ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA
Mención en Sistemas de Información

MOTOR DE RECOMENDACIÓN CONSTRUIDO SOBRE IMPLICACIONES
USING IMPLICATIONS TO BUILT A RECOMENDER SYSTEM

Realizado por
DAVID BARRIENTOS BRENES
Tutorizado por
CARLOS ROSSI JIMÉNEZ
Co-tutorizado por
MANUEL ENCISO GARCÍA-OLIVEROS
Tutor coordinador
CARLOS ROSSI JIMÉNEZ
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre de 2014

Fecha defensa:
El Secretario del Tribunal

Resumen: Este Trabajo Fin de Grado (TFG) tiene como objetivo la creación de un framework para su uso en sistemas de recomendación. Se ha realizado por dos personas en la modalidad de trabajo en equipo. Las tareas de este TFG están divididas en dos partes, una realizada conjuntamente y la otra de manera individual. La parte conjunta se centra en construir un sistema que sea capaz de, a partir de comentarios y opiniones sobre *puntos de interés* (POIs) y haciendo uso de la herramienta de procesamiento de lenguaje natural AlchemyAPI, construir *contextos formales* y *contextos formales multivaluados*. Para crear este último es necesario hacer uso de ontologías. El *context formal multivaluado* es el punto de partida de la segunda parte (individual), que consistirá en, haciendo uso del *contexto multivaluado*, obtener un conjunto de *dependencias funcionales* mediante la implementación en Java del algoritmo FDMine. Estas *dependencias* podrán ser usados en un motor de recomendación. El sistema se ha implementado como una aplicación web Java EE versión 6 y una API para trabajar con *contextos formales multivaluados*. Para el desarrollo web se han empleado tecnologías actuales como Spring y jQuery. Este proyecto se presenta como un trabajo inicial en el que se expondrán, además del sistema construido, diversos problemas relacionados con la creacion de conjuntos de datos validos. Por último, también se propondrán líneas para futuros TFGs.

Palabras clave: Sistemas de recomendación, Descubrimiento de conocimiento, Procesamiento del lenguaje natural, Obtención de dependencias, Contexto formal multivaluado

Abstract: The aim of this Degree Thesis (TFG) is to create and provide a framework to be used in recommendation systems. This TFG has been carried out by two students in a co-operation teamwork model. Tasks in this TFG are divided into two main parts. The first part was made jointly in collaboration by both students and the second part was made individually by each student. The aim of the collaborative part is to build a system that should be able to build *formal contexts* and *multivalued formal contexts*, from comments and suggestions about *points of interest* (POIs) and using the natural language processing tool AlchemyAPI. To build such a system, it is necessary to use ontologies. The *many-valued context* is the starting point for the second part of the individual TFG. This part consists on using the *many-valued context* to obtain a set of *functional dependencies* through the Java implementation of the algorithm FDMine. These *dependencies* might be used in a recommendation engine. The system has been implemented as Java EE version 6 Web Application and an API that supports *many-valued contexts*. Current technologies such as Spring and jQuery have been used for the web development. This project is presented (submitted) as a preliminary job in which several problems related to the creation of a valid set of data will be described, as well as the main system topics. Finally, some ideas will be also proposed for future TFG.

Keywords: Recommender systems, Knowledge discovery, Natural language processing, Get dependencies, Many-Valued Context

Índice

1. Introducción

Este es un Trabajo Fin de Grado (TFG) desarrollado en la línea de un Social CRM. Esta línea busca la creación de un sistema de recomendación, desde la extracción de la información de las fuentes de datos: comentarios, opiniones, reseñas, etc. hasta dar una recomendación al usuario. Principalmente se centra en recomendaciones para el sector turístico, uno de los sectores más importantes de Málaga. Es tal la importancia que han adquirido estos sistemas que existen numerosos grupos de investigación trabajando en estos temas, como puede ser el grupo SICUMA (Sistemas de Información Cooperativos de la Universidad de Málaga)[20].

Al tratarse de un trabajo fin de grado de la modalidad de equipo es importante destacar el reparto de las tareas a realizar. Una primera tarea fue desarrollada conjuntamente, mientras que las otras dos fueron realizadas por cada uno de los miembros de manera individual. Estas dos tareas individuales están bien diferenciadas pero relacionadas entre sí.

En la parte común del TFG se estudió como, desde unas opiniones sobre *puntos de interés* (POIs), extraer información sobre atributos o características de los mismos. Esta información debía de ser tratada y recogida en una tabla binaria en la que se relacionasen a los autores de los comentarios (objetos) con las características más relevantes (atributos). A esta tabla se le denomina *contexto formal*. Posteriormente, fue necesario ampliar este concepto a *contexto formal multivaluado* o FCMV, donde la tabla pasa de ser binaria a tener múltiples valores. Para ello se empleó la teoría conocida como *analisis formal de conceptos multivaluado* o *formal concept analysis many-valued (FCAMV)*.

La parte específica del TFG presentada en esta memoria consistió en el descubrimiento de relaciones de dependencias entre las características de los POIs. Partiendo del *contexto formal multivaluado* se aplica un algoritmo denominado FDMine para obtener una serie de dependencias entre el conjunto de atributos. Estas dependencias serán interpretadas a posteriori para poder ser usadas en un sistema de recomendación. Ademas, dentro de este apartado, la fase de experimentos ha sido muy importante y es donde se interpreta los resultados y extraen las conclusiones sobre las dependencias obtenidas de un conjunto de POIs.

Para ilustrar el objetivo de este TFG se expone el siguiente ejemplo:

Supongamos que tenemos como objetos un conjunto de comentarios de distintos usuarios sobre un determinado hotel y como atributos una serie de características del mismo como: estación del año, medio de trasporte, servicios extra, con quien va acompañado, etc. Analizando estos comentarios, podríamos averiguar que por ejemplo, con conocer las caracterisiticas *la estación del año* y *con quien va acompañado*, podemos saber cuál será el valor del *servicio* extra usado. Así, podríamos decir que si, por ejemplo, fue en verano y con la familia, el *servicio extra* buscado fue playa. De esta forma podríamos

reducir el número de datos que necesite introducir un usuario para poder darle una buena recomendación con sólo conocer algunas preferencias del usuario.

Para la construcción de este sistema se ha realizado un desarrollo web aplicando la tecnología Java EE versión 6 haciendo uso de frameworks y librerías como Spring, jQuery, Bootstrap, etc. Además para la parte de minería de datos y obtención de atributos a partir de los textos se empleó la API de AlchemyAPI que funciona mediante un servicio web.

En las posteriores secciones se desarrollarán todos los aspectos relativos al TFG. En la sección 2 se explica la metodología seguida durante todo el desarrollo del proyecto. A continuación, en la sección 3 se nombran los requisitos que debe de cumplir dicho sistema. Seguidamente, en la sección 4, se expondrá el estado del arte y los fundamentos teóricos necesarios para la construcción del sistema. En la sección 5 se analizarán uno a uno los requisitos enunciados en la sección 3. La sección 6 describe el entorno tecnológico del sistema y la sección 7 los aspectos relativos al desarrollo de la aplicación. A continuación se exponen y analizan una serie de experimentos desarrollados a partir del sistema construido (Sección 8). También se presentan unas conclusiones generales de todo el TFG (Sección 9). Además de estas secciones, en la memoria se incluye la bibliografía y una serie de anexos que se componen de: Un desglose a alto nivel del tiempo empleado en cada tarea del TFG (Sección 10.1), los *contextos formales multivaluados* (Sección 10.3) con sus ontologías (Sección 10.2 y un pequeño manual de usuario (Sección 10.4).

2. Metodología

Para la realización de este trabajo fin de grado se ha seguido Se siguió el modelo de desarrollo ágil Scrum [17] por los dos miembros del equipo. A continuación se detallan algunas particularidades:

- Se definieron *sprints* de una semana de duración. De esta forma se conseguía un seguimiento más exhaustivo del proyecto.
- Al final de cada *sprint* se realizaba una reunión con los tutores , que harían las veces tanto de *Scrum Master* como de *Product Owner*.
- En esa reunión se realizaba el *Sprint Review* con los productos, pruebas y conclusiones obtenidos del anterior *sprint*, y el *Sprint Planning* en el que se decidían que *historias de usuario* entrarían en la siguiente iteración.
- Se hizo una primera versión del product backlog a partir de los requisitos.

Generalmente se dividió cada *sprint* en cuatro fases: recolección de información y estudio, construcción y, verificación y pruebas. En las primeras iteraciones se dedicó un mayor esfuerzo en tareas de estudio, mientras que en las finales se trabajaron más las partes de construcción, verificación y pruebas. A continuación se explica brevemente cada una de estas fases.

- Recolección de información y estudio: Para cada *historia de usuario* del *sprint* se realizó un estudio previo del estado del arte. Para ello se revisaron artículos relacionados con la materia y páginas web dedicadas al tema en concreto.
- Construcción: Se desarrolló el código necesario para implementar las historias de usuario.
- Verificación y pruebas: Todo el código creado se testeó y se comprobó que los datos generados fueran correctos, empleando para ello una serie de casos de prueba.

En la sección 7.4 se explicará con más detalle cómo fueron resueltas las historias de usuario. Además, en el anexo 10.1 se puede encontrar un resumen de la planificación temporal del proyecto.

3. Requisitos

Para la creación del sistema se tuvieron en cuenta una serie de requisitos. A continuación se expondrán los requisitos relativos a la parte común del TFG:

RNF-01	**Desarrollo como aplicación web**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe de seguir una arquitectura web.
Importancia	Alta
Prioridad	Baja
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RNF-02	**Interfaz sencilla**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe tener una UI minimalista y sencilla de usar.
Importancia	Baja
Prioridad	Baja
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RNF-03	**Tiempo de respuesta**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe de tener un tiempo de respuesta respuesta bajo
Importancia	Baja
Prioridad	Baja
Asignado a:	David Barrientos Brenes

RNF-04	**Volumen de datos**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación solo debe trabajar correctamente con un volumen de datos pequeño.
Importancia	Baja
Prioridad	Baja
Asignado a:	David Barrientos Brenes

RNF-05	**Ampliación de las funcionalidades**
Tipo de requisito	No Funcional
Dependencias	
Descripción	La aplicación debe ampliar el desarollo realiazdo en la otra parte del tfg para que se contemple el *contexto formal multivaluado* y las obtención de dependencias.
Importancia	Alta
Prioridad	Alta
Asignado a:	David Barrientos Brenes

RF-01	**Subir una ontología**
Tipo de requisito	Funcional
Dependencias	
Descripción	Subir una ontología de atributos.
Importancia	Alta
Prioridad	Alta
Asignado a:	David Barrientos Brenes

RF-02	**Subir comentarios**
Tipo de requisito	Funcional
Dependencias	
Descripción	Subir una lista de comentarios.
Importancia	Alta
Prioridad	Alta
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RF-03	**Obtener un *contexto formal multivaluado***
Tipo de requisito	Funcional
Dependencias	▪ RF-01 ▪ RF-02
Descripción	Crear el *contexto formal multivaluado* a partir de los comentarios y la ontología subidas.
Importancia	Alta
Prioridad	Alta
Asignado a:	Christian Cintrano López y David Barrientos Brenes

RF-04	**Subir un *contexto formal multivaluado***
Tipo de requisito	Funcional
Dependencias	
Descripción	Subir un fichero con el *contexto formal multivaluado*.
Importancia	Alta
Prioridad	Alta
Asignado a:	David Barrientos Brenes

RF-05	**Descargar *contexto formal multivaluado***
Tipo de requisito	Funcional
Dependencias	▪ RF-04
Descripción	Descargar un *contexto formal multivaluado* con un formato similar al del requisito funcional 04.
Importancia	Media
Prioridad	Media
Asignado a:	David Barrientos Brenes

RF-06	**Obtener dependencias**
Tipo de requisito	Funcional
Dependencias	▪ RF-03
Descripción	A partir de un *contexto formal multivaluado* emplear algun algoritmo para obtener las dependencias funcionales.
Importancia	Alta
Prioridad	Alta
Asignado a:	David Barrientos Brenes

RF-07	**Descargar las dependencias**
Tipo de requisito	Funcional
Dependencias	▪ RF-06
Descripción	Descargar las dependencias obtenidas en el reqisuito funcional 06.
Importancia	Media
Prioridad	Media
Asignado a:	David Barrientos Brenes

En la sección 5 de esta memoria se analizarán con un mayor nivel de detalle estos requisitos.

4. Fundamentos y estado del arte

En esta sección se describe el estado del arte relativo a los sistemas de recomendación así como los fundamentos teóricos relativos a la teoría del *análisis formal de conceptos*.

4.1. Introducción

Hoy en día Internet es la mayor fuente de información, gracias al auge del comercio electrónico, las redes sociales, etc. De esta red podemos extraer gran cantidad de información, que puede resultar muy beneficiosa tanto para una empresa como para un usuario. Esta información suele llegar en forma de opiniones. Cuando los usuarios comentan alguna característica de un producto o servicio de una empresa, les proporcionan una gran cantidad de datos, que puede ser transformada en información útil. Esta información puede servir para proveer a los clientes de un mejor servicio y más personalizado. Para satisfacer esta necesidad nacen los sistemas de recomendación. Pero esta información no es nada si no es tratada y transformada en conocimiento útil. En esta línea se reúnen áreas tan relevantes hoy en día como la minería de datos, el análisis de sentimientos, big data, etc.

4.2. Big data en sistemas de recomendación

Como ya se ha mencionado las redes sociales son una gran fuente de información, principalmente sobre opiniones. Una inmensa cantidad de personas emplean Facebook, Twitter y otras más para expresar frecuentemente sus gustos y opiniones. Por ello, las empresas buscan medios para poder obtener y tratar esta cantidad gigantesca de datos. El *big data* nos proporciona un conjunto de técnicas y tecnologías para ello.

De manera sencilla, podemos ver el big data como el tratamiento de grandes cantidades de información de manera eficiente. Para ello usa la técnica del MapReduce, que permite la computación paralela sobre grandes colecciones de datos.

Big data posee un campo de estudio muy extenso, con gran cantidad de herramientas y tecnologías, tales como frameworks de computación paralela, herramientas de recolección de datos de redes sociales, o tecnologías especializadas para el almacenamiento de información.

4.3. Opiniones como eje principal de los sistemas de recomendación

Todos los sistemas de recomendación basados en redes sociales, y muchos otros más, tienen como base las opiniones de los usuarios.

Una opinión se puede definir como una quíntupla $(o_j, f_{jk}, oo_{ijkt}, h_i, t_t)$ [29]. Siendo o_j el objeto sobre el que se habla (p. e. un hotel), f_{jk} la característica o atributo del objeto

que se está valorando (p. e. la limpieza de la habitación o las vistas), h_i el *holder* o autor de la opinión, t_t la fecha en el que se realizó el comentario y oo_{ijkt} que es la valoración positiva, negativa o neutral de la opinión sobre la característica del objeto realizada por el *holder* en ese momento.

Esta definición de opinión puede expandirse y ser tratada con otros conceptos como una taxonomía de atributos o sinónimos.

La taxonomía de atributos amplia el termino objeto. La mayoría de los objetos poseen gran cantidad de características. Y sus componentes también pueden poseer atributos. Esto nos permite definir un objeto como $O = (T, A)$ [28], siendo T una taxonomía de partes con la relación parte de; y A un conjunto de atributos. Esto permite crear una jerarquía de atributos y componentes dentro de un mismo objeto. Con esta técnica se consigue una gestión de la información más eficiente. Además se debe de tener en cuenta que un mismo atributo puede pertenecer a varios objetos o partes y valorarse de manera distinta para cada uno de ellos.

Por otro lado se encuentran los sinónimos. Los sinónimos son de vital importancia a la hora de interpretar la opinión del usuario, ya que nos permite un mejor análisis y percepción de lo que quiere expresar. Existen una gran cantidad de técnicas dedicadas a resolver este problema, pero ninguna lo hace de manera eficiente. La mayoría de las reseñas solo hablan sobre el uso de diccionarios de términos o tesauros para agrupar los conjuntos de sinónimos [30].

4.4. Sistemas de recomendación en el sector turístico

Toda esta teoría se presenta de forma general sin entrar en ningún campo de estudio. Sin embargo, uno de los sectores en los que posee especial relevancia es en el sector turístico [27], que es el tratado en este TFG. En las opiniones sobre destinos turísticos (restaurantes, hoteles, parques, museos, etc.) existen dos elementos especialmente relevantes: el turista y los Puntos de Interés, usualmente abreviado como POIs [27]. En este tipo de sistemas obtiene una especial relevancia las características de estos usuarios. Estas pueden ser intrínsecas como lugar de procedencia, gustos, etc. o aspectos extrínsecos, por ejemplo el modo de realizar la visita turística (solo, en familia, con amigos, etc.) o las condiciones climáticas del día en cuestión.

Este último tipo de características son muy interesantes para los sistemas de recomendación sensibles al contexto. Estos sistemas emplean factores externos para dar la recomendación, por ejemplo recomendar ferias para salidas nocturnas o proponer ir a centros comerciales en días lluviosos.

Para la mayoría de entornos, la recomendación se realiza a un individuo concreto, sin embargo, el turismo es una actividad que comúnmente se realiza en grupo. Para mejorar la experiencia de estas agrupaciones existen varias técnicas llamadas filtrado en grupo.

Ellas intentan obtener una recomendación que pueda satisfacer al grupo en general más que a algunos individuos en concreto.

Por otro lado, los POIs pueden ser restaurantes, hoteles, paseos, parques, centros de ocio, etc. en resumen cualquier lugar que pueda ser objeto de interés para el turista. Estos forman el principal objeto de estudio en la recomendación y son la base para construir los sistemas.

4.5. Filtrado de información

Existen una gran cantidad de algoritmos y técnicas distintas en el campo de la recomendación. Gracias a ellas se consigue el filtrado de la información menos relevante para el usuario. Estos sistemas de filtrado, aunque suelen ser muy distintos entre sí, se pueden catalogar en:

- Colaborativos: Se basan en que los usuarios dan una serie de calificaciones a un conjunto de objetos. Estas calificaciones se asignan mediante un rango de valores y describe de manera subjetiva como de "bueno" es el objeto. De esta forma el sistema recomienda al usuario una serie de objetos que fueron valorados positivamente por otros usuarios con calificaciones similares a las del recomendado.

- Basados en contenido: Similar al anterior. Recomiendan objetos similares que los que se valoraron positivamente en el pasado.

- Demográficos: La recomendación se enfoca a conjuntos de usuarios, agrupándolos por geografía, sexo, edad, etc.

- Basados en conocimiento: Tienen información sobre como un objeto satisface a un usuario, y establece una relación entre sus necesidades y la recomendación.

4.6. Análisis Formal de Conceptos

Para poder obtener información de las opiniones de los usuarios es necesario algún proceso de análisis con el que conseguir una forma más eficiente y ordenada de estructurar los datos. Para ello, en este trabajo, se hace uso de una teoría matemática llamada Análisis Formal de Conceptos (Formal Concept Analysis o FCA) [24].

Esta teoría propuesta por Bernhard Ganter ha tenido una gran relevancia gracias a sus buenos resultados y su simplicidad. Se basa en lo que Ganter denomina un *contexto formal*, que no es más que una tupla (G, M, I) tal que: G es un conjunto de objetos de estudio, M un conjunto de atributos e I una relación binaria entre G y M, $I \subseteq G \times M$. En nuestro caso, podemos observar los diferentes elementos de la tupla (G, M, I) como:

- G son los usuarios.

- M las características de POIs.
- I relación que se establece entre una opinión de un usuario (objeto) y las características (atributo) del POI sobre el que opina.

Esta tupla puede ser representada como una tabla, siendo G las filas, M las columnas y I el valor 0 o 1 de una celda, según exista o no la relación.

En el contexto podemos definir relaciones binarias entre los conjuntos G M llamadas Conexiones de Galois y son la base para la definición formal de *concepto* de un *contexto formal*. Un *concepto* es un conjunto de objetos y atributos con nombre y significado. Además poseen una estructura de retículo. Todo ello nos permite generar grupos de usuarios con los mismos intereses u opiniones. Para poder obtener todos los *conceptos* de un *contexto formal* existen diversos algoritmos, entre los cuales se destacan:

- NextClousure: Fue propuesto por el propio Ganter y fue el primero. Calcula de forma sencilla como obtener todos los conceptos de un contexto formal pero tiene muy poca eficiencia [24].
- Titanic: Se basa en la obtención de una función de peso que reduce en gran medida el número de cierres a realizar. Titanic no reduce la complejidad del problema pero si lo vuelve más eficiente [33].

Gracias al FCA y a estos algoritmos, es posible emplear técnicas de recomendación, tales como el filtrado colaborativo, o descubrir tendencias en los gustos de los usuarios.

Aun así, esta teoría posee una gran limitación, los valores del *contexto formal* son binarios; es decir, si observamos el *contexto* como una tabla las filas serían los objetos, los atributos las columnas y las celdas tomarían el valor de 0 o 1 según existiera la relación entre el objeto y el atributo correspondiente. En el mundo real, podemos encontrar que la mayoría de los datos son de carácter heterogéneo, como por ejemplo la meteorología, que puede tomar valores como soleado, lluvioso, nublado, etc. y con los que la teoría FCA no puede trabajar.

Este problema es posible resolverlo de múltiples formas. Un modo posible es creando intervalos discretos de un atributo con múltiples valores [25]. Por ejemplo, el atributo de meteorología que puede tomar valores como lluvioso, soleado, nublado. Este método divide el atributo meteorología en una serie de valores. Así, obtenemos que meteorología ya no es un atributo, pero si lluvioso, soleado y nublado; y la relación sólo se establecerá con uno de esos atributos.

Otra forma posible de solucionar dicho problema es considerar que la relación entre cierto objeto y atributo existe si dicho objeto toma un determinado valor. Este valor pertenece a un conjunto de valores llamados *valores de verdad* [23].

Todas estas formas de homogeneizar los valores provocan la perdida de información. Por ello, la mejor forma de resolver dicho problema es creando una nueva teoría llamada análisis formal de conceptos multivaluado o Formal Concept Analysis Many-Valued (FCAMV)[34]. Con esta nueva teoría se persigue mantener las mismas propiedades que en FCA pero haciendo uso de múltiples valores en lugar de binarios. Para ello debemos de adaptar los conceptos de dicha teoría. Un *contexto formal multivaluado* (FCMV) es denotado por la tupla (G, M, W, I) donde G es el conjunto de objetos, M el conjunto de atributos, W es el conjunto de valores de los atributos e I es una relación ternaria entre G,M e I tal que $(I \subseteq G \times M \times W)$, o lo que es lo mismo para $(g, m, w) \in I$ podemos afirmar el hecho de que el atributo m toma el valor w para el objeto g (también puede ser denotado por $m(g) = w$).

El término *retículo de conceptos* también es adaptado en el FCAMV. Para ello se emplea el concepto de *retículo de conceptos multivaluado*:

- Dado un contexto MV (G, M, W, I) y un umbral $\theta \in [0, 1]$: dos objetos g_i y g_j en G comparte un atributo m en M si y sólo si $m(g_i) = w_i$, $m(g_j) = w_j$, y $|w_i - w_j| \leq \theta$. Asumiendo que $w_i \leq w_j$, g_i y g_j comparten $m[w_i, w_j]$, el intervalo $[w_i, w_j]$ es llamado intervalo de similaridad de m para g_i y g_j, y w_i y w_j se dice que son similares.

- Siendo A un conjunto tal que $A \subseteq G$ de objetos que comparten un atributo m siempre que cualquier dos objetos en A compartan m. El intervalo de similaridad de m para A es $[min_{g \in A}(m(g)), max_{g \in A}(m(g))]$ y el atributo m compartido por los objetos en A es denotado por $m_{[min_{g \in A}(m(g)), max_{g \in A}(m(g))]}$.

- Dado un contexto MV (G, M, W, I), un umbral $\theta \in [0, 1]$ y I_θ el conjunto de todos los intervalos $[\alpha, \beta]$ tales que $\beta - \alpha \leq \theta$ con $[\alpha, \beta] \subseteq [0, 1]$:

  - Para un conjunto $A \subseteq G$: $A'_\theta = \{m[\alpha, \beta] \in M \times I_\theta$ tal que $m(g) = \oslash$ y $\forall g_i, g_j \in A, |m(g_i) - m(g_j)| \leq \theta, \alpha = min_{g \in A}(m(g))$ *y* $\beta = max_{g \in A}(m(g))\}$.
  - Para un conjunto $B \subseteq M \times I_\theta : B_\theta = \{g \in G$ *tales que* $\forall m[\alpha, \beta] \in B, m(g) \in [\alpha, \beta]\}$.

- Un *concepto MV* es un par A, B donde $A \subseteq G$ y $B \subseteq M \times I_\theta$ tales que t $A_\theta = B$ y $B'_\theta = A$.

- Si A_1B_1 y ${}_A2B_2$ son conceptos MV, A_1B_1 es un subconcepto de A_2B_2 cuando $A_1 \subseteq A_2($ *o* $B_2 \subseteq_\theta B_1)$. El conjunto ordenado de todos los conceptos MV de G, M, W, I es llamado *retículo de conceptos multivaluado.*

Con esto, el FCAMV tiene las mismas propiedades que el FCA pero con la ventaja de poder usar datos heterogéneos. De este modo, podemos obtener mucha más información

aplicando ciertos algoritmos, como los algoritmos de obtención de dependencias funcionales. Una *dependencia funcional* (Funcional Dependency, FD) expresa una restricción de valor entre atributos en una relación [31]. Formalmente, para un esquema relacional R con $X \subseteq R$ y $A \in R$, una dependencia funcional $X \rightarrow A$ puede ser definida si para todo par de tuplas t_1 y t_2 sobre R, para todo $B \in X$ si $t_1[B] = t_2[B]$ entonces $t_1[A] = t_2[A]$.

Gracias a esto podemos, por ejemplo, reducir enormemente el trabajo que debe realizar un motor de recomendaciones para poder realizar alguna recomendación a un usuario. Con conocer unos pocos datos de un usuario, podemos ir deduciendo los demás gracias a las dependencias y mostrarle en poco tiempo y eficazmente la recomendación.

Para obtener estas dependencias funcionales, podemos hacer uso de varios algoritmos, como por ejemplo Tane [26]. Tane fue uno de los primeros algoritmos de obtención de dependencias funcionales de manera eficiente. Este algoritmo consiste en hacer particiones en conjuntos de filas según los valores de los atributos. Además, realiza rápidamente pruebas de validez de FD, incluso con una gran cantidad de filas. Posteriormente surgieron algoritmos que mejoran la eficiencia de Tane, a la vez que aumentan la cantidad de información que se puede obtener. Un ejemplo puede ser el algoritmo FUN [32]. Sin embargo, el algoritmo FD Mine [22] reduce significativamente el tiempo de ejecución y de consultas con respecto al Tane o FUN. Además nos proporciona un conjunto de claves y de equivalencias de atributos. Este algoritmo se basa en los *axiomas de Armstrong* que se fundamentan en tres reglas básicas:

- Regla de reflexibilidad: si $Y \subseteq X$, entonces $X \rightarrow Y$
- Regla de argumentación: si $X \rightarrow Y$, entonces $XZ \rightarrow YZ$
- Regla de transitividad: si $X \rightarrow Y$ y si $Y \rightarrow Z$, entonces si $X \rightarrow Z$

Gracias a estas reglas es posible podar, en gran medida, las comprobaciones entre los diferentes conjuntos de atributos. Previamente es necesario definir estos cuatro conceptos:

- Candidato: conjunto de atributos que podrían tener alguna dependencia funcional con algún otro atributo.
- Equivalencia de candidatos: dado X e Y del conjunto de atributos, si $X \rightarrow Y$ y $Y \rightarrow X$, entonces decimos que X e Y son equivalentes, y se denota como $X \leftrightarrow Y$. Gracias a esta regla, y a otras que se deducen de ella, es posible calcular el conjunto de atributos equivalentes a la par que se reduce el número de comprobaciones de FD.
- Cierre no-trivial: dado un conjunto de dependencias funcionales F sobre el conjunto de datos D y un posible candidato X, el cierre del candidato X respecto a F se denota como $Clousure(X)$ y es definido como $\{Y | X \rightarrow Y$ puede ser deducido de F

por los axiomas de Armstrong}. El cierre no-trivial no es más que el cierre de X sin el elemento X, o lo que es lo mismo, $Closure'(X) = Closure(X) - X$.

- Reglas de Poda: existen varias reglas de poda que son usadas para reducir la lista de candidatos a comprobar. Un ejemplo es: Si $X, Y \in$ *Conjunto de Candidatos* y $X \leftrightarrow Y$, entonces Y puede ser borrada de dicho conjunto.

El algoritmo FD Mine consta de varias fases:

1. Inicialización de las variables. El conjunto de candidatos inicialmente es el conjunto de atributos.

2. Mientras que el conjunto de candidatos sea diferente del vacío, repetir:

   a) Para todo el conjunto de candidatos, comprobar con cada atributo si existe una dependencia funcional.

   b) Cotejar si algún candidato es clave.

   c) Examinar si algún candidato es equivalente a otro candidato.

   d) Podar el conjunto de candidatos.

   e) Generar el siguiente nivel de candidatos, añadiendo a cada elemento un atributo más que no tuviera.

5. Análisis de requisitos

En las siguientes secciones se analizarán cada uno de los requisitos funcionales enunciados en la sección 3, que llevaron hasta la solución elegida.

5.1. RF-01 Subir ontología

Para obtener un contexto formal multivaluado es necesario prefijar cada atributo con sus diferentes posibles valores. Esto es necesario para que, analizando los comentarios, podamos obtener si un atributo ha sido nombrado y con qué valor. Por ello, es imprescindible contar con una ontología preparada por un experto. Dicha ontología puede representarse en forma de árbol en tres niveles, como se ilustra en la figura 1, tal que:

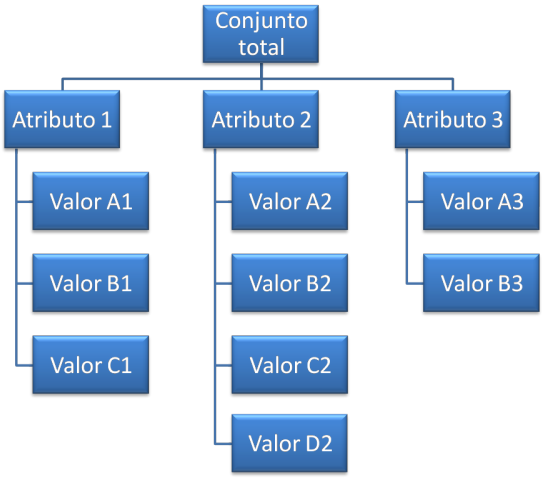

Figura 1: Ejemplo de ontología de atributos

- Primer nivel: equivale al conjunto total de todos los atributos.
- Segundo nivel: este nivel contiene a todos los atributos, cada uno separado en una rama.
- Tercer nivel: las hojas son los diferentes valores que puede adoptar cada atributo.

Sería deseable una ampliación de este árbol con un cuarto nivel extra: sinónimos. Como el tercer nivel representa los diferentes valores de un atributo, los sinónimos representarían las diferentes formas de hacer referencia a el mismo atributo. De este modo, la ontología sería mucho más potente y robusta, a la par que aumentaría la cantidad de información que podemos recibir de los comentarios de cada usuario. Sin embargo debido a su complejidad, queda fuera del alcance de este proyecto.

Para la creación de la ontología se usará un fichero JSON [13] con cada atributo y sus diferentes valores. Se ha elegido JSON por su simplicidad y su sencilla pero potente

estructura de datos, que es fácilmente manipulable desde Java, el lenguaje seleccionado para el desarrollo de la aplicación. La estructura sería tal que así:

```
"atribubte1" : {"valuetr1.1", "valuetr1.2","valuetr1.3"},
"atribubte2" : {"valuetr3.2", "valuetr2.2","valuetr2.3"},
```

5.2. RF-02 Subir comentarios

Un *contexto formal multivaluado* puede representarse en forma de tabla, siendo los atributos las columnas y los objetos las filas. El conjunto de atributos se consigue gracias a la ontología, en cambio, los objetos que rellenan la tabla se obtienen de los comentarios de los usuarios.

Como fuente de información se ha usado los comentarios de TripAdvisor [19], captados manualmente. Aunque no entra en el caso de estudio de este TFG, sería deseable la integración con algún servicio web o API para automatizar este proceso.

Para obtener información útil de los comentarios, es necesario que se analicen. Esta información puede definirse como el conjunto de *keywords* o conjunto palabras claves. Para conseguirlas, se investigó un gran amplio abanico de herramientas, tales como Gate [6] o R [16]. Todas ellas se desecharon ya que no cumplían con las expectativas. En cambio, y siendo ésta la herramienta elegida, AlchemyAPI [2] si nos proveía de un modo rápido y directo el conjunto de *keywords* de cada texto, generando un documento con los datos del análisis.

Una vez obtenidos el conjunto de *keywords* de cada comentario, solo resta comprobar con cada *keyword* si adopta el de valor de algún atributo de la ontología.

5.3. RF-03 Obtener *contexto formal multivaluado*

Con los comentarios y la ontología, se crea un *contexto formal multivaluado*, que será representado en forma de tabla, siendo la cabecera los atributos y las filas los valores posibles, reflejados en la ontología. El proceso de creación es simple: se analizan las palabras de un comentario y se obtienen las *keyword*. Seguidamente, comprueba con cada atributo si alguna *keyword* conincide con el valor de cada atributo. Si lo hace, se añade ese valor a la tabla. Así sucesivamente con todos los comentarios introducidos.

5.4. RF-04 Subir *contexto formal multivaluado*

La aplicación permitirá subir un *fcmv* en formato csv. Este contexo será mostrado luego al igual que en requisito funcional 03, y deberá tener su mismo formato.

5.5. RF-05 Descargar *contexto formal multivaluado*

Una vez creado el *fcmv*, podrá ser descargado para su posterior analisis. El archivo está en formado csv y formateado igual que la tabla presentada en el requisito funcional 03 y la subida en el requisito funcional 04.

5.6. RF-06 Obtener dependencias

Existen diversos algoritmos de extracción de dependencias funcionales a partir de una tabla. Por ello, se realizó un estudio sobre que algoritmo usar. Para poder hacer uso de un contexto formal multivaluado, era necesario que aceptara cualquier conjunto de valores, no solo binarios. Primero se investigó Tane, uno de los primeros algoritmos de este tipo, y del cual nacen muchos otros algoritmos, tales como el FUN o FDMine. Después de un análisis sobre estos tres algoritmos, se decidió emplear FDMine.

FDmine no disminuye la complejidad respecto a Tane o FUN, se mantiene en 2^n, siendo n el numero de atributos. El numero de filas no influye en la complejidad. En cambio, si reduce en gran medida el número de comprobaciones de posibles dependencias funcionales, lo que le da una eficiencia aun mayor que los otros dos algoritmos. Además, aporta más información, como el conjunto de atributos equivalentes o el de claves primarias.

5.7. RF-07 Descargar dependencias

Desde la aplicación se podrá descargar las dependencias obtenidas, una vez que sean analizadas. El fichero estará en formato csv, con la primer columna el conjunto de atributos separados por espacios, y en las siguientes columnas sus dependencias.

6. Entorno tecnológico

Se ha desarrollado una aplicación web Java EE (antes conocido como J2EE) versión 6, implementada haciendo uso de los siguientes frameworks y librerías

- JBoss 6 [7]: Servidor de aplicaciones. Fue elegido debido a la familiaridad previa que se tenía con el mismo.
- Spring v4 [18]: Framework de desarrollo de entornos web. Es una de las mejores tecnologías de modelo vista-controlador. Es fiable, posee una gran cantidad de módulos fácilmente configurables y es fácil de usar. Se ha empleado como controlador de la aplicación web.
- Bootstrap 2.3.1 (look & feel) [4]: Hojas de diseños originalmente diseñada por Twitter. Se ha vuelto muy popular debido a los buenos resultados que se obtienen y a su sencillez.
- JSP: Para el diseño de la interfaz web. Son rápidos, potentes, dinámicos, sencillos y altamente extendidos. Su simplicidad fue la clave para su selección.
- JTLS: Etiquetas para JSP que añaden funcionalidad extra. Concretamente se han empleado para sentencias de control de flujo <c:if> y <c:forEach> además de acceso a los ficheros .properties para implementar el multilenguaje.
- Apache Ant (Solo en front-end) [3]: Librería para la compilación del código.
- Maven (Solo en back-end) [15]: Herramienta para el control de configuración. Ampliamente extendido, potente y fácil de usar.
- AlchemyAPI [2]: Librería para el procesamiento del lenguaje natural que ofrece una amplia gama de funcionalidades. Se usa bajo licencia de compra, aunque posee licencias de prueba y académica (limitada a un número máximo de peticiones).

Además se han empleado una serie de plugins para mejorar la funcionalidad y la apariencia de la web.

- jQuery [8]: Biblioteca de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Es la librería de JavaScript mas utilizada. Además de JQuery se hace uso de una serie de plugins concretos de su biblioteca.
  - JQuery BlockUI [9]: Bloquear la pantalla mientras se procesa la información.
  - jQuery jGrowl [11]: Mostrar alertas.

    - jQuery Upload File [12]: Subir archivos de manera eficiente, integrable con AJAX y permitiendo Drag & Drop.
    - jQuery File Download [10]: Descargar archivos.
    - Wizard [21]: Barra de flujo de trabajo.
- AJAX [1]: Empleado para cargar solo lo necesario de la página. Permite además gestión de eventos y control de las peticiones al servidor.

La parte específica del TFG se desarrolló como una ampliación de la funcionalidad del sistema desarrollado en la parte común, por lo que solo se añadió al código de back-end la siguiente librería y herramienta:

- JSON.simple [14]: Librería para el tratamiento de archivos JSON.

En el esquema representado en la figura 2 se resume el funcionamiento básico del sistema completo:

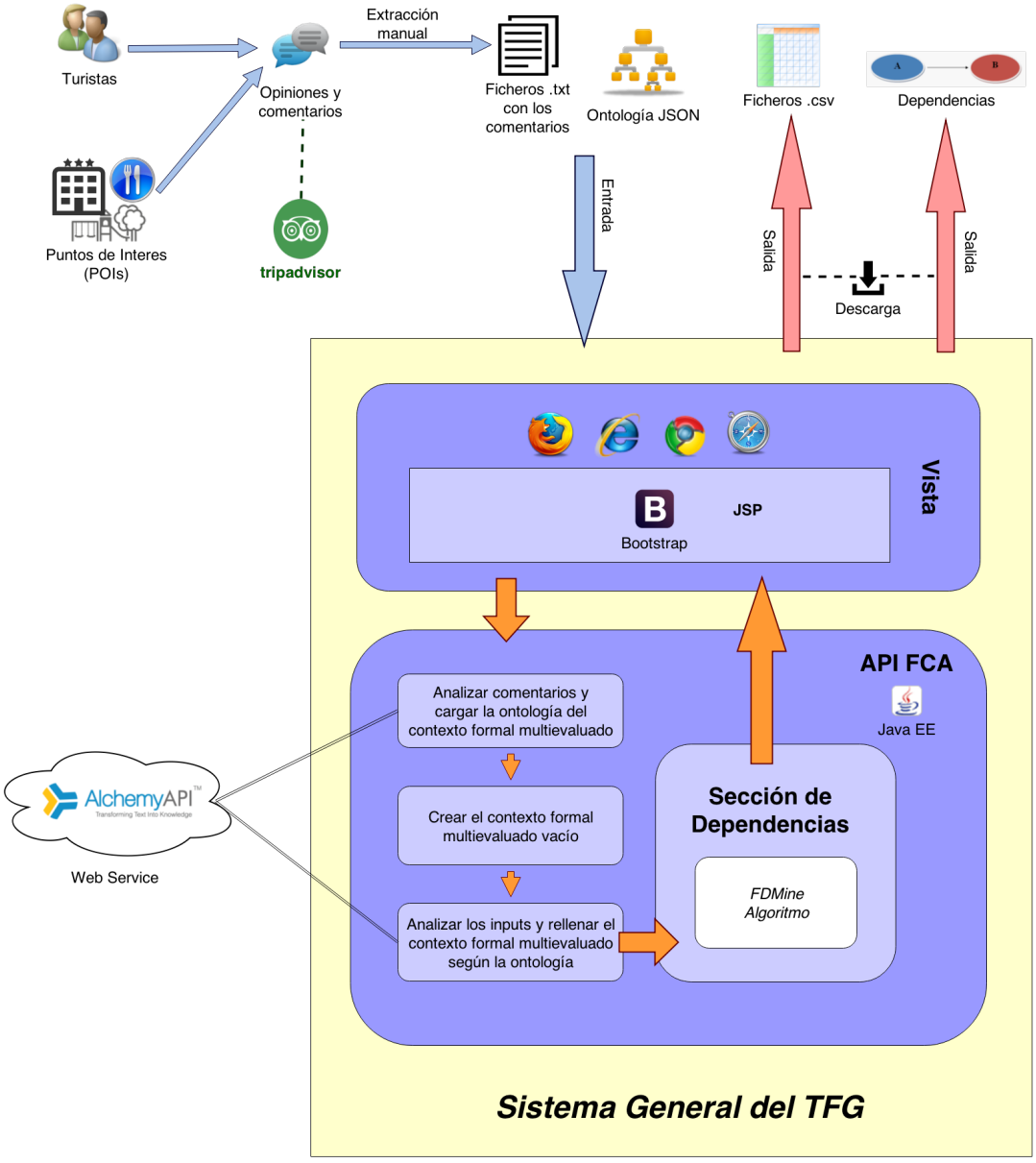

Figura 2: Diagrama de arquitectura del sistema completo

7. Desarrollo

En esta sección se expondrán los principales aspectos del desarrollo realizado para construir el sistema.

7.1. Aspectos básicos

El desarrollo se realizó sobre el sistema operativo Windows 7 empleando el IDE Eclipse [5] para la construcción de la aplicación. El sistema se divide en dos proyectos diferentes:

- FCAApplication: Interfaz web para poder visualizar el trabajo realizado sobre FCA y proporcionar una UI. Es el front-end del sistema.
- FCAUtilitiesProject: Una API con todas las funcionalidades relacionadas con FCA y el FCAMV desarrolladas para el sistema descrito en esta memoria. Es el back-end del sistema. El desarrollo de la parte específica del TFG se realizó ampliando la funcionalidad de esta API.

Se tomó la decisión de realizar esta separación para mejorar su integración con otras aplicaciones. Se siguió esta estructura para seguir con la arquitectura que realizó mi compañero de trabajo. De este modo, sigue manteniendo las funcionalidades anteriores a la par que se añaden nuevas, descritas en la sección 7.3.

7.2. Estructura de los proyectos

A continuación se presenta la estructura de los proyectos desarrollados. Todo el código desarrollado parte de la base creada por mi compañero de proyecto, por lo que las estructuras son similares para mantener la compatibilidad. Por ello, en esta sección se hará hincapié en los elementos desarrollados en esta parte.

7.2.1. FCAApplication

La extructura es la de un proyecto Java Web siguiendo el patrón modelo-vista-controlador (ver figura 3). En él se distinguen las siguientes partes:

- src.controllers, en donde se encuentra el controlador de la aplicación
- build, carpeta en la que se almacena el FCAApplication.war
- Libraries, librerías externas necesarias para la aplicación. En esta carpeta va incluido el código compilado de la API de FCA: FCAUtilitiesProject.jar
- web, en donde se encuentra el código relativo a la vista de la aplicación. Se compone de los directorios:

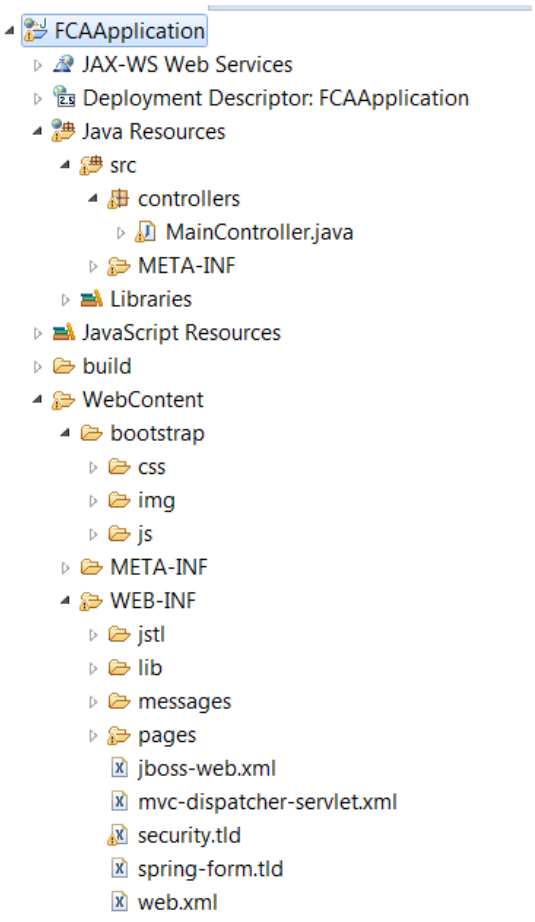

Figura 3: Proyecto de front-end: FCAApplication

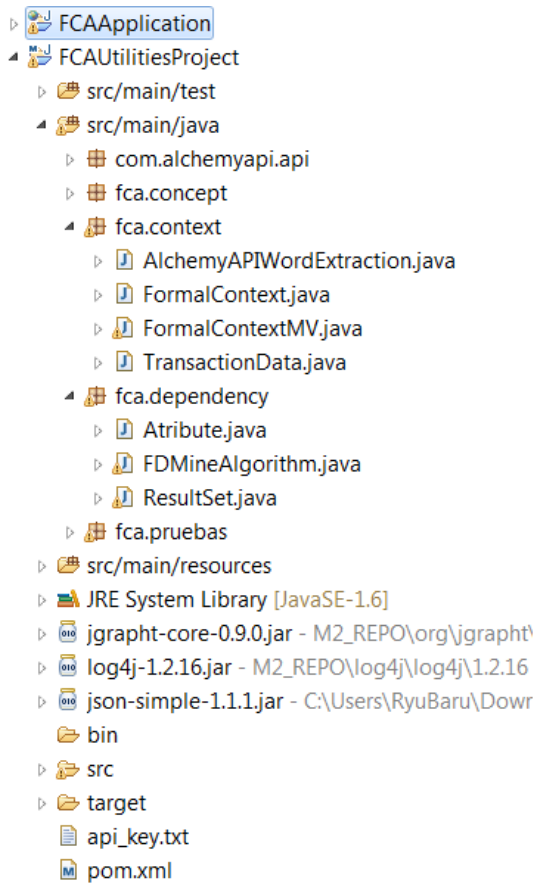

Figura 4: Proyecto de back-end: FCAUtilitiesProject

- bootstrap, que contiene los archivos css, imagen y javascript.
- WEB-INF, que se compone a su vez de:
  - jstl: contiene los archivos .tld necesarios para el multi-idioma.
  - messages: en él se encuentran los archivos .properties para mostrar el texto en diferentes idiomas. Solo se han implementado los idiomas español e inglés (el idioma por defecto es inglés).
  - pages, que contiene las páginas JSP de la aplicación.

- build.xml, archivo Ant para la compilación.

7.2.2. FCAUtilitiesProject

Este proyecto se estructura como un proyecto Maven y está dividido en 4 paquetes (ver figura 4):

- com.alchemyapi.api: contiene los códigos referente a AlchemyAPI.
- fca.dependecy: en él se encuentra la implementación del algoritmo FDMine y clases auxiliares.

- fca.context: paquete principal en el que se encuentran las clases básicas para trabajar con *contextos formales multivaluados*.
- fca.pruebas: en su interior se encuentran las clases de pruebas que se empleó en el desarrollo del back-end.

7.3. Modelos

Todos los modelos realizados han sido creados usando la herrmanienta de modelado de UML MagicDraw. Todos estos modelos mostrados a continuación están adjuntados en la entrega del TFG.

7.3.1. Casos de Uso

En la figura 5 se ilustran los casos de uso principales del sistema.

- Subir ontología: Subir una ontología en formato JSON. El usuario pulsa subir ontología, selecciona la que desea y pulsa el botón aceptar.
- Subir comentarios: Subir una serie de comentarios en formato txt. El usuario pulsa subir comentarios, selecciona los comentarios y pulsa el botón aceptar.
- Subir contexto formal multivaluado: Subir un *contexto formal multivaliado* con la extensión csv. El usuario pulsa subir *contexto formal multivaluado*, selecciona un contexto y pulsa el botón aceptar.
- Obtener un *contexto formal multivaliado*: Se crea una tabla que represente el *contexto formal multivaluado* a partir de la ontología y los comentarios. El usuario pulsa obtener *contexto formal multivaliado*, mostrándose una tabla que representa el contexto.
- Descargar *contexto formal multivaliado*: Descargar la tabla obtenida con formato csv. El usuario pulsa descargar *contexto formal multivaliado*, obteniendo un archivo que contenga dicho contexto.
- Obtener dependencias: Obtener el conjunto de dependencias a partir de un *contexto formal multivaliado*. El usuario pulsa obtener dependencias y éstas serán mostradas en forma de lista.
- Descargar dependencias: Descargar el conjunto de dependencias obtenidos, en formato csv. El usuario pulsa descargar dependencias, obteniendo un archivo que contenga dichas dependencias.

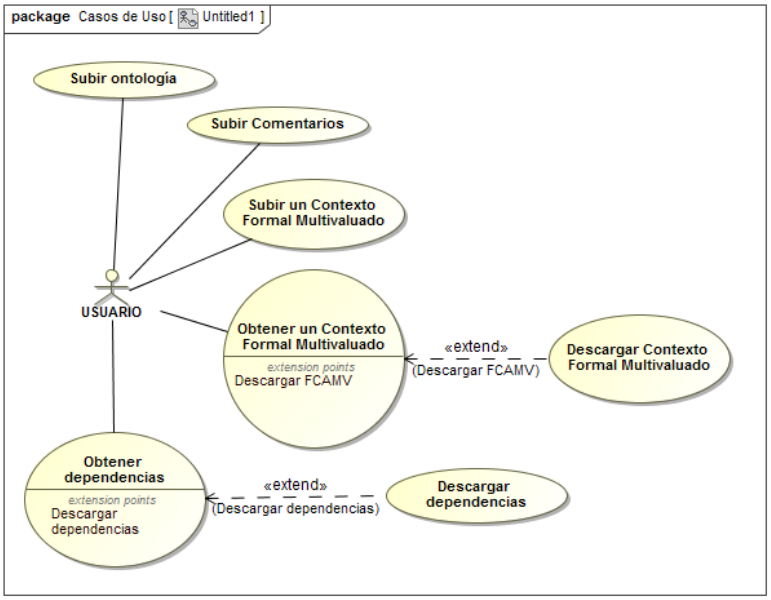

Figura 5: Diagrama con los casos de uso

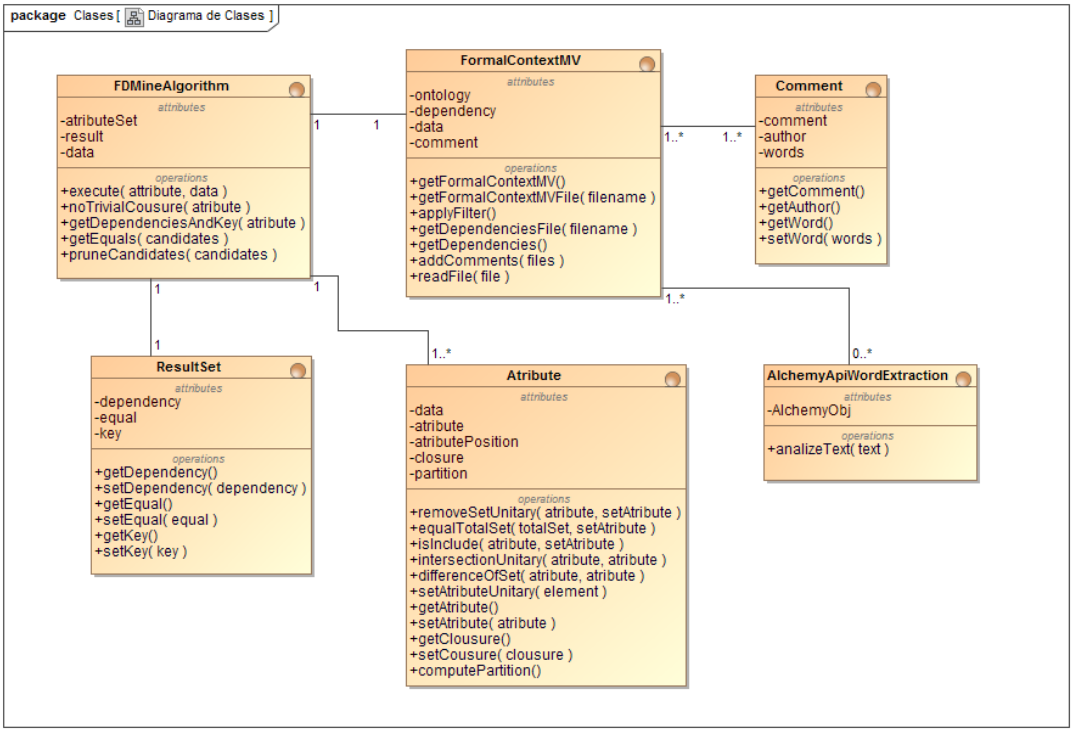

Figura 6: Diagrama de clases

7.3.2. Diagramas de Clases

Para la implementación de FCAUtilitiesProject se desarrollaron diferentes clases, que se analizaran a continuación. Estas clases se ilustran en el diagrama de clases de la figura 6.

FormalContextMV Esta clase representa a un *contexto formal multivaluado* y es la clase principal del proyecto. Se compone de una serie de constructores y métodos, los cuales se describen a continuación

Constructores

```
public FormalContextMV();
public FormalContext(File ontology, List<File> files); // Constructor
 principal
public FormalContext(File fcmv); // Constructor principal
```

Los dos constructores principales son, en primer caso, para introducir una ontología en extensión JSON y una lista de comentarios en txt, y en segundo caso, para introducir un contexto formal multivaluado con extensión csv. Este primero realizará una llamada al codigo de AlchemyAPI para procesar los textos. Cabe destacar que el *contexto multivaluado* se rellena de la siguiente forma:

- Si el comentario contiene una *keyword* perteneciente a un atributo de la ontología, se colocara esa keyword en ese atributo.
- En caso contrario un "NA".

Métodos sobre el contexto formal multivaluado Los métodos principales para interactuar con el *contexto formal multivaluado* son los siguientes:

```
public List<List<String>> getFormalContextMV() {};
public File getFormalContextMVFile(String fileName) {};
```

El primer método sirve para obtener el contexto formal multivaluado. El segundo método es usado para poder almacenar en un archivo csv dicho *contexto*.

Métodos sobre las dependencias La parte específica del TFG trata sobre las *dependencias funcionales*. Estos son los métodos que proporcionan las funcionalidades necesarias:

```
public ResultSet getDependecies() {};
public File getDependeciesFile(String fileName) {};
```

Estos métodos nos devuelven las *dependencias* en la clase ResultSet que será explicada más adelante. Ambos poseen llamadas a la implementación del algoritmo FDMine.

AlchemyAPIWordExtraction Esta clase es la encargada de realizar la extracción de las palabras claves de los textos que posteriormente serán cribadas para obtener los atributos del *contexto multivaluado*. Es una abstracción de la API de AlchemyAPI.

FDmineAlgorithm Es una implementación directa del algoritmo FDMINE. Consta de un único método público

```
public static ResultSet execute (Map<String, Integer> atributePosition,
    List<List<String>> data) {};
```

Este método recibe como entrada un mapa con el nombre de los atributos y su posición en la tabla del *contexto formal multivaluado*, y una lista de listas de String , que representa la tabla del *contexto formal multivaluado*. El resultado es un objeto de la clase ResultSet, que contendrá todas las dependencias que se obtienen. Para facilitar la lectura del código se mantuvieron tanto los nombres de los métodos como los de las variables que aparecen en la definición de FDMINE. Estos métodos son:

```
private static void ComputeNonTrivialClosure (Atribute) {};
private static void ObtaintFDandKey (Atribute) {};
private static void ObtainEQSet (Set<Atribute>) {};
private static void PruneCandidates (Set<Atribute>) {};
private static Set<Atribute> GenerateNextLevelCandidates (Set<Atribute>)
 {};
//Method to remove de dependencies of type  X -> Y, XZ -> YZ
private static void isRedundant (Atribute) {};
```

Atribute Representa a un atributo o conjunto de atributos del algoritmo FDMINE. Sus principales parámetros son:

```
// Set of attributes. Represent the attribute
private Set<String> attribute;

// Clousure of the Attribute
private Set<Attribute>  clousure;

//The partition of the Attribute
private Set<List<String>> paritition;
```

Además la clase cuenta con una serie de métodos para realizar operaciones con conjuntos.

ResultSet Representa al conjunto que se obtiene al ejecutar el algoritmo FDMINE. Sus principales parámetros son:

```
//Represent the set of attribute and its dependencies
private Map<Atribute,Set<Atribute>> dependency;
```

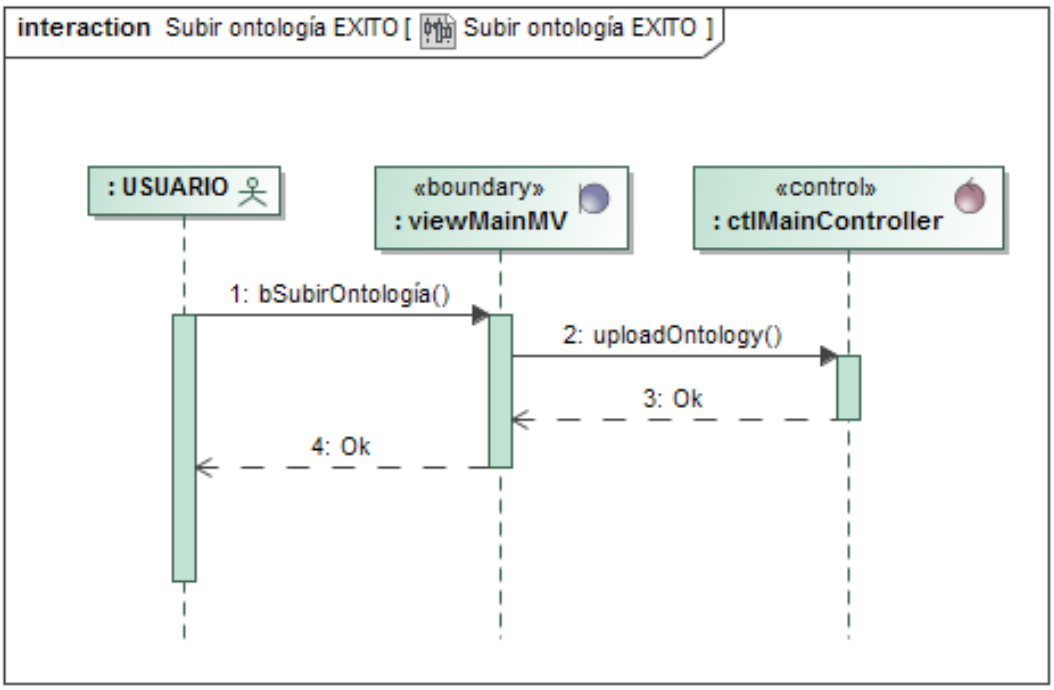

Figura 7: Diagrama de secuencia del caso de uso *Subir ontologias*

```
// Set of attribute that are equal
private Map<Atribute,Atribute> equal;

//Set of attribute that are primary key
private Set<Atribute> key;
```

Además, esta clase cuenta con sus correspondientes *getters and setters*.

7.3.3. Diagramas de Secuencia

Los diagramas de secuencia representados en las figuras 7, 8,9 , 10, 11, 12 y 13 incluidos en esta sección ilustran los comportamientos básicos y exitosos de los casos de uso descritos anteriormente.

7.4. Historias de usuario

A continuación se describe como se fueron resueltas las historias de usuario.

1. Inicialmente se desarrolló el algoritmo FDMine con sus clases auxiliares. Se realizaron pruebas de cada función y pruebas completas de todo el sistema.

2. Tras analizar los resultados obtenidos por el algoritmo FDMine, se implementó una mejora del mismo, que eliminaba las dependencias del tipo: $X \rightarrow Y$, $XZ \rightarrow YZ$

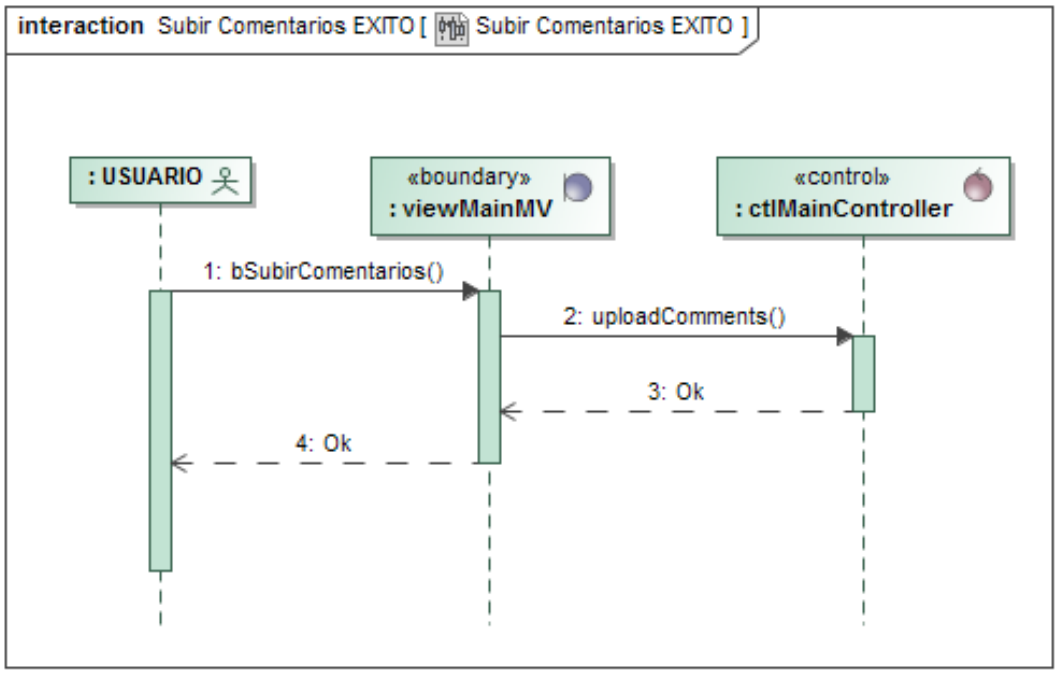

Figura 8: Diagrama de secuencia del caso de uso *Subir comentarios*

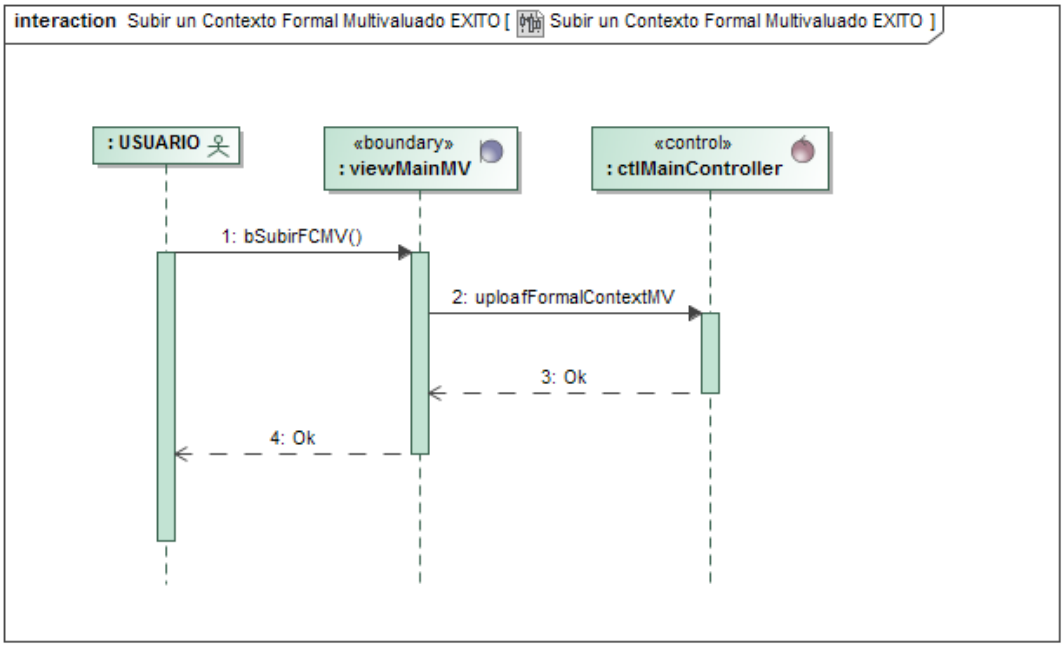

Figura 9: Diagrama de secuencia del caso de uso *Subir un contexto formal multivaluado*

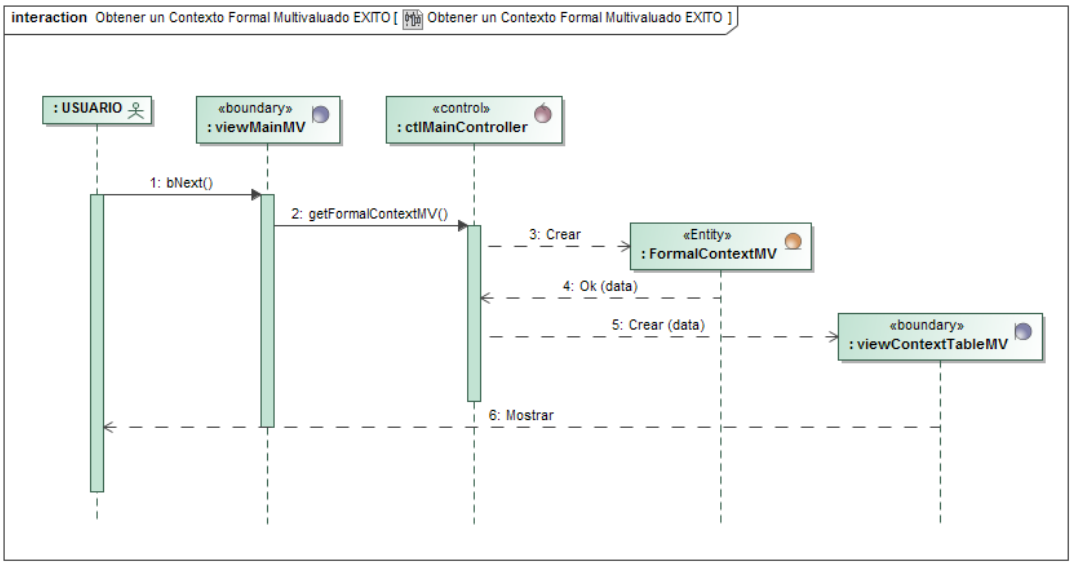

Figura 10: Diagrama de secuencia del caso de uso *Obtener el contexto formal multivaluado*

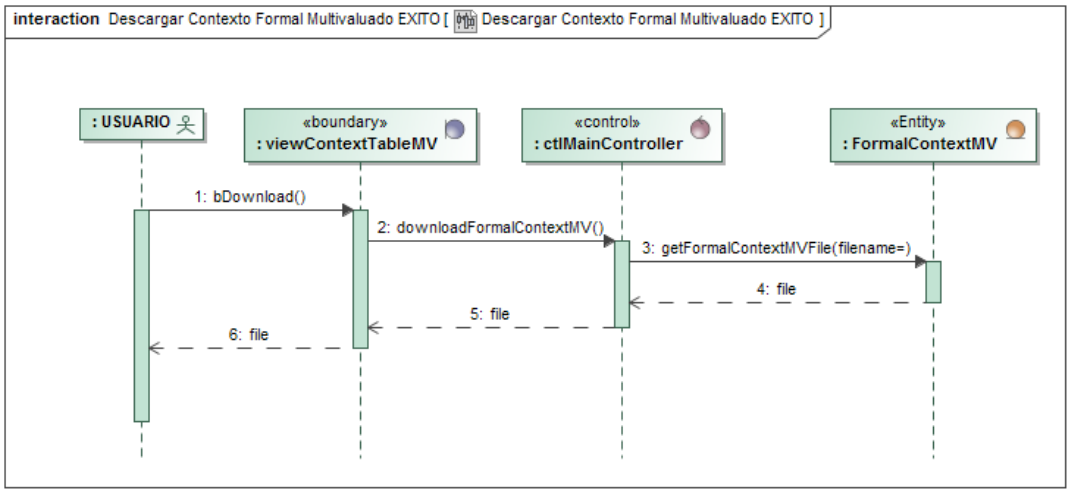

Figura 11: Diagrama de secuencia del caso de uso *Descargar contexto formal multivaluado*

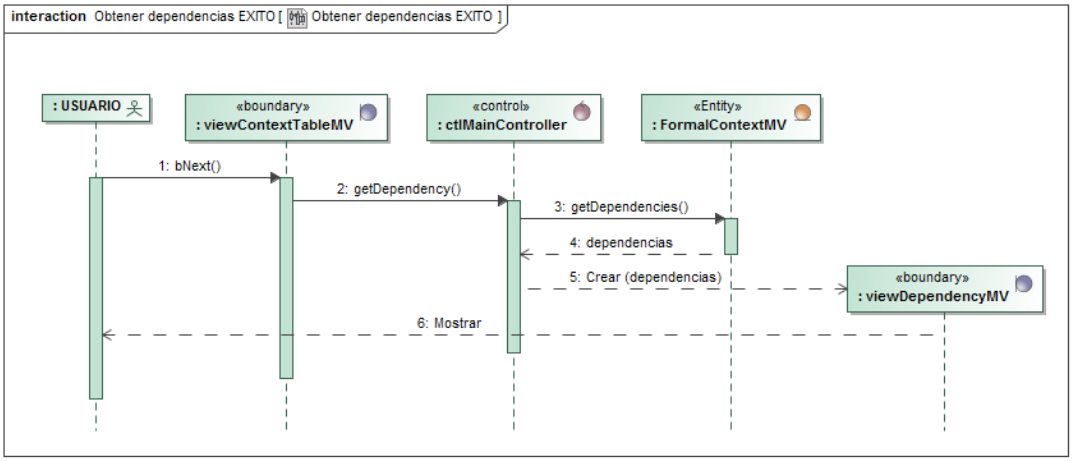

Figura 12: Diagrama de secuencia del caso de uso *Obtener dependencias*

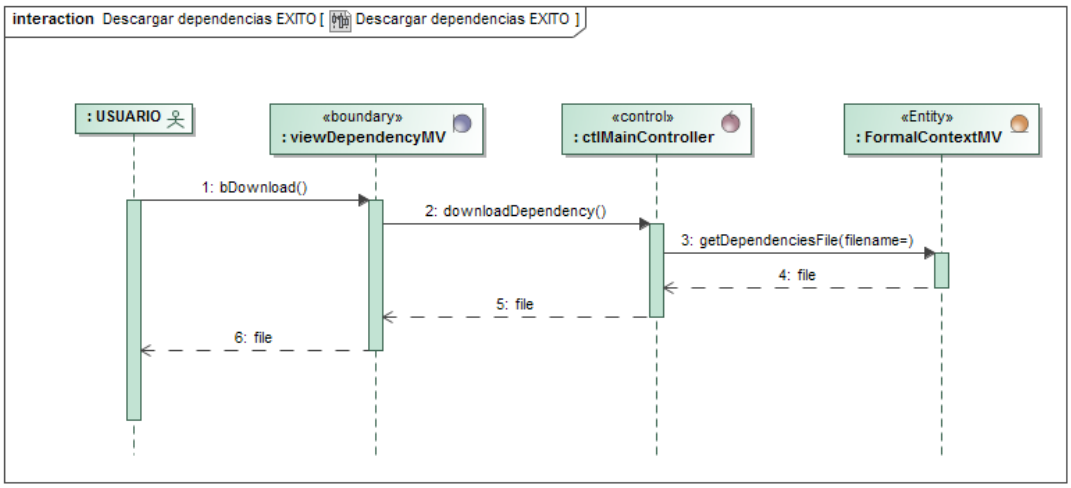

Figura 13: Diagrama de secuencia del caso de uso *Descargar dependencias*

3. El conjunto de clases del algoritmo FDMine fue integrado con la API de FCAUtilities existente, y se realizaron varias pruebas con comentarios reales obtenidos previamente.

4. Se creó la nueva API de FCAUtilities, contando con toda su funcionalidad anterior y añadiendo los puntos anteriores.

5. El desarrollo cambia del back-end al front-end, y se crea una versión beta de la web, sin centrarse en la visualización. En cambio, implementaba las funcionalidades básicas.

6. Se integró la parte web creada en el punto anterior con lo desarrollado previamente en el TFG de mi compañero.

7. Se cambió la plantilla o *template* de la aplicación web.

8. Se realizó una nueva versión de la web, en la que se definió visualmente el flujo de la aplicación, para que el usuario pudiera ver los pasos que tiene que hacer.

9. Se corrigieron pequeños fallos y se añadió el multi-idioma en la última parte desarrollada.

7.5. Pruebas

Se realizaró una batería de cinco tipos de pruebas para API y una para la interfaz web. Las pruebas del código de back-end consistieron en:

Pruebas de FDMine Se introdujeron manualmente en una clase de test los valores de un *contexto formal multivaluado*, constatando que las dependencias obtenidas eran

las esperadas. Esto se probó en unas ocho tablas con diferentes números de filas, columnas y valores.

Pruebas de creación de tabla *FCMV* Se comprobó con un conjunto de datos controlados (entre 10 y 50 comentarios) y una ontología adecuada a ese conjunto, que se obtenía una tabla *FCMV* correcta.

Pruebas de funcionamiento real Fueron repetidos los anteriores test con comentarios reales como inputs. De esa manera se aseguró el correcto funcionamiento de la API.

La interfaz web fue testeada una vez que la API pasó toda su batería de pruebas. Para estas pruebas se emplearon directamente datos reales y se comprobó tanto su correcto funcionamiento como algunas situaciones anómalas, tales como:

- No introducir una ontología.
- No introducir comentarios.
- No introducir una tabla *FCMV*.
- No introducir los ficheros con la extensión correcta.

7.6. Consideraciones

A continuación enumeramos algunas restricciones para el correcto funcionamiento de la aplicación:

- Los comentarios deben de estar en ficheros .txt.
- La primera línea de cada comentario debe de ser el nombre del usuario.
- Los comentarios deben de estar en inglés.
- Es recomendable que cada comentario aparezca en un fichero diferente.
- La ontología debe introducirse con formato JSON.
- El *contexto formal multivaluado* debe estar en formato csv.
- Emplear el sistema operativo Windows para ejecutar la aplicación.
- No se ha aplicado responsive design debido a que las funcionalidades del sistema no están pensadas para el uso en smartphones.
- Se debe seguir el flujo marcado en el manual (Sección 10.4), introduciendo los ficheros necesarios en cada parte. Sino, no se asegura su correcto funcionamiento.

8. Experimentos

En esta sección se explica en profundidad todos los experimentos realizados y las conclusiones obtenidas.

8.1. Introducción

El objetivo final de este TFG es aumentar la eficacia de un motor de recomendación. Para ello, se quiere reducir el número de parámetros que necesita conocer para poder llegar a dar una recomendación. De este modo, se pretende obtener un conjunto de dependencias con las que, conociendo ciertos valores, podamos conocer otros. Así, reduciríamos el número de datos que se debe conseguir para proveer de una recomendación. Para conseguir todo esto es necesario haber obtenido las dependencias entre los diferentes parámetros o atributos, y antes de eso, previamente ha de crearse un *contexto formal multivaluado*. Tener un buen conjunto de datos o *data set* es fundamental para crearlo. Si la base de donde queremos obtener conocimiento no es la adecuada, la información que adquiramos no será útil. Por ello, es necesario analizar en profundidad el conjunto de datos de donde se parte. En el contexto de este TFG, obtener un buen conjunto de datos nos permitirá obtener un buen *FCMV*. Para crear un *FCMV* son necesarios varios elementos:

- Fuente de información: De donde se obtiene la información. Puede ser de comentarios, tweets, valoraciones, etc. Cualquier objeto que nos proporcione la opinión de los usuarios. Principalmente, representada en lenguaje natural.

- Análisis: Es necesario analizar la fuente de información para convertir los datos en conocimiento. Para ello se pueden usar diferentes técnicas, como la obtención de *keyword* o palabras clave, análisis de sentimientos, sinónimos, etc. En este TFG se ha usado una herramienta llamada AlchemyAPI con la que se consiguen las palabras clave.

- Ontología: Para crear el *contexto formal multivaluado* es necesario contar con una ontología. Gracias a ella, se organizan los diferentes atributos con sus respectivos valores. La ontología deberá reflejar las características del objeto de estudio, por ello, es preciso que un experto en la materia cree la ontología, ya que se necesitan conocimientos específicos de ese área.

El análisis de las fuentes de información es un problema sumamente complejo. Esto es porque la información recogida, al ser en lenguaje natural, es muy difícil de estudiar.

En la sección 8.2 se muestra en detalle los elementos usados y el proceso seguido para realizar los experimentos.

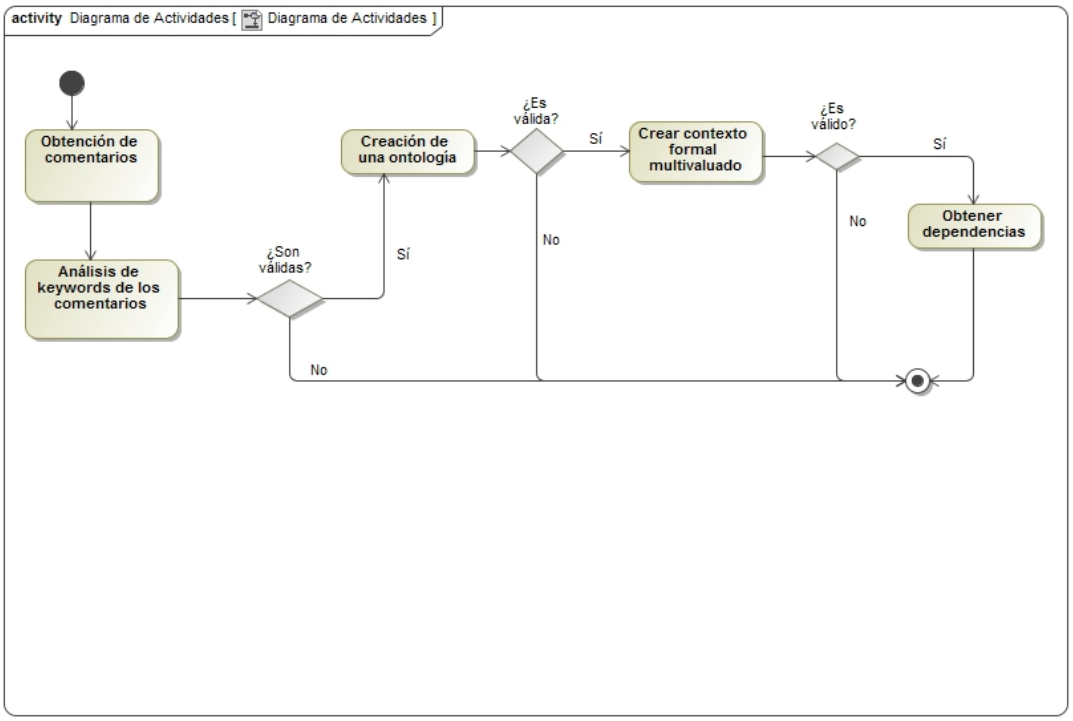

Figura 14: Diagrama de flujo de procesos seguidos

8.2. Proceso de análisis

Para realizar todos los experimentos se han usado principalmente hoteles en la provincia de Málaga como objeto de estudio.

Para poder crear un *contexto formal multivaluado* y obtener sus dependencias, se han seguido los siguientes pasos, descritos en la figura 14.

8.2.1. Obtención de comentarios

Se han usado los comentarios de TripAdvisor [19] como fuente de información. Se ha elegido TripAdvisor por la multitud de comentarios y la gran cantidad de hoteles que tiene. Se ha usado la versión inglesa ya que la herramienta realiza el análisis de texto en inglés.

8.2.2. Análisis

Se ha usado la herramienta AlchemyAPI para realizar el analisis de los comentarios. Este analisis nos proporcionará las *keyword* junto con su frecuencia en todos los comentarios. El conjunto de *keyword* ha de ser válido, o lo que es lo mismo, debe de tener un cierto número de *keyword* con una frecuencia alta. Si no se cumple, cuando creemos el *contexto formal multivaluado*, la mayoría de los valores serán NA o *No Aparece*. Aún así, es frecuente encontrar una gran cantidad de valores NA. Por ello, se realizó un tratamiento no automático de sinónimos y así evitar que el *contexto formal multivaluado* estuviera

repleto de NA.

8.2.3. Ontología

La ontología juega un papel muy importante. En ella se recogen los diferentes características o atributos que se analizaran junto con sus respectivos valores. Las ontologías creadas son muy dependientes del contexto. Si las *keyword* no aparecen en la ontología aparecerá un NA en el *contexto formal multivaluado* que se cree. Por ello, es de vital importancia que se usen los mismos términos para referirse a los mismos objetos, o lo que es lo mismo, que en el análisis se obtengan un grupo de palabras con una frecuencia elevada. Aún así, no siempre que se obtiene una ontología válida. Aunque se obtengan un conjunto válido de *keyword*, puede que no tena sentido en una ontología. También es posible que la ontología formada tenga pocos atributos con infinidad de valores, o al contrario, que posea una enorme cantidad de atributos con valores binarios.

8.2.4. Contexto formal multivaluado

Una vez realizado los tres pasos anteriores, podemos crear un *contexto formal multivaluado*. Éste se forma recorriendo todos los atributos y comprobando si alguna *keyword* tiene el valor de algún atributo. El contexto creado se representa en modo de tabla, donde las columnas son los atributos y las filas representan las *keyword* obtenidas en cada comentario, y siendo el valor de la celda el valor del atributo que coincide con una *keyword* nombrada en ese comentario.

Dicho contexto puede no ser útil o valido ,si queremos obtener conocimiento de él, por ejemplo, si tiene valores muy dispersos, si tiene una mayoría de valores NA, etc. Esto es evitable, en la mayoría de los casos, si en los anteriores pasos se obtienen resultados válidos.

8.2.5. Dependencias

Una vez creada la tabla que representa el *contexto formal multivaluado*, pasamos a obtener las dependencias de él. Para ello se usa el algoritmo *FDMine*.

El analisis de las dependencias se desarrollará en la sección 8.3.

8.3. Estudio de los experimentos

En esta sección se describe, en orden cronológico, los diferentes experimentos que se realizaron y las conclusiones obtenidas en cada uno.

8.3.1. Análisis del hotel La Villa Marbella

Fue el primer hotel que se analizó. Se eligió este hotel por tener un gran número de comentarios. Inicialmente, se obtuvieron 15 comentarios como fuente de información. Después del análisis, se estudiaron el conjunto de *keyword* obtenidas. La frecuencias de las *keyword* eran muy bajas. Además, el conjunto de *keyword* con las frecuencias más alta era muy reducido. Por ello, fue imposible crear una ontología. Para resolver este problema, se realizaron varios análisis, aumentando el número de comentarios. Las pruebas fueron de 30, 50, 100, 150 y 300 comentarios. Cada vez que aumentaba el número de comentarios, aumentaba las *keyword* obtenidas y su frecuencia, como cabría esperar. Aún así, de estas pruebas se dedujo que las *keyword* que tenían más frecuencia eran, a partir de cierto punto, muy similares. Después de varias pruebas se dedujo que, usando 100 comentarios aproximadamente, es posible conseguir un conjunto de *keyword* válido. Una vez obtenida el conjunto de *keyword*, se pasó a crear la ontología. En este caso, no se consiguió crear una ontología válida. Los atributos que se obtenían era escasos y tenían pocos valores. Esto ocurría por que el conjunto de *keyword* obtenido era muy amplío y difícilmente agrupable bajo el mismo atributo. Por otro lado, en los hoteles, el número de características a comentar es muy elevado. Por ello, se decidió cambiar el objeto de estudio a los restaurantes, que tienen muchos menos atributos.

8.3.2. Análisis de restaurantes

Este estudio se compone de múltiples análisis realizados a diferentes restaurantes, tales como Restaurante Cervantes, Restaurante Farolillos o Restaurante Paella.

Se realizaron varios análisis y obtuvieron resultados satisfactorios: el numero de palabras clave era más reducido, y por tanto, la frecuencia de las *keyword*, en general, era más elevado. Así, cuando se creara la tabla del *contexto formal multivaluado*, no encontraríamos una gran cantidad de NA. De todos modos, la creación de una ontología válida resultó inviable debido a que las *keywords* que se obtuvieron no eran adecuadas para formar una ontología. Esto ocurre porque la mayoría de los comentarios se extraían las mismas palabras o características, y éstas no aportaban información útil para el estudio. Todo ello tenía como raíz del problema el número reducido de palabras de los comentarios.

A diferencia de los comentarios en hoteles, en los restaurantes se comentan muchas menos características y todo ello repercute en la disminución del número de palabras por comentario, y por tanto de *keyword*. Si hay menos palabras, habrá menos *keywords*. Así que una vez comprobado que han de aumentar significativamente el número de palabras por comentarios, sin aumentar en gran medida las características comentadas, se pensó en estudiar los diferentes hoteles en los que un usuario se ha hospedado. Esto permitiría disminuir el número de características comentadas, tener más *keyword* comunes (ya que el vocabulario es el mismo) y con una cantidad de palabras adecuadas.

8.3.3. Análisis de un usuario

Para analizar los comentarios de hoteles de un usuario, fue necesario realizar un estudio previo, hasta encontrar con un usuario que hubiera contribuido en un gran número de hoteles similares. Después de un extenso estudio, se encontró a varios usuarios con estas características. Aún así, no tenían el número suficiente comentarios para crear un *contexto formal multivaluado*. Por ello, el objeto de estudio volvió a ser los hoteles.

8.3.4. Análisis del hotel Marriott's Marbella

Después de un análisis en profundidad de los datos, el conjunto de *keywords* obtenidas eran válidas, así que se realizó una ontología sobre este caso y, posteriormente, un *contexto formal multivaluado*. Dicho contexto presenta pocos NA y una variedad aceptable de valores. Por ello, se realizó un analisis de dependencias funcionales, que se muestran a continuación:

- time transport room → services
- time environment room → services
- environment company room → services
- time environment extras → company,room
- time extras → services
- environment room extras → time,services,company
- environment transport extras → company

De ese conjunto de dependencias podemos decir, principalmente, que el número de dependencias obtenidas es escaso, siendo tres el número de atributos en la parte izquierda predominante. De estas dependencias, en su mayoría solo se obtiene un atributo, por lo que el uso que pueda dar a un motor de recomendación será muy limitado. Cabe destacar que el atributo *services* es el atributo referenciado en la mayoría de las dependencias, en gran parte por su alto número de valores semejantes. Como dato de interés, desde *environment, room, extras* pueden obtenerse la mayoría de los atributos. Esto puede ser usado en un motor de recomendación para reducir el número de atributos que necesitamos obtener para una recomendación. Siguiendo esta ontología, serían necesario conocer el valor de los 7 atributos , una por cada uno, para dar una recomendación, pero aplicando esta dependencia, con 4 sería suficiente ya que con esos tres podemos obtener tres más. Esto es muy útil ya que se evita trabajo al usuario. En muchos casos, el usuario no querrá rellenar grandes formularios ni compartir cierta información. Así que, de este modo, podemos proveer al usuario de una recomendación con menos esfuerzo. Una vez analizado un hotel en la costa, se cambió el tipo de hotel que se estudiaba y se buscó un hotel de interior.

8.3.5. Análisis del hotel La Fuente De La Higuera

Este hotel contaba con un número de comentarios más reducido, así que fue posible estudiar la mayoría de los comentarios. Inicialmente, las *keyword* analizadas diferían en mucho a las anteriormente vistas al ser los anteriores hoteles de la costa y este de interior. En cambio, la ontología creada era muy similar. Esto ocurre porque la ontología creada previamente era, en cierto modo, muy general. Por ello, podemos agrupar valores muy diferentes bajo el mismo atributo. Una vez formada la ontología, se creó el *contexto formal multivaluado*, y posteriormente, se analizaron sus dependencias, mostradas a continuación.

- time characteristics extras → transport
- services environment room → extras
- time food company → environment,extras
- food characteristics room → extras
- food characteristics extras → transport
- characteristics room → transport
- services time food → transport
- food extras room → transport
- food company room → environment,extras,characteristics,transport
- time extras room → services
- company characteristics extras → transport
- environment company characteristics → transport
- environment company room → extras
- time company room → services
- services company room → time
- time food room → environment,extras,characteristics,transport,services,company
- company characteristics room → environment,extras,food
- environment food room → extras,characteristics,transport,company
- company room extras → environment

En estas dependencias, se aprecia una notable diferencia con la anterior: el número de dependencias es mucho más alto (el doble). En cambio, la mayoría de las dependencias son de tres atributos. La información más relevante que se obtiene es del conjunto de atributos *{food, room, time}*. De ellos, al ser clave, podemos obtener todos los demás valores. Esto le proporciona una capacidad muy potente a un motor de recomendación, dado que con solo esos tres atributos, se podría saber qué tipo de hotel se está buscando. De este modo, en vez de preguntar a un usuario por nueve valores, con tres sería más que suficiente. Así se simplifica el proceso de recogida de información para generar una recomendación. Ese proceso es uno de los más costosos, ya que suele resultar muy tedioso para el usuario. Además, es posible que algún atributo pueda ser obtenido por otros medios, sin tener que preguntarle directamente al usuario. Por ejemplo, podemos averiguar mucha información con su historial de recomendación. También es posible conocer el valor de algun atributo por otro canal, consultando información contextual como por ejemplo, el tiempo meteorológico. De este modo, podemos, ampliar la eficacia del motor de recomendación.

Por último, se estudió un hotel boutique situado en Marbella.

8.3.6. Análisis del hotel Claude

De este hotel se pudo crear un *contexto formal multivaluado* sin ningun problema. Seguidamente se obtuvieron las siguientes dependencias:

- transport drinks room → extras,time
- time characteristics extras → environment,transport,company,room,drinks
- characteristics room extras → company,drinks
- transport characteristics drinks → company
- company characteristics extras → drinks
- environment transport company → time,drinks
- time transport characteristics → environment,extras,company,room,drinks
- characteristics drinks extras → company
- time environment drinks → transport,company
- time environment room → extras,characteristics,transport,company,drinks
- characteristics drinks room → company
- environment characteristics room → extras,time,transport,company,drinks

- transport room extras → time,drinks
- time environment extras → transport
- transport characteristics extras → company,drinks
- time environment company → transport,drinks
- transport company extras → characteristics,drinks
- environment transport characteristics → extras,time,company,room,drinks
- environment characteristics extras → company,drinks
- environment room extras → characteristics,time,transport,company,drinks
- environment transport extras → time
- transport characteristics room → company
- company room extras → drinks
- time characteristics drinks → company,room

Al igual que en el caso anterior, el número de dependencias ha aumentado notablemente (casi el doble). También se denota en gran medida que el número de atributos dominante en la parte izquierda son tres, como en los casos anteriores.

De estas dependencias podemos obtener un dato claro: existen muchas claves. De cinco conjuntos de tres atributos diferentes, podemos obtener los posibles valores de los demás. Además, analizándolo con más detalle, la parte izquierda de esos cinco conjuntos está formada por atributos similares. Así, éstos forman un conjunto de valores.**{time, characteristics, extras, transport, environment, room}**. Con casi cualquiera tres valores de ese conjunto, podemos conseguir todos los demás atributos de la ontología. Esto provee a un motor de recomendación con una potencia superior a lo citado en los experimentos anteriores. De este modo, un usuario sólo tendría que definir las 3 opciones que prefiriera para conseguir la recomendación.

8.4. Conclusión

Tras este conjunto de análisis de casos de prueba, queda constatado que la base de donde obtengamos conocimiento ha de cumplir ciertas características para que sea válida y podamos obtener información de ella:

- Número de comentarios: 100 aproximadamente.
- Media de numero de palabras: 200 aproximadamente.

- Similitud de terminología usada.
- Numero de características limitado.

De este modo, se ha constatado que, obteniendo un buen *contexto formal multivaluado*, es posible obtener un conjunto de dependencias que ayude en gran medida a un motor de recomendación, evitando tediosas tareas al usuario.

Ademas de estas conclusiones se añaden en el anexo 10.2 las ontologías usadas, junto con sus contextos en el anexo 10.3.

9. Conclusiones y trabajos futuros

En este TFG se ha estudiado como aumentar la eficacia de un motor de recomendación. Para ello ha sido necesario realizar un estudio sobre la teoría de *análisis formal de conceptos multivaluados*. En lo que respecta al estudio de dicha teoría, fue una grata sorpresa al descubrir su simplicidad. El mayor problema de ella consistió en la creación de un *contexto formal multivaluado*. Aunque en un principio se predijo que esto podría ser un problema, no se pensó que su complejidad sería tan alta. Realmente era laborioso obtener un conjunto de comentarios y una ontología con que la poder trabajar, como se demostro en el apartado de experimentos 8. Todo ello debido a que cada usuario usa su propio vocabulario y es difícil obtener términos comunes.

Otra parte fundamental son las dependencias. Inicialmente, se pensó en usar dependencias binarias (de un atributo a otro atributo) para usarlas en el motor de recomendación. En la práctica, no se obtuvo ese tipo de dependencias, sin embargo, sí que se obtuvieron datos interesantes: el conjunto de claves. Con los atributos clave, que son un pequeño conjunto de todos los atributos, es posible obtener el valor del resto.

De todo el estudio que se ha realizado, se han descubierto posibles líneas futuras de TFGs relacionadas:

- Emplear alguna herramienta que automatice la obtención de comentarios.
- Aplicar técnicas de sinónimos.
- Aplicar analisis de sentimientos.
- Ampliar este desarollo con técnicas de big data.
- Emplear diferentes algoritmos para la obtención de dependencias.

10. Anexo

10.1. Temporización del trabajo

A continuación se expone un desglose del tiempo empleado en cada tarea del TFG:

Tarea	**Duración (en horas)**
Planificación y seguimiento	10
Diseño	5
Estudio y Aprendizaje	42
Desarrollo Beta de la aplicación	30
Desarrollo Final del sistema	57
Experimentos	100
Problemas y mejoras	35
Testing	12
Desarrollo de la memoria	75
Total	366

10.2. Ontología

10.2.1. Ontología Marriot

```
{
"time" : ["evening","morning","night"],
"company" : ["family","friends","couple","marriage"],
"environment" : ["location","people","atmosphere","relaxing"]
   ,
"extras" : ["restaurant","bars","gym","shops","beach","pool",
   "spa"],
"transport" : ["parking", "bus", "walk", "car", "taxi"],
"room" : ["bathroom", "balcony", "tv", "bed"],
"services" : ["service"]
}
```

10.2.2. Ontología Higera

```
{
"time" : ["evening","morning","night"],
"company" : ["family","friends","couple","marriage"],
"characteristics" : ["price","luxury","comfort"],
```

```
"environment" : ["location","people","atmosphere","peace"],
"extras" : ["restaurant","bars","gym","shops","beach","pool",
   "spa"],
"food" : ["breakfast","dinner", "lunch", "brunch","dinners"],
"transport" : ["bus", "walk", "car", "taxi"],
"room" : ["bathroom","balcony","tv", "bed","view","air
   conditioning"],
"services" : ["service"]
}
```

10.2.3. Ontología Claude

```
{
"time" : ["night","evening","morning"],
"company" : ["family","friends","couple","marriage"],
"characteristics" : ["price","luxury","comfort"],
"environment" : ["location","people","atmosphere","relaxing"]
   ,
"extras" : ["restaurants","bars","gym","shops","beach","pool"
   ,"spa"],
"food" : ["breakfast", "dinner", "lunch", "brunch"],
"transport" : ["bus", "walk", "car", "taxi"],
"room" : ["bathroom", "balcony", "tv","bed"],
"drinks" : ["coffee", "cava", "wine", "juice"],
"services" : ["service"]
}
```

10.3. Contextos Formales Multivaluados

10.3.1. Contexto Formal Multivaluado Marriot

services	time	environment	transport	company	room	extras
service	NA	location	NA	family	NA	restaurant
NA	NA	relaxing	NA	couple	balcony	pool
NA	NA	NA	NA	family	balcony	NA
service	morning	NA	car	family	NA	restaurant
NA	NA	NA	NA	family	balcony	pool
service	NA	people	taxi	family	bathroom	beach
service	NA	people	NA	family	bathroom	restaurant
service	NA	NA	NA	family	bed	restaurant
NA	night	NA	taxi	friends	bathroom	pool
service	NA	location	car	family	NA	restaurant
NA	evening	people	NA	NA	balcony	NA
service	evening	location	car	family	bed	shop
NA	NA	NA	car	family	bathroom	shop
service	NA	location	NA	couple	bed	beach
NA	night	NA	NA	family	bathroom	restaurant
NA	night	location	car	couple	NA	beach

10.3.2. Contexto Formal Multivaluado Higera

services	time	environment	transport	food	company	characteristics	room	extras
service	NA	NA	NA	dinner	couple	NA	NA	pool
NA	evening	location	NA	dinner	NA	NA	balcony	NA
NA	night	atmosphere	NA	dinner	NA	comfort	view	restaurant
NA	NA	people	NA	NA	couple	NA	balcony	pool
service	NA	location	NA	NA	NA	NA	view	pool
NA	NA	people	car	dinner	NA	NA	bed	restaurant
service	evening	location	car	dinner	NA	comfort	NA	NA
NA	night	location	car	lunch	couple	NA	bed	NA
service	NA	location	NA	dinner	family	NA	view	pool
NA	night	location	NA	NA	NA	NA	view	pool
service	evening	location	NA	NA	NA	price	NA	NA
service	evening	location	NA	lunch	NA	NA	bathroom	pool
service	evening	location	NA	NA	marriage	NA	balcony	pool
service	night	peace	car	dinner	friends	luxury	NA	NA
service	night	location	car	breakfast	family	luxury	NA	NA

10.3.3. Contexto Formal Multivaluado Claude

time	environment	transport	company	characteristics	drinks	room	extras
evening	relaxing	car	NA	NA	NA	NA	beach
night	NA	car	NA	NA	coffee	bathroom	NA
evening	NA	walk	couple	price	wine	NA	restaurants
night	location	walk	marriage	price	NA	bathroom	NA
NA	location	NA	NA	NA	NA	bathroom	bars
NA	people	car	NA	NA	NA	bathroom	gym
NA	relaxing	NA	friends	luxury	NA	balcony	NA
morning	location	taxi	friends	NA	juice	bed	restaurants
evening	people	NA	couple	NA	wine	NA	restaurants
NA	NA	walk	NA	NA	coffee	NA	NA
evening	location	walk	couple	comfort	NA	NA	spa
evening	location	walk	NA	luxury	wine	bathroom	spa
NA	location	NA	NA	comfort	NA	balcony	NA
morning	NA	NA	couple	price	coffee	bed	NA
morning	location	car	NA	NA	coffee	NA	NA

10.4. Manual de usuario

En esta sección se explican como usar la aplicación web. Se desarrollará paso por paso la secuencia que se debe seguir.

10.4.1. Página principal

Para acceder a la pagina principal, es necesario que el archivo *run* haya sido ejecutado. Para acceder a la pagina principal debemos poner en nuestro navegador e ir a la dirección http://127.0.0.1:8080/Application/. Una vez accedamos a la pagina principal (ver figura 15, debemos pulsar en *Nuevo Contexto Formal Multivaluado* para empezar el proceso.

10.4.2. Subir ontoloía o FCMV

En esta página (ver figura 16)se decide que camino usar. Si subimos una ontología, pasaremos al paso 2 (Sección 10.4.3). En cambio si decidimos subir un *contexto formal multivaluado* pasamos directamente al paso 3 (Sección 10.4.4). Una vez que subamos alguno de los, pulsamos el botón de *siguiente* para ir al siguiente paso.

10.4.3. Comentarios

En este paso se suben los comentarios que se analizarán (ver figura 17). Para ello deben de añadirse pulsando el botón azul *subir*, o haciendo *Drag and Drop*. Una vez que se carguen todos los comentarios que queramos analizar, se debe de pulsar el botón superior de *subir* para que los archivos se suban. Finalmente se pulsa el botón *siguiente* para ir al siguiente paso.

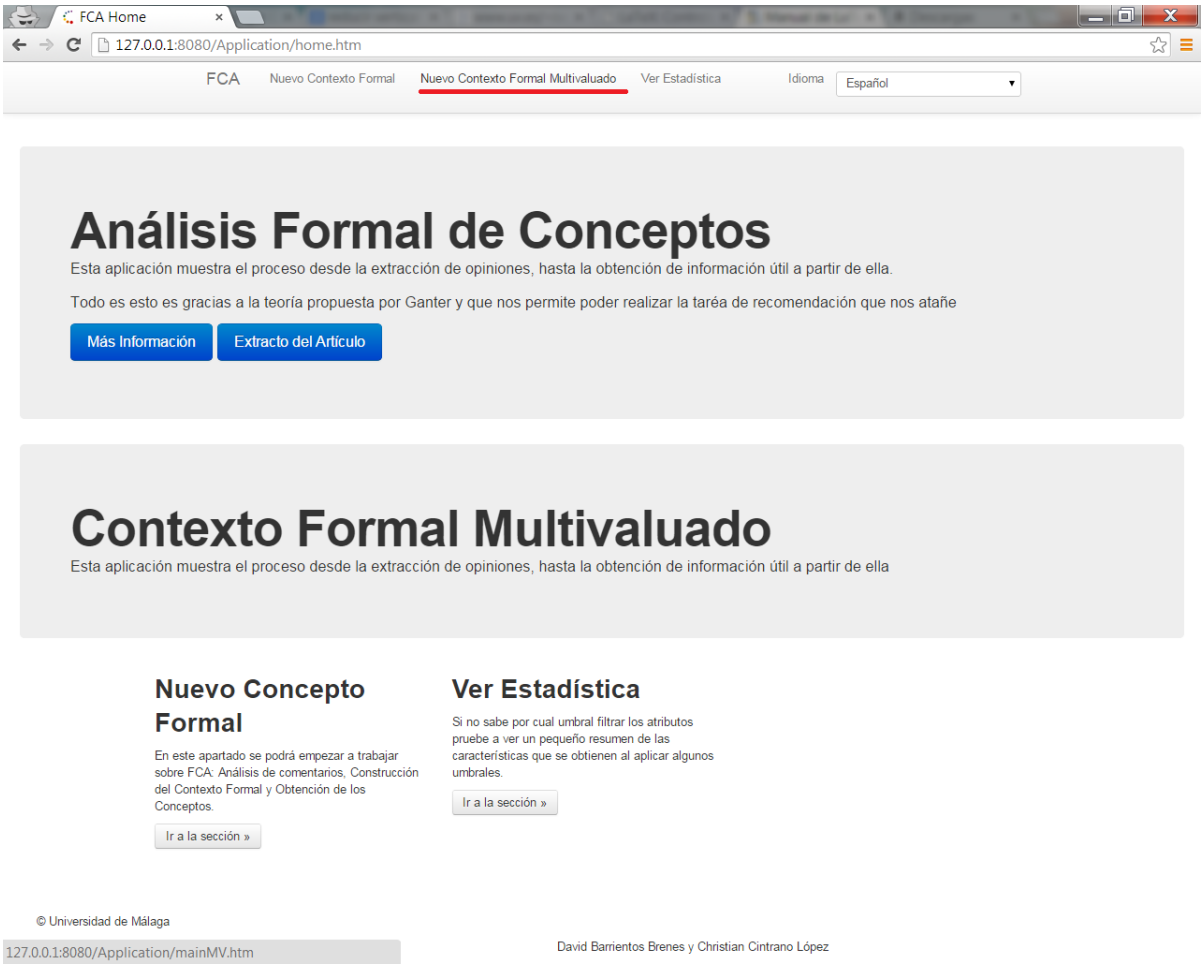

Figura 15: Home Page

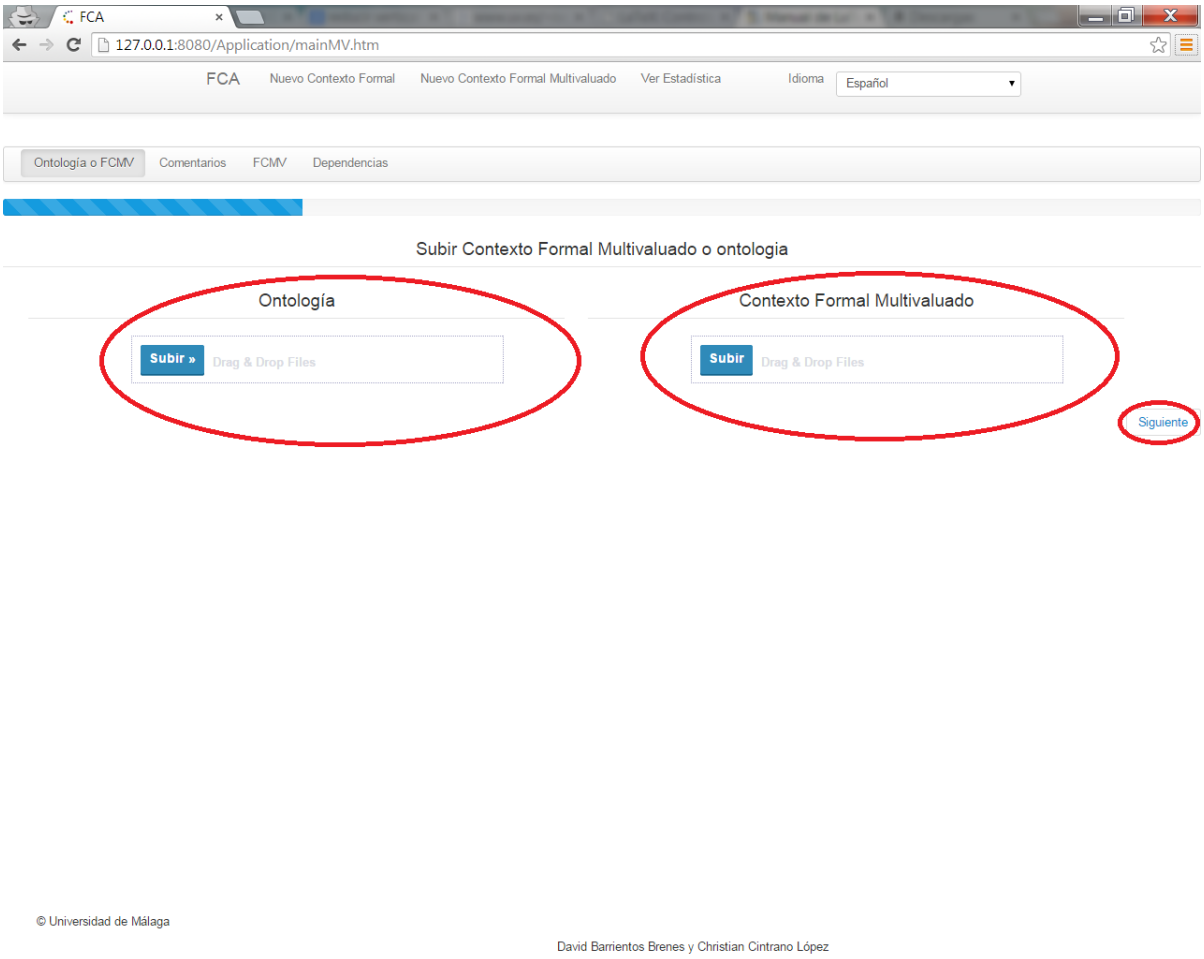

Figura 16: Paso uno: Subir ontología o FCMV

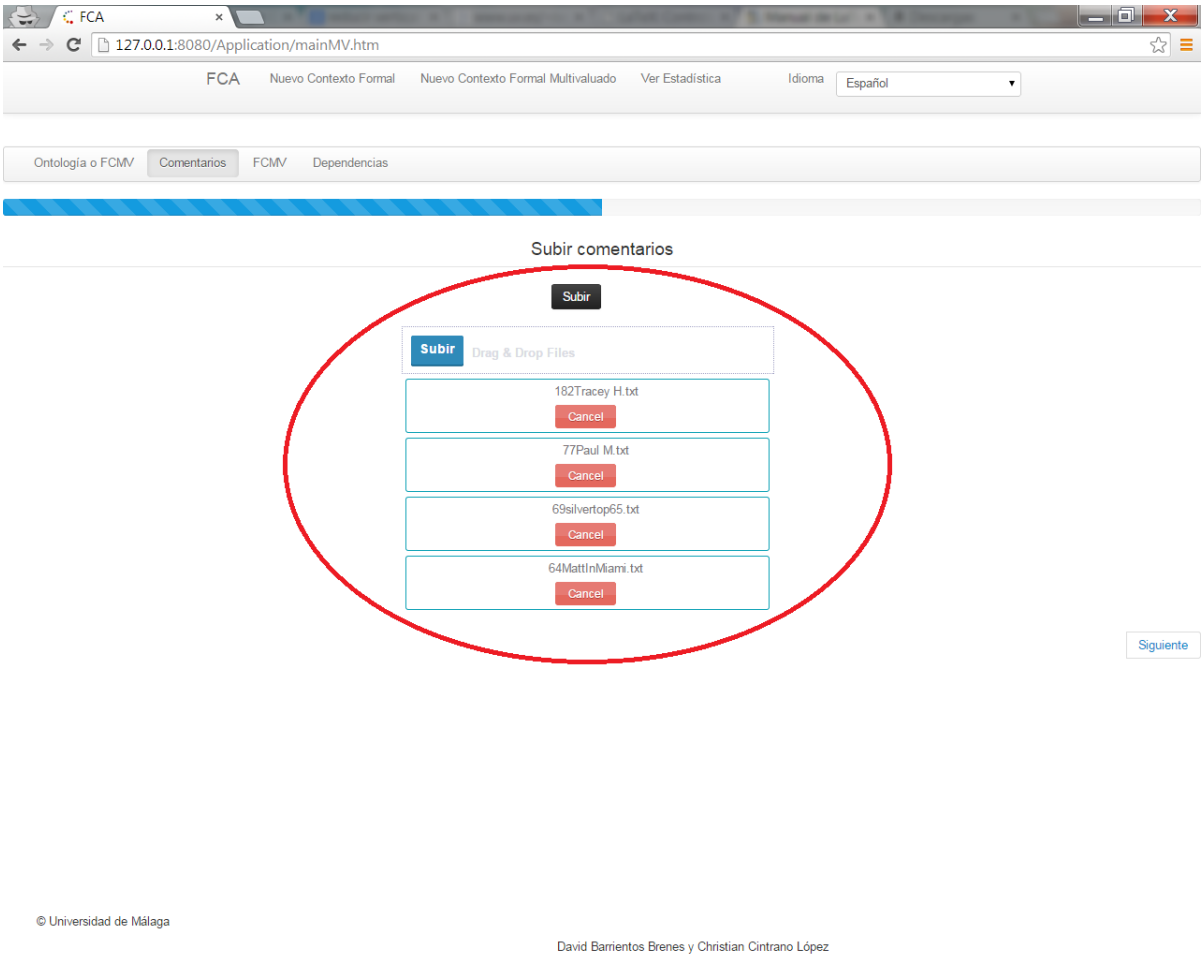

Figura 17: Paso 2: Subir comentarios

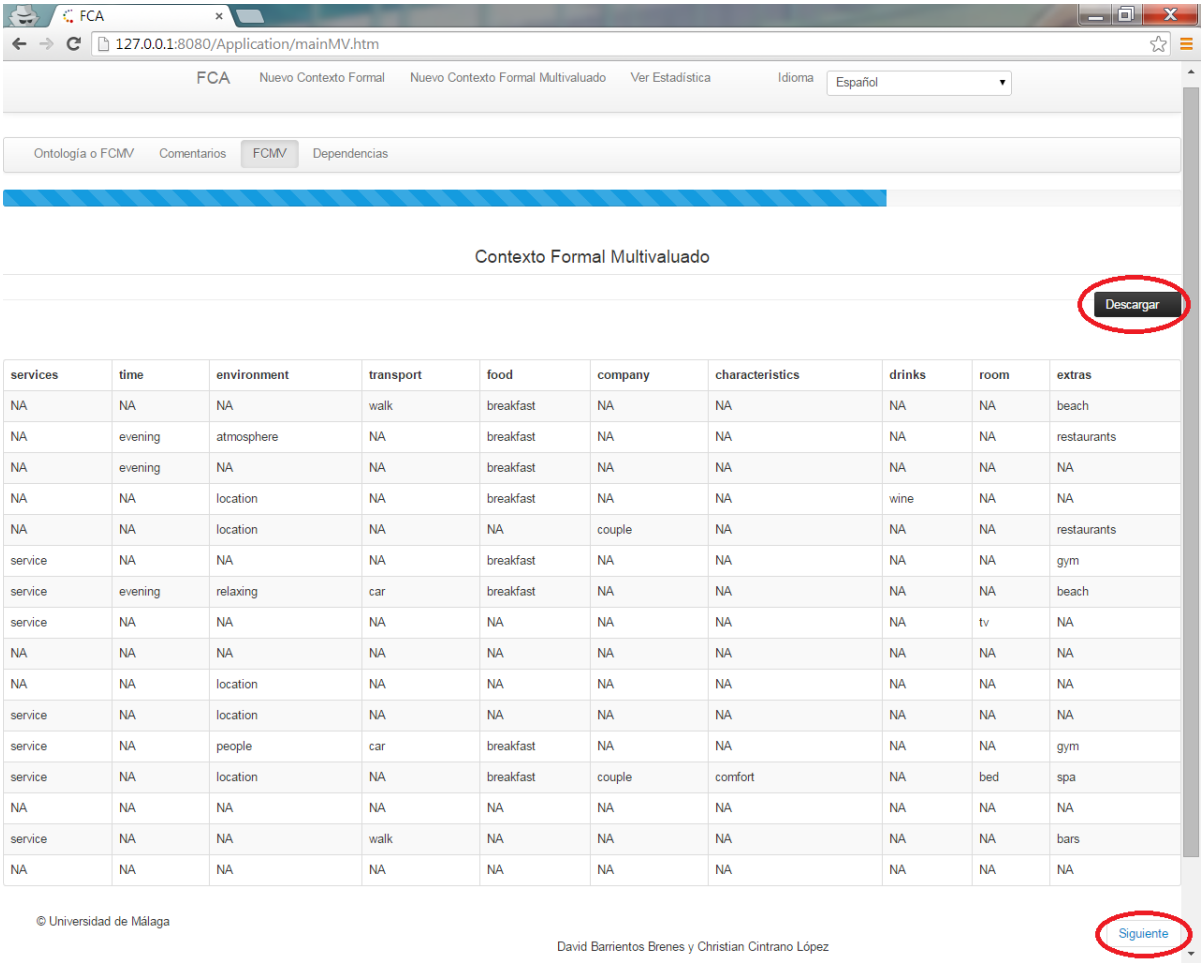

Figura 18: Paso 3: Obtener contexto formal multivaluado

10.4.4. Contexto formal multivaluado

Se muestra el *contexto formal multivaluado* creado gracias a los archivos anteriores (ver figura 18). Es posible descargarlo pulsando el botón de *descargar*. Si pulsamos siguiente pasaremos al último paso.

10.4.5. Dependencias

Se mostrará la lista de dependencias de la tabla creada en el punto 10.4.4 (ver figura 19). Es posible descargarlo pulsando el botón de *descargar*.

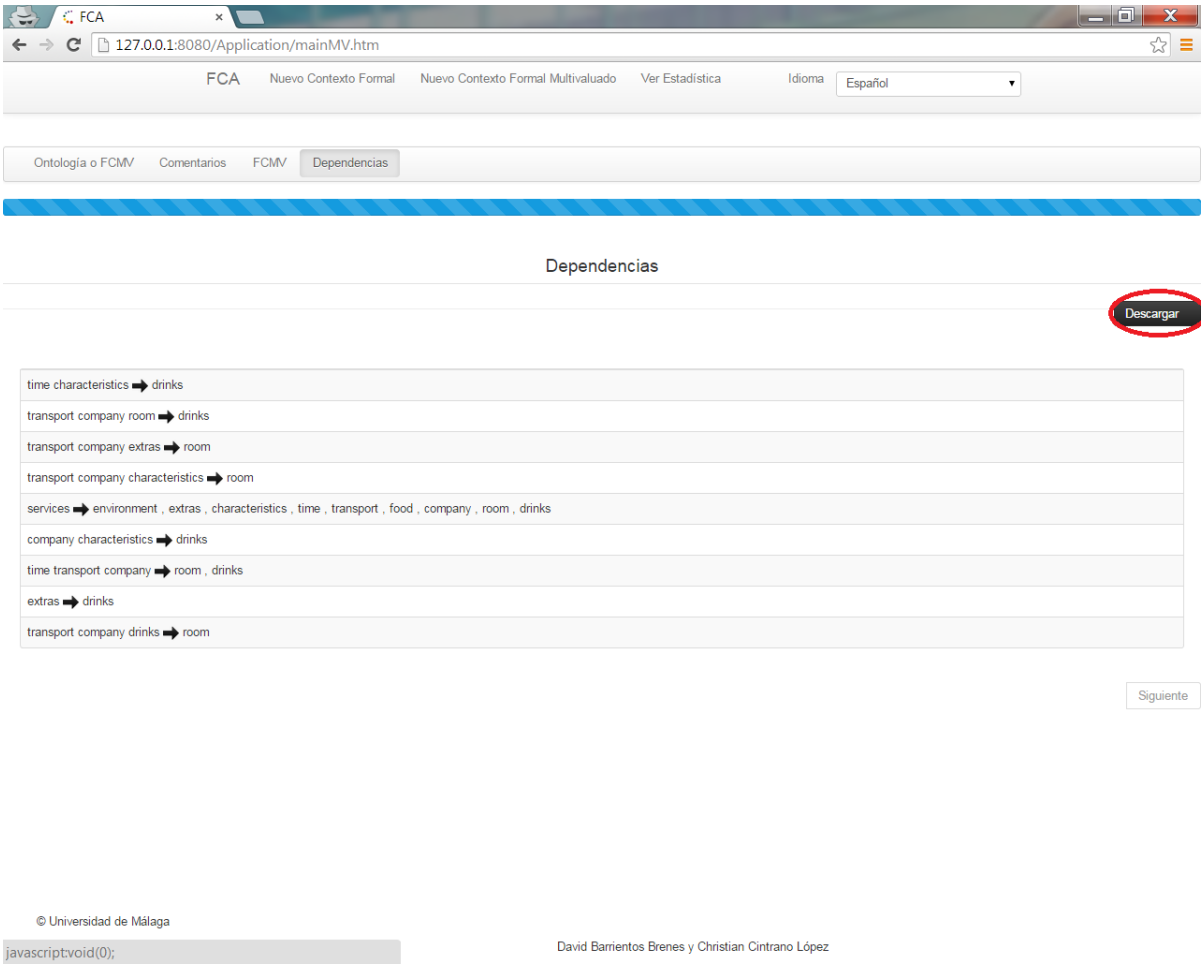

Figura 19: Paso 4: Obtener dependencias

Bibliografía

[1] AJAX. http://es.wikipedia.org/wiki/AJAX.

[2] AlchemyAPI. http://www.alchemyapi.com/.

[3] Apache Ant. http://ant.apache.org/.

[4] Bootstrap. getbootstrap.com.

[5] Eclipse. https://www.eclipse.org/.

[6] Gate. https://gate.ac.uk/.

[7] JBoss. http://www.jboss.org/.

[8] jQuery. http://jquery.com/.

[9] jQuery BlockUI. http://malsup.com/jquery/block/.

[10] jQuery File Download. http://jqueryfiledownload.apphb.com/.

[11] jQuery jGrowl. http://plugins.jquery.com/jgrowl/.

[12] jQuery Upload File. http://plugins.jquery.com/uploadfile/.

[13] Json. http://json.org/.

[14] JSON simple. http://www.json.org/.

[15] Maven. maven.apache.org/.

[16] R. http://www.r-project.org/.

[17] Scrum. https://www.scrum.org/.

[18] Spring Framework. http://projects.spring.io/spring-framework/.

[19] Tripadvisor. http://www.tripadvisor.com/.

[20] Web del grupo SICUMA. http://www.sicuma.uma.es/es/.

[21] Wizard. http://vadimg.com/twitter-bootstrap-wizard-example/.

[22] *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society, 2002.

[23] Radim Belohlávek and Vilém Vychodil. *Fuzzy Equational Logic*, volume 186 of *Studies in Fuzziness and Soft Computing*. Springer, 2005.

[24] Bernhard Ganter and Gerd Stumme. Methods and applications in computer science. *Formal Concept Analysis*, 46:1–117, 2002-2003.

[25] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[26] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[27] José L. Leiva, Antonio Guevara, Carlos Rossi, and Andrés Aguayo. Realidad aumentada y sistemas de recomendación grupales. *Estudios y Perspectivas de Turismo*, 23:40–59, 2014.

[28] Bing Liu. Opinion mining. pages 1–7.

[29] Bing Liu. Sentiment analysis: A multi-faceted problem. *IEEE Intelligent Systems*, 2010.

[30] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 415–463. Springer, 2012.

[31] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[32] Noel Novelli and Rosine Cicchetti. Functional and embedded dependency inference: a data mining point of view. *Inf. Syst.*, 26(7):477–506, 2001.

[33] G. Stumme, R. Taouil, Y. Bastide, N. Pasqier, and L. Lakhal. Computing iceberg concept lattices with Titanic. *J. Data and KnowledgeEngineering (DKE)*, 42(2):189–222, 2002.

[34] Nizar Messai Zainab Assaghir, Mehdi Kaytoue and Amedeo Napoli. On the mining of numerical data with formal concept analysis and similarity.