho trabajo por delante para conseguir que las ontologías sean una realidad en el ámbito de la computación móvil. Hacen falta razonadores portados y optimizados para poder trabajar en los pequeños dispositivos y, probablemente, algo de potencia de computación para que estos puedan realizar los cálculos necesarios en tiempos razonables. Una segunda conclusión es la potencia expresiva y el abanico de posibilidades que ofrece la lógica difusa junto con la Inteligencia Artificial en este tipo de aplicaciones, donde se le pide al usuario que introduzca sus preferencias y gustos, ya que, mientras que es un concepto algo extraño cuando hablamos de informática, el tratamiento de la incertidumbre es algo a lo que el ser humano está más que acostumbrado.

Personalmente, la elaboración de este trabajo de fin de grado me ha ayudado a comprender las implicaciones que tiene abordar un "gran" proyecto más o menos en solitario, teniendo que asumir todas las responsabilidades y la toma de decisiones. Además, me ha ayudado a darme cuenta del valor que tiene mi trabajo. Sobre todo al realizar la memoria, ya que había muchas cosas a las que había tiempo y que no valoraba o consideraba obviedades y no lo son. En el ámbito técnico, me ha permitido profundizar en distintos aspectos con los que he trabajado a lo largo de la carrera, como son las ontologías, la lógica difusa, la programación en Android o el lenguaje Python, el cual tuve que aprender específicamente para este trabajo ya que no lo había utilizado nunca.

Como aspectos que me habría gustado mejorar o cambiar, está el hecho de necesitar un ordenador que actúe de servidor para poder responder a las búsquedas lanzadas por el usuario. Esto limita el funcionamiento de la aplicación y le resta la independencia

con la que estaba pensada desde un principio, ya que requiere de consumo de datos y de conectividad a Internet para funcionar. Aunque, por otro lado, también la hace más rápida y ligera, reduciendo a menos de la mitad el tamaño final de la app, de unos 11 MB a no llegar a 5 MB. Por último, pienso que sería interesante completar los datos sobre las cervezas de la ontología, ya que la amargura o el país de procedencia no está presente en todos los individuos. Además, el añadir información sobre las cervecerías ampliaría bastante el abanico de posibilidades y la utilidad de la app.

8. Referencias

[1] Definicion de ontologia - ¿qué es una ontología?, 2004. https://sites.google.com/site/jojooa/inteligencia-artificial/definicion-de-ontologia-que-es-una-ontologia , accedido noviembre 2017.

[2] Wikipedia. Interfaz de programación de aplicaciones, 2017. https://es.wikipedia.org/wiki/Interfaz_de_programacion_de_aplicaciones, accedido noviembre 2017.

[3] Wikipedia. Ale, 2017. https://es.wikipedia.org/wiki/Ale, accedido noviembre 2017.

[4] Wikipedia. Lager, 2017. https://es.wikipedia.org/wiki/Lager, accedido noviembre 2017.

[5] Rául Sánchez Gómez, Universidad Carlos III. Lógica difusa, 2009. https://es.slideshare.net/mentelibre/logica-difusa-introduccion, accedido noviembre 2017.

[6] Wikipedia. Algoritmo de agrupamiento, 2017. https://es.wikipedia.org/wiki/Algoritmo_de_agrupamiento, accedido noviembre 2017.

[7] Wikipedia. K-means, 2017. https://es.wikipedia.org/wiki/K-means, accedido noviembre 2017.

[8] W3C OWL Working Group. Owl 2 web ontology language document overview (second edition), 2012. https://www.w3.org/TR/owl2-overview/, accedido noviembre 2017.

[9] Wikipedia. Android runtime, 2017. https://es.wikipedia.org/wiki/Android_Runtime, accedido noviembre 2017.

[10] Wikipedia. Socket de internet, 2017. https://es.wikipedia.org/wiki/Socket_de_Internet, accedido noviembre 2017.

[11] Wikipedia. Protocolo de control de transmisión, 2017. https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisión, accedido noviembre 2017.

[12] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decision making. *EEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190, 1988.

[13] Jeff Heflin. Beer Ontology draft, 2000. https://www.cs.umd.edu/projects/plus/SHOE/onts/beer1.0.html, accedido noviembre 2017.

[14] Joseph Tucker. Ratebeer, 2000-2017. https://www.ratebeer.com, accedido noviembre 2017.

[15] M.A Musen. The Protégé project: A look back and a look forward *Association of Computing Machinery Specific Interest Group in Artificial Intelligence*, 2015. https://protege.stanford.edu/, accedido noviembre 2017.

[16] Andrew Lilja. *Get information out of RateBeer.com without losing your mind*, 2017. https://github.com/OrganicIrradiation/ratebeer, accedido noviembre 2017.

[17] Matthew Horridge and Peter F. Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition), 2012. https://www.w3.org/TR/owl2-manchester-syntax/, accedido noviembre 2017.

[18] RateBeer. Ratebeer api documentation, 2017. https://www.ratebeer.com/api-documentation.asp, accedido noviembre 2017.

[19] Graphql, 2017. http://graphql.org/, accedido noviembre 2017.

[20] F. Bobillo and U. Straccia. Fuzzy ontology representation using owl 2. *International Journal of Approximate Reasoning*, 52(7):1073–1094, 2011.

[21] Grupo de sistemas de información distribuidos de la Universidad de Zaragoza. Android goes semantic!, 2016. http://sid.cps.unizar.es/AndroidSemantic/index.html, accedido noviembre 2017.

[22] C. Bobed, R. Yus, F. Bobillo, and E. Mena. Semantic reasoning on mobile devices: Do androids dream of efficient reasoners? *Journal of Web Semantics*, 35(4):167–183, 2015.

[23] IntelliJ. Android Studio *IDE oficial para Android*, 2017. https://developer.android.com/studio/index.html, accedido noviembre 2017.

[24] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. Owl 2 web ontology language profiles (second edition), 2012. https://www.w3.org/TR/owl2-profiles/, accedido noviembre 2017.

[25] University of Oxford Department of Computer Science. A java-based owl 2 el reasoner, 2011-2012. https://github.com/liveontologies/elk-reasoner, accedido noviembre 2017.

[26] Jeff Z. Pan. TrOWL just another configurable, transformation based, tractable approximate reasoner for owl2, 2015. http://trowl.org/, accedido noviembre 2017.

[27] Julian Mendez. jcel reasoner for the description logic el+, 2015-2017. http://julianmendez.github.io/jcel/, accedido noviembre 2017.

[28] Tim Mather and Greg Avola. Untappd drink socially, 2017. https://untappd.com/, accedido noviembre 2017.

[29] Beer citizen, 2012-2017. https://www.beercitizen.com/, accedido noviembre 2017.

[30] Wesley Davison. beer-ontology, 2014. https://github.com/wesleydavison/beer-ontology/blob/master/ontology.xml, accedido noviembre 2017.

[31] W3C. Wine ontology, 2009. https://www.w3.org/TR/owl-guide/wine.rdf, accedido noviembre 2017.

[32] Consum. El recomendador, 2017. https://labodega.consum.es/recomendador-vinos, accedido noviembre 2017.

[33] Consum. Hello Vino your virtual wine assistant, 2017. https://labodega.consum.es/recomendador-vinos, accedido noviembre 2017.

[34] József Mezei Christer Carlsson, Matteo Brunelli. Decision making with a fuzzy ontology. *Soft Computing - A Fusion of Foundations, Methodologies and Applications - Special Issue on Fuzzy Ontologies and Fuzzy Markup Language Applications*, 16(7):1143–1152, 2012.

A. Ingeniería del software

A.1. Diagrama de casos de uso

El primer diagrama presentado en la Figura 8 es el diagrama de casos de uso de la aplicación. En él se ve que el principal cometido de la app es la búsqueda de cervezas, permitiendo tres maneras diferentes de hacerlo. Ya sea mediante la búsqueda simple, por coincidencia de nombre, la búsqueda similitud entre cervezas o la búsqueda avanzada, que permite elegir más parámetros especializados. Además, los tres tipos de búsqueda permiten después obtener información detallada sobre cualquiera de los resultados obtenidos.

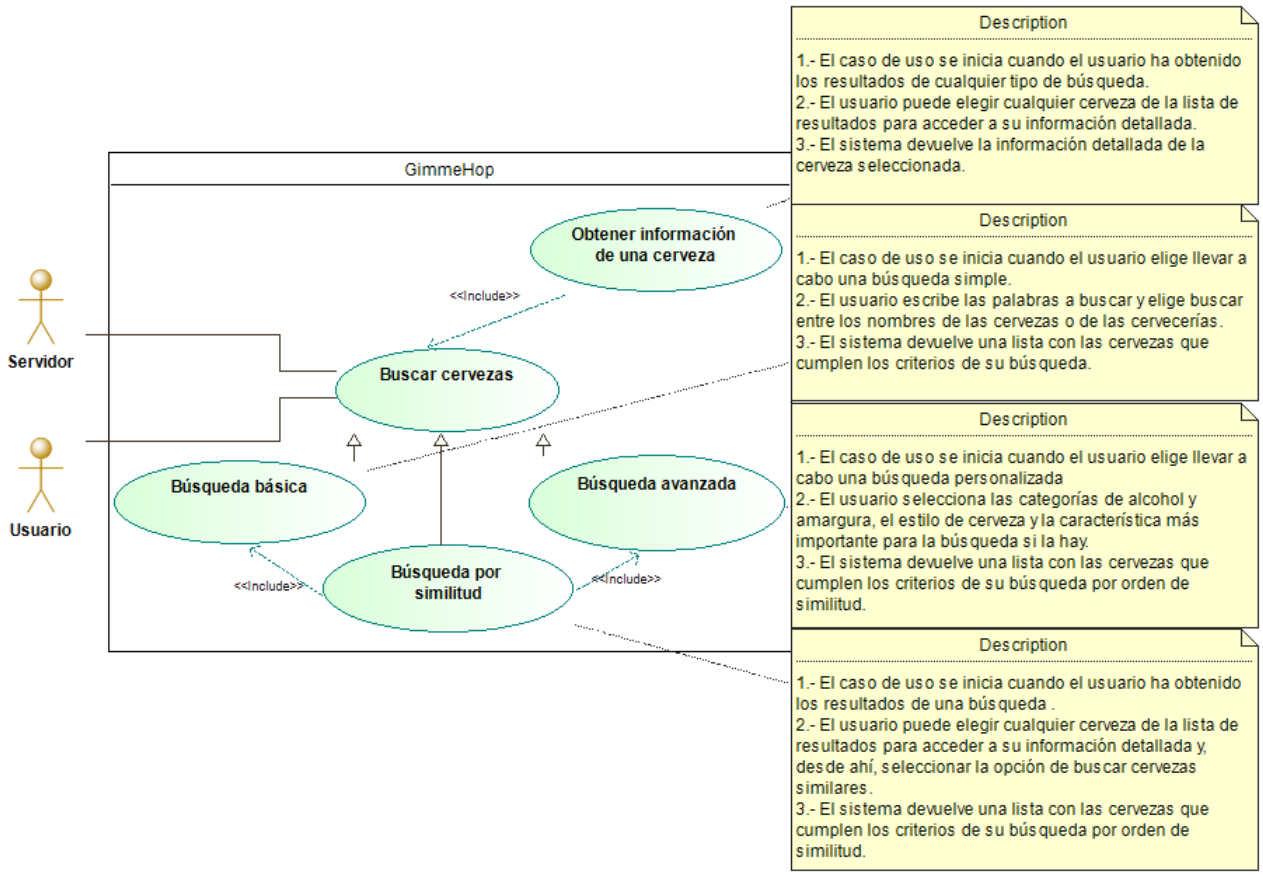


Figura 8: Diagrama de casos de uso de la app

A.2. Diagrama de despliegue

En la Figura 9 se muestra el diagrama correspondiente al despliegue del sistema. El dispositivo Android sería cualquier móvil en el que estuviera instalada la app mientras que la máquina servidor está alojada en AWS. En está máquina están tanto el servidor con el razonador como la propia ontología.

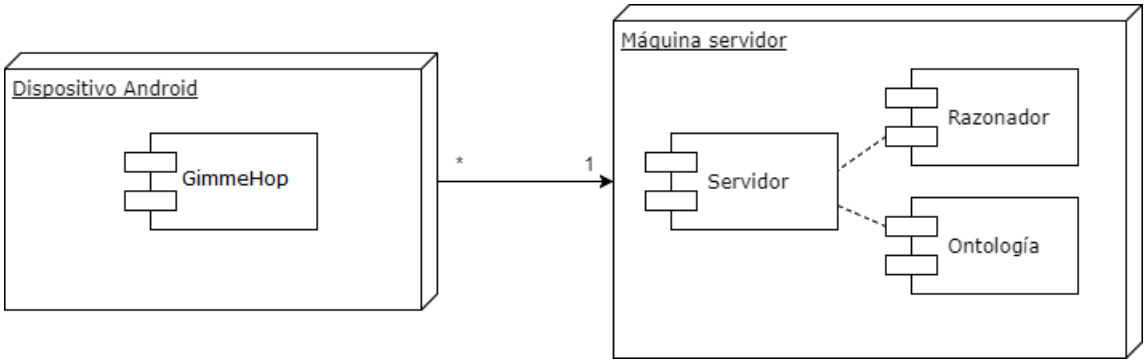


Figura 9: Diagrama de despliegue del sistema

A.3. Diagrama de clases

Los diagramas de clases, por su complejidad y tamaño están divididos en dos: el diagrama de clases de la propia aplicación y el del servidor. En ambos casos, la clase *Beer* tiene un método *getter* y *setter* por cada atributo que, por motivos de espacio y al ser redundantes, no aparecen en los diagramas.

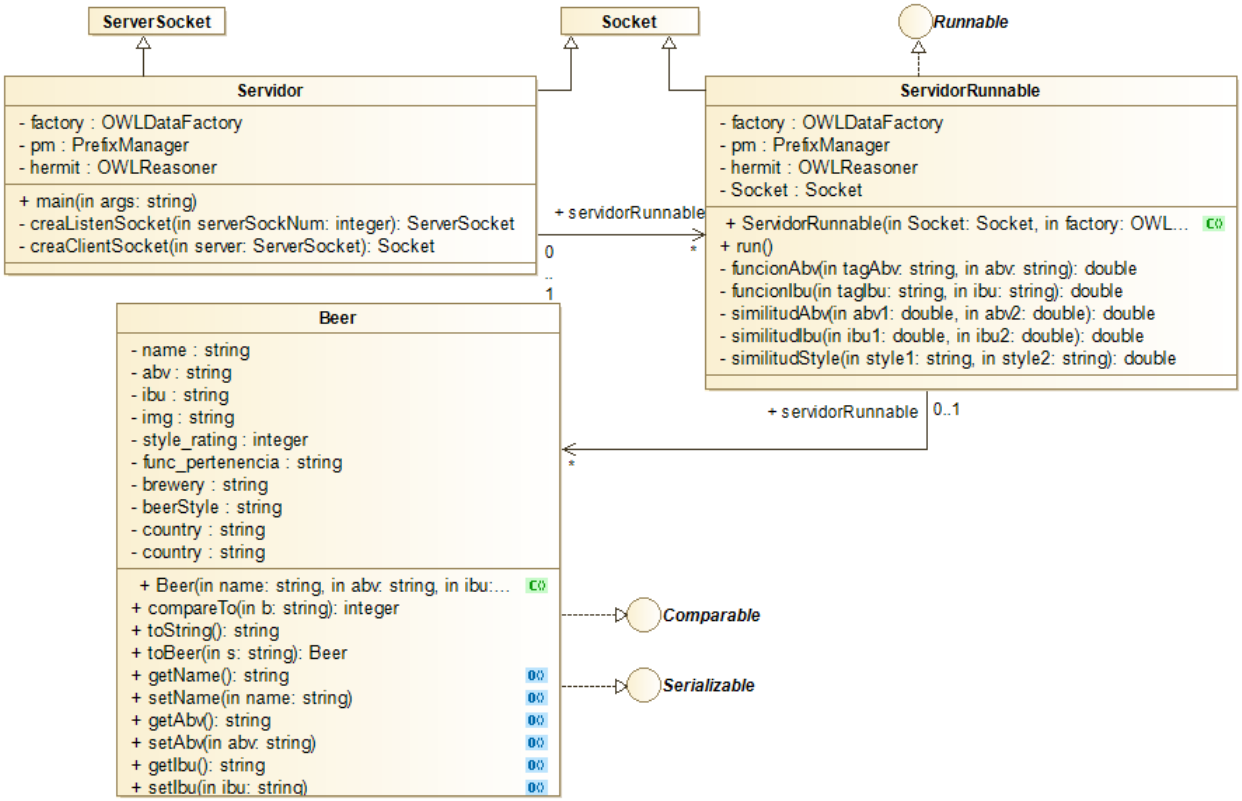


Figura 10: Diagrama de clases del servidor

En este diagrama queda representada la distribución del servidor. Está la clase principal llamada *Servidor*, que es la encargada de crear tanto el socket de escucha como el socket que posteriormente utilizará cada *thread* para comunicarse con la app, de clasificar la ontología y de crear una instancia del razonador. Una vez recibe una comunicación, crea un nuevo *thread* de la clase *ServidorRunnable* que será el encargado de responder a la app.

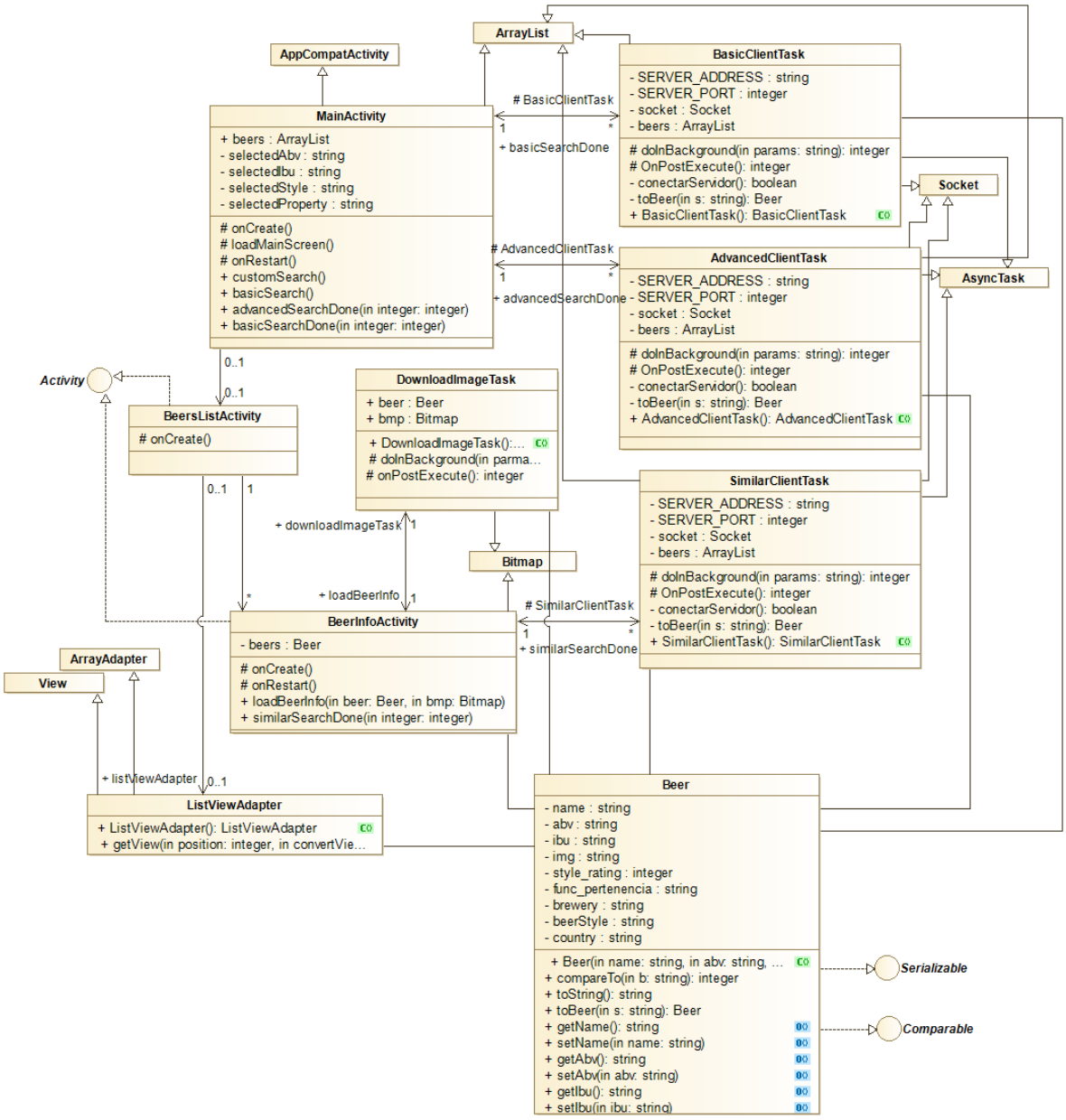


Figura 11: Diagrama de clases de la aplicación

El diagrama de clases de la app es bastante más complicado que el del servidor por la propia tecnología Android en la que está basada. La clase principal es la *MainActivity*, que es la que maneja el flujo principal de la app. Es ella quien lanza una búsqueda básica o avanzada en función de lo que elija el usuario y, una vez tiene los resultados devueltos por la tarea asíncrona correspondiente, cede el control de la app a la siguiente actividad, llamada *BeersListActivity*, que muestra una lista con las cervezas obtenidas. Desde esta nueva actividad se puede lanzar otra nueva, *BeerInfoActivity*, que será la que muestre al usuario la información detallada de la cerveza en cuestión, además de dar la posibilidad de buscar cervezas similares en el caso de que se haya tratado de una búsqueda básica.

A.4. Diagrama de secuencia

Por último, en la Figura 12, se muestra el diagrama de secuencia de la app para una búsqueda básica y la posterior consulta sobre la información detallada de la cerveza seleccionada. Este diagrama de secuencia es perfectamente análogo tanto para la búsqueda avanzada como para la búsqueda por similitud, salvo que para acceder a la búsqueda por similitud, hay que seleccionar primero una cerveza obtenida mediante una búsqueda normal o avanzada. Es decir, que el diagrama de secuencias para una búsqueda por similitud, sería un diagrama de secuencia de una búsqueda básica o avanzada más una nueva búsqueda al final.

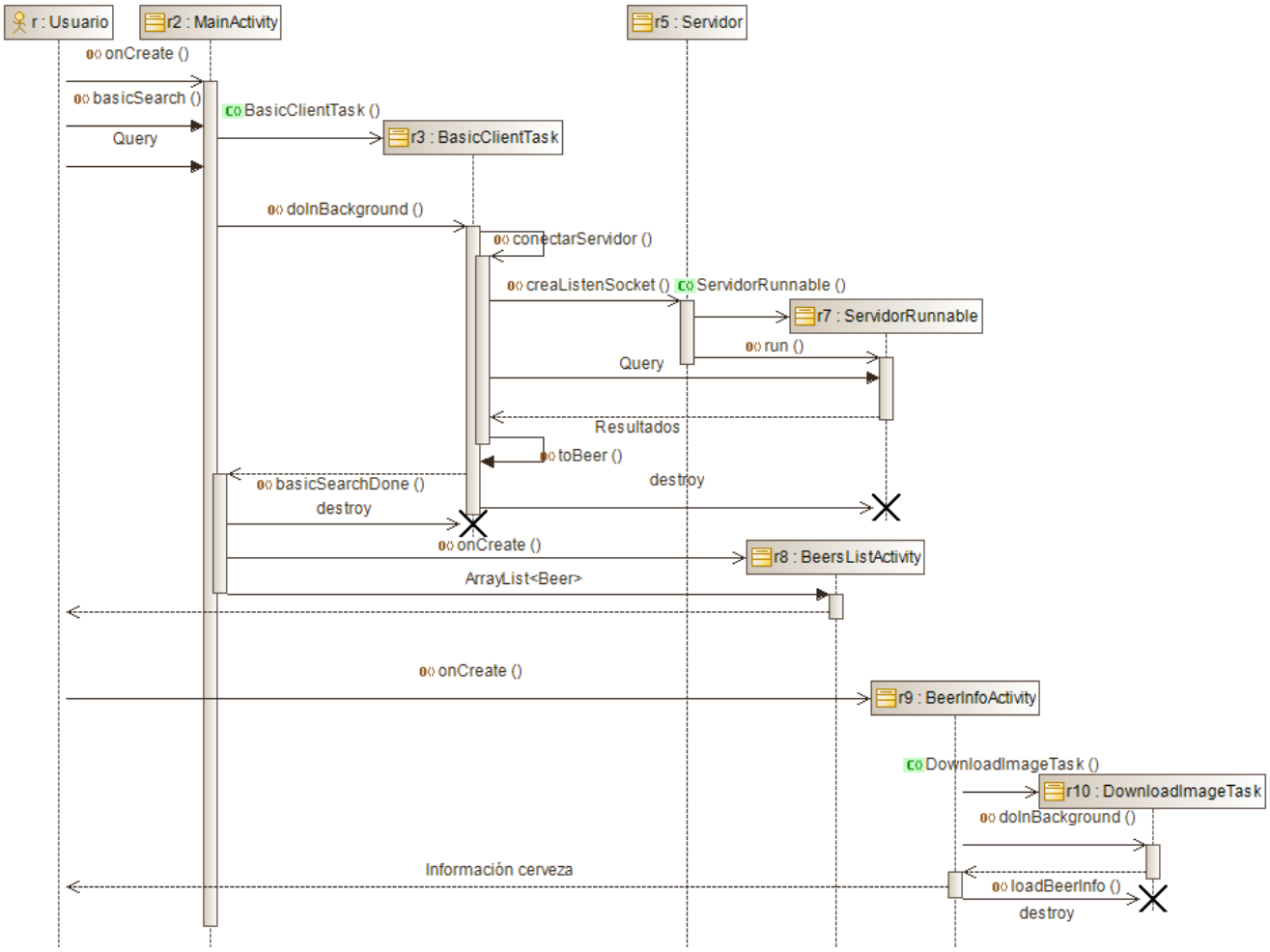


Figura 12: Diagrama de secuencia de una búsqueda básica

B. Manual de usuario

El uso de la aplicación es bastante sencillo. En la pantalla principal (Figura 13) aparecen dos opciones y un pequeño icono con el símbolo de información. Al pulsarlo aparece un mensaje con información acerca del creador de la app y del propósito con el que ésta ha sido creada. Al pulsar sobre cualquiera de las dos opciones de búsqueda, nos llevará a una nueva pantalla en la que elegir los parámetros de búsqueda.

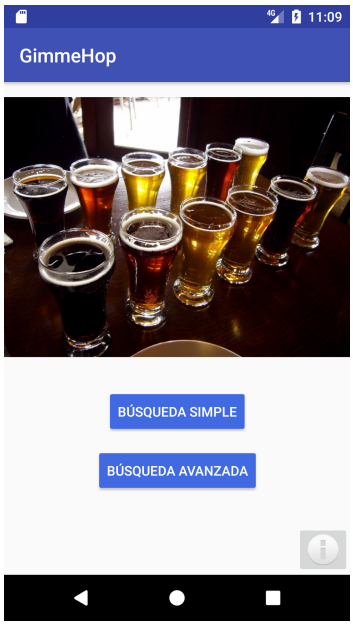
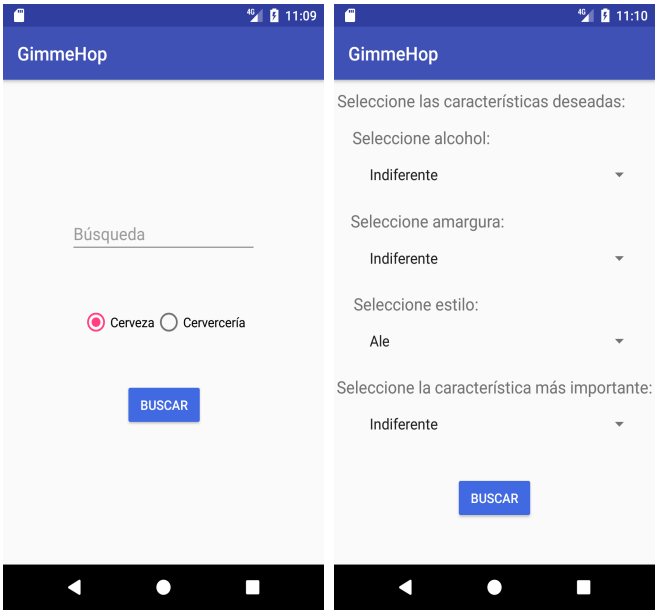


Figura 13: Diagrama de casos de uso de la app

En la Figura 14 se muestran las dos pantallas de búsqueda. La Figura 14.a, la correspondiente a la búsqueda básica, muestra un cuadro en el que escribir la búsqueda que se desee y sendos botones para indicar si buscar entre los nombres de las cervezas o entre los nombres de las cervecerías. Una vez rellenado esto (si no se escribe nada no permite la búsqueda) puede comenzar la búsqueda. El apartado b de la misma figura es la pantalla correspondiente a la búsqueda avanzada. En ella aparecen 4 campos desplegables que permiten elegir entre las diferentes etiquetas lingüísticas definidas para alcohol y amargura, el estilo de cerveza y cuál es la característica más importante, si es que la hay, entre los criterios seleccionados anteriormente. Aunque no aparezca como parámetro a elegir en ninguna de las dos pantallas de búsqueda, el servidor tiene en cuenta automáticamente el país del usuario para cualquier tipo de búsqueda, aumentando la relevancia de las cervezas nacionales y haciendo que éstas aparezcan con una mayor frecuencia que las extranjeras.



(a) Búsqueda básica (b) Búsqueda avanzada

Figura 14: Pantalla principal e información de una cerveza

A continuación, independientemente del tipo de búsqueda que se haya elegido, y siempre y cuando se hayan encontrado elementos, aparece una lista de cervezas como la que aparece en la Figura 15, siendo posible elegir cualquiera de ellas para obtener la información detallada de ese individuo en particular. Al acceder a la información detallada de una cerveza, aparece la opción de buscar cervezas de características similares, en cuyo caso volvería a aparecer una nueva lista de cervezas con las cervezas ordenadas en función de cuánto se parecen a la cerveza seleccionada en un principio.

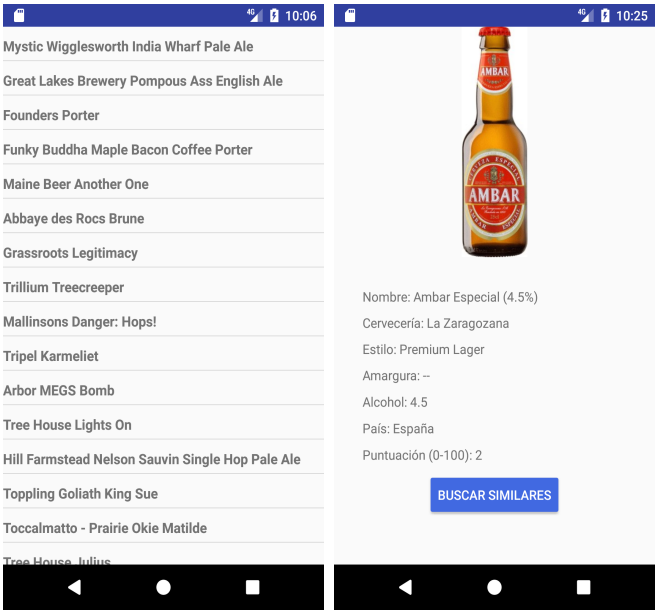


Figura 15: Lista de cervezas e información detallada de una de ellas

Por último, hay otras tres posibles pantallas, estas son, como se puede ver en la Figura 16, la pantalla que se muestra mientras la app se esta comunicando con el servidor y formateando la respuesta obtenida, la pantalla que muestra que no ha habido resultados en la búsqueda y la pantalla que aparece en el caso de que haya algún problema con el servidor o este no se encuentre accesible en ese momento.

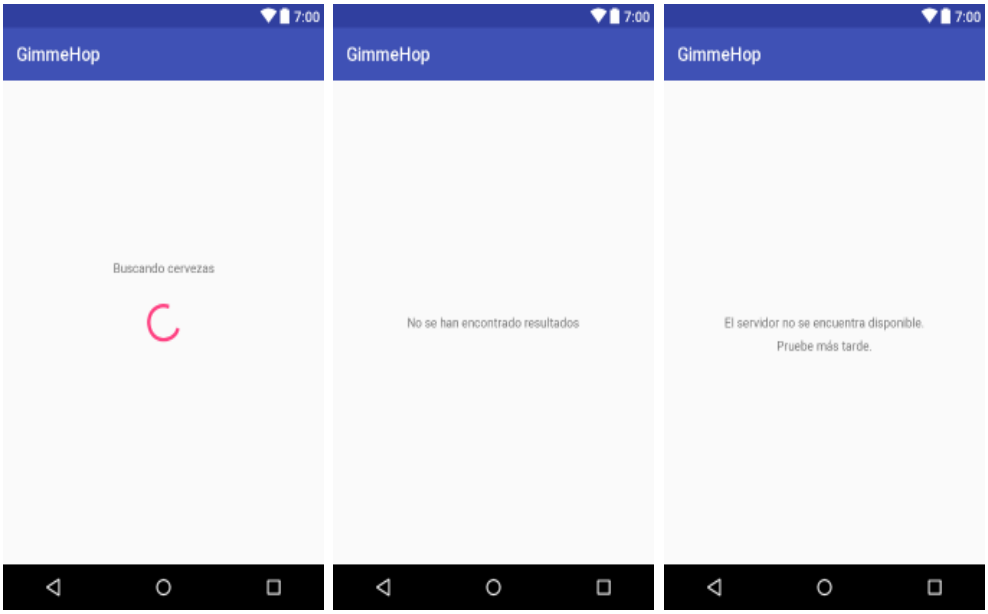


Figura 16: Pantallas de buscando cervezas, no se han encontrado resultados y error del servidor

C. Temporización y planificación

En la Figura 17 está el diagrama de Gantt de lo que ha sido el desarrollo de este proyecto. Cada recuadro de cada mes implica una semana de trabajo en la tarea especificada. Si bien, como se puede ver, no significa que se haya trabajado única y exclusivamente en esa tarea en concreto. La tarea más larga ha sido sin duda la creación de la app, en la que están incluidas todas las pruebas e intentos realizados con otros razonadores. Su investigación incluye tanto como aprendizaje de Android por mi parte como investigación acerca de los razonadores a utilizar.

En cuanto a la ontología, que ha sido la otra tarea que más tiempo se ha llevado del proyecto, requirió una gran cantidad de investigación y documentación, tanto técnica (ontologías, razonadores...) como del dominio de aplicación (cervezas) para poder clasificar las cervezas adecuadamente. Posteriormente, fue necesaria una considerable cantidad de tiempo también para obtener todos los individuos con los que poder poblarla y pulirla hasta llegar a su versión final.

Tanto julio como agosto y octubre no tienen todas las semanas que deberían en el gráfico. Esto es debido a que ha habido algunas semanas de inactividad (5) en el desarollo del proyecto.

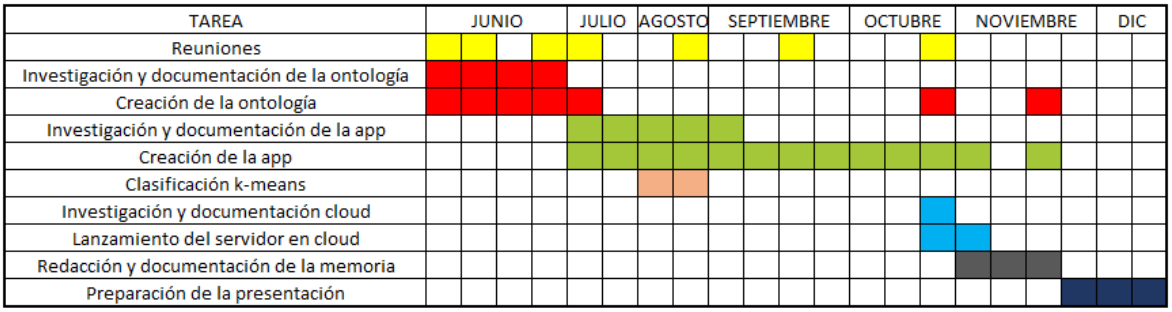


Figura 17: Diagrama de Gantt

D. Pseudocódigo

D.1. Funcionamiento general del servidor

```
Servidor{

    CrearServerSocket(puerto);
    file_ontología = CargarFicheroOntología(path);
    ontología = CargarOntología(file_ontología);
    hermit = CrearRazonador(ontología);
    hermit.clasificar();        //Esto fuerza la clasificación de la ontología
    while(true){
        clientSocket = creaClientSocket(serverSocket);        //Acepta peticiones
        sr = new ServidorRunnable(clientSocket, hermit);
        t = new Thread(sr);
        t.start();        //Lanza el nuevo thread para resolver la consulta
    }
}
```

D.2. Funcionamiento de cada *thread* del servidor

```
ServidorRunnable{

    if (búsqueda basica){
        beers = hermit.cogerTodasLasCervezas();
        for beer in beers{
            if (beer.name().contains(Query)){
                resultados.add(beer);
            }
        }
    } else if (búsqueda avanzada){
        beers = hermit.cogerCerverzasEstilo(Query.estilo);
        for beer in beers{
            func_abv = calcularFuncAbv(beers, Query.abv);
            func_ibu = calcularFuncIbu(beers, Query.ibu);
            score = beer.getScore();
            func_total = calcularFuncTotal(func_abv, func_ibu, score)
                    //OWA o media ponderada, según haya elegido el usuario
            beer.setFuncTotal(func_total);
            resultados.add(beer);
        }
        resultados.sort(func_total);
    } else{
        for beer in beers{
            similitud = búsqueda por similitud (beer_seleccionada, beer)
                        //Apartado D3 apéndices
            beer.setFuncTotal(similitud);
            restultados.add(beer);
            resultados.sort(similitud);
        }
    }
    return resultados;
}
```

D.3. Algoritmo de búsqueda de cervezas por similitud

Algoritmo búsqueda similar(cerveza1, cerveza2){

similitud_abv = 0.0, similitud_ibu = 0.0, similitud_style =0.0;
diferencia_abv = |cerveza1.abv - cerveza2.abv|;
if (diferencia_abv ≤ 0.5){
similitud_abv = 1.0;
} else if(diferencia_abv ≥ 1.5){
similitud_abv = 0.0;
} else{
similitud_abv = (1.5 - diferencia_abv) / 1;
}

diferencia_ibu = |cerveza1.ibu - cerveza2.ibu|;
if (diferencia_ibu ≤ 3.0){
similitud_ibu = 1.0;
} else if(diferencia_ibu ≥ 15.0){
similitud_ibu = 0.0;
} else{
similitud_ibu = (15.0 - diferencia_ibu) / 12.0;
}

if(cerveza1.clase = cerveza2.clase){
similitud_style = 1.0;
} else if(cerveza1.ancestroComunCon(cerveza2)){
similitud_style = 0.5;
} else{
similitud_style = 0.0;
}

similitud_total = (similitud_abv + similitud_ibu + similitud_style) / 3;
}