Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido

Estudiante: Miguel Ángel García Roig
Enginyeria d'Informàtica (segundo ciclo)

Consultor: David Isern Alarcón

Diciembre 2015

Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido

Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido

FICHA DEL TRABAJO FINAL

Título del trabajo:	Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido
Nombre del autor:	Miguel Ángel García Roig
Nombre del consultor:	David Isern Alarcón
Fecha de entrega (mm/aaaa):	22/12/15
Área del Trabajo Final:	Inteligencia Artificial
Titulación:	*Enginyeria d'Informàtica (segundo ciclo)*

Resum del Treball (màxim 250 paraules):

En l'àmbit d'aquest projecte, utilitzarem *Formal Concept Analysis (FCA)* com a tècnica per a poder generar coneixement i establir relacions sobre les dades del nostre domini; K-Means com algoritme de discretització sobre les dades d'entrada (que ens ajudaran a generar un model vàlid susceptible per al processament requerit per la tècnica del *FCA*) i finalment, farem servir un *framework* sobre computació distribuïda basat en l'algoritme *map-reduce (Hadoop)* com a estratègia de processament potencial de grans quantitats de dades. La sortida d'aquest procés serà un graf anomenat *concept lattice*, que pot ser usat per extreure relacions existents entre les dades.

Per a això, utilitzarem com a font de dades la informació sobre els rectors nuclears que es troben desplegats pel món que subministra l'agència internacional sobre energia atòmica (*IAEA*); i que pot ser descarregats lliurement des http://nucleus.iaea.org/RRDB/RR/ReactorSearch.aspx?rf=1.

Amb les dades del *concept lattice* generat, implementarem un navegador interactiu sobre les dades de reactors disponibles.

El navegador, estant visualitzant les dades d'un reactor en concret, presentarà a l'usuari enllaços a altres reactors similars, que comparteixin alguna característica. Els enllaços a reactors amb major grau de similitud, seran presentats d'una forma més rellevant.

Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido

Abstract (in English, 250 words or less):

In the scope of this project, we will use the *Formal Concept Analysis (FCA)* technique to generate knowledge and establish relations on the data of our domain; *K-Means* as the discretization algorithm on the input data (that will help us to generate a valid model to be processed by means of *FCA*) and finally, we will use a framework for distributed computing based on map-reduce algorithm (*Hadoop*) as a potential strategy for processing large amounts of data. The output of this process will be a graph called *concept lattice*, which can be used to extract existing data relationships.

With such objective in mind, we will use as our data source the world nuclear reactor data provided by the International Atomic Energy Agency (*IAEA*); that can be freely downloaded from http://nucleus.iaea.org/RRDB/RR/ReactorSearch.aspx?rf=1.

With the *concept lattice* data as input, we will implement an interactive browser over the available reactors.

The browser, showing a concrete reactor, will present to the user links to other similar reactors, sharing some characteristics. That reactor links with more similitude will be highlighted in a more relevant way.

Mapa interactivo automático de conceptos: Combinación de Big Data y FCA distribuido

Palabras clave (entre 4 y 8):
FCA, K-Means, Concept Lattice, Formal Context, Hadoop, Map-Reduce, Mahout, Java, JEE

Índice de contenido

Índice de tablas

Índice de ilustraciones

1. Introducción

Durante los últimos años, diversas aplicaciones sobre la adquisición de conocimiento automático o semiautomático están cobrando gran relevancia, tanto en fuentes de datos estructuradas cómo -cada vez más- en fuentes de datos no estructuradas; abandonando los círculos más académicos e incorporándose paulatinamente a la realidad de muchas empresas e instituciones; habitualmente sobre una cantidad cada vez mayor de los datos sobre los que extraer este conocimiento.

Estas técnicas ya se están usando en sistemas de información tan cotidianos como sistemas de recomendación de compra en grandes sitios de e-comercio o los sistemas de digitalización y catálogo de los grandes museos, por ejemplo.

Concretamente, en el ámbito de este proyecto, utilizaremos *Formal Concept Analysis* (*FCA*) como técnica para poder generar conocimiento y establecer relaciones sobre los datos de nuestro dominio; *K-Means* como algoritmo de discretización sobre los datos de entrada (que nos ayudarán a generar un modelo válido susceptible para el procesamiento requerido para la técnica del *FCA*) ; finalmente, usaremos un *framework* sobre computación distribuida basado en el algoritmo *map-reduce* como estrategia de procesado potencial de grandes cantidades de datos.

Cabe decir que, aunque la fuente de datos con la que trataremos no es de una magnitud especialmente grande[1], nos interesa poder profundizar en el conocimiento para el desarrollo de soluciones capaces de procesar grandes cantidades de datos. El cálculo del algoritmo FCA no es especialmente complicado, aunque sí costoso en términos de computación. Cómo puede verse en [8], dependiendo de la implementación, el **coste** de la construcción de un *concept lattice* puede llegar a ser **exponencial**; de ahí que conseguir realizarlo en paralelo y de forma distribuida es de especial relevancia en aras de potenciales aplicaciones en entornos *big data*, que como hemos comentado, suele ser el escenario cada vez más habitual en el día a día de la parte de procesamiento analítico de cada vez más empresas e instituciones.

El objetivo del presente proyecto final de carrera será por tanto:

1. Profundizar en el aprendizaje del algoritmo *FCA*, tanto a nivel teórico cómo a nivel técnico.
2. Conseguir una implementación eficiente del algoritmo en términos de tiempo de computación capaz de ser ejecutada sobre volúmenes medio-altos de datos.
3. Aplicar de forma adecuada técnicas de preparación de datos, transformación de valores y discretización (*K-Means* en nuestro caso) en la capa *ETL* en un problema concreto.
4. Aplicar todas esta serie de técnicas de IA para resolver problemas reales potenciales a los que se enfrentan empresas e instituciones actualmente.

Me gustaría destacar que, desde el punto de vista de la extracción de conocimiento, uno de los usos más popular del algoritmo FCA vendría encuadrado en el *ontology matching,* esto es, modelar conceptos de semejanza o proximidad entre instancias de objetos a priori desconocidos en una jerarquía que nos permita recorrer los individuos de nuestra población basándonos en estas relaciones que se establecen.

Ello (obviamente cambiando las fuentes de datos utilizadas) no nos podría permitir resolver cuestiones tales como analizar la evolución sobre el precio de los pisos de alquiler en una región; pero si otras como, identificado un piso de alquiler adecuado para una persona, poder encontrar pisos similares -por múltiples criterios como: precio, tamaño, localización, orientación, tipo de vecindario, condiciones de arrendamiento- de forma automática

1 Tampoco tenemos acceso a un clúster de computación para poder enfrentarnos a volúmenes de datos medios-altos en el contexto del desarrollo del presente PFC

1.1 Formal Concept Analysis

Una de estas técnicas punteras que han alcanzado una gran popularidad en los últimos tiempos ha sido el "*Formal Concept Analysis*", *FCA* a partir de este momento. *FCA* es una técnica basada en derivar una jerarquía de conceptos en una ontología formal desde una colección de objetos y sus propiedades.

Cada concepto en la jerarquía representa un conjunto de objetos compartiendo los mismos valores para un cierto conjunto de propiedades y cada subconcepto en la jerarquía contiene un subconjunto de los objetos por encima de él. De esta forma, la navegación por objetos similares es muy intuitiva, basada en las relaciones que se establecen entre unos conceptos y otros.

Al conjunto de objetos, junto con las propiedades de estudio, así como la función binaria I de pertenencia, que indica si para un objeto aplica una determinada propiedad, es llamada *Formal Context*. A continuación presentamos un representación gráfica extraída de [3].

Objetos\Atributos[2]	Compuesto (c)	Par (e)	Impar (o)	Primo (p)	Cuadrado (s)
1	No	No	**Sí**	No	**Sí**
2	No	**Sí**	No	**Sí**	No
3	No	No	**Sí**	**Sí**	No
4	**Sí**	**Sí**	No	No	**Sí**
5	No	No	**Sí**	**Sí**	No
6	**Sí**	**Sí**	No	No	No
7	No	No	**Sí**	**Sí**	No
8	**Sí**	**Sí**	No	No	No
9	**Sí**	No	**Sí**	No	**Sí**
10	**Sí**	**Sí**	No	No	No

Tabla 1: Formal context -representación tabular-

En la tabla superior puede observarse la definición del contexto, en el que tenemos 10 objetos (representados por filas en la tabla) correspondientes a números enteros en este dominio de problema en concreto. Así mismo, las columnas corresponden a las propiedades, que en el dominio de este ejemplo sería: Si el número se puede obtener adicionando otros, si es par, si es impar, si es primo o si su raíz cuadrada es un número a su vez entero. A su vez, la función de pertenencia vendría representada mediante el valor lógico de verdadero o falso; en el sentido que si un objeto *i* tiene asociada la propiedad *j*, en la tabla la intersección *i,j* contiene el valor "Sí". En la tabla, podemos ver que el número entero "2" contiene las propiedades "par" y "primo".

Entonces, el algoritmo *FCA*, a partir de un contexto como el mostrado, construye una jerarquía que facilita la extracción de conocimiento respecto a la similitud de cada objeto respecto a los demás. Este grafo es denominado "*Concept Lattice*":

2 Del inglés: ***c****omposite,* ***e****ven,* ***o****dd,* ***p****rime* y ***s****quare*

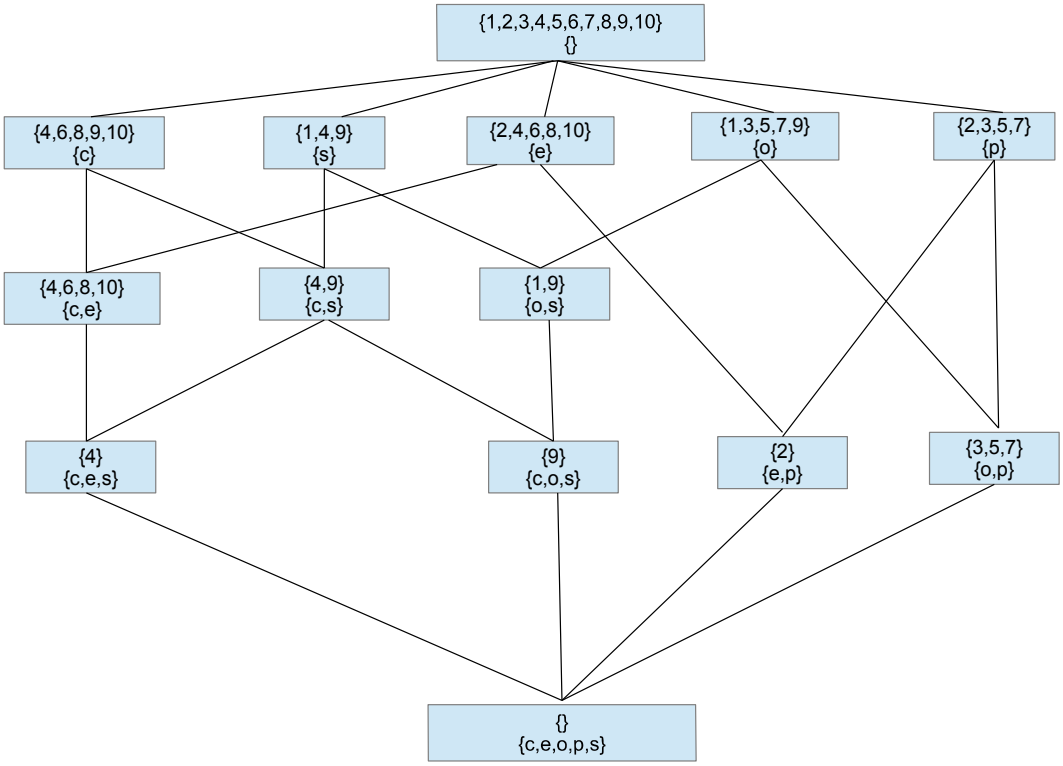

Ilustración 1: Concept lattice

Como podemos observar en el *concept lattice* superior, tan sólo los enteros 1 y 9 contienen las propiedades "impar" y "raíz cuadrada". Así mismo este concepto "hereda" de otros dos, en el que los objetos pertenecientes amplían el suyo propio ({1,**4**,9} y {1,**3,5,7**,9}) y el conjunto de propiedades no obstante en un subconjunto del propio ({s} y {p} respectivamente). Por otra parte sus descendientes (el concepto [{9},{c,o,s}]) tienen como objetos un subconjunto de objetos de sí mismo ({9}); pero amplían el número de propiedades ({**c**,o,s}).

Respecto al *concept lattice*, sólo añadiremos que siempre se incluyen dos nodos especiales:

- Nodo máximo. Agrupa todos los objetos ([{1,2,3,4,5,6,7,8,9,10},{}] en nuestro ejemplo).
- Nodo mínimo. Agrupa todas las propiedades ([{},{c,e,o,p,s}] en nuestro ejemplo)

Algunos de los usos del algoritmo *FCA*, que se mencionan en [1] son :

- **Ontology alignment**: Minería de datos orientada a la correspondencia entre conceptos
- **Ontology matching**: Encuentra correspondencias entre objetos
- **Similarity reasoning**: Identificación de conceptos diferentes sintácticamente que son semánticamente cercanos (obtención e integración de información en la web semántica).

En el contexto de este proyecto final de carrera, usaremos una aproximación cercana al *ontology matching*, muy utilizada en procesos de digitalización de catálogos de algunos museos, como puede verse en [4].

Cabe reseñar que se ha definido una extensión al *FCA* clásico para añadir razonamiento basado en

lógica difusa, *Fuzzy Fornal Concept Analysis* (*FFCA* en adelante). *FFCA* es una generalización de *FCA*, en el sentido que la función de pertenencia no es binaria; devuelve un valor entre entre 0 y 1, en el que 0 significa que "el objeto no tiene la propiedad en absoluto"; 1 significa que "el objeto tiene la propiedad totalmente"; y el resto de valores indican el grado de pertenencia de la propiedad.

FCCA quedará fuera del ámbito en este proyecto, debido a la complejidad que implica implementarlo en un contexto de procesamiento de grandes cantidades de datos en un entorno distribuido; y nos centraremos en implementar la versión clásica, pero adaptada a un algoritmo del tipo *map-reduce* [2].

Nótese que *FCA* requiere que las propiedades sean binarias (0 ó 1). Ello requerirá en la mayoría de los casos aplicar procesos de normalización, transformación y discretización de los datos de entrada.

1.2 Discretización basada en el algoritmo K-Means

La discretización es un proceso que consiste en establecer un criterio por medio del cual se puedan dividir los valores de un atributo en dos o mas conjuntos disjuntos.

Tal como hemos comentado anteriormente, *FCA* necesita definir propiedades binarias (0 ó 1); con lo cual si necesitamos construir un modelo basado en atributos no discretos, será necesario un proceso previo de *discretización* o *agrupación* sobre el dominio de estos atributos. Entonces, para cada grupo generado en el proceso de discretización, generaremos un atributo binario en el *Formal Context*. Este atributo binario tomará el valor de '1' si la instancia contiene un valor dentro de la agrupación correspondiente; '0' en caso contrario.

En nuestro caso, hemos seleccionado el algoritmo *K-Means* dado que se trata de un método no supervisado: Es decir, no necesita información sobre el modelo sobre el cual estamos trabajando; solamente utilizamos la información proveniente de las observaciones que estamos analizando. Esto es de vital importancia, ya que no disponemos de conocimiento de negocio sobre el dominio de nuestro problema (datos técnicos sobre reactores nucleares).

Concretamente, el método *K-Means* afronta el problema de la discretización o reducción de valores para una variable repartiendo los valores entre los diversos intervalos de manera que, dados *k* intervalos, se minimice la distancia entre cada valor y el valor medio del intervalo correspondiente. A efectos prácticos, lo que realiza este algoritmo es generar diversos grupos o clústeres; e ir fusionando los más próximos.

Para el concepto de distancia, diversas funciones pueden ser usadas. En concreto, la denominada distancia de *Minkowski* se considera base para otras funciones de distancia.

Dados dos puntos $P=(x_{1,}x_{2,}..,x_n), Q=(y_{1,}y_{2,}..,y_n)\in R^n$ se define como $(\sum_{i=1}^{n}|x_i-y_i|^p)^{1/p}$.

Particularizando, si elegimos $p=2$ tenemos la definición de la distancia euclidiana, que suele ser la medida más habitual:

$$Dist(x1,x2)=\sqrt{(\sum(x1-x2)^2)}$$

En este proyecto nos hemos decantado por la distancia euclidiana, pues es muy eficiente y viene implementada en numerosos paquetes de minería de datos. Además, dado que nuestro conocimiento de negocio respecto al dominio del problema es reducido, y aplicaremos la discretización sobre diferentes dominios numéricos de valores; es más prudente decantarnos por la función de medida más usada para implementar el algoritmo *K-Means*. Además, existen numerosos ejemplos en la literatura de aplicación de la distancia euclidiana conjuntamente con *K-Means*, siempre que la dispersión de valores no sea muy elevada, como será nuestro caso.

Continuando con la discretización en sí, el método de *K-Means* consiste en comparar cada valor x_j con el valor medio de los intervalos adyacentes, $Intervalo_i$ que es el que ocupa en un momento dado, y el siguiente $Intervalo_{i+1}$:

- Si la distancia al valor medio de su intervalo $\bar{X}_i$ es más pequeña que respecto al siguiente $\bar{X}_{i+1}$, el valor permanece en su intervalo habitual.
- En caso contrario, pasa al intervalo siguiente.

Nos hemos decantado por *K-Means* debido a su abundante documentación, pero sobre todo por su disponibilidad en paquetes de uso habitual en minería de datos, como *Weka* o *Apache Mahout*; con lo cual se simplifica el proceso de desarrollo de los procesos de carga y transformación de los datos (*ETL*). Hablaremos más en detalle de este algoritmo en la siguiente sección.

1.3 Procesamiento distribuido

Hoy en día en el ámbito de la explotación de la información de organizaciones y empresas en diferentes ámbitos, se hace necesario el procesado de grandes cantidades de datos como parte de la operativa fundamental que permita extraer conocimiento tanto de fuentes de datos propias, como de terceros; con el objetivo de saber adaptar y alinear los objetivos de la organización respecto a un mundo en constante evolución y que genera una ingente cantidad de información, pudiendo constituir un recurso fundamental que marque la diferencia respecto a otros competidores.

Como respuesta tecnológica a la necesidad de procesamiento de grandes cantidades de datos de manera eficiente, nace *Apache Hadoop*. *Hadoop* es un *framework* de desarrollo de software para el procesado distribuido de ingentes cantidades de datos (del orden de *petabytes*), así como su almacenamiento, también de forma distribuida, sobre clústeres de computadores usando modelos de programación simples. Información más detallada sobre *Apache Hadoop* puede encontrarse en el proyecto final de carrera "*OpenNebula y Hadoop: Cloud Computing con herramientas Open Source*" de Francisco Magaz Villaverde, de Junio de 2012.

Así mismo, siendo *Hadoop* un *framework* de procesamiento distribuido de carácter general, *Apache Mahout* nace implementado sobre él como una serie de algoritmos de inteligencia artificial y aprendizaje automático, que podrán ser aplicados entonces sobre cantidades muy grandes de datos. En el ámbito de este PFC estaremos interesados especialmente en algoritmos de agrupación o *clustering* que usaremos para implementar los procesos ETL, concretamente y como hemos comentado anteriormente, *K-Means*.

Nos decantamos por el uso de *Mahout* debido a que se integra sobre el mismo clúster de *Hadoop*, con lo cual no sería necesario mover los datos de entrada del mismo clúster, para poder ejecutar los procesos de *clustering*.

2. Descripción de la propuesta de trabajo

Cabe decir que, para poder construir un *concept lattice* sobre el que realizar un análisis FCA, necesitaremos construir primero un *formal context,* que no es más que una matriz de objetos y propiedades binarias que tienen asociada una función de pertenencia sobre ese conjunto de atributos.

Será el objetivo de la capa *ETL* poder llegar a construir un modelo que cumpla estas características (es decir, un listado de objetos, un listado de atributos binarios, y asignar a cada uno de los objetos los atributos que "posee").

Para construir los modelos requeridos, nos basaremos en la construcción a partir de datos tabulares en formato *Microsoft Excel*, por ser de uso común; y poder encontrar abundantes datos de fuentes

abiertas publicados en este formato.

Concretamente, los datos en los que basaremos este estudio en concreto, proceden de los datos que la agencia mundial sobre energía atómica (a partir de este momento IAEA - https://www.iaea.org/) pone a disposición del público datos sobre los reactores nucleares que están activos, han estado activos o cuya puesta en marcha está planeada, tanto a través del documento "*Reference data series*" (*http://www-pub.iaea.org/MTCD/Publications/PDF/rds-2-34_web.pdf*) como de forma interactiva a través de su plataforma web (*http://nucleus.iaea.org/RRDB/RR/ReactorSearch.aspx?rf=1*), que además permite exportar los datos en formatos PDF o como hemos citado, en *Microsoft Excel*.

En principio el estudio es independiente de la fuente de datos usada. Si quisiéramos utilizar una fuente de datos distinta, tan sólo deberíamos configurar adecuadamente la capa *ETL* de la aplicación. Necesitamos trabajar con un formato de documento *Excel* en el que tengamos identificados las celdas que identifican las propiedades objeto de estudio, y las celdas que definen los valores para esas propiedades.

Así mismo si el formato de datos a importar fuese diferente, tan sólo sería necesario adaptar esta capa *ETL*, quedando el resto de partes de la aplicación inalteradas.

Dada la importancia del tratamiento de datos energéticos en la actualidad, y el interés en este proyecto de profundizar en técnicas de *FCA* y *Ontology Matching*; el objetivo del presente PFC será el de, basándonos en el análisis de los datos que la IAEA pone a disposición del gran público, generar un *concept lattice* con el objetivo de implementar un navegador interactivo sobre los datos de reactores disponibles.

El navegador, estando visualizando los datos de un reactor en concreto, presentará al usuario enlaces a otros reactores similares, que compartan alguna característica. Los enlaces a reactores con mayor grado de similitud, serán presentados de una forma más relevante. El navegador será implementado como una aplicación web J2EE, que utilizará los datos generados del *concept lattice*, de una forma similar a la propuesta expuesta en [4].

Como hemos comentado anteriormente, para la parte ETL de carga y transformación de datos, utilizaremos *Hadoop* y *Apache Mahout* para la discretización de los valores de algunos campos. La generación del *concept context* y *concept lattice* se implementará nativamente en el *framework Hadoop*.

La información generada del *concept lattice, una vez generada,* será exportada a una base de datos relacional, para que pueda ser consultada por la aplicación web.

En este punto haremos una reflexión sobre el uso de esta tecnología. Como venimos comentando, para el cálculo de los conceptos nos basaremos en un *framework* distribuido tipo *map-reduce* como *Hadoop*, aunque para el desarrollo de este PFC disponemos solamente de un clúster de una sola máquina. Esto es así porque uno de los objetivos más importantes del proyecto es el de poder desarrollar soluciones potenciales capaces de tratar con cantidades grandes de datos.

Aunque el juego de datos empleado no es especialmente alto, el uso del *framework* nos garantiza que nuestro código será capaz de utilizar las prestaciones de computación de un clúster real, sin que hubiese que adaptar ni reconfigurar nuestra aplicación. El lector debe tener en cuenta que como estudiantes, nuestros recursos de computación son limitados; sin embargo las habilidades y técnicas para manejar problemas de un complejidad alta son objetivos que se desean abordar desde el presente trabajo.

La interacción entre todos estos componentes de la aplicación puede visualizarse de forma visual en la siguiente Ilustración 2: Arquitectura de la aplicación:

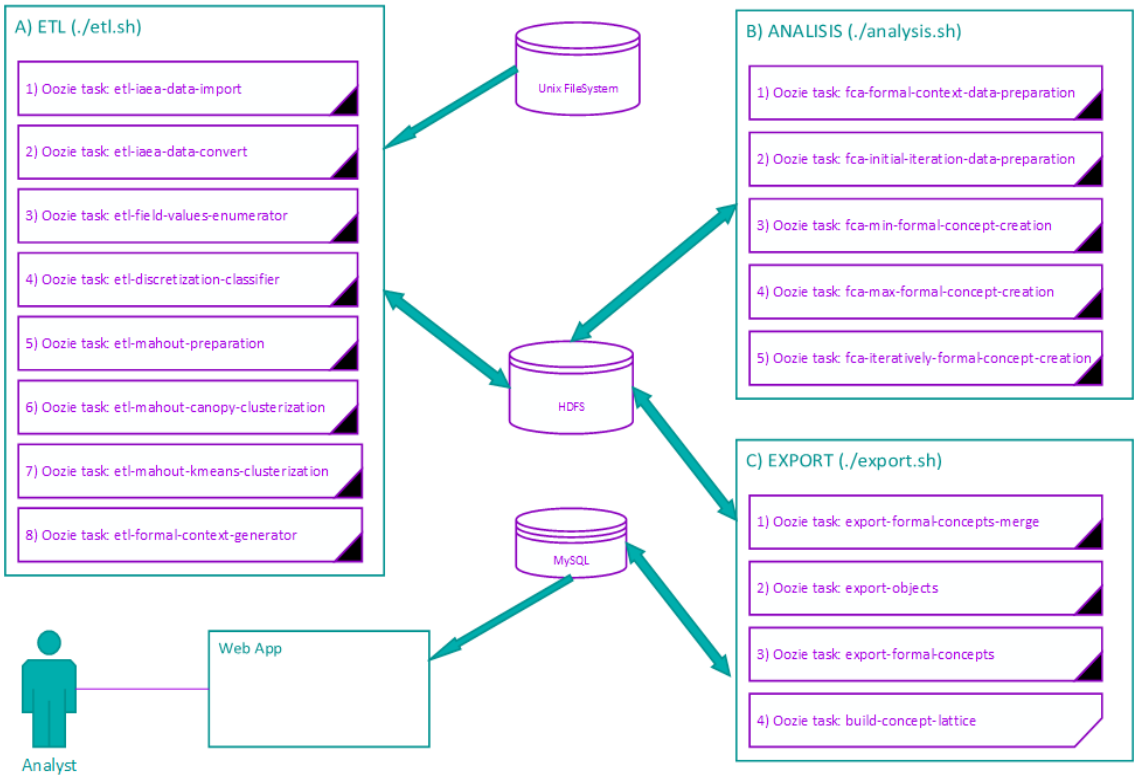

Ilustración 2: Arquitectura de la aplicación

2.1 Fase ETL

2.1.1 Tratamiento previo

La fuente de datos se puede consultar en el sitio web de datos de la IAEA, concretamente en http://nucleus.iaea.org/RRDB/RR/ReactorSearch.aspx?rf=1.

Cómo paso previo, se cargará el contenido desde formato *Excel* proporcionado por la IAEA en un *sequence file* en el sistema de ficheros *HDFS* del clúster de *Hadoop*. Posteriormente, se exportará a una base de datos relacional. De esta forma, los datos sin tratar estarán disponibles desde la aplicación del sistema de navegación para el cliente final (hay campos informativos, que no forman parte del proceso de análisis que, sin embargo, será necesario mostrar en la interfaz de usuario).

Cabe decir, que con los datos existentes actualmente desde la IAEA, tenemos 150 reactores. No es una cantidad elevada, sin embargo será suficiente para los objetivos de estudio del presente proyecto.

No obstante, el número de atributos es elevado, por lo que la elaboración del *concept lattice* asociado es de suficiente complejidad para justificar el presente estudio.

2.1.2 Análisis de campos de reactores a importar

Se muestra a continuación información sobre los campos presentes en la fuente de datos (*Microsoft Excel*) proveniente de la IAEA. Los campos marcados con el método 'automático' ya vienen codificados, y no será necesario pues ningún proceso de discretización adicional.

Por ejemplo, para el campo '*Category*', los valores posibles son:

- RESEARCH
- TRAINING
- CRITICAL ASSEMBLY
- TEST
- PROTOTYPE REACTOR
- SUB-CRITICAL ASSEMBLY
- ELECTRICITY PRODUCING
- SUB-CRITICAL ASSEMBLY

No obstante, no será necesario enumerar los valores posibles para cada campo, pues la capa ETL procesará y codificará todos estos valores de forma automática, a partir de las instancias encontradas en los datos de entrada.

Una excepción a este proceso serán los campos marcados con método "*Splitted*", que se sumariza al finalizar esta sección.

El caso de los campos marcados con método *K-Means* tiene que ver con el tratamiento de valores numéricos que serán discretizados mediante este algoritmo de clusterización. Por defecto, utilizaremos tres niveles de agregación (K = 3), dado que, al desconocer el dominio del problema, será suficiente con tener agrupados los datos en los niveles: LOW, MEDIUM y HIGH de forma genérica.

Finalmente, en cuestión de campos de fechas (marcados además con el método "*Timestamped*") tendremos cuatro niveles (K = 4): REALLY_OLD, OLD, NEW, FUTURE. Cabe decir que primeramente se aplicará una transformación para obtener el valor numérico -*timestamp*- correspondiente de la fecha.

Campo	Tratamiento	Método
Location.country	Codificación	Automático
Location.city	Codificación	Automático
Facility.name	N/A (info field)	N/A
Facility.IAEA_code	N/A (info field)	N/A
Status.status	Codificación	Automático
Status.comment	N/A (info field)	N/A
Category	Codificación	Automático
InformationDate	Codificación	Timestamped, K-Means K = 4
Information.owner	N/A (info field)	N/A
Information.operator	N/A (info field)	N/A
Information.licensing	N/A (info field)	N/A
Information.administrator	N/A (info field)	N/A
Information.safeguards	Codificación	Automático
Information.construction	Codificación	Timestamped, K-Means K = 4
Information.criticality	Codificación	Timestamped, K-Means K = 4
Information.operators	Codificación	K-Means K = 3
Information.total_staff	Codificación	K-Means K = 3
TechData.reactor_type	Codificación	Automático
TechData.thermal_power.steady_kw	Codificación	K-Means K = 3
TechData.thermal_power.pulsed_mw	Codificación	K-Means K = 3
TechData.max_flux_steady_thermal	Codificación	K-Means Discretización, K = 3
TechData.max_flux_steady_fast	Codificación	K-Means K = 3
TechData.max_flux_pulsed_thermal	Codificación	K-Means K = 3
TechData.max_flux_pulsed_fast	Codificación	Kmeans K = 3
TechData.moderator_material	Codificación	Automático
TechData.coolant_material	Codificación	Automático
TechData.control_rods_material	Codificación	Automático
TechData_control_rods_number	Codificación	K-Means K = 3
TechData.reflector_material	Codificación	Automático
TechData.sites_number	Codificación	K-Means K = 3
TechData.cooling_natural_convection	Codificación	Boolean
TechData.cooling_forced	Codificación	Boolean
Experimental.vertical_channels	Codificación	K-Means K = 3
Experimental.vertical_max_flux	Codificación	K-Means K = 3

Experimental.horizontal_channels	Codificación	K-Means K = 3
Experimental.horizontal _max_flux	Codificación	K-Means K = 3
Experimental.use	Codificación	Split
Experimental.incore_irradiation_facilities	Codificación	K-Means K = 3
Experimental.incore_max_flux	Codificación	K-Means K = 3
Experimental.reflector_max_flux	Codificación	K-Means K = 3
Experimental.irradiation_number	Codificación	K-Means K = 3
Experimental.irradiation_channels	Codificación	K-Means K = 3
Utilization.hours_per_day	Codificación	K-Means K = 4
Utilization.days_per_week	Codificación	K-Means K = 4
Utilization.weeks_per_year	Codificación	K-Means K = 4
Utilization.mw_days_per_year	Codificación	K-Means K = 4
Utilization.materials	Codificación	Boolean
Utilization.runs_number_per_year	Codificación	K-Means K = 3
Utilization.isotope_total_activity_per_year	Codificación	K-Means K = 3
Utilization.isotopes	Codificación	Split
Utilization.neutron_scattering_hours_per_year	Codificación	K-Means K = 4
Utilization.neutron_scattering_methods	Codificación	Split
Utilization.neutron_radiography	Codificación	Boolean
Utilization.neutron_radiography_hours_per_year	Codificación	K-Means K = 4
Utilization.neutron.capture_therapy	Codificación	Boolean
Utilization.neutron.capture_therapy_patients_per _year	Codificación	K-Means K = 4
Utilization.neutron.activation_analysis_samples_ per_year	Codificación	K-Means K = 3
Utilization.neutron.activation_analysis_methods	Codificación	Split
Utilization.transmutation_mass_kg_per_year	Codificación	K-Means K = 3
Utilization.transmutation_gemstone_kg_per_year	Codificación	K-Means K = 3
Utilization.geochronology_samples_per_year	Codificación	K-Means K = 3
Utilization.geochronology_methods	Codificación	Split
Utilization.teaching	Codificación	Boolean
Utilization.teaching_students_year	Codificación	K-Means K = 3
Utilization.training	Codificación	Boolean
Utilization.experimenters_number	Codificación	K-Means K = 3
Utilization.other_use	Codificación	Boolean
Utilization.other_use_describe	N/A (info field)	Info field

Tabla 2: Análisis de campos para la fase de ETL

En cuanto al tratamiento de los campos, distinguimos entre:

- Codificación: El valor se codifica para su uso en el análisis, con el objetivo de generar el formal context.
- N/A: Es un campo informativo únicamente, y no se usará para generar el *formal context*. Sin embargo, se mostrará en la interfaz del sistema de navegación.

Respecto al método de tratamiento:

- N/A: No se efectuará tratamiento alguno.
- Automático: Se generará un código numérico por cada valor diferente que aparezca en el dominio.
- Timestamped: En caso de fechas, se convertirán primero a formato numérico (unix timestamp)
- K-Means: Se realizará una discretización de los valores del campo siguiendo el método no supervisado de clusterización de K-Means.
- Split: Se da en el caso de campos multivaluados. En estos casos, se obtendrá cada valor individual, y se generará un campo el *formal context* por cada valor diferente del dominio encontrado.
- Boolean: Se codificará como 1 los valores con el literal 'Yes' y a 0 los valores con el literal 'No'

2.1.3 Generación del formal context

Una vez disponibles los datos de la fase ETL en el clúster *Hadoop*, pasaremos a generar el *formal context*. Para ello -partiendo de las transformaciones realizadas en cada campo en el paso previo- por cada código obtenido en el dominio de valores de cada campo, generaremos una propiedad en el *formal context* final.

Por ejemplo, si para el campo 'location.country' el dominio de valores obtenidos ha sido 'Spain', 'France' y 'Germany', el *formal context* final incluirá las propiedades: 'location.country.spain', 'location.country.france' y 'location.country.germany'. Como hemos comentado en secciones anteriores, cada propiedad tendrá un valor binario de '1' ó '0'.

Más concretamente, en nuestra implementación, el resultado final de la fase *ETL* es un fichero de secuencia *HDFS*, donde cada registro es un mapa con clave nombre de campo generado; y el valor será un booleano indicando si el objeto tiene el atributo o no.

2.1.4 Subdivisión en tareas

Dado que la fase de la ETL es compleja, y se necesitan realizar diversas transformaciones hasta alcanzar el resultado final, se ha divido la ejecución en diversas tareas. Cada una de estas tareas utiliza como entrada alguna salida generada en alguna fase anterior.

Nombre	**etl-iaea-data-import**
Descripción	Normalizar formato *Excel* de datos de reactores de la IAEA
Entrada	`/user/cloudera/etl/import/ReactorResearch.xls`
Salida	`/user/cloudera/etl/import/Merged-ReactorResearch.xls`
Tipo	*Java action*

Nombre	**etl-iaea-data-convert**
Descripción	Generar datos de reactores en formato de fichero de secuencia HDFS
Entrada	`/user/cloudera/etl/import/Merged-ReactorResearch.xls`
Salida	`/user/cloudera/etl/converted/etl-input.seq`
Tipo	*Java action*

Nombre	**etl-field-values-enumerator**
Descripción	Generar ficheros *HDFS* con mapeos literal – código numérico para todos los posible valores que pueden tomar los campos de tipo enumerado
Entrada	`/user/cloudera/etl/converted/etl-input.seq`
Salida	`/user/cloudera/etl/enumerations`
Tipo	*Map-Reduce*

Nombre	**etl-discretization-classifier**
Descripción	Generar ficheros *HDFS* con datos discretizados para campos de tipo enumerado
Entrada	`/user/cloudera/etl/converted/etl-input.seq,` `/user/cloudera/etl/enumerations/*`
Salida	`/user/cloudera/etl/discretized`
Tipo	*Map-Reduce (map only)*

Nombre	**etl-mahout-preparation**
Descripción	Generar ficheros de entrada susceptibles de ser procesados por *Mahout* para la clusterización *K-Means*
Entrada	`/user/cloudera/etl/discretized`
Salida	`/user/cloudera/etl/mahout-input`
Tipo	*Map-Reduce*

Nombre	**etl-mahout-canopy-clusterization**
Descripción	Generar clusterización inicial de *Mahout* requerida para la posterior clusterización *K-Means* final (clusterización *canopy*)
Entrada	`/user/cloudera/etl/mahout-input`

Salida	`/user/cloudera/etl/mahout-output/canopy`
Tipo	*Java action + Map-Reduce (Mahout)*

Nombre	**etl-mahout-kmeans-clusterization**
Descripción	Realizar la clusterización final de *K-Means* de *Mahout*
Entrada	`/user/cloudera/etl/mahout-input, /user/cloudera/etl/mahout-output/canopy`
Salida	`/user/cloudera/etl/mahout-output/kmeans`
Tipo	*Java action + Map-Reduce (Mahout)*

Nombre	**etl-formal-context-generator**
Descripción	Generación del formal context final en un fichero de secuencia *HDFS* formado por registros compuestos por un mapa campo, booleano
Entrada	`/user/cloudera/etl/discretized, /user/cloudera/etl/mahout-output/kmeans/*-kmeans/clusteredPoints`
Salida	`/user/cloudera/etl/formal_context`
Tipo	*Map-Reduce (map only)*

2.2 Fase análisis

2.2.1 Descripción del algoritmo

En esta fase se llevará a cabo la ejecución del algoritmo *FCA* sobre el clúster *Hadoop*. Para ello adaptaremos la versión *map-reduce* que se puede encontrar en [2].

Antes de nada, necesitaremos introducir algunos conceptos teóricos. Como se ha venido esbozando hasta ahora, el análisis del *formal context* puede ser visto como el procesamiento de tablas de datos binarios describiendo relaciones entre objetos y atributos respectivamente. Entonces, como se ha comentado con anterioridad, la entrada para el algoritmo *FCA* es una tabla de datos con filas correspondientes a objetos, columnas correspondientes a atributos y entradas en la tabla donde 'x' indicará si el objeto denotado por la fila actual tiene o no el atributo denotado por la columna. Un ejemplo puede verse en Tabla 1: Formal context -representación tabular- en la página 9 y Ilustración 1: Concept lattice en la página 10.

Una tabla de datos como esta puede ser interpretada como una relación binaria $I \subseteq X \times Y$ tal que $\langle x, y \rangle \in I$ sí y solo sí el objeto x tiene el atributo y.

Cada *formal concept* en $I \subseteq X \times Y$ induce un par de operadores $\uparrow$ y $\downarrow$ definidos para cada $A \subseteq X$ y $B \subseteq Y$ como se muestra a continuación:

1. $A\uparrow = \{y \in Y \mid para\ cada\ x \in A : \langle x, y \rangle \in I\}$
2. $B\downarrow = \{x \in X \mid para\ cada\ y \in B : \langle x, y \rangle \in I\}$

De manera informal, $A\uparrow$ es el "conjunto de todos los atributos compartidos por todos los objetos de A", y $B\downarrow$ es el "conjunto de todos los objetos que comparten todos los atributos de B".

Entonces, si identificamos $\langle A, B \rangle$ como un *formal concept*, A será llamado el *extent* de

$\langle A, B \rangle$ y estará constituido precisamente por el conjunto de todos los objetos compartiendo todos los atributos de B, y B el *intent* de $\langle A, B \rangle$ como el conjunto de todos los atributos compartidos por todos los objetos de A.

Con todas estas definiciones realizadas, podremos configurar ahora el contenido de los objetos *key* y *value* en la cual se basará la adaptación al *framework map-reduce*.

- **Key**: Tupla $\langle B, y \rangle$ donde B es un *intent* del concepto $\langle A, B \rangle \in \mathrm{B}(X, Y, I)$ y donde $y \in Y$ es un atributo.
- **Value**: Es un nuevo concepto $\langle C, D \rangle \in \mathrm{B}(X, Y, I)$.

Una vez realizada esta pequeña introducción, seremos capaces de presentar el pseudocódigo para la función *map*:

Función Map

Input: Par $\langle key, value \rangle$ donde *key* es $\langle B_{0,y} \rangle$ y *value* es $\langle A, B \rangle$

Result = 0

Para j = y **hasta** $|Y|$ **hacer**

Si $j \in B$ **entonces** continuar;

$C = A \cap \{j\}\downarrow$

$D = C\uparrow$

$result = result \cup \{\langle \langle B, j \rangle, \langle C, D \rangle \rangle\}$

Fin Para

Return result

Ilustración 3: Función Map

El arquetipo de la función *reduce* se presenta a continuación:

Función reduce

$$ReducirConceptos(\{\langle \langle B, j \rangle, \langle C, D \rangle \rangle\}) = \begin{cases} \{\langle \langle B, j+1 \rangle, \langle C, D \rangle \rangle\}, si\ B \cap Y_j = D \cap Y_j \\ vacío, en\ otro\ caso \end{cases}$$

$$Dónde\ Y_j \subseteq Y\ está\ definido\ como: Y_j = \{y \in Y | y < j\}$$

Ilustración 4: Función Reduce

Esta adaptación del algoritmo clásico a un procesamiento tipo *map-reduce* se basa en la generación de todos los nuevos conceptos formales desde los datos de entrada en la función *map*, incluso aunque se generen duplicados. En la función *reduce* se controla esta situación implementando un test canónico, que descartará valores generados incorrectos.

Las fases *map-reduce* se deben ejecutar de forma iterativa hasta alcanzar la condición de parada: La función *reduce* no devuelve ningún resultado.

Básicamente, cada iteración representa un nivel del árbol; mientras que la función *map* calcula nodos descendientes, la función *reduce* determina qué nodos deben ser utilizados en los siguientes cálculos.

Este flujo de fases *map-reduce* se puede esquematizar de la siguiente manera:

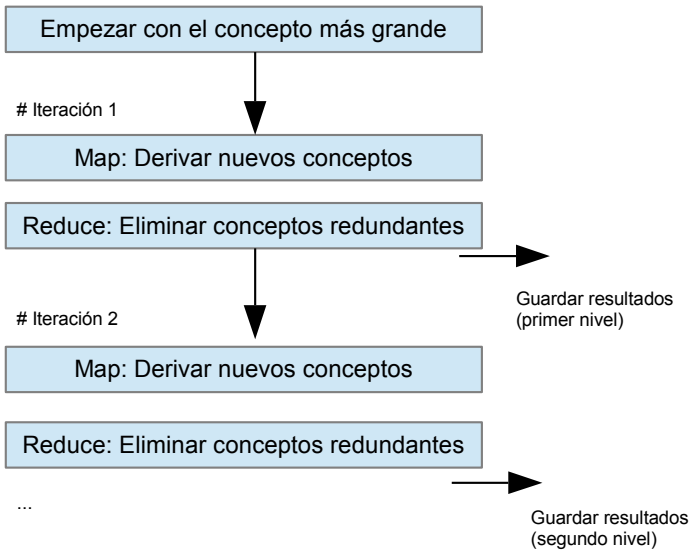

Ilustración 5: Ejemplo de flujo de ejecuciones map-reduce

Siguiendo el ejemplo mostrado en Tabla 1: Formal context -representación tabular- en la página 9 y Ilustración 1: Concept lattice en la página 10; para la ejecución de la primera función *map*, tendríamos:

- *Key*: $\langle \emptyset, composite \rangle$. Nótese que el conjunto de atributos está vacío, dado que se trata de la iteración inicial, y *composite* es el primero de los atributos.
- *Value*: $\langle \{1,2,3,4,5,6,7,8,9,10\}, \{\} \rangle$. Nótese que este *formal concept* inicial corresponde al nodo máximo[3].

A partir de la ejecución inicial con estos datos de entrada, se irían completando la computación de los *formal concept* de cada nivel correspondiente, hasta alcanzar la condición de parada (no se han podido generar nuevas ocurrencias).

Al finalizar el cálculo global, se exportarán todos los datos del *concept lattice* generado desde el sistema de ficheros *HDFS* del clúster *Hadoop* a una base de datos relacional, para que estén disponibles en la aplicación web cliente.

2.2.2 Tratamiento previo

Como se ha comentado en la sección anterior, necesitamos generar los datos previos a la primera iteración, con los datos correspondientes al nodo máximo y la clave inicial; por lo que programaremos una acción específica para generar en el clúster estos datos.

3 Para consultar la definición de nodo máximo, véase 1.1 Formal Concept Analysis, página 9

Así mismo, hay otro caso especial en relación a los nodos presentes en el *concept lattice*: El nodo mínimo (*formal concept* compuesto por todos los atributos del dominio como *intent*). Hemos podido constatar que este caso especial no es generado durante la ejecución iterativa de este algoritmo. Este *formal concept* no es de especial utilidad para la aplicación en nuestro problema de referencia; no obstante en aras de una completa corrección formal crearemos una acción específica *ad-hoc* para generar también este *formal concept*.

Para terminar con las cuestiones relativas al tratamiento previo a realizar, haremos mención a una operación que puede ser costosa en términos de ejecución:

$\{j\}\downarrow$

En lenguaje natural, podría enunciarse como: "Conjunto de todos los objetos compartiendo el atributo 'j' ". Para obtener esta información, en principio deberíamos recorrer todo el *formal context* generado; e ir seleccionando los objetos que contengan el atributo en cuestión.

Dado que esta es una operación usada con frecuencia; prepararemos un fichero por cada atributo presente en nuestro *formal context* con todos los identificadores de los objetos que lo comparten ya pregenerados.

2.2.3 Validación de la implementación realizada

La generación del *concept lattice* es un problema de complejidad exponencial; por lo que las pruebas en el clúster son bastante caras en términos de tiempo de computación y depuración. Es por ello que es de vital importancia poder lanzar la ejecución sobre el set de datos de nuestro problema una vez hayamos validado la corrección del algoritmo implementado.

Para ello hemos usado *MRUnit* (https://mrunit.apache.org/). Como se puede leer en la página principal del proyecto: "*MRUnit* es una librería java que ayuda a los desarrolladores a construir test unitarios sobre *jobs Apache Hadoop map-reduce*".

Concretamente, nos permite lanzar ejecuciones de *mappers*, *reducers* y *jobs* completos *map-reduce*; y validar que los resultados son los esperados; sin la necesidad de ejecutar en un clúster *Hadoop*; y permitiendo así la depuración con las herramientas habituales y rápidos ciclos desarrollo-prueba.

Para poder validar que la implementación ha sido correcta, hemos usado el dominio del problema y los datos de entrada descritos en **[2],** creando aserciones a cumplir basados en los datos de salida descritos en este mismo documento. La única dificultad ha sido poder evitar el acceso al clúster para la lectura de objetos y atributos durante la ejecución de los *tests*; pero dado que tenemos esto encapsulado en un único componente, tan sólo ha sido necesario realizar un *mock* adaptado a los datos de test, con las librerías de *testing* habituales[4].

2.2.4 Subdivisión en tareas

Dado que, como hemos comentado, necesitamos generar datos previos y casos especiales; hemos divido la ejecución en varias acciones.

Nombre	**fca-formal-context-data-preparation**
Descripción	Generar ficheros con datos de objetos compartidos por atributo pregenerados en formato de fichero de secuencia HDFS
Entrada	`/user/cloudera/etl/formal_context`
Salida	`/user/cloudera/analysis/data_preparation`

4 *Mockito* (http://mockito.org/)

Tipo	*Map-reduce*

Nombre	**fca-initial-iteration-data-preparation**
Descripción	Generar datos iniciales, previos a la primera iteración
Entrada	`/user/cloudera/etl/formal_context,` `/user/cloudera/analysis/data_preparation`
Salida	`/user/cloudera/analysis/fca/max_concept`
Tipo	*Java action*

Nombre	**fca-min-formal-concept-creation**
Descripción	Generar datos de caso especial de nodo mínimo
Entrada	`/user/cloudera/etl/formal_context,` `/user/cloudera/analysis/data_preparation`
Salida	`/user/cloudera/analysis/fca/min_concept`
Tipo	*Java action*

Nombre	**fca-iteratively-formal-concept-creation**
Descripción	Generar *formal concepts* de forma iterativa

Entrada	`/user/cloudera/etl/formal_context,`
	`/user/cloudera/analysis/data_preparation,`
	`/user/cloudera/analysis/fca/max_concept`

Salida	`/user/cloudera/analysis/fca/iterations`
Tipo	*Java action, que lanza jobs Map-reduce hasta alcanzar condición de parada*

2.3 Fase aplicación de interfaz de usuario

Como hemos venido comentado, durante esta fase nos centraremos en desarrollar una aplicación web que permita ir explorando el catálogo de reactores de forma interactiva, de tal forma que estando visualizando los datos de un reactor en concreto, se nos presenten enlaces a otros reactores similares.

Los reactores con un grado de similitud más alto, se presentarán de una forma más resaltada en la interfaz de usuario.

Para implementar esta parte, cargaremos datos en una base de datos relacional a partir de los resultados previos de la fase de análisis.

2.3.1 Modelo de datos

Para el modelo de datos, se ha definido la siguiente estructura relacional:

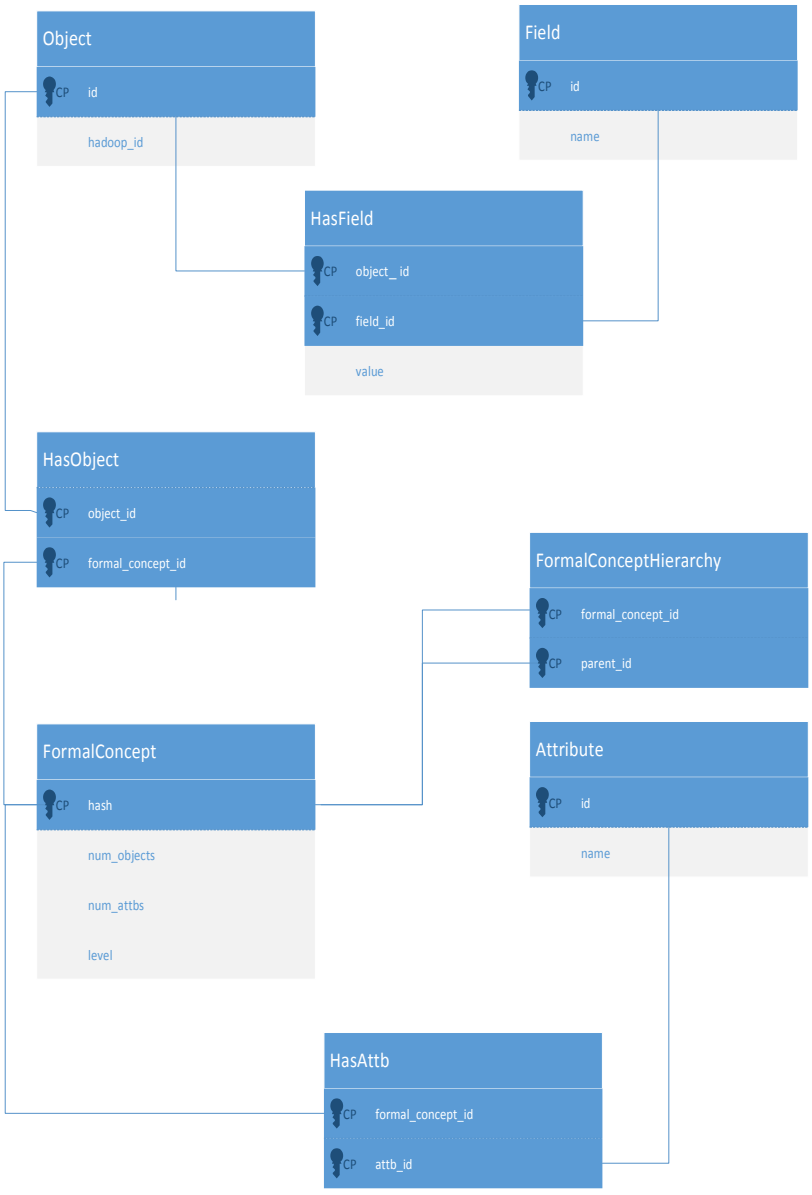

Ilustración 6: Modelo de datos

Según este esquema, la entidad "Object" corresponde a los objetos del dominio de nuestro problema que deseamos analizar (en el caso que nos atañe, reactores). Cabe decir que el diseño es flexible y susceptible de ser utilizado como resultado del análisis -fca- de cualquier tipo de problema.

De esta forma, cada objeto tiene asociados una serie de campos (entidad "Field"), que dependen del dominio del problema que estemos tratando. En nuestro caso, corresponderán a cada una de las columnas de la hoja *excel* de reactores que publica la IAEA.

Respecto a los conceptos formales en sí, vienen representados por la entidad "FormalConcept". Los conceptos vienen identificados por el *hash* proveniente del modelo de análisis a partir del cual es serializado en la base de datos. Esto es así ya que la búsqueda en la base de datos de un concepto a partir de su lista de objetos y atributos es una operación costosa; de esta manera ganamos en eficiencia (ya que la generación del *hash* es una operación de coste constante). Por cada concepto, almacenamos también meta-información que será de utilidad a la hora de explotar los datos como el

número de objetos y atributos, así como del nivel en el grafo final resultante (*concept lattice*).

Los atributos generados en la fase de generación del *formal context* están representados por la entidad "Attribute". Cabe decir en este punto que un *attribute* es diferente de un *field*, en el sentido que este viene definido directamente en los datos de entrada que pretendemos analizar; mientras que un atributo es generado en el *formal context* como resultado del análisis.

Un concepto formal está asociado tanto con los atributos del *formal context* generado que satisfacen todos sus objetos, como de los objetos en sí que lo componen.

Por último, y para representar la herencia de conceptos que compone el grafo denominado *concept lattice*, tenemos la entidad "FormalConceptHierarchy", donde vinculamos cada concepto con el concepto del que hereda. Cabe reseñar aquí que un nodo puede heredar de más de un padre.

2.3.2 Algoritmo de generación del concept lattice

Los conceptos formales son generados durante la fase de análisis, siguiendo el algoritmo *map-reduce* definido en *[2]*. No obstante mediante este algoritmo se generan los conceptos en sí de una manera distribuida; pero no su relación entre ellos: Es decir, no contempla su ordenación ni jerarquización en lo que se denomina *concept lattice*[5].

Para ello nos hemos basado en el algoritmo definido en *[9]*. Sin embargo su aplicación en nuestra implementación en particular presentaba algunos problemas :

- Implicaba generar todas las clases de equivalencia posibles para todos los atributos definidos en el *formal context*. Esto es, todas las combinaciones posibles de atributos que potencialmente pueden llegar a darse. Dado que el número de atributos que manejamos es elevado (547); esta operación consumía una cantidad de recursos (*heap space*) que no se podía manejar.
- Implicaba generar todos los conceptos posibles; para luego descartar los que realmente no tienen cabida en nuestro problema. Esto es especialmente ineficiente en nuestro caso, dado que los conceptos ya han sido generados (aunque puede ser una estrategia válida para generar conceptos y jerarquizar cuando se parte de cero).
- Hubiera tenido como consecuencia ir re-ordenando la jerarquía de conceptos de forma iterativa. Esto es un problema en nuestro caso, pues implica operaciones adicionales de inserción/eliminación sobre la base de datos.

Para solucionar estos problemas, hemos optado por adaptar el algoritmo para que:

1. Obtenga únicamente los conceptos existentes directamente desde la base de datos (no sería necesaria la generación de clases de equivalencia).
2. Inserte siempre los nodos hoja en la jerarquía; de esta forma, no será necesario re-ordenar los nodos colocados previamente (es decir, evitar los casos en los que un concepto debe ser insertado en una posición intermedia en el grafo).

A continuación detallamos la solución final adoptada:

5 Véase 1.1 Formal Concept Analysis página 9 e Ilustración 1: Concept lattice, página 10

Construir *concept lattice*

```
leafConcepts = {maximumFormalConcept()} # root node
level = 0
for conceptAttbSize : 0 .. numberOfAttbs do
  levelInheritance = map() # tipo de datos modo 'diccionario' o array asociativo
  pendingConcepts = findPendingConceptsBySize(conceptAttbSize)
  for concept :  pendingConcepts do
    parents = searchForParents(concept, leafNodes)
    levelInheritance[concept] = parents
    insertInheritance(concept,parents)
    markConceptAsProcessed(concept,level)
  end
  if  pendingConcepts <> {} then
    updateLeafConcepts(leafConcepts,levelInheritance)
    level = level + 1
  end
end
```

Ilustración 7: Algoritmo de construcción del concept lattice

Como se puede observar, la estrategia se basa en mantener el conjunto de nodos hoja (esto es, que no tienen ningún descendiente). Como en cada nivel del grafo, los conceptos tienen necesariamente más atributos que sus ancestros, si empezamos por el concepto raíz (nodo máximo, 0 atributos); podemos ir completando ordenadamente por niveles todos los nodos del grafo (conceptos con un atributo, conceptos con dos atributos.. hasta el nodo mínimo; el concepto con todos los atributos). En cada nuevo nivel, los padres de cada nodo deben pertenecer al conjunto de nodos hoja (o alguno de los ancestros de estos, para casos especiales).

Como el conjunto de nodos hoja no es elevado, se pueden mantener en memoria, evitando así accesos a la base de datos en la mayoría de situaciones.

Se detalla a continuación la función *searchForParents*:

Función searchForParents (concept, leafConcepts)

```
parents = filterConceptsICanInheritFrom(concept,leafConcepts)
if parents = {} then
  parents = closestAncestorsFor(concept,leafConcepts)
end
return parents
```

Ilustración 8: Detalle de función searchForParents()

Como queda especificado, si es posible heredar de alguno de los actuales nodos hoja del grafo lo haremos evitando accesos innecesarios a la base de datos (caso más común). En caso contrario buscaremos en sus padres recursivamente hasta encontrar los conceptos más cercanos de los cuales podamos heredar (en este caso sí que deberemos realizar accesos a la base de datos). Esta segunda casuística queda de relieve en el detalle de la función *closestAncestorsFor:*

Función closestAncestorsFor(concept, childrenToCheck)

```
if childrenToCheck = {} then return {}
parents = findParents(childrenToCheck)
validAncestors = filterConceptsICanInheritFrom(concept,parents)
if validAncestors = {} then
  return closestAncestorsFor(concept,parents)
end
return validAncestors
```

Ilustración 9: Detalle de función closestAncestorsFor()

Por otra parte, tras procesar cada nivel del grafo, deberemos actualizar el conjunto de nodos hoja para que estén disponibles en la siguiente iteración. Se muestra detalle seguidamente:

Función updateLeafConcepts (leafConcepts, levelInheritance)

```
for child, parents : levelInheritance do
  leafConcepts = leafConcepts + {child}
  for parent : parents do
    leafConcepts = leafConcepts - {parent}
  end
end
```

Ilustración 10: Detalle de función updateLeafConcepts()

Es decir, todos los conceptos insertados en el grafo durante el procesamiento del nivel en curso pasan a formar parte del conjunto de nodos hoja; mientras que sus padres deben dejar de formar parte, si lo estaban haciendo.

2.3.3 Estrategia de explotación del concept lattice generado

Como se ha venido remarcando durante el presente proyecto, el objetivo es analizar el grado de similitud entre objetos (reactores en el dominio de nuestro problema). Para ello, tras generar el grafo del *concept lattice*, debemos desarrollar una aplicación cliente (web en este caso) que explote toda esta información.

Tal como hemos comentado, dado un reactor; presentaremos en orden de relevancia enlaces a otros reactores 'similares'.

Siguiendo con la reflexión realizada sobre el *concept lattice*, el nodo superior corresponde al concepto máximo (todos los objetos, ningún atributo). De este concepto máximo, heredan otros conceptos sucesivamente y formando una jerarquía; los cuales van incrementando paulatinamente el número de atributos que manifiestan todos los objetos que lo forman. Por otra parte, dado un concepto; los conceptos de los que hereda en la jerarquía incluyen todos los objetos que forman parte del mismo.

Entonces, dado un objeto (reactor) que debe ser mostrado en la aplicación cliente, para obtener todos los objetos similares, tan sólo deberemos buscar el concepto que lo contenga en la base de datos con mayor número de atributos. De esta forma el resto de objetos presentes en el concepto hallado serán los más similares con el reactor en cuestión.

A partir de este concepto, recorriendo la jerarquía de herencia hasta alcanzar el concepto raíz o máximo nos permitirá obtener el resto de objetos con un grado de similitud menor de manera progresiva[6].

2.3.4 Subdivisión en tareas

Nombre	**export-formal-concepts-merge**
Descripción	Unificar todos los ficheros de formal concepts generados en uno único para poder ser procesado
Entrada	`/user/cloudera/analysis/fca/*/*`
Salida	`/user/cloudera/export/mergedFormalConcepts`
Tipo	*Java action*

Nombre	**export-objects**
Descripción	Exportar los objetos (reactores) a la base de datos
Entrada	`/user/cloudera/etl/converted/etl-input.seq,` `/user/cloudera/etl/discretized`
Salida	`Tablas object,field,has_field`
Tipo	*Java action*

Nombre	**export-formal-concepts**
Descripción	Exportar los formal concept a la base de datos
Entrada	`/user/cloudera/export/mergedFormalConcepts,` `Tablas object,field,has_field`
Salida	`Tablas formal_concept, attribute, has_object, has_attb, formal_concept_hierarchy`
Tipo	*Java action*

6 Véase “Anexo V. Ejecución del cliente web”, página 59

Nombre	**build-concept-lattice**
Descripción	Constuir jerarquía “concept lattice” entre los conceptos exportados a la base de datos
Entrada	`Tablas object, formal_concept, attribute, has_object, has_attb`
Salida	`Tablas formal_concept, formal_concept_hierarchy`
Tipo	*Java action*

3. Plataforma tecnológica

Tanto ***Hadoop*** (https://hadoop.apache.org) como ***Mahout*** (http://mahout.apache.org) son proyectos *Open Source* de licencias abiertas, cuyas fuentes están disponibles de forma pública. Sin embargo, dada la gran cantidad de componentes que puede ser necesario utilizar en un entorno de procesamiento *Big Data*, y la necesidad de herramientas adicionales que faciliten la monitorización y despliegue del software desarrollado, así como de la plataforma en sí misma; diversas empresas han desarrollado distribuciones de *Hadoop* testadas y estables, facilitando la puesta en marcha de proyectos basados en estas tecnologías.

Dado que ***Cloudera*** destaca como la distribución de *Hadoop* más usada, basaremos nuestro trabajo en la penúltima versión estable disponible, *CDH5.3*, que además incluye *Mahout* pre-instalado, que como hemos explicado será necesario en la fase de *ETL* para lanzar los procesos de clusterización. No hemos basado este trabajo en la última versión disponible en aras de una mayor estabilidad, no de la versión de *Hadoop* en sí misma; pero si en la compatibilidad con algunas otras herramientas disponibles[7].

Cloudera proporciona una imagen de una máquina virtual preparada con todo el software necesario, que puede ser descargada desde:
http://www.cloudera.com/content/cloudera/en/downloads/quickstart_vms/cdh-5-3-x.html.

Así mismo, dado que diversos procesos *map-reduce* son necesarios tanto durante las fases de *ETL* como análisis, será necesario poder lanzar los procesos de forma coordinada. Para ello nos serviremos de apache ***Oozie*** (http://oozie.apache.org). *Oozie* es un software de planificación que permite no sólo la ejecución de flujos combinados de procesos *map-reduce*, si no también la creación de ficheros y directorios HDFS, e incluso la ejecución de programas java convencionales.

Para comunicar los datos de las fases *ETL* y análisis con la aplicación cliente, nos serviremos de una base de datos relacional. En este respecto, no es el objetivo del presente proyecto profundizar en el mundo de las bases de datos, así que haremos uso de la última versión estable de **MySQL** (http://www.mysql.com) por estar disponible de forma gratuita en diversos sistemas operativos, y disponer de las características más demandadas en aplicaciones convencionales. En principio, solamente sería necesario almacenar los resultados finales del proceso, por lo cual no sería necesario optar por sistemas más potentes como Oracle 11g ó 12c (los resultados intermedios se almacenan directamente en el clúster *Hadoop* en un sistema de ficheros distribuido *HDFS*).

En cuanto a la implementación de la aplicación web cliente, utilizaremos ***apache tomcat*** (http://tomcat.apache.org), siendo un contenedor web ligero y ampliamente utilizado; pues no requeriremos la funcionalidad completa de un servidor de aplicaciones completo para una aplicación tan ligera.

7 Cómo por ejemplo la integración con *Oozie*, véase Clúster Hadoop página 37.

4. Planificación

En cuanto a la planificación, se puede encontrar la descomposición del trabajo en tareas realizada en la tabla siguiente:

Entorno		
Instalación VM cloudera	7 día/s	**100 %**
Pruebas Mahout	5 día/s	**100 %**
Entorno desarrollo	5 día/s	**100 %**
Conversión datos IAEA		
Mapper lector Excel IAEA	10 día/s	**100 %**
Reducer generador datos reactor	10 día/s	**100 %**
Normalización datos		
Mapper codificador datos	10 día/s	**100 %**
Procesos Mahout KMeans	10 día/s	**100 %**
Generar f. formal context	2 día/s	**100 %**
Implementación FCA		
Mapper FCA	10 día/s	**100 %**
Reducer FCA	10 día/s	**100 %**
Implementación cliente web		
Exportar datos de cluster *Hadoop* a BD	7 día/s	**100 %**
Desarrollo aplicación web	7 día/s	**100 %**
Conclusiones		
Sumarización resultados	5 día/s	**100 %**
Generación vídeo proyecto	3 día/s	**100 %**

Tabla 3: Descomposición del trabajo

A continuación, el detalle de tareas que se han completado durante en proyecto actual:

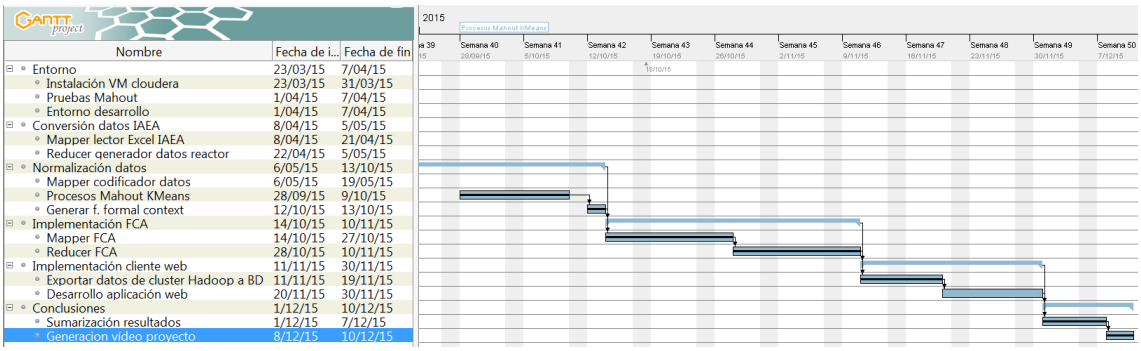

Ilustración 11: Diagrama de tareas completadas en el proyecto

5. Conclusiones

En el desarrollo del presente proyecto hemos tenido la oportunidad de aprender y profundizar sobre diversos corrientes o paradigmas dentro de la inteligencia artificial; especialmente en aquellas que versan sobre el establecimiento de conocimiento entre las relaciones de los objetos o instancias presentes en un dominio concreto.

Concretamente, e influenciados por el proyecto descrito en [4], *The virtual museum of the pacific*, hemos profundizado en la técnica del *formal concept analysis*, que consiste en derivar una jerarquía de conceptos en una ontología formal desde una colección de objetos y sus propiedades; donde la estructura jerárquica de la ontología nos permite extraer conocimiento sobre la interrelación de los elementos que la forman con facilidad.

Para ello hemos asimilado conceptos como el *formal context*, como modelo de partida inicial compuesto por un conjunto de objetos de la población; así como el conjunto de atributos de estudio y una función de pertenencia que nos indica si una instancia en concreto presenta o no un atributo en cuestión. A partir de esta estructura hemos aprendido a identificar los *formal concepts*, o subconjuntos de elementos derivados a partir del modelo inicial del *formal context* que cumplen una serie de requisitos que los hacen ideales para razonar sobre similitud; así como su jerarquización y ordenación en el grafo final denominado *concept lattice, que* tal como comentamos anteriormente, nos permite extraer conocimiento sobre el grado de similitud de los objetos en nuestro dominio.

Así mismo, hemos detallado los principales usos del FCA: *ontology alignment*, *ontology matching* y similarity reasoning; y hemos encuadrado nuestro problema en concreto dentro del *ontology matching*.

Estudiando el coste de las implementaciones más habituales de la FCA en [8], hemos visto como este puede ser exponencial. Así mismo hemos razonado que en la sociedad de hoy en día, empresas e instituciones precisan extraer conocimiento sobre cantidades cada vez mayores de datos, por lo que hemos puesto en valor la posibilidad descrita en [2] de realizar una implementación sobre un algoritmo adaptado a un *framework* de procesamiento distribuido como es *Hadoop*.

Centrados en este entorno de computación, hemos aprovechado implementaciones existentes de diversos algoritmos de *IA* -concretamente clusterización *Kmeans*- con *Apache Mahout*, pensadas para operar sobre grandes volúmenes de datos de entrada. Ello nos ha permitido implementar la primera capa de *ETL* para generar el modelo inicial del *formal context*, también dentro del mismo entorno.

Hemos tenido la oportunidad de comprobar la complejidad de desarrollo en un entorno de esta naturaleza, y del alto volumen de datos que se pueden generar; incluso con un *dataset* reducido como el empleado en el presente trabajo. A este respecto, nos hemos dado cuenta que:

1. El número de atributos considerados incrementa exponencialmente la complejidad. En nuestro caso de estudio, con un número de objetos reducido (150); pero con un número amplio de campos a estudiar (69), el número de conceptos generados ha sido más de 70000. Quizá en un modelo de explotación real hubiese sido más interesante empezar el estudio únicamente con el conjunto de campos más relevante a priori, e ir ampliándolo progresivamente.
2. Hay que tener en cuenta que la parte de la *ETL* (generación del modelo del *formal context* inicial) es un proceso complejo a considerar en cuento al esfuerzo necesario a realizar.

Por último hemos constatado como la generación del grafo final o *concept lattice* no es trivial cuando se trabaja con un volumen de datos mediano/grande; y hemos adaptado un algoritmo propio susceptible de ser aplicado para tratar *formal concepts* cargados desde bases de datos relacionales.

Así mismo hemos aprendido a conceptualizar arquitecturas complejas que combinen diferentes aproximaciones para tratar problemas cuya resolución se sustenta sobre algoritmos y tecnologías heterogéneas.

5.1 Futuras mejoras y ampliaciones

Cabe decir que, aunque el dominio de nuestro problema han sido los reactores nucleares de la *IAEA*, la solución y el modelo desarrollado es independientemente del problema en concreto. No obstante, en aras de agilizar el desarrollo, algunas configuraciones, así como algunas constantes que indican los nombres y tipos de los campos están dentro del código java del módulo del *backend*. Lo más correcto sería extraer toda esta configuración para que fuese 100% independiente del problema.

También nos hemos enfrentado a algunos problemas de memoria en el procesamiento *map-reduce*, por lo que una optimización debería ser realizada a ese respecto.

Aprovecho para apuntar que, aunque el algoritmo apuntado en [2] ha sido relativamente sencillo de implementar y ha funcionado bien, no contempla la ordenación de los conceptos para construir el *concept lattice* final. Sería muy interesante extender el algoritmo, y así evitar la última fase final, que ha planteado algunas dificultades; y es relativamente costosa.

Para finalizar, cabe decir que esta implementación se basa en la versión clásica de la *FCA*, que permite razonar sobre la base de que un objeto presenta o no un atributo. Sería interesante también profundizar sobre la versión difusa (fuzzy *FCA*); que permite razonar teniendo en consideración una función de pertenencia también difusa: Esto es, un objeto presenta un atributo no de una forma absoluta, sino en un grado o factor (por ejemplo, al 20%).

6. Bibliografía

[1] *A. Formica*. **Similarity reasoning for fuzzy concept lattices**. *2009*. Istituto di Analisi dei sistemi ed informatica. Consiglio nazionale delle riserche.
http://www.iasi.cnr.it/reports/R9002/R9002.pdf

[2] *Petr Krajca and Vilem Vychodil*. **Distributed algorithm for computing formal concepts using map-reduce framework**. *2009*. Watson School, State University of New York at Binghamton Dept. Computer Science, Palacky University, Olomouc.
http://upcase.inf.upol.cz/download/conf/ida/KrVychodil09_Dafcfcumrf.pdf

[3] *Wikipedia*. **Formal concept analysis**. https://en.wikipedia.org/wiki/Formal_concept_analysis

[4] *Youtube*. **The virtual museum of the pacific**.
https://www.youtube.com/user/VirtualMuseumPacific

[5] *Pavel Shvaiko and Jérôme Euzenat*. **Ontology matching: state of the art and future challenges**. *2011*. Department of Information Engineering and Computer Science. University of Trento, Italy. http://disi.unitn.it/~p2p/RelatedWork/Matching/SurveyOMtkde_SE.pdf

[6] *Cloudera product documentation*.
http://www.cloudera.com/content/cloudera/en/documentation.html#ClouderaDocumentation

[7] *Francisco Magaz Villaverde, David Isern Alarcón*. **OpenNebula y Hadoop: Cloud Computing con herramientas Open Source**. *Junio 2012*. Universitat Oberta de Catalunya, Área de Inteligencia Artificial, Proyecto final de carrera.

[8] *Sergei O. Kuznetsov and Sergei A. Ob"edkov*. **Comparing Performance of Algorithms for Generating Concept Lattices**. *2002* . All-Russia Institute for Scientific and Technical Information (VINITI), Moscow, Russia. Russian State University for the Humanities, Moscow, Russia. http://ceur-ws.org/Vol-42/paper3_kuznetsov.pdf

[9] *Vicky Choi*. **Faster Algorithms for Constructing a Concept (Galois) Lattice**. *2006* . Department of Computer Science, Virginia Tech, USA. http://people.cs.vt.edu/vchoi/Choi-CLA.pdf

Anexo I. Definiciones, Acrónimos y abreviaturas

- **FCA** . *Formal Concept Analysis.*
- **FFCA**. *Fuzzy Formal Concept Analysis.*
- **IAEA**. *International Atomic Energy Agency.*
- **ETL**. *Extract, Transformation and Load*.
- **Función binaria I de pertenencia**. la entrada para el algoritmo *FCA* es una tabla de datos con filas correspondientes a objetos, columnas correspondientes a atributos y entradas en la tabla donde 'x' indicará si el objeto denotado por la fila actual tiene o no el atributo denotado por la columna. Una tabla de datos como esta puede ser interpretada como una relación binaria $I \subseteq X\,x\,Y$ tal que $\langle x, y\rangle \in I$ sí y solo sí el objeto x tiene el atributo y .
- **Formal context**. En *FCA*, usualmente a la anterior función I de pertenencia se le denomina *formal context*.
- **Formal concept**. En *FCA*, un par $\langle A, B\rangle$ donde $A \subseteq X, B \subseteq Y, A \uparrow = B, B \downarrow = A$ es llamado *formal context* en $I \subseteq X\,x\,Y$ [8]. Intuitivamente sería cada uno de los nodos del *concept lattice*[9].
- **Concept lattice**. Grafo obtenido a partir del conjunto ordenado de *formal concepts* en el cual el elemento inferior tiene un subconjunto de los objetos el concepto superior, pero por el contrario es un superconjunto de atributos respecto a su superior. En este grafo se identifican dos nodos especiales: nodo máximo y nodo mínimo[10].
- **Ontology alignment**. Minería de datos orientada a la correspondencia entre conceptos.
- **Ontology matching**. Encuentra correspondencias entre objetos.
- **Similarity reasoning**. Identificación de conceptos diferentes sintácticamente que son semánticamente cercanos (obtención e integración de información en la web semántica).
- **Map-Reduce**. Patrón de diseño originario del paradigma de programación funcional que divide la ejecución de una cierta unidad de proceso en dos funciones diferenciadas: *map* y *reduce*. La función *map* recibe una clave y un valor, y retorna otra clave y otro valor. Por su parte la función *reduce* recibe en cada invocación la clave, y la lista de valores generada por las ejecuciones de las funciones *map* para esa misma clave. Al estar aislado el proceso de los datos en funciones diferenciadas, estas funciones son susceptibles de ser paralelizadas. Google en primera instancia, y más tarde *frameworks* como *Apache Hadoop* han explotado esta forma de ejecución paralela para poder tratar grandes cantidades de datos.

8 Para una definición más precisa de las funciones $\uparrow\downarrow$ véase 2.2 Fase análisis página 20
9 Véase Ilustración 1: Concept lattice página 10
10 Véase Ilustración 1: Concept lattice página 10 y 1.1 Formal Concept Analysis página 9

Anexo II. Configuración de herramientas utilizadas y guía de operación

Clúster Hadoop

Para poner disponer de un clúster de *Hadoop* de manera rápida y totalmente funcional, tal como se ha reseñado en la sección 3. Plataforma tecnológica página 31, hemos optado por usar la distribución *Hadoop* de *cloudera*.

Concretamente hemos descargado la imagen *cloudera-quickstart-vm-5.3.0-0-virtualbox.7z*, y la hemos importado en *Virtual Box*. *Virtual Box* es un software de virtualización que nos permite ejecutar imágenes de otros sistemas operativos. En nuestro caso estamos corriendo la imagen de *cloudera*, un sistema *linux Cent-OS* de 64 *bits* sobre un sistema anfitrión *Windows7* de 64 *bits* con procesador *Intel Core i7*, con unos recursos asignados de 8*GB* de memoria *RAM* y dos procesadores.

Tras esta operación disponemos de un *cluster* de *Hadoop* totalmente funcional con una instalación de *oozie* lista para ejecutar *jobs* desde el primer instante.

Como se ha comentando con anterioridad, *oozie* es un sistema que nos permite lanzar *workflows* complejos de ejecución, pudiendo combinar *jobs* java estándar, junto con otros *map-reduce* e incluso operaciones de fichero sobre *hdfs*.

Acceso al clúster desde el ordenador anfitrión

Para ejecutar el cliente web, será necesario que la comunicación de red ordenador anfitrión – máquina virtual sea posible. Para ello en la configuración de la máquina virtual de *cloudera*, apartado de red, estableceremos el modo de red del adaptador a "adaptador puente"[11].

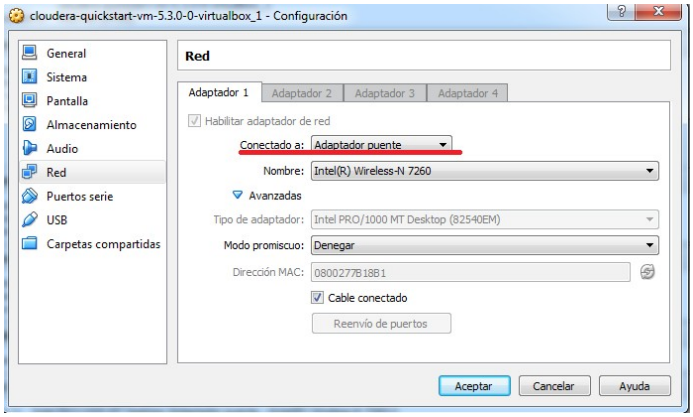

Ilustración 12: Modo de red de la máquina virtual

Repositorio de código fuente

En cuanto al alojamiento del código del proyecto, nos hemos decidido por el control de versiones *git*, dado que ofrece una gran versatilidad; y se dispone de soporte de repositorios gestionados por terceros, tanto gratuitos como de pago.

Concretamente, hemos optado por *bitbucket*; dado que podemos tener la posibilidad de tener repositorios no públicos, a coste cero (gratuito). La *url* del repositorio del proyecto es: https://bitbucket.org/mangelgarciaroig/pfc-fca.

11 *Bridged* en inglés.

Lo primero que deberemos hacer para descargar el proyecto es, autentificados como usuario "cloudera" en la máquina virtual de *VirtualBox*[12], configurar las claves *ssh* para acceder al código del proyecto[13]:

```
[cloudera@localhost ~]$ cp id_rsa  id_rsa.pub $HOME/.ssh
```

Para más información sobre la generación del claves *ssh* para el acceso a *bitbucket*, véase https://confluence.atlassian.com/display/BITBUCKET/Set+up+SSH+for+Git.

Una vez configurado el acceso, con el comando:

```
[cloudera@localhost ~]$ git clone git@bitbucket.org:mangelgarciaroig/pfc-fca.git
```

Tendremos en el directorio *pfc-fca* el código descargado del proyecto.

Puesta a punto de la base de datos

Será necesario configurar adecuadamente la base de datos mysql que ya se haya instalada en el clúster. Se trata de habilitar un esquema para nuestro proyecto, así como crear un usuario con los permisos adecuados. Para ello, ejecutaremos sobre una shell:

```
[cloudera@quickstart ~]$ sudo su -
[root@quickstart ~]# mysql -u root -p
```

(insertar password – 'cloudera' por defecto)

Dentro del entorno de mysql, pasaremos a crear la base de datos y el usuario:

```
mysql> create database fca character set 'utf8' collate 'utf8_unicode_ci';
mysql> use mysql;
mysql> CREATE USER 'fca'@'localhost' IDENTIFIED BY 'fca';
mysql> GRANT ALL PRIVILEGES ON fca.* TO 'fca'@'localhost';
mysql> FLUSH PRIVILEGES;
mysql> quit
```

Una vez creada la base de datos y el usuario 'fca', nos conectaremos usando estas credenciales para crear el esquema (introducir la contraseña 'fca' cuando sea requerido):

```
[cloudera@quickstart scripts]$ mysql -u fca fca -p
mysql> CREATE TABLE object (
    -> id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    -> hadoop_id CHAR(36) NOT NULL UNIQUE
    -> )
    -> Engine=InnoDB;
```

12 Arrancando la máquina virtual, la sesión del usuario "cloudera" es iniciada de manera automática

13 El repositorio es privado; para solicitar acceso se me puede solicitar por correo electrónico a mgarciaroig@uoc.edu. *id_rsa* e *id_rsa.pub* son los ficheros de clave privada y pública respectivamente; que se han registrado en el servicio web de *bitbucket* previamente

```
mysql> CREATE TABLE field (
    -> id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    -> name CHAR(100) NOT NULL UNIQUE
    -> )
    -> Engine=InnoDB;

mysql> CREATE TABLE has_field (
    -> object_id INT NOT NULL,
    -> field_id INT NOT NULL,
    -> value VARCHAR(200) NULL,
    ->
    -> PRIMARY KEY (object_id, field_id),
    -> CONSTRAINT FK_HAS_FIELD_OBJECT FOREIGN KEY (object_id) REFERENCES object(id) ON
DELETE CASCADE,
    -> CONSTRAINT FK_HAS_FIELD_FIELD FOREIGN KEY (field_id) REFERENCES field(id) ON
DELETE CASCADE
    -> )
    -> Engine=InnoDB;

mysql> CREATE TABLE formal_concept (
    -> hash INT NOT NULL PRIMARY KEY,
    -> num_objects INT NOT NULL DEFAULT 0,
    -> num_attbs INT NOT NULL DEFAULT 0,
    -> level INT,
    ->
    -> CONSTRAINT CK_NUM_OBJECTS_POSITIVE CHECK (num_objects >= 0),
    -> CONSTRAINT CK_NUM_ATTBS_POSITIVE CHECK (num_attbs >= 0),
    -> CONSTRAINT CK_LEVEL_POSITIVE CHECK (level IS NULL OR level >= 1)
    -> )
    -> Engine=InnoDB;

mysql> CREATE INDEX IDX_FORMAL_CONCEPT_HASH_OBJECTS ON formal_concept(hash_objects);

mysql> CREATE INDEX IDX_FORMAL_CONCEPT_HASH_ATTBS ON formal_concept(hash_attbs);

mysql> CREATE TABLE attribute (
    -> id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    -> name CHAR(100) NOT NULL UNIQUE
    -> )
    -> Engine=InnoDB;

mysql> CREATE INDEX IDX_ATRIBUTE_NAME on attribute(name);

mysql> CREATE TABLE has_object (
```

```
    -> formal_concept_id INT NOT NULL,
    -> object_id INT NOT NULL,
    -> PRIMARY KEY (formal_concept_id, object_id),
    -> CONSTRAINT FK_HAS_OBJECT_FORMAL_CONCEPT FOREIGN KEY (formal_concept_id) REFERENCES
formal_concept(hash) ON DELETE CASCADE,
    -> CONSTRAINT FK_HAS_OBJECT_OBJECT FOREIGN KEY (object_id) REFERENCES object(id) ON
DELETE CASCADE)
    -> Engine=InnoDB;


mysql> CREATE TABLE has_attb (
    -> formal_concept_id INT NOT NULL,
    -> attb_id INT NOT NULL,
    -> PRIMARY KEY (formal_concept_id, attb_id),
    -> CONSTRAINT FK_HAS_ATTB_FORMAL_CONCEPT FOREIGN KEY (formal_concept_id) REFERENCES
formal_concept(hash) ON DELETE CASCADE,
    -> CONSTRAINT FK_HAS_ATTB_ATTRIBUTE FOREIGN KEY (attb_id) REFERENCES attribute(id) ON
DELETE CASCADE)
    -> Engine=InnoDB;


mysql> CREATE TABLE formal_concept_hierarchy (
    -> formal_concept_id INT NOT NULL,
    -> parent_id INT NOT NULL,
    ->
    -> PRIMARY KEY (formal_concept_id,parent_id),
    -> CONSTRAINT FK_FORMAL_CONCEPT_HIERARCHY_CONCEPT FOREIGN KEY (formal_concept_id)
REFERENCES formal_concept(hash) ON DELETE CASCADE,
    -> CONSTRAINT FK_FORMAL_CONCEPT_HIERARCHY_PARENT FOREIGN KEY (parent_id) REFERENCES
formal_concept(hash) ON DELETE CASCADE,
    -> CONSTRAINT CK_FORMAL_CONCEPT_HIERARCHY_NOT_POINT_THE_SAME CHECK (formal_concept_id
<> parent_id)
    -> )
    -> Engine=InnoDB;
```

Deploy de la aplicación en el clúster Hadoop

Sobre el directorio raíz del proyecto, generamos un *bundle* de instalación con el comando:

```
[cloudera@localhost pfc-fca]$ mvn clean package
```

Después de completarse de este comando, dispondremos del archivo `mgarciaroig-uoc-pfc-fca-1.0-bin.tar.gz` dentro del directorio fca-backend-app/target; con lo cual procederemos a descomprimir los ficheros:

```
[cloudera@localhost target]$ tar zxvf mgarciaroig-uoc-pfc-fca-1.0-bin.tar.gz
```

Esto creará el directorio `mgarciaroig-uoc-pfc-fca-1.0`. Para instalar la aplicación tan sólo nos quedará ya ejecutar el *script* `deploy.sh`:

```
[cloudera@localhost target]$ cd mgarciaroig-uoc-pfc-fca-1.0-SNAPSHOT/deploy/
[cloudera@localhost deploy]$ ./deploy.sh
```

La última acción creará las siguientes rutas sobre el sistema de archivos *HDFS* del clúster:

- */user/cloudera/app/oozie* (configuración del workflow a ejecutar)
- */user/cloudera/share/lib* (librerías con código java ejecutable)

Seguidamente agregaremos algunas librerías adicionales al *classpath* de la instalación de *Hadoop*, necesarias para que funcionen correctamente algunas utilidades adicionales que se han desarrollado; y que pasaremos a comentar más adelante.

```
[cloudera@localhost deploy]$  cp ../lib/*mahout*.jar /usr/lib/hadoop/lib

[cloudera@localhost deploy]$  cp ../lib/commons-lang3*.jar /usr/lib/hadoop/lib
```

Ejecución de la fase ETL

Para ejecutar la parte previa de la aplicación, los procesos de carga y transformación de datos (ETL), nos situamos el la carpeta donde hemos descomprimido el *bundle* de la aplicación, y ejecutamos el *script* `etl.sh`:

```
[cloudera@localhost pfc-fca]$ cd target/mgarciaroig-uoc-pfc-fca-1.0/scripts/

[cloudera@localhost scripts]$ ./etl.sh
```

Después de completarse la ejecución de los diferentes jobs, tendremos disponibles las siguientes rutas sobre el sistema de archivos *HDFS* del clúster:

- */user/cloudera/etl/import* Datos consolidados a formato "normalizado de *Excel*" (una fila por objeto, y una columna por cada atributo).
- */user/cloudera/etl/converted* Datos convertidos a fichero con formato de secuencia apto para ser tratado mediante la implementación del algoritmo *map-reduce* de *Hadoop.*
- */user/cloudera/etl/enumerations* Ficheros por cada campo de tipo enumerado; donde se mapea cada posible valor con un código numérico.
- */user/cloudera/etl/discretized* Ficheros por cada campo de tipo enumerado; donde se mapea cada posible valor con un código numérico.
- */user/cloudera/etl/discretization* Fichero de secuencia *HDFS* con los campos de tipo enumerado con valor codificado en forma numérica.
- */user/cloudera/etl/mahout-input* Ficheros binarios de secuencia *HDFS* con los valores de los campos a clusterizar mediante *KMeans*, en formato apto para ser tratados mediante *Mahout.*
- */user/cloudera/etl/formal_context* Ficheros binarios de secuencia *HDFS* de *Hadoop* con el *formal context* generado. Esto es, en formato "nombre de campo" - "booleano"; que indica si el flag está activo para el registro actual.

Ejecución de la fase de análisis (generación formal concepts)

Para poder generar el resultado de la parte de análisis o generación de los *formal concepts*, nos situamos en la carpeta donde hemos descomprimido el *bundle* de la aplicación, y ejecutamos el *script* `analysis.sh`:

```
[cloudera@localhost pfc-fca]$ cd target/mgarciaroig-uoc-pfc-fca-1.0/scripts/

[cloudera@localhost scripts]$ ./analysis.sh
```

Después de completarse la ejecución de los diferentes jobs, tendremos disponibles las siguientes rutas sobre el sistema de archivos *HDFS* del clúster[14]:

- */user/cloudera/analysis/data_preparation* Ficheros con objetos indexados por atributo, para

14 Este proceso puede ser muy costoso. Con el hardaware utilizado en el desarrollo del presente proyecto, el proceso tardó en completarse varios días

acelerar la búsqueda de objetos compartiendo un atributo específico durante la generación de los *formal concepts*.

- */user/cloudera/analysis/fca/max_concept/max_concept.seq* Fichero con los datos del nodo máximo (*maximum formal concept*), que constituye los datos de entrada iniciales del algoritmo para el cálculo del resto de *formal concepts*.
- */user/cloudera/analysis/fca/min_concept/min_concept.seq* Fichero con los datos persistidos del nodo mínimo (*minimum formal concept*), esto es, *formal concept* con todos los atributos y ninguno de los objetos. Es necesario crear específicamente estos datos; ya que se trata de un caso especial que no es generado durante la ejecución iterativa del algoritmo.
- */user/cloudera/analysis/fca/iterations/iteration-N* Ficheros con los datos persistidos de los *formal concepts* generados durante la iteración enésima.

Preparación para la fase de interfaz de usuario (export de datos)

Para poder exportar todos los datos generados a la base de datos, y generar el *concept lattice* final, deberemos ejecutar el *script* de exportación. Nos situamos en la carpeta donde hemos descomprimido el *bundle* de la aplicación, y ejecutamos el *script* `export.sh`:

```
[cloudera@localhost pfc-fca]$ cd target/mgarciaroig-uoc-pfc-fca-1.0/scripts/
[cloudera@localhost scripts]$ ./export.sh
```

Después de completarse la ejecución de los diferentes jobs, tendremos información disponible en las siguientes tablas:

- *Tablas object, field, has_field* Toda la información de los reactores, tal cual fue definida por la *IAEA*.
- *Tablas attribute, has_attb* Todos los datos relativos al *formal context* generado.
- *Tabla formal_concept* Información de los conceptos formales generados durante la fase de análisis.

Tan sólo nos faltará la ejecución del script que construir el concept lattice en sí mismo:

```
[cloudera@localhost scripts]$ ./buildConceptLattice.sh
```

De esta forma, los datos de la última tabla habrán sido generados:

- *Tabla formal_concept_hierarchy* Relación jerárquica entre los *formal concepts* que constituye lo que se denomina *concept lattice*.

Tras haberse generado todos estos datos, la aplicación cliente está lista para ser ejecutada.

Ejecución del servicio web

El cliente web final debe ser ejecutado desde el ordenador anfitrión, no desde la máquina virtual[15]. Esto es así ya que la parte web utiliza algunas etiquetas *html5* que no están soportadas por la versión de *mozilla firefox* que viene con la máquina virtual *Linux* de *cloudera*.

Como ya hemos comentado en la sección "Acceso al clúster desde el ordenador anfitrión", la comunicación entre anfitrión y huésped debe estar habilitada. Una vez realizado esto, desde la máquina virtual averiguamos la dirección *ip* asignada a nuestra red a la máquina virtual:

```
[cloudera@quickstart ~]$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:7B:18:B1
          inet addr:192.168.1.131  Bcast:192.168.1.255  Mask:255.255.255.0
```

15 El navegador donde ha sido testeada la aplicación, y donde la experiencia de usuario es la óptima, es *google chrome*

```
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:442275 errors:0 dropped:0 overruns:0 frame:0
          TX packets:641158 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:253877412 (242.1 MiB)  TX bytes:768248073 (732.6 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:763855 errors:0 dropped:0 overruns:0 frame:0
          TX packets:763855 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:543136978 (517.9 MiB)  TX bytes:543136978 (517.9 MiB)
```

Tras averiguar la dirección, será necesario habitar el acceso a base de datos desde el ordenador anfitrión. Para ello, y de manera análoga a lo realizado en "Puesta a punto de la base de datos", haremos:

```
mysql> CREATE USER 'fca'@'192.168.1.131' IDENTIFIED BY 'fca';
mysql> GRANT ALL PRIVILEGES ON fca.* TO 'fca'@'192.168.1.131';
mysql> FLUSH PRIVILEGES;
```

Sólo nos resta descargar el código del proyecto en el ordenador anfitrión, y editar la propiedad *databaseConnString* del fichero *app.properties* apropiadamente. En nuestro caso:

```
databaseConnString=jdbc:mysql://192.168.1.131/fca
```

Desde este punto, para arrancar el servicio web, desde una consola, ejecutamos en el ordenador anfitrión[16]:

```
C:\Users\mangel\Documents\dev\code\pfc-fca>cd fca-web-app
C:\Users\mangel\Documents\dev\code\pfc-fca\fca-web-app>mvn clean tomcat:run-war
```

Tras esto, abrimos un navegador web (se recomienda *google chrome*), y vamos a la *url*:

```
http://localhost:8080/pfc-fca-web/
```

Tools adicionales para depurar los ficheros generados

Con el objetivo de ser capaces de depurar las diferentes salidas de las tareas ejecutadas durante la fase de la ETL, se han desarrollado las siguientes utilidades (todas ellas disponibles bajo la sub-carpeta *scripts* :

`etlFileViewerTool.sh [ruta_hdfs_fichero_generado]`

Esta utilidad permite visualizar el contenido de cualquier fichero de secuencia generado.

16 Será necesario tener instalado y configurado *maven* en el ordenador anfitrión, tal cómo se describe en https://maven.apache.org/install.html

Por ejemplo:

```
[cloudera@localhost scripts]$ ./etlFileViewerTool.sh /user/cloudera/etl/discretized/part-m-00000


[ (null) ]
'Category.category': '2'
'Experimental.horizontal_channels': '2'
'Experimental.horizontal_max_flux': '6.0E11'
…
```

mahoutInputFileViewerTool.sh [ruta_hdfs_fichero_generado]

Esta utilidad permite visualizar el contenido de los ficheros que toma *Mahout* como entrada para clusterizar los valores mediante *K-Means*.

Por ejemplo:

```
[cloudera@localhost scripts]$ ./mahoutInputFileViewerTool.sh /user/cloudera/etl/mahout input/Experimental.horizontal_channels-r-00000


{0:3.0}
==========
{0:8.0}
==========
{0:10.0}
…
```

mahoutOutputFileViewerTool.sh [ruta_hdfs_fichero_generado]

Por último, esta utilidad nos permitirá visualizar el contenido de la clusterización *K-Means* realizada por *Mahout*.

Por ejemplo:

```
[cloudera@localhost scripts]$ $ ./mahoutOutputFileViewerTool.sh /user/cloudera/etl/mahout-output/kmeans/Experimental.horizontal_channels-kmeans/clusteredPoints/part-m-00000


'wt: 1.0 distance: 0.2692307692307684  vec: [3.000]' belongs to cluster '1'
'wt: 1.0 distance: 0.6111111111110985  vec: [8.000]' belongs to cluster '38'
'wt: 1.0 distance: 1.3888888888888788  vec: [10.000]' belongs to cluster '38'
...
```

Anexo III. Detalle de *formal context* generado

Como hemos comentado anteriormente, hemos generado la información relativa al *formal context* como un fichero de secuencia *HDFS* donde cada registro es un mapa cuya clave será el nombre del campo generado; y el valor asociado será un valor booleano que indica si el objeto tiene el atributo o no.

Se muestra a continuación a modo de ejemplo únicamente la información relativa al primer registro; dado que hay bastante información, en aras de ilustrar que la generación del *concept lattice* en general y del *formal context* en particular, es un proceso que puede conllevar el manejo de gran cantidad de datos. El volumen de elementos generados es de 150 (es decir, un objeto por reactor a analizar). Lo que es elevado es el número de atributos generados por registro que es de 547.

```
[ 64ed3be0-4a59-4ca2-8f50-c282cf1ca774 ]
'Category.category-ENUM-000': 'false'
'Category.category-ENUM-001': 'false'
'Category.category-ENUM-002': 'true'
'Category.category-ENUM-003': 'false'
'Category.category-ENUM-004': 'false'
'Category.category-ENUM-005': 'false'
'Category.category-ENUM-006': 'false'
'Category.category-ENUM-007': 'false'
'Experimental.horizontal_channels-CLUSTER-001': 'true'
'Experimental.horizontal_channels-CLUSTER-007': 'false'
'Experimental.horizontal_channels-CLUSTER-038': 'false'
'Experimental.horizontal_channels-CLUSTER-UND': 'false'
'Experimental.horizontal_max_flux-CLUSTER-015': 'false'
'Experimental.horizontal_max_flux-CLUSTER-023': 'true'
'Experimental.horizontal_max_flux-CLUSTER-035': 'false'
'Experimental.horizontal_max_flux-CLUSTER-UND': 'false'
'Experimental.incore_irradiation_facilities-CLUSTER-006': 'false'
'Experimental.incore_irradiation_facilities-CLUSTER-031': 'false'
'Experimental.incore_irradiation_facilities-CLUSTER-040': 'true'
'Experimental.incore_irradiation_facilities-CLUSTER-UND': 'false'
'Experimental.incore_max_flux-CLUSTER-003': 'false'
'Experimental.incore_max_flux-CLUSTER-012': 'true'
'Experimental.incore_max_flux-CLUSTER-015': 'false'
'Experimental.incore_max_flux-CLUSTER-UND': 'false'
'Experimental.irradiation_channels-CLUSTER-000': 'true'
'Experimental.irradiation_channels-CLUSTER-003': 'false'
'Experimental.irradiation_channels-CLUSTER-007': 'false'
'Experimental.irradiation_channels-CLUSTER-UND': 'false'
'Experimental.irradiation_number-CLUSTER-006': 'true'
'Experimental.irradiation_number-CLUSTER-014': 'false'
'Experimental.irradiation_number-CLUSTER-018': 'false'
```

```
'Experimental.irradiation_number-CLUSTER-UND': 'false'
'Experimental.reflector_max_flux-CLUSTER-001': 'false'
'Experimental.reflector_max_flux-CLUSTER-003': 'false'
'Experimental.reflector_max_flux-CLUSTER-004': 'false'
'Experimental.reflector_max_flux-CLUSTER-UND': 'true'
'Experimental.use-ENUM-000': 'false'
'Experimental.use-ENUM-001': 'false'
'Experimental.use-ENUM-002': 'false'
'Experimental.use-ENUM-003': 'false'
'Experimental.use-ENUM-004': 'false'
'Experimental.use-ENUM-005': 'false'
'Experimental.use-ENUM-006': 'false'
'Experimental.use-ENUM-007': 'true'
'Experimental.use-ENUM-008': 'false'
'Experimental.use-ENUM-009': 'false'
'Experimental.use-ENUM-010': 'false'
'Experimental.use-ENUM-011': 'false'
'Experimental.use-ENUM-012': 'false'
'Experimental.use-ENUM-013': 'false'
'Experimental.use-ENUM-014': 'false'
'Experimental.use-ENUM-015': 'false'
'Experimental.use-ENUM-016': 'false'
'Experimental.use-ENUM-017': 'false'
'Experimental.use-ENUM-018': 'false'
'Experimental.use-ENUM-019': 'false'
'Experimental.use-ENUM-020': 'false'
'Experimental.use-ENUM-021': 'false'
'Experimental.use-ENUM-022': 'false'
'Experimental.use-ENUM-023': 'false'
'Experimental.use-ENUM-024': 'false'
'Experimental.use-ENUM-025': 'false'
'Experimental.use-ENUM-026': 'false'
'Experimental.use-ENUM-027': 'false'
'Experimental.use-ENUM-028': 'false'
'Experimental.use-ENUM-029': 'false'
'Experimental.use-ENUM-030': 'false'
'Experimental.use-ENUM-031': 'false'
'Experimental.use-ENUM-032': 'false'
'Experimental.use-ENUM-033': 'true'
'Experimental.use-ENUM-034': 'false'
'Experimental.use-ENUM-035': 'false'
'Experimental.use-ENUM-036': 'false'
'Experimental.use-ENUM-037': 'false'
```

```
'Experimental.use-ENUM-038': 'false'
'Experimental.use-ENUM-039': 'false'
'Experimental.use-ENUM-040': 'false'
'Experimental.use-ENUM-041': 'false'
'Experimental.use-ENUM-042': 'false'
'Experimental.use-ENUM-043': 'false'
'Experimental.use-ENUM-044': 'false'
'Experimental.use-ENUM-045': 'false'
'Experimental.use-ENUM-046': 'false'
'Experimental.use-ENUM-047': 'false'
'Experimental.vertical_channels-CLUSTER-000': 'false'
'Experimental.vertical_channels-CLUSTER-011': 'true'
'Experimental.vertical_channels-CLUSTER-043': 'false'
'Experimental.vertical_channels-CLUSTER-UND': 'false'
'Experimental.vertical_max_flux-CLUSTER-004': 'true'
'Experimental.vertical_max_flux-CLUSTER-005': 'false'
'Experimental.vertical_max_flux-CLUSTER-025': 'false'
'Experimental.vertical_max_flux-CLUSTER-UND': 'false'
'Information.construction-CLUSTER-044': 'false'
'Information.construction-CLUSTER-061': 'true'
'Information.construction-CLUSTER-097': 'false'
'Information.construction-CLUSTER-102': 'false'
'Information.construction-CLUSTER-UND': 'false'
'Information.criticality-CLUSTER-007': 'false'
'Information.criticality-CLUSTER-036': 'true'
'Information.criticality-CLUSTER-063': 'false'
'Information.criticality-CLUSTER-107': 'false'
'Information.criticality-CLUSTER-UND': 'false'
'Information.date-CLUSTER-050': 'false'
'Information.date-CLUSTER-083': 'true'
'Information.date-CLUSTER-095': 'false'
'Information.date-CLUSTER-107': 'false'
'Information.date-CLUSTER-UND': 'false'
'Information.operators-CLUSTER-011': 'true'
'Information.operators-CLUSTER-058': 'false'
'Information.operators-CLUSTER-104': 'false'
'Information.operators-CLUSTER-UND': 'false'
'Information.safeguards-ENUM-000': 'false'
'Information.safeguards-ENUM-001': 'false'
'Information.safeguards-ENUM-002': 'false'
'Information.safeguards-ENUM-003': 'false'
'Information.safeguards-ENUM-004': 'true'
'Information.safeguards-ENUM-005': 'false'
```

```
'Information.safeguards-ENUM-006': 'false'
'Information.safeguards-ENUM-007': 'false'
'Information.safeguards-ENUM-008': 'false'
'Information.safeguards-ENUM-009': 'false'
'Information.safeguards-ENUM-010': 'false'
'Information.total_staff-CLUSTER-004': 'false'
'Information.total_staff-CLUSTER-034': 'true'
'Information.total_staff-CLUSTER-107': 'false'
'Information.total_staff-CLUSTER-UND': 'false'
'Location.city-ENUM-000': 'false'
'Location.city-ENUM-001': 'false'
'Location.city-ENUM-002': 'false'
'Location.city-ENUM-003': 'false'
'Location.city-ENUM-004': 'false'
'Location.city-ENUM-005': 'false'
'Location.city-ENUM-006': 'false'
'Location.city-ENUM-007': 'false'
'Location.city-ENUM-008': 'false'
'Location.city-ENUM-009': 'false'
'Location.city-ENUM-010': 'false'
'Location.city-ENUM-011': 'false'
'Location.city-ENUM-012': 'false'
'Location.city-ENUM-013': 'false'
'Location.city-ENUM-014': 'false'
'Location.city-ENUM-015': 'false'
'Location.city-ENUM-016': 'false'
'Location.city-ENUM-017': 'false'
'Location.city-ENUM-018': 'false'
'Location.city-ENUM-019': 'false'
'Location.city-ENUM-020': 'false'
'Location.city-ENUM-021': 'false'
'Location.city-ENUM-022': 'false'
'Location.city-ENUM-023': 'false'
'Location.city-ENUM-024': 'false'
'Location.city-ENUM-025': 'false'
'Location.city-ENUM-026': 'false'
'Location.city-ENUM-027': 'false'
'Location.city-ENUM-028': 'false'
'Location.city-ENUM-029': 'false'
'Location.city-ENUM-030': 'false'
'Location.city-ENUM-031': 'false'
'Location.city-ENUM-032': 'false'
'Location.city-ENUM-033': 'false'
```

'Location.city-ENUM-034': 'false'
'Location.city-ENUM-035': 'false'
'Location.city-ENUM-036': 'false'
'Location.city-ENUM-037': 'false'
'Location.city-ENUM-038': 'false'
'Location.city-ENUM-039': 'false'
'Location.city-ENUM-040': 'false'
'Location.city-ENUM-041': 'false'
'Location.city-ENUM-042': 'false'
'Location.city-ENUM-043': 'false'
'Location.city-ENUM-044': 'false'
'Location.city-ENUM-045': 'false'
'Location.city-ENUM-046': 'false'
'Location.city-ENUM-047': 'false'
'Location.city-ENUM-048': 'false'
'Location.city-ENUM-049': 'false'
'Location.city-ENUM-050': 'false'
'Location.city-ENUM-051': 'false'
'Location.city-ENUM-052': 'false'
'Location.city-ENUM-053': 'false'
'Location.city-ENUM-054': 'false'
'Location.city-ENUM-055': 'false'
'Location.city-ENUM-056': 'false'
'Location.city-ENUM-057': 'false'
'Location.city-ENUM-058': 'false'
'Location.city-ENUM-059': 'false'
'Location.city-ENUM-060': 'false'
'Location.city-ENUM-061': 'false'
'Location.city-ENUM-062': 'false'
'Location.city-ENUM-063': 'false'
'Location.city-ENUM-064': 'false'
'Location.city-ENUM-065': 'false'
'Location.city-ENUM-066': 'false'
'Location.city-ENUM-067': 'false'
'Location.city-ENUM-068': 'false'
'Location.city-ENUM-069': 'false'
'Location.city-ENUM-070': 'false'
'Location.city-ENUM-071': 'false'
'Location.city-ENUM-072': 'true'
'Location.city-ENUM-073': 'false'
'Location.city-ENUM-074': 'false'
'Location.city-ENUM-075': 'false'
'Location.city-ENUM-076': 'false'

```
'Location.city-ENUM-077': 'false'
'Location.country-ENUM-000': 'false'
'Location.country-ENUM-001': 'false'
'Location.country-ENUM-002': 'false'
'Location.country-ENUM-003': 'false'
'Location.country-ENUM-004': 'false'
'Location.country-ENUM-005': 'false'
'Location.country-ENUM-006': 'false'
'Location.country-ENUM-007': 'false'
'Location.country-ENUM-008': 'false'
'Location.country-ENUM-009': 'true'
'Location.country-ENUM-010': 'false'
'Location.country-ENUM-011': 'false'
'Location.country-ENUM-012': 'false'
'Location.country-ENUM-013': 'false'
'Location.country-ENUM-014': 'false'
'Location.country-ENUM-015': 'false'
'Location.country-ENUM-016': 'false'
'Location.country-ENUM-017': 'false'
'Location.country-ENUM-018': 'false'
'Location.country-ENUM-019': 'false'
'Location.country-ENUM-020': 'false'
'Location.country-ENUM-021': 'false'
'Location.country-ENUM-022': 'false'
'Location.country-ENUM-023': 'false'
'Location.country-ENUM-024': 'false'
'Location.country-ENUM-025': 'false'
'Location.country-ENUM-026': 'false'
'Location.country-ENUM-027': 'false'
'Location.country-ENUM-028': 'false'
'Location.country-ENUM-029': 'false'
'Location.country-ENUM-030': 'false'
'Location.country-ENUM-031': 'false'
'Location.country-ENUM-032': 'false'
'Status.status-ENUM-000': 'false'
'Status.status-ENUM-001': 'false'
'Status.status-ENUM-002': 'false'
'Status.status-ENUM-003': 'true'
'Status.status-ENUM-004': 'false'
'Status.status-ENUM-005': 'false'
'Status.status-ENUM-006': 'false'
'TechData.control_rods_material-ENUM-000': 'false'
'TechData.control_rods_material-ENUM-001': 'true'
```

'TechData.control_rods_material-ENUM-002': 'false'
'TechData.control_rods_material-ENUM-003': 'false'
'TechData.control_rods_material-ENUM-004': 'false'
'TechData.control_rods_material-ENUM-005': 'false'
'TechData.control_rods_material-ENUM-006': 'false'
'TechData.control_rods_material-ENUM-007': 'false'
'TechData.control_rods_material-ENUM-008': 'false'
'TechData.control_rods_material-ENUM-009': 'false'
'TechData.control_rods_material-ENUM-010': 'false'
'TechData.control_rods_material-ENUM-011': 'false'
'TechData.control_rods_material-ENUM-012': 'false'
'TechData.control_rods_material-ENUM-013': 'false'
'TechData.control_rods_material-ENUM-014': 'false'
'TechData.control_rods_material-ENUM-015': 'false'
'TechData.control_rods_material-ENUM-016': 'false'
'TechData.control_rods_material-ENUM-017': 'false'
'TechData.control_rods_material-ENUM-018': 'false'
'TechData.control_rods_number-CLUSTER-030': 'false'
'TechData.control_rods_number-CLUSTER-049': 'false'
'TechData.control_rods_number-CLUSTER-052': 'true'
'TechData.control_rods_number-CLUSTER-UND': 'false'
'TechData.coolant_material-ENUM-000': 'false'
'TechData.coolant_material-ENUM-001': 'false'
'TechData.coolant_material-ENUM-002': 'true'
'TechData.coolant_material-ENUM-003': 'false'
'TechData.coolant_material-ENUM-004': 'false'
'TechData.coolant_material-ENUM-005': 'false'
'TechData.coolant_material-ENUM-006': 'false'
'TechData.coolant_material-ENUM-007': 'false'
'TechData.coolant_material-ENUM-008': 'false'
'TechData.coolant_material-ENUM-009': 'false'
'TechData.coolant_material-ENUM-010': 'false'
'TechData.coolant_material-ENUM-011': 'false'
'TechData.coolant_material-ENUM-012': 'false'
'TechData.cooling_forced-FALSE': 'false'
'TechData.cooling_forced-TRUE': 'true'
'TechData.cooling_forced-UND': 'false'
'TechData.cooling_natural_convection-FALSE': 'true'
'TechData.cooling_natural_convection-TRUE': 'false'
'TechData.cooling_natural_convection-UND': 'false'
'TechData.max_flux_pulsed_fast-CLUSTER-001': 'false'
'TechData.max_flux_pulsed_fast-CLUSTER-002': 'false'
'TechData.max_flux_pulsed_fast-CLUSTER-003': 'false'

'TechData.max_flux_pulsed_fast-CLUSTER-UND': 'true'
'TechData.max_flux_pulsed_thermal-CLUSTER-000': 'false'
'TechData.max_flux_pulsed_thermal-CLUSTER-004': 'false'
'TechData.max_flux_pulsed_thermal-CLUSTER-005': 'false'
'TechData.max_flux_pulsed_thermal-CLUSTER-UND': 'true'
'TechData.max_flux_steady_fast-CLUSTER-004': 'false'
'TechData.max_flux_steady_fast-CLUSTER-030': 'false'
'TechData.max_flux_steady_fast-CLUSTER-060': 'true'
'TechData.max_flux_steady_fast-CLUSTER-UND': 'false'
'TechData.max_flux_steady_thermal-CLUSTER-001': 'false'
'TechData.max_flux_steady_thermal-CLUSTER-022': 'true'
'TechData.max_flux_steady_thermal-CLUSTER-038': 'false'
'TechData.max_flux_steady_thermal-CLUSTER-UND': 'false'
'TechData.moderator_material-ENUM-000': 'false'
'TechData.moderator_material-ENUM-001': 'true'
'TechData.moderator_material-ENUM-002': 'false'
'TechData.moderator_material-ENUM-003': 'false'
'TechData.moderator_material-ENUM-004': 'false'
'TechData.moderator_material-ENUM-005': 'false'
'TechData.moderator_material-ENUM-006': 'false'
'TechData.moderator_material-ENUM-007': 'false'
'TechData.moderator_material-ENUM-008': 'false'
'TechData.moderator_material-ENUM-009': 'false'
'TechData.moderator_material-ENUM-010': 'false'
'TechData.moderator_material-ENUM-011': 'false'
'TechData.reactor_type-ENUM-000': 'false'
'TechData.reactor_type-ENUM-001': 'false'
'TechData.reactor_type-ENUM-002': 'false'
'TechData.reactor_type-ENUM-003': 'false'
'TechData.reactor_type-ENUM-004': 'false'
'TechData.reactor_type-ENUM-005': 'false'
'TechData.reactor_type-ENUM-006': 'false'
'TechData.reactor_type-ENUM-007': 'true'
'TechData.reactor_type-ENUM-008': 'false'
'TechData.reactor_type-ENUM-009': 'false'
'TechData.reactor_type-ENUM-010': 'false'
'TechData.reactor_type-ENUM-011': 'false'
'TechData.reactor_type-ENUM-012': 'false'
'TechData.reactor_type-ENUM-013': 'false'
'TechData.reactor_type-ENUM-014': 'false'
'TechData.reactor_type-ENUM-015': 'false'
'TechData.reactor_type-ENUM-016': 'false'
'TechData.reactor_type-ENUM-017': 'false'

'TechData.reactor_type-ENUM-018': 'false'
'TechData.reactor_type-ENUM-019': 'false'
'TechData.reactor_type-ENUM-020': 'false'
'TechData.reactor_type-ENUM-021': 'false'
'TechData.reactor_type-ENUM-022': 'false'
'TechData.reactor_type-ENUM-023': 'false'
'TechData.reactor_type-ENUM-024': 'false'
'TechData.reactor_type-ENUM-025': 'false'
'TechData.reactor_type-ENUM-026': 'false'
'TechData.reactor_type-ENUM-027': 'false'
'TechData.reactor_type-ENUM-028': 'false'
'TechData.reactor_type-ENUM-029': 'false'
'TechData.reactor_type-ENUM-030': 'false'
'TechData.reactor_type-ENUM-031': 'false'
'TechData.reactor_type-ENUM-032': 'false'
'TechData.reactor_type-ENUM-033': 'false'
'TechData.reactor_type-ENUM-034': 'false'
'TechData.reactor_type-ENUM-035': 'false'
'TechData.reactor_type-ENUM-036': 'false'
'TechData.reactor_type-ENUM-037': 'false'
'TechData.reactor_type-ENUM-038': 'false'
'TechData.reactor_type-ENUM-039': 'false'
'TechData.reactor_type-ENUM-040': 'false'
'TechData.reflector_material-ENUM-000': 'false'
'TechData.reflector_material-ENUM-001': 'false'
'TechData.reflector_material-ENUM-002': 'true'
'TechData.reflector_material-ENUM-003': 'false'
'TechData.reflector_material-ENUM-004': 'false'
'TechData.reflector_material-ENUM-005': 'false'
'TechData.reflector_material-ENUM-006': 'false'
'TechData.reflector_material-ENUM-007': 'false'
'TechData.reflector_material-ENUM-008': 'false'
'TechData.reflector_material-ENUM-009': 'false'
'TechData.reflector_material-ENUM-010': 'false'
'TechData.reflector_material-ENUM-011': 'false'
'TechData.reflector_material-ENUM-012': 'false'
'TechData.reflector_material-ENUM-013': 'false'
'TechData.reflector_material-ENUM-014': 'false'
'TechData.reflector_material-ENUM-015': 'false'
'TechData.reflector_material-ENUM-016': 'false'
'TechData.reflector_material-ENUM-017': 'false'
'TechData.sites_number-CLUSTER-005': 'false'
'TechData.sites_number-CLUSTER-011': 'true'

```
'TechData.sites_number-CLUSTER-030': 'false'
'TechData.sites_number-CLUSTER-UND': 'false'
'TechData.thermal_power_pulsed_mw-CLUSTER-019': 'false'
'TechData.thermal_power_pulsed_mw-CLUSTER-080': 'true'
'TechData.thermal_power_pulsed_mw-CLUSTER-086': 'false'
'TechData.thermal_power_pulsed_mw-CLUSTER-UND': 'false'
'TechData.thermal_power_steady_kw-CLUSTER-099': 'false'
'TechData.thermal_power_steady_kw-CLUSTER-115': 'true'
'TechData.thermal_power_steady_kw-CLUSTER-118': 'false'
'TechData.thermal_power_steady_kw-CLUSTER-UND': 'false'
'Utilization.days_per_week-CLUSTER-022': 'false'
'Utilization.days_per_week-CLUSTER-074': 'false'
'Utilization.days_per_week-CLUSTER-096': 'false'
'Utilization.days_per_week-CLUSTER-101': 'true'
'Utilization.days_per_week-CLUSTER-UND': 'false'
'Utilization.experimenters_number-CLUSTER-011': 'false'
'Utilization.experimenters_number-CLUSTER-013': 'true'
'Utilization.experimenters_number-CLUSTER-017': 'false'
'Utilization.experimenters_number-CLUSTER-UND': 'false'
'Utilization.geochronology_methods-ENUM-000': 'true'
'Utilization.geochronology_methods-ENUM-001': 'false'
'Utilization.geochronology_samples_per_year-CLUSTER-000': 'false'
'Utilization.geochronology_samples_per_year-CLUSTER-001': 'false'
'Utilization.geochronology_samples_per_year-CLUSTER-UND': 'true'
'Utilization.hours_per_day-CLUSTER-013': 'false'
'Utilization.hours_per_day-CLUSTER-020': 'true'
'Utilization.hours_per_day-CLUSTER-059': 'false'
'Utilization.hours_per_day-CLUSTER-099': 'false'
'Utilization.hours_per_day-CLUSTER-UND': 'false'
'Utilization.isotope_total_activity_per_year-CLUSTER-003': 'false'
'Utilization.isotope_total_activity_per_year-CLUSTER-004': 'false'
'Utilization.isotope_total_activity_per_year-CLUSTER-005': 'false'
'Utilization.isotope_total_activity_per_year-CLUSTER-UND': 'true'
'Utilization.isotopes-ENUM-000': 'true'
'Utilization.isotopes-ENUM-001': 'false'
'Utilization.isotopes-ENUM-002': 'false'
'Utilization.isotopes-ENUM-003': 'false'
'Utilization.isotopes-ENUM-004': 'false'
'Utilization.isotopes-ENUM-005': 'false'
'Utilization.isotopes-ENUM-006': 'false'
'Utilization.isotopes-ENUM-007': 'false'
'Utilization.isotopes-ENUM-008': 'false'
'Utilization.isotopes-ENUM-009': 'false'
```

```
'Utilization.isotopes-ENUM-010': 'false'
'Utilization.isotopes-ENUM-011': 'false'
'Utilization.isotopes-ENUM-012': 'false'
'Utilization.isotopes-ENUM-013': 'false'
'Utilization.isotopes-ENUM-014': 'false'
'Utilization.isotopes-ENUM-015': 'false'
'Utilization.isotopes-ENUM-016': 'false'
'Utilization.isotopes-ENUM-017': 'false'
'Utilization.isotopes-ENUM-018': 'false'
'Utilization.isotopes-ENUM-019': 'false'
'Utilization.isotopes-ENUM-020': 'false'
'Utilization.isotopes-ENUM-021': 'false'
'Utilization.isotopes-ENUM-022': 'false'
'Utilization.isotopes-ENUM-023': 'false'
'Utilization.isotopes-ENUM-024': 'false'
'Utilization.isotopes-ENUM-025': 'false'
'Utilization.isotopes-ENUM-026': 'false'
'Utilization.isotopes-ENUM-027': 'false'
'Utilization.isotopes-ENUM-028': 'false'
'Utilization.isotopes-ENUM-029': 'false'
'Utilization.isotopes-ENUM-030': 'false'
'Utilization.isotopes-ENUM-031': 'false'
'Utilization.isotopes-ENUM-032': 'false'
'Utilization.isotopes-ENUM-033': 'false'
'Utilization.isotopes-ENUM-034': 'false'
'Utilization.isotopes-ENUM-035': 'false'
'Utilization.isotopes-ENUM-036': 'false'
'Utilization.isotopes-ENUM-037': 'false'
'Utilization.isotopes-ENUM-038': 'false'
'Utilization.isotopes-ENUM-039': 'false'
'Utilization.isotopes-ENUM-040': 'false'
'Utilization.isotopes-ENUM-041': 'false'
'Utilization.isotopes-ENUM-042': 'false'
'Utilization.isotopes-ENUM-043': 'false'
'Utilization.isotopes-ENUM-044': 'false'
'Utilization.materials-FALSE': 'true'
'Utilization.materials-TRUE': 'false'
'Utilization.materials-UND': 'false'
'Utilization.mw_days_per_year-CLUSTER-036': 'false'
'Utilization.mw_days_per_year-CLUSTER-058': 'true'
'Utilization.mw_days_per_year-CLUSTER-077': 'false'
'Utilization.mw_days_per_year-CLUSTER-101': 'false'
'Utilization.mw_days_per_year-CLUSTER-UND': 'false'
```

'Utilization.neutron_activation_analysis_methods-ENUM-000': 'true'
'Utilization.neutron_activation_analysis_methods-ENUM-001': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-002': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-003': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-004': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-005': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-006': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-007': 'false'
'Utilization.neutron_activation_analysis_methods-ENUM-008': 'false'
'Utilization.neutron_activation_analysis_samples_per_year-CLUSTER-007': 'false'
'Utilization.neutron_activation_analysis_samples_per_year-CLUSTER-013': 'false'
'Utilization.neutron_activation_analysis_samples_per_year-CLUSTER-017': 'true'
'Utilization.neutron_activation_analysis_samples_per_year-CLUSTER-UND': 'false'
'Utilization.neutron_capture_therapy-FALSE': 'true'
'Utilization.neutron_capture_therapy-TRUE': 'false'
'Utilization.neutron_capture_therapy-UND': 'false'
'Utilization.neutron_capture_therapy_patients_per_year-CLUSTER-000': 'false'
'Utilization.neutron_capture_therapy_patients_per_year-CLUSTER-001': 'false'
'Utilization.neutron_capture_therapy_patients_per_year-CLUSTER-UND': 'true'
'Utilization.neutron_radiography-FALSE': 'true'
'Utilization.neutron_radiography-TRUE': 'false'
'Utilization.neutron_radiography-UND': 'false'
'Utilization.neutron_radiography_hours_per_year-CLUSTER-000': 'false'
'Utilization.neutron_radiography_hours_per_year-CLUSTER-002': 'false'
'Utilization.neutron_radiography_hours_per_year-CLUSTER-003': 'false'
'Utilization.neutron_radiography_hours_per_year-CLUSTER-004': 'false'
'Utilization.neutron_radiography_hours_per_year-CLUSTER-UND': 'true'
'Utilization.neutron_scattering_hours_per_year-CLUSTER-001': 'false'
'Utilization.neutron_scattering_hours_per_year-CLUSTER-002': 'false'
'Utilization.neutron_scattering_hours_per_year-CLUSTER-003': 'false'
'Utilization.neutron_scattering_hours_per_year-CLUSTER-004': 'false'
'Utilization.neutron_scattering_hours_per_year-CLUSTER-UND': 'true'
'Utilization.neutron_scattering_methods-ENUM-000': 'true'
'Utilization.neutron_scattering_methods-ENUM-001': 'false'
'Utilization.neutron_scattering_methods-ENUM-002': 'false'
'Utilization.neutron_scattering_methods-ENUM-003': 'false'
'Utilization.neutron_scattering_methods-ENUM-004': 'false'
'Utilization.neutron_scattering_methods-ENUM-005': 'false'
'Utilization.neutron_scattering_methods-ENUM-006': 'false'
'Utilization.neutron_scattering_methods-ENUM-007': 'false'
'Utilization.neutron_scattering_methods-ENUM-008': 'false'
'Utilization.neutron_scattering_methods-ENUM-009': 'false'
'Utilization.neutron_scattering_methods-ENUM-010': 'false'

```
'Utilization.neutron_scattering_methods-ENUM-011': 'false'
'Utilization.neutron_scattering_methods-ENUM-012': 'false'
'Utilization.neutron_scattering_methods-ENUM-013': 'false'
'Utilization.neutron_scattering_methods-ENUM-014': 'false'
'Utilization.neutron_scattering_methods-ENUM-015': 'false'
'Utilization.neutron_scattering_methods-ENUM-016': 'false'
'Utilization.neutron_scattering_methods-ENUM-017': 'false'
'Utilization.neutron_scattering_methods-ENUM-018': 'false'
'Utilization.neutron_scattering_methods-ENUM-019': 'false'
'Utilization.neutron_scattering_methods-ENUM-020': 'false'
'Utilization.neutron_scattering_methods-ENUM-021': 'false'
'Utilization.neutron_scattering_methods-ENUM-022': 'false'
'Utilization.neutron_scattering_methods-ENUM-023': 'false'
'Utilization.neutron_scattering_methods-ENUM-024': 'false'
'Utilization.other_use-FALSE': 'true'
'Utilization.other_use-TRUE': 'false'
'Utilization.other_use-UND': 'false'
'Utilization.runs_number_per_year-CLUSTER-001': 'false'
'Utilization.runs_number_per_year-CLUSTER-004': 'false'
'Utilization.runs_number_per_year-CLUSTER-006': 'false'
'Utilization.runs_number_per_year-CLUSTER-UND': 'true'
'Utilization.teaching-FALSE': 'false'
'Utilization.teaching-TRUE': 'true'
'Utilization.teaching-UND': 'false'
'Utilization.teaching_students_year-CLUSTER-003': 'false'
'Utilization.teaching_students_year-CLUSTER-018': 'false'
'Utilization.teaching_students_year-CLUSTER-019': 'true'
'Utilization.teaching_students_year-CLUSTER-UND': 'false'
'Utilization.training-FALSE': 'false'
'Utilization.training-TRUE': 'true'
'Utilization.training-UND': 'false'
'Utilization.transmutation_gemstone_kg_per_year-CLUSTER-000': 'false'
'Utilization.transmutation_gemstone_kg_per_year-CLUSTER-004': 'false'
'Utilization.transmutation_gemstone_kg_per_year-CLUSTER-UND': 'true'
'Utilization.transmutation_mass_kg_per_year-CLUSTER-001': 'false'
'Utilization.transmutation_mass_kg_per_year-CLUSTER-002': 'false'
'Utilization.transmutation_mass_kg_per_year-CLUSTER-003': 'false'
'Utilization.transmutation_mass_kg_per_year-CLUSTER-UND': 'true'
'Utilization.weeks_per_year-CLUSTER-019': 'false'
'Utilization.weeks_per_year-CLUSTER-025': 'true'
'Utilization.weeks_per_year-CLUSTER-059': 'false'
'Utilization.weeks_per_year-CLUSTER-110': 'false'
'Utilization.weeks_per_year-CLUSTER-UND': 'false'
```

Anexo IV. Detalle de uno de los *formal concept* generado

Cada uno de los *formal concept* son generados en un proceso *map-reduce* iterativo; donde son persistidos en ficheros de secuencia *HDFS* en el clúster *hadoop*.

Un *formal concept* consta de un *extent*, o conjunto de objetos compartiendo una serie de atributos; y de un *intent*, o conjunto de atributos compartidos por todos estos objetos.

A continuación se muestra a modo de ejemplo el primer *formal concept* generado durante la primera iteración. El número de *formal concepts* generado ha sido de 70642.

Formal Concept	
Extent (objetos)	08788fd0-e062-4032-afe2-f9f0e257f6a5, 0a249935-cad6-4a2b-a3da-3f9411e0af06, 19382775-f900-447c-a2d6-fefc0779b8af, 1e15d776-2b87-4542-b3ce-a5251f571d10, 2411a0b4-aa68-424e-912d-fd57d2f3396e, 2a759fd6-0eec-4333-b9c3-2077418af813, 33c88f1a-77ae-4717-a84e-e57fba5a113e, 482ebb64-faf3-40bf-b9e2-3bc2df1d7c38, 5242c50f-eddb-4054-95d8-81c72f34a736, 5cd11958-e799-4688-aa87-88dea7e57785, 67256703-3996-424f-b457-f6e4f8b8b6a4, 6b166a13-4ff6-4d90-9c00-a6f36845516e, 6cab2341-bb11-4d52-9b24-5e17874cd1f4, 6dea0738-eaba-4b15-8500-6f8123e20807, 72c7f6e3-187c-45f2-b542-2fdd73bed902, 7906a4e6-619b-4805-8e2e-7c75976feec7, 7eaffa98-d5dc-4954-9559-bc89f7584724, 96cce36a-1f28-4423-9139-80a07f0b887c, 9c8ccf47-bcf1-421f-be2c-bd8c4bae2c36, b7d07d88-643c-4be7-8f8c-2d23ab9c9ef8, cebf11a4-5a82-4774-a9c5-43c01f17de67, d95e4c22-7ab4-4650-9e71-2ca2c4741c83, ed0bf5e9-8c45-4349-aeae-1df0624a0f77
Intent (atributos)	Category.category-ENUM-001, Experimental.horizontal_channels-CLUSTER-UND, Experimental.horizontal_max_flux-CLUSTER-UND, Experimental.irradiation_channels-CLUSTER-UND, Experimental.irradiation_number-CLUSTER-UND, Experimental.reflector_max_flux-CLUSTER-UND, Experimental.use-ENUM-000, TechData.max_flux_pulsed_fast-CLUSTER-UND, TechData.max_flux_pulsed_thermal-CLUSTER-UND, Utilization.geochronology_methods-ENUM-000, Utilization.geochronology_samples_per_year-CLUSTER-UND, Utilization.isotope_total_activity_per_year-CLUSTER-UND, Utilization.isotopes-ENUM-000, Utilization.neutron_activation_analysis_methods-ENUM-000, Utilization.neutron_activation_analysis_samples_per_year-CLUSTER-UND, Utilization.neutron_capture_therapy-FALSE, Utilization.neutron_capture_therapy_patients_per_year-CLUSTER-UND, Utilization.neutron_radiography-FALSE, Utilization.neutron_radiography_hours_per_year-CLUSTER-UND, Utilization.neutron_scattering_hours_per_year-CLUSTER-UND, Utilization.neutron_scattering_methods-ENUM-000, Utilization.transmutation_gemstone_kg_per_year-CLUSTER-UND, Utilization.transmutation_mass_kg_per_year-CLUSTER-UND

Tabla 4: Detalle de formal concept

Anexo V. Ejecución del cliente web

Una vez el servicio web está arrancado[17] , abrir un navegador web (preferiblemente *google chrome)* e ir a la *url* http://localhost:8080/pfc-fca-web:

Ilustración 13: Vista inicial del cliente web

Como se puede observar en la figura superior, en la parte lateral izquierda dispondremos de enlaces para acceder a los datos de cada uno de los objetos de nuestro dominio. En el caso que nos ocupa, cada uno de ellos corresponderá a un reactor de la *IAEA*.

Si pinchamos en uno de ellos :

Ilustración 14: Vista de objeto seleccionado en cliente web

En la parte superior derecha tendremos acceso a los datos del objeto, tal cual fueron importados desde la fuente de datos original y agrupados en categorías. Podremos desplegar sobre cada categoría para poder visualizar la información.

Pero es en la parte inferior donde podremos obtener los datos del resultado del análisis, bajo la

17 Tal cómo se detalla en “Ejecución del servicio web” página 42

sección "Objetos similares". Dispondremos de diversas colecciones de agrupaciones; en orden decreciente de similitud (esto es, en las primeras posiciones obtendremos información sobre los otros objetos que coinciden con el objeto seleccionado en un mayor número de atributos).

Ilustración 15: Detalle de coincidencias de un objeto

Por ejemplo, en la figura superior podemos observar que tenemos 16 objetos de máxima similitud con el objeto seleccionado; así como el detalle de los campos coincidentes.

Si pinchamos en uno de de los enlaces a objetos coincidentes, podremos pasar a su vez a analizar el objeto en cuestión.