ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería Informática

Redes de influencias y líderes de opinión en las redes sociales

Influence networks and opinion leaders in social networks

Realizado por
Pablo Márquez Márquez
Tutorizado por
Carlos Rossi Jiménez
Manuel Enciso García-Oliveiros
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Octubre 2015

Fecha defensa:
El Secretario del Tribunal

Resumen: Este Trabajo de Fin de Grado (TFG) se engloba en la línea general Social CRM. Concretamente, está vinculado a un trabajo de investigación llamado "*Knowledge discovery in social networks by using a logic-based treatment of implications"* desarrollado por P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego y C. Rossi en la Universidad de Málaga, en el cual se ofrecen nuevas soluciones para la identificación de influencias de los usuarios en las redes sociales mediante herramientas como el Analisis de Conceptos Formales (FCA).

El TFG tiene como objetivo el desarrollo de una aplicación que permita al usuario crear una configuración minimal de usuarios en Twitter a los que seguir para conocer información sobre un número determinado de temas. Para ello, obtendremos información sobre dichos temas mediante la API REST pública que proporciona Twitter y procesaremos los datos mediante algoritmos basados en el Análisis de Conceptos Formales (FCA). Posteriormente, la interpretación de los resultados de dicho análisis nos proporcionará información útil sobre lo expuesto al principio. Así, el trabajo se ha dividido en tres partes fundamentales:

1. Obtención de información (fuentes)
2. Procesamiento de los datos
3. Análisis de resultados

El sistema se ha implementado como una aplicación web Java EE 7, utilizando JSF para las interfaces. Para el desarrollo web se han utilizado tecnologías y frameworks como Javascript, JQuery, CSS3, Bootstrap, Twitter4J, etc. Además, se ha seguido una metodología incremental para el desarrollo del proyecto y se ha usado UML como herramienta de modelado. Este proyecto se presenta como un trabajo inicial en el que se expondrán, además del sistema implementado, diversos problemas reales y ejemplos que prueben su funcionamiento y muestren la utilidad práctica del mismo.

Palabras clave: Análisis de conceptos formales; análisis de redes sociales; descubrimiento de conocimiento; Contexto Formal

Abstract: This Degree Thesis (TFG) is included in the main line Social CRM. Specifically, it's linked to a research work called "*Knowledge discovery in social networks by using a logic-based treatment of implications"* developed by P.Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego y C. Rossi in the University of Málaga, which offers new solutions to identify user's influence in social networks by using tolos like Formal Concept Analysis (FCA).

The TFG aims to develop an application which permits the user to create a minimal configuration of Twitter users who follow to be well informed about some topics of interest. To do this, we'll get information about that topics from the Twitter's public REST API and process it by aplying some algorythms based on Formal Concept Analysis (FCA). Later, the interpretation of the results of this analysis will provide us useful information about the stuff exposed at the begining of this document. Thus, this work has been divided in three essential parts:

1. Getting information (sources)
2. Data processing
3. Result analysis

The system has been implemented as a Java EE 7 web application, using JSF on the interfaces. For the web development, technologies and frameworks like Javascript, JQuery, CSS3, Bootstrap or Twitter4J have been used. Furthermore, UML is been used as modeling tool. This project is presented as an initial work in which some real problems and examples will be exposed to prove the correct behaviour of the system and show the practical use of it.

Keywords: Formal Concept Analysis; Social Network Analysis; Knowledgement discovery; Formal Context

Índice

1. Introducción

Este TFG se desarrolla dentro de la línea “Social CRM”, cuyo objetivo general es investigar y desarrollar tecnologías que, aplicando técnicas formales, permitan un mayor conocimiento de los usuarios o clientes de una entidad a través del análisis de su comportamiento en redes sociales. Dentro de esta, se busca la creación de un sistema que permita la identificación de distintos tipos de relaciones entre lo que, de ahora en adelante, llamaremos Atributos y Objetos. En concreto, para este trabajo, se busca identificar un conjunto de usuarios (objetos) de Twitter a los que tenemos que seguir si queremos estar informados sobre ciertos temas de actualidad (atributos). Además, este conjunto de usuarios debe ser mínimo (lo más pequeño posible). Un ejemplo práctico de lo anteriormente descrito podría ser el siguiente:

Tenemos un conjunto de usuarios, por ejemplo: {@el_pais, @bcc, @EFE , @cnn} y un conjunto de temas {#ebola, #Siria, #ONU, #DerechosHumanos}. Un ejemplo de la salida de nuestro sistema podría ser: “Si quiere estar informado sobre #ebola y #ONU, debe seguir a @el_pais y @cnn” o “Si quiere estar informado sobre #ebola, #ONU y #DerechosHumanos pero quiere evitar #Siria, deberá seguir a @el_pais, @bbc y @EFE”. El ejemplo anteriormente descrito no es más que una pequeña aplicación práctica del funcionamiento de las técnicas y algoritmos basados en FCA, pero la intención de este trabajo es presentar un punto de partida desde el que puedan abordarse problemas más complejos.

Este trabajo de fin de grado consta de cuatro etapas:

1. **Obtención de información**: nuestro sistema obtiene la información pertinente directamente desde internet. Para ello, en este caso, se ha optado por utilizar la API pública que ofrece Twitter, a la cual accederemos mediante una librería llamada Twitter4J, la cual crea una capa por encima de la de transporte (http) que nos permitirá el acceso a dicha API de una forma mucho más sencilla (la comunicación http desde ficheros Java es bastante tediosa y propensa a errores). Además, al devolvernos directamente objetos de Java, esta librería nos evita el procesamiento inicial de dichos datos y nos facilita el mapeado y la construcción de las estructuras de datos necesarias para aplicar los algoritmos posteriores.
2. **Procesamiento de los datos**: una vez construidas las estructuras de datos necesarias (las cuales explicaremos con más detalle en apartados posteriores), aplicaremos los algoritmos (basados en FCA) especificados en el artículo “*Knowledge discovery in social networks by using a logic-based treatment of implications”* como el de obtención de los Labeled Closed Sets (LCS) a partir del Contexto Formal, datos que deberemos interpretar en la siguiente fase para obtener conocimiento real de la información obtenida de las fuentes. Para la implementación de dichos algoritmos, se ha tenido presente, constantemente, el hecho de que su rendimiento fuera óptimo, utilizando las técnicas y tipos de datos más idóneos para lograr este objetivo.

3. **Interpretación de los datos y presentación**: en esta etapa nuestro sistema interpreta los datos de salida de los algoritmos aplicados en la fase anterior. Esta interpretación consiste en la obtención de los Conceptos: éstos, al estar relacionados entre sí, formarán lo que llamaremos un Concept Lattice, un grafo que reflejará las relaciones entre los atributos del Contexto Formal y a partir del cual se pueden obtener distintas conclusiones. Nuestra aplicación generará una representación gráfica de este grafo y realizará una presentación de este, además de exponer los distintos conjuntos de datos obtenidos en el proceso, de forma que sean comprensibles para el usuario de la aplicación. Que dicha presentación sea correcta, simple y entendible será el objetivo final del sistema.
4. **Análisis de resultados**: en esta etapa se realizarán las pruebas pertinentes para comprobar que el sistema funciona correctamente y satisface los requisitos especificados. Además, sacaremos conclusiones sobre el comportamiento del algoritmo sobre los conjuntos de datos.

La estructura de esta memoria es la siguiente: en la sección 2 se describe la motivación del trabajo; en la sección 3 se explica la metodología usada en el desarrollo del trabajo; en la sección 4 se enumeran y describen los objetivos del trabajo; en la sección 5 se hace un análisis detallado de los requisitos del sistema; en la sección 6 se explican los fundamentos teóricos de los algoritmos implementados; en la sección 7 se describen las tecnologías usadas, así como el entorno de desarrollo; en la sección 8 se presentan los aspectos fundamentales del desarrollo del trabajo tales como el modelado, la estructura del proyecto, etc.; la sección 9 ha sido dedicada a la presentación de las pruebas y sus resultados; en la sección 10 se hace un repaso de las interfaces del sistema; en la sección 11 se exponen algunos ejemplos prácticos del funcionamiento global de la aplicación; en la sección 12 se presentan las conclusiones sacadas de la realización de este TFG así como algunos desarrollos futuros posibles; por último, en la sección 13 se incluye la bibliografía utilizada durante todo el desarrollo.

2. Motivación del TFG y objetivos principales

Desde hace unos años, las redes sociales se han convertido en un hervidero de información objetiva y subjetiva sobre prácticamente cualquier tema de mayor o menor relevancia. En particular, concentran gran parte de las opiniones sobre la mayoría de tópicos, productos, noticias, etc. que circulan por la red, las cuales son una suculenta fuente de datos de cara a la extracción de información relevante para un amplio abanico de interesados. Este TFG propone un sistema de identificación de las relaciones entre esta información y las personas que la emiten, de forma que puedan sacarse conclusiones relevantes para el usuario, todo ello basado en el Formal Concept Analysis (FCA). En concreto, este trabajo pretende identificar a qué personas seguir en Twitter para estar informado de ciertos temas, un ejemplo básico y punto de partida para otras aplicaciones más complejas como la detección de líderes de opinión (qué personas influyen más en las opiniones de otras y de qué forma) o la clasificación de los turistas en distintos grupos de interés (para identificar las demandas concretas de cada grupo). En definitiva, la motivación de este TFG surge de la necesidad de obtener conocimiento real sobre las influencias de los usuarios en las redes sociales, sus intereses comunes, actividades, etc. Mediante el uso de técnicas basadas en el uso de la lógica y frameworks como el FCA. Este conocimiento puede suponer una base de información muy útil de cara a conocer a los usuarios o clientes de una entidad a través de su comportamiento en las redes.

En resumen, los objetivos principales de este TFG han sido los siguientes

- Extracción de los datos desde internet de forma automática a través de APIs.
- Construcción del contexto formal y extracción del conocimiento en forma de implicaciones.
- Implementación del algoritmo de cálculo de todos los conjuntos cerrados de atributos y sus generadores minimales poniendo énfasis en la maximización del rendimiento.
- Realización de una aplicación que nos muestre una estructura completa de relaciones de influencia entre usuarios y temas, a partir de información recabada de redes sociales
- Aplicación en entornos reales y ejecución y análisis de pruebas.
- Adquisición de conocimientos sobre el desarrollo de aplicaciones web
- Familiarización con técnicas de gestión de proyectos.

3. Metodología

La metodología utilizada en este proyecto ha consistido en un desarrollo incremental, de manera que, a partir de una primera aproximación funcional, se ha construido poco a poco una aplicación completa que cumple todos los requisitos especificados. Para ello, se ha hecho uso de varias herramientas de gestión de proyectos y de métodos para el análisis y diseño de sistemas de información aprendidos durante la carrera. Se ha empleado UML como técnica de modelado y MagicDraw como herramienta de soporte. Por otra parte, se ha utilizado Trello como herramienta de apoyo para la gestión del trabajo.

Las iteraciones que se han realizado son las siguientes:

0. **Estudio previo**: esta iteración se ha dedicado al estudio y comprensión de los fundamentos teóricos necesarios para el desarrollo de la aplicación (Formal Concept Analysis), así como a la familiarización con las diferentes herramientas y tecnologías utilizadas.
1. **Obtención y transformación de los datos de entrada**: al final de esta iteración, la aplicación era capaz, a partir de una fuente, transformar y estructurar los datos obtenidos de ésta para su posterior procesamiento. Todo ello, a través de una interfaz web.
2. **Implementación de las estructuras de datos y algoritmos para el procesamiento de los datos:** al final de esta iteración, la aplicación era capaz de, a partir de los datos ya estructurados, aplicar los algoritmos del FCA para la obtención de los *Labeled Closed Sets.*
3. **Presentación**: al final de esta iteración, la aplicación era capaz de, a partir de los *Labeled Closed Sets* obtenidos, realizar una presentación gráfica de las conclusiones que se pueden sacar del análisis de dichos datos, es decir, presentar las relaciones de influencia entre los participantes y los atributos obtenidos de los datos.
4. **Pruebas experimentales**: en esta iteración, se aplicó la solución desarrollada a diferentes datasets de entrada y se analizaron los resultados (en calidad y eficiencia) para establecer conclusiones.
5. **Cierre**: en esta iteración se procedió a terminar la redacción de la memoria y a la preparación de la presentación del TFG.

Además, cada iteración (de las asociadas a desarrollo de software) ha constado de las siguientes fases:

1. Análisis
    - **Entregable**: Documento de requisitos
    - **Entregable**: Documento de casos de uso
2. Diseño
    - **Entregable**: Diagramas de secuencia
    - **Entregable**: Diagrama de clases
    - **Entregable**: Documento de diseño de pruebas
3. Implementación y pruebas unitarias
    - **Entregable**: Código de la parte implementada

    - **Entregable**: Informe de pruebas unitarias
4. Integración
    - **Entregable**: Código de la aplicación
5. Pruebas
    - **Entregable**: Informe de pruebas de caja blanca

Descripción de los entregables:

- **Documento de requisitos**: contiene una lista de los requisitos funcionales, no funcionales, de información, etc. derivados del análisis de la parte de la aplicación correspondiente a la iteración en curso, todos ellos acompañados de su descripción.
- **Documento de casos de uso**: contiene los diagramas de casos de uso referentes a la parte de la aplicación correspondiente a la iteración en curso.
- **Diagramas de secuencia**: contiene los diagramas de secuencia que especificarán el flujo de mensajes entre las distintas clases de la aplicación.
- **Documento de diseño de pruebas**: contiene un informe en el que se describen las pruebas que se van a realizar en las fases posteriores (unitarias y de caja blanca) y se especifica el resultado previsto en éstas.
- **Informe de pruebas unitarias**: contiene el resultado de las pruebas unitarias realizadas y una valoración del cumplimiento de los resultados esperados.
- **Informe de pruebas de caja blanca**: contiene el resultado de las pruebas de caja blanca realizadas con el código ya integrado en nuestra aplicación principal y una valoración del cumplimiento de los resultados esperados.
- **Código**: contiene el código de la aplicación (controladores, interfaces, scripts, etc.)
- **Diagrama de clases**: contiene un diagrama en el que se especifican las distintas clases y las relaciones entre ellas.

A continuación se expone una tabla que refleja el esfuerzo dedicado a cada una de las iteraciones, así como una comparación entre el tiempo estimado y el tiempo real empleado:

Iteración	Horas estimadas	Horas reales aproximadas
0	27.5	40
1	71.5	110
2	66	80
3	55	50
4	27.5	
5	49.5	

En general, las horas empleadas no varían en exceso de las estimadas. Como se puede observar, en las primeras iteraciones existe una diferencia mayor debido al tiempo empleado en la adaptación a las tecnologías y herramientas

utilizadas, así como en la resolución de problemas derivados de éstas y en el aprendizaje y comprensión de toda la teoría en la que está basada el trabajo.

4. Requisitos

Como hemos indicado anteriormente, este trabajo se ha realizado siguiendo una metodología de desarrollo incremental, por lo que los requisitos se han ido cubriendo poco a poco en cada iteración (recordemos que hay una fase para cada iteración dedicada exclusivamente a ello). Así, a continuación, se presentan los requisitos del sistema. Posteriormente se realizará un análisis de requisitos, especificando cuales de ellos han sido tratados en cada incremento o iteración del proyecto. Los requisitos son los siguientes:

4.1. Requisitos funcionales

RF - 01	Obtener tweets de la fuente de datos (API)
Dependencias	
Descripción	A través de la API de Twitter, obtener los tweets necesarios.
Importancia	Alta
Prioridad	Alta

RF - 02	Procesar los tweets para obtener el Contexto Formal (FC)
Dependencias	
Descripción	Automatizar el procesamiento de los tweets para construir una tabla binaria que represente las relaciones de implicación entre atributos y objetos.
Importancia	Alta
Prioridad	Alta

RF - 03	Obtener los LCS (Labeled Closed Sets)
Dependencias	
Descripción	A partir del algoritmo correspondiente y los datos obtenidos en la generación del contexto formal, obtener los LCS.
Importancia	Alta
Prioridad	Alta

RF - 04	Obtener Concept Lattice o Red de Conceptos
Dependencias	
Descripción	A partir de los LCS, generar un diagrama con los conceptos en el que se aprecien las relaciones entre ellos.
Importancia	Alta
Prioridad	Alta

RF - 05	Obtener tabla de conceptos
Dependencias	
Descripción	A partir de los LCS, generar una tabla con todos los conceptos, es decir, una tabla en la que diga a qué usuarios hay que seguir para estar informados de según qué temas.
Importancia	Alta
Prioridad	Alta

RF - 06	Presentación final de los datos
Dependencias	
Descripción	Realizar una presentación de los conceptos obtenidos, así como del grafo de conceptos (Concept Lattice) y de los conjuntos de datos que se han ido obteniendo en el proceso.
Importancia	Alta
Prioridad	Alta

4.2. Requisitos no funcionales (Seguridad, Fiabilidad, Disponibilidad, Mantenibilidad, Portabilidad)

RNF - 01	La aplicación debe tener arquitectura web
Dependencias	
Descripción	La aplicación debe estar construida y ejecutada sobre una arquitectura web
Importancia	Alta
Prioridad	Alta

RNF - 02	Rendimiento
Dependencias	
Descripción	Poner especial énfasis en la velocidad y recursos utilizados por el algoritmo de FCA.
Importancia	Alta
Prioridad	Alta

RNF - 03	Interfaz sencilla
Dependencias	
Descripción	La aplicación debe tener una interfaz simple y fácil de usasr
Importancia	Baja
Prioridad	Baja

4.3. Análisis de requisitos

Iteración 1

RF-01 Obtener tweets de la fuente de datos (API)

Inicialmente se barajaron varias formas de conseguir los tweets para obtener los objetos y los atributos. Una de ellas fue descargar ficheros con miles de tweets que habría que parsear y analizar. Esta solución era relativamente sencilla a la hora de obtener la información, pero más compleja para procesarla y obtener las palabras o datos que nos interesaban, ya que necesitaríamos la ayuda de más herramientas externas que nos permitieran analizar los textos de forma automática (por ejemplo: Monkey Learn). La otra forma (la finalmente elegida) era descargar los tweets haciendo consultas a la API de Twitter, la cual requería un mayor esfuerzo de programación para hacer las llamadas oportunas, además de una conexión a internet. Pero, por otro lado, el uso de la biblioteca *Twitter4J* (la cual nos permite conectarnos a la API de Twitter desde un fichero Java) nos facilitaba mucho el camino a la hora de procesar los datos. Esto es debido a que, en primer lugar, Twitter4J se encargaba de gestionar las conexiones http oportunas simplemente dándole las credenciales necesarias para conectarse a la API (gestionar estas conexiones en Java es bastante tedioso y propenso a errores, además de dificultar mucho el trabajo de programación), y en segundo lugar, esta librería hacía que pudiéramos trabajar con los tweets como si fueran objetos de Java (objetos de tipo Tweet) , por lo que el "parseo" y procesamiento inicial de la información se hacía bastante más sencillo, ya que se podía a acceder a distintos aspectos del tweet (usuario que lo escribió, número de retweets, fecha, texto, etc.) con solo llamar a ciertos métodos (tweet.getText(), tweet.getUser(), …).

RF-02 Procesar los tweets para obtener el contexto formal.

La elección de la librería Twitter4J, hizo que la cobertura de este requisito fuera relativamente sencilla. Como ya hemos comentado antes, las peticiones realizadas a la API de Twitter mediante Twitter4J nos devolvían objetos de tipo Tweet (una clase propia de dicha librería), todos ellos referenciados desde una instancia de la clase QueryResult (otra clase propia que almacenaba el resultado de la consulta). Esta clase implementa métodos del tipo getTweets() (para devolver una lista de objetos de tipo Tweet), entre otros. Por otra parte, Twitter4J también permite la realización de consultas ad-hoc, es decir, realizar consultas de tweets referentes a unos usuarios en concreto o que incluyan ciertas palabras. Por ejemplo, si queremos todos los tweets escritos por @el_pais en los que hable de #ebola, tendremos que hacer la consulta:

```
twitter.search("from: @el_pais #ebola");
```

Donde *twitter* es un objeto de tipo TwitterFactory, una clase propia de Twitter4J que se encarga de realizar las conexiones http y devolver los resultados mediante objetos. Además, estas consultas ad-hoc permiten restricciones de fecha, coordenadas, etc.

Nuestro contexto formal se compone de una tabla binaria en la que las columnas son los usuarios y las filas los tópicos o hashtags, de manera que, si en la celda (0,0) que corresponde, por ejemplo, a (@el_pais,#ebola) hay un 1, quiere decir que @el_pais tiene tweets en los que habla de #ebola y si tiene un 0 todo lo contrario. Así, rellenar esta tabla resulta más o menos trivial con los datos obtenidos de las consultas mediante Twitter4J: solamente tendremos que comprobar si, para cada consulta del tipo *twitter.search("from: @el_pais #ebola")*, la API devuelve tweets o no. Si devuelve alguno, se pone un 1 en la casilla de la tabla correspondiente a ese usuario y ese tópico y en el caso contrario se pone un 0.

Una vez que tenemos esta tabla, para aplicar los algoritmos necesarios, necesitamos un conjunto de implicaciones, es decir, un conjunto de relaciones de dependencia entre los distintos elementos de la tabla o lo que, en teoría de Bases de Datos, se conoce como "dependencias funcionales". La obtención de este conjunto de implicaciones no es cosa fácil; es más, aún sigue siendo objeto de estudio dentro del campo de las bases de datos. Aun así, existen algoritmos como el TANE o el FD-MINE que, en tiempos de ejecución razonables, te dan una lista de todas las dependencias funcionales de una tabla (en nuestro caso, la tabla mencionada un poco más arriba). En nuestro caso, hemos utilizado una implementación en java del FD-MINE realizada por David Barrientos y Christian Cintrano, dos alumnos de Ingeniería Informática de la Universidad de Málaga cuyo proyecto, realizado en 2014, también pertenecía a la línea de "Social CRM". Como entrada, este algoritmo precisa de una tabla en formato .csv y como salida devuelve un mapa de objetos de tipo Atribute (una clase propia del proyecto mencionado), el cual representa el conjunto de implicaciones. Posteriormente, el programa realiza las modificaciones necesarias para guardar estos datos en las estructuras pertinentes que utilizaremos más tarde para aplicar los algoritmos de FCA.

Iteración 2

RF-03 Obtener los LCS

Los LCS (Labeled Closed Sets) se obtienen mediante el algoritmo *LabeledClosedSets(M,Label,Cicerone,I)*, cuya especificación es la siguiente:

```
Function LabeledClosedSets(M,Label,Cicerone,Γ)
    input  : M, the set of all attributes;
             Label, a structure to build a minimal generator;
             Cicerone, a structure to build a closed set;
             Γ, an implicational system on M;
    output: The set of Labeled Closed Sets
    begin
        repeat
            Σ := Γ;
            Γ := ∅;
            foreach A → B ∈ Σ do
                if A ⊆ Cicerone then Cicerone := Cicerone ∪ B else if
                B ⊈ Cicerone then
                    Γ := Γ ∪ {A ∖ Cicerone → B ∖ Cicerone}
        until Σ = Γ ;
(1)     M := M ∖ Cicerone;
(2)     Mnl := {A ⊆ M | A → B ∈ Γ for some B ⊆ M and
                        C ⊆ A implies C = A for all C → D ∈ Γ} ;
(3)     NC:= {X ⊆ M | A ⊈ X for all A ∈ Mnl};
        LCS := ∅;
(4)     foreach X ∈ NC do LCS := LCS ⊔ {⟨Cicerone ∪ X, {Label ∪ X}⟩}
(5)     foreach A ∈ Mnl do
            LCS := LCS ⊔ LabeledClosedSets(M, Label ∪ A, Cicerone ∪ A, Γ)
        return LCS
    end
```

Este algoritmo está presente en el artículo "*Knowledge discovery in social networks by using a logic-based treatment of implications*", escrito por P.Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego y C. Rossi, el cual ha sido la base fundamental para el desarrollo de este TFG.

Para aplicarlo a nuestro caso, se ha realizado una implementación del mismo en Java. Los datos de entrada han sido ligeramente modificados para obtener un mayor rendimiento y facilitar las labores de programación.

Iteración 3

RF-04 Obtener Concept Lattice o Red de Conceptos

La Red de Conceptos o Concept Lattice es un grafo que representa las relaciones entre los LCS obtenidos anteriormente. Concretamente, este grafo forma un diagrama de Hasse. Un diagrama de Hasse es un grafo no dirigido, cuyos nodos forman un conjunto parcialmente ordenado, en el que, para que exista una arista desde el nodo A hasta el nodo B, se tiene que cumplir que $B \leq A$ y además que no exista ningún nodo C tal que $B \leq C \leq A$. En nuestro caso, se tiene que cumplir que el conjunto cerrado B esté incluido en A y que, además, no exista ningún conjunto C talque B esté incluido en C y C esté incluido en A. De esta manera, eliminamos información redundante sobre las relaciones entre los nodos del grafo.

Como teníamos que realizar operaciones con grafos, se optó por la utilización de una librería llamada JGraph. Esta librería nos proporciona clases y métodos con

los que poder hacer operaciones sobre grafos. En concreto, se ha utilizado la clase UndirectedGraph<String,DefaultEdge>, la cual nos permite generar un grafo no dirigido cuyos vértices son cadenas de caracteres y cuyas aristas son del tipo DefaultEdge, el tipo estándar de arista más simple que proporciona JGraph. Una vez construido el grafo, había que exportarlo a un fichero de tipo imagen para poder visualizarlo. Como JGraph no soporta este tipo de acciones, se utilizó el software Graphviz, un programa que, a partir de un fichero con formato .dot, puede generar una imagen en formato .jpg o .png. Además, Graphviz puede llamarse desde la línea de comandos, por lo que desde nuestra clase Lattice se podía ejecutar perfectamente. Para ello, se genera un fichero .dot del grafo mediante la clase DOTExporter proporcionada por JGraph. Una vez generado este fichero, podemos crear un proceso desde nuestra clase Java con la clase Runtime y ejecutar Graphviz mediante el comando correspondiente.

RF-05 Obtener tabla de conceptos

La tabla de conceptos es una tabla en la que, para cada conjunto posible de hashtags o temas, nos dice a qué usuarios tenemos que seguir. Es el objetivo principal de esta aplicación y refleja de manera clara y concisa el conocimiento extraído de los tweets. La construcción de esta tabla resulta relativamente sencilla si conocemos el contexto formal y tenemos el conjunto de LCS, ya que solo tendríamos que ver, para cada conjunto cerrado de usuarios, la unión de los temas de los que hablan cada uno de ellos. Una vez tenemos esto, si alguno de estos conjuntos de usuarios tuviera un generador minimal de menor tamaño que éste, lo sustituimos (queremos siempre el conjunto mínimo de usuarios).

RF-06 Presentación final de los datos

Para realizar la presentación final de los datos, se optó por crear una interfaz en JSF utilizando Bootstrap como framework de vistas. Bootstrap ofrece una serie de clases CSS que nos permiten darle un diseño bastante presentable a la interfaz sin necesidad de incluir mucho código. Algunas de estas clases nos permiten crear, en una misma vista, varias páginas independientes a través de las cuales podemos navegar gracias a una serie de “pestañas” o “etiquetas”. Este diseño hace que se pueda incluir mucha información en una misma vista sin que la página sea excesivamente grande. En realidad, lo que hace Bootstrap es, una vez renderizada la vista completa, mediante Javascript, ocultar y mostrar la información que queramos en función de la pestaña seleccionada. De esta manera nos ahorramos cargar una vista nueva cada vez que queramos acceder a distintas partes del resultado. En nuestro caso se ha optado por incluir 4 pestañas:

1. **Contexto**: en este apartado se incluye el contexto formal en forma de tabla, así como el contexto “contrario” que es el que se va a utilizar en el procesamiento de los datos.
2. **Implicaciones**: aquí se van a mostrar las implicaciones o dependencias funcionales deducidas del contexto formal.
3. **LCS**: en esta pestaña se mostrarán los LCS obtenidos mediante la aplicación del algoritmo LabeledClosedSets sobre el contexto formal.

4. **Concept Lattice**: aquí se muestra el grafo que representa la red de conceptos mediante una imagen en formato .jpg. Cada nodo del grafo incluye un conjunto cerrado de usuarios, su generador minimal y los temas o hashtags de los que hablan.
5. **Tabla de conceptos**: en este apartado se muestra la tabla de conceptos mencionada anteriormente. Esta se compone de dos columnas: “Interesado en” y “Seguir”.

5. Fundamentos

Este apartado del trabajo se va a dedicar a analizar los fundamentos teóricos relativos al Análisis de Conceptos Formales (FCA).

5.1. Introducción

Es innegable el hecho de que estamos viviendo una nueva etapa en lo que a la sociedad de la información respecta. Internet nos ha hecho accesible información que hasta ahora sólo podía encontrarse mediante semanas o meses de intensa búsqueda. Hoy, con las palabras clave adecuadas y un "clic", tenemos a nuestra disposición millones de libros, artículos, ensayos, textos, vídeos, etc. sobre casi cualquier tema. Tal es la cantidad de datos, que podemos decir que vivimos en un estado de "sobreinformación" permanente. En particular, las redes sociales han sido una de las grandes "culpables" en este asunto. Culpables en el sentido de que han sido las responsables de que se creen grandes redes de información proporcionada por una variedad de fuentes incalculable. Así, cada día son más las personas que se dedican a la búsqueda de herramientas que nos permitan hacer un uso útil de toda esta información. En este trabajo se ha propuesto la creación de una herramienta que minimice la cantidad de fuentes a las que tenemos que acceder para informarnos de los temas o tópicos en los que estemos interesados. En concreto, un sistema que nos proporcione un número concreto y mínimo de usuarios de la red social Twitter (una de las RRSS en las que más información se mueve) a los que debemos seguir para estar enterados de dichos temas. Para ello, el trabajo se ha basado en un framework puramente teórico basado en FCA (Formal Context Analysis), el cual persigue la construcción de un modelo matemático que permita la extracción de información relevante o conocimiento a partir de datos más desestructurados. A continuación, se va a proceder a realizar una explicación más profunda del FCA, recalcando los aspectos que más nos han servido para lograr nuestros objetivos en este trabajo.

5.2. Análisis de Conceptos Formales (FCA)

Como ya hemos mencionado anteriormente, FCA es una teoría matemática cuyo objetivo más general es la extracción de conocimiento útil a partir de datos más desestructurados. Esta teoría modela el término de *concepto* en términos de la teoría de retículos. Fue ideada por Bernhard Ganter y ha sido muy útil en los sistemas de extracción de conocimiento por sus buenos resultados y simplicidad. FCA parte de lo que Ganter denomina un *Contexto Formal* $\boldsymbol{K}$ que se define como una tupla (G, M, I). G es un conjunto de objetos de estudio, M un conjunto de atributos e I una relación binaria entre G y M. En nuestro caso, los atributos representan a los usuarios que escriben la información, los objetos a los temas o tópicos y la relación al hecho de que, si el usuario u ha escrito sobre el tema t, entonces u está relacionado con t. Por ejemplo, un contexto formal representado en forma de tabla podría ser el siguiente:

	#ébola	#Obama	#educación
@el_pais	x	x	
@EFE		x	
@CNN	x		x

Además, en FCA se definen dos operadores de cierre:

Dado un $A \subseteq G$:

$$A' := \{m \in M | \forall g \in A : (g, m) \in I\}$$

Y dado un $B \subseteq M$:

$$B' := \{g \in G | \forall m \in B : (g, m) \in I\}$$

Estos operadores constituyen una relación dentro del conjunto potencia de los conjuntos G y M. A esta relación se le llama *Conexiones de Galois*, y son la base fundamental para la definición de *Concepto*. Un *Concepto Formal* de un *Contexto formal* (G, M, I) es un par (A, B) tal que:

$$A \subseteq \mathrm{G},\ \mathrm{B} \subseteq \mathrm{M},\ A' = B,\ \mathrm{y}\ A = B'$$

En cierto modo, podríamos decir que un concepto formal es un conjunto de objetos y atributos que se determinan los unos a los otros mutuamente.

Básicamente, el objetivo principal de este trabajo consiste en obtener los *Conceptos Formales* a partir del *Contexto* y que estos, a su vez, sean *mínimos*. Para ello, vamos a hacer uso de varios términos y conceptos que expondremos a continuación, los cuales están basados tanto en la lógica formal como en la teoría de bases de datos y dependencias funcionales, así como en la llamada **Lógica Simplificada de Dependencias Funcionales** o $\boldsymbol{SL_{FD}}$:

- Sea Γ un conjunto de implicaciones lógicas (del tipo $a \rightarrow b$) y A un conjunto de atributos. El **cierre** de A en $\boldsymbol{SL_{FD}}$ está definido como el conjunto máximo de atributos A_Γ^+ tal que Γ implica que $A \rightarrow A_\Gamma^+$. Es decir, es el conjunto máximo de atributos que pueden "deducirse" a partir del conjunto A y el conjunto de implicaciones Γ.
- Sea Γ un conjunto de implicaciones, M un conjunto de atributos y A un subconjunto de M. Decimos que A es un **generador minimal** (mingen) si $X_\Gamma^+ = A_\Gamma^+$ implica que $X = A$ para todo conjunto de atributos X (subconjunto de A). Es decir, es el subconjunto mínimo de A a partir del cual se puede generar A_Γ^+.
- Sea M un conjunto de atributos y Γ un conjunto de implicaciones sobre éste. Decimos que el *Conjunto de conjuntos etiquetados* o *LCS* (Labeled Closed sets) respecto a Γ es:

$$\{\langle C, mg(C)\rangle \mid C \subseteq M, C_\Gamma^+ = C\}$$

- Donde $mg(C)$ es un generador minimal de C. Es decir, un LCS está formado por un conjunto de atributos y un generador minimal de éste.

Así, el conjunto de todos los LCS sobre un contexto formal puede interpretarse como una "red de conceptos" jerárquica, a la que se han añadido "etiquetas" con el generador minimal para cada conjunto cerrado. Existe un algoritmo especificado en el artículo "*Knowledge discovery in social networks by using a logic-based treatment of implications*" que nos permite la extracción de todos los LCS a partir de un contexto formal. Es el siguiente:

Function `LabeledClosedSets`$(M, Label, Cicerone, \Gamma)$

input : M, the set of all attributes;
Label, a structure to build a minimal generator;
Cicerone, a structure to build a closed set;
Γ, an implicational system on M;
output: The set of Labeled Closed Sets
begin
  repeat
    $\Sigma := \Gamma$;
    $\Gamma := \varnothing$;
    foreach $A \to B \in \Sigma$ **do**
      if $A \subseteq$ Cicerone **then** Cicerone := Cicerone $\cup B$ **else if**
      $B \not\subseteq$ Cicerone **then**
        $\Gamma := \Gamma \cup \{A \setminus \text{Cicerone} \to B \setminus \text{Cicerone}\}$
  until $\Sigma = \Gamma$;
(1) $M := M \setminus$ Cicerone;
(2) Mnl := $\{A \subseteq M \mid A \to B \in \Gamma$ for some $B \subseteq M$ and $C \subseteq A$ implies $C = A$ for all $C \to D \in \Gamma\}$;
(3) NC:= $\{X \subseteq M \mid A \not\subseteq X$ for all $A \in$ Mnl$\}$;
    LCS := $\varnothing$;
(4) **foreach** $X \in$ NC **do** LCS := LCS $\sqcup \{\langle$Cicerone $\cup X, \{$Label $\cup X\}\rangle\}$
(5) **foreach** $A \in$ Mnl **do**
      LCS := LCS $\sqcup$ LabeledClosedSets$(M,$ Label $\cup A,$ Cicerone $\cup A, \Gamma)$
  return LCS
end

Por tanto, a partir de los LCS generados sobre un Contexto Formal, se pueden estudiar los *Conceptos* de los que hablábamos al principio, de manera que estos sean minimales. El mayor problema de los sistemas basados en implicaciones radica en que el conjunto de éstas suele ser demasiado grande y, por lo tanto, poco eficiente. Sin embargo para solucionar este problema existen técnicas matemáticas para transformar este conjunto en *base*, es decir, el menor conjunto de implicaciones que mantiene las mismas propiedades. Además es posible usar esta base y los operadores de cierre para obtener *conceptos*.

La implementación de este algoritmo, así como la obtención de los datos de sus fuentes y su posterior interpretación, serán los puntos clave de este TFG.

Para comprender mejor todos estos conceptos, vamos a exponer un pequeño ejemplo expuesto en el artículo "*Knowledge discovery in social networks by using a logic-based treatment of implications"*:

Suponemos que tenemos el siguiente contexto formal:

	@BBCNews	@CNN	@rtve	@AlJazeera
#HongKong	×		×	
#Ebola		×		×
#WorldSeries			×	×
#Africa	×	×		×

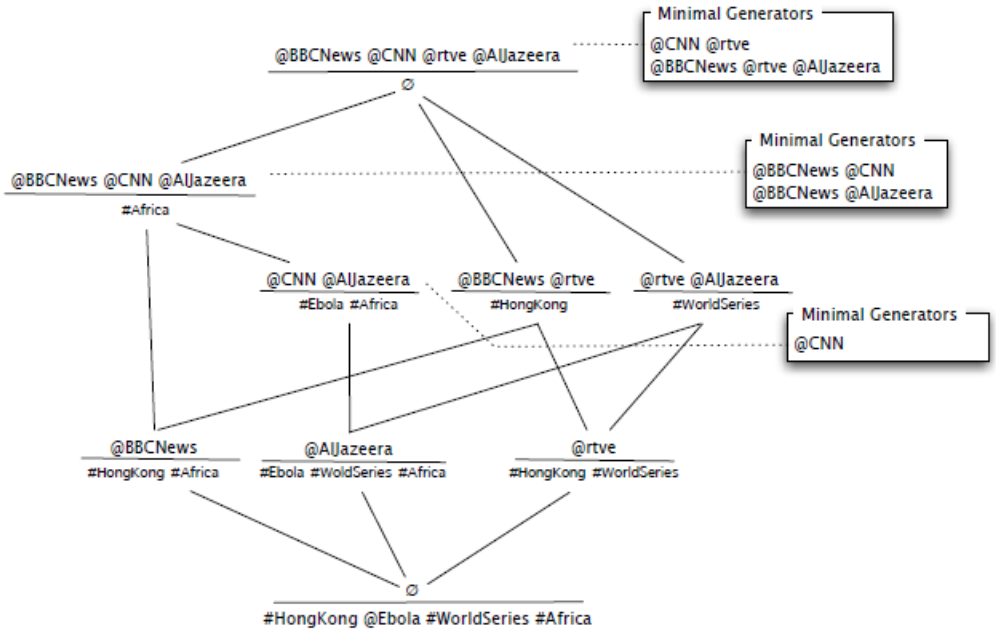
En esta tabla, las casillas marcadas simbolizan la relación "el usuario @xxx no habla sobre el tema #yyy". Se ha optado por este contexto "inverso" debido a que la interpretación de las implicaciones o dependencias funcionales derivadas del mismo, encaja mucho mejor con el problema que queremos resolver. Dichas implicaciones son las siguientes:

$$\begin{gathered}\text{@CNN} \rightarrow \text{@AlJazeera} \\ \text{@CNN, @rtve, @AlJazeera} \rightarrow \text{@BBCNews} \\ \text{@BBCNews, @AlJazeera} \rightarrow \text{@CNN}\end{gathered}$$

Aplicando el algoritmo $LabeledClosedSets()$ obtenemos los siguientes LCS:

{ ⟨ @BBCNews,{@BBCNews} ⟩,
⟨ @rtve,{@rtve} ⟩,
⟨ @AlJazeera,{@AlJazeera} ⟩,
⟨ @BBCNews @rtve,{@BBCNews@rtve} ⟩,
⟨ @rtve@AlJazeera,{@rtve@AlJazeera} ⟩,
⟨ @CNN@AlJazeera,{ @CNN} ⟩,
⟨ @BBCNews@CNN@AlJazeera,{@BBCNews@CNN,@BBCNews@AlJazeera} ⟩,
⟨ @BBCNews@CNN@rtve@AlJazeera,{@CNN@rtve,@BBCNews@rtve@AlJazeera} ⟩ }

El siguiente grafo corresponde a la "Red de conceptos" o "Concept Lattice" de la que hemos hablado anteriormente. Este refleja las relaciones entre los distintos conjuntos cerrados mediante un diagrama de Hasse:



Cada uno de estos conjuntos consta de un conjunto cerrado (primer conjunto) y un generador minimal del mismo (segundo conjunto), lo que quiere decir que a partir de este segundo conjunto, aplicando las implicaciones lógicas especificadas más arriba, se puede obtener el primero (cierre). Así, una vez tenemos los LCS, podemos empezar a sacar conclusiones. Por ejemplo, si consideramos el Concepto (#Ebola #Africa, @CNN @AlJazeera), podemos interpretarlo como “Los usuarios que solo siguen a @CNN y @AlJazeera no están informados sobre los temas #Ebola y #Africa”, lo que implícitamente nos está diciendo “Si un usuario está interesado en todas las noticias excepto en #Ebola y #Africa, siguiendo solamente a @CNN y a @AlJazeera está bien informado” (por este motivo utilizamos el contexto “contrario”).

Siguiendo esta interpretación, se puede obtener conocimiento completo sobre la relación entre todos los tópicos y usuarios (todos los conceptos). La siguiente tabla expone los usuarios a los que se debe seguir para estar informado de los temas especificados.

#HongKong	@AlJazeera
#Ebola and #WolrdSeries	@BBCNews
#Ebola and #Africa	@rtve
#HongKong and #WorldSeries	@CNN
#HongKong, #Ebola and #Africa	@rtve and @AlJazeera
#Ebola, #WorldSeries and #Africa	@BBCNews and @rtve
#HongKong, #Ebola and #WorldSeries	@BBCNews and @CNN either @BBCNews and @AlJazeera
All the news from the four topics	@CNN and @rtve either @BBCNews, @rtve and @AlJazeera

Este ejemplo es una pequeña prueba de la gran potencia y utilidad de los algoritmos expuestos anteriormente.

6. Entorno tecnológico

Para este TFG, se ha desarrollado una aplicación Java EE (versión 7), implementada sobre las siguientes tecnologías y herramientas:

- **Glassfish**: como servidor de aplicaciones sobre el que va montado todo el sistema. Su rendimiento, sencilla configuración y fácil integración con el IDE Netbeans fueron los puntos clave para su elección.
- **JSF**: como framework para las interfaces. La posibilidad de utilizar XHTML junto con las etiquetas específicas de JSF lo hacen una opción escalable y rápida para mostrar interfaces sencillas. Además, se integra a la perfección con el "backend" de la aplicación.
- **JTLS**: librería adicional que añade algunas etiquetas a JSF, como sentencias de control de flujo(<c:if>) y bucles (<c:forEach>) entre otras.
- **CSS3**: framework principal utilizado para modelar el aspecto de las interfaces.
- **Bootstrap**: framework para CSS y JavaScript. Su sencillez y su extendido uso lo hacen una herramienta esencial para realizar interfaces sencillas.
- **Javascript**: usado para ejecutar ciertos programas en el cliente, sobre todo de validación de formularios.
- **JQuery**: librería para Javascript que permite simplificar la manipulación del documento HTML y el DOM, la captura y manejo de eventos, etc.
- **Twitter public API**: esta API es la fuente principal de datos de la aplicación. Mediante una cuenta "developer" en Twitter, se puede acceder a ella como una API REST que permite realizar todo tipo de consultas personalizadas.
- **Twitter4J**: librería para la utilización de la API de Twitter desde ficheros Java. Debido a que las conexiones http desde ficheros Java son tediosas de programar y muy propensas a errores, Twitter4J nos proporciona una biblioteca relativamente sencilla de usar con la que, además, podemos manipular los datos recibidos desde la API como si fueran objetos de Java. Esto ha facilitado mucho las labores de procesamiento de los datos para su adaptación a la entrada de los algoritmos utilizados.
- **FCAUtilitiesProject API**: API desarrollada por Christian Cintrano y David Barrientos, alumnos que realizaron en 2014 un TFG en la línea "Social CRM". Esta API nos proporciona clases y métodos realmente útiles para trabajar con el Formal Concept Analysis. Concretamente, se ha utilizado para obtener un conjunto de dependencias funcionales o implicaciones a partir de un contexto dado en formato .csv (tabla) mediante una implementación en Java del algoritmo FD-Mine.
- **JGraph**: librería de Java para el modelado de grafos.
- **Graphviz**: software para representación gráfica de grafos. Ofrece la lectura de ficheros .dot para transformarlos en imágenes .jpg y .png.

7. Desarrollo

En este apartado del trabajo se van a explicar aspectos más de tallados relativos al desarrollo del proyecto, tales como su estructura y principales modelos.

7.1. Aspectos básicos

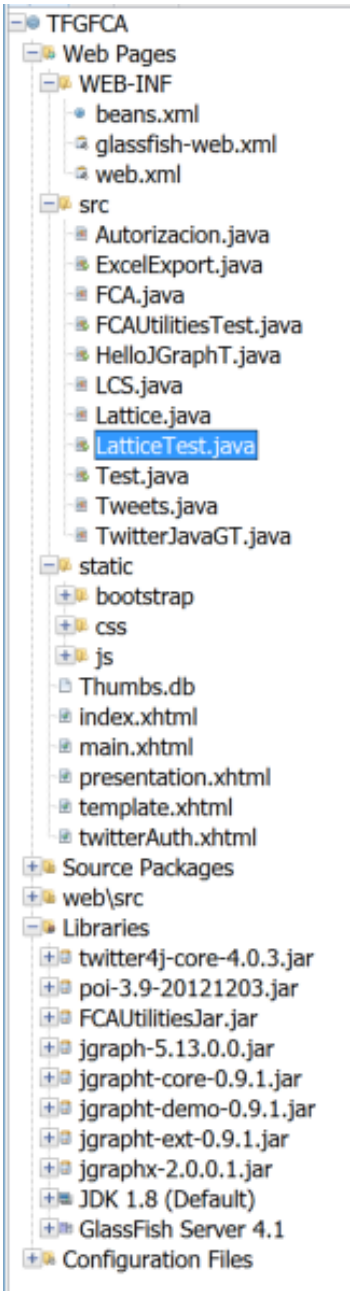
El desarrollo del sistema se realizó en un PC con Windows 8.1 empleando el IDE Netbeans 8.0.2 y un servidor Glassfish para la construcción de la aplicación web. Está estructurado en un solo proyecto al que hemos llamado TFGFCA. Dentro de este proyecto se encuentran los ficheros fuente en Java, así como las interfaces en JSF necesarias. Por motivos de facilidad en el desarrollo se ha incluido todo en este proyecto único, pero, como veremos más tarde, los ficheros están estructurados de manera que la parte de FCA (el pilar central del trabajo) sea totalmente independiente del resto de la aplicación, con la intención de poder utilizarla en otros sistemas a modo de API (sin acoplamiento).

7.2. Estructura del proyecto

La estructura es la de un proyecto Java EE, siguiendo el patrón *modelo-vista-controlador*. En él se distinguen las siguientes partes:

- **Src**: en donde se encuentran los controladores de la aplicación (ficheros .java).
- **Libraries**: librerías externas necesarias para la aplicación, como por ejemplo: Twitter4J (para acceder a la API de Twitter) o FCAUtilitiesProject (la API utilizada para realizar algunas operaciones de FCA)
- **Web Pages**: es el directorio donde se encuentran los ficheros de las interfaces JSF, además de los siguientes subdirectorios:
    - **WEB-INF**: aquí se encuentran algunos ficheros de configuración del servidor y de la aplicación web.
    - **Src**: es una “copia” de la carpeta SRC, una especie de directorio espejo que facilita las rutas de acceso a los ficheros.
    - **Static**: incluye los ficheros estáticos de la aplicación, como por ejemplo los ficheros CSS, las librerías de Bootstrap o los ficheros Javascript.

La estructura completa se puede ver en la siguiente imagen:

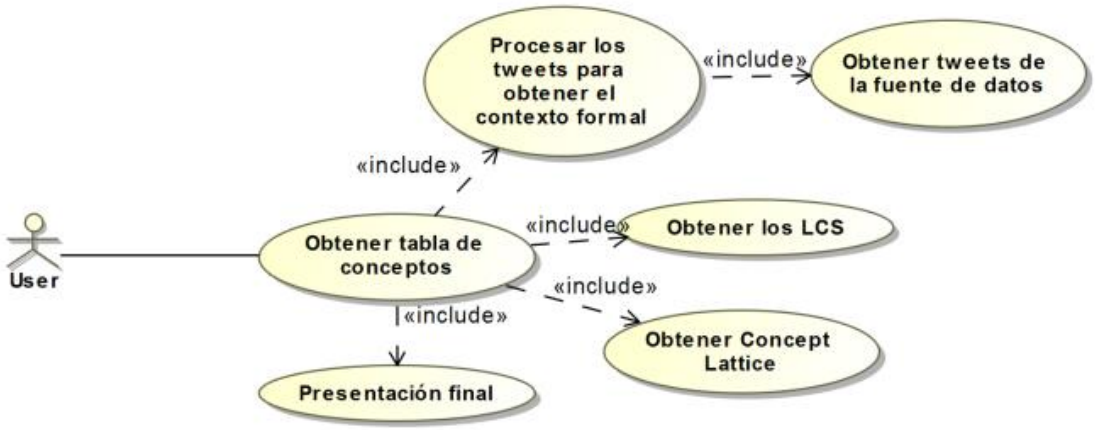


Los ficheros “Test.java” y “FCAUtilitiesTest.java” son ficheros de prueba para realizar distintos tests de la funcionalidad del sistema. Del resto de clases hablaremos más en profundidad en el apartado de “Diagramas de clase”.

7.3. Modelos

7.3.1. Casos de uso

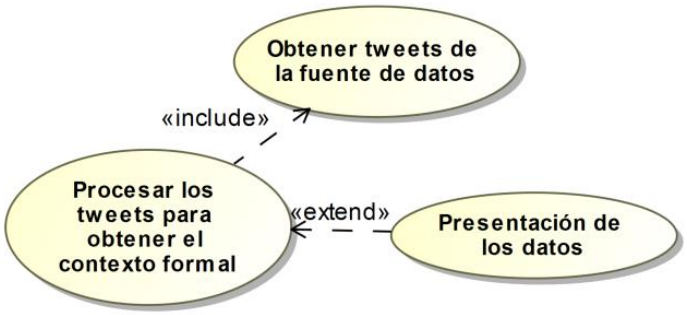
Como ya hemos comentado anteriormente, el desarrollo de nuestro proyecto ha seguido un modelo incremental, por lo que los casos de uso se han ido tratando progresivamente en las distintas iteraciones. El diagrama completo de casos de uso se puede apreciar en la siguiente imagen:



A continuación se van a describir todos los casos de uso ordenados por iteración, según se han ido tratando:

Iteración 1

Como podemos observar en el análisis de requisitos (punto 4.3), en esta iteración se tiene como objetivo la obtención del contexto formal. Así, se pueden distinguir los siguientes casos de uso:

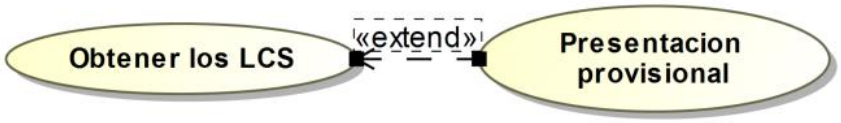


- **Obtener tweets de la fuente de datos**: Indica la conexión con la API de Twitter para la obtención de los tweets con los datos que ha introducido el usuario desde la interfaz web.
- **Procesar los tweets para obtener el contexto formal**: una vez que se tienen los tweets, se realiza un procesamiento de los mismos para obtener una tabla que será nuestro Contexto Formal. Además, este caso de uso incluye la obtención de las implicaciones o dependencias de dicha tabla.
- **Presentación de los datos**: este caso de uso es auxiliar. Consiste en una presentación provisional de los datos. Este caso de uso seguirá presente

en nuestro diagrama a medida que avancen las iteraciones, pero irá cambiando de lugar en función de qué datos queramos presentar.

Iteración 2

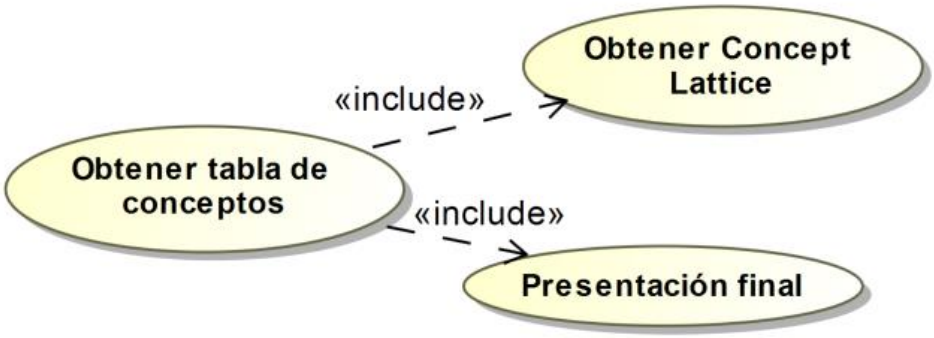
En esta iteración, el objetivo era obtener un conjunto de LCS a partir del contexto formal. Atendiendo exclusivamente a los requisitos de esta iteración, el diagrama de casos de uso quedaría de la siguiente forma:



- **Obtener los LCS**: a partir del Contexto Formal, aplicando los algoritmos y técnicas de FCA correspondientes, obtendremos los LCS.
- **Presentación provisional**: este caso de uso representa exactamente lo mismo que en la iteración anterior.

Iteración 3

En esta iteración se procedió a la interpretación de los LCS anteriormente obtenidos, así como a la presentación del conocimiento obtenido de ésta. Los casos de uso abordados en esta iteración son los siguientes:

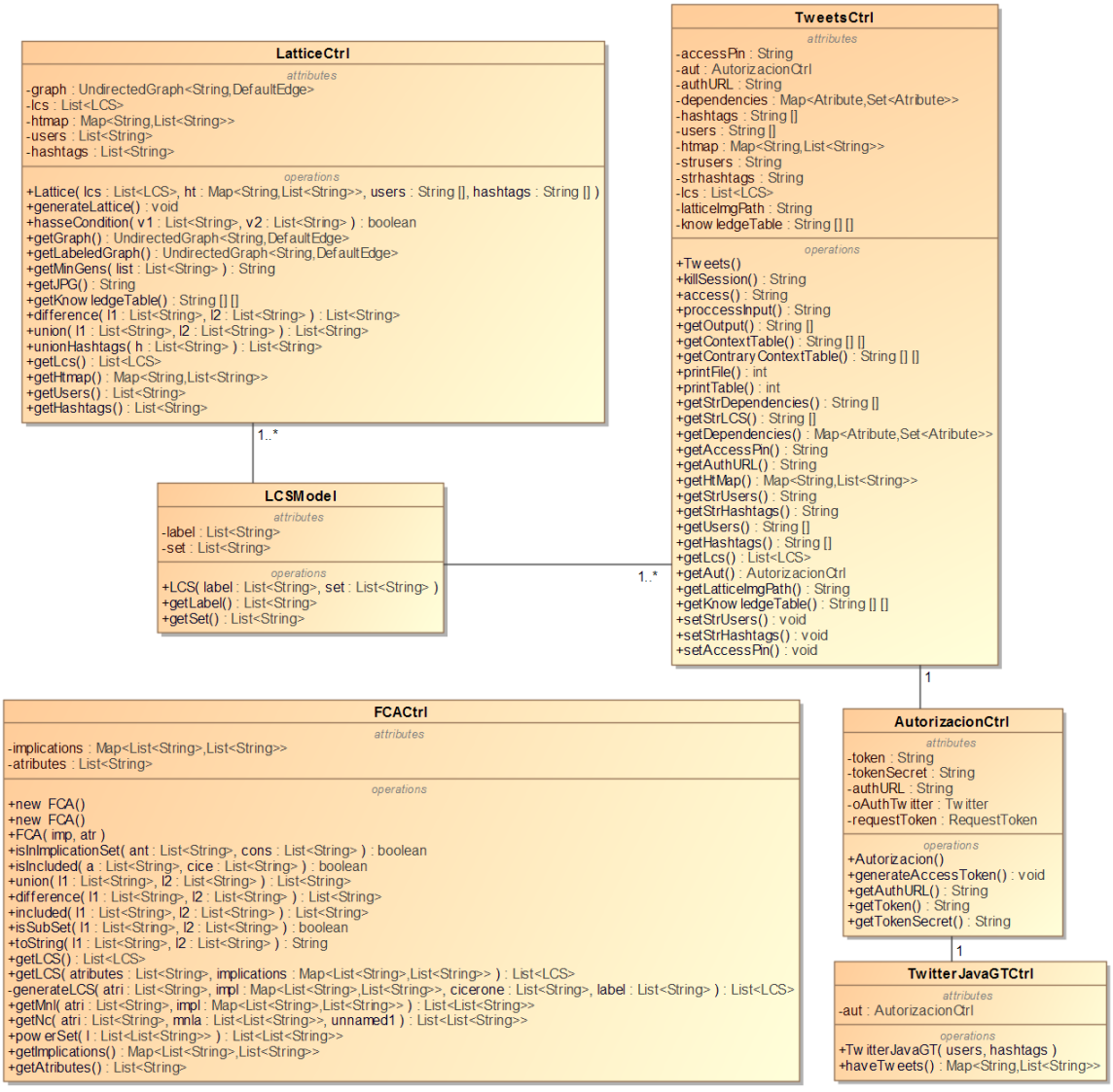


- **Obtener tabla de conceptos**: a partir de los LCS obtenidos en la iteración 2, la aplicación determina, basándose en el Contexto Formal, todos los conceptos que se pueden sacar del estudio del Contexto, es decir, nos dice a qué usuarios debemos seguir para informarnos de según qué temas o tópicos.
- **Obtener Concept Lattice**: como hemos indicado más arriba, el Concept Lattice o Red de Conceptos es un grafo o diagrama de Hasse que nos expone de manera explícita la relación entre los atributos del contexto. A partir de los LCS, la aplicación debe generar este diagrama y guardarlo en una imagen.

- **Presentación final**: en este punto, la aplicación debe presentar todos los datos obtenidos del contexto de la forma más simple e intuitiva posible para facilitar la comprensión del usuario.

7.3.2. Diagramas de clases

A continuación se definen las clases y métodos principales de este proyecto. Estas clases se ilustran en siguiente diagrama:



Como se puede observar, se ha seguido el patrón Modelo-Vista-Controlador. Cada clase está etiquetada en función del rol que cumple (Model,Ctrl).

Autorización

La clase Autorización es la encargada de la autenticación con la API de Twitter. Dentro de esta, mediante la librería ya mencionada anteriormente Twitter4J, se crean las conexiones necesarias y se realizan las operaciones pertinentes para obtener los tokens que nos autorizan a usar la API de Twitter para obtener los tweets. Contiene un constructor sin argumentos y varios métodos entre los que destaca *getToken()*: este método devolverá el token de autenticación necesario para hacer las consultas a la API.

TwitterJavaGT

Esta clase es la encargada de hacer las consultas a la API de Twitter. Se compone de un constructor con un argumento *a* de tipo *Autorización* y un método *haveTweets(String[] users, String[] hashtags)* que nos devolverá un *Map<String,List<String>>* donde la clave (key) es el usuario en cuestión y el valor asociado a esta clave (value) es la lista de los hashtags o temas de los que la API ha encontrado tweets escritos por el usuario.

LCS

La clase LCS representa un Labeled Closed Set. Tiene dos atributos de instancia que representan un conjunto cerrado (*label*) y su generador minimal (*set*). Además de los getters y setters, la clase implementa un método de comparación (*equals*). No implementa ningún tipo de funcionalidad adicional, ya que el objetivo de esta clase es dar estructura y aumentar el nivel de abstracción del LCS para facilitar su procesamiento.

FCA

La clase FCA es la encargada de realizar todas las operaciones relacionadas con el Formal Concept Analysis. Como se puede observar en el diagrama de clases, no está relacionada con ninguna otra clase. Además, todos sus atributos de instancia son de tipos primitivos o nativos de Java. Esto es porque uno de los requisitos primarios de esta clase era que fuera totalmente independiente del resto, con el objetivo de poder "extraer" o empaquetar dicha clase a modo de API y poder realizar las operaciones de FCA en cualquier otro contexto. Por tanto, los niveles de cohesión y de abstracción debían ser mínimos o nulos.
Consta de dos atributos de instancia:

- **atributes**: es una lista de todos los atributos (usuarios, en nuestro caso).
- **implications**: es un conjunto de implicaciones entre los atributos.

Juntas, estas dos variables determinan el contexto formal al que vamos a aplicar los algoritmos y operaciones del FCA. Además, podemos ver un constructor principal *public* ***FCA****(Map<Atribute,Set<Atribute>> imp, List<String> attr)* (existen más constructores por escalabilidad, pero se ha especificado este debido a que es el que vamos a utilizar en nuestro caso), cuyos parámetros representan una lista de implicaciones y una lista de atributos, respectivamente. Este constructor se implementa con la intención de transformar los datos que

nos devuelve la librería *FCAUtilitiesProject* en datos más simples y adecuados para su posterior procesamiento. Dicha librería utiliza la clase *Atribute* para representar un atributo o conjunto de atributos.
Por otra parte, además de los getters y setters, esta clase dispone de varios métodos importantes:

public boolean ***isInImplicationSet*** *(List<String> ant , List<String> cons);*
public boolean ***isIncluded*** *(List<String> a , List<String> cice);*
public List<String> ***union*** *(List<String> l1 , List<String> l2);*
public List<String> ***difference*** *(List<String> l1 , List<String> l2);*
public List<String> ***included*** *(List<String> l1 , List<String> l2);*
public boolean ***isSubSet*** *(List<String> l1 , List<String> l2);*
public List<String> ***addNoDuplicates*** *(List<String> l1 , List<String> l2);*
public List<List<String>> ***powerSet****(List<String> l);*

Estos métodos realizan operaciones sobre conjuntos. Estas operaciones son esenciales para la implementación de los algoritmos de FCA y suponen un ahorro de tiempo y de código muy grande, ya que se utilizan en muchos casos.

public String ***toString*** *()*

Este método devuelve una representación entendible del contexto formal.

public List<LCS> ***getLCS*** *();*
public List<LCS> ***getLCS*** *(List<String> atributes , Map<List<String>,List<String>> implications);*
public List<LCS> ***generateLCS*** *(List<String> atri , Map<List<String> , List<String>> impl , List<String> cicerone , List<String> label);*
public List<List<String>> ***getMnl*** *(List<String> atri , Map<List<String>, List<String>> impl);*
public List<List<String>> ***getNc*** *(List<String> atri , List<List<String>> mnla);*

Estos métodos son los encargados de hacer las operaciones los algoritmos de FCA. Concretamente, los dos métodos principales son ***getLCS()***, y ***getLCS(List<String> atributes , Map<List<String>,List<String>> implications)***, los cuales son dos implementaciones del mismo algoritmo (*LabeledClosedSets*, del que hablamos en la sección de fundamentos). El primero de ellos genera los LCS a partir del contexto formal que está guardado como variables de instancia y el segundo hace exactamente lo mismo a partir de un contexto dado en los parámetros. El método ***generateLCS(…)*** es al que llaman los dos anteriores. Éste se llama de forma recursiva como parte del algoritmo *LabeledClosedSets*. Los dos últimos métodos, ***getMnl(…)*** y ***getNc(…)*** son métodos auxiliares que calculan los conjuntos *MNL* y *NC* del algoritmo *LabeledClosedSets*.

Lattice

La clase Lattice es la encargada de realizar todas las operaciones y acciones relacionadas con la Red de Conceptos. En un principio se optó por implementar los métodos de esta clase en la clase FCA, pero como queríamos que esta última

fuera lo más independiente posible del resto de la aplicación, se optó por hacer una nueva clase que se encargara de ello. Consta de un constructor *public* ***Lattice*** *(List<LCS> lcs, Map<String,List<String>> ht, String[] users, String[] hashtags)* cuyos parámetros representan al conjunto de los LCS, una estructura que representa al contexto formal (la misma que el atributo *htmap* de la clase Tweets), el conjunto de usuarios y el conjunto de hashtags, respectivamente. Tiene 5 variables de instancia:

- **graph**: representa el Concept Lattice propiamente dicho. Consiste en un grafo no dirigido implementado con la clase UndirectedGraph de JGraph. Como ya hemos comentado anteriormente, cada nodo representa un conjunto cerrado de usuarios y cada arista la relación entre ellos.
- **lcs, htmap, users, hashtags**: estas variables almacenan los parámetros pasados en el constructor que ya hemos comentado antes.

Además del constructor, las variables de instancia y los getters y setters, la clase se compone de los siguientes métodos:

public void ***generateLattice*** *();*

En el constructor, la variable **grafo** se inicializa con un objeto vacío. Este método se encarga de añadir a la variable **grafo** los vértices y aristas correspondientes que determinan el Concept Lattice. Esta operación se realiza fuera del constructor debido a que es necesario el uso de métodos de la propia clase por cuestiones de limpieza del código.

public boolean ***hasseCondition*** *(List<String> v1, List<String> v2);*

Este método comprueba si los vértices v1 y v2 cumplen la condición de Hasse (recordemos que este grafo es un diagrama de Hasse) para que exista una arista desde v1 hasta v2 en el grafo.

public UndirectedGraph<String,DefaultEdge> ***getLabeledGraph*** *();*

En el método ***generateLattice()*** los nodos del grafo sólo contienen los conjuntos cerrados. Este método genera otro grafo cuyos nodos contienen, además de dichos conjuntos, los generadores minimales del mismo y el conjunto de hashtags asociados. Realmente éste es el grafo que vamos a exponer en la interfaz de presentación final, ya que contiene toda la información relevante del estudio del contexto.

public String ***getMinGens*** *(List<String> closedSet);*

Suponiendo que el parámetro *closedSet* represente un conjunto cerrado de usuarios, este método devuelve una cadena de caracteres con los generadores minimales del mismo separados por comas. Básicamente, este método se encarga de realizar una presentación de todos los minGens del conjunto en una sola cadena de caracteres.

public String ***getJPG*** *();*

Este método es uno de los más importantes de toda la clase. Su cometido es generar una imagen en formato .jpg que represente el Concept Lattice. Para ello, se genera un fichero en formato .dot mediante la clase DOTExporter de JGraph, el cual guarda los datos del grafo. Luego se hace una llamada al programa Graphviz, al que se le pasa como parámetro la ruta del fichero .dot, el cual genera la imagen .jpg. Por último, el método devuelve la ruta a dicha imagen en una cadena.

*public String[][] **getKnowledgeTable** ();*

Este método devuelve un array de dos dimensiones que representa la tabla en la que se exponen todos los Conceptos que el sistema ha generado. Es utilizado por la interfaz de presentación.

*public List<String> **difference** (List<String> l1, List<String> l2);*
*public List<String> **union** (List<String> l1, List<String> l2);*
*public List<String> **unionHashtags** (List<String> h);*

Estos métodos realizan operaciones con conjuntos similares a los métodos con mismo nombre de la clase FCA.

Tweets

La clase Tweets es el nodo central de todo el proyecto. Esta es la encargada de crear instancias de las demás clases y llamar a los métodos correspondientes. Como comentamos anteriormente, este proyecto se ha realizado con la intención de que cada componente sea lo más independiente posible. Así, esta clase sería el nexo de unión de todas las demás clases (integra todos los comportamientos) y la encargada de la comunicación con las interfaces (es un *Backing Bean* de Java EE). La clase Tweets consta de algunas variables de instancia:

- **String strusers**: almacena los usuarios en formato String separados por punto y coma.
- **String[] users**: almacena los usuarios en un array de String
- **String strhashtags**: almacena los hashtags/temas en formato String separados por punto y coma.
- **String[] hashtags**: almacena los hashtags/temas en un array de String.
- **Map<String, List<String>> htmap**: almacena un mapa en el que la clave (key) es un usuario y el valor asociado a esta clave (value) es la lista de los hashtags o temas relacionados con éste.
- **Map<Atribute, Set<Atribute>> dependencies**: almacena un conjunto de dependencias funcionales o implicaciones generadas a partir del contexto formal.
- **List<LCS> lcs**: almacena un conjunto de LCS devuelto por la clase FCA.
- **String authURL**: almacena una url necesaria para realizar la autenticación con la API de Twitter.
- **String accessPin**: almacena una cadena alfanumérica necesaria para realizar la autenticación con la API de Twitter.

- **Autorizacion aut**: es el objeto desde el que vamos a obtener el token de autenticación para la API de Twitter (la clase *Autorizacion* definida anteriormente).
- **latticeImgPath**: almacena la ruta donde se encuentra la imagen en formato .jpg del grafo de conceptos.
- **knowledgeTable**: almacena la tabla con todos los conceptos generados por el sistema.

A primera vista puede parecer que en estas variables de instancia se almacenan muchos datos redundantes o calculables, pero se ha hecho por una buena razón. Esta clase, al ser un *Backing Bean*, tiene lo que se llama un *ámbito* definido. Esto quiere decir que el objeto se crea y se destruye de forma automática según el ámbito que le demos. En este caso, este *Backing Bean* tiene un ámbito *de sesión* o *Session Scoped*. Esto se refiere a que se crea una instancia de esta clase cuando comienza la sesión HTTP y se destruye cuando termina la sesión. Por tanto, al tener este ciclo de vida, podemos guardar todos estos datos durante todo el tiempo que dura la sesión y disponer de ellos sin tener que guardarlos en una base de datos. Una vez que se realiza la presentación final de los resultados, la sesión se destruye y con ella el objeto *Tweets*. Así, si volvemos al comienzo de la aplicación, no se almacenarán datos de la sesión anterior. Esto nos evita tener que lidiar con bases de datos para almacenar esta información (el acceso a bases de datos desde Java EE se hace a través de un *contexto de persistencia*, lo que es relativamente complejo y muy propenso a errores). Por otra parte, cuando se envían datos desde la interfaz hasta el backing bean que han sido introducidos en una caja de texto, JSF nos permite de forma fácil guardar estos datos dentro de variables de instancia, pero no nos permite procesarlos en ese mismo instante. Así, cuando se introducen los usuarios y los hashtags desde la interfaz, estos se guardan directamente en dos variables de tipo String (*strusers* y *strhashtags*) se parados por comas porque es la única opción. Luego, para no tener que parsear estas cadenas cada vez que queramos acceder a algún usuario o hashtag, se han creado los dos arrays *users* y *hashtags* y así ahorrar tiempo en la ejecución simplemente "sacrificando" algo de memoria para los arrays. Estas razones, entre otras, justifican la redundancia de datos en las variables de instancia. Aun así, debido a la utilización de APIs y librerías externas, hemos tenido que guardar algunos datos en ficheros, ya que varias de estas librerías leen los datos de entrada desde ficheros.

Además de los getters y los setters, esta clase implementa ciertos métodos que definen el comportamiento "global" de la aplicación:

*public String **killSession** ();*

Este método termina la sesión HTTP actual. Se utiliza para forzar el cierre de sesión cuando termina el flujo de datos de la aplicación u ocurre algún error.

*public String **access** ();*

Este método realiza algunas operaciones relacionadas con la autenticación en la API de Twitter.

public String ***processInput*** *();*

Este es el método principal de la clase. Una vez que la aplicación se ha autenticado en la API de Twitter, este método se encarga de crear las instancias necesarias de las demás clases, llamar a los métodos pertinentes y almacenar los datos necesarios para cumplir los requisitos del sistema. Una vez hecho todo esto, nos redirige a la interfaz de presentación de los datos.

public String ***getOutput*** *();*

Este método devuelve una cadena con una representación legible del contexto formal.

public int ***printFile*** *();*

Este método imprime el contexto formal en un fichero con formato .txt y devuelve un identificador único para poder acceder a este posteriormente.

public int ***printTable*** *();*

Este método imprime el contexto formal en forma de tabla en un fichero .csv y devuelve un identificador único para poder acceder a este posteriormente. La creación de este fichero es muy importante debido a que va a ser la única información “permanente” que vamos a manejar en el sistema.

public String[] ***getStrDependencies*** *();*
public String[] ***getStrLCS*** *();*

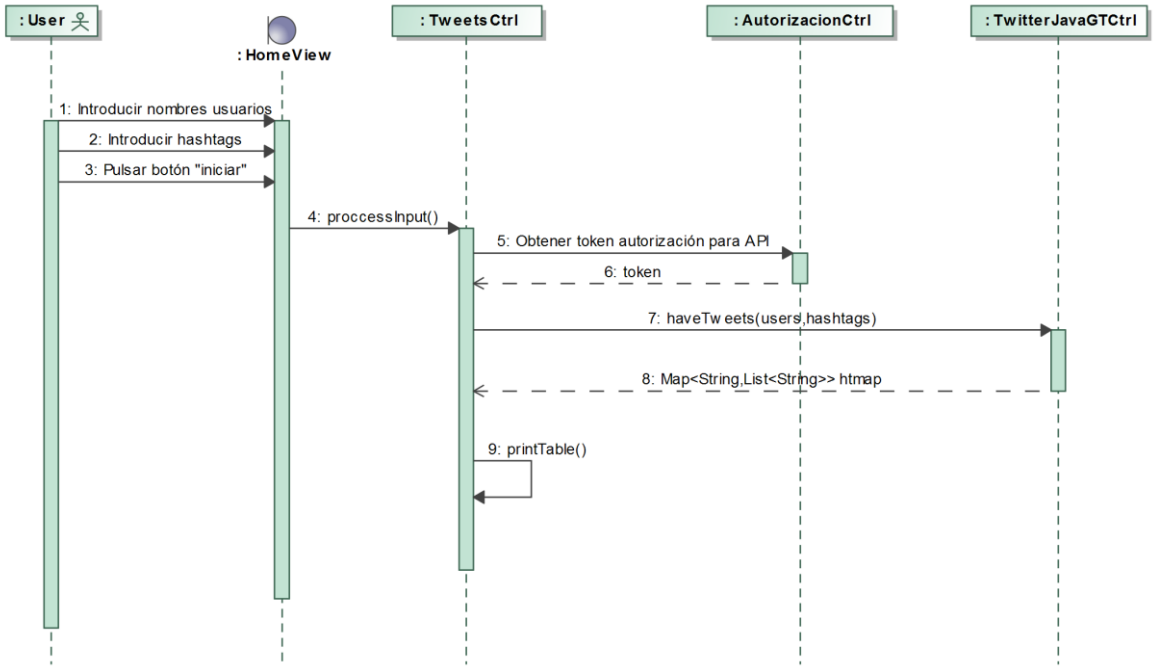
Estos métodos devuelven un array con las implicaciones y los LCS, respectivamente. Son los utilizados por las interfaces para la representación final de los datos. La librería JTLS de Java EE nos permite iterar sobre estos arrays y realizar una presentación algo más elaborada directamente desde la interfaz.

7.3.3. Diagramas de secuencia

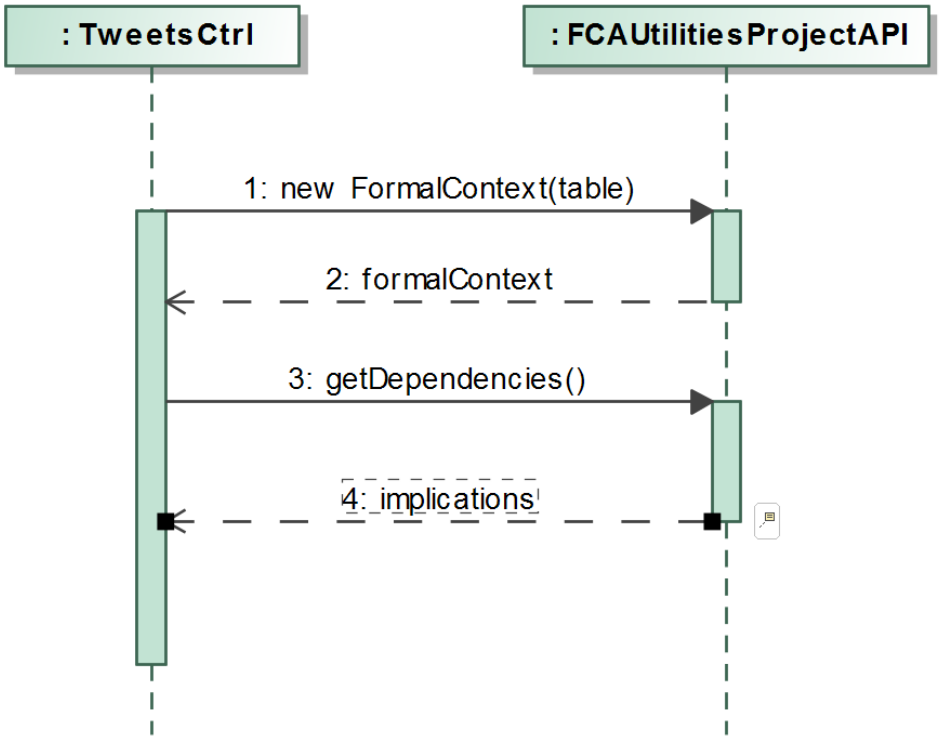
En los siguientes diagramas de secuencia se puede observar el funcionamiento de la aplicación, así como el flujo de datos para cada caso de uso.

Iteración 1

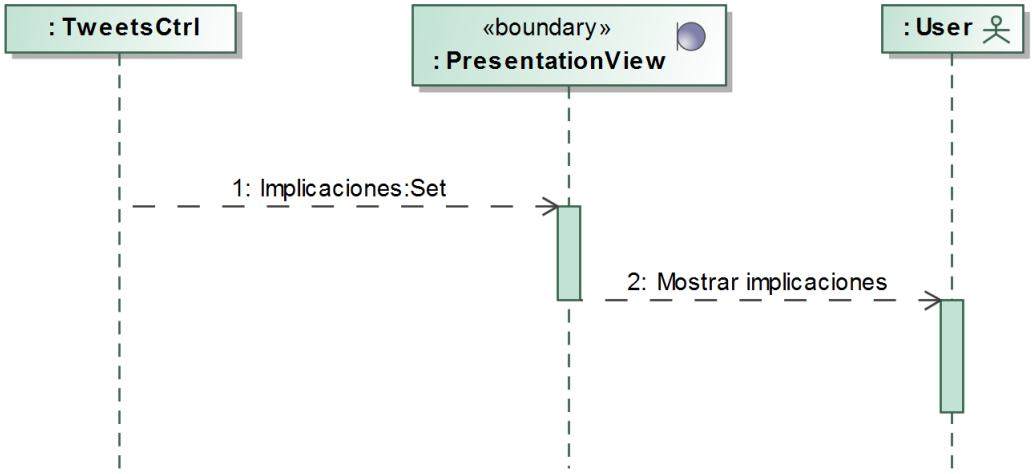
Caso de uso: Obtener los tweets de la fuente de datos:



Caso de uso: Procesar tweets para obtener el Contexto Formal:

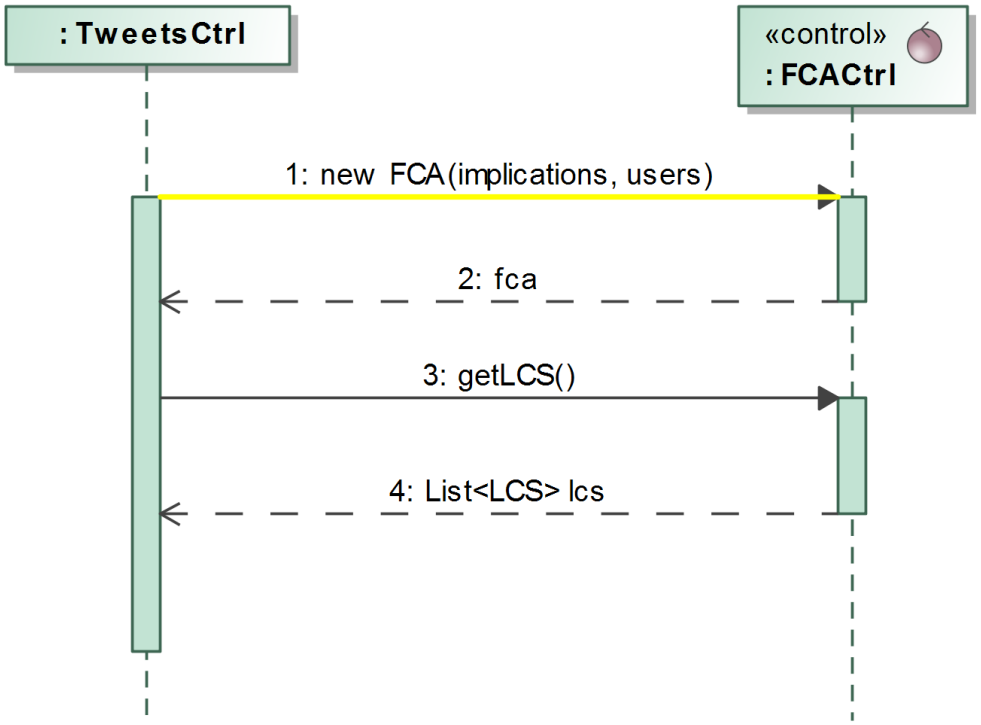


Caso de uso: Presentación de los datos:

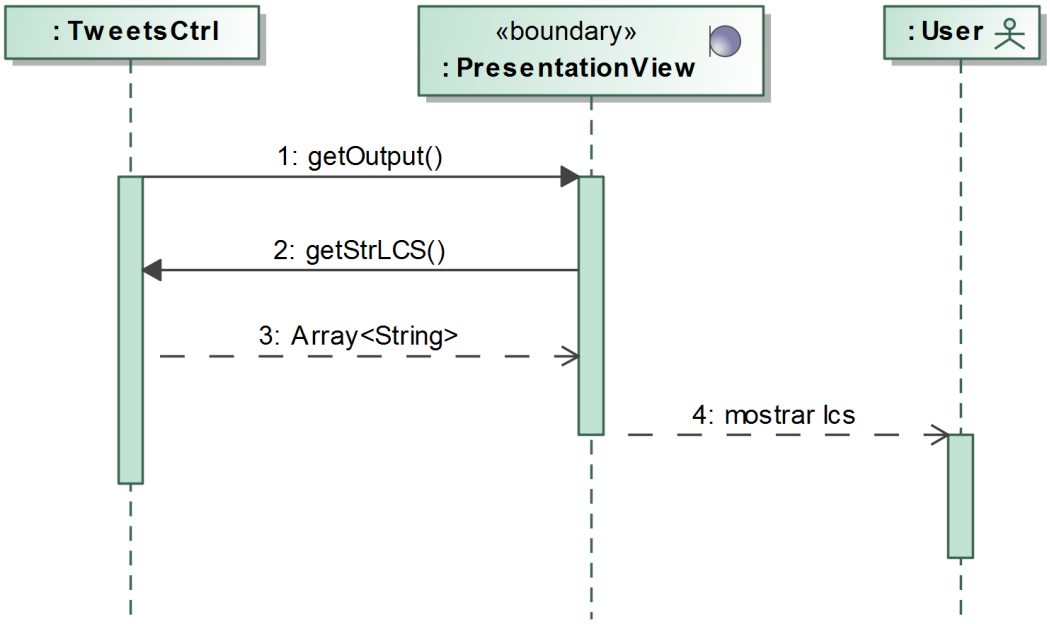


Iteración 2

Caso de uso: Obtener los LCS:

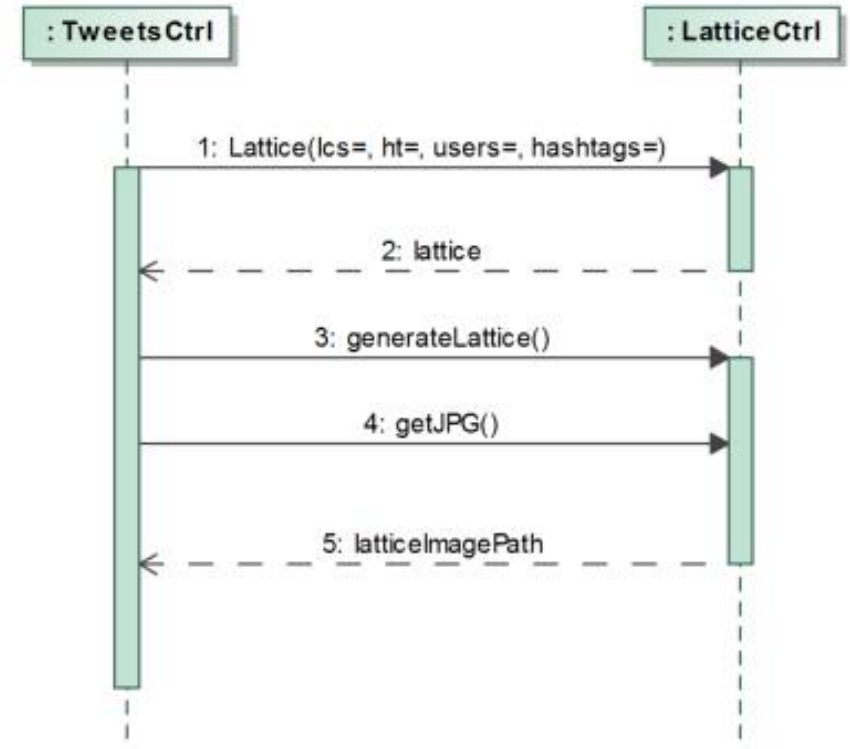


Caso de uso: presentación de los datos:

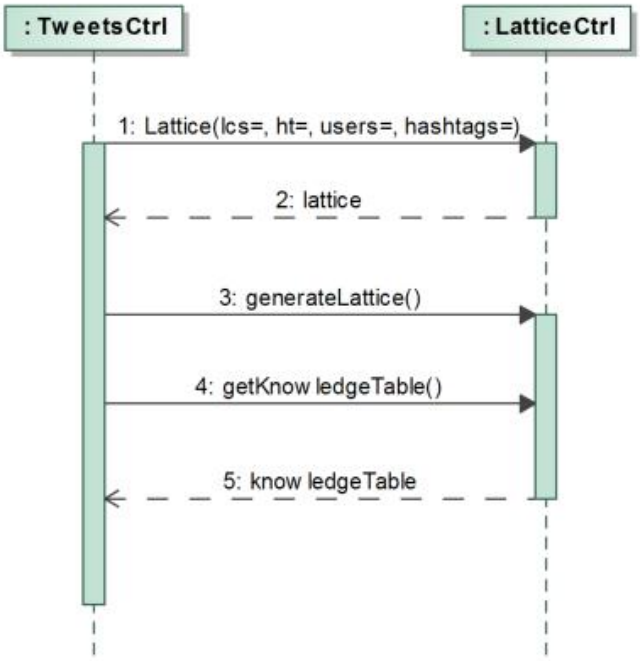


Iteración 3

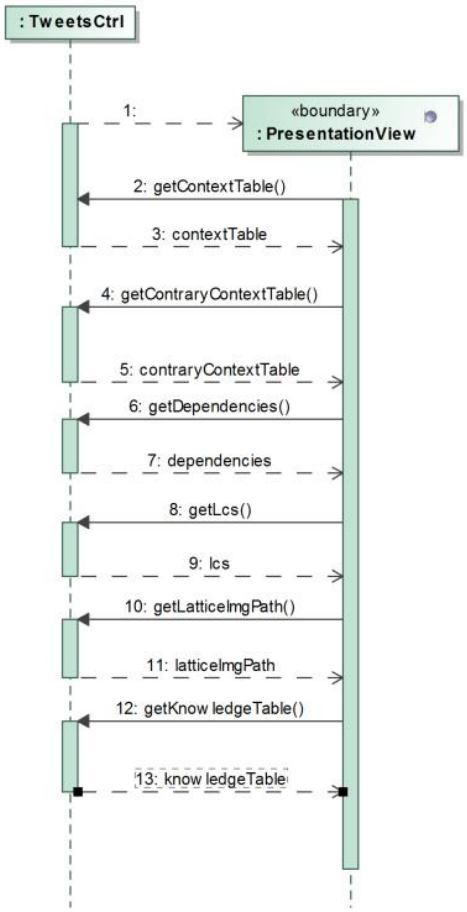
Caso de uso: Obtener Concept Lattice



Caso de uso: Obtener tabla de conceptos:



Caso de uso: Presentación final:



8. Pruebas

Al final de cada iteración del TFG se han realizado algunas pruebas de caja negra con el objetivo de comprobar que la aplicación cubría los requisitos especificados. A continuación se presentan dichas pruebas y los resultados de su realización, ordenadas por iteración:

Iteración 1

IDENTIFICADOR	TFG.P - 01
Descripción	La interfaz envía los datos al controlador correctamente
Componentes Afectadas	User, HomeView, TweetsCtrl
Resultado Esperado	Los datos llegan al controlador
Resultado Obtenido	Los datos llegan al controlador sin ningún problema
Correción Realizada	
Fecha	17/07/2015

IDENTIFICADOR	TFG.P - 02
Descripción	La aplicación se autentica correctamente en la API de Twitter
Componentes Afectadas	TweetsCtrl, AutorizationCtrl
Resultado Esperado	La API de Twitter devuelve el token de autenticación
Resultado Obtenido	La aplicación se autentica correctamente y se obtiene el token.
Correción Realizada	
Fecha	17/07/2015

IDENTIFICADOR	TFG.P - 03
Descripción	La API de Twitter devuelve los tweets requereridos a partir de los datos de entrada
Componentes Afectadas	User, HomeView, TweetsCtrl, AutorizationCtrl
Resultado Esperado	La API devuelve los tweets que hemos consultado con los datos de entrada
Resultado Obtenido	La api devuelve un número limitado de tweets
Correción Realizada	No hay corrección posible, ya que es una limitación de la propia API.
Fecha	17/07/2015

IDENTIFICADOR	TFG.P - 04
Descripción	El método procesarDatos() genera una lista de implicaciones a partir de los tweets
Componentes Afectadas	TweetsCtrl
Resultado Esperado	Un set de implicaciones lógicamente válidas
Resultado Obtenido	Un set de implicaciones lógicamente válidas
Correción Realizada	
Fecha	17/07/2015

Iteración 2

IDENTIFICADOR	TFG.P - 05
Descripción	Generación de los LCS a partir del conjunto de implicaciones
Componentes Afectadas	TweetsCtrl, FCA.java
Resultado Esperado	El algoritmo genera un conjunto de LCS correcto
Resultado Obtenido	El algoritmo genera un conjunto de LCS correcto
Correción Realizada	
Fecha	27/08/2015

Iteración 3

IDENTIFICADOR	TFG.P - 06
Descripción	La aplicación crea el Concept Lattice y lo exporta en formato jpg
Componentes Afectadas	TweetsCtrl, Lattice.java, Graphviz
Resultado Esperado	Se crea un fichero .jpg que incluye una representación del grafo
Resultado Obtenido	Se crea el fichero .jpg con el grafo.
Correción Realizada	
Fecha	18/09/2015

IDENTIFICADOR	TFG.P - 07
Descripción	La aplicación crea la tabla de conceptos
Componentes Afectadas	TweetsCtrl, Lattice.java
Resultado Esperado	Se crea una tabla con dos columnas (Interested in, follow) en la que cada fila es un concepto.
Resultado Obtenido	Se crea la tabla sin problemas.
Correción Realizada	
Fecha	18/09/2015

IDENTIFICADOR	TFG.P - 08
Descripción	La aplicación realiza una presentación sencilla e intuitiva de los resultados
Componentes Afectadas	TweetsCtrl, PresentationView
Resultado Esperado	Se exponen los resultados en una interfaz JSF.
Resultado Obtenido	Se exponen los resultados de forma clara.
Fecha	18/09/2015

9. Interfaces

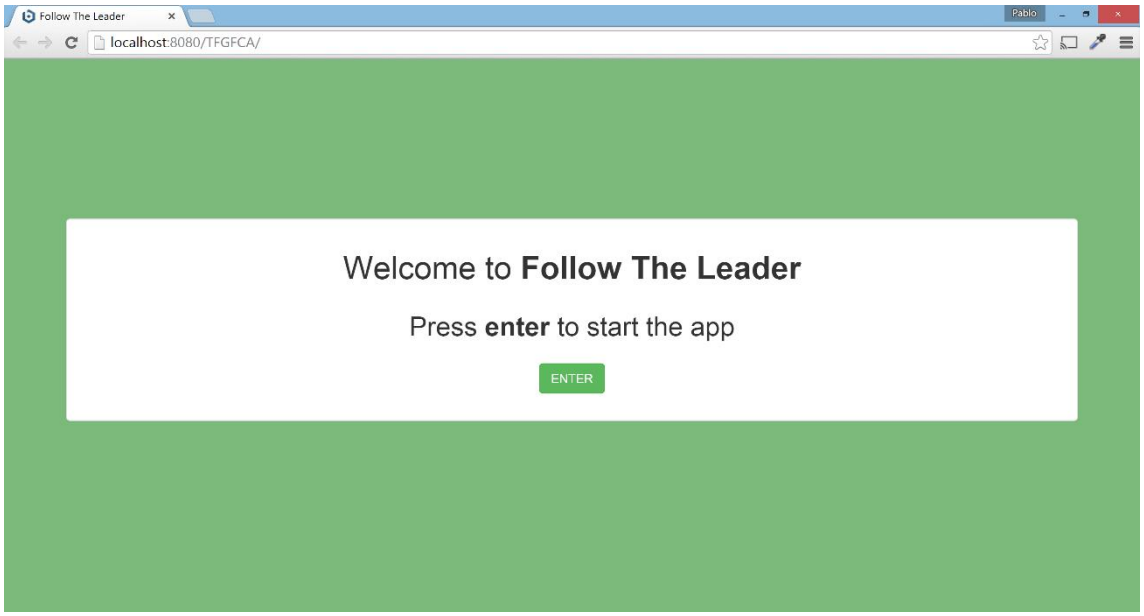
A continuación se muestra una descripción de las interfaces de usuario del sistema, así como su funcionamiento básico.

9.1. Plantilla

JSF nos permite la creación de plantillas que podemos “embeber” en otras interfaces cuando queremos crear patrones comunes. En nuestro caso hemos hecho una plantilla que consta simplemente de una cabecera o header en la que se muestran las siglas de la aplicación y un pie o footer en el que se muestran los autores y el propósito de la ésta.

9.2. Página de inicio

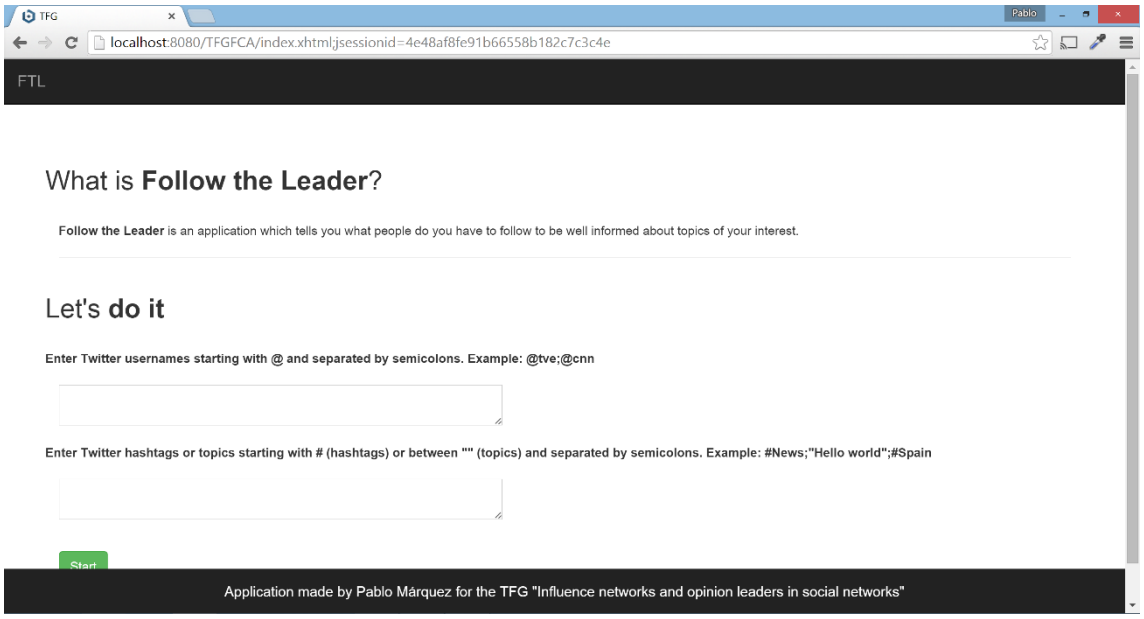
La página de inicio es simplemente una interfaz de bienvenida a la aplicación. Esta, a través del botón “Enter” nos redirige a la interfaz principal de la aplicación:



9.3. Página principal

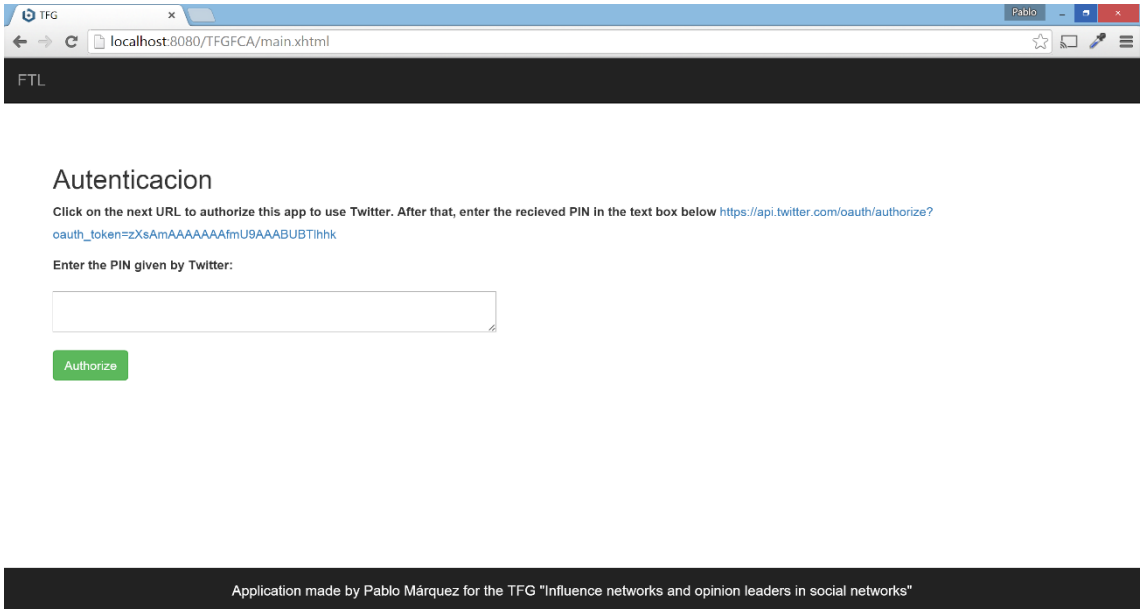
En la página principal nos encontramos con la aplicación ya iniciada. Lo primero que se nos muestra es una breve descripción del funcionamiento de la aplicación. Más abajo hay dos cajas de texto: en la primera introduciremos los usuarios de Twitter que queremos considerar en nuestro contexto y en la segunda los tópicos o hashtags. Encima de cada caja hay una breve explicación del formato que deben tener los datos introducidos. Debajo de estas dos cajas

hay un botón “Start” que nos llevará a la siguiente interfaz del sistema. A continuación se muestra una captura de la interfaz:



9.4. Página de autenticación

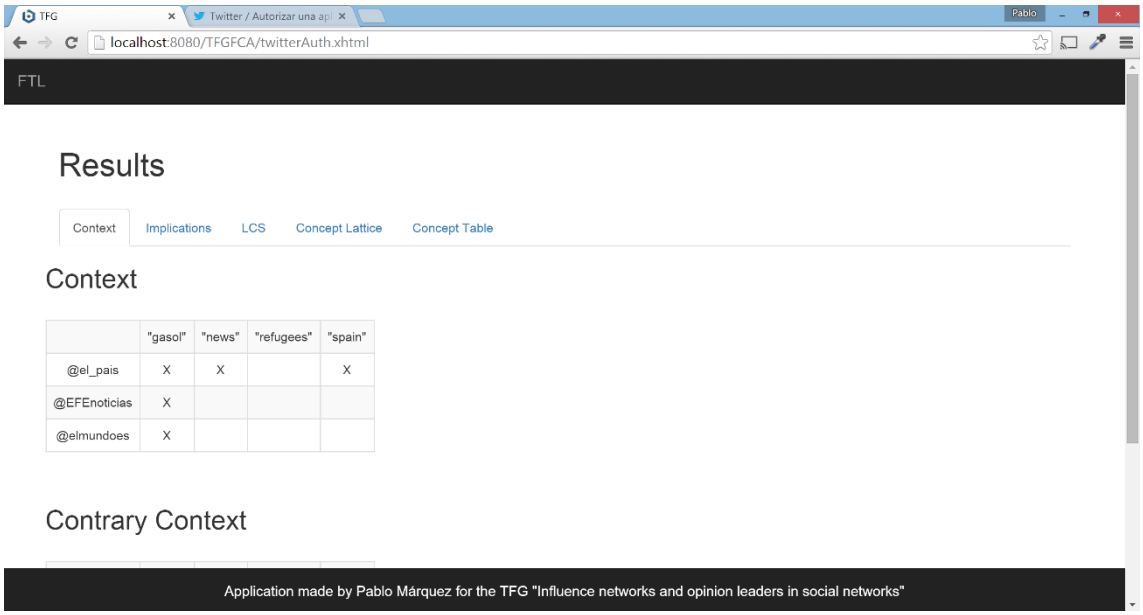
En esta interfaz realizaremos la autenticación con la API de Twitter. Nos aparecerá un enlace y una caja de texto. Si pinchamos en el enlace, se nos abrirá una nueva pestaña en la que tendremos que autorizar a nuestra aplicación para usar Twitter. Una vez autorizada, se nos dará un código numérico que tendremos que introducir en la caja de texto proporcionada en nuestra aplicación para dicho cometido. A continuación se muestra una captura de la página:



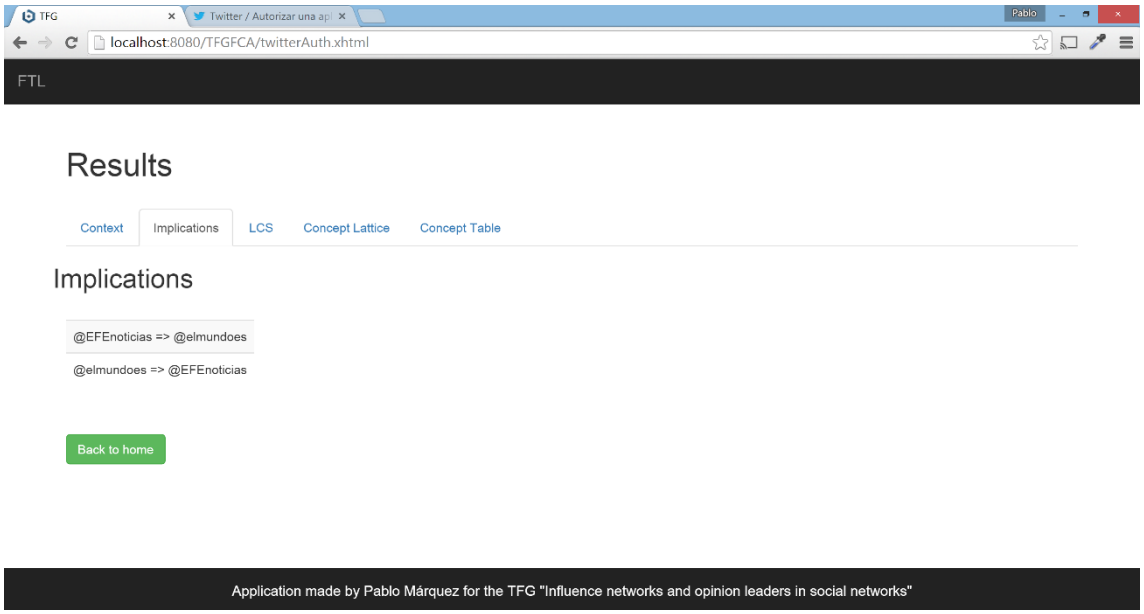
9.5. Página de resultados

En esta página se muestran los resultados obtenidos por nuestra aplicación. Para mostrar los distintos resultados se ha optado por un modelo con “pestañas” a través de las cuales podemos navegar. A continuación se muestran unas capturas de dicha interfaz:

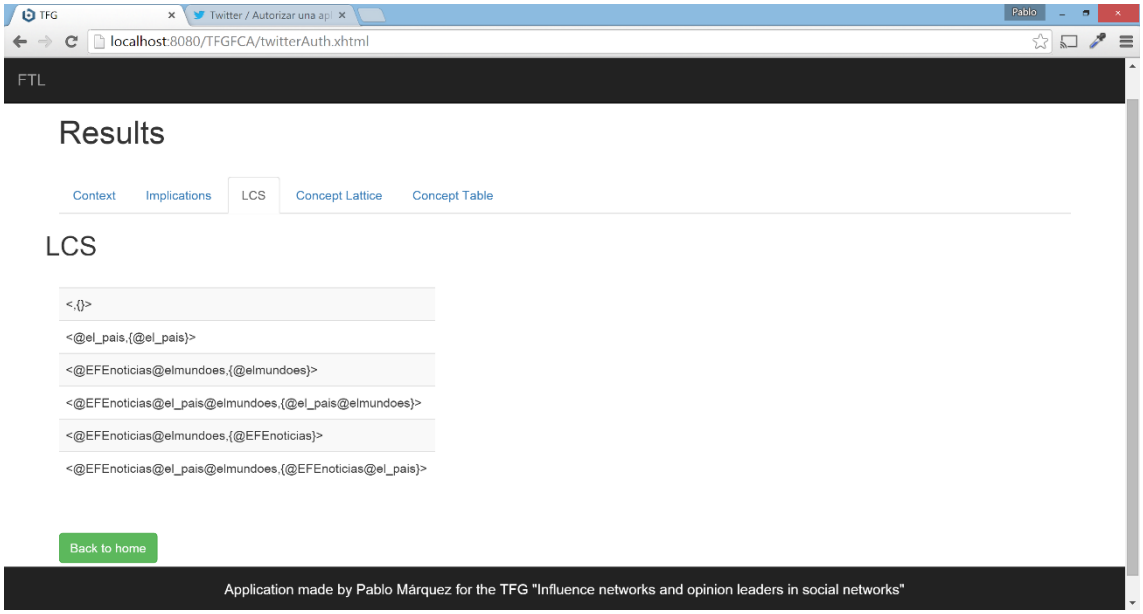
Contexto formal:



Implicaciones:



LCS:



Retículo de conceptos:

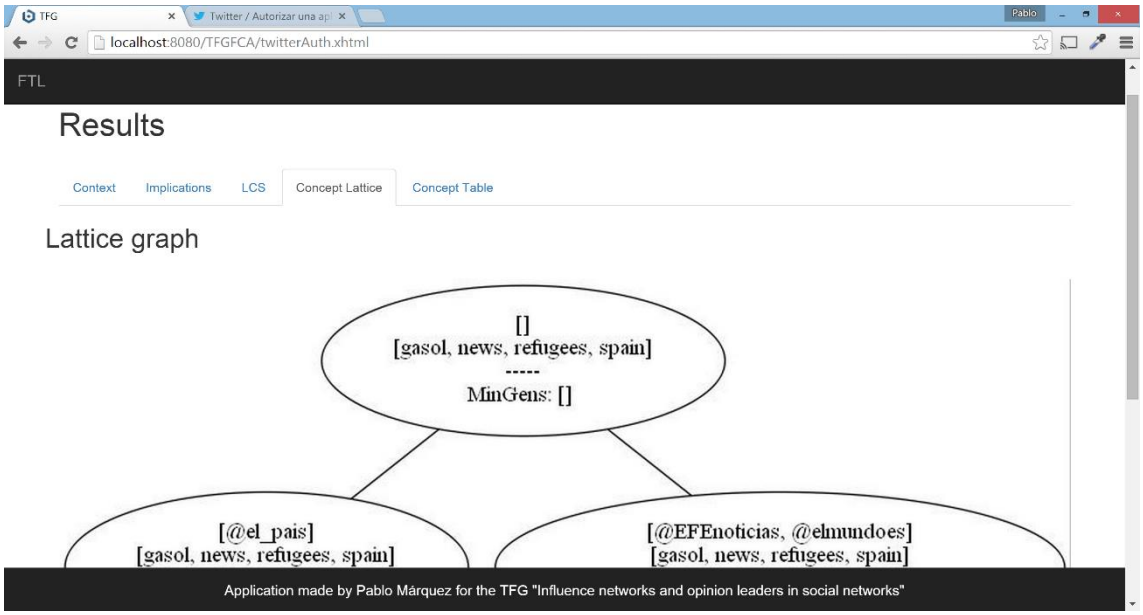
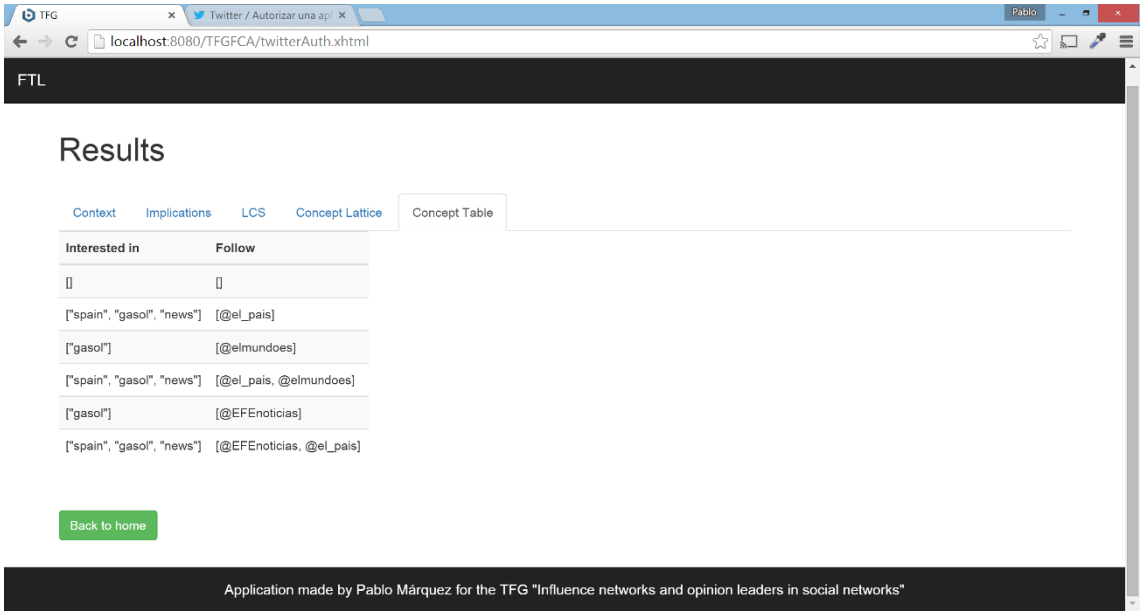


Tabla de conceptos:

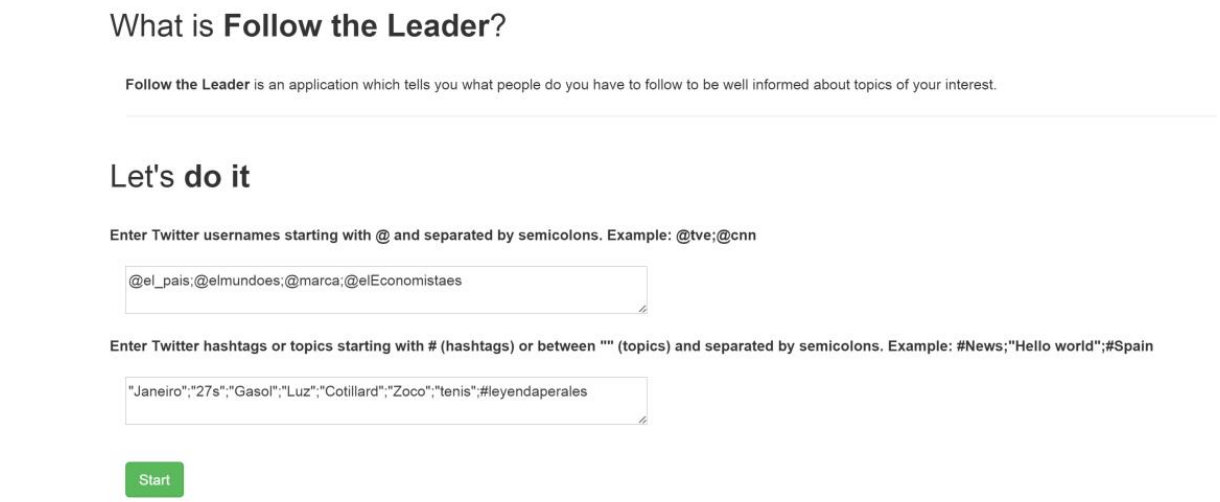


Una vez hayamos terminado de analizar los resultados, el botón “Back to home” nos llevará a la interfaz principal para comenzar de nuevo el proceso.

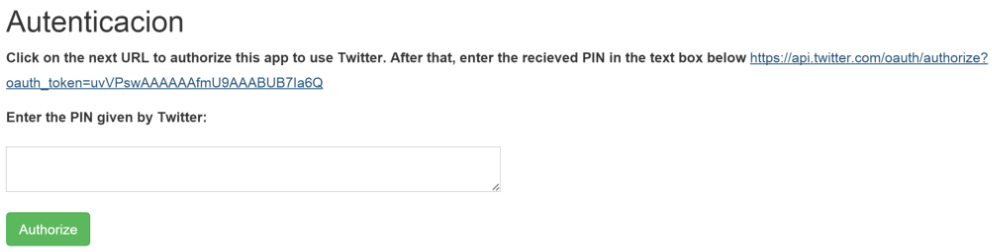
10. Ejemplos prácticos

A continuación se va a exponer un ejemplo completo que explica de forma detallada el funcionamiento de la aplicación.

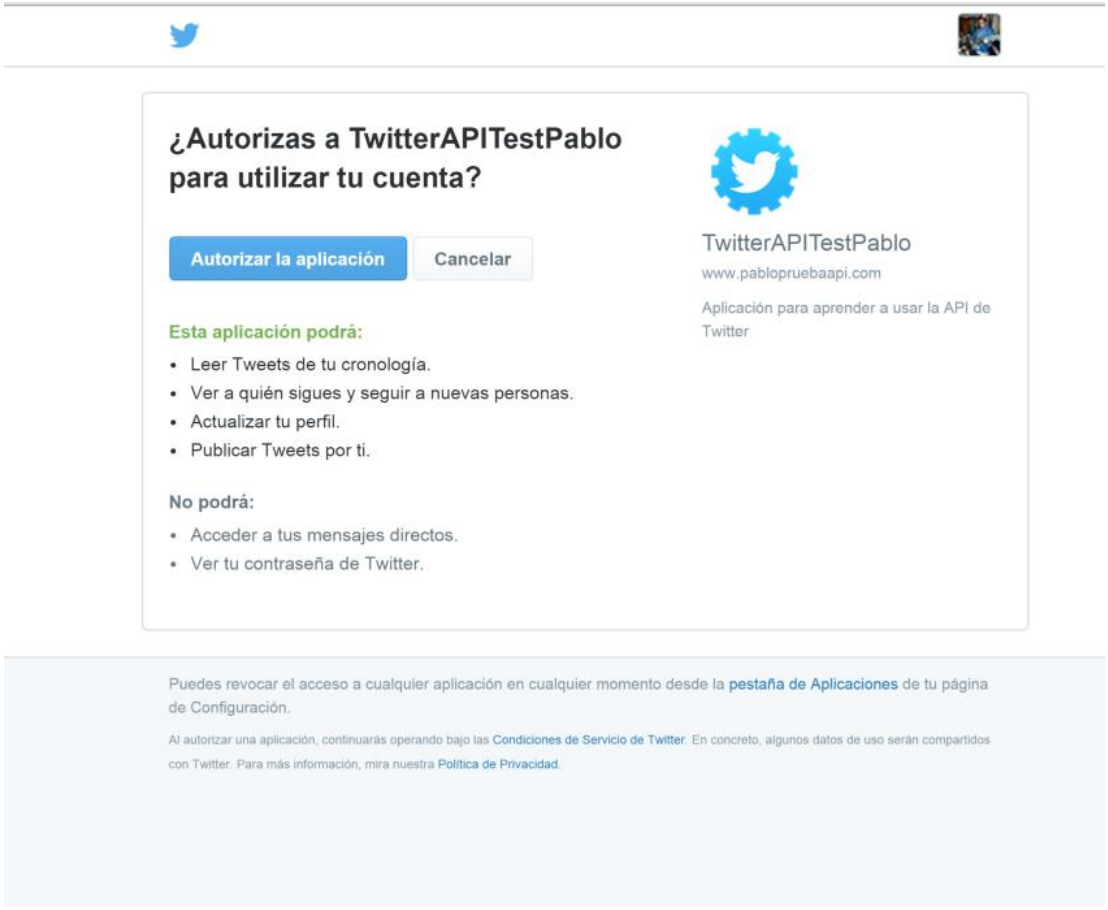
En este caso vamos a analizar un contexto formal de tamaño reducido para que pueda seguirse con facilidad en la memoria y sea ilustrativo. Vamos a tener como datos de entrada una lista de 4 usuarios de Twitter (atributos) y 8 tópicos o temas (objetos) separados por punto y coma. A continuación se muestra una captura en la que se muestra la introducción de estos datos mediante la interfaz principal:



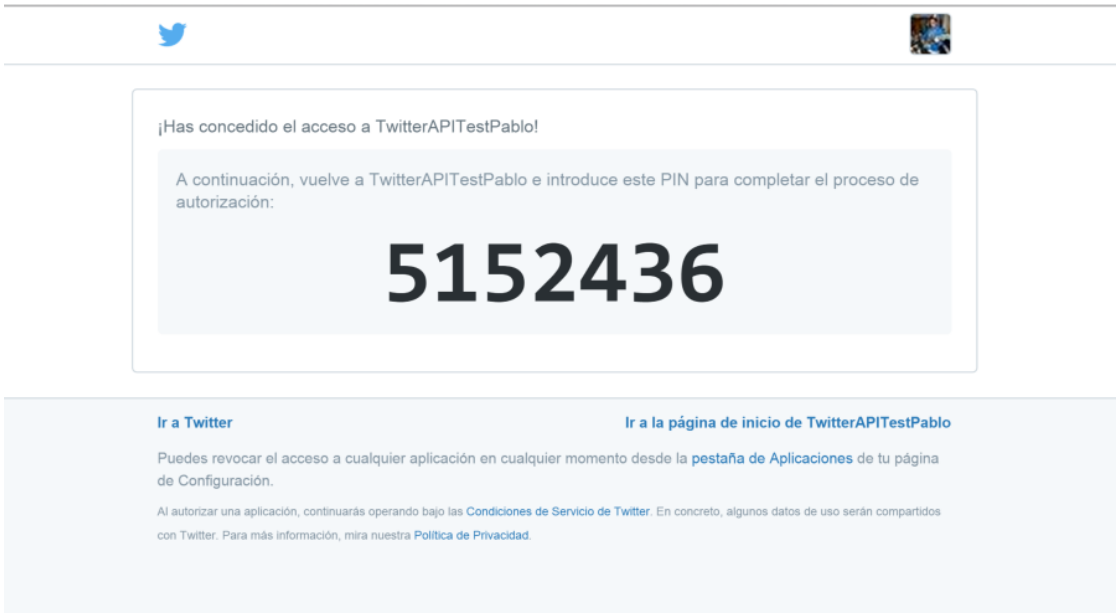
Una vez introducidos estos datos, pulsamos el botón “Start”, el cual nos redirigirá a la interfaz de autorización de twitter. En esta interfaz, nos aparecerá un enlace y una caja de texto. En la siguiente captura se muestra la interfaz:



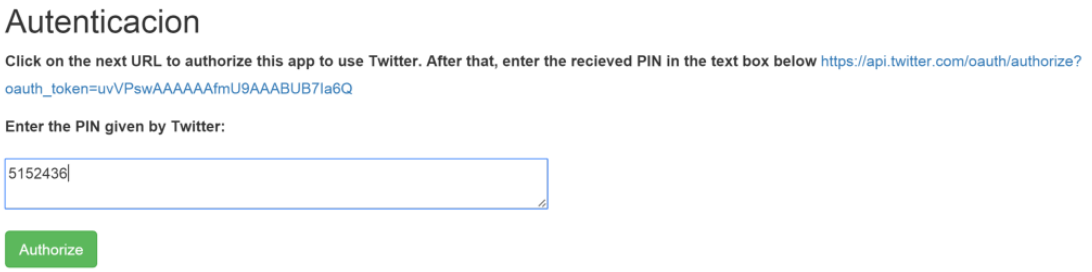
Siguiendo las instrucciones, pinchamos en el enlace y se nos abrirá una nueva pestaña en la que Twitter nos pedirá que autoricemos la aplicación:



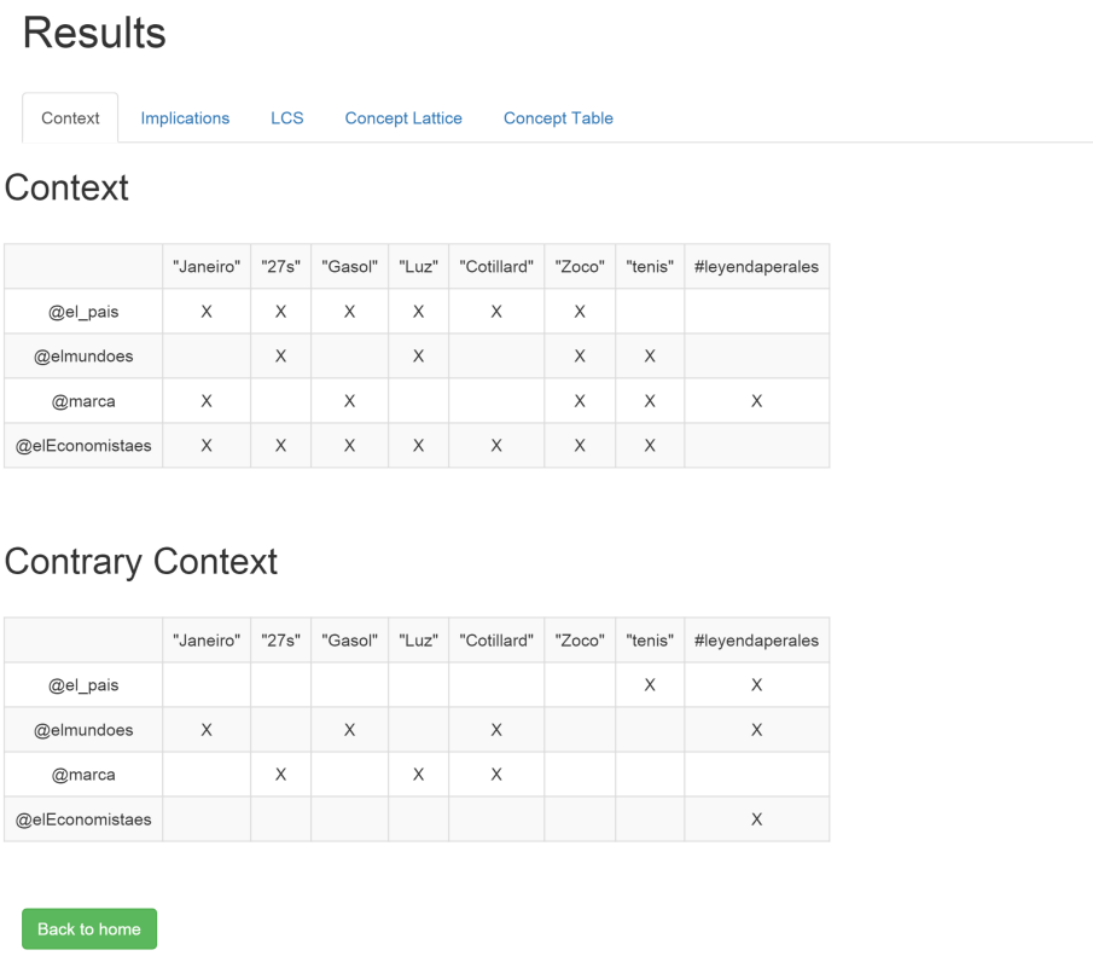
Pulsamos en “Autorizar la aplicación” y nos aparecerá un código numérico como el siguiente:



Este es el código que tenemos que copiar y pegar en la caja de texto de la interfaz de autorización de nuestra aplicación tal que así:



Este proceso es necesario cada vez que se ejecute la aplicación, ya que no hemos diseñado ninguna política de almacenamiento de cookies que nos permita mantenernos autenticados. Una vez hayamos Introducido el código, pulsamos el botón “Authorize” y empezará a cargar una nueva interfaz. En este momento, la aplicación está aplicando todos los algoritmos necesarios para obtener los resultados. Una vez terminado este proceso, nos aparecerá la interfaz de presentación. En primer lugar, se nos muestra el contexto formal que nuestro sistema ha creado a través del análisis de los tweets:

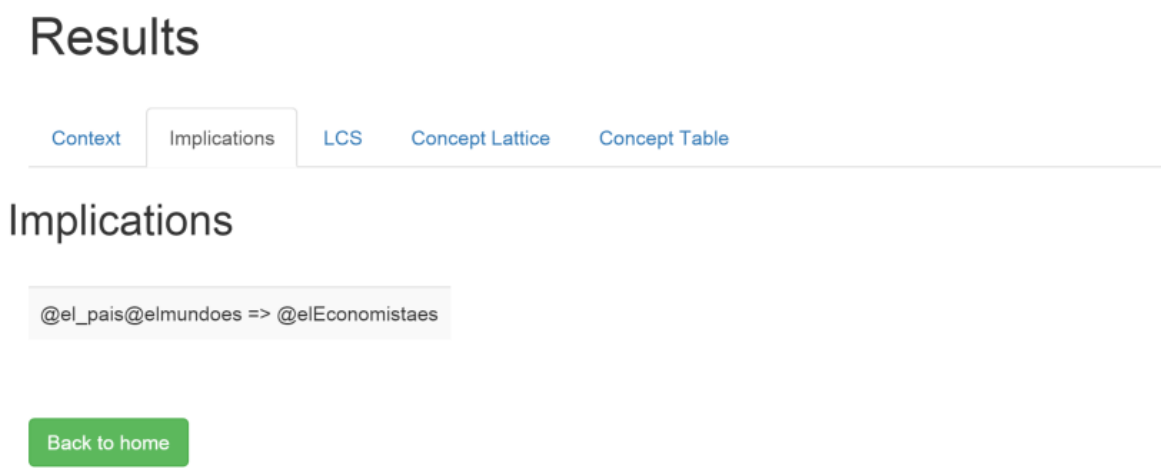


	"Janeiro"	"27s"	"Gasol"	"Luz"	"Cotillard"	"Zoco"	"tenis"	#leyendaperales
@el_pais	X	X	X	X	X	X		
@elmundoes		X		X		X	X	
@marca	X		X			X	X	X
@elEconomistaes	X	X	X	X	X	X	X	

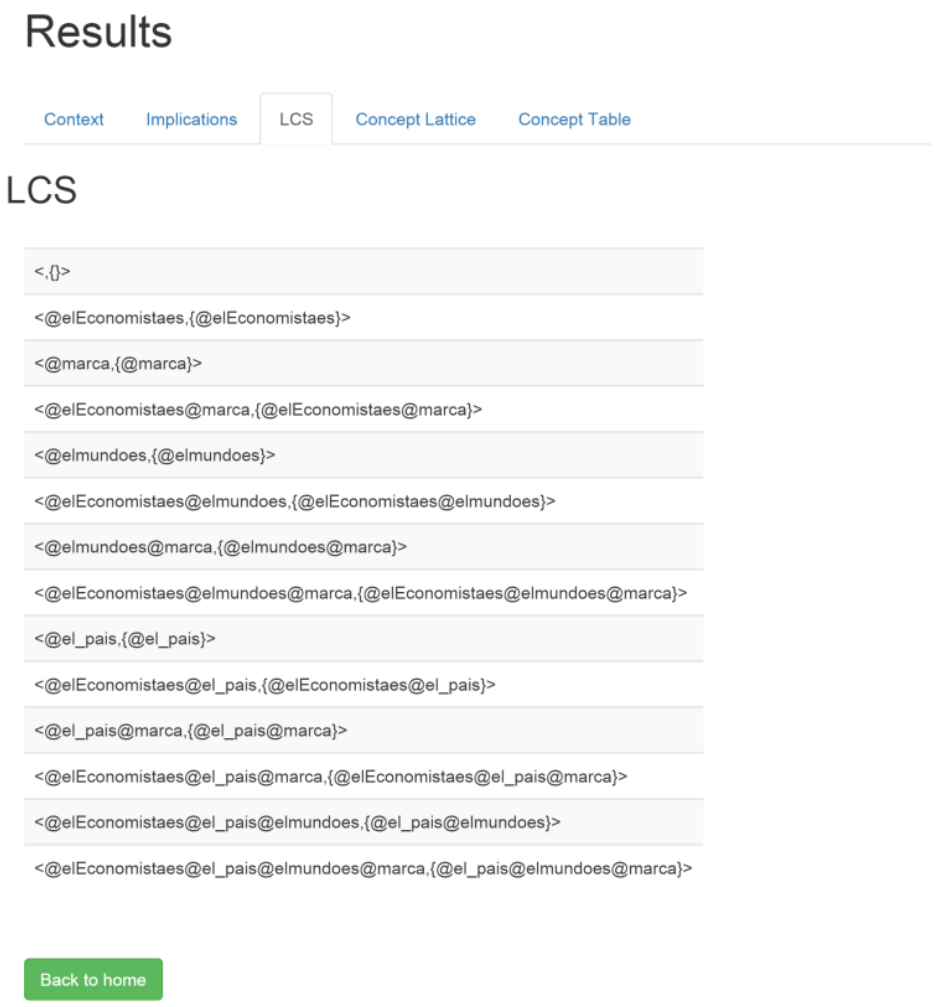
	"Janeiro"	"27s"	"Gasol"	"Luz"	"Cotillard"	"Zoco"	"tenis"	#leyendaperales
@el_pais							X	X
@elmundoes	X		X		X			X
@marca		X		X	X			
@elEconomistaes								X

La primera tabla simboliza el contexto tal cual, es decir, si en la casilla (x,y) existe una cruz, quiere decir que el usuario x habla del tema y. La segunda tabla simboliza el contexto contrario y, por ende, la relación contraria a la

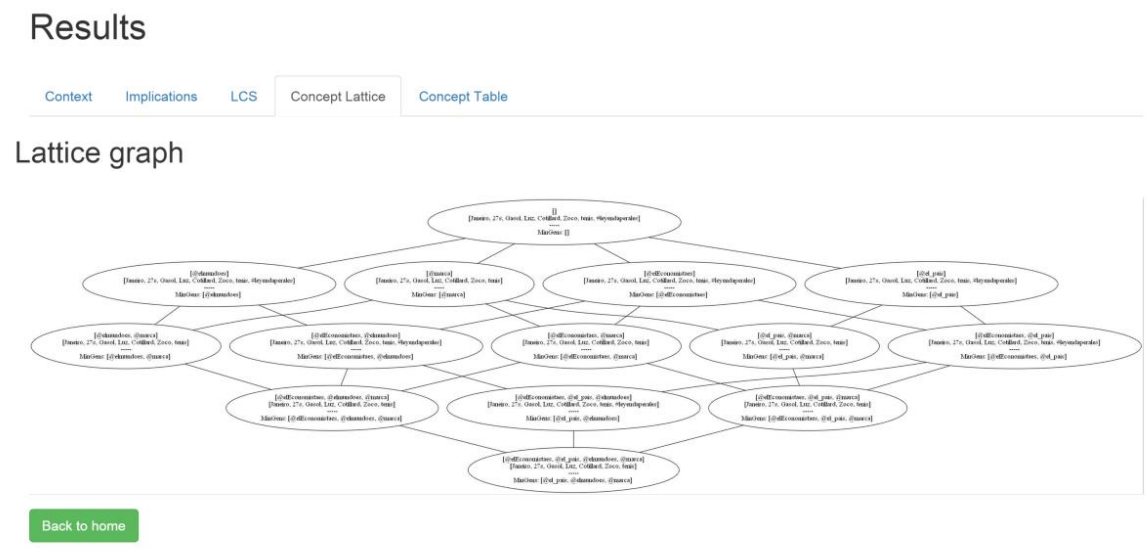
anteriormente descrita. Este es el contexto sobre el que van a trabajar los algoritmos, tal y como viene especificado en el artículo *"Knowledge discovery in social networks by using a logic-based treatment of implications"*, ya que la interpretación de las implicaciones cuadra mucho mejor con el problema que queremos resolver. Si pinchamos en la siguiente pestaña, podremos ver las implicaciones deducidas del contexto formal:



En este caso solo ha encontrado una implicación, ya que es la única dependencia funcional que ha encontrado en la tabla. Esto explica el porqué hay tantos LCS, ya que al haber pocas implicaciones, existen muchos atributos que, combinados entre sí, producen conjuntos cerrados diferentes (los atributos son muy independientes). Por este mismo motivo, el retículo de conceptos sale también muy grande. Si clicamos en la siguiente pestaña, se nos muestran los LCS generados por el algoritmo LabeledClosedSets:



Recordemos que cada LCS se compone de un conjunto cerrado y un generador minimal del mismo. En la siguiente pestaña se nos muestra el grafo que representa el Concept Lattice o Retículo de Conceptos:



Este grafo nos muesta la relación entre los distintos conjuntos cerrados. En cada nodo se incluye un conjunto cerrado, el conjunto de temas que abarca y los generadores minimales. En la siguiente imagen se aprecia mejor la estructura del nodo:

Por último, en la última pestaña se nos muestra la tabla con todos los conceptos generados por el sistema. Como bien se indica en la cabecera de la tabla, para enterarnos de según qué temas estemos interesados, tenemos que seguir a un conjunto u otro de usuarios:

Results

Context | Implications | LCS | Concept Lattice | Concept Table

Interested in	Follow
[]	[]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", "tenis", "Gasol"]	[@elEconomistaes]
["Janeiro", "Zoco", #leyendaperales, "tenis", "Gasol"]	[@marca]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", #leyendaperales, "tenis", "Gasol"]	[@elEconomistaes, @marca]
["27s", "Luz", "Zoco", "tenis"]	[@elmundoes]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", "tenis", "Gasol"]	[@elEconomistaes, @elmundoes]
["27s", "Luz", "Zoco", "Janeiro", #leyendaperales, "tenis", "Gasol"]	[@elmundoes, @marca]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", #leyendaperales, "tenis", "Gasol"]	[@elEconomistaes, @elmundoes, @marca]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", "Gasol"]	[@el_pais]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", "tenis", "Gasol"]	[@elEconomistaes, @el_pais]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", #leyendaperales, "tenis", "Gasol"]	[@el_pais, @marca]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", #leyendaperales, "tenis", "Gasol"]	[@elEconomistaes, @el_pais, @marca]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", "tenis", "Gasol"]	[@el_pais, @elmundoes]
["27s", "Luz", "Janeiro", "Zoco", "Cotillard", #leyendaperales, "tenis", "Gasol"]	[@el_pais, @elmundoes, @marca]

Back to home

Una vez tenemos esta tabla, la interpretación de los conceptos sería la siguiente: por ejemplo, si nos fijamos en la quinta fila, el concepto significaría: “para estar informado de los temas “27S”, “luz”, “Zoco” y “tenis”, tiene que seguir a @elmundoes”.

Como se puede observar, en la tabla no se dan todas las posibilidades. Esto es porque, como se puede observar en el contexto, hay casos en los que no se puede dar una respuesta. Por ejemplo, si solo queremos estar informados sobre el tema “27S” la aplicación no nos da una solución porque ningún usuario habla sólo de ese tema en concreto, es decir, no hay ninguna combinación posible de usuarios que nos de ese concepto. Además, algunas de las filas de la izquierda están repetidas. Esto es debido a que, en ocasiones, para un grupo de temas existen varias combinaciones de usuarios a los que podemos seguir.

11. Conclusiones y trabajos futuros

En este TFG se ha desarrollado un sistema que implementa algunas técnicas innovadoras para la identificación de líderes de opinión en las redes sociales. En esta memoria se ha tratado de exponer de manera resumida la necesidad de estos sistemas y, en lugar de exponer un gran número de teorías, se ha profundizado en la teoría del Análisis de Conceptos Formales y su utilidad para este tipo de sistemas. En lo que respecta al estudio del FCA, fue una grata sorpresa descubrir la simplicidad de esta teoría. Otros aspectos destacables pueden ser su aplicabilidad en múltiples campos de estudio y la cantidad de algoritmos que trabajan sobre ella.

La implementación de este sistema se realizó en varias etapas, pero se debe hacer especial mención a la segunda de ellas, en la que se implementaron los algoritmos basados en FCA que son, en definitiva, el núcleo de la aplicación y la parte más relevante de este trabajo.

El sistema final obtenido ha sido bastante satisfactorio, cumpliéndose todos los requisitos especificados al principio. Además, ha sido una prueba fehaciente de que los algoritmos especificados en el artículo "*Knowledge discovery in social networks by using a logic-based treatment of implications"* (artículo nombrado varias veces en esta memoria y en el cual nos hemos basado para implementar la aplicación) funcionan en diversidad de ámbitos y con un rendimiento espectacular.

Cabe mencionar que los resultados obtenidos por la aplicación no son siempre relevantes. El hecho de coger los datos de las fuentes en tiempo real hace que éstos sean imprevisibles y cambiantes, por lo que se ha llegado a la conclusión de que, para obtener resultados fiables, es necesario un pequeño estudio previo de los datos de entrada así como una reflexión final sobre los resultados para obtener conocimiento verdadero y real.

Finalmente, se presentan una serie de propuestas para futuros trabajos relacionados con éste:

- Aplicar técnicas de minería de datos y análisis de sentimientos para obtener los datos de entrada de otras fuentes.
- Indagar en la teoría de los retículos de conceptos para identificar relaciones más complejas entre los atributos.
- Ampliar este sistema para analizar otro tipo de conceptos, como la influencia de unos usuarios sobre las opiniones de otros en las redes sociales o la fluctuación de dicha influencia según qué aspectos.

12. Bibliografía

1. A. Gupta. *Java EE 7: Essentials.* Editorial O'Reilly. 2013.
2. Bootstrap. http://getbootstrap.com/
3. Concept Explorer. http://sourceforge.net/projects/conexp/
4. D. Heffelfinger. *Java EE 7 with GlassFish 4 Application Server.* Editorial Packt. 2014.
5. E. Stone, S. Pehar, A. Nichol. *MonkeyLearn documentation v0.6.4.* Source: http://www.monkeylearn.com/docs/
6. Flexbox Grid. http://flexboxgrid.com/
7. FCA algorithms (FCALGS). https://code.google.com/p/erca/
8. Graphviz. http://www.graphviz.org/
9. I. Sommerville. *Software Engineering: 8th edition.* Editorial Pearson .2010
10. JGraphT. http://jgrapht.org/
11. JGraph. http://www.jgraph.com/
12. JGraphX https://github.com/jgraph/jgraphx
13. JQuery. http://api.jquery.com/
14. P. Cordero, M. Enciso, A. Mora, M. Ojeda-Aciego, C. Rossi. *Knowledge discovery in social networks by using a logic-based treatment of implications.* 2014.
15. R. Wille. *Formal Concept Analysis as Mathematical Theory of concepts and Concept Hierachies.* 2005.
16. Scrum. https://www.scrum.org/
17. Uta Priss. http://www.upriss.org.uk/fca/fcasoftware.html
18. Xtract https://svn.win.tue.nl/trac/prom/browser/XTract