TariyKDD: Una herramienta genérica para el Descubrimiento de Conocimiento en Bases de Datos, débilmente acoplada con el SGBD PostgreSQL.

Andrés Oswaldo Calderon Romero
Iván Dario Ramirez Freyre
Alvaro Fernando Guevara Unigarro
Juan Carlos Alvarado Perez

Universidad de Nariño
Facultad de Ingeniería
Programa de Ingeniería de Sistemas
San Juan de Pasto
2006

TariyKDD: Una herramienta genérica para el Descubrimiento de Conocimiento en Bases de Datos, débilmente acoplada con el SGBD PostgreSQL.

Andrés Oswaldo Calderon Romero
Iván Dario Ramirez Freyre
Alvaro Fernando Guevara Unigarro
Juan Carlos Alvarado Perez

Trabajo de Grado presentado como requisito para optar la título de Ingenieros de Sistemas

Ricardo Timaran Pereira
Ph.D
Director del Proyecto

Universidad de Nariño
Facultad de Ingeniería
Programa de Ingeniería de Sistemas
San Juan de Pasto
2006

Índice general

Capítulo 1

INTRODUCCIÓN

El proceso de extraer conocimiento a partir de grandes volúmenes de datos ha sido reconocido por muchos investigadores como un tópico de investigación clave en los sistemas de bases de datos, y por muchas compañías industriales como una importante área y una oportunidad para obtener mayores ganancias [46].

El Descubrimiento de Conocimiento en Bases de Datos (DCBD) es básicamente un proceso automático en el que se combinan descubrimiento y análisis. El proceso consiste en extraer patrones en forma de reglas o funciones, a partir de los datos, para que el usuario los analice. Esta tarea implica generalmente preprocesar los datos, hacer minería de datos (data mining) y presentar resultados [3, 6, 9, 25, 36]. El DCBD se puede aplicar en diferentes dominios por ejemplo, para determinar perfiles de clientes fraudulentos (evasión de impuestos), para descubrir relaciones implícitas existentes entre síntomas y enfermedades, entre características técnicas y diagnóstico del estado de equipos y máquinas, para determinar perfiles de estudiantes "académicamente exitosos" en términos de sus características socioeconómicas, para determinar patrones de compra de los clientes en sus canastas de mercado, entre otras.

Las investigaciones en DCBD, se centraron inicialmente en definir nuevas operaciones de descubrimiento de patrones y desarrollar algoritmos para estas. Investigaciones posteriores se han focalizado en el problema de integrar DCBD con Sistemas Gestores de Bases de Datos (SGBD) ofreciendo como resultado el desarrollo de herramientas DCBD cuyas arquitecturas se pueden clasificar en una de tres categorías: débilmente acopladas, medianamente acopladas y fuertemente acopladas con el SGBD [45].

Una herramienta DCBD debe integrar una variedad de componentes (técnicas de

minería de datos, consultas, métodos de visualización, interfaces, etc.), que juntos puedan eficientemente identificar y extraer patrones interesantes y útiles de los datos almacenados en las bases de datos. De acuerdo a las tareas que desarrollen, las herramientas DCBD se clasifican en tres grupos: herramientas genéricas de tareas sencillas, herramientas genéricas de tareas múltiples y herramientas de dominio específico [36].

En este documento se presenta el trabajo de grado para optar por el título de Ingeniero de Sistemas. Fruto de la presente investigación es el desarrollo de "TariyKDD: Una herramienta genérica de Descubrimiento de Conocimiento en Bases de Datos débilmente acoplada con el SGBD PostgreSQL", en la cual también se implementaron los algoritmos EquipAsso [49, 48, 47] y MateTree [47] para las tareas de Asociación y Clasificación propuestos por Timarán en [47] y sobre los cuales se realizarón ciertas pruebas para medir su rendimiento.

El resto de este documento esta organizado de la siguiente manera. En la siguiente sección se especifica el Tema de la propuesta, se lo enmarca dentro de una línea de investigación y se lo delimita. A continuación se describe el problema objeto de estudio. En la sección 4 se especifican los objetivos generales y específicos del anteproyecto. En la sección 5 se presenta la justificación de la propuesta de trabajo de grado. En la sección 6 se presenta el estado general del arte en el área de integración de DCBD y SGBD. En la sección 7 se desarrolla todo lo concerniente al análisis orientado a objetos UML que se realizo para construir la herramienta y finalmente en la sección 8 se presentan las conclusiones, recomendaciones, referencias bibliográficas y anexos.

Capítulo 2

TEMA

2.1. Titulo

TariyKDD: Una herramienta genérica para el Descubrimiento de Conocimiento en Bases de Datos, débilmente acoplada con el SGBD PostgreSQL.

2.2. Línea de Investigación

El presente trabajo de grado, se encuentra inscrito bajo la **línea de software y manejo de información**, se enmarca dentro del área de las Bases de Datos y específicamente en la subárea de arquitecturas de integración del Proceso de descubrimiento en Bases de datos con Sistemas Gestores de bases de Datos.

2.3. Alcance y Delimitación

TARIYKDD es una herramienta que contempla todas las etapas del proceso DCBD, es decir: Selección, preprocesamiento, transformación, minería de datos y visualización [3, 6, 23]. En la etapa de minería de datos se implementaron las tareas de Asociación y Clasificación. En estas dos tareas, se utilizaron los operadores algebraicos relacionales y primitivas SQL para DCBD, desarrollados por Timarán [50, 47], con los algoritmos EquipAsso [49, 48, 47] y Mate-tree [47]. En la etapa de Visualización se desarrollo una interfaz gráfica que le permite al usuario interac-

tuar de manera fácil con la herramienta.

Para los algoritmos implementados, se hicieron pruebas de rendimiento, utilizando conjuntos de datos reales y se los comparo con los algoritmos Apriori Híbrido [6], FP-Growth [22, 25] y C.4.5 [37, 28].

Capítulo 3

PROBLEMA OBJETO DE ESTUDIO

3.1. Descripción del Problema

Muchos investigadores [9, 8, 22, 41] han reconocido la necesidad de integrar los sistemas de descubrimiento de conocimiento y bases de datos, haciendo de esta un área activa de investigación. La gran mayoría de herramientas de DCBD tienen una arquitectura débilmente acoplada con un Sistema Gestor de Bases de Datos [45].

Algunas herramientas como Alice [40], C5.0 RuleQuest [26], Qyield [10], CoverStory [30] ofrecen soporte únicamente en la etapa de minería de datos y requieren un pre y un post procesamiento de los datos. Hay una gran cantidad de este tipo de herramientas [29], especialmente para clasificación apoyadas en árboles de decisión, redes neuronales y aprendizaje basado en ejemplos. El usuario de este tipo de herramientas puede integrarlas a otros módulos como parte de una aplicación completa [36].

Otros ofrecen soporte en más de una etapa del proceso de DCBD y una variedad de tareas de descubrimiento, típicamente, combinando clasificación, visualización, consulta y clustering, entre otras [36]. En este grupo están Clementine [42], DBMiner [21, 20, 18], DBLearn [19], Data Mine [25], IMACS [7], Intelligent Miner [11], Quest [4] entre otras. Una evaluación de un gran número de herramientas de este tipo se puede encontrar en [14].

Todas estas herramientas necesitan de la adquisición de costosas licencias para

su utilización. Este hecho limita a las pequeñas y medianas empresas u organizaciones, al acceso de herramientas DCBD para la toma de decisiones, que inciden directamente en la obtención de mayores ganancias y en el aumento de su competitividad.

Por esta razón, se plantea el desarrollo de una herramienta genérica de DCBD, débilmente acoplada, bajo software libre, que permita el acceso a este tipo de herramientas sin ningún tipo de restricciones, a las pequeñas y medianas empresas u organizaciones de nuestro país o de cualquier parte del mundo.

3.2. Formulación del Problema

¿El desarrollo de una herramienta genérica para el Descubrimiento de Conocimiento en bases de datos débilmente acoplada con el SGBD PostgreSQL bajo software libre, facilitará a las pequeñas y medianas empresas la toma de decisiones?

Capítulo 4

OBJETIVOS

4.1. Objetivo General

Desarrollar una herramienta genérica para el Descubrimiento de Conocimiento en bases de datos débilmente acoplada con el sistema gestor de bases de datos PostgreSQL, bajo los lineamientos del Software Libre, que facilite la toma de decisiones a las pequeñas y medianas empresas u organizaciones de nuestro país y de cualquier parte del mundo.

4.2. Objetivos Específicos

1. Estudiar y Analizar diferentes herramientas DCBD débilmente acopladas con un SGBD.

2. Analizar, diseñar y desarrollar programas que permitan la selección, preprocesamiento y transformación de datos.

3. Analizar, diseñar y desarrollar programas que implementen los operadores algebraicos y primitivas SQL para las tareas de Asociación y Clasificación de datos.

4. Analizar, diseñar y desarrollar programas que implementen los algoritmos EquipAsso, MateTree, Apriori Híbrido, FP-Growth y C4.5.

5. Analizar, diseñar y desarrollar programas que permitan visualizar de manera gráfica las reglas de asociación y clasificación.

6. Integrar todos los programas desarrollados en una sola herramienta para DCBD.

7. Implementar la conexión de la herramienta DCBD con el SGBD PostgreSQL.

8. Obtener conjuntos de datos reales para la realización de las pruebas con la herramienta DCBD.

9. Realizar las pruebas y analizar los resultados con la herramienta DCBD débilmente acoplada con PostgreSQL.

10. Realizar las pruebas de rendimiento con los diferentes algoritmos implementados.

11. Dar a conocer los resultados del proyecto a través de la publicación de un artículo en una revista o conferencia nacional o internacional.

Capítulo 5

JUSTIFICACIÓN

Todas las herramientas de DCBD o comúnmente conocidas como herramientas de minería de datos sirven en las organizaciones para apoyar la toma de decisiones de tipo semiestructurado, única o que cambian rápidamente, y que no es fácil especificar por adelantado. Es evidente que por su diseño, estas herramientas tienen mayor capacidad analítica que otras. Están construidas explícitamente con diversos modelos para obtener patrones insospechados a partir de los datos. Apoyan la toma de decisiones al permitir a los usuarios extraer información útil que antes estaba enterrada en montañas de datos. Diversas herramientas de minería de datos disponibles en el mercado ofrecen diferentes tipos de arquitecturas que determinan, en alguna medida, su versatilidad y su costo. Por lo general todas estas son demasiado costosas, necesitan de licenciamiento para su uso y de software específico.

El desarrollo de TARIYKDD, como una herramienta DCBD bajo licencia pública GNU, permitirá que empresas u organizaciones que por su tamaño no puedan acceder a herramientas DCBD propietarias, utilicen esta tecnología para mejorar la toma de decisiones, maximicen sus ganancias con decisiones acertadas y eleven su poder competitivo, ya que el ritmo actual del mundo y la globalización así lo requieren.

Por otra parte, TARIYKDD se convierte en otro aporte más en el área del Descubrimiento de Conocimiento en bases de datos que la Universidad de Nariño, a través de su programa de Ingeniería de Sistemas, hace al mundo, contribuyendo a la investigación científica y al desarrollo de la región y del país.

Por ser TARIYKDD una herramienta genérica de DCBD, puede utilizarse en diferentes campos como la industria, la banca, la salud y la educación, entre otros.

Capítulo 6

MARCO TEORICO

6.1. El Proceso de Descubrimiento de Conocimiento en Bases de Datos - DCBD

El proceso de DCBD es el proceso que utiliza métodos de minería de datos (algoritmos) para extraer (identificar) patrones que evaluados e interpretados, de acuerdo a las especificaciones de medidas y umbrales, usando una base de datos con alguna selección, preprocesamiento, muestreo y transformación, se obtiene lo que se piensa es conocimiento [13].

El proceso de DCBD es interactivo e iterativo, involucra numerosos pasos con la intervención del usuario en la toma de muchas decisiones y se resumen en las siguientes etapas:

- Selección.
- Preprocesamiento / Data cleaning.
- Transformación / Reducción.
- Minería de Datos (Data Mining).
- Interpretación / evaluación.

6.2. Arquitecturas de Integración de las Herramientas DCBD con un SGBD

Las arquitecturas de integración de las herramientas DCBD con un SGBD se pueden ubicar en una de tres tipos: herramientas débilmente acopladas, medianamente acopladas y fuertemente acopladas con un SGBD [45].

Una arquitectura es débilmente acoplada cuando los algoritmos de Minería de Datos y demás componentes se encuentran en una capa externa al SGBD, por fuera del núcleo y su integración con este se hace a partir de una interfaz [45].

Una arquitectura es medianamente acoplada cuando ciertas tareas y algoritmos de descubrimiento de patrones se encuentran formando parte del SGBD mediante procedimientos almacenados o funciones definidas por el usuario [45].

Una arquitectura es fuertemente acoplada cuando la totalidad de las tareas y algoritmos de descubrimiento de patrones forman parte del SGBD como una operación primitiva, dotándolo de las capacidades de descubrimiento de conocimiento y posibilitándolo para desarrollar aplicaciones de este tipo [45].

Por otra parte, de acuerdo a las tareas que desarrollen, las herramientas DCBD se clasifican en tres grupos: herramientas genéricas de tareas sencillas, herramientas genéricas de tareas múltiples y herramientas de dominio específico.

Las Herramientas genéricas de tareas sencillas principalmente soportan solamente la etapa de minería de datos en el proceso de DCBD y requieren un pre y un post procesamiento de los datos. El usuario final de estas herramientas es típicamente un consultor o un desarrollador quien podría integrarlas con otros módulos como parte de una completa aplicación.

Las Herramientas genéricas de tareas múltiples realizan una variedad de tareas de descubrimiento, típicamente combinando clasificación, asociación, visualización, clustering, entre otros. Soportan diferentes etapas del proceso de DCBD. El usuario final de estas herramientas es un analista quien entiende la manipulación de los datos.

Finalmente, las Herramientas de dominio específico, soportan descubrimiento solamente en un dominio específico y hablan el lenguaje del usuario final, quien necesita conocer muy poco sobre el proceso de análisis.

6.3. Implementación de Herramientas DCBD débilmente acopladas con un SGBD

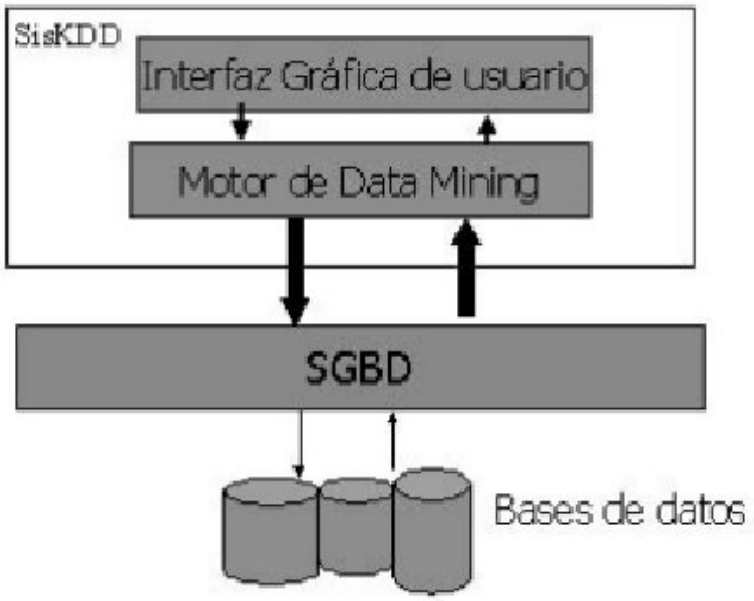

Figura 6.1: Arquitectura DCBD débilmente acoplada

La implementación de herramientas DCBD débilmente acopladas con un SGBD se hace a través de SQL embebido en el lenguaje anfitrión del motor de minería de datos [5]. Los datos residen en el SGBD y son leídos registro por registro a través de ODBC, JDBC o de una interfaz de cursores SQL. La ventaja de esta arquitectura es su portabilidad. Sus principales desventajas son la escalabilidad y el rendimiento. El problema de escalabilidad consiste en que las herramientas y aplicaciones bajo este tipo de arquitectura, cargan todo el conjunto de datos en memoria, lo que las limita para el manejo de grandes cantidades de datos. El bajo rendimiento se debe a que los registros son copiados uno por uno del espacio de direccionamiento de la base de datos al espacio de direccionamiento de la aplicación de minería de datos [8, 24] y estas operaciones de entrada/salida, cuando se manejan grandes volúmenes de datos, son bastante costosas, a pesar de la optimización de lectura por bloques presente en muchos SGBD (Oracle, DB2, Informix, PostgreSQL.) donde un bloque de tuplas puede ser leído al tiempo (figura 1).

6.4. Algoritmos implementados en TariyKDD

Dentro de la herramienta de Minería de Datos TariyKDD fueron implementados los siguientes algoritmos de Asociación: *Apriori, FPGrowth y EquipAsso*, así como también algoritmos de clasificación: *C.4.5 y MateBy*, los cuales son explicados a continuación:

6.4.1. Apriori

La notación del algoritmo Apriori [6] es la siguiente:

Cuadro 6.1: Notación algoritmo Apriori

k-**itemset**	**Un itemset con** k **items**
L_k	Conjunto de itemsets frecuentes k (Aquellos con soporte mínimo).
C_k	Conjunto de itemsets candidatos k (Itemstes potencialmente frecuentes)

A continuación se muestra el algoritmo Apriori:

$L_1 =$ {`Conjunto de itemsets frecuentes 1` }
`for (` $k = 2;$ $L_{k-1} \neq 0;$ $k++$ `) do begin`
$C_k =$ `apriori-gen`(L_{k-1}) `// Nuevos candidatos`
`forall` *transacciones* $t \in D$ `do begin`
$C_t =$ `subconjunto` $(C_k, t);$ `// Candidatos en t`
`forall` *candidatos* $c \in C_t$ `do`
$c.cont++;$
`end`
$L_k = \{ \ c \in C_k | c.count \geq minsup \ \}$
`end Answer` $\cup_k L_k$

La primera pasada del algoritmo cuenta las ocurrencias de los items en todo el conjunto de datos para determinar los itemsets frecuentes 1. Los subsecuentes

pasos del algoritmo son basicamente dos, primero, los itemsets frecuentes L_{k-1} encontrados en la pasada $(k-1)$ son usados para generar los itemsets candidatos C_k, usando la función apriori-gen descrita en la siguiente subsección. Y segundo se cuenta el soporte de los itemsets candidatos C_k a través de un nuevo recorrido a la base de datos. Se realiza el mismo proceso hasta que no se encuentren más itemsets frecuentes.

Generación de candidatos en Apriori

La función apriori-gen toma como argumento L_{k-1}, o sea todos los itemsets frecuentes $(k-1)$. La función trabaja de la siguiente manera:

```
 insert into C_k
select p.item_1,   p.item_2,...,   p.item_{k-1},   q.item_{k-1}
from L_{k-1}   p,   L_{k-1}   q
where p.item_1 = q.item_1,...,   p.item_{k-2} = q.item_{k-2},
p.item_{k-1}q.item_{k-1};
```

A continuación, en el paso de poda, se borran todos los itemsets $c \in C_k$ tal que algún subconjunto $(k-1)$ de c no este en L_{k-1}:

```
 forall itemsets   c ∈ C_k  do
forall subconjuntos (k-1)   s de c do
if (s ∉ L_{k-1}) then
delete c from C_k;
```

6.4.2. FPGrowth

Como se puede ver en [24] la heurística utilizada por Apriori logra buenos resultados, ganados por (posiblemente) la reducción del tamaño del conjunto de candidatos. Sin embargo, en situaciones con largos patrones, soportes demasiado pequeños, un algoritmo tipo Apriori podría sufrir de dos costos no triviales:

- Es costoso administrar un gran número de conjuntos candidatos. Por ejemplo, si hay 10^4 Itemsets Frecuentes, el algoritmo Apriori necesitará generar más de 10^7 Itemsets Candidatos, así como acumular y probar su ocurrencia. Además, para descubrir un patrón frecuente de tamaño 100, como $a_1, ..., a_{100}$, se debe generar más de 2^{100} candidatos en total.
- Es una tarea demasiado tediosa el tener que repetidamente leer la base de datos para revisar un gran conjunto de candidatos.

El cuello de botella de Apriori es la generación de candidatos [24]. Este problema es atacado por los siguientes tres aspectos:

Primero, una innovadora y compacta estructura de datos llamada *Árbol de Patrones Frecuentes* o *FP-tree* por sus siglas en ingles (*Frequent Pattern Tree*), la cual es una estructura que almacena información crucial y cuantitativa acerca de los patrones frecuentes. Únicamente los itemsets frecuentes 1 tendrán nodos en el árbol, el cual esta organizado de tal forma que los items más frecuentes de una transacción tendrán mayores oportunidades de compartir nodos en la estructura.

Segundo, un método de Minería de patrones crecientes basado en un FP-tree. Este comienza con un patrón frecuente tipo 1 (como patrón sufijo inicial), examina sus Patrones Condicionales Base (una "sub-base de datos" que consiste en el conjunto de items frecuentes, que se encuentran en patrón sufijo), construye su FP-tree (condicional) y dentro de este lleva a cabo recursivamente Minería. El patrón creciente es conseguido a través de la concatenación del patrón sufijo con los nuevos generados del FP-tree condicional. Un itemset frecuente en cualquier transacción siempre se encuentra en una ruta de los árboles de Patrones Frecuentes.

Tercero, la técnica de busqueda empleada en la Minería esta basada en particionamiento, con el método "divide y venceras". Esto reduce dramaticamente el tamaño del Patrón Condicional Base generado en el siguiente nivel de busqueda, así como el tamaño de su correspondiente FP-tree. Es más, en vez de buscar grandes patrones frecuentes, busca otros más pequeños y los concatena al sufijo. Todas estas técnicas reducen los costos de busqueda.

Diseño y construcción del Árbol de Patrones Frecuentes (FP-tree)

Sea $I = a_1, a_2, ..., a_m$ un conjunto de items, y $DB = T_1, T_2, ..., T_m$ una base de datos de transacciones, donde $T_i (i \in [1...n])$ es una transacción que contiene un conjunto de items en I. El soporte (u ocurrencia) de un patrón A o conjunto de items, es el número de veces que A esta contenida en DB. A es un patrón frecuente, si el soporte de A es mayor que el umbral o soporte mínimo, ξ.

Árbol de Patrones Frecuentes A través del siguiente ejemplo se examina como funciona el diseño de la estructura de datos para minar con eficiencia patrones frecuentes.

Ejemplo 1 Sea la base de datos de transacciones, cuadro 6.2 y $\xi = 3$. Una estructura de datos puede ser diseñada de acuerdo a las siguientes observaciones:

1. Aunque solo los items frecuentes jugarán un rol en la Minería de Patrones Frecuentes, es necesario leer la BD para identificar este conjunto de items.

2. Si se almacenan items frecuentes de cada transacción en una estructura compacta, se podría evitar el tener que leer repetidamente la BD.

3. Si múltiples transacciones comparten un conjunto idéntico de items frecuentes, estas pueden ser fusionadas en una sola, con el número de ocurrencias como contador.

4. Si dos transacciones comparten un prefijo común, las partes compartidas pueden ser unidas usando una estructura prefija. Si los items frecuentes son ordenados descendientemente, habrá mayor probabilidad de que los prefijos de las cadenas esten compartidos.

Cuadro 6.2: Base de Datos de transacciones

TID	**Items**	**Items frecuentes (ordenados)**
100	f,a,c,d,g,i,m,p	f,c,a,m,p
200	a,b,c,f,l,m,o	f,c,a,b,m
300	b,f,h,j,o	f,b
400	b,c,k,s,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

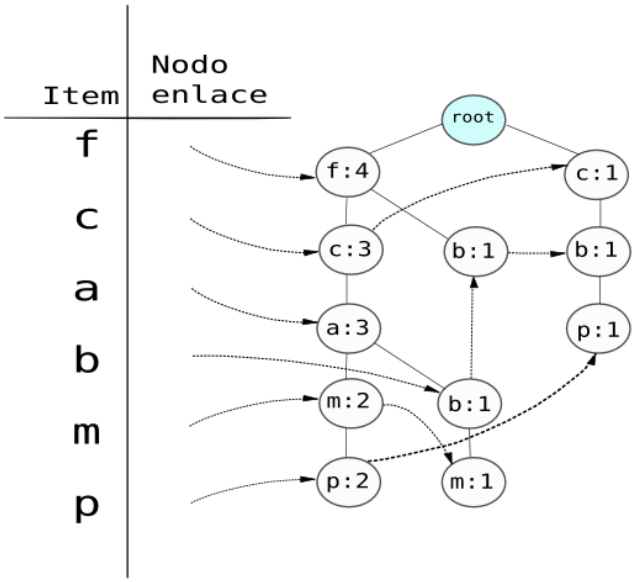

Figura 6.2: Tabla de cabeceras y Arbol FP-tree del ejemplo 1

Con estas observaciones se puede construir un Árbol de Patrones Frecuentes de la siguiente forma:

Primero, el leer DB genera una lista de items frecuentes, $\{(f : 4), (c : 4), (a : 3), (b : 3), (m : 3), (p : 3)\}$, (el número indica el soporte) ordenados descendientemente.

Segundo, crear la raíz del árbol con un *null*. Al leer la primera transacción se construye la primera rama del árbol $\{(f : 1), (c : 1), (a : 1), (m : 1), (p : 1)\}$. Para la segunda transacción ya que su lista de items frecuentes (f, c, a, b, m) comparte el prefijo común (f, c, a) con la rama existente (f, c, a, m, p), el conteo de cada nodo en el árbol prefijo es incrementado en 1, dos nuevos nodos son creados, $(b : 1)$ enlazado como hijo de $(a : 2)$ y $(m : 1)$ enlazado como hijo de $(b : 1)$. Para la tercera transacción como su lista de frecuentes solamente es (f, b) y el único prefijo compartido es (f), el soporte de (f) se incrementa en 1, y un nuevo nodo $(b : 1)$ es creado y enlazado como hijo de $(f : 3)$. La lectura de la cuarta transacción lleva a la construcción de la segunda rama del árbol, $\{(c : 1), (b : 1), (p : 1)\}$. Para la última transacción ya que su lista de items frecuentes (f, c, a, m, p) es identica a la primera, la ruta es compartida, incrementado el conteo de cada nodo en 1.

Para facilitar más el funcionamiento del árbol una Tabla de Cabeceras es construi-

da, en la cual cada item apunta a su ocurrencia en el árbol. Los nodos con el mismo nombre son enlazados en secuencia y después de leer todas las transacciones, se puede ver el árbol resultante en la figura 6.2.

Por tanto un **Árbol de Patrones Frecuentes (FP-tree)** es una estructura como se define a continuación:

- Consiste de una raiz etiquetada como *null*, un conjunto de árboles hijos de la raiz y una Tabla de Cabeceras de items frecuentes.
- Los nodos del Árbol de Patrones Frecuentes o FP-tree tienen tres campos: nombre del item, contador y enlaces a los demás nodos. El nombre del item registra que item este nodo representa, el contador registra el número de transacciones representadas por la porción de la ruta que alcanzan a este nodo y los enlaces llevan al siguiente nodo en el FP-tree, que tiene el mismo nombre o a *null* si no hay nada.
- Cada entrada en la Tabla de Cabeceras de items frecuentes tiene dos campos, nombre del item y el primer nodo enlazado, al cual apunta la cabecera con el mismo nombre.

Minando Patrones Frecuentes con FP-tree

Existen ciertas propiedades del Árbol de Patrones Frecuentes que facilitarán la tarea de Minería de Patrones Frecuentes:

1. **Propiedad de nodos enlazados**. Para cualquier nodo frecuente a_i, todos los posibles Patrones Frecuentes que contenga a_i pueden ser obtenidos siguiendo los nodos enlazados de a_i, comenzando desde a_i en la Tabla de Cabeceras de items frecuentes.

Ejemplo 2 El siguiente es el proceso de Minería basado en el FP-tree de la figura 6.2. De acuerdo a la propiedad de nodos enlazados para obtener todos los Patrones Frecuentes de un nodo a_i, se comienza desde la cabeza de a_i (en la Tabla de Cabeceras).

Comenzando por los nodos enlazados de p su Patrón Frecuente resultante es $(p:3)$ y sus dos rutas en el FP-tree son $(f:4, c:3, a:3, m:2, p:2)$ y $(c:1, b:1, p:1)$. La primera ruta indica que la cadena "(f, c, a, m, p)" aparece dos veces en la base

de datos. Se puede observar que "(f, c, a)" aparece tres veces y "(f)" se encuentra cuatro veces, pero con p solo aparecen dos veces. Además para estudiar que cadena aparece con p, unicamente cuenta el prefijo de p, $(f : 4, c : 3, a : 3, m : 2)$. Similarmente, la segunda ruta indica que la cadena "(c, b, p)" aparece solo una vez en el conjunto de transacciones de DB, o que el prefijo de p es $(c : 1, b : 1)$. Estos dos prefijos de p, $(f : 2, c : 2, a : 2, m : 2)$ y $(c : 1, b : 1)$, forman los Sub-Patrones Base de p o llamados Patrones Condicionales Base. La construcción de un FP-tree sobre este Patrón Condicional Base lleva a unicamente una rama $(c : 3)$. Así que solo existe un Patrón Frecuente $(cp : 3)$.

Para el nodo m, se obtiene el Patrón Frecuente $(m : 3)$ y las rutas $(f : 4, c : 3, a : 3, m : 2)$ y $(f : 4, c : 3, a : 3, b : 1, m : 1)$. Al igual que en el análisis anterior se obtiene los Patrones Condicionales Base de m, que son $(f : 2, c : 2, a : 2)$ y $(f : 1, c : 1, a : 1, b : 1)$. Al construir un FP-tree sobre estos, se obtiene que el FP-tree condicional de m es $(f : 3, c : 3, a : 3)$. Después se podría llamar recursivamente a una función de Minería basada en un FP-tree (mine$((f : 3, c : 3, a : 3)|m)$).

La figura 6.3 muestra como funciona (mine$((f : 3, c : 3, a : 3)|m)$) y que incluye minar tres items a, c, f. La primera deriva en un Patrón Frecuente $(am : 3)$, y una llamada a (mine$((f : 3, c : 3)|am)$); la segunda deriva en un Patrón Frecuente $(cm : 3)$, y una llamada a (mine$((f : 3)|cm)$); y la tercera deriva en unicamente el Patrón Frecuente $(fm : 3)$. Con adicionales llamadas recursivas a (mine$((f : 3, c : 3)|am)$) se obtiene $(cam : 3)$, $(fam : 3)$ y con una llamada a (mine$((f : 3)|cam)$), se obtendrá el patrón más largo $(fcam : 3)$. Similarmente con la llamada de (mine$((f : 3)|cm)$), se obtiene el patrón $(fcm : 3)$. Además todo el conjunto de Patrones Frecuentes que tienen a m es $\{(m : 3), (am : 3), (cm : 3), (fm : 3), (cam : 3), (fam : 3), (fcam : 3), (fcm : 3)\}$ o que explicado de otra forma, por ejemplo para m los Patrones Frecuentes son obtenidos combinando a m con todos sus Patrones Condicionales Base $(f : 3, c : 3, a : 3)$, entonces sus Patrones Frecuentes serían los ya mencionados $\{(m : 3), (am : 3), (cm : 3), (fm : 3), (cam : 3), (fam : 3), (fcam : 3), (fcm : 3)\}$. .

Así mismo, con el nodo b se obtiene $(b : 3)$ y tres rutas: $(f : 4, c : 3, a : 3, b : 1)$, $(f : 4, b : 1)$ y $(c : 1, b : 1)$. Dado que los Patrones Condicionales Base de b: $(f : 1, c : 1, a : 1)$, $(f : 1)$ y $(c : 1)$ no generan items frecuentes, el proceso de Minería termina. Con el nodo a solo se obtiene un Patrón Frecuente $\{(a : 3)\}$ y un Patrón Condicional Base $\{(f : 3, c : 3)\}$. Además su conjunto de Patrones Frecuentes puede ser generado a partir de sus combinaciones. Concatenandolas con $(a : 3)$, obtenemos $\{(fa : 3), (ca : 3), (fca : 3), \}$. Del nodo c se deriva $(c : 4)$ y un

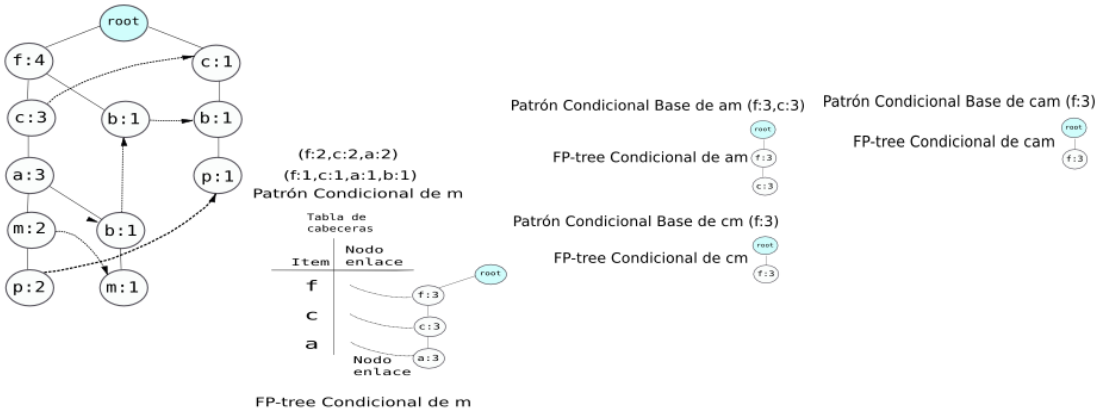

Figura 6.3: FP-tree condicional para m

Patrón Condicional Base $\{(f:3)\}$, y el conjunto de Patrones Frecuentes asociados con $(c:3)$ es $\{(fc:3)\}$. Con el nodo f solo se obtiene $(f:4)$ sin Patrones Condicionales Base.

En la siguiente tabla se muestran los Patrones Condicionales Base y los FP-trees generados.

Cuadro 6.3: Patrones Condicionales Base y FP-trees Condicionales

Item	**Patrón Condicional Base**	**FP-tree condicional**
p	$\{(f:2, c:2, a:2, m:2), (c:1, b:1)\}$	$\{(c:3)\}\|p$
m	$\{(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)\}$	$\{(f:3, c:3, a:3)\}\|m$
b	$\{(f:1, c:1, a:1), (f:1), (c:1)\}$	ϕ
a	$\{(f:3, c:3)\}$	$\{(f:3, c:3)\}\|a$
c	$\{(f:3)\}$	$\{(f:3)\}\|c$
f	ϕ	ϕ

Como se dijo anteriormente los Patrones Frecuentes se obtienen a partir de las combinaciones de cada uno de los items con sus Patrones Condicionales Base. Por ejemplo para m, sus Patrones Frecuentes $\{(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)\}$, son obtenidos de combinar a m con cada uno de sus Patrones Condicionales Base $\{(f:2, c:2, a:2), (f:1, c:1, a:1, b:1)\}$.

6.4.3. EquipAsso

1. Nuevos Operadores Del Algebra Relacional Para Asociación.

   a) Operador Associator (α)
   Associator(α) es un operador algebraico unario que al contrario del operador Selección o Restricción (σ), aumenta la cardinalidad o el tamaño de una relación ya que genera a partir de cada tupla de una relación, todas las posibles combinaciones de los valores de sus atributos, como tuplas de una nueva relación conservando el mismo esquema. Por esta razón esta operación, debe ser posterior a la mayoría de operaciones en el proceso de optimización de una consulta.

   Su sintaxis es la siguiente:

   $$\alpha_{tam_inicial,tam_final}(\mathrm{R})$$

   El operador Associator genera, por cada tupla de la relación R, todos sus posibles subconjuntos (itemsets) de diferente tamaño. Associator toma cada tupla t de R y dos parámetros: $tam\_inicial$ y tam_final como entrada, y retorna, por cada tupla t, las diferentes combinaciones de atributos X_i, de tamaño $tam\_inicial$ hasta tamaño tam_final, como tuplas en una nueva relación. El orden de los atributos en el esquema de R determina los atributos en los subconjuntos con valores, el resto se hacen nulos. El tamaño máximo de un itemset y por consiguiente el tamaño final máximo (tam_final) que se puede tomar como entrada es el correspondiente al valor del grado de la relación.
   Formalmente, sea $A = \{A_1, ..., A_n\}$ el conjunto de atributos de la relación R de grado n y cardinalidad m, IS y ES el tamaño inicial y final respectivamente de los subconjuntos a obtener. El operador α aplicado a R.

   $$\alpha_{IS,ES}(R) = \{\cup_{all} \quad X_i \quad | \quad X_i \subseteq t_i, \quad t_i \in R, \forall_i \forall_k (X_i = v_i(A_1), v_i(A_2), null, ..., v_i(A_k), null\ , v_i(A_k)\ null), (i = (2^n - 1) * m), (k = IS, ..., ES), \quad A_1 A_2 ... A_k, \quad IS = 1, ES = n\}$$

   produce una nueva relación cuyo esquema R(A) es el mismo de R de grado n y cardinalidad $m' = (2^n - 1) * m$ y cuya extensión r(A) está formada por todos los subconjuntos X_i generados a partir de todas las combinaciones posibles de los valores no nulos $v_i(A_k)$ de los atributos

de cada tupla t_i de R. En cada tupla X_i únicamente un grupo de atributos mayor o igual que IS y menor o igual que ES tienen valores, los demás atributos se hacen nulos.

Ejemplo 1. Sea la relación R(A,B,C,D):

A	**B**	**C**	**D**
a1	b1	c1	d1
a1	b2	c1	d2

El resultado de $R1 = \alpha_{2,4}(R)$, se puede observar en la figura 6.4.

b) Operador Equikeep (χ)
Equikeep (χ) es un operador unario que restringe los valores de los atributos de cada una de las tuplas de la relación R a únicamente los valores de los atributos que satisfacen una expresión lógica.

Su sintaxis es la siguiente:

$\chi_{expresion\_logica}(R)$
El operador EquiKeep restringe los valores de los atributos de cada una de las tuplas de la relación R a únicamente los valores de los atributos que satisfacen una expresión lógica *expr_log* , la cual esta formada por un conjunto de cláusulas de la forma Atributo=Valor, y operaciones lógicas AND, OR y NOT. En cada tupla, los valores de los atributos que no cumplen la condición *expr_log* se hacen nulos. EquiKeep elimina las tuplas vacías, i.e. las tuplas con todos los valores de sus atributos nulos.

Formalmente, sea $A = A_1, ..., A_n$ el conjunto de atributos de la relación R de esquema R(A), de grado n y cardinalidad m. Sea p una expresión lógica integrada por cláusulas de la forma $A_i = const$ unidas por los operadores booleanos AND ($\hat{}$), OR (v), NOT ($\neg$). El operador χ aplicado a la relación R con la expresión lógica p:

$$\chi_p(R) = \{t_i(A) | \forall_i \forall_j (p(v_i(A_j)) = v_i(A_j) \quad si \quad p = true \quad y \quad p(v_i(A_j)) = null \quad si \quad p = false), i = 1...m', j = 1...n, \quad m' \leq m\}$$

produce una relación de igual esquema R(A) de grado n y cardinalidad m', donde $m' \leq m$. En su extensión, cada n-tupla t_i, esta formada por los valores de los atributos de R, $v_i(A_j)$, que cumplan la expresión lógica p, es decir $p(v_i(Y_j))$ es verdadero, y por valores nulos si $p(v_i(Y_j))$ es falso.

A	B	C	D
a1	b1	null	null
a1	null	c1	null
a1	null	null	d1
null	b1	c1	null
null	b1	null	d1
null	null	c1	d1
a1	b1	c1	null
a1	b1	null	d1
a1	null	c1	d1
null	b1	c1	d1
a1	b1	c1	d1
a1	b2	null	null
a1	null	c1	null
a1	null	null	d2
null	b2	c1	null
null	b2	null	d2
null	null	c1	d2
a1	b2	c1	null
a1	b2	null	d2
a1	null	c1	d2
null	b2	c1	d2
a1	b2	c1	d2

Cuadro 6.4: Resultado de $R1 = \alpha_{2,4}(R)$

Ejemplo 2. Sea la relación R(A,B,C,D) y la operación $\chi_{A=a1 \vee B=b1 \vee C=c2 \vee D=d1}(R)$:

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b1	c1	d1
a2	b2	c1	d2
a1	b2	c2	d1

El resultado de esta operación es la siguiente:

A	B	C	D
a1	b1	null	d1
a1	null	null	null
null	null	c2	null
null	b1	null	d1
null	null	null	null
a1	null	c2	d1

2. Algoritmo EquipAsso
   El primer paso del algoritmo simplemente cuenta el número de ocurrencias de cada item para determinar los 1-itemsets frecuentes. En el subsiguiente paso, con el operador EquiKeep se extraen de todas las transacciones los itemsets frecuentes tamaño 1 haciendo nulos el resto de valores. Luego se aplica el operador Associator desde $I_s = 2$ hasta el grado n. A continuación se muestra el algoritmo Equipasso:

   $L1 = \{1 - itemsets \quad frecuentes\}$;
   `forall` $transacciones \quad t \in D$ `do begin`
   $R = \chi_{L1}(D)$
   $K = 2$
   $g = grado(R)$
   $R' = \alpha_{k,g}(R)$
   `end`
   $L_k = \{count(R')|c.count \geq minsup\}$
   `Respuesta` $= \cup L_k$;

6.4.4. Mate

El operador Mate genera, por cada una de las tuplas de una relación, todas los posibles combinaciones formadas por los valores no nulos de los atributos pertenecientes a una lista de atributos denominados Atributos Condición, y el valor no nulo del atributo denominado Atributo Clase.

Tiene la sigiente sintaxis:

$$\mu \text{ lista\_atributos\_condición; atributo\_clase(R)}$$

donde lista_atributos_condición es el conjunto de atributos de la relación R a combinar con el atributo clase y atributo_clase es el atributo de R definido como clase.

Mate toma como entrada cada tupla de R y produce una nueva relación cuyo esquema esta formado por los atributos condición, lista_atributos_condición y el

atributo clase, atributo_clase con tuplas formadas por todas las posibles combinaciones de cada uno de los atributos condición con el atributo clase, los demás valores de los atributos se hacen nulos.

Formalmente, sea $A = \{A1, ..., An\}$el conjunto de atributos de la relación R de grado n y cardinalidad m, $LC[]A, LC \neq []$ la lista de atributos condición a emparejar y n' el número de atributos de LC, $[]LC[] = n', n'n, Ac[]A, Ac[]L = []$ el atributo clase con el que se emparejarán los atributos de LC. El operador μ aplicado a la lista de atributos LC, al atributo clase Ac de la relación R:

$$\mu_{LC;Ac}(R) = \{ti(M)[]M = LC[] \;\; Ac, LC[]A, []LC[] = n', n', Ac[]LC = [], \;\; ti = Xi, \;\; i = 1..m', \;\; m' = (2^{n'} - 1) * m, []_i[]_k(X_i = null, ..., v_i(A_k)..., null, ..., v_i(Ac)\ , \;\; v_i(A_k)v_i(Ac)\ null), \;\; k = 1..n'\}$$

produce una relación cuyo esquema es R(M) ,$M = LC[]Ac$, de grado g, $g = n' + 1$ y cuya extensión r(M) de cardinalidad $m', m' = (2n' - 1) * m$ es el conjunto de g-tuplas ti, tal que en cada g-tupla únicamente los atributos que forman la combinación Xi [] LC, $(k = 1..n')$ y el atributo Ac tienen valor, el resto de atributos se hacen nulos.

Mate empareja en cada partición todos los atributos condición con el atributo clase, lo que facilita el conteo y el posterior cálculo de las medidas de entropía y ganancia de información. El operador Mate genera estas combinaciones, en una sola pasada sobre la tabla de entrenamiento (lo que redunda en la eficiencia del proceso de construcción del árbol de decisión).

Ejemplo 4.7. Sea la relación R(A,B,C,D) del cuadro 6.5 obtener las diferentes combinaciones de los atributos A,B con el atributo D, es decir, $R1 = \mu_{A,B;D}(R)$.

A	**B**	**C**	**D**
a1	b1	c1	d1
a1	b1	c1	d1

Cuadro 6.5: Relación R

El resultado de la operación $R1 = \mu_{A,B;D}(R)$, se muestra en el cuadro 6.6:

A	B	C
a1	null	d1
null	b1	d1
a1	b1	d1
a1	null	d2
null	b2	d2
a1	b2	d2

Cuadro 6.6: Resultado de la Operación $R1 = \mu_{A,B;D}(R)$

Función Algebraica Agregada Entro

La función Entro() permite calcular la entropía de una relación R con respecto a un atributo denominado atributo condición y un atributo clase.

Tiene la siguiente sintaxis:

Entro(Atributo; Atributo_clase; R)

donde *atributo* es el atributo condición de la relación R y *atributo_clase* es el atributo con el que se combina el atributo *atributo*.

Formalmente, sea $A = \{A1, ..., An, Ac\}$ el conjunto de atributos de la relación R con esquema $R(A)$, extensión $r(A)$, grado n y cardinalidad m. Sea t el número de distintos valores del *atributo_clase* Ac, $Ac[]R(A)$ que divide a $r(A)$ en t diferentes clases, $Ci(i = 1..t)$. Sea ri el número de tuplas de $r(A)$ que pertenecen a la clase Ci. Sea q el número de destintos valores $\{v_1(A_k), v_2(A_k), .., v_q(A_k)\}$ del *atributo* A_k , $A_k[]R(A)$, el cual particiona a $r(A)$ en q subconjuntos $\{S_1, S_2, ...S_q\}$, donde S_j contiene todos las tuplas de $r(A)$ que tienen el valor $v_j(A_k)$ del atributo A_k. Sea sij el número de tuplas de la clase C_i en el subconjunto S_j. La función $Entro(A_k; Ac; R)$, retorna la entropía de R con respecto al atributo A_k, que se obtiene de la siguiente manera :

$$Entro(A_{k;}Ac; R) = \{y|y = -\sum p_{ij} log_2(p_{ij}), i = 1..t, j = 1..q, p_{ij} = s_{ij}/|S_j|\}$$

donde $p_{ij} = s_{ij}/|S_j es la probabilidad que una tupla en S_j pertenezca a la clase C_i$.

La entropía de R con respecto al atributo clase Ac es:

$$Entro(Ac; Ac; R) = \{y|y = -\sum p_i log_2(p_i), i = 1..t, p_i = r_i/m\}$$

donde p_i es la probabilidad que un tupla cualquier pertenezca a la clase $C_i y r_i$ el número de tuplas de r(A) que pertenecen a la clase C_i.

Función Algebraica Agregada Gain

La función Gain() permite calcular la reducción de la entropía causada por el conocimiento del valor de un atributo de una relación.

Su sintaxis es:

Gain(atributo;atrib_clase;R).

donde *atributo* es el atributo condición de la relación R y *atributo_clase* es el atributo con el que se combina el atributo *atributo*.

La función Gain() permite calcular la ganancia de información obtenida por el particionamiento de la relación R de acuerdo con el atributo *atributo* y y se define:

$$Gain(A_{k;}Ac; R) = \{y|y = Entro(Ac; Ac; R) - Entro(A_{k;}Ac; R)\}$$

donde Entro (Ac; Ac; R) es la entropía de la relación R con respecto al atributo clase Ac y $Entro(A_k; Ac; R)$ es la entropía de la relación R con respecto al atributo A_k.

Operador Describe Classifier ($[]\mu$)

Describe Classifier ($[]\mu$) es un operador unario que toma como entrada la relación resultante de los operadores Mate By, Entro() y Gain() y produce una nueva relación donde se almacenan los valores de los atributos que formarán los diferentes nodos del arbol de decisión.

La sintaxis del operador Describe Classifier es la siguiente:

$$[]\mu(R)$$

Formalmente, sea $A = \{A_1, .., A_n, E, G\}$ el conjunto de atributos de la relación R de grado $n + 2$ y cardinalidad m. El operador $[]//$ aplicado a R:

$$[]//(R) \quad = \quad \{t_i(Y)[] \;\; Y = \{N.P, A.V, C\}, \quad si \;\; t_i = val(N), null, val(A), null, null \;\; []raiz, \quad si \;\; t_i = val(N), val(P), val(A), val(V), val(C) \;\; [] \;\; hoja, \quad si \;\; t_i = val(N), val(P), val(A), val(V), null \;\; [] \;\; nodo \;\; interno\}$$

produce una nueva relación con esquema R(Y), Y=N,P,A,V,C donde N es el atributo que identifica el número de nodo, P identifica el nodo padre, A identifica el nombre del atributo asociado a ese nodo, V es el valor del atributo A y C es el atributo clase. Su extensión r (Y), está formada por un conjunto de tuplas en las cuales si los valores de los atributos son:

$Nnull, \;\; P = null, \;\; Anull, \;\; V = null \;\; y \;\; C = null$ *corresponde a un nodo raiz*; *si* $Nnull$, $Pnull, \;\; Anull, Vnull \;\; y \;\; Cnull$ *corresponde a una hoja o nodo terminal y si* $Nnull$, $P\,null, \;\; A\,null, V\,null \;\; y \;\; C = null \;\;$ *corresponde a un nodo interno.*

Describe Classifier facilita la construcción del árbol de decisión y por consiguiente la generación de reglas de clasificación.

Ejemplo 4.8. Sea la relación ARBOL(NODO,PADRE,ATRIBUTO,VALOR,CLASE) del cuadro 6.4.4 resultado del operador Describe Classifier . Construir el árbol de decisión.

NODO	**PADRE**	**ATRIBUTO**	**VALOR**	**CLASE**
N0	Null	Temperatura	Null	Null
N1	N0	Temperatura	Alta	Si
N2	N0	Temperatura	Media	No
N3	N0	Temperatura	Normal	Null
N4	N3	D_muscular	Si	Si
N5	N3	D_muscular	No	No

Cuadro 6.7: Relación Árbol

El resultado se muestra en la figura 6.4

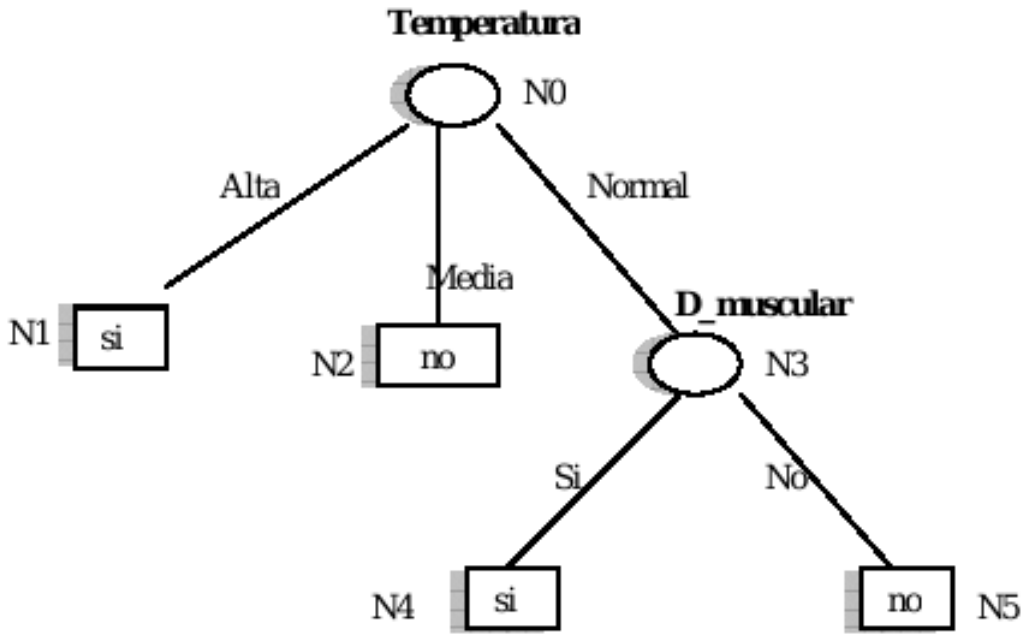

Figura 6.4: Árbol de decisión

6.5. Estado del Arte

En el desarrollo de nuestro trabajo de grado realizamos un estudio de herramientas de minería de datos elaboradas por otras universidades y centros de investigación que nos permitieron ver el estado actual de este tipo de aplicaciones. A continuación se describen las herramientas estudiadas.

6.5.1. WEKA - Waikato Environment for Knowledge Analysis

WEKA es una herramienta libre de minería de Datos realizada en el departamento de Ciencias de la Computación de la Universidad de Waikato en Hamilton, Nueva Zelanda. Los principales gestores de este proyecto son Ian H. Waitten y Eibe Frank autores del libro Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations [51] cuyo octavo capítulo sirve como tutorial de Weka y es libremente distribuido junto con el ejecutable y el código fuente. Las ultimas versiones estables de Weka pueden ser descargadas de la página oficial del Proyecto [17].

La implementación de la herramienta fue hecha bajo la plataforma Java utilizando la versión 1.4.2 de su máquina virtual. Al ser desarrollada en Java, Weka posee todas las características de una herramienta orientada a objetos con la ventaja de ser multiplataforma, la ultima versión (3.4) ha sido evaluada bajo ambientes GNU/Linux, Macintosh y Windows siguiendo una arquitectura Cliente/Servidor lo que permite ejecutar ciertas tareas de manera distribuida.

La conexión a la fuente de datos puede hacerse directamente hacia un archivo plano, considerando varios formatos como C4.5 y CVS aunque el formato oficial es el ARFF que se explica más adelante, o a través de un driver JDBC hacia diferentes Sistemas Gestores de Bases de Datos(SGBD).

Entre las principales características de Weka esta su modularidad la cual se fundamenta en un estricto y estandarizado formato de entrada de datos que denominan ARFF (Atributte - Relation File Format), a partir de este tipo de archivo todos los algoritmos de minería implementados en Weka son trabajados. Nuevas implementaciones y adiciones deben ajustarse a este formato.

El formato ARFF consiste en una archivo sencillo de texto que funciona a partir de etiquetas, similar a XML, y que debe cumplir con dos secciones: una cabecera y un conjunto de datos. La cabecera contiene las etiquetas del nombre de la relación (@relation) y los atributos (@atributte), que describen los tipos y el orden de los datos. La sección datos (@data) en un archivo ARFF contiene todos los datos del conjunto que se quiere evaluar separados por comas y en el mismo orden en el que aparecen en la sección de atributos [15].

La conexión al SGBD se hace en primera instancia a través de una interfaz gráfica de usuario construyendo una sentencia SQL siguiendo un modelo relacional pero a partir de construida la tabla a minar se trabaja en adelante fuera de línea a través de flujos (streams) que se comunican con archivos en disco duro.

WEKA cubre gran parte del proceso de Descubrimiento de Conocimiento, las características especificas de minería de datos que se pueden ver son pre-procesamiento y análisis de datos, clasificación, clustering, asociación y visualización de resultados.

WEKA posee una rica colección de algoritmos que soportan el proceso de minería que aplican diversas metodologías como arboles de decisión, conjuntos difusos, reglas de inducción, métodos estadísticos y redes bayesianas.

Por poseer diferentes modos de interfaces gráficas, WEKA se puede considerar una herramienta orientada al usuario aunque cabe aclarar que exige un buen dominio de conceptos de minería de datos y del objeto de análisis. Muchas tareas se encuentran soportadas aunque no del todo automatizadas por lo que el proceso de descubrimiento debe ser guiado aún por un analista.

Un análisis completo de las características de WEKA con respecto a otras aplicaciones presentes en el mercado se puede encontrar en [14].

6.5.2. ADaM - Algorithm Development and Mining System

El proyecto ADaM es desarrollado por el Centro de Sistemas y Tecnologías de la Información de la Universidad de Alabama en Huntsville, Estados Unidos. Este sistema es usado principalmente para aplicar técnicas de minería a datos científicos obtenidos de sensores remotos [44].

ADaM es un conjunto de herramientas de minería y procesamiento de imágenes que consta de varios componentes interoperables que pueden usarse en conjunto para realizar aplicaciones en la solución de diversos problemas. En la actualidad, ADaM (en su versión 4.0.2) cubre cerca de 120 componentes [43] que pueden ser configurados para crear procesos de minería personalizados. Nuevos componentes pueden fácilmente integrarse a un sistema o proceso existente.

ADaM 4.0.2 provee soporte a través del uso de componentes autónomos dentro de una arquitectura distribuida. Cada componente esta desarrollado en C, C++ u otra interfaz de programación de aplicaciones y entrega un ejecutable a modo de script donde cada componente recibe sus parámetros a través de línea de comandos y arroja los resultados en ficheros que pueden ser utilizados a su vez por otros componentes ADaM. Eventualmente se ofrece Servicios Web de algunos componentes por lo que se puede acceder a ellos a través de la Web.

Lastimosamente el acceso a fuentes de datos es limitada no ofreciendo conexión directa hacia un sistema gestor de bases de datos. ADaM trabaja generalmente con ficheros ARFF [15] que son generados por sus componentes.

Dentro de las herramientas ofrecidas por ADaM encontramos soporte al preprocesamiento de datos y de imágenes, clasificación, clustering y asociación. La vi-

sualización y análisis de resultados se deja para ser implementado por el sistema que invoca a los componentes. ADaM 4.0.2 ofrece un amplio conjunto de herramienta que implementa diversos metodologías dentro del área del descubrimiento de conocimiento. Existen módulos que implementan árboles de decisión, reglas de asociación, métodos estadísticos, algoritmos genéticos y redes bayesianas.

ADaM 4.0.2 no soporta una interfaz gráfica de usuario, se limita a ofrecer un conjunto de herramientas para ser utilizadas en la construcción de sistemas que cubran diferentes ámbitos. Por tal motivo, es necesario un buen conocimiento de los conceptos de minería de datos, a parte de fundamentos en el análisis y procesamiento de imágenes. No obstante, ADaM ofrece un muy buen soporte a sistemas donde se busque descubrimiento de conocimiento, un ejemplo de la implementación de ADaM puede verse en [2] donde se utilizan componentes ADaM en el análisis e interpretación de imágenes satelitales de ciclones para estimar la máxima velocidad de los vientos.

6.5.3. Orange - Data Mining Fruitful and Fun

Construído en el Laborátorio de Inteligencia Artifícial de la Fácultad de Computación y Ciencias de la Información de la Universidad de Liubliana, en Eslovenia y ya que fué liberado bajo la Licencia Pública General (GPL) puede ser descargado desde su sitio oficial [32].

Orange [12] en su núcleo es una librería de objetos C++ y rutinas que incluyen entre otros, algoritmos estandar y no estandar de Minería de Datos y Aprendizaje Maquinal, además de rutinas para la entrada y manipulación de datos. Orange provee un ambiente para que el usuario final pueda acceder a la herramienta a través de scripts hechos en Python y que se encuentran un nivel por encima del núcleo en C++. Entonces el usuario puede incorporar nuevos algoritmos o utilizar código ya elaborado y que le permiten cargar, limpiar y minar datos, así como también imprimir árboles de desición y reglas de asociación.

Otra característica de Orange es la inclusión de un conjunto de Widgets gráficos que usan métodos y módulos del núcleo central (C++) brindando una intefáz agradable e intuitiva al usuario.

Los Widgets de la interfaz gráfica y los módulos en Python incluyen tareas de Minería de Datos desde preprocesamiento hasta modelamiento y evaluación. Entre otras funciones estos componentes poseen técnicas para:

Entrada de datos Orange proporciona soporte para varios formatos populares de datos. Como por ejemplo:

.tab Formato nativo de Orange. La primera línea tiene los nombres de los atributos, la segunda línea dice cual es el tipo de datos de cada columna (discreto o continuo) y en adelante se encuentran los datos separados a través de tabuladores.

.c45 Estos archivos están compuestos por dos archivos uno con extención .names que tiene los nombres de las columnas separados por comas y otro .data con los datos separados también con comas.

Manipulación de datos y preprocesamiento Entre las tareas que Orange incluye en este apartado tenemos visualización gráfica y estadística de datos, procesamiento de filtros, discretización y construcción de nuevos atributos entre otros métodos.

Minería de Datos y Aprendizaje Máquinal Dentro de esta rama Orange incluye variedad de algoritmos de asociación, clasificación, regresión logistica, regresión lineal, árboles de regresión y acercamientos basados en instancias (instance-based approaches).

Contenedores Para la calibración de la predicción de probabilidades de modelos de clasificación.

Métodos de evaluación Que ayudan a medir la exactiud de un clasificador.

Podría catalogarse como una falencia el hecho de que Orange no incluya un modulo para la conexión a un Sistema Gestor de Bases de Datos, pero si se revisa bien la documentación de los modulos se encuentra que ha sido desarrollado uno para la conexión a MySQL (orngMySQL [33]). El modulo provee una entrada a MySQL a través de este modulo y de sencillos comandos en Python los datos de las tablas pueden transferidos desde y hacia MySQL. Así mismo programadores independientes han desarrollado varios modulos entre los que se destaca uno para algoritmos de Inteligencia Artificial.

Dentro de las herramientas de Minería de Datos, Orange podría catalogarse como una Herramienta Genérica Multitarea. Estas herramientas realizan una variedad de tareas de descubrimiento, típicamente combinando clasificación, asociación , visualización, clustering, entre otros. Soportan diferentes etapas del proceso de DCBD. El usuario final de estas herramientas es un analista quien entiende la manipulación de los datos.

Orange funciona en varias plataformas como Linux, Mac y Windows y desde su sitio web [32] se puede descargar toda la documentación disponible para su instalación. Al instalar Orange el usuario puede acceder a una completa información de la herramienta, así como a prácticos tutoriales y manuales que permiten familiarizarse con la misma. Su sitio web [32] incluye tutoriales básicos sobre Python y Orange, manuales para desarrolladores más avanzados y además tener acceso a los foros de Orange en donde se despejan todos los interrogantes sobre esta herramienta.

En si Orange esta hecho para usuarios experimentados e investigadores en aprendizaje maquinal con la ventaja de que el software fue liberado con licencia GPL, por tanto cualquier persona es libre de desarrollar y probar sus propios algoritmos reusando tanto código como sea posible.

6.5.4. TANAGRA - A Free Software for Research and Academic Purposes

TANAGRA [38] es software de Minería de Datos con própositos acádemicos e investigativos, desarrollado por Ricco Rakotomalala, miembro del Equipo de Investigación en Ingeniería del Conocimiento (ERIC - Equipe de Recherche en Ingénierie des Connaissances [1]) de la Universidad de Lyon, Francia. Conjuga varios métodos de Minería de Datos como análisis exploratorio de datos, aprendizaje estadístico y aprendizaje maquinal.

Este proyecto es el sucesor de SIPINA, el cuál implementa varios algoritmos de aprendizaje supervisado, especialmente la construcción visual de árboles de desición. TANAGRA es más potente, contiene además de algunos paradigmas supervisados, clustering, análisis factorial, estadística paramétrica y no-paramétrica, reglas de asociación, selección de características y construcción de algoritmos.

TANAGRA es un proyecto open source, así que cualquier investigador puede acceder al código fuente y añadir sus propios algoritmos, en cuanto este de acuerdo con la licencia de distribución.

El principal próposito de TANAGRA es proponer a los investigadores y estudiantes una arquitectura de software para Minería de Datos que permita el análisis de datos tanto reales como sintéticos.

El segundo próposito de TANAGRA es proponer a los investigadores una arquitec-

tura que les permita añadir sus propios algoritmos y métodos, para así comparar sus rendimientos. TANAGRA es más una plataforma experimental, que permite ir a lo esencial y obviarse la programación del manejo de los datos.

El tercero y último próposito va dirigido a desarrolladores novatos y consiste en difundir una metodología para construir este tipo de software con la ventaja de tener acceso al código fuente, de esta forma un desarrollador puede ver como ha sido construido el software, cuales son los problemas a evadir, cuales son los principales pasos del proyecto y que herramientas y librerías usar.

Lo que no incluye TANAGRA es lo que constituye la fuerza del software comercial en el campo de la Minería de Datos: Grandes fuentes de datos, acceso directo a bodegas y bases de datos (Data Warehouses y Data Bases) así como la aplicación de data cleaning.

Para realizar trabajos de Minería de Datos con Tanagra, se deben crear esquemas y diagramas de flujo y utilizar sus diferentes componentes que van desde la visualización de datos, estadísticas, construcción y selección de instancias, regresión y asociación entre otros.

El formato del conjunto de datos de Tanagra es un archivo plano (.txt) separado por tabulaciones, en la primera línea tiene un encabezado con el nombre de los atributos y de la clase, a continuación vienen los datos con el mismo formato de separación con tabuladores. Tanagra también acepta conjuntos de datos generados en Excel y uno de los estandares en Minería de Datos, el formato de WEKA (archivos .arff). Tanagra permite cargar un solo conjunto de datos.

A medida que se trabaja con TANAGRA y se van generando reportes de resultados, existe la posibilidad de guardarlos en formato HTML.

La paleta de componentes de Tanagra tiene implementaciones de tareas de Minería de Datos que van desde preprocesamiento hasta modelamiento y evaluación. Entre otras TANAGRA permite usar técnicas para:

Entrada de datos Con soporte para varios formatos populares de datos.

Manipulación de datos y preprocesamiento Como muestreo, filtrado, discretización y construcción de nuevos atributos entre otros métodos.

Construcción de Modelos Métodos para la construcción de modelos de clasificación, que incluyen árboles de clasificación, clasificadores Naive-Bayes y

regresión logística.

Métodos de regresión Como regresión multiple lineal.

Métodos de evaluación Utilizados para medir la exactitud de un clasificador.

Al ser TANAGRA un proyecto Open Source es facilmente descargable desde su sitio web [39], el cual contiene tutoriales, referencias y conjuntos de datos.

6.5.5. AlphaMiner

AlphaMiner es desarrollado por el E-Business Technology Institute (ETI) de la Universidad de Hong Kong [27] bajo el apoyo del Fondo para la Innovación y la Tecnología (ITF) [34] del Gobierno de la Región Especial Administrativa de Hong Kong (HKSAR).

AlphaMiner es un sistema de Minería de Datos, Open Source, desarrollado en java de propósito general pero más enfocado al ambiente empresarial. Implementa algoritmos de asociación, clasificación, clustering y regresión logística. Posee una gama amplia de funcionalidad para el usuario, al realizar cualquier proceso minero dando la posibilidad al usuario de escoger los pasos del proceso KDD que más se ajusten a sus necesidades, por medio de nodos que el analista integra a un árbol KDD siguiendo la metodología Drag & Drop. Es por eso que el proceso KDD en AlphaMiner no sigue un orden estructurado, sino que sigue la secuencia brindada por el propósito u objetivo del analista, ademas brinda la visualicion estadística de distintas maneras, permitiendo un analisis más preciso por parte del analista.

El principal objetivo de AlphaMiner es la inteligencia Comercial Económica o Inteligencia de Negocios (BI - Business Inteligence) es uno de los medios más importantes para que las compañías tomen las decisiones comerciales más acertadas. Las soluciones de BI son costosas y sólo empresas grandes pueden permitirse el lujo de tenerlas. Por tanto las compañías pequeñas tienen una gran desventaja. AlphaMiner proporciona las tecnologías de BI de forma económica para dichas empresas para que den soporte a sus decisiones en el ambiente de negocio cambiante rápido.

AlphaMiner tiene dos componentes principales:

1. Una base de conocimiento.
2. Un árbol KDD, que permite integrar nodos artefacto que proporcionan varias funciones para crear, revisar, anular, interpretar y argumentar los distintos análisis de datos

AlphaMiner implementa los siguientes pasos del proceso KDD:

1. Acceso de diferentes formas a las fuentes datos.
2. Exploración de datos de diferentes maneras.
3. Preparación de datos.
4. Vinculación de los distintos modelos mineros.
5. Análisis a partir de los modelos.
6. Despliegue de modelos al ambiente empresarial.

La característica más importante de AlphaMiner, es su capacidad para almacenar los datos después del núcleo minero en una base de conocimiento, que puede ser reutilizada. Esta función aumenta su utilidad significativamente, y brinda un gran apoyo logístico al nivel estratégico de una empresa. Aventajando cualquier sistema tradicional de manipulación de datos. AlphaMiner proporciona una funcionalidad adicional para construir los modelos de Minería de Datos, conformando una sinergia entre los distintos algoritmos de minería.

AlphaMiner se asemeja a Tariy en varias características, por un lado el lenguaje de programación en el que fue implementado es JAVA, por otro lado es Open Source, tiene conectividad JDBC a los distintos gestores de bases de datos, además de conectividad con archivos de Excel. Otra semejanza es el tipo de formato que presenta para el manejo de tablas unívaluadas en algoritmos de asociación, específicamente en bases de datos de supermercados, ya que después de escoger los campos de dicha base de datos se la lleva a un formato de dos columnas una para la correspondiente transacción y otra para el ítem.

6.5.6. YALE - Yet Another Learning Environment

YALE [52, 31] es un ambiente para la realización de experimentos de aprendizaje maquinal y Minería de Datos. A través de un gran número de operadores se pueden crear los experimentos, los cuales pueden ser anidados arbitrariamente y cuya configuración es descrita a través de archivos XML que pueden ser fácilmente creados por medio de una interfaz gráfica de usuario. Las aplicaciones de YALE cubren tanto investigación como tareas de Minería de Datos del mundo real.

Desde el año 2001 YALE ha sido desarrollado en la Unidad de Inteligencia Artificial de la Universidad de Dortmund [35] en conjunto con el Centro de Investigación

Colaborativa en Inteligencia Computacional (Sonderforschungsbereich 531).

El concepto de operador modular permite el diseño de cadenas complejas de operadores anidados para el desarrollo de un gran número de problemas de aprendizaje. El manejo de los datos es transparente para los operadores. Ellos no tienen nada que ver con el formato de datos o con las diferentes presentaciones de los mismos. El kernel de Yale se hace a cargo de las transformaciones necesarias. Yale es ampliamente usada por investigadores y compañías de Minería de Datos.

Modelando Procesos De Descubrimiento De Conocimiento Como Arboles De Operadores

Los procesos de descubrimiento de conocimiento son vistos frecuentemente como invocaciones secuenciales de métodos simples. Por ejemplo, después de cargar datos, uno podría aplicar un paso de preprocesamiento seguido de una invocación a un método de clasificación. El resultado en este caso es modelo aprendido, el cual puede ser aplicado a datos nuevos o no revisados todavía. Una posible abstracción de estos métodos simples es el concepto de operadores. Cada operador recibe su entrada, desempeña una determinada función y entrega una salida. Desde ahi, el método secuencial de invocaciones corresponde a una cadena de operadores. Aunque este modelo de cadena es suficiente para la realización de muchas tareas básicas de descubrimiento de conocimiento, estas cadenas planas son a menudo insuficientes para el modelado de procesos de descubrimiento de conocimiento más complejas.

Una aproximación común para la realización del diseño de experimentos más complejos es diseñar las combinaciones de operadores como un gráfico direccionado. Cada vertice del gráfico corresponde a un operdor simple. Si dos operadores son conectados, la salida del primer operador será usada como entrada en el segundo operador. Por un lado, diseñar procesos de descubrimiento de conocimiento con la ayuda de gráficosdireccionados es muy poderoso. Por el otro lado, existe una desventaja principal: debido a la pérdida de restricciones y la necesidad de ordenar topológicamente el diseño de experimentos es menudo poco intuitivo y las validaciones automáticas son difíciles de hacer.

Yale ofrece un compromiso entre la simplicidad de cadenas de operadores y la potencia de los gráficos direccionados a través del modelamiento de procesos de descubrimiento de conocimiento por medio de árboles de operadores. Al igual que los lenguajes de programación , el uso de árboles de operadores permite el uso de conceptos como ciclos, condiciones, u otros esquemas de aplicación. Las hojas

en el árbol de operadores corresponden a pasos sencillos en el proceso modelado como el aprendizaje de un modelo de predicción ó la aplicación de un filtro de preprocesamiento. Los nodos interiores del árbol corresponden a más complejos o abstractos pasos en el proceso. Este es a menudo necesario si los hijos deben ser aplicados varias veces como, por ejemplo, los ciclos. En general, los nodos de los operadores internos definen el flujo de datos a través de sus hijos. La raiz del árbol corresponde al experimento completo.

Qué Puede Hacer YALE? YALE provee mas de 200 operadores incluyendo:

Algoritmos de aprendizaje maquinal Un gran número de esquemas de aprendizaje para tareas de regresión y clasificación incluyendo Máquinas para el Soporte de Vectores (SVM), árboles de decisión e inductores de reglas, operadores Lazy, operadores Bayesianos y operadores Logísticos. Varios operadores para la minería de reglas de asociación y Clustering son también parte de YALE. Además, se adicionaron varios esquemas de Meta Aprendizaje.

Operadores WEKA Todos los esquemas de aprendizaje y evaluadores de atributos del ambiente de aprendizaje WEKA también están disponibles y pueden ser usados como cualquier otro operador de YALE.

6.6. Conceptos Preliminares durante la Implementación de TariyKDD

Durante la implementación del proyecto TariyKDD fueron utilizadas una serie de herramientas para su desarrollo. Entre las principales de ellas podemos nombrar el lenguaje de programación Java, el entorno de desarrollo NetBeans y la biblioteca gráfica Swing para Java.

A continuación se explican brevemente los conceptos preliminares de estas herramientas para ser tenidas en cuenta en la posterior descripción de la implementación que se realiza en este capítulo.

6.6.1. Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos desarrollado por James Gosling y sus compañeros de Sun Microsystems al inicio de la década

de 1990. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es ejecutado (usando normalmente un compilador JIT), por una máquina virtual Java.

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Características Principales

Orientado a Objetos La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que use estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos).

El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de

comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

Independencia de la plataforma La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Es lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, "write once, run everywhere".

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode) que son instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (VM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran librerías adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

El recolector de basura Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria de forma manual. En C++, el desarrollador debe asignar memoria en una zona conocida como heap (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente

solicitada, o si no lo hace en el instante oportuno, puede llevar a una fuga de memoria, ya que el sistema operativo piensa que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual cuelgue.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o automatic garbage collector). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y frecuentemente más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

Durante la implementación de este proyecto se utilizo la versión JDK1.5.0 para arquitecturas AMD64 en su actualización número 9.

Entorno de Desarrollo NetBeans NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un Entorno

integrado de desarrollo (IDE) desarrollado usando la Plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

La Plataforma NetBeans es un framework reusable que simplifica el desarrollo de otras aplicaciones de escritorio. Cuando se ejecuta una aplicación basada en la Plataforma NetBeans, la plataforma ejecuta la clase Main. Los módulos disponibles están localizados y puestos en un registro en memoria, y son ejecutadas las tareas de inicialización de los módulos. Generalmente el código de un módulo es cargado en memoria solo cuando se necesita.

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

1. Administración de las interfases de usuario (ej. menús y barras de herramientas).
2. Administración de las configuraciones del usuario.
3. Administración del almacenamiento (guardando y cargando cualquier tipo de dato).
4. Administración de ventanas.
5. Framework basado en asistentes (diálogos paso a paso).

Controlador JDBC JDBC es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee en localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar cualquier tipo de tareas con la base de datos a las que tenga permiso: consultas, actualizaciones, creado modificado y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Cada base de datos emplea un protocolo diferente de comunicación, protocolos que normalmente son propietarios. El uso de un manejador, una capa intermedia entre el código del desarrollador y la base de datos, permite independizar el código Java que accede a la BD del sistema de BD concreto a la que estamos accediendo, ya que en nuestro código Java emplearemos comandos estándar, y estos comandos serán traducidos por el manejador a comandos propietarios de cada sistema de BD concreto. Si queremos cambiar el sistema de BD que empleamos lo único que deberemos hacer es reemplazar el antiguo manejador por el nuevo, y seremos capaces de conectarnos la nueva BD. Durante el desarrollo de este proyecto se utilizo un controlador JDBC tipo 4 para el SGDB PostgreSQL.

Manejador de Protocolo Nativo (tipo 4) El manejador traduce directamente las llamadas al API JDBC al protocolo nativo de la base de datos. Es el manejador que tiene mejor rendimiento, pero está más ligado a la base de datos que empleemos que el manejador tipo JDBC-Net, donde el uso del servidor intermedio nos da una gran flexibilidad a la hora de cambiar de base de datos. Este tipo de manejadores también emplea tecnología 100 % Java.

Biblioteca gráfica Swing de Java Swing es una biblioteca gráfica para Java que forma parte de las Java Foundation Classes (JFC). Incluye componentes para interfaz gráfica de usuario tales como cajas de texto, botones, listas desplegables y tablas.

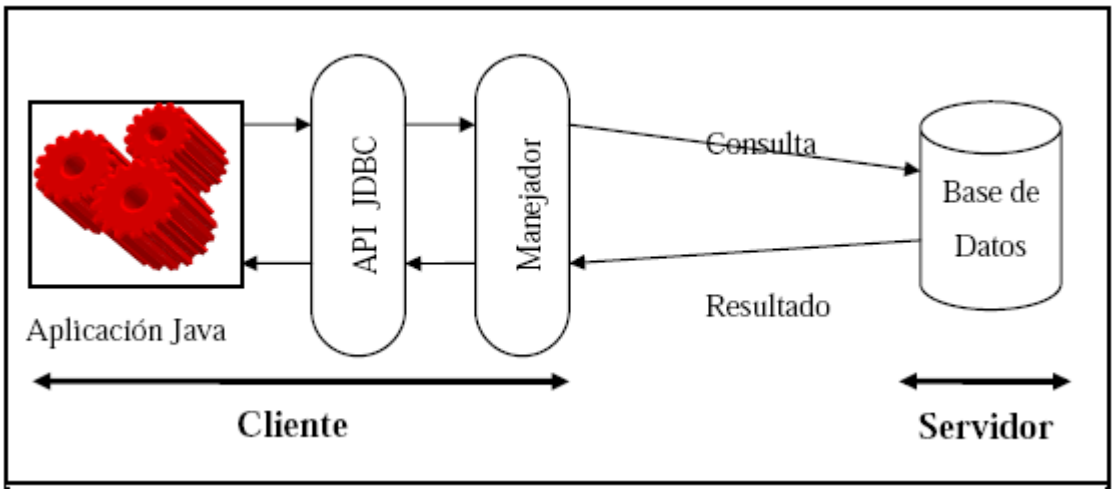

Figura 6.5: Controlador JDBC tipo 4

Las Internet Foundation Classes (IFC) eran una biblioteca gráfica para el lenguaje de programación Java desarrollada originalmente por Netscape y que se publicó en 1996. En 1997, Sun Microsystems y Netscape Communications Corporation anunciaron su intención de combinar IFC con otras tecnologías de las Java Foundation Classes. Además de los componentes ligeros suministrados originalmente por la IFC, Swing introdujo un mecanismo que permitía que el aspecto de cada componente de una aplicación pudiese cambiar sin introducir cambios sustanciales en el código de la aplicación. La introducción de soporte ensamblable para el aspecto permitió a Swing emular la apariencia de los componentes nativos manteniendo las ventajas de la independencia de la plataforma.

Algunos de los componentes utilizados en el desarrollo de este proyecto y pertenecientes a la biblioteca Swing se explican a continuación:

JComponent La clase base para todos los componentes de Swing exceptuando los contenedores de nivel superior. Para utilizar un componente que herede de JComponent, se debe poner el componente en una jerarquía de la contención cuya raíz sea un contenedor de nivel superior Swing tal como Ventanas o Paneles. Los contenedores de nivel superior son los componentes especializados que proporcionan un lugar para que otros componentes de Swing puedan pintarse.

JFrame Un JFrame es una ventana de nivel superior con un título y un borde. El tamaño del JFrame incluye cualquier área señalada por los bordes donde puede ser incluido un o mas componentes de Swing.

JPanel El JPanel es la clase más simple del contenedor. Un JPanel proporciona el espacio en el cual una aplicación puede unir cualquier otro componente, incluyendo otros paneles.

JTabbedPane Un componente que deja a usuario cambiar entre un grupo de componentes pulsando en una etiqueta con un título dado y/o un icono. Las etiquetas y los componentes son agregados a un objeto de TabbedPane usando los métodos del addTab e insertTab. Una etiqueta es representada por un índice que corresponde a la posición que fue agregado adentro, donde la primera etiqueta tiene un índice igual a 0 y la etiqueta pasada tiene un índice igual a la cuenta de la etiqueta menos 1.

JSplitPane JSplitPane se utiliza para dividir dos (y solamente dos) componentes. Los dos componentes se pueden entonces volver a clasificar según el tamaño recíprocamente por el usuario. La información sobre como usar JSplitPane está en cómo utilizar los paneles partidos en la clase particular de Java. Se pueden incluir nuevos componentes en cada una de las divisiones de un JSplitPanne. Los dos componentes en un panel partido se pueden alinear a la izquierda y a la derecha usando JSplitPane.HORIZONTAL_SPLIT, o en la parte superior o inferior de la ventana con JSplitPane.VERTICAL_SPLIT.

JScrollPanne JScrollPanne proporciona una vista deslizante de un componente ligero. Un JScrollPane maneja un viewport, barras de desplazamiento verticales y horizontales opcionales, y viewports opcionales del título de la fila y de columna. El viewport, o punto de vista, proporciona una ventanasobre una fuente de datos, por ejemplo, un archivo de texto o una imagen sobre la cual se va ha deslizar.

JFileChooser Los seleccionadores de archivos proporcionan una interfaz gráfica de usuario para navegar el sistema de ficheros, y después elegir un archivo o un directorio de una lista o incorporar el nombre de un archivo o un directorio. Para exhibir un seleccionador de archivo, se utiliza generalmente el JFileChooser para mostrar un diálogo modal que contiene el seleccionador de

archivo. Otra manera de presentar un seleccionador es agregar una instancia de JFileChooser a un contenedor.

JTable Con la clase de JTable se puede exhibir una tabla de datos, permitiendo opcionalmente que el usuario edite el contenido de las celdas que contienen los datos. JTable no contiene ni deposita datos; es simplemente una vista de los datos.

JTree Con la clase JTree, puedes exhibir datos jerárquicos. Un objeto JTree no contiene realmente los datos sino que proporciona simplemente una vista de ellos. Como cualquier componente no trivial del Swing, el árbol consigue los datos conectándose a un modelo de los datos.

JTextArea Un JTextArea es un área multilínea que exhibe el texto plano y que permite opcionalmente que el usuario corrija el texto. Este es un componente ligero que provee de compatibilidad a la hora de introduccion o desplejar pequeñas cantidades de información.

JSpinner Esta clase provee en una sola línea la posibilidad al usuario de entrar o seleccionar un valor de una secuencia pedida. Los JSpinner proporciona típicamente un par de los botones minúsculos de flechas para cambiar entre los elementos de la secuencia. Las teclas de flecha arriba/abajo del teclado también completan un ciclo a través de los elementos. El usuario puede también mecanografiar el valor directamente en un JSpinner.

JComboBox Un componente que combina un botón o un campo editable y una lista desplegable. El usuario puede seleccionar un valor de la lista, que aparece a petición del usuario. Si la caja de texto es editable, entonces se incluye un campo en el cual el usuario pueda mecanografiar un valor.

JRadioButton Una puesta en práctica de un botón de radio. Este es un componente que puede ser seleccionado o deseleccionado, y que exhibe su estado al usuario. Utilizado con un objeto ButtonGroup crea un grupo de botones en los cuales solamente un botón puede ser seleccionado a la vez.

Capítulo 7

DESARROLLO DEL PROYECTO

7.1. Análisis UML

7.1.1. Funciones

Ref#	**Función**	**Cat.**	**Atributo**	**Detalles y Restricciones**	**Cat.**
R1	Ejecutar la herramienta.	Evidente.	Interfaz.		Obligatorio.
R1.1	Mostrar interfaz gráfica de la herramienta	Evidente.	Interfaz.	Pestañas desplegables.	Obligatorio.
R2	Mostrar mensajes de ayuda, respuesta o error cuando sea necesario.	Evidente.	Interfaz.	Cuadros de dialogo, barra de estado.	Deseable.
R3	Permitir el establecimiento de conexiones.	Evidente.	Interfaz.		Obligatorio.
R3.1	Autorizar conexiones a Bases de Datos.	Evidente.	Interfaz.		Opcional.
R3.1.1	Permitir la selección de atributos.	Evidente.	Interfaz.		Obligatoria.
R3.1.2	Cargar el conjunto de datos.	Oculto.	Información	Establecer relaciones correctas	Obligatorio.
R3.2	Autorizar conexiones a Archivos Planos.	Evidente.	Interfaz.		Opcional.

Ref#	Función	Cat.	Atributo	Detalles y Restricciones	Cat.
R3.2.1	Recorrer archivo y cargar datos.	Oculto.	Información.	El archivo debe cumplir con el formato ARFF.	Obligatorio.
R4	Permitir la aplicación de filtros	Evidente	Interfaz		Opcional
R4.1	Permitir remover datos nulos	Evidente	Interfaz		Opcional
R4.2	Permitir actualizar datos nulos	Evidente	Interfaz		Opcional
R4.3	Permitir seleccionar un conjunto de registros	Evidente	Interfaz		Opcional
R4.4	Permitir seleccionar un conjunto de registros según un atributo dado.	Evidente	Interfaz		Opcional
R4.5	Permitir reducir el rango de los datos	Evidente	Interfaz		Opcional
R4.6	Permitir codificar los datos	Evidente	Interfaz		Opcional
R4.7	Permitir reemplazar un valor determinado	Evidente	Interfaz		Opcional
R4.8	Permitir seleccionar una muestra del conjunto de datos	Evidente	Interfaz		Opcional
R4.9	Permitir discretizar valores continuos	Evidente	Interfaz		Opcional
R5	Aplicar técnicas de Minería de Datos	Evidente	Interfaz		Obligatorio
R5.1	Proveer algoritmos que cumplan tareas de asociación	Evidente	Interfaz		Opcional
R5.1.1	Implementar algoritmo Apriori	Evidente	Interfaz		Opcional
R5.1.2	Implementar algoritmo FPGrowth	Evidente	Interfaz		Opcional
R5.1.3	Implementar algoritmo EquipAsso	Evidente	Interfaz		Opcional

Ref#	Función	Cat.	Atributo	Detalles y Restricciones	Cat.
R5.2	Proveer algoritmos que cumplan tareas de clasificación	Evidente	Interfaz		Opcional
R5.2.1	Implementar algoritmo C4.5	Evidente	Interfaz		Opcional
R5.2.2	Implementar algoritmo MateBy	Evidente	Interfaz		Opcional
R6	Permitir visualización de reglas	Evidente	Interfaz		Obligatorio
R6.1	Organizar las reglas de asociación a través de una tabla	Evidente	Interfaz		Obligatorio
R6.2	Organizar las reglas de clasificación a través de un árbol	Evidente	Interfaz		Obligatorio

7.1.2. Diagramas de Casos de Uso

Tariy

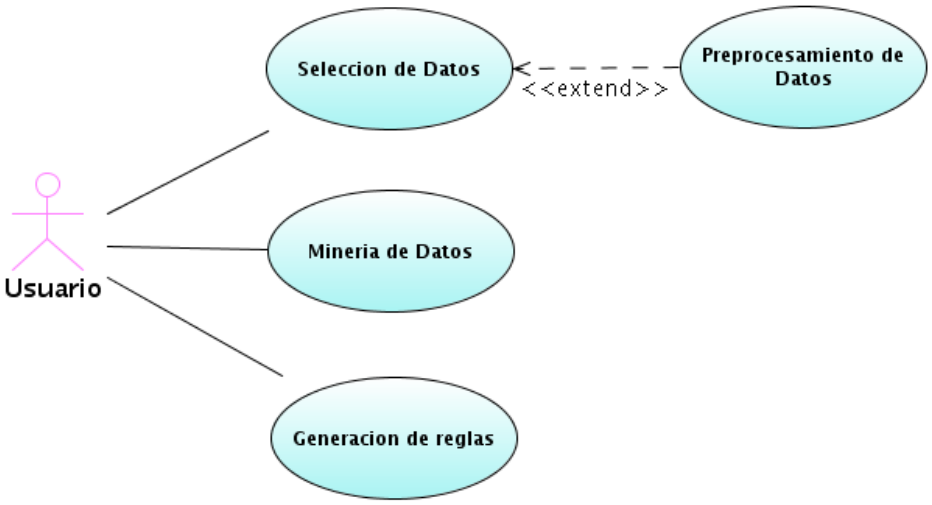

Figura 7.1: Diagrama de caso de uso Tariy

Módulo de Selección

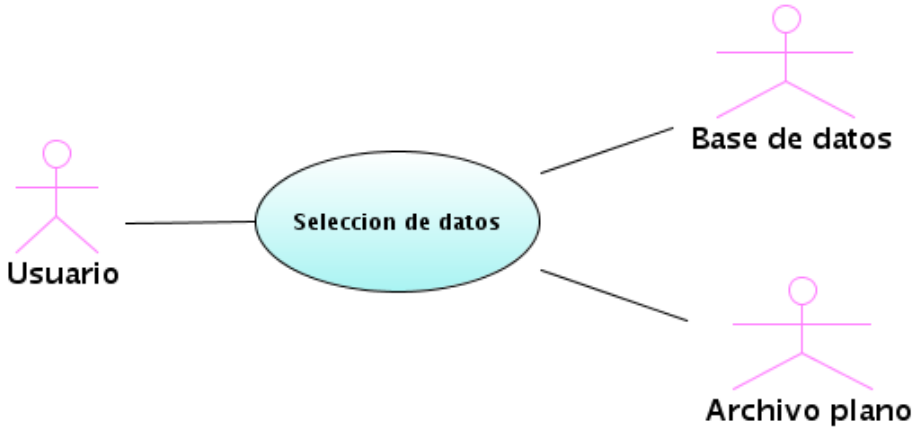

Figura 7.2: Módulo de Selección

Módulo de Conexión a Base de Datos

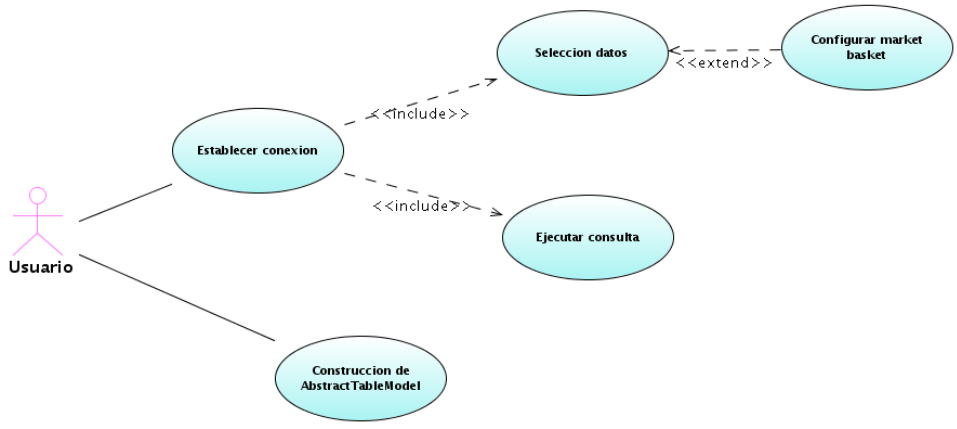

Figura 7.3: Base de Datos

Módulo de Conexión a Archivo Plano

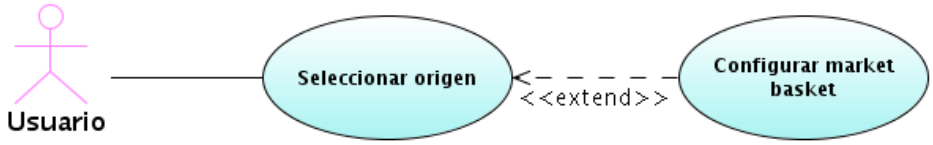

Figura 7.4: Archivo Plano

Módulo de Preprocesamiento

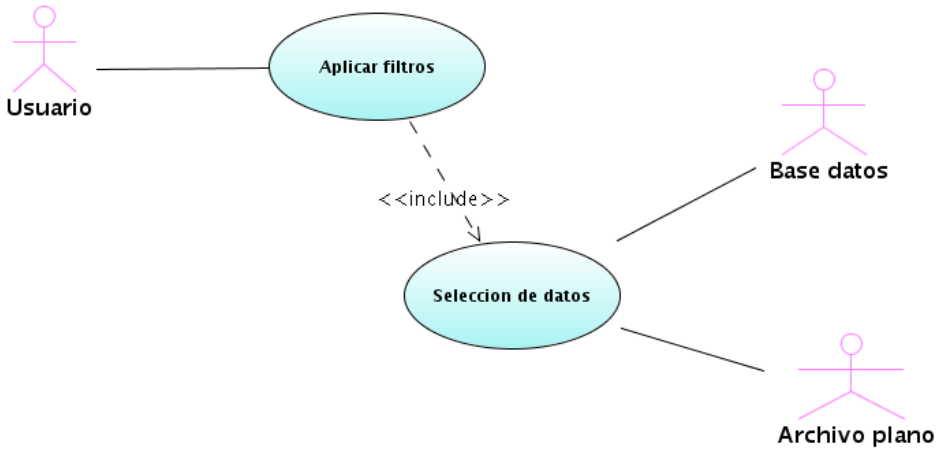

Figura 7.5: Preprocesamiento

Módulo de Minería de Datos

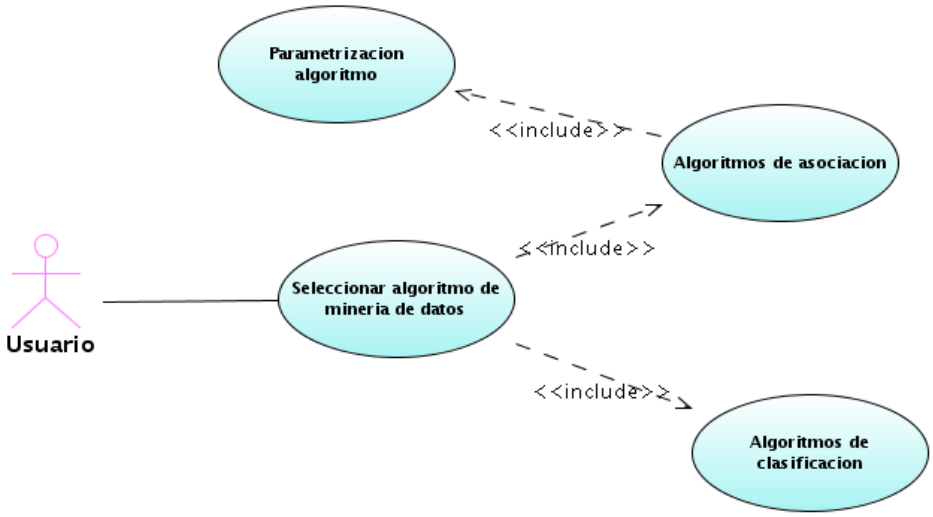

Figura 7.6: Minería de Datos

Módulo de Reglas

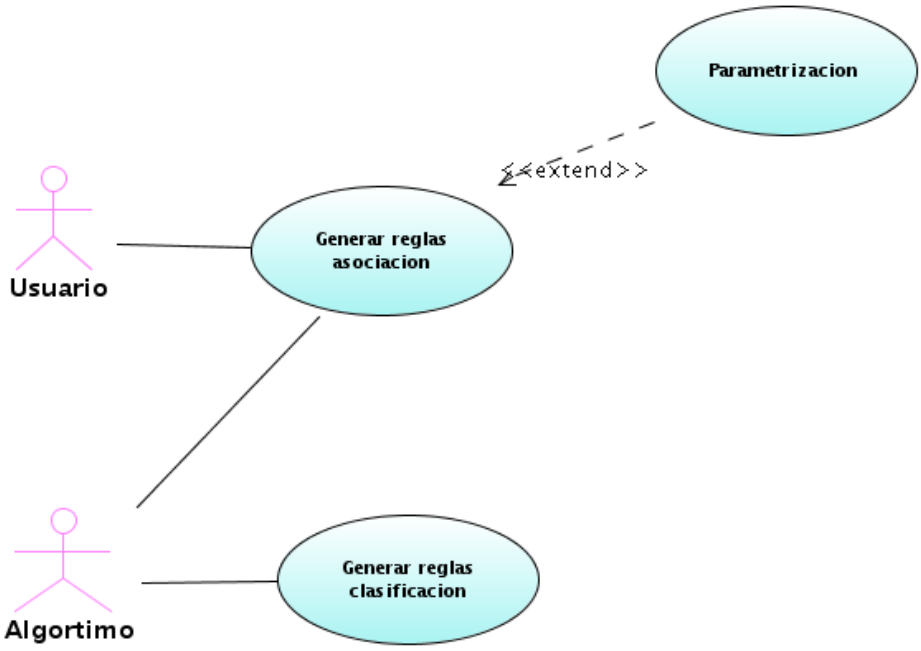

Figura 7.7: Reglas

7.1.3. Diagramas de Secuencia

Clase Apriori

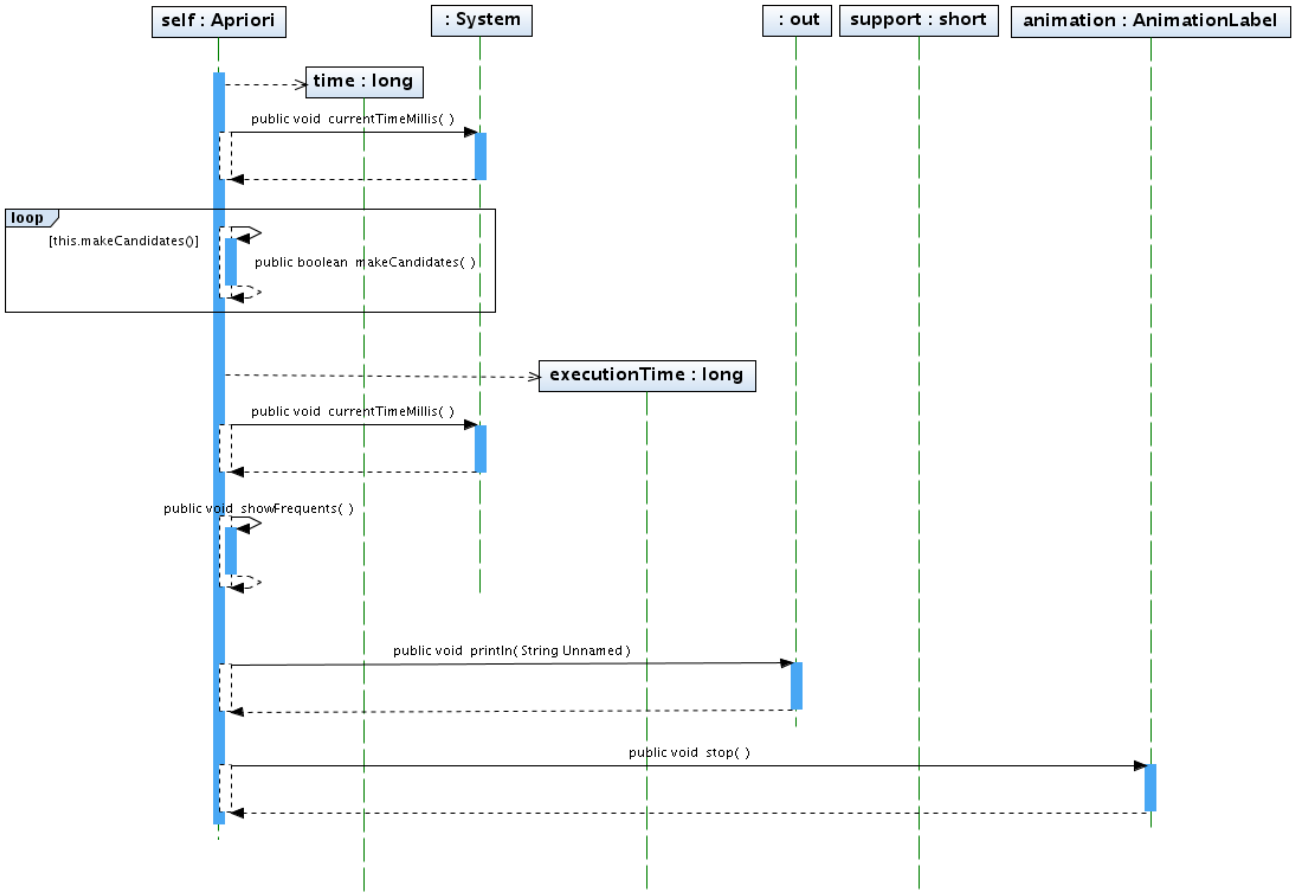

Figura 7.8: run

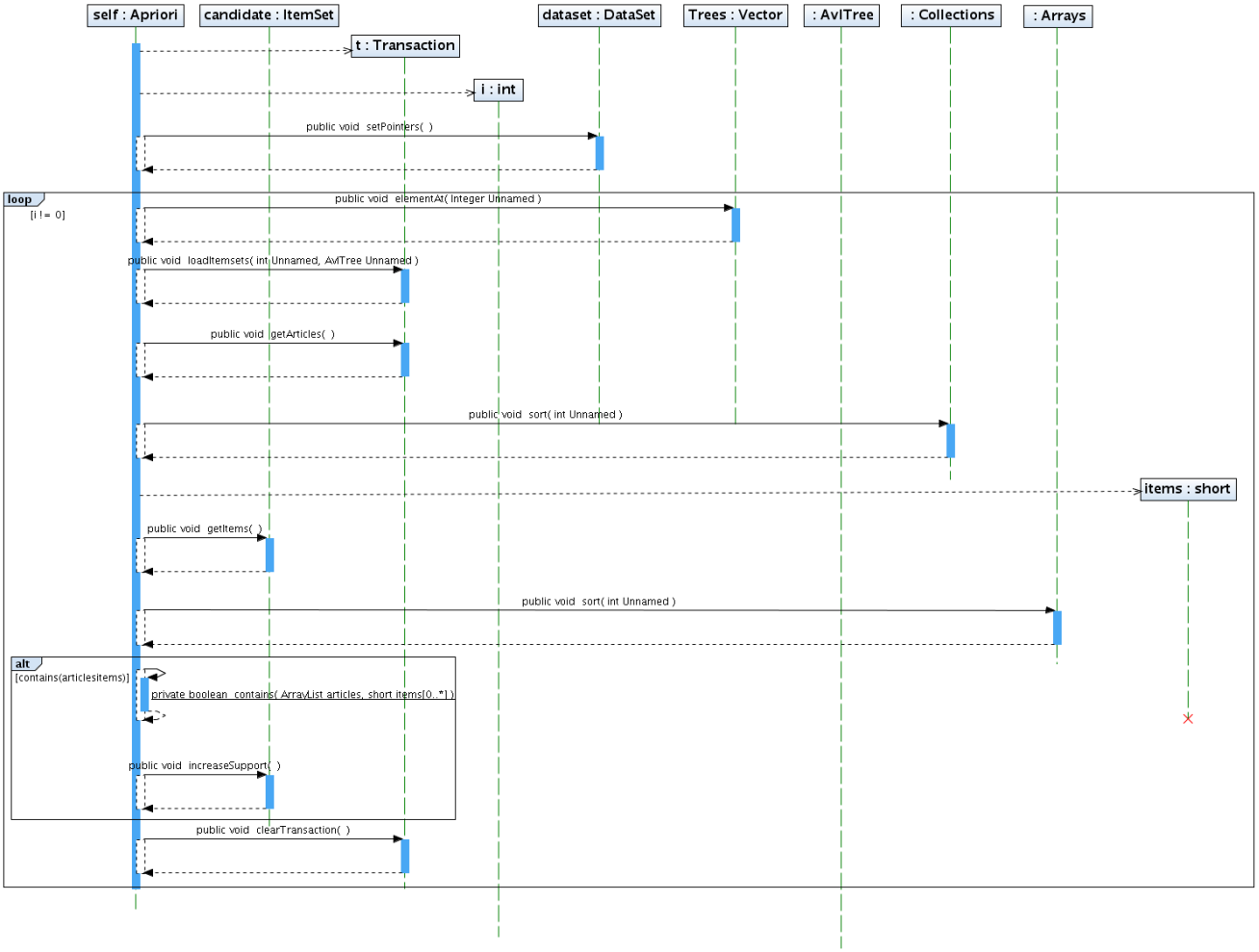

Figura 7.9: increaseSupport

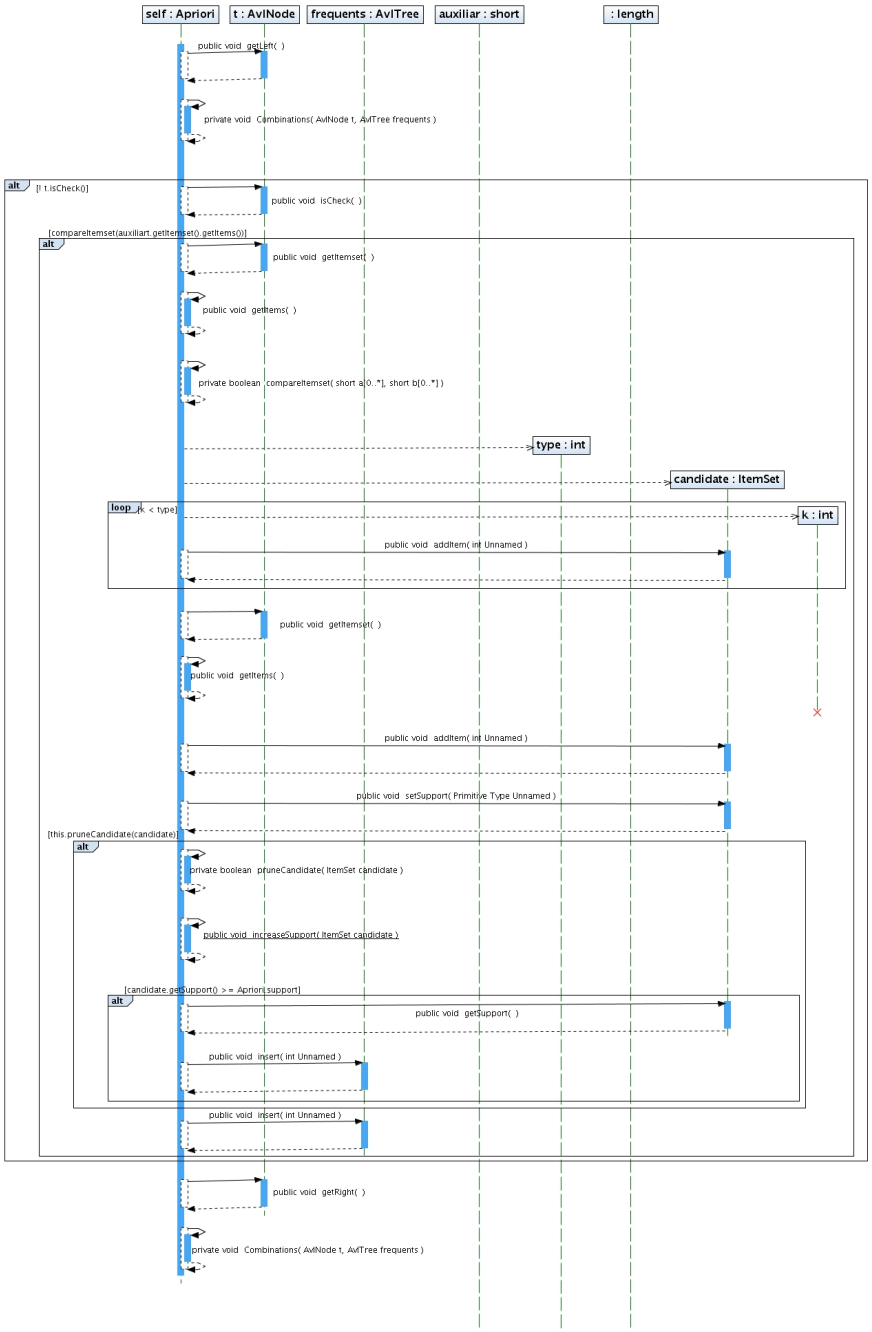

Figura 7.10: Combinations

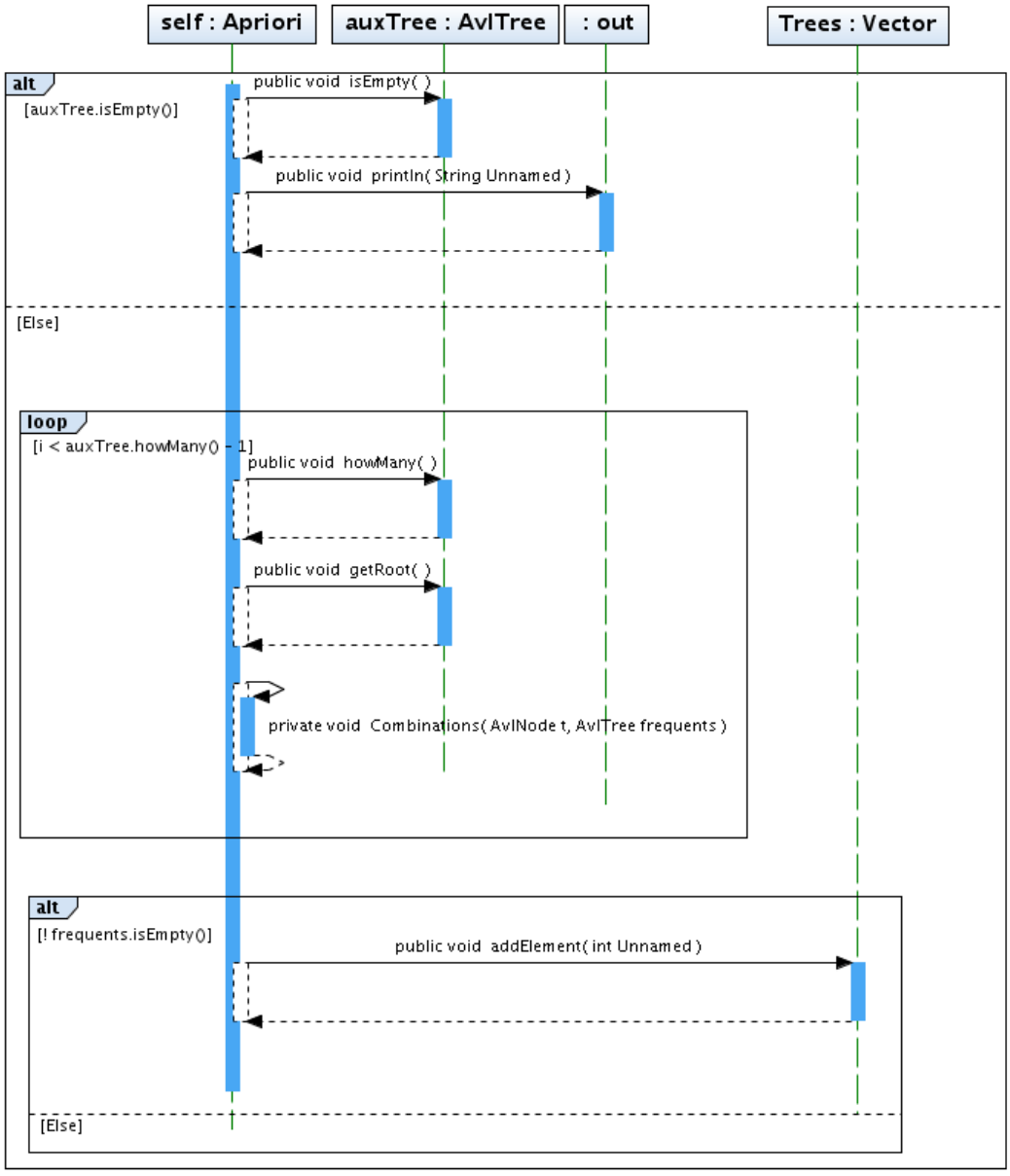

Figura 7.11: makeCandidates

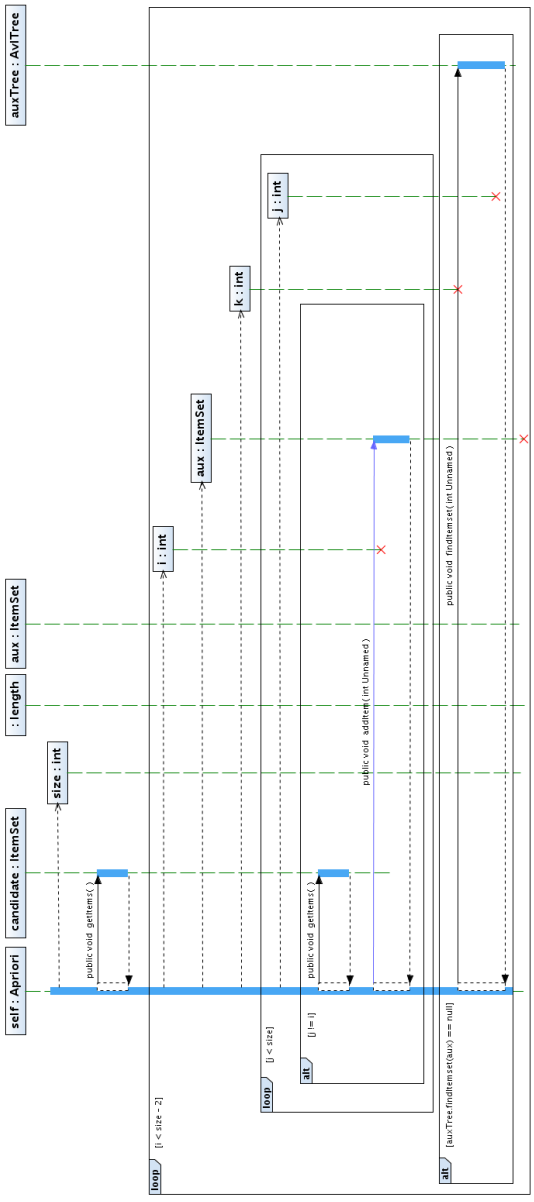

Figura 7.12: pruneCandidates

Clase EquipAsso

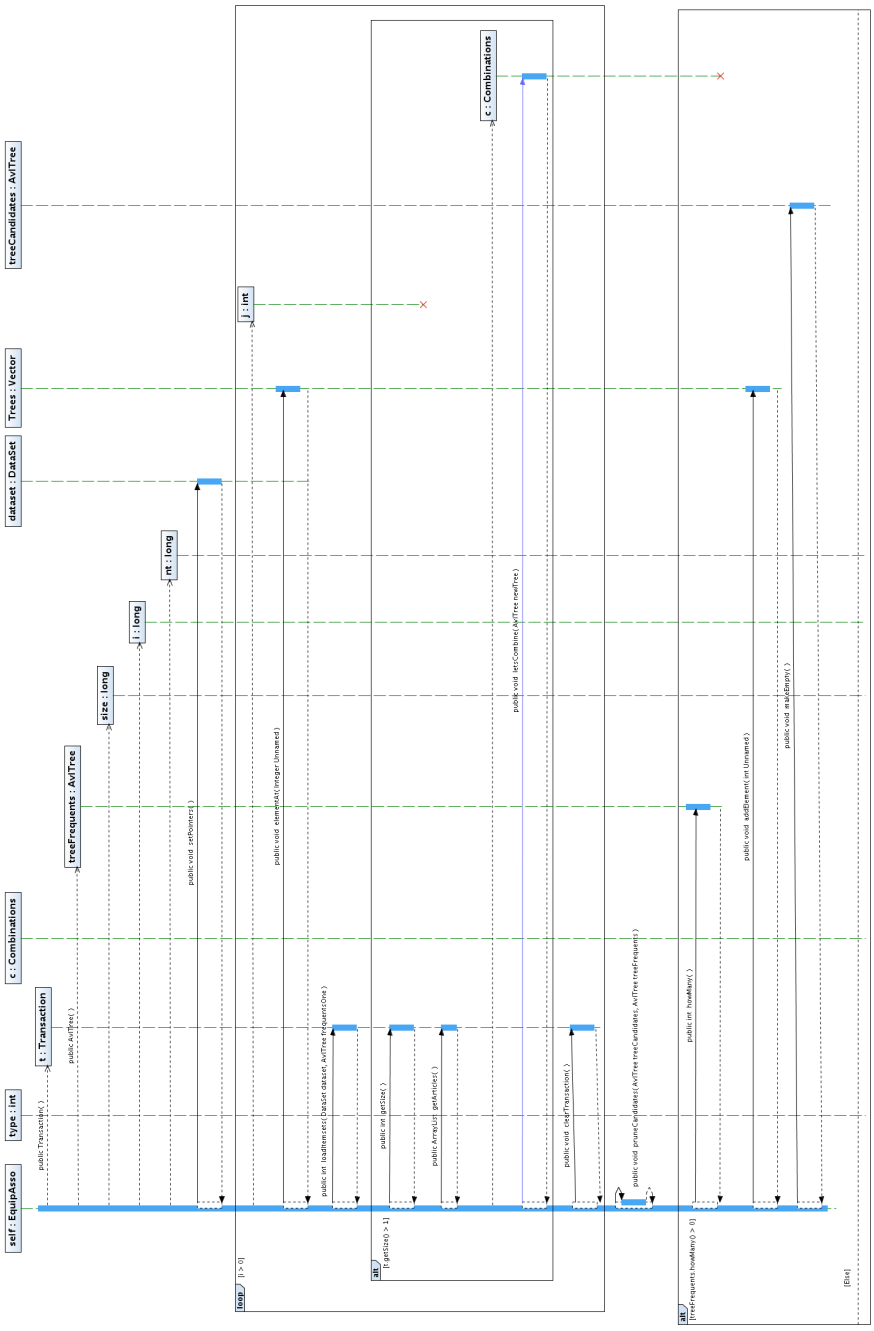

Figura 7.13: findInDataSet

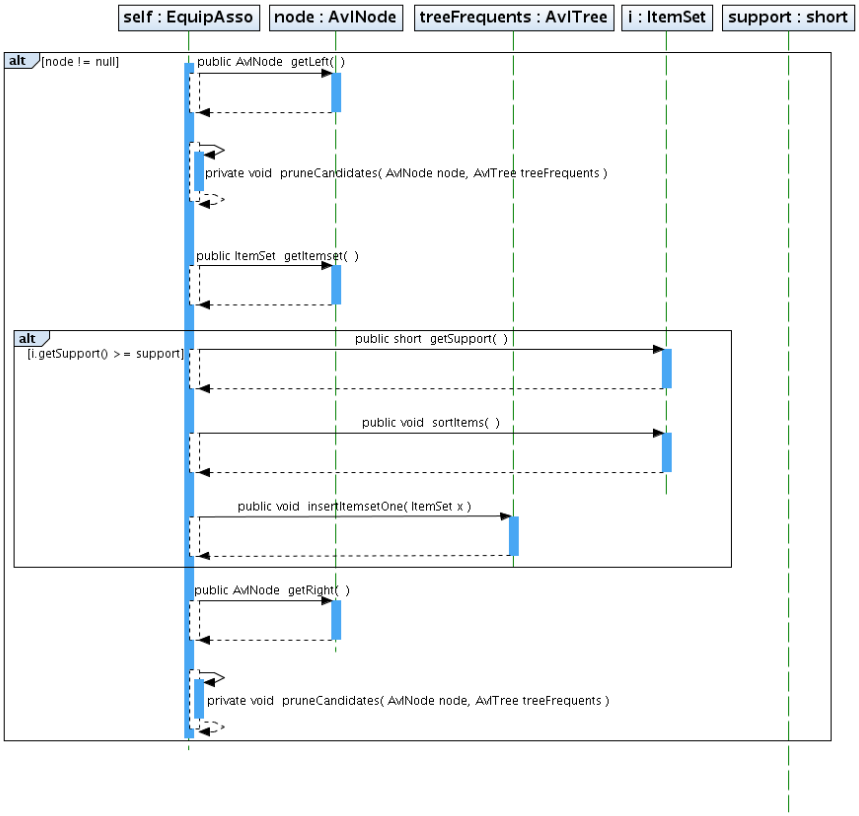

Figura 7.14: pruneCandidate-recursive

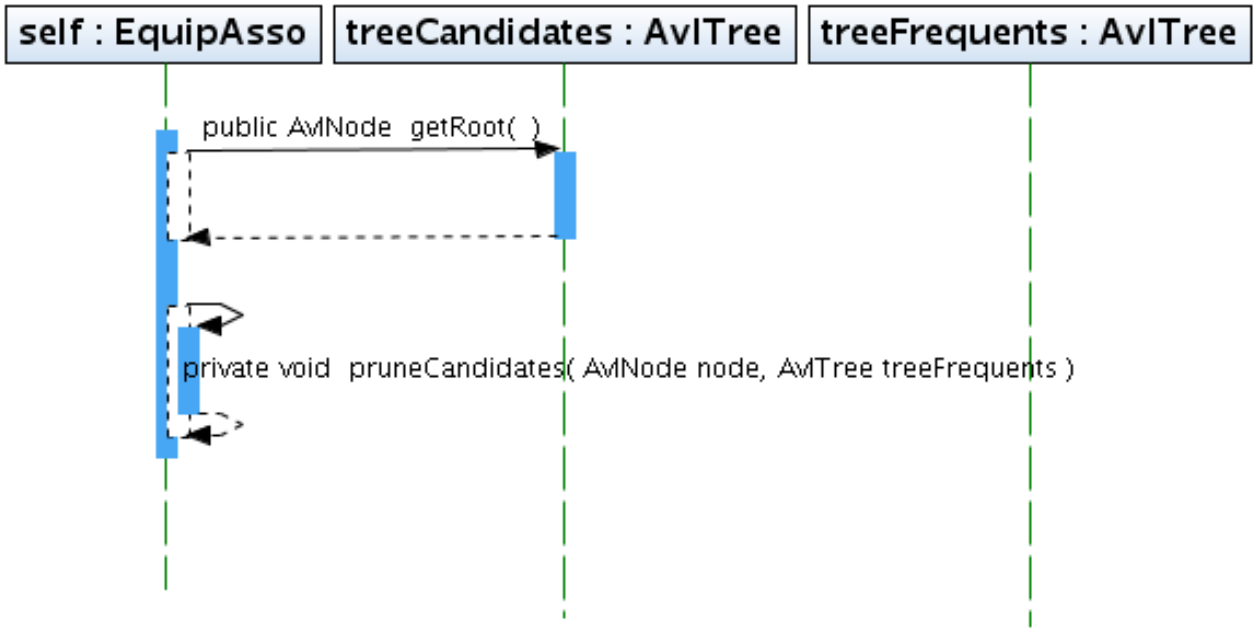

Figura 7.15: pruneCandidate-recursive

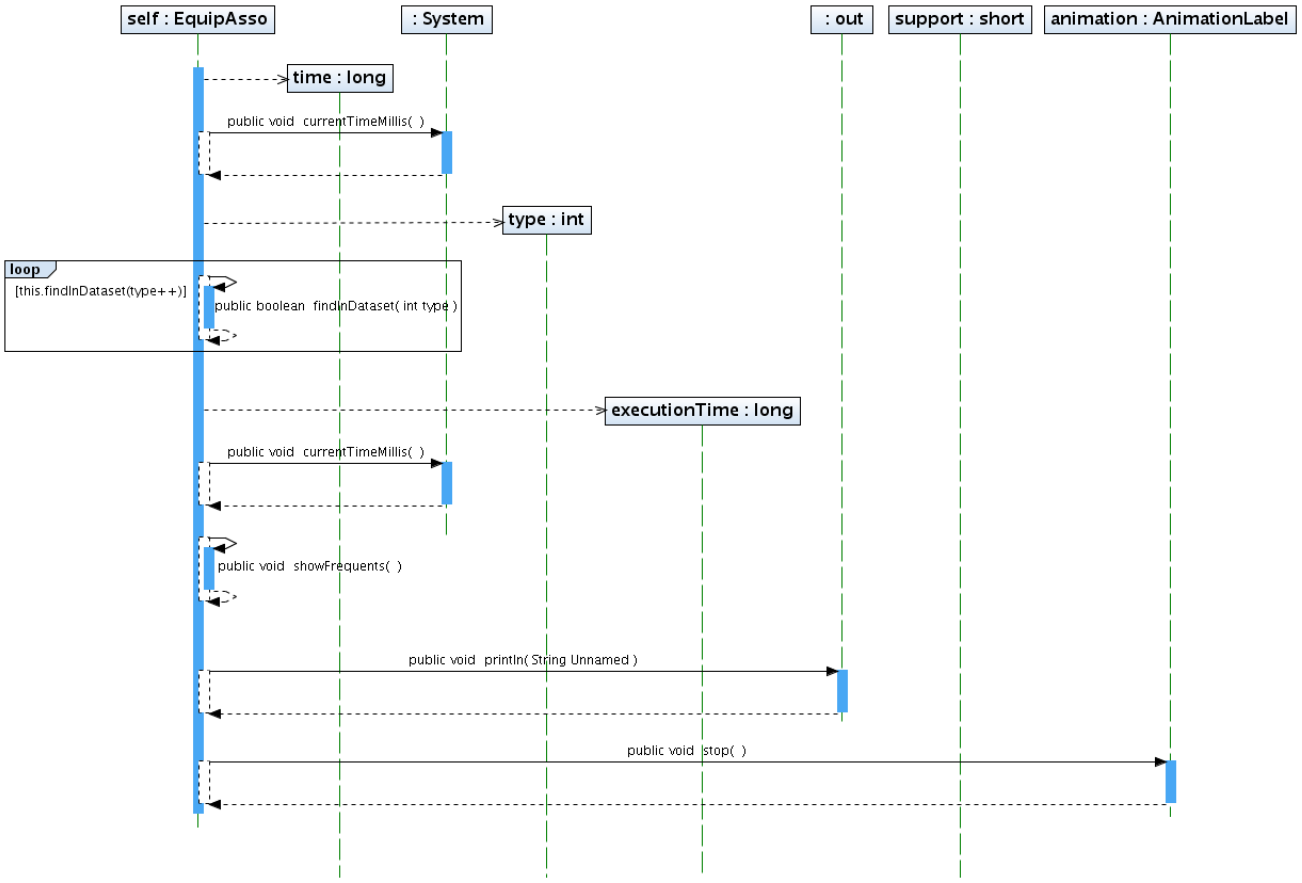

Figura 7.16: run

Clase ItemSet

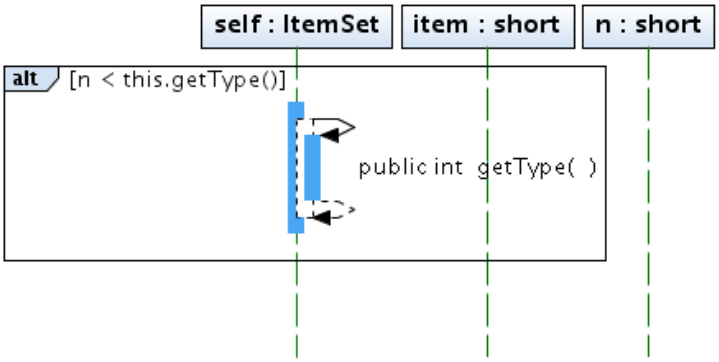

Figura 7.17: addItems

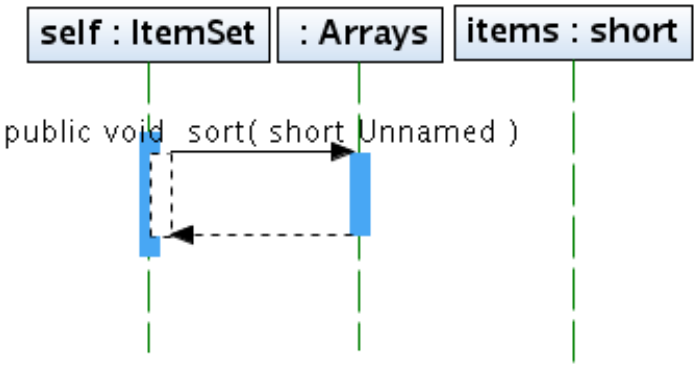

Figura 7.18: sortItems

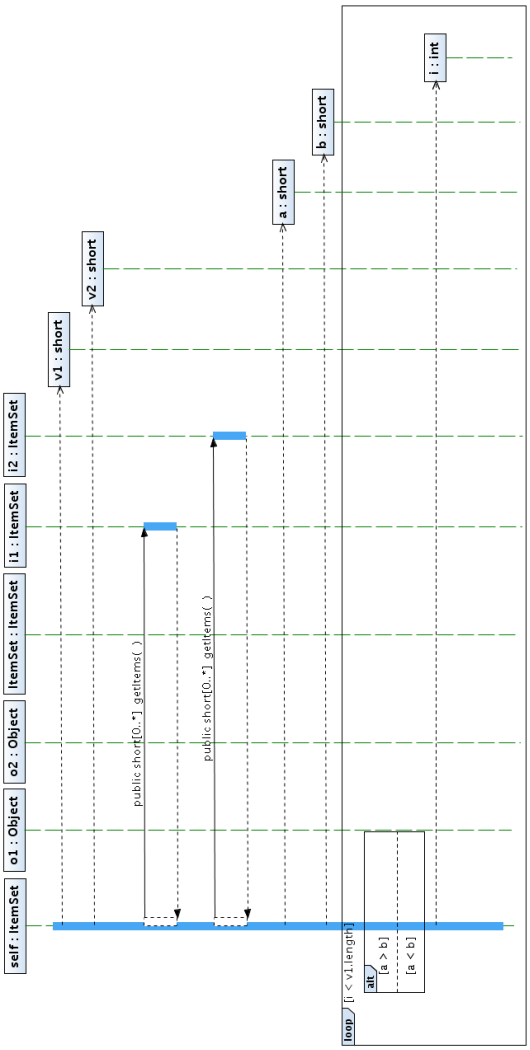

Figura 7.19: compare

Clase DataSet

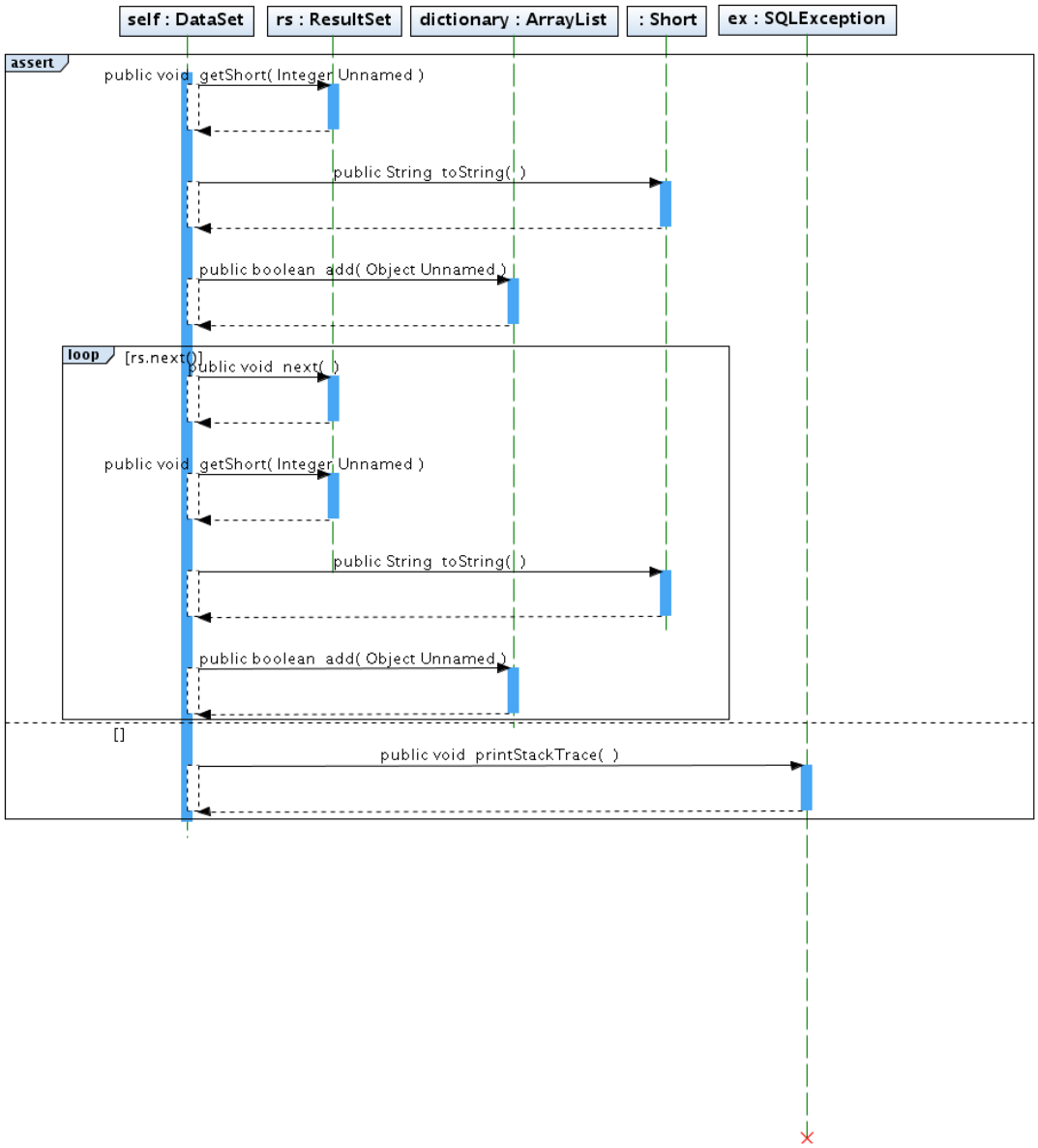

Figura 7.20: buildDictionary

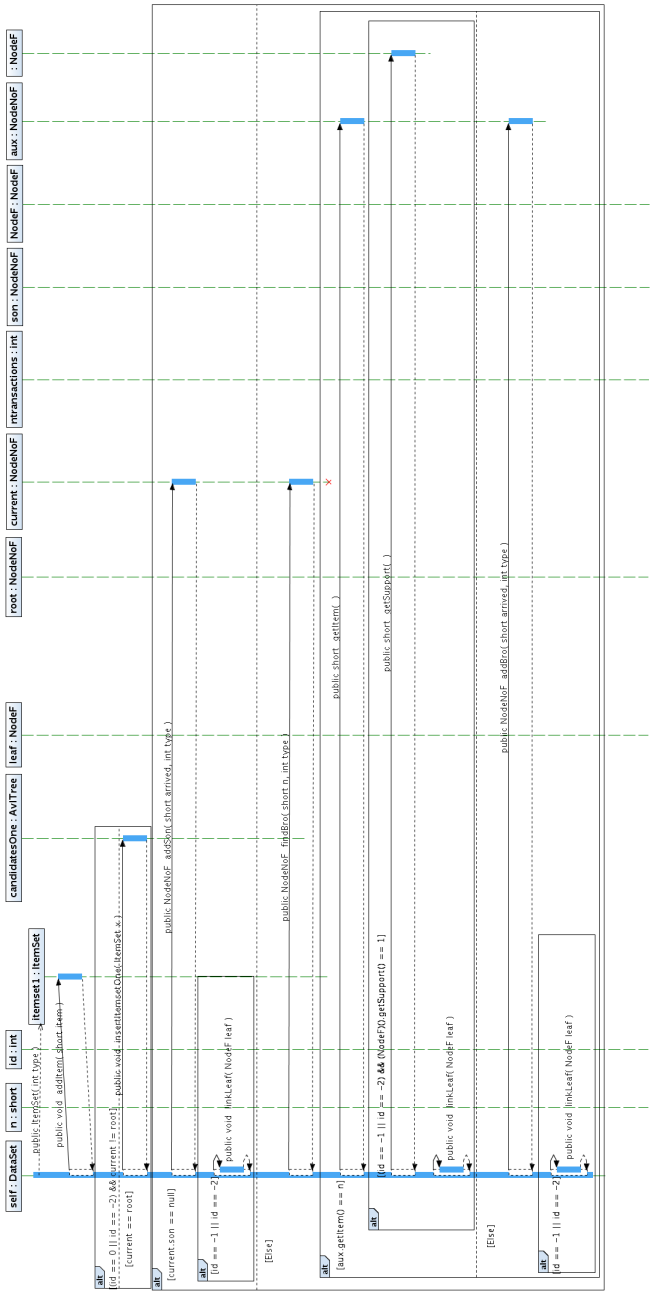

Figura 7.21: buildNTree

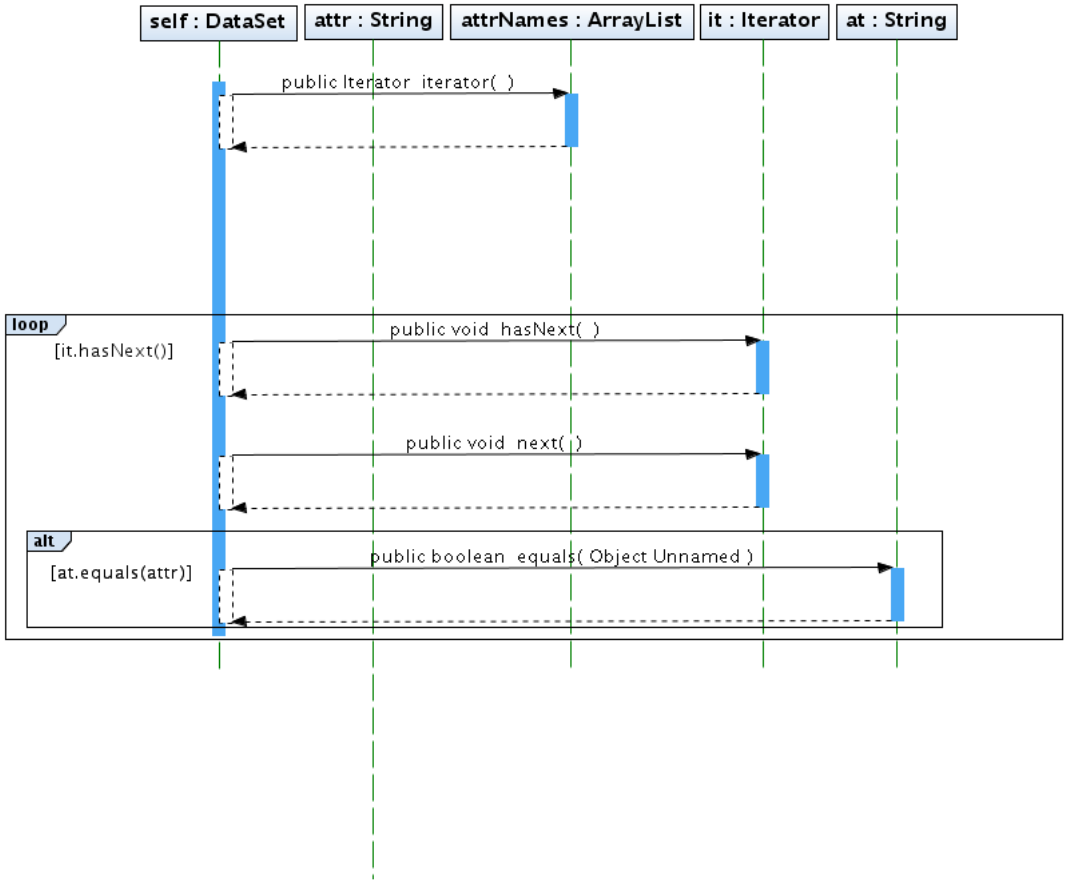

Figura 7.22: findAttrName

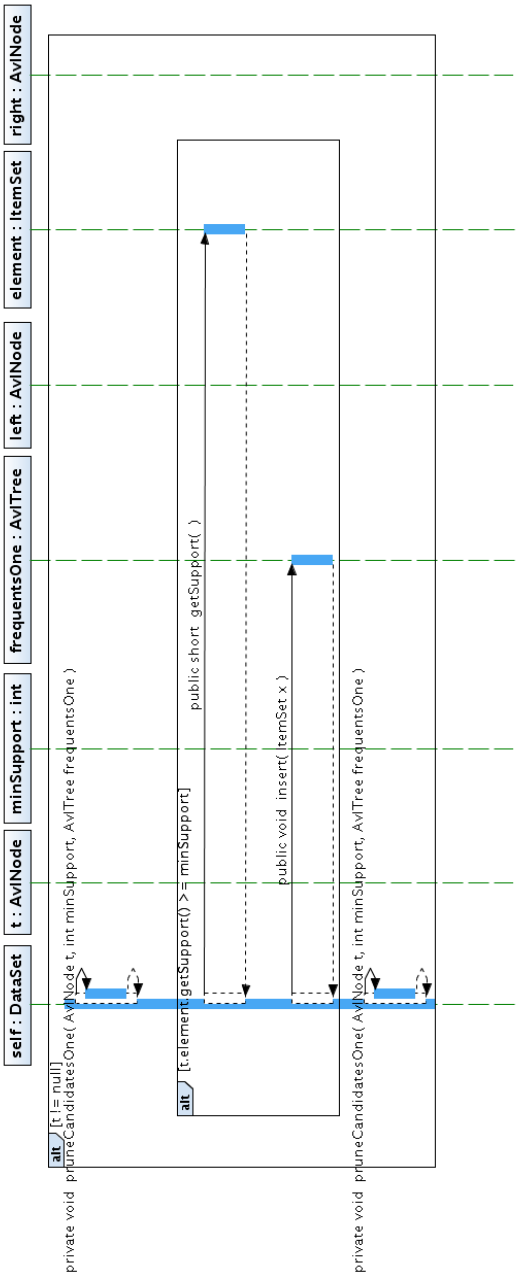

Figura 7.23: pruneCandidatesOne

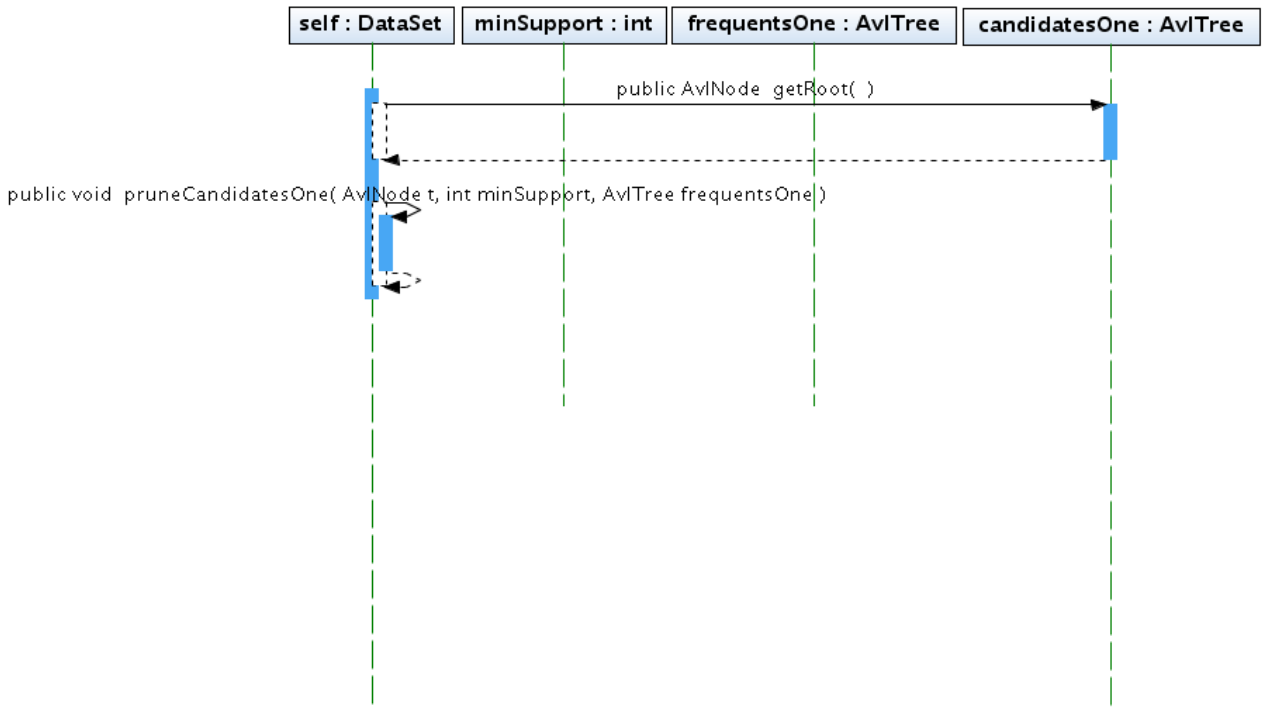

Figura 7.24: public-pruneCandidatesOne

Clase FPGrowth

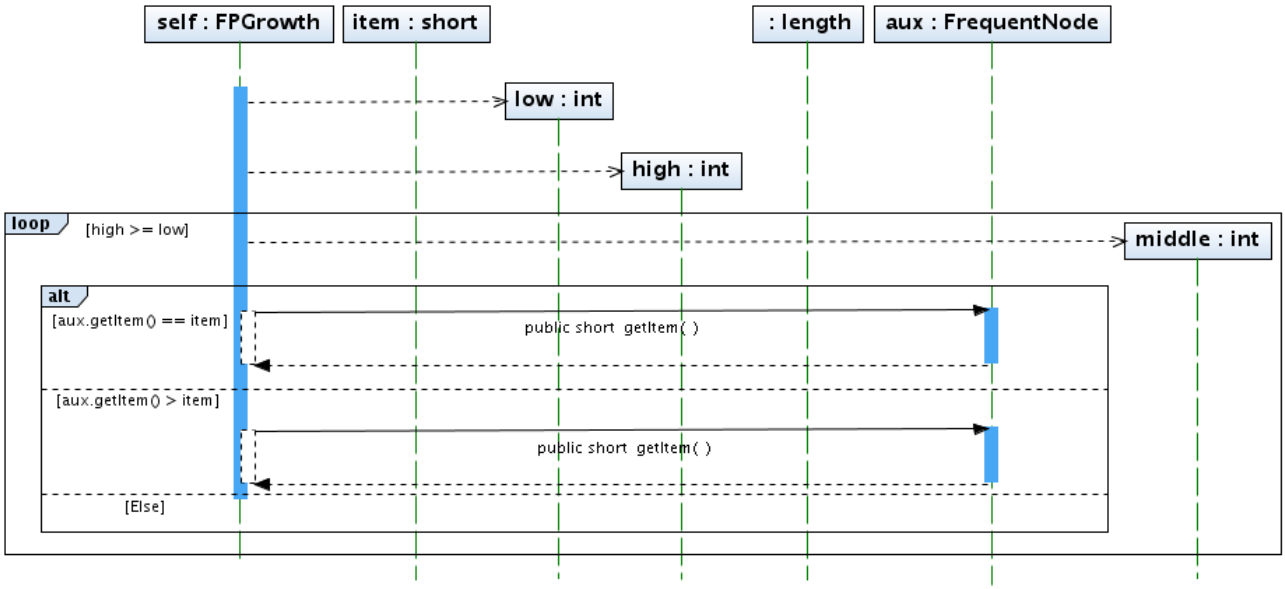

Figura 7.25: buildFrequentsNodes

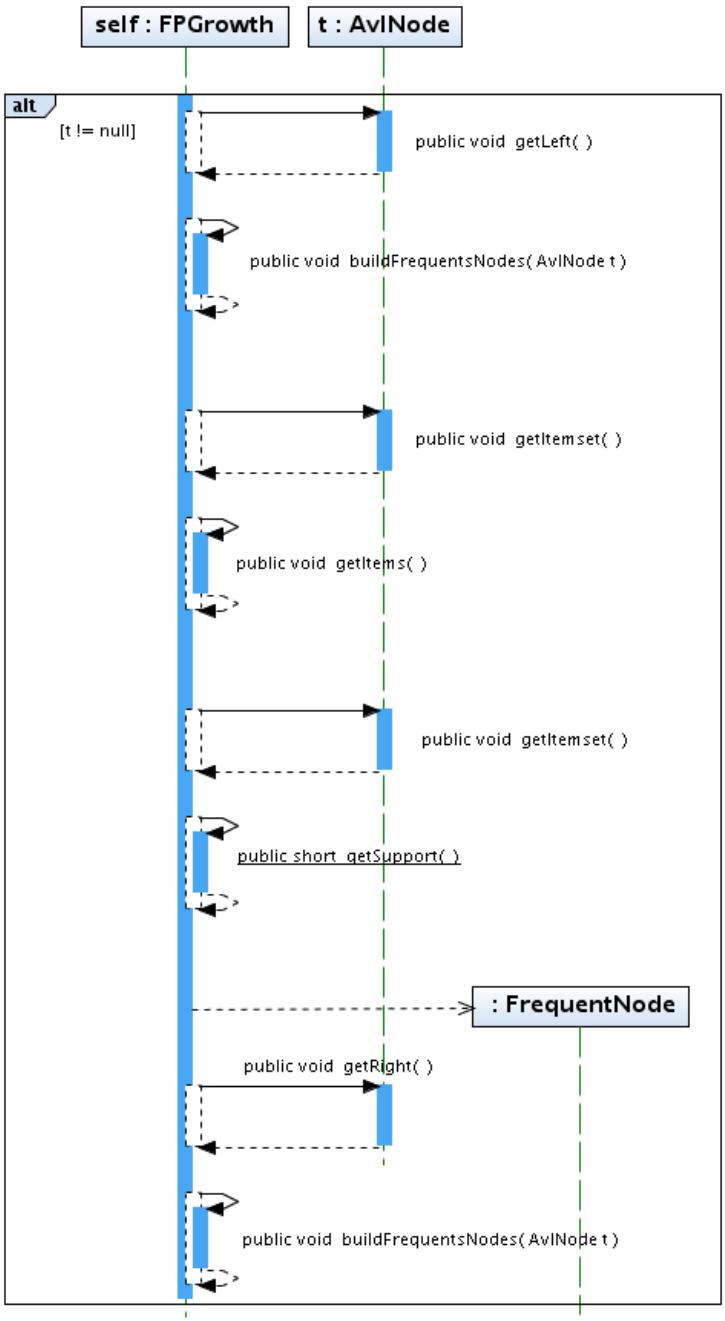

Figura 7.26: findNode

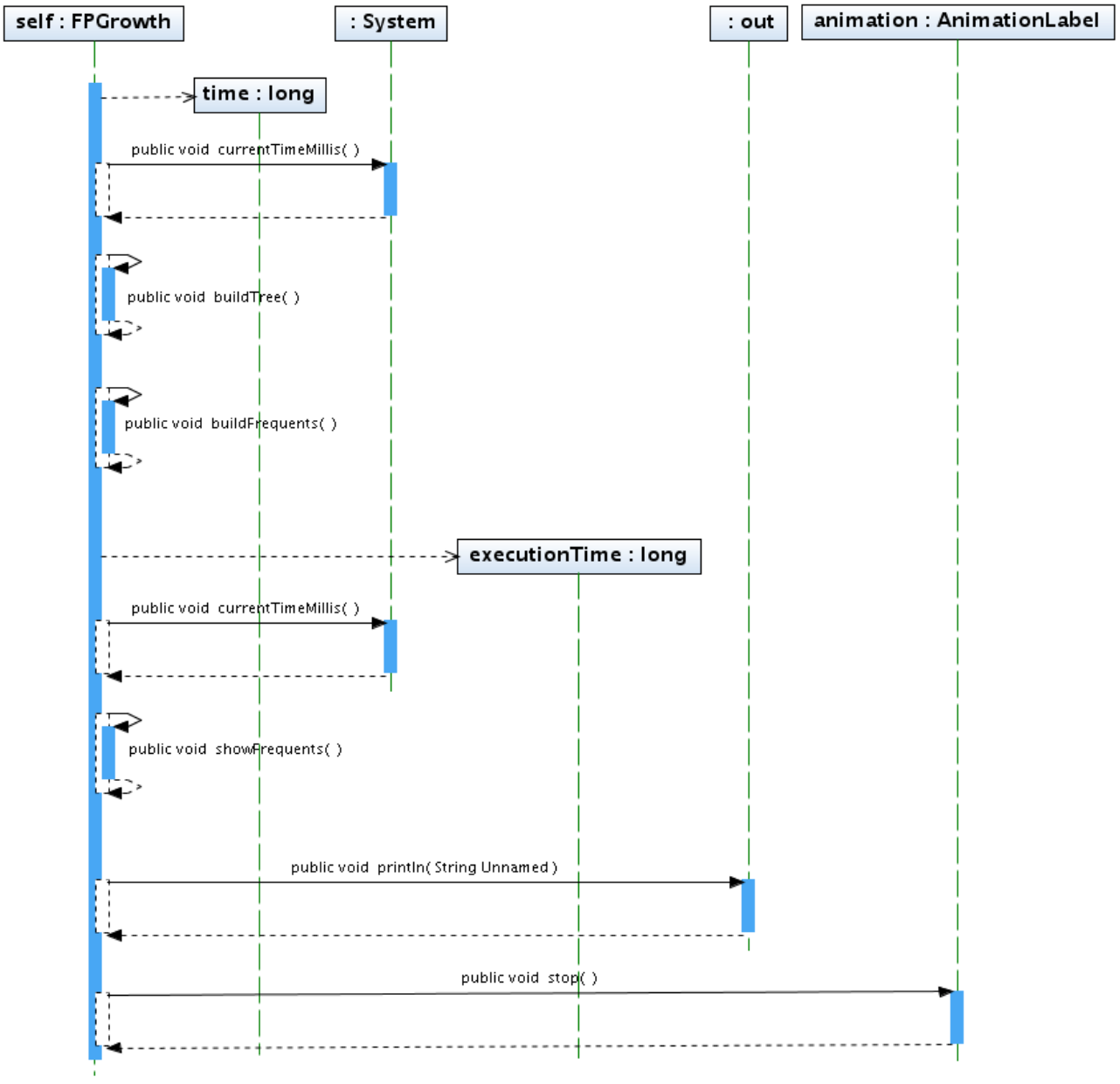

Figura 7.27: run

Clase AvlTree

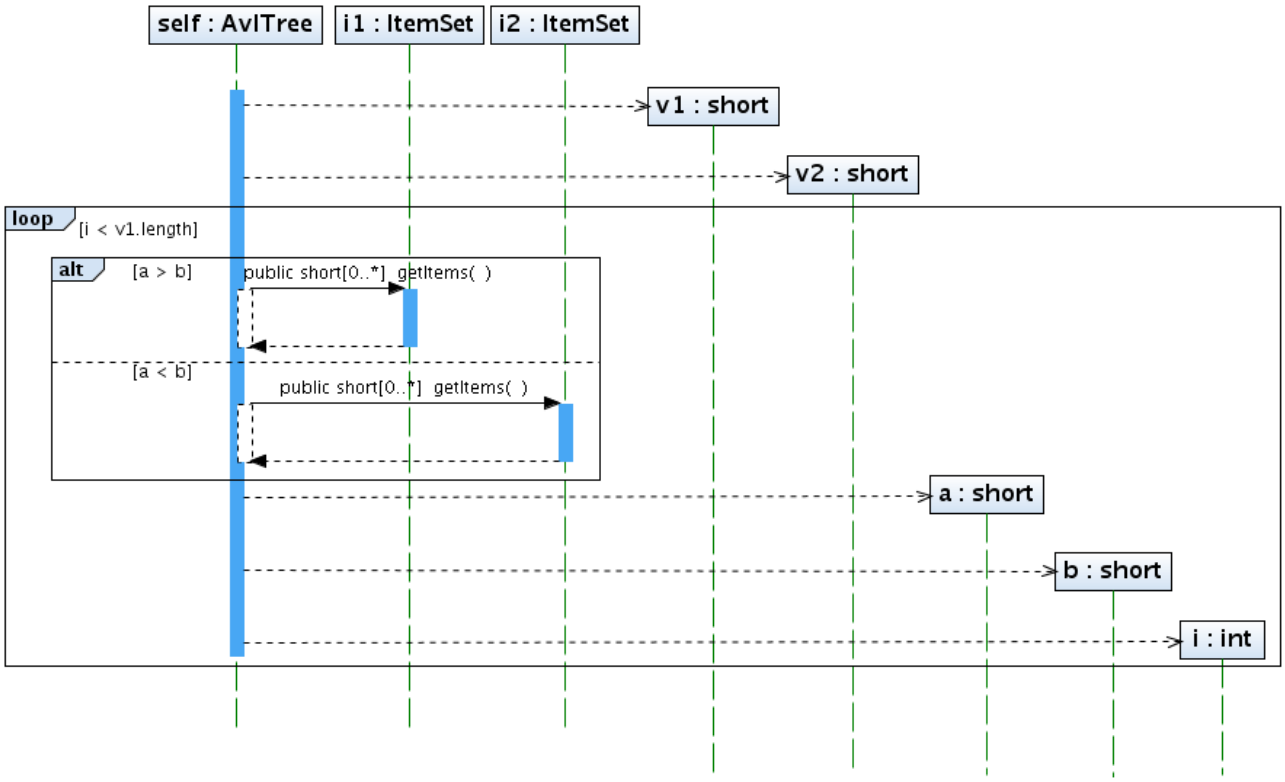

Figura 7.28: compareItemSet

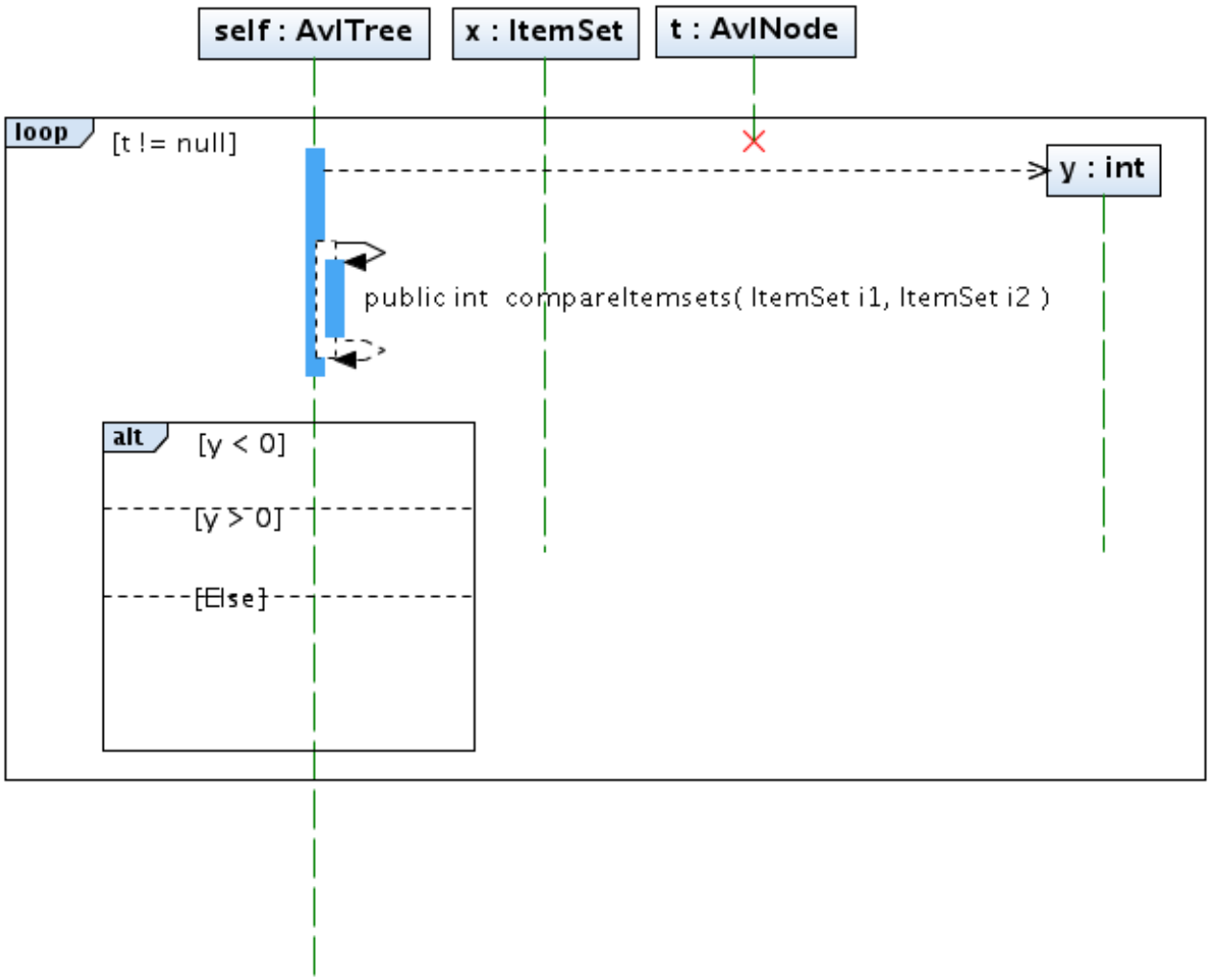

Figura 7.29: find

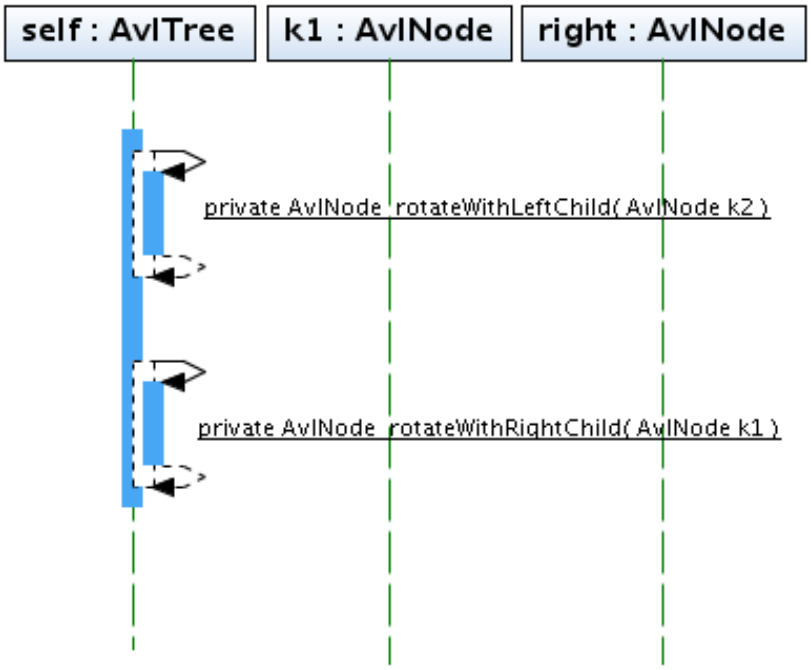

Figura 7.30: doubleWithRightChild

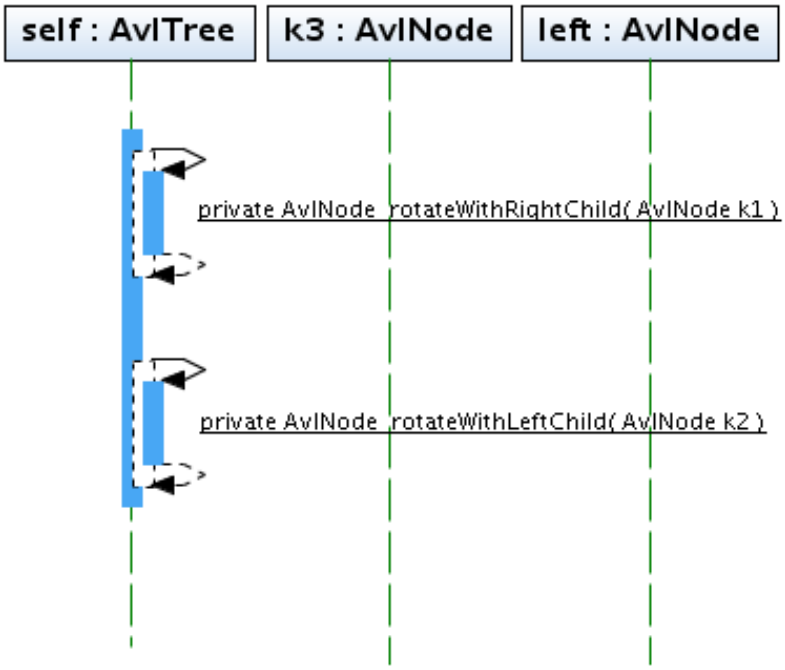

Figura 7.31: doubleWithLeftChild

Clase Transaction

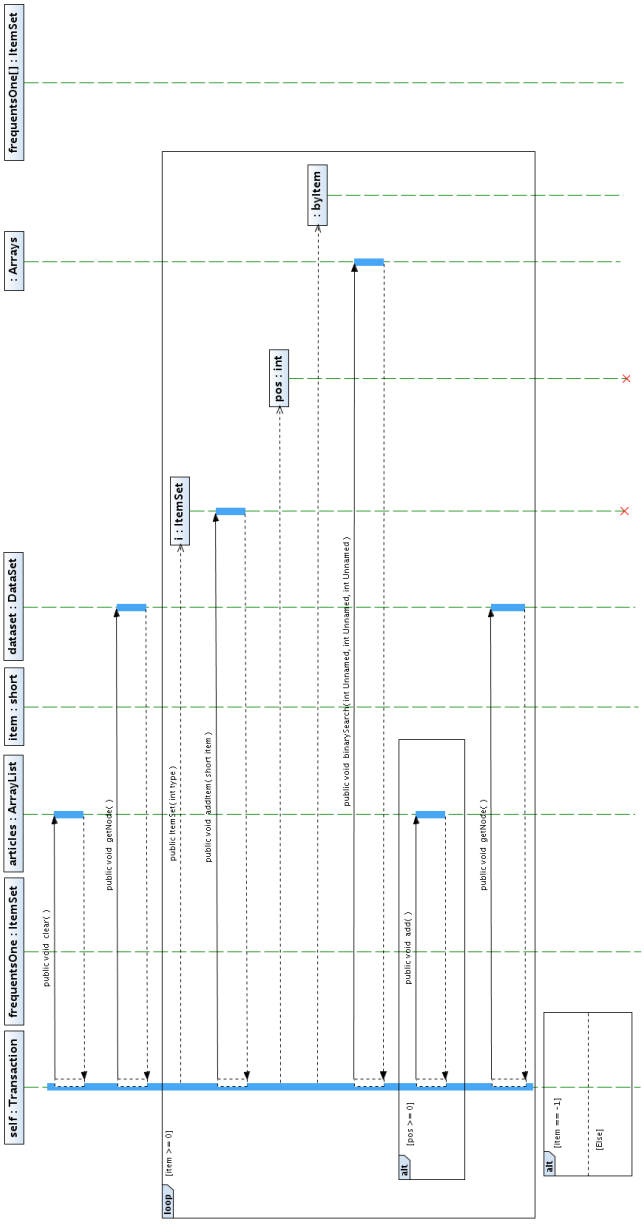

Figura 7.32: loadItemset

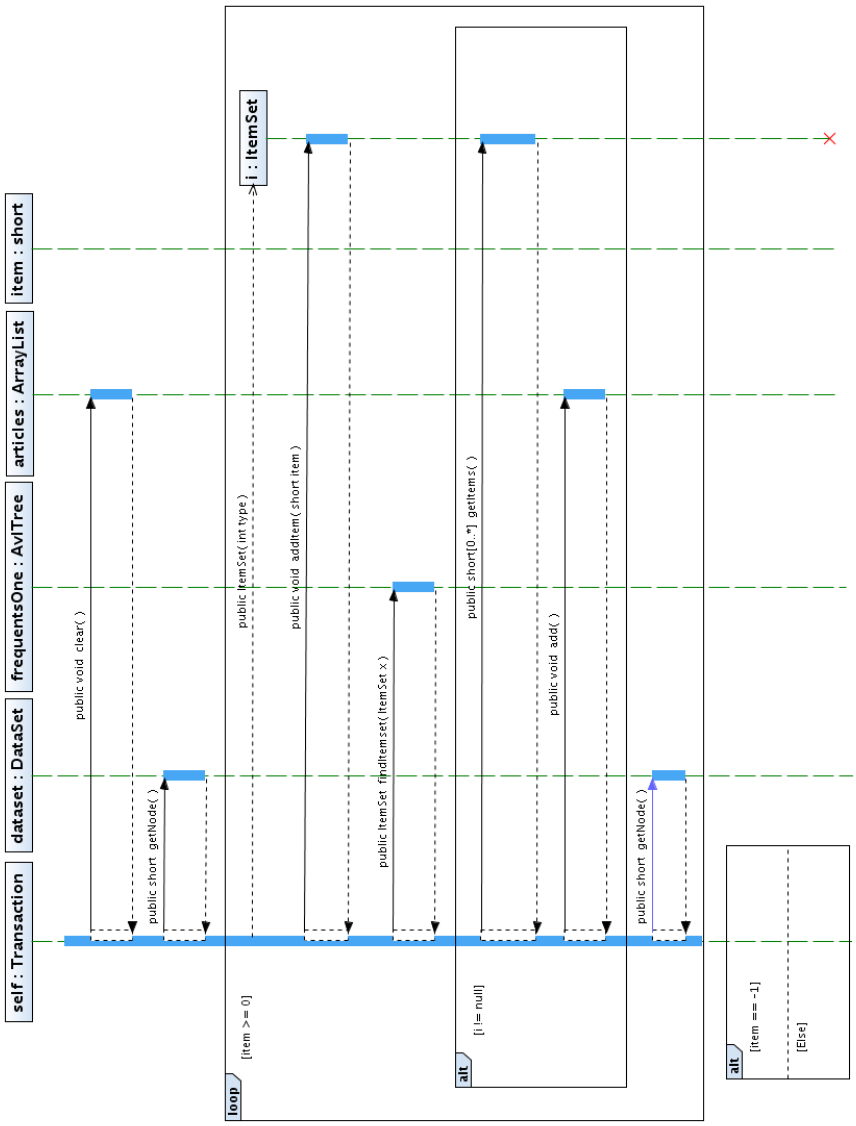

Figura 7.33: public-loadItemset

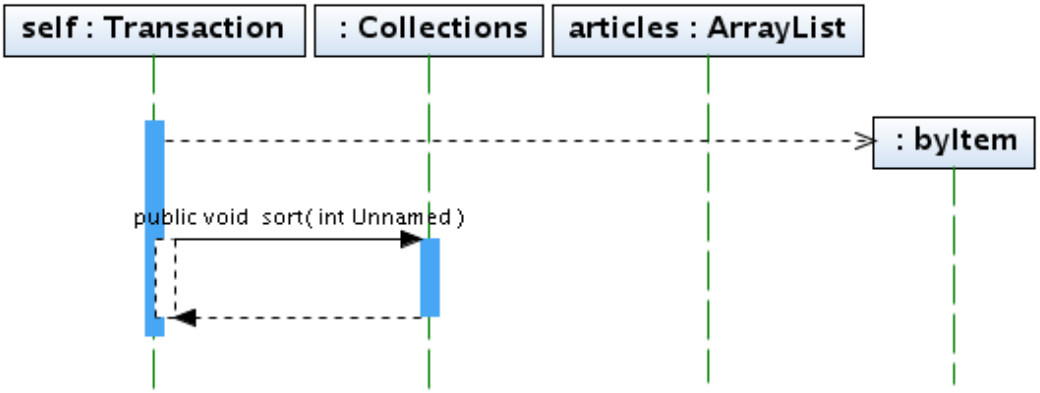

Figura 7.34: sortByItem

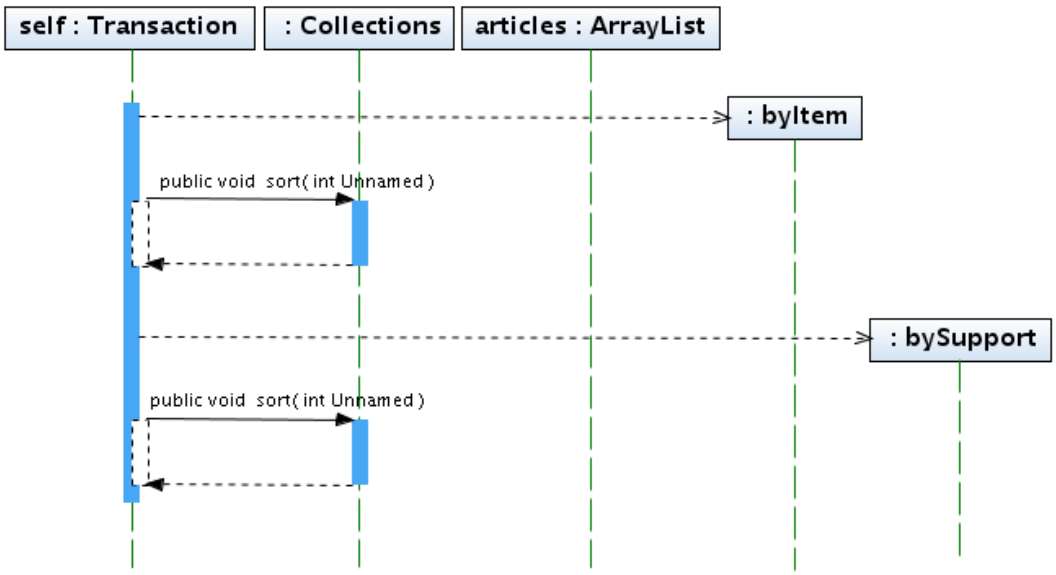

Figura 7.35: sortBySupport

Clase NodeF

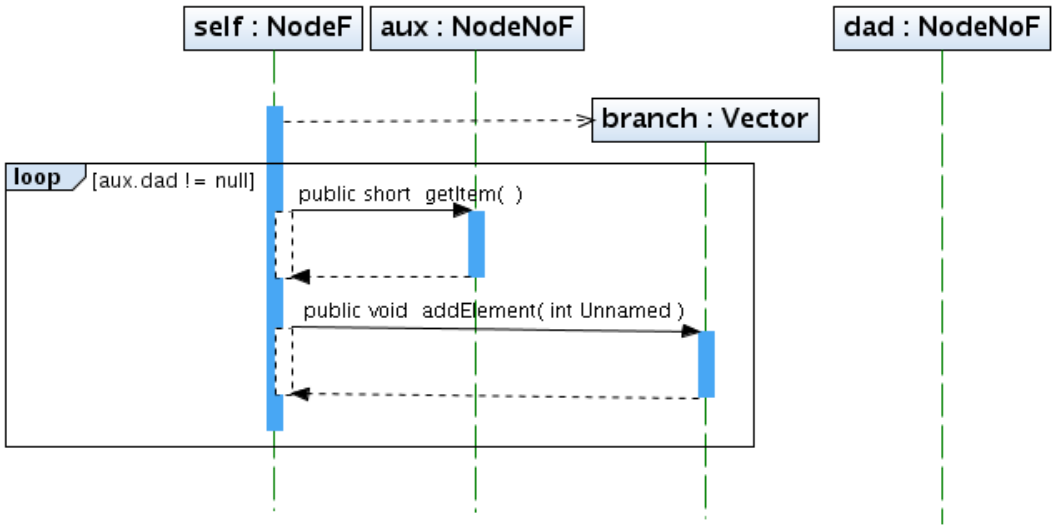

Figura 7.36: getBranch

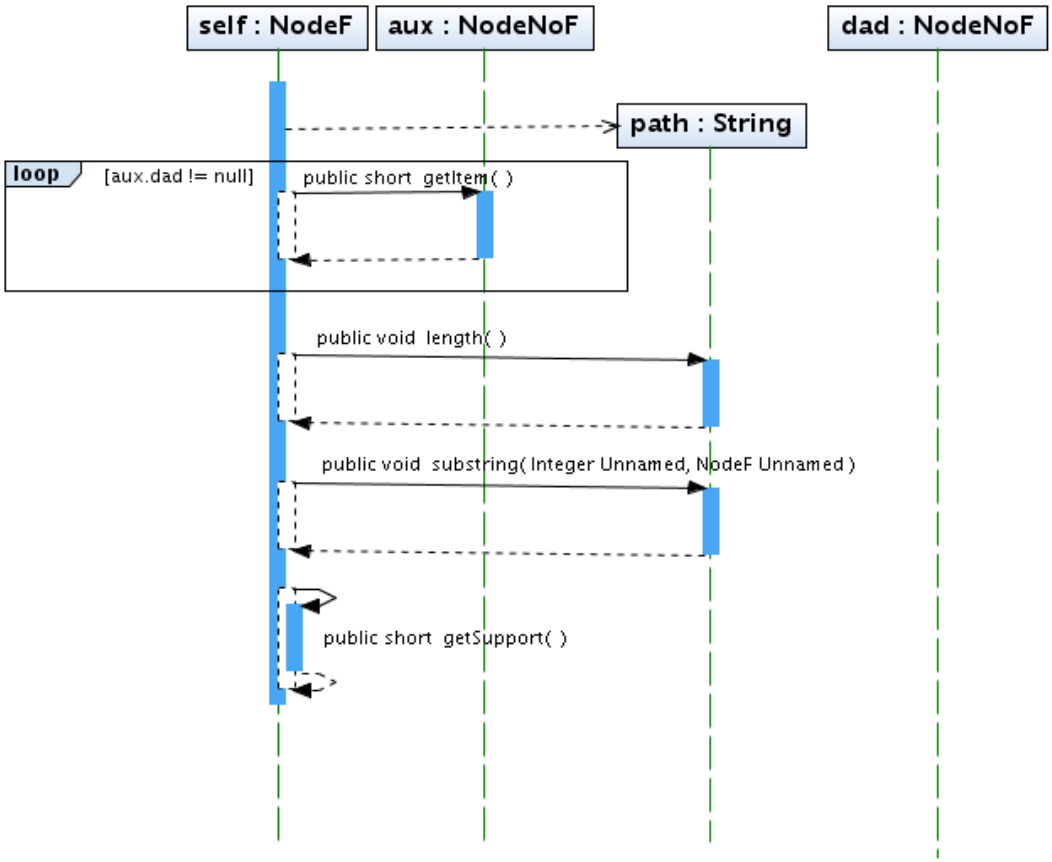

Figura 7.37: getPath

Clase NodeNoF

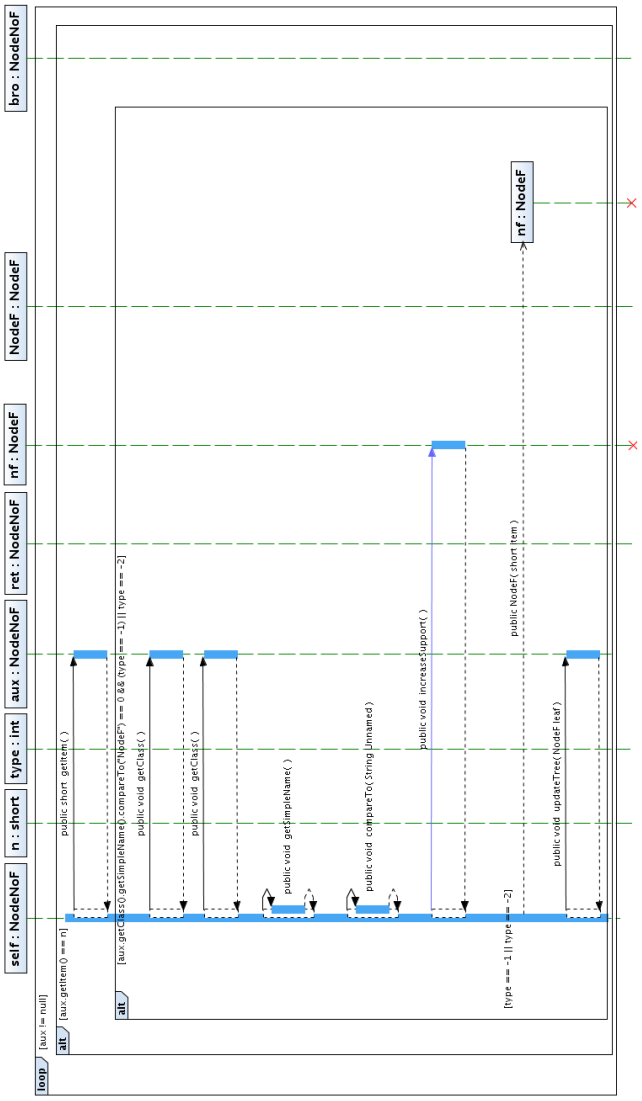

Figura 7.38: findBro

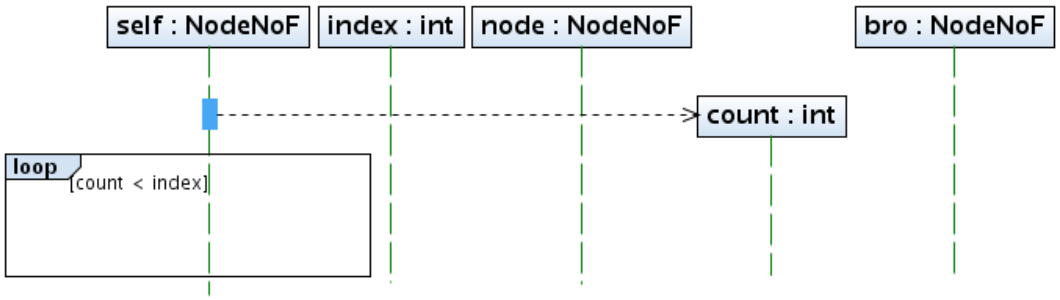

Figura 7.39: getChild

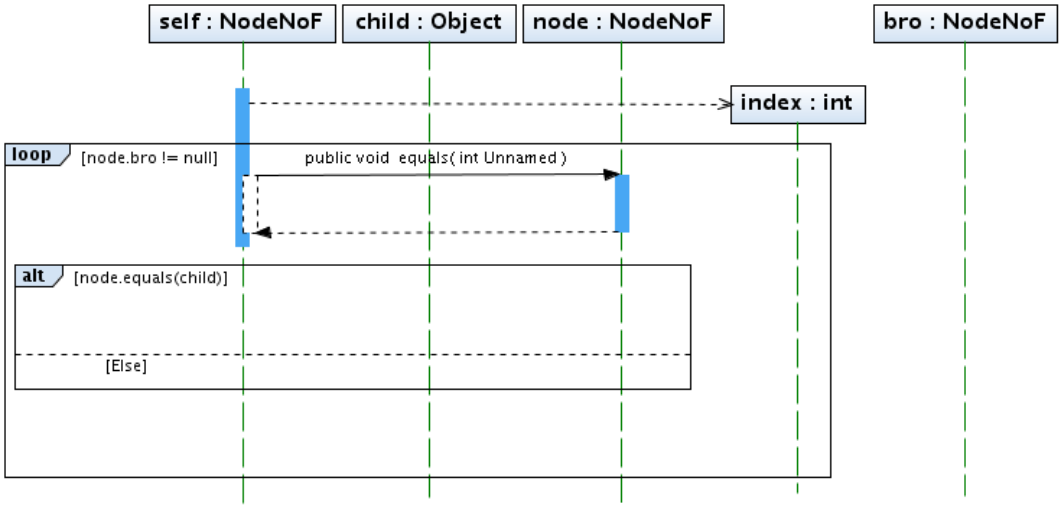

Figura 7.40: getIndexOfChild

Clase C45

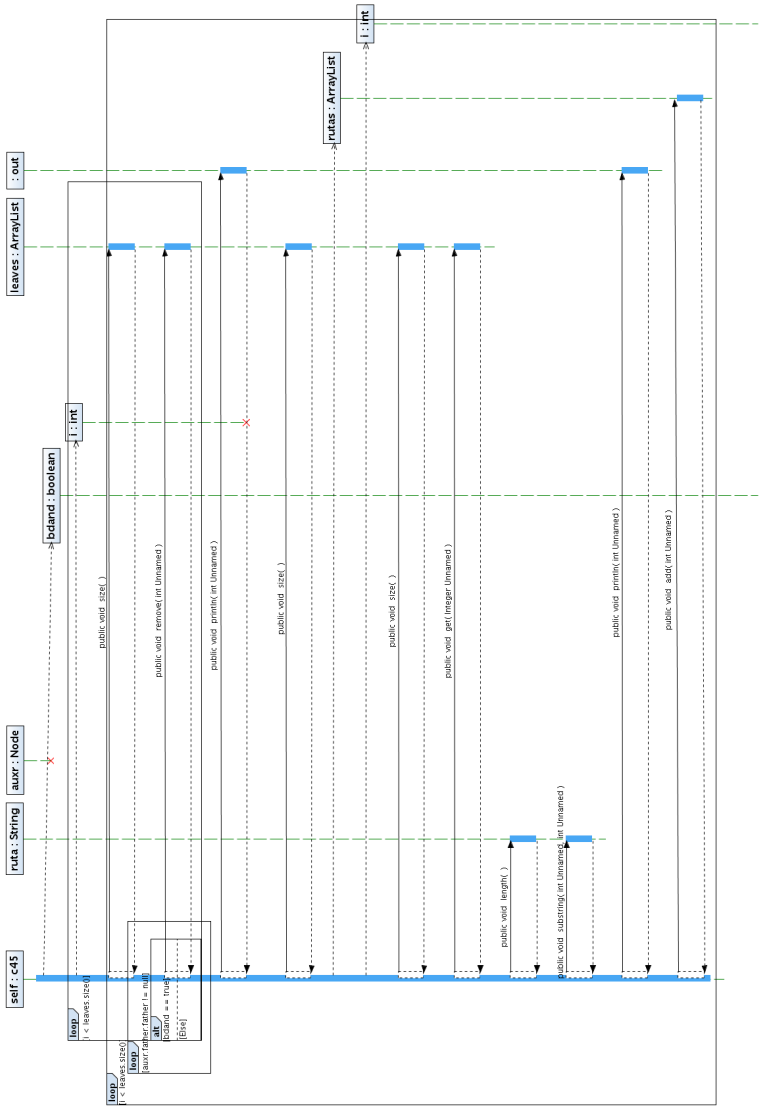

Figura 7.41: C45Rules

Clase myHasMap

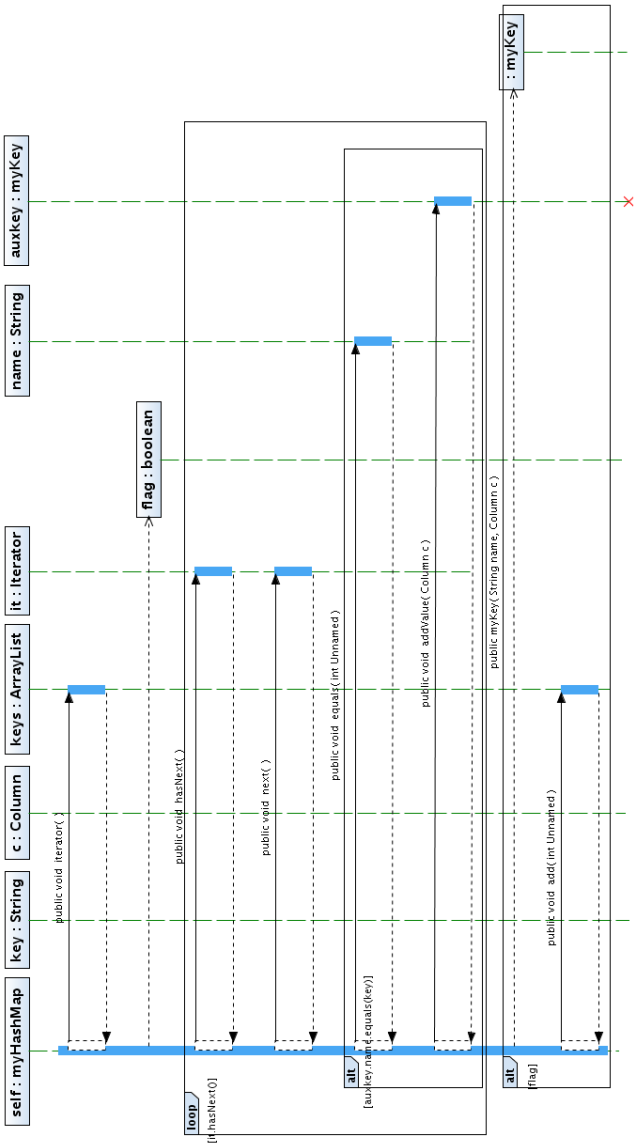

Figura 7.42: addColumn

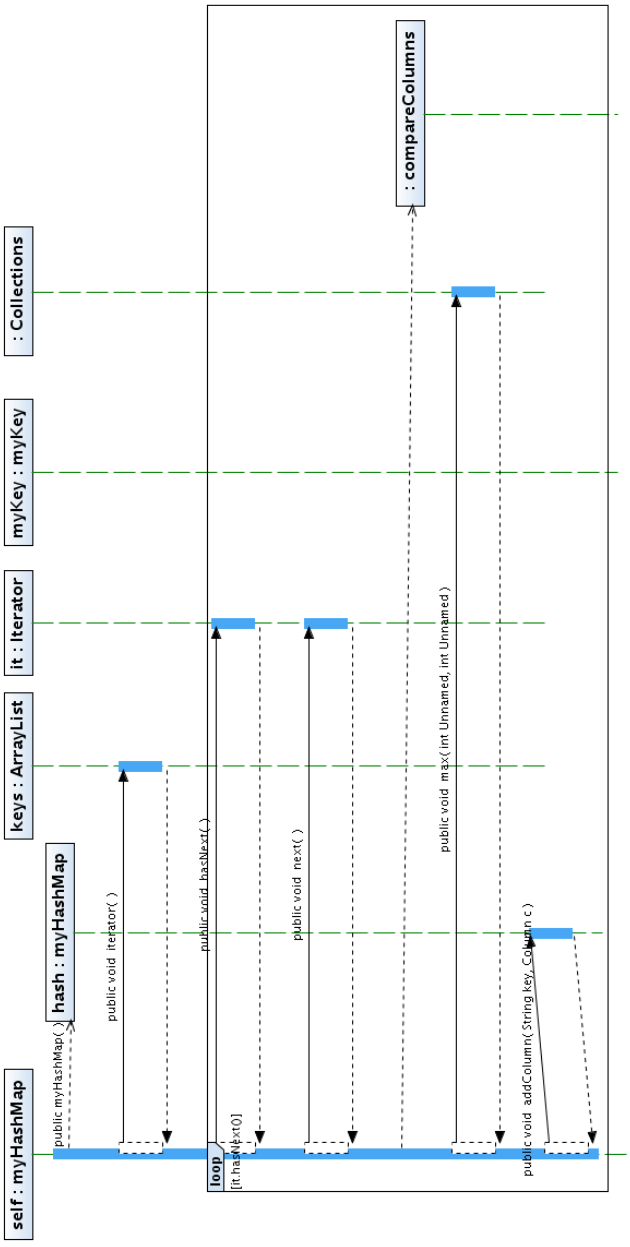

Figura 7.43: SearchColumn

Clase Route

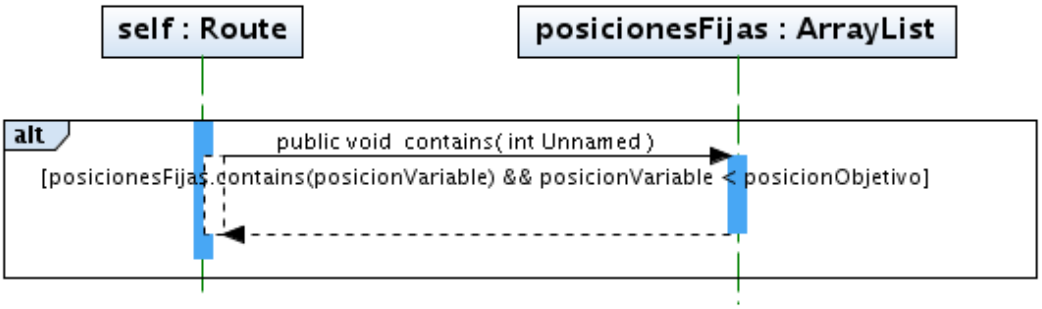

Figura 7.44: avanceVariable

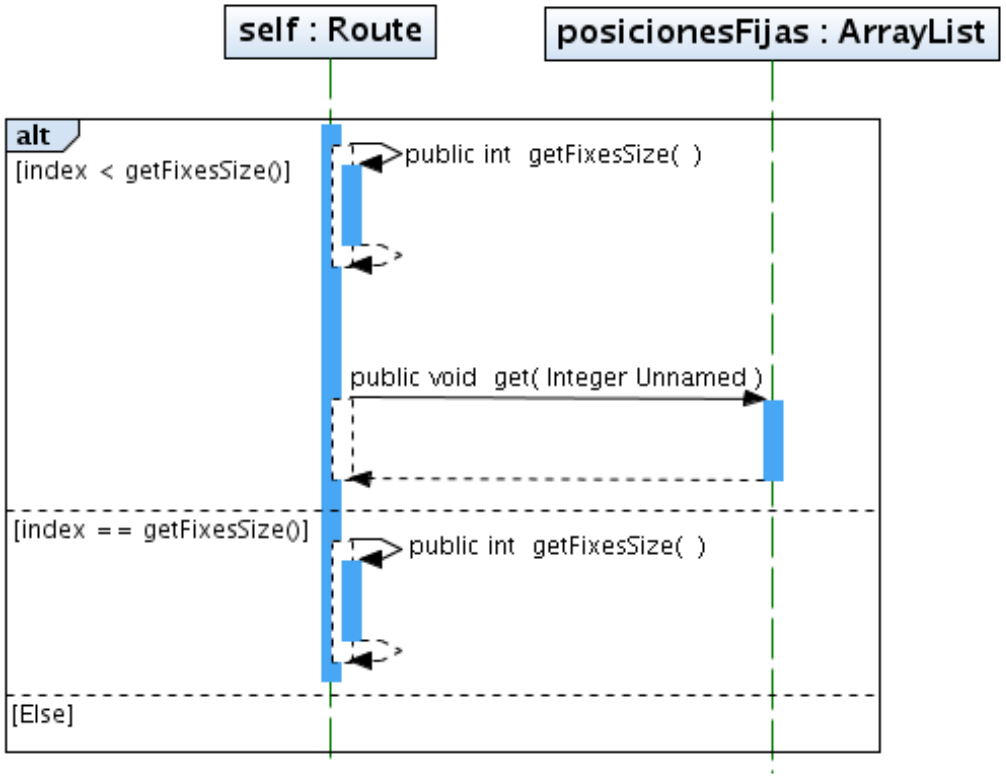

Figura 7.45: getIndex

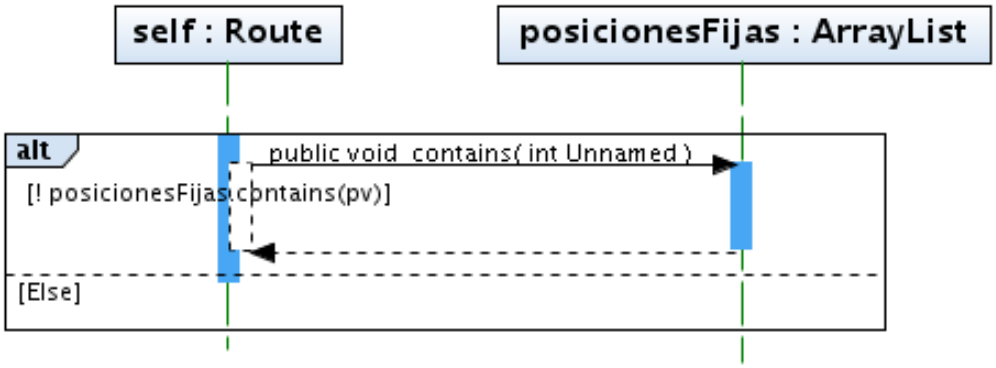

Figura 7.46: setPoscicionVariable

Clase TreeCounter

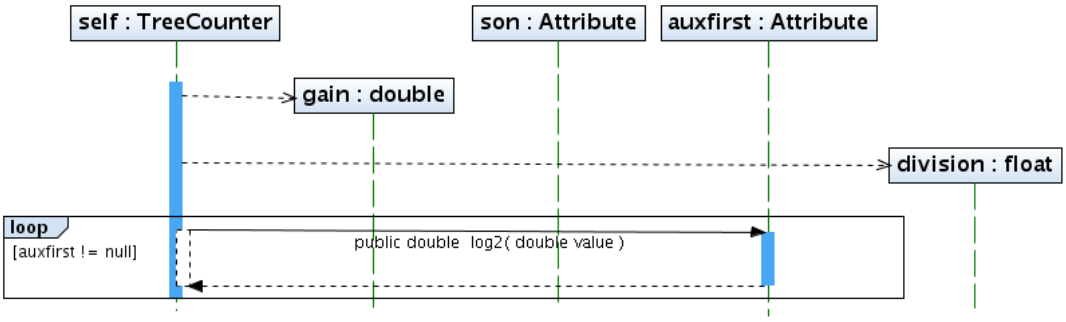

Figura 7.47: firstGain

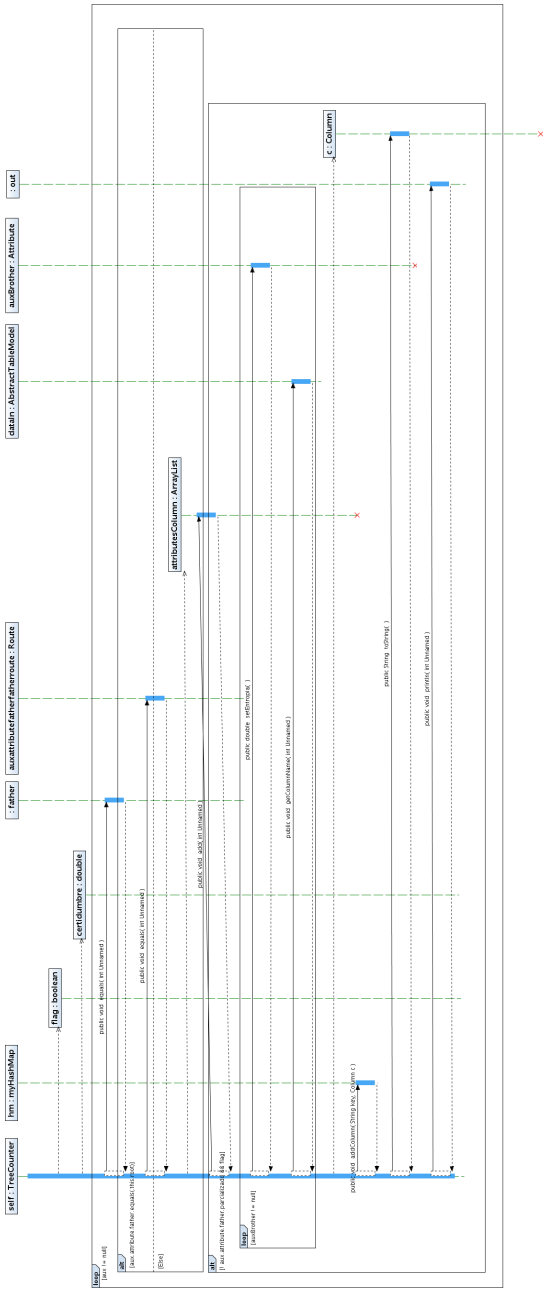

Figura 7.48: gain

Clase Attribute

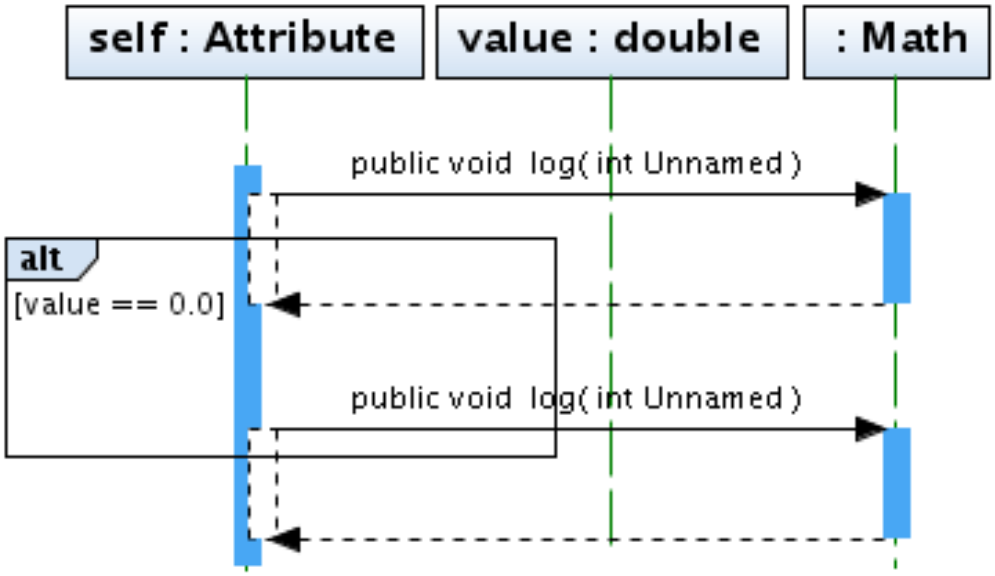

Figura 7.49: log2

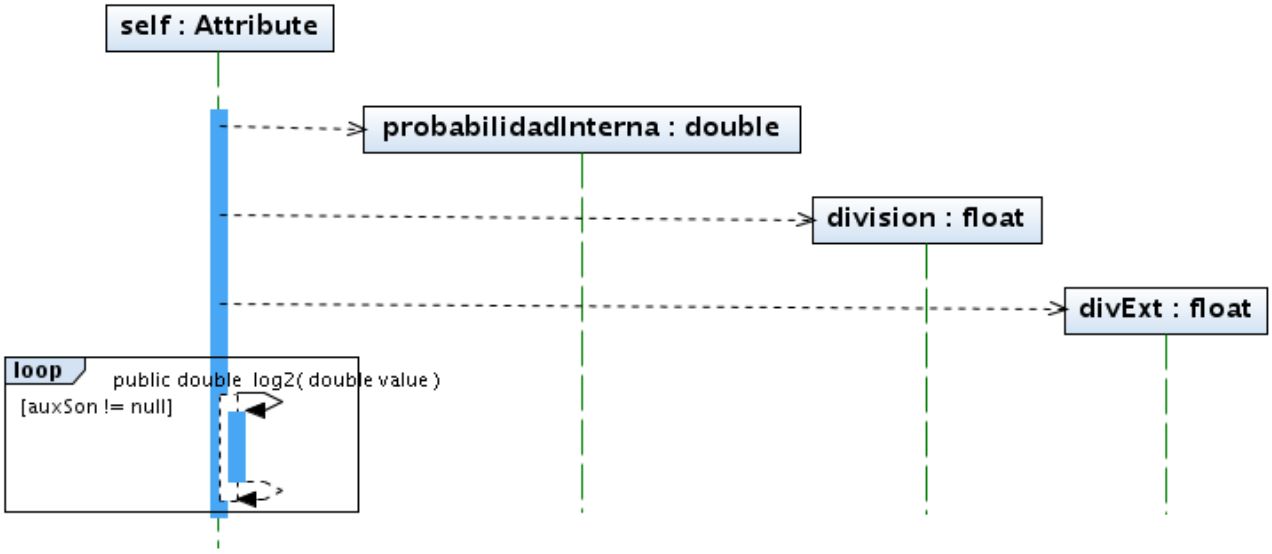

Figura 7.50: setEntropia

Clase Node

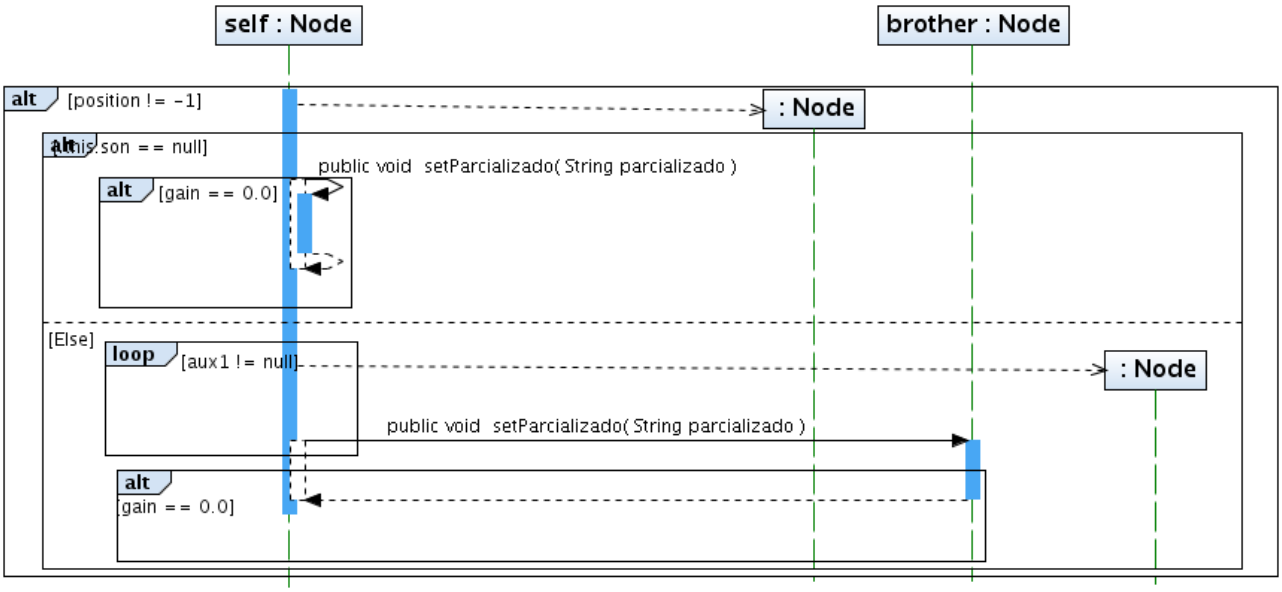

Figura 7.51: addSon

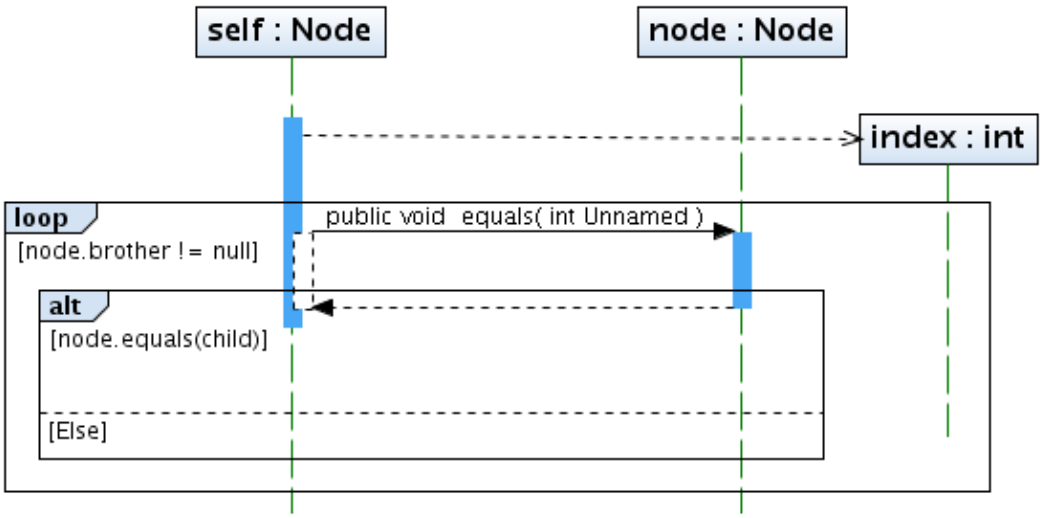

Figura 7.52: getIndexOfChild

7.2. Diseño

7.2.1. Diagramas de Colaboración

Clase Apriori

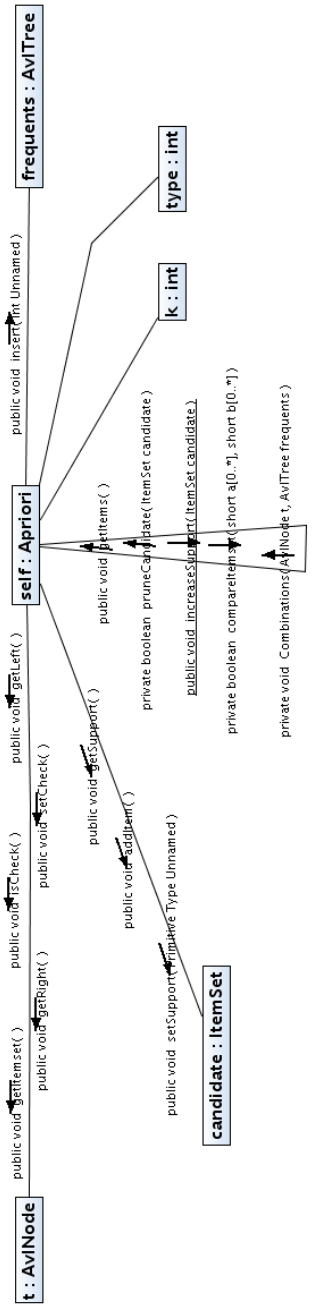

Figura 7.53: Combinations

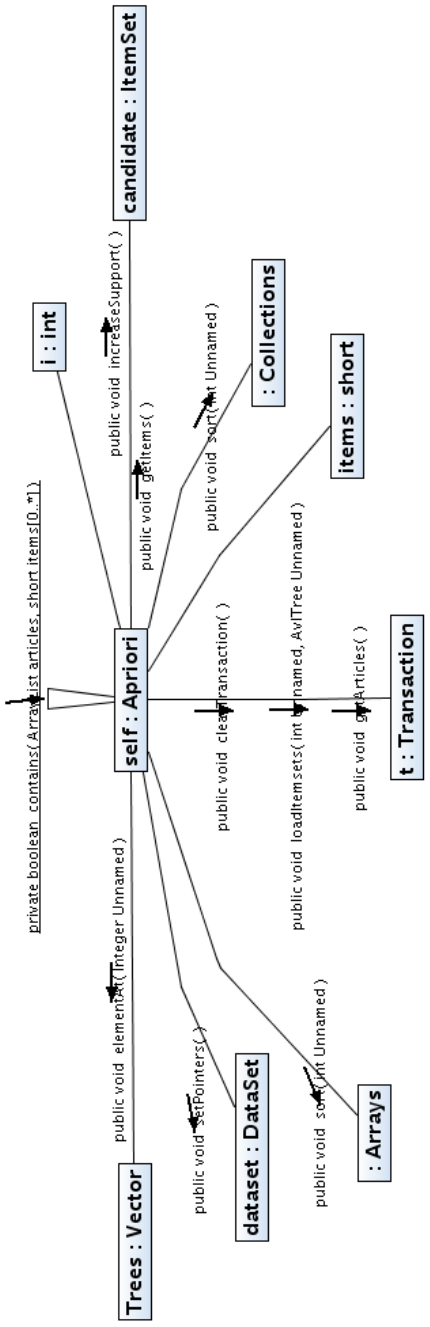

Figura 7.54: IncreaseSuport

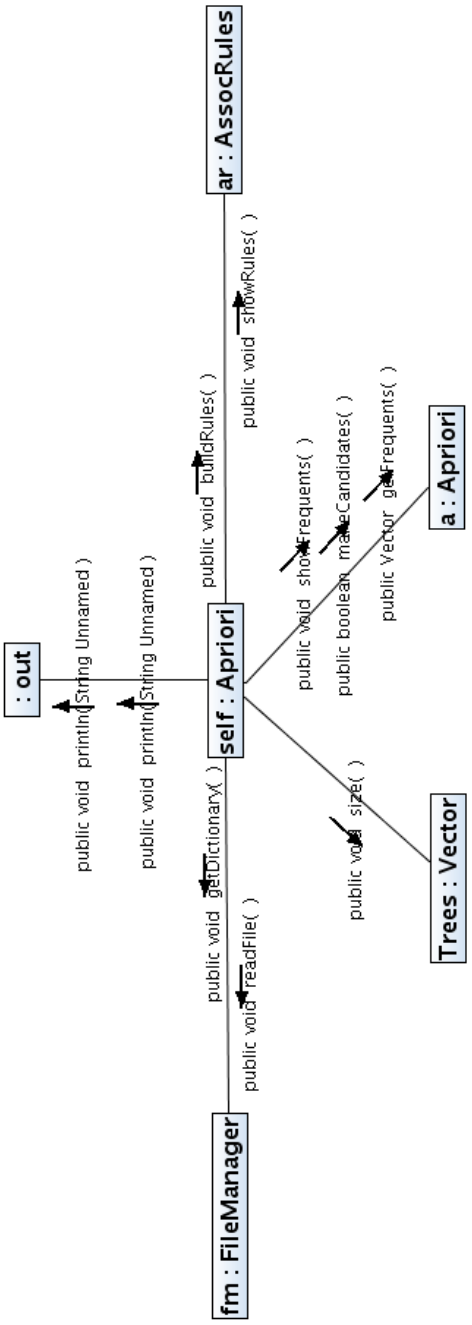

Figura 7.55: main

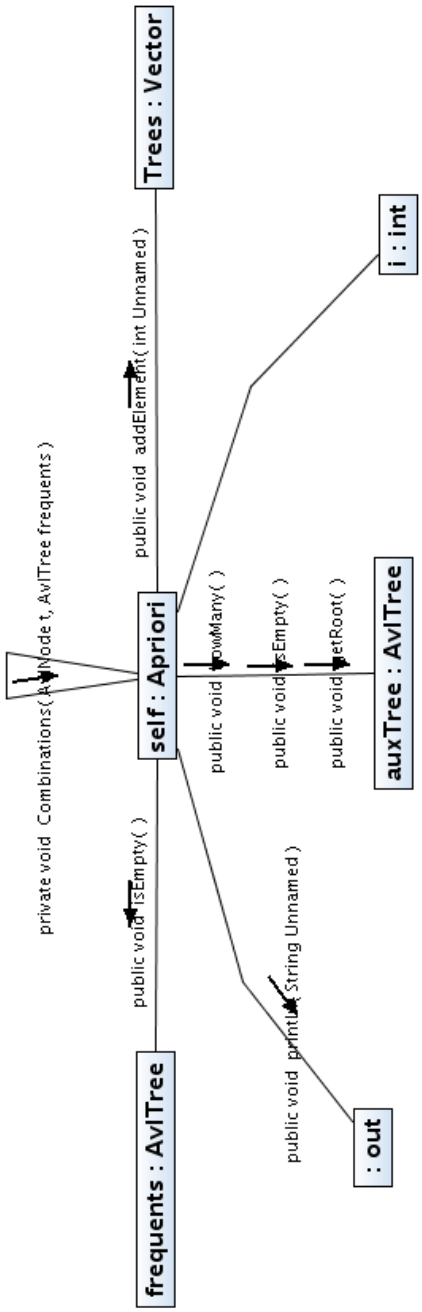

Figura 7.56: makeCandidates

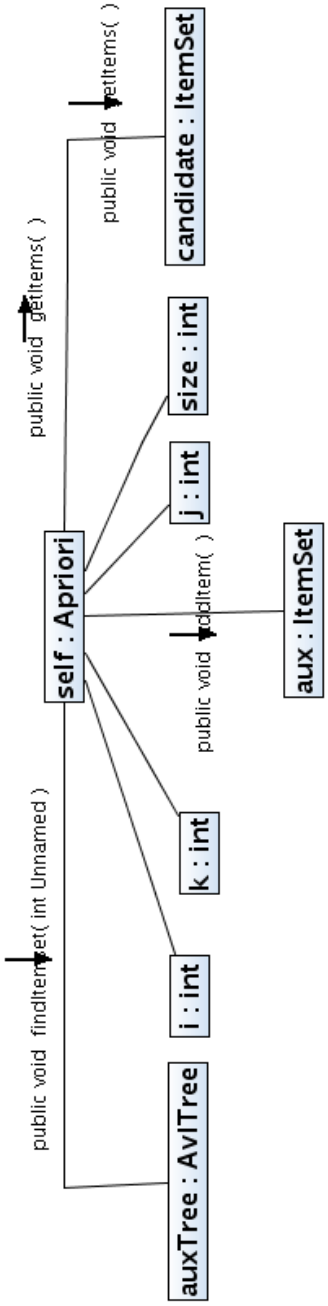

Figura 7.57: pruneCandidate

Clase EuipAsso

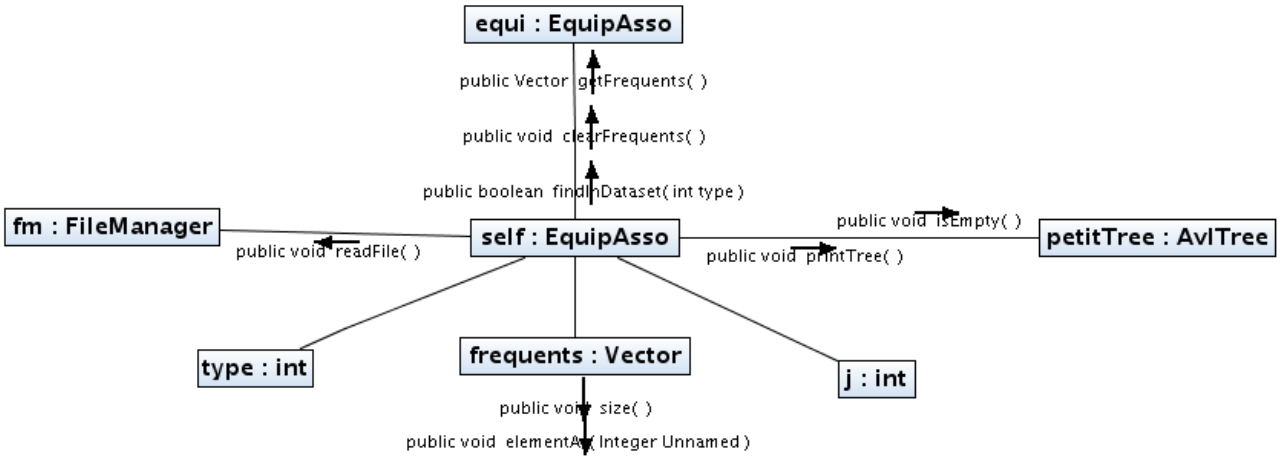

Figura 7.58: main

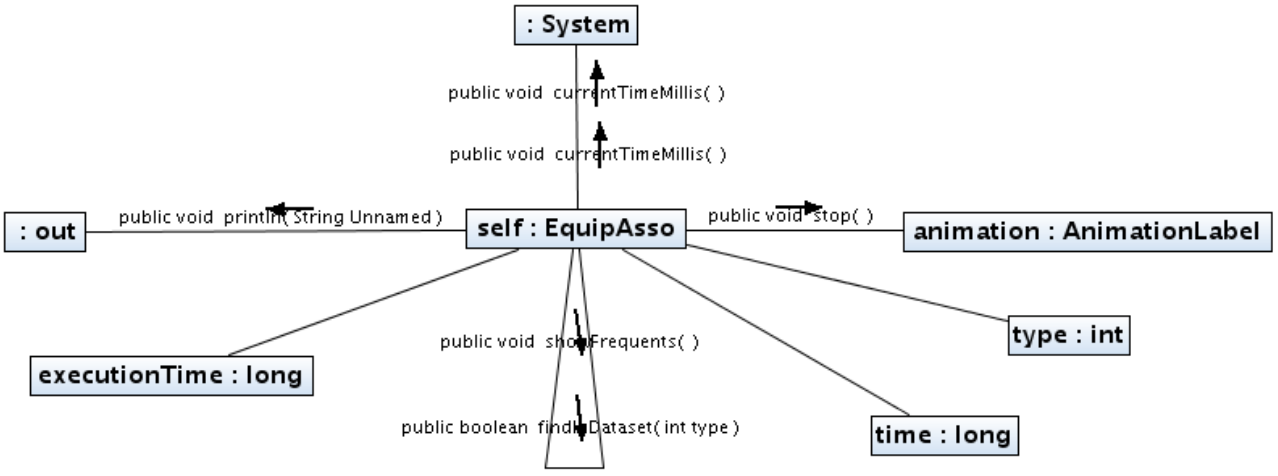

Figura 7.59: run

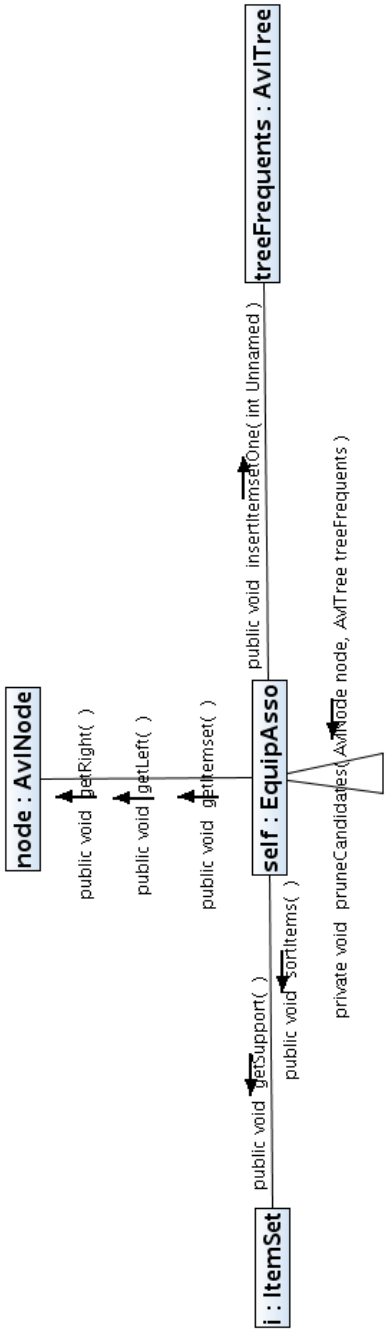

Figura 7.60: pruneCandidates

Clase Combinations

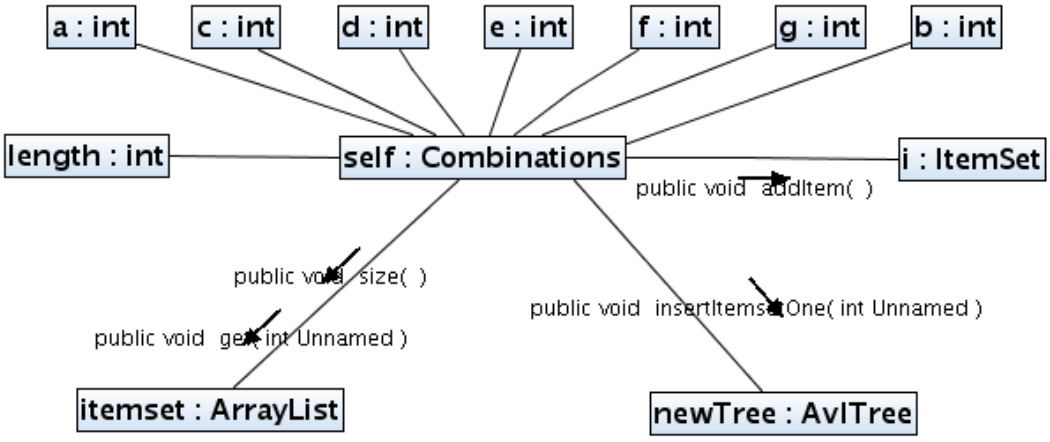

Figura 7.61: combine7

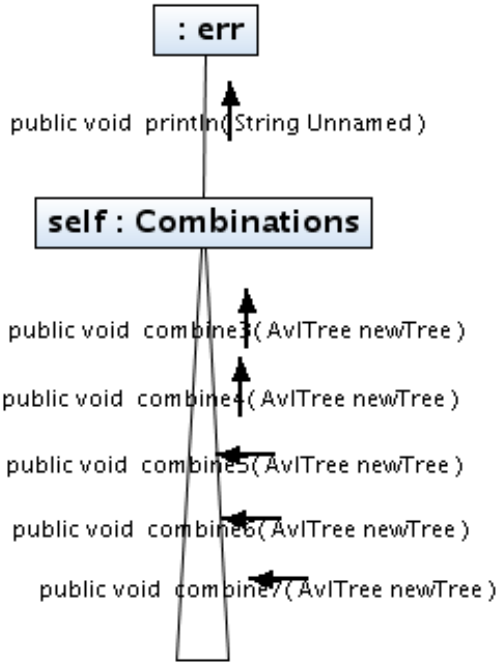

Figura 7.62: letsCombine.png

Clase BaseConditionals

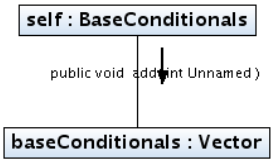

Figura 7.63: addBaseConditionals

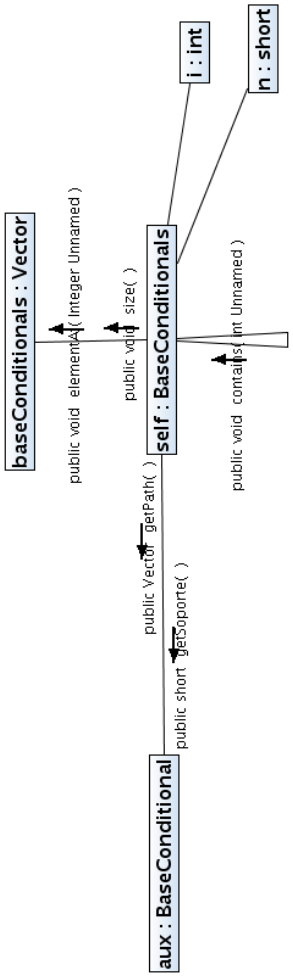

Figura 7.64: findItem

Clase Combinations

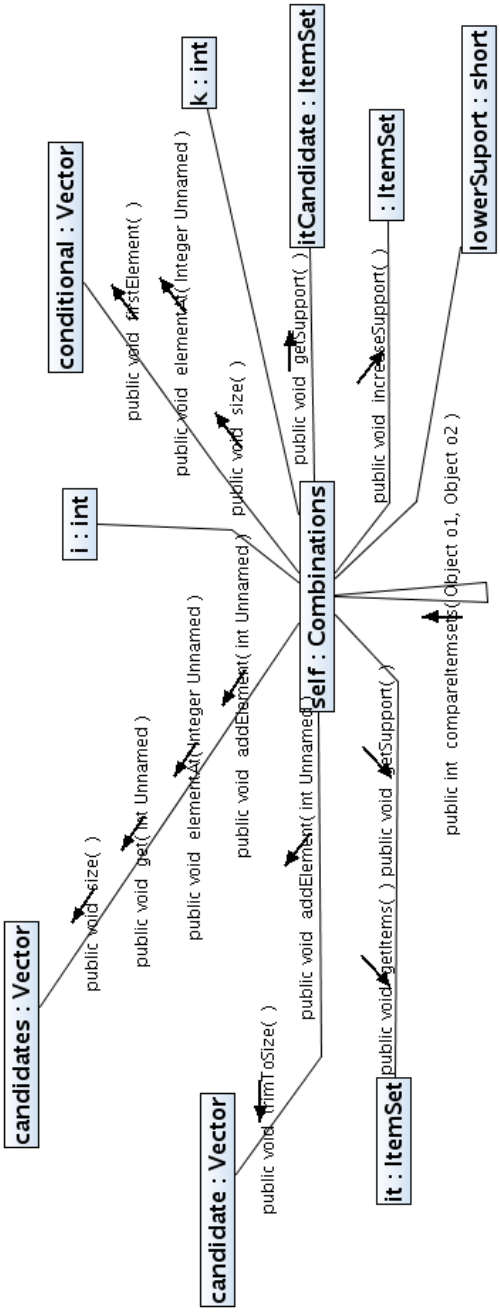

Figura 7.65: addCandidates

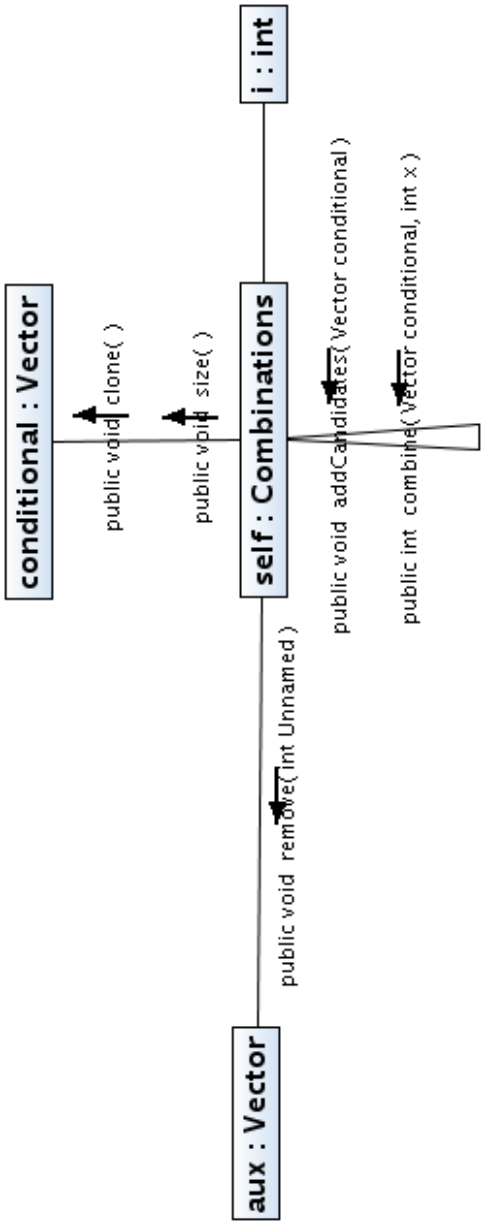

Figura 7.66: Combine

Clase FPGrowth

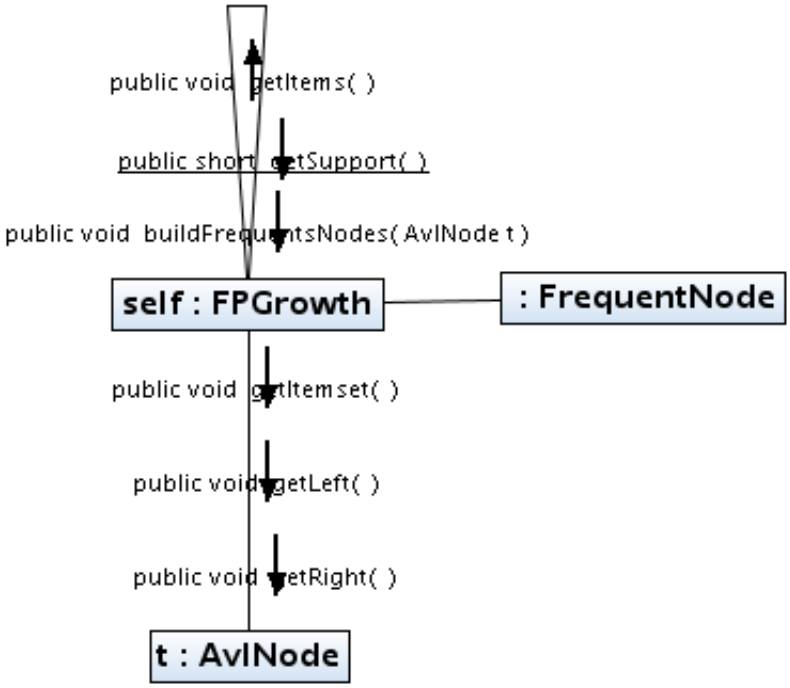

Figura 7.67: builtFrecuenceNode

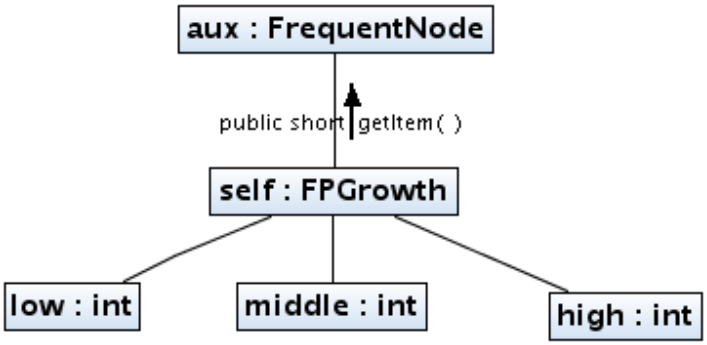

Figura 7.68: FrequentNode

Clase AvlTree

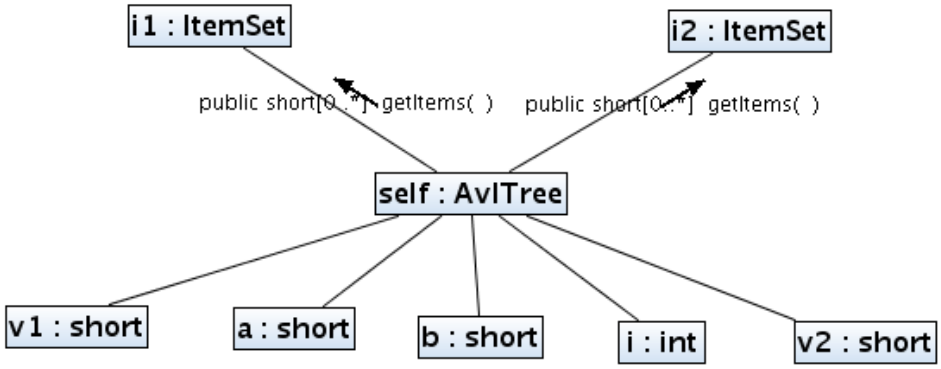

Figura 7.69: compareItems

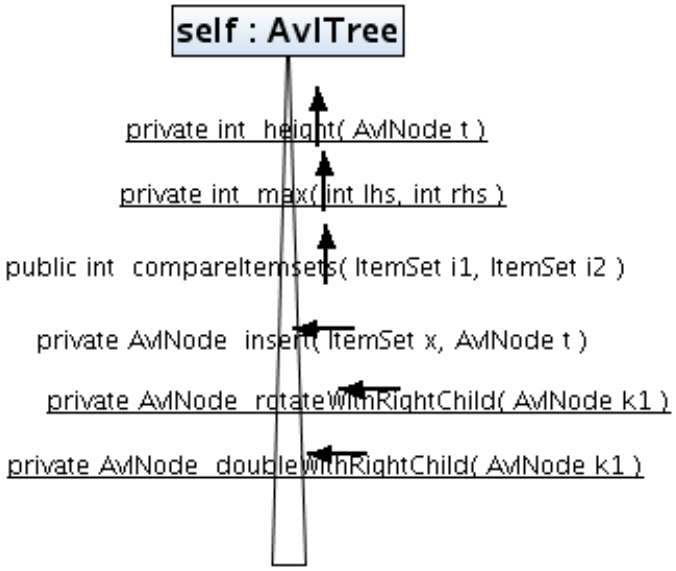

Figura 7.70: insert

7.2.2. Diagramas de Clase

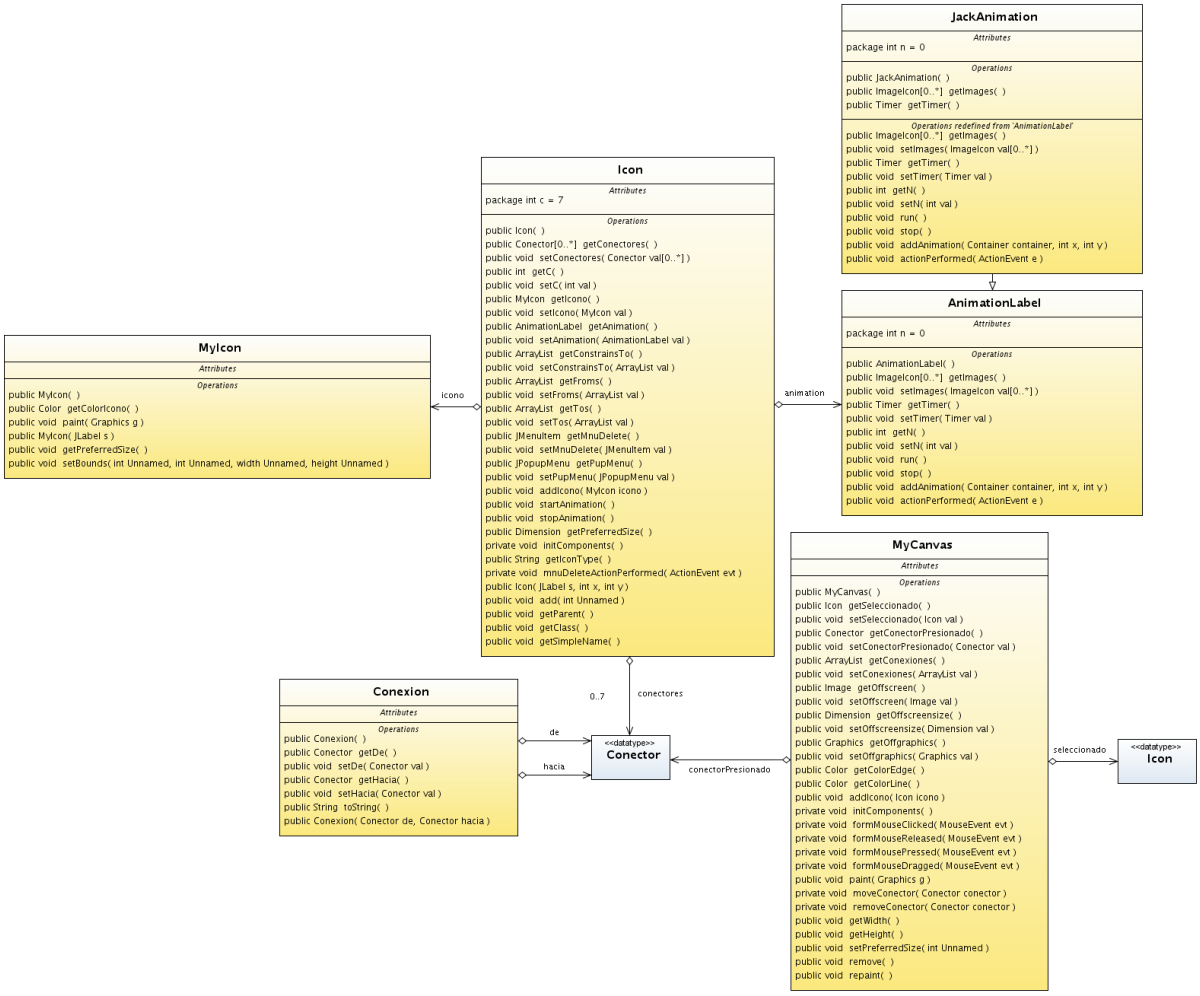

Figura 7.71: Paquete KnowledgeFlow

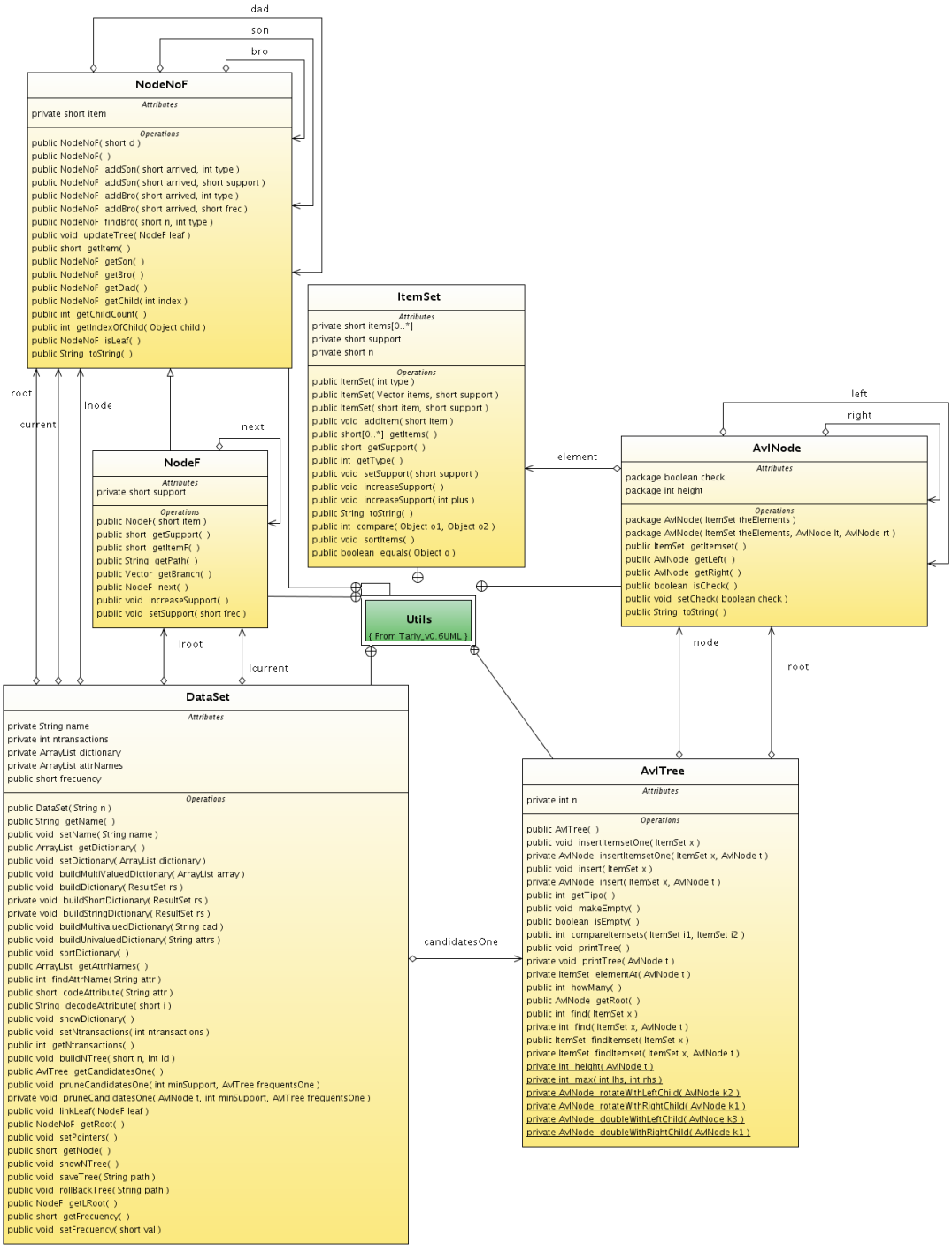

Figura 7.72: Paquete Utils

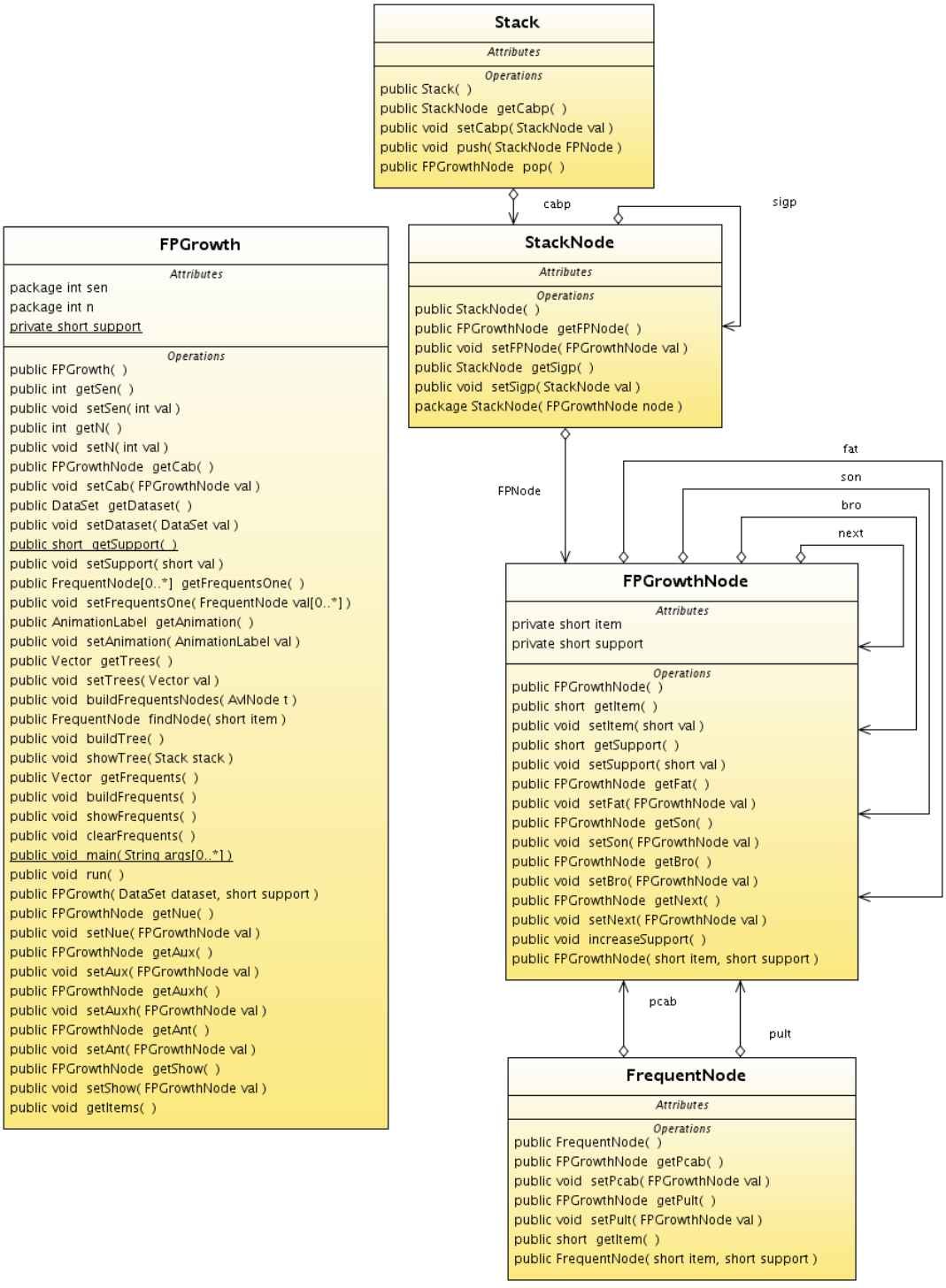

Figura 7.73: Paquete FPGrowth

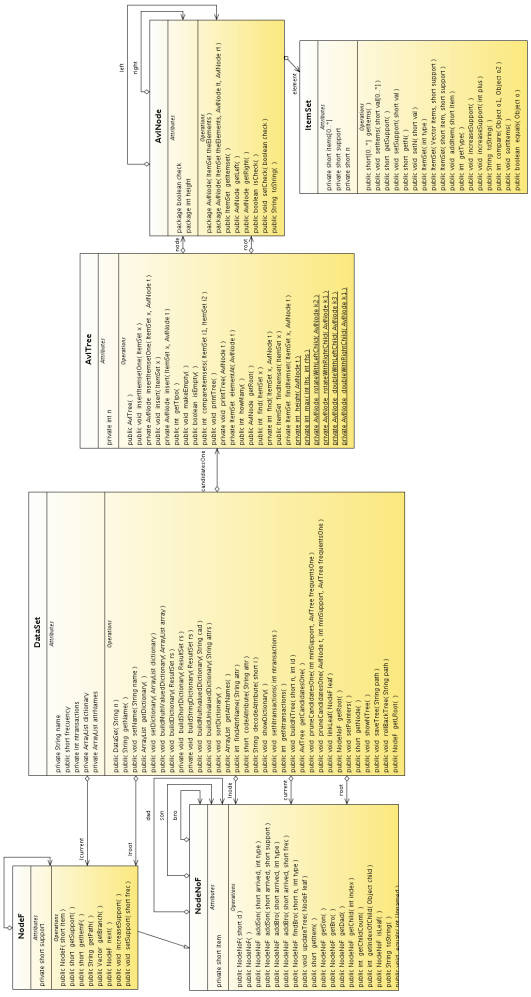

Figura 7.74: Paquete EquipAsso

Figura 7.75: Paquete Mate

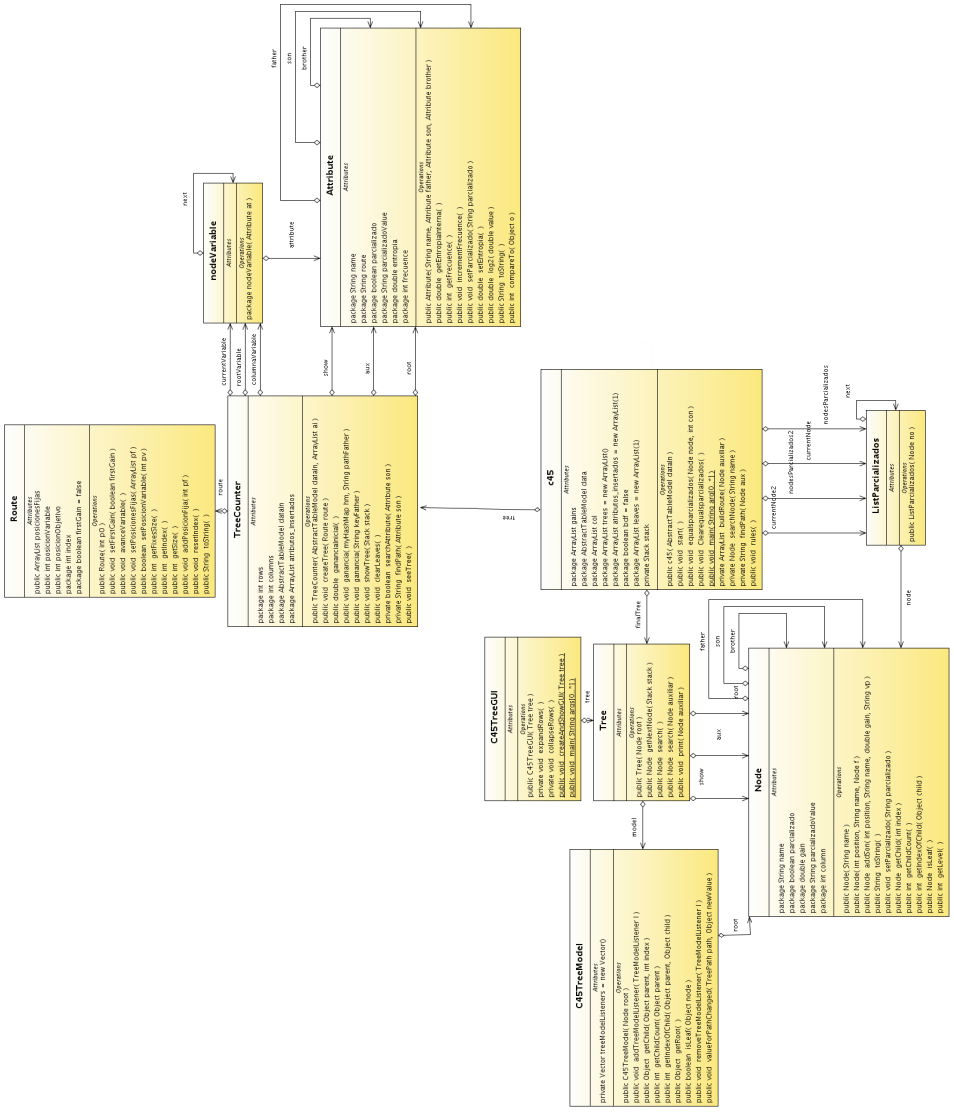

Figura 7.76: Paquete C45

7.2.3. Diagramas de Paquetes

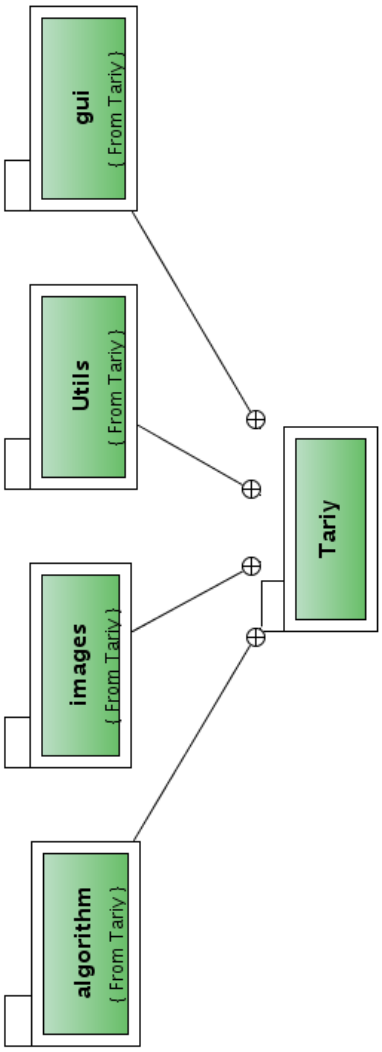

Figura 7.77: Paquete Principal

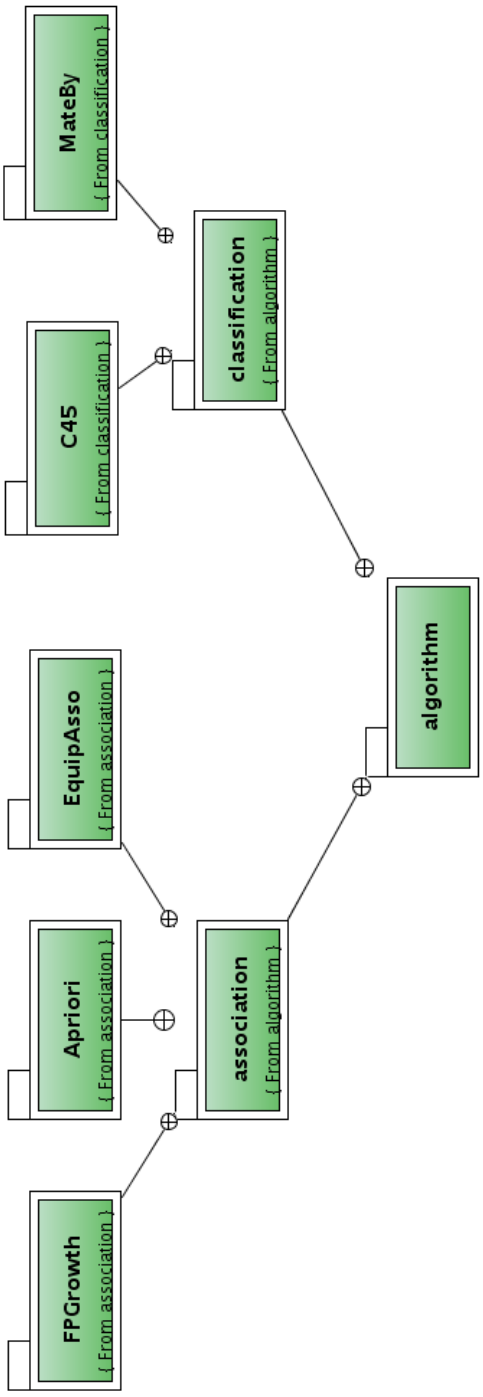

Figura 7.78: Paquete Algoritmos

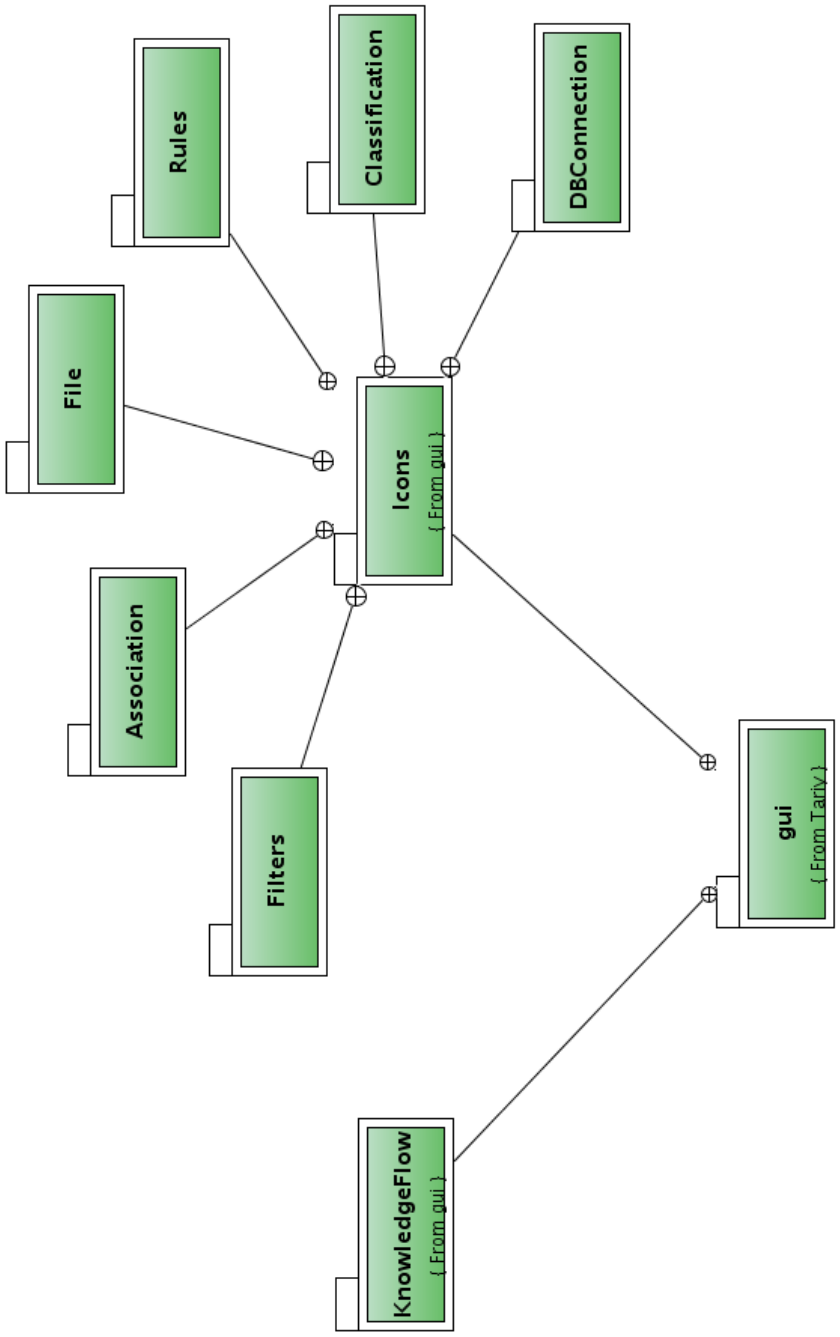

Figura 7.79: Paquete Interfaz Gráfica

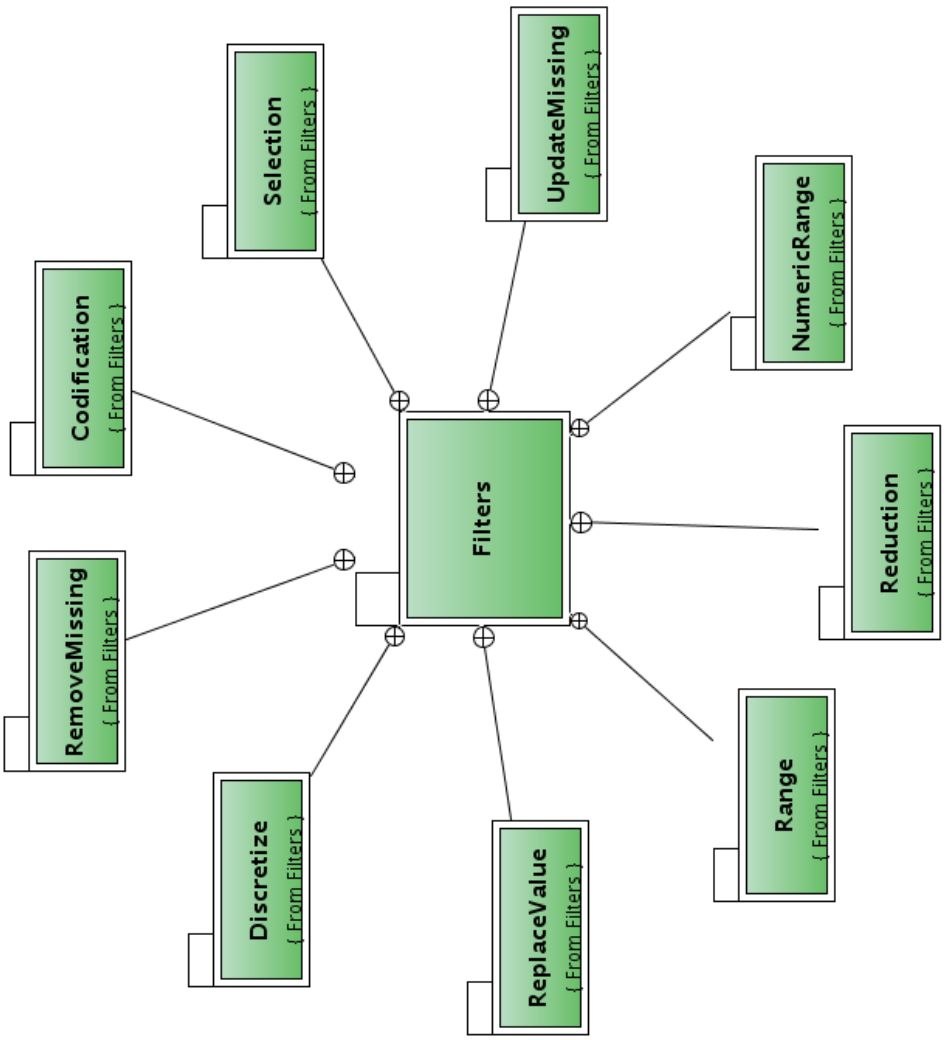

Figura 7.80: Paquete GUI Filtros

Capítulo 8

IMPLEMENTACIÓN

8.1. Arquitectura de TariyKDD

Para el desarrollo de TariyKDD se utilizaron computadores con procesador AMD 64 bits, disco duro Serial ATA, útil al tomar los datos desde un repositorio y al momento de realizar pruebas de rendimiento de los algoritmos, ya que su velocidad de transferencia es de 150 MB/sg; además la RAM que se utilizo fué superior a los 512 MB, ya que la Minería de Datos requiere grandes cantidades de memoria por el tamaño de los conjuntos de datos.

El sistema operativo sobre el cual se trabajo durante la implementación de TariyKDD es Fedora Core en su versiones 3 y 5. El lenguaje de programación en el que esta elaborado TariyKDD es Java 5.0, actualización 06.

Dentro del proceso de Descubrimiento de Conocimiento, TariyKDD comprende las etapas de Selección, Preprocesamiento, Minería de Datos y Visualización de Resultados. De esta forma la implementación de la herramienta se hizo a través de los siguientes módulos de software cuya estructura se muestra en la figura 8.1.

8.2. Descripción del desarrollo de TariyKDD

A continuación, en primera instancia se describe de manera general la estructura de TariyKDD para dar una idea global de como esta herramienta fué implementada. A continuación, se amplia y se detalla más la forma en que TariyKDD fué desarrollada.

- utils: Utilidades de TariyKDD. Dentro de este paquete encontramos clases como DataSet...

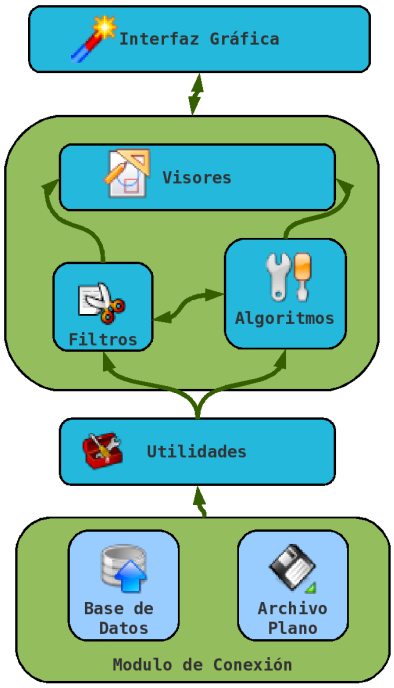

Figura 8.1: Árquitectura TariyKDD

- algorithm: Esta compuesta de los paquetes association y classification, los cuales implementan mediante sus respectivas clases los algoritmos de asociación (Apriori, FPgrowth y EquipAsso) y clasificación (C4.5 y MateBy).
- gui: Comprende los paquetes Icons y KnowledgeFlow, los cuales a través de sus clases implementan la interfaz gráfica de la herramienta.

8.2.1. Paquete Utils

Esta clase esta compuesta por las siguientes clases:

Clase AssocRules

Esta clase es la encargada de generar reglas a partir de los árboles de itemsets frecuentes que generan los algoritmos de asociación. Las reglas se generan teniendo en cuenta el parámetro *confianza* . Los atributos que maneja esta clase son los siguientes:

frequents de tipo Vector: vector de árboles AVL que contienen su ves itemsets frecuentes.

dictionary de tipo ArrayList: arreglo que contiene los nombres de los itemsets frecuentes.

confidence de tipo entero: variable en la que se especifica la confianza de evaluación de las reglas.

rules de tipo ArrayList: arreglo en el que se almacenan las reglas que cumplan con la confianza.

El curso normal de los eventos empieza con el llamado al método *buildRules* en el cual se inicia el recorrido del vector de de árboles AVL. La raiz de cada árbol es enviada como parámetro al método *walkTree* que es el encargado de recorrer el árbol y armar por cada rama recorrida un arreglo que es enviado al método *combinations* para que se generen todas las combinaciones posibles de esa rama y obtener las nuevas reglas. Cada regla es evaluada para verificar que cumpla el nivel de confianza. La fórmula para la evaluación de confianza es: $(soporte_Frecuentes/soporte_Antecedente) * 100$. Si una regla supera el nivel de confianza es enviada al método *decodeFrecuents* para su decodificación. Al final para efectos prácticos de representación existen varios métodos que permiten oredenar las reglas generadas de acuerdo a varios parámetros: soporte, confianza u arden alfabético.

Clase AvlNode

En esta clase se construye la estructura interna de datos de un nodo perteneciente a un árbol Avl. Los atributos que maneja esta clase son los siguientes:

element de tipo ItemSet: aqui se guardan los datos con los que se carga el nodo.

left, right de tipo Avlnode: punteros hacia los hijos izquierdo y derecho respectivamente.

height de tipo entero: altura del árbol Avl.

check de tipo booleano: variable de control del balance del árbol.

Clase AvlTree

Esta clase es la encargada tanto de crear o armar un arbol Avl como de proveer los métodos necesarios para su manejo. Un árbol Avl es un tipo de árbol binario que es balanceado de tal manera que la profundidad de dos hojas cualquiera en el árbol difiera como máximo en uno. Los atributos de esta cln los siguientes:

root de tipo Avlnode: raiz del árbol Avl

n de tipo entero: variable de control de la profundidad o número de niveles del árbol.

node de tipo AvlNode: nodo para recorrer el árbol.

stack de tipo Stack: pila utilizada para almacenar la ruta del recorrido hecho en una inserción, búsqueda o eliminación.

Para lograr la construcción y manejo de un árbol Avl son necesarios algunos métodos, los cuales serán mencionados más no explicados debido a su complejidad y a que estos métodos son bastante conocidos dentro del ámbito de las estructuras de datos. Los métodos implementados son: *insert, find, height, rotateWithLeftChild, rotateWithRightChild, doubleWithLeftChild, doubleWithRightChild*, métodos de inserción, búsqueda, consulta de profundidad del árbol, rotación por el hijo izquierdo, rotación por el hijo derecho, doble rotación por izquierda y doble rotación por derecha respectivamente.

Clase BaseDatos

Esta clase es utilizada siempre que se necesite efectuar conexiones a bases de datos por medio del objeto de conexión java para Postgres. Cabe aclarar que el sistema gestor de bases de datos (SGBD) inicial es Postgres pero el nombre del sistema gestor es totalmente parametrizable, permitiendo de esta manera la conexión a casi cualquier SGBD. Los atributos de esta clase son:

db de tipo Connection: objeto encargado de establecer la conexión.

stm de tipo Statement: variable encargada de ejecutar un query determinado.

nombre de tipo String: nombre de la base de datos.

Inicialmente se estable el nombre del contolador de la base de datos que se desea consultar. En este caso el controlador Postgres. Luego se llama al método *iniciarBD* que se encarga de establecer la conexión a la base de datos. Tiene como parámetros el nombre de la base de datos, el nombre del usuario y constraseña.

Existen otros métodos implementados para la consulta del nombre de las tablas, inserción y consulta de los datos de una tabla. El método *getTablas* retorna un vector con los nombres de las tablas que contenga la base de datos. El método *insertarBD* permite insertar un registro nuevo en una tabla. Tiene como parámetros el nombre de la tabla, el identificador de u item y el nombre del item a insertar. *getDatos* permite ejecutar una consulta general a una tabla. Solo requiere el nombre de la tabla a consultar.

Clase FileManager

Esta clase se encarga de todo lo que tiene que ver con el manejo de archivos de acceso a aleatorio y el flujo de información entre el archivo leído, el DataSet y el diccionario de datos. Los atributos de esta clase son:

out de tipo File: archivo al cual se va a leer o escribir.

outChannel de tipo RandomAccessFile: puntero para ubicarse en el archivo de acceso aleatorio. Proporciona el flujo hacia el archivo.

data de tipo Object: matriz con los datos de un archivo de acceso aleatorio .arff

attributes de tipo Object: arreglo con los nombres de los atributos de un archivo de acceso aleatorio .arff.

dictionary de tipo Array List: arreglo en el que se almacena el diccionario de datos de un archivo .arff

Dentro de la lista de utilidades que se pueden encontrar en esta clase se pueden listar las siguientes:

writeItem: método utilizado para escribir un entero corto (short) en un archivo. Código 8.2.1

writeString: método utilizado para escribir una cadena de caracters en un archivo. Código 8.2.2.

```
...
  public void writeItem(short s) {
  try{
    outChannel.seek( out.length() );
    outChannel.writeShort(s);
  } catch( IOException e ) {
  e.printStackTrace();
  }
  }
...
```

Código 8.2.1: Escribir un item a archivo

```
...
  public void writeString(String s){
  byte b[];
  try{
    b = s.getBytes();
    outChannel.seek( out.length() );
    outChannel.write(b);
  } catch( IOException e ) {
  e.printStackTrace();
  }
  }
...
```

Código 8.2.2: Escribir una cadena a archivo

getFileSize: retorna el tamaño de un archivo en bytes. Cuadro 8.2.3

```
...
  public long getFileSize() {
  return out.length();
  }
...
```

Código 8.2.3: Tamaño de un archivo

getFileName: retorna el nombre del archivo de acceso aleatorio. Cuadro 8.2.4

closeFile: cierra el flujo hacia el archivo de acceso aleatorio. Cuadro 8.2.5

```
...
  public String getFileName() {
  return out.getName();
  }
...
```

Código 8.2.4: Nombre de un archivo

```
...
  public void closeFile() {
  try{
    outChannel.close();
  } catch(IOException e) {
  e.printStackTrace();
  }
  }
...
```

Código 8.2.5: Cierra conexión a archivo

deleteFile: borra físicamente el archivo de acceso aleatorio. 8.2.6

```
...
  public void deleteFile() {
  out.deleteOnExit();
  }
...
```

Código 8.2.6: Borrar un archivo

setOutChannel: ubica el flujo en una posición determinada del archivo de acceso aleatorio. Código 8.2.7.

ReadTransaction: lee y muestra el contenido de un archivo de acceso aleatorio. Esta función es útil para el antiguo formato de archivos Tariy, lee un flujo de tipo *Short* y muestra cada transacción separada por el cero. Código 8.2.8

getAttributes: retorna el arreglo que tiene los nombres los atributos del archivo de acceso aleatorio .arff. Código 8.2.9

```
...
  public void setOutChannel(int pos) throws IOException {
  outChannel.seek(pos);
  }
...
```

Código 8.2.7: Manejo de posición en un archivo

```
...
  public void ReadTransaction(int readposition) {
  try{
    outChannel.seek(readposition);
    short s;
    while( true ) {
    s = outChannel.readShort();
    if(s != 0)
      System.out.print(s + " ");
    else
      System.out.println();
    }
  } catch(EOFException e1) {
    this.closeFile();
  } catch(IOException e2) {
    e2.printStackTrace();
  }
  }
...
```

Código 8.2.8: Lectura de un archivo

```
...
  public Object[] getAttributes() {
  return attributes;
  }
...
```

Código 8.2.9: Nombres de atributos de un archivo plano

getData: retorna la matriz que almacena los datos del archivo de acceso aleatorio .arff. Código 8.2.10

```
...
  public Object[][] getData() {
  return data;
  }
...
```

Código 8.2.10: Matriz de datos del archivo plano

getDictionary: retorna el diccionario de datos construido a partir del archivo de acceso aleatorio .arff.

buildMultivaluedDataset: este es uno de los métodos más importantes en la face de carga de los datos desde el archivo plano a memoria. El archivo de extención arff se recorre completamente pero solo se cargan los datos necesarios al DataSet. Por ejemplo, en un archivo arff se pueden encontrar algunas etiquetas propias como lo son *@data* y *@attribute*. Este tipo de etiquetas son ignoradas al igual que los espacios en blanco. Debido a que el objetivo es tratar de ahorrar espacio en memoria y a la naturaleza del DataSet los datos del archivo plano no pueden ser almacenados directamente como vienen. Los datos se codifican como enteros cortos *shorts* y sus nombres originales se almacenan en un diccionario para poder recuperarlos en el proceso inverso en el momento de generar las reglas.

buildUnivaluedDataSet: este es un método similar al anterior pero optimizado para manejar archivos univaluados, es decir, aquellos que solo tienen dos columnas una de las cuales es el identificador y la otra el nombre del artículo.

dataAndAttributes: a partir de un archivo de acceso aleatorio .arff, almacena los nombres de los atributos en un arreglo, los datos en una matriz y el diccionario de datos en un ArrayList.

builtDataMatrix: retorna una matriz que representa un archivo de acceso aleatorio .arff con los nombres de sus atributos y sus datos.

Clase ItemSet

Esta clase se encarga del manejo de los conjuntos de items. Un itemSet puede ser entendido como una trasacción. Los atributos de esta clase son:

items de tipo short: arreglo que almacena los items que forman el itemset.

support de tipo short: soporte asociado a este itemset.

n de tipo short: tamaño del itemset.

Los métodos implementados son:

getItems: retorna el item asociado.

getSupport: retorna el soporte de un itemset

gettype: retorna el tipo de un itemset, el tipo se deriva del tamaño del itemset.

setSupport: establece el soporte de un itemset.

increaseSupport: incrementa el soporte de un itemset, ya sea en uno o el número que se requiera.

Clase NodeNoF

Esta clase se encarga de crear los nodos intermedios de un árbol de datos. También se implementan los métodos para enlazar un nodo a otro. Entre ellos se encuentran: addSon, addBro y findBro que se encargan de adicionar un hijo, adicionar un hermano y buscar si un nodo tiene hermanos respectivamente.

Clase NodeF

Esta clase extiende a la clase NodeNoF. este tipo de nodos son las hojas de las ramas. Lo que las hace diferentes, son dos atributos adicionales, uno es el soporte, que se encarga de llevar a cabo el conteo de veces que una transacción repetida ha sido tenida en cuenta, y el otro es un puntero a otra hoja. En el momento de recorrer un árbol, el tipo de nodo nos permite saber que hemos llegado al final de una rama.

Clase Transaction

Esta clase es la encargada de manejar los itemsets o conjuntos de items por cada transacción. Cada uno de los itemsets son almacenados en vectores. Esta clase se

encarga de alimentar esos vectores, permite hacer consulta sobre ellos u organizarlos. Estos métodos se muestran a continuación:

getArticles: permite ver los articulos en una transacción.

getSize: devuelve el tamaño de una transacción.

getItemset: devuelve un item de la trnsacción.

srtBySupport: ordena las transacciones por soporte.

srtByItem: ordena las transacciones por el item.

loadItemsets: carga los artículos o items de una transacción.

clearTransaction: borra los items de una transacción.

8.2.2. Paquete algorithm

Este paquete esta compuesto a su vez por los paquetes association y classification, los cuales implementan los algoritmos de Asociación (Apriori, FPgrowth y EquipAsso) y Clasificaión (C4.5 y MateBy).

Paquete association

El paquete association esta conformado por los paquetes que implementan los algoritmos de asociación y que se explican a continuación:

Paquete Apriori La implementación del algoritmo apriori se realizó utilizando una clase denominada apriori.java que interactua directamente con otras clases definidas en el paquete Utils como DataSet, Transaction, AvlTree e Itemset. Al igual que las otras clases que implementan algoritmos de asociación, en el constructor de la clase Apriori se usan 2 parametros: una instancia de la clase DataSet, donde vienen comprimidos el conjunto de datos desde el módulo de conexión o el módulo de filtros, y un entero corto, que es suministrado por el usuario, que será usado como el soporte del sistema durante la ejecución de este algoritmo.

De igual manera, en la construcción de la clase se instancia e inicializa un objeto de la clase AvlTree que almacenará los itemsets frecuentes tipo 1, el cual es alimen-

tado haciendo uso del método *pruneCandidatesOne* de la clase DataSet, el cual recibe el soporte del sistema como parametro. En este punto también se instancia e inizializa un arreglo que guardará los diferentes árboles balanceados (AvlTree), que contendrán los Itemsets Frecuentes de cada tipo y que serán calculados por la herramienta. Al disponer ya de los itemsets frecuentes tipo 1, estos son almacenados en la posicion 0 de este arreglo. Los detalles del constructor de esta clase se aclaran en el siguiente listado:

```
public Apriori(DataSet dataset, short support) {
  this.support = support;
  this.dataset = dataset;
  AvlTree frequentsOne = new AvlTree();
  frequentsOne = dataset.pruneCandidatesOne(support);
  Trees.addElement(frequentsOne);
  auxTree = frequentsOne;
}
```

Código 8.2.11: Constructor de la clase *Apriori*

Para disparar la ejecución del algoritmo usamos el método *run* el cual ejecuta un ciclo dentro del cual el método *makeCandidates* se encarga de generar todos los itemset candidatos para cada iteración. Para esto el método *makeCandidates* llama al objeto *auxTree*, instancia de la clase AvlTree que conserva el ultimo árbol de itemset frecuentes y pregunta cuantos itemsets tiene en ese instante utilizando el método *howMany* de esta clase.

Para cada elemento encontrado dentro del árbol de Itemsets Frecuentes se recorre un ciclo con los demás miembros del árbol armando combinaciones que generan un nuevo itemset candidato haciendo uso del método *combinations* que recibe como parametros el árbol *auxTree* de Itemset Frecuentes y un árbol AvlTree donde se guardaran los Itemsets Frecuentes tipo k + 1 llamando *frequents*. Se aprovecha la ventaja de que los itemsets están ordenados en el árbol y se verifica que el itemset evaluado coincida en sus $n-1$ primeros items, siendo n el tamaño de ese itemset, con el proximo itemset del árbol. De ser así, se instancia un nuevo itemset con los $n-1$ items coincidentes y los 2 últimos items de cada itemset involucrado. De esta manera se generan itemsets candidatos de tamaño $n+1$.

Cada itemset candidato es pasado como parámetro al método *increaseSupport* donde es contado el número de ocurrencias de ese itemset candidato dentro del

conjunto de datos. En este punto se hace una consideración importante y si los itemsets candidatos que se están construyendo son de tipo 3 o superior se evalua el paso de poda, esto es, se descompone el itemset candidato construido en sus $n-2$ posibles combinaciones del tipo anterior, siendo n el tamaño del itemset candidato que se esta generando. No se evaluan las dos últimas combinaciones pues fueran estas las que generaron el candidato que se esta analizando y se tiene certeza de que existen en el árbol de Itemsets Frecuentes. Cada una de las combinaciones generadas son buscadas en *auxTree*, hay que recordar que este es el árbol que contiene todos los Itemsets Frecuentes del orden anterior al que se esta generando. Si una de las combinaciones del itemset candidato actual no es encontrada en el árbol *auxTree*, este itemset ya no es considerado y se procede a evaluar el próximo. El método *pruneCandidate* se encarga de realizar este análisis y se presenta en el cuadro 8.2.12.

```
private boolean pruneCandidate(ItemSet candidate) {
  int size = candidate.size;
  ItemSet auxiliar;
  for(int i = 0; i < size - 2; i++) {
  aux = new ItemSet(size - 1);
  int k = 0;
  for(int j = 0; j < size; j++) {
    if(j != i) {
    auxiliar.addItem(candidate.getItems()[j]);
    }
  }
  if(auxTree.findItemset(auxiliar) == null) {
    return false;
  }
  }
  return true;
}
```

Código 8.2.12: Función *pruneCandidates*

Si el itemset candidato actual es de tipo 2 o ha superado el paso de poda se procederá a hacer su conteo. Para ello, se hace una nueva instancia de la clase *Transaction* donde se carga una a una las transacciones del conjunto de datos y se compara con el contenido del itemset. Si coinciden los elementos del registro y el itemset, este último incrementa su soporte interno en uno.

Al final de este proceso se cuenta con un itemset candidato con su soporte establecido, este se compara con el soporte del sistema y de superarlo es incluido en el árbol *frequents*. Cuando ya se han evaluado todos los candidatos para un árbol *auxTree*, se pregunta si se han generado nuevos Itemsets Frecuentes en *frequents*. Si el método *howMany* de *frequents* arroja existencias se almacena en el arreglo de árboles frecuentes y se asigna a *auxTree* el árbol *frequents* para iniciar de nuevo el proceso de generación de candidatos. Esto se hará hasta que el árbol *frequents* resulte vacío (no hayan nuevos Itemsets Frecuentes). En el arreglo de árboles frecuentes quedan almacenados todos los Itemsets Frecuentes organizados por tipo que serán pasados al Módulo de visores para ser visualizados como reglas.

Paquete FPgrowth Las clases que implementan el algoritmo FPgrowth se encuentran agrupadas en el paquete con el mismo nombre y las más importantes para su funcionamiento son FPGrowth, FPGrowthNode, FrequentNode, BaseConditional, BaseConditionals, Conditionals y AvlTree.

Tal y como se puede ver en la sección 6.4.2, el algoritmo FP-growth se basa en la generación de itemsets candidatos a partir de un árbol N-Ario. Para comprender mejor la implementación de FP-growth, se deben revisar primero las clases que forman la estructura del árbol N-Ario, las cuales han sido llamadas FrequentNode y FPGrowthNode.

Clase FrequentNode Para la posterior creación de itemsets frecuentes se almacenan en un array de objetos de tipo FrequentNode los itemsets frecuentes-1 o de tamaño 1. Cada uno de los items frecuentes-1 de esta lista se encuentra enlazado a un nodo del árbol N-ario, de ahora en adelante FPtree, que tenga el mismo nombre que el itemset frecuente-1.

Clase FPGrowthNode Esta clase proporciona la estructura del FPtree, en donde cada nodo tiene un valor de item, un soporte y los punteros respectivos para realizar los enlaces entre nodos (Punteros al nodo hijo, al nodo padre, al nodo hermano y al siguiente nodo con el mismo nombre). Su estructura se puede ver en el cuadro 8.2.13.

Clase FPGrowth FPGrowth es la clase principal del paquete, tiene los métodos más importantes del algoritmo y a través de los cuales se obtienen los itemsets

```
Atributos:
  short item //Dato que se va a a\~nadir al FPtree.
  short support //Soporte del item a\~nadido.
  FPGrowthNode fat //Puntero al nodo padre.
  FPGrowthNode son //Puntero al nodo hijo.
  FPGrowthNode bro //Puntero al nodo hermano.
  FPGrowthNode next //Puntero al siguiente nodo con el mismo valor.
```

Código 8.2.13: Estructura *FPGrowthNode*

frecuentes. Su estructura se puede ver en el cuadro 8.2.14.

```
Atributos:
  DataSet dataset //Estructura que comprime el conjunto de datos.
  short support //Soporte con el que se van a evaluar los datos.
  FrequentNode[] frequentsOne //Array que almacena a los itemsets
       //frecuentes-1.
  FPGrowthNode cab //Raiz del FPtree.
  Vector Trees //Almacena los itemsets frecuentes de todos los
        //tamanos.
```

Código 8.2.14: Estructura *FPGrowth*

Los parametros que el algoritmo FP-growth recibe en su constructor son, DataSet y el soporte suministrado por el usuario. El primer paso de la implementación de FPGrowth es crear la lista con los itemsets frecuentes-1, la cual se almacena en el array frequentsOne. El siguiente paso es construir el FPtree, para esto se lee cada una de las transacciones de DataSet y para aquellos items cuyo soporte sea mayor o igual al mínimo se los almacena en la clase del paquete Utils *Transaction*. A partir de estos items filtrados se construye el FPtree, cuyo seudocódigo se puede observar en el cuadro 8.2.15.

Para una mejor compresión sobre la construcción del FPtree se puede observar el conjunto de datos del cuadro 8.1 y su representación gráfica en la figura 8.2 y para profundizar más en la implementación se puede observar en el cuadro 8.2.16 el código fuente de la función que construye el FPtree (*buildTree*):

```
boolean comienzo = true
puntero_referencia = null
do while (!fin de DataSet)
  transaccion_filtrada = transaccion(i)
  si (comienzo)
  Crear raiz de FPtree
  comienzo = false
  end si
  while (!fin de transaccion_filtrada)
  si (puntero_referencia == null)
    anadir nuevo_nodo como hijo
  end si
  si_no
  si (puntero_referencia = nuevo_nodo)
    incrementar_soporte ultimo_nodo
  end si
  si_no
    while(puntero_referencia tenga hermanos)
    si (nodo_hermano = nuevo_nodo)
      incrementar_soporte nodo_hermano
      terminar while
    end si
    end while
    si (puntero_referencia no tiene mas hermanos)
    anadir nuevo_nodo como hermano
    end si
  end while
end do
```

Código 8.2.15: Seudocodigo construcción *FPtree*

Como se puede observar en la figura 8.2, cada uno de los items frecuentes-1 se encuentra enlazado a un nodo del FPtree, por ejemplo el item frecuente-1, 5 se encuentra enlazado al nodo de FPtree 5, cuyo soporte es 1 y a la vez cada nodo de FPtree se enlaza al siguiente nodo con el mismo nombre, en el caso del nodo 5, este se enlaza a otro nodo con valor 5 y cuyo soporte es 1.

El siguiente paso de la implementación cosiste en recorrer la estructura de los items frecuentes-1 y por cada uno de estos construir sus Patrones Condicionales Base,

```
short frequent;
BaseConditional baseconditional;
BaseConditionals bcs;
Vector path = new Vector(1,1);
aux = cab.son;
FPGrowthNode pcb, pcab;
Arrays.sort(frequentsOne, new compareFrequentNode());
for(int i = frequentsOne.length - 1; i >= 0; i--){
  FrequentNode aux = (FrequentNode) frequentsOne[i];
  pcb = aux.pcab;
  pcab = aux.pcab;
  frequent = pcb.getItem();
  bcs = new BaseConditionals(frequent, support);
  while(pcab != null){
  path.clear();
  pcb = pcb.fat;
  while(pcb != null){
    path.add((short) pcb.getItem());
    pcb = pcb.fat;
  }
  if(path.size() != 0){
    baseconditional = new BaseConditional(path,
        pcab.getSupport());
    bcs.addBaseConditionals(baseconditional);
  }
  pcab = pcab.next;
  pcb = pcab;
  }
  bcs.sortByElement();
  Trees = bcs.buildConditionals(Trees);
}
```

Código 8.2.16: Función *buildTree*

los cuales se obtienen recorriendo FPtree desde cada nodo enlazado por los items frecuentes-1 a través de su puntero al nodo padre hasta la raiz. Por ejemplo como se puede observar en la figura 8.2 para el item frecuente-1, 5 sus Patrones Condicionales Base son dos: (1,2:1) y (3,1,2:1), donde el número después de ":" es el soporte del Patrón, el cual corresponde al mismo soporte que tenga el item frecuente-1 en

Transacción	Lista de items
T01	{(1,2,5)}
T02	{(2,4)}
T03	{(2,3)}
T04	{(1,2,4)}
T05	{(1,3)}
T06	{(2,3)}
T07	{(1,3)}
T08	{(1,2,3,5)}
T09	{(1,2,3)}

Cuadro 8.1: Conjunto de datos transaccional

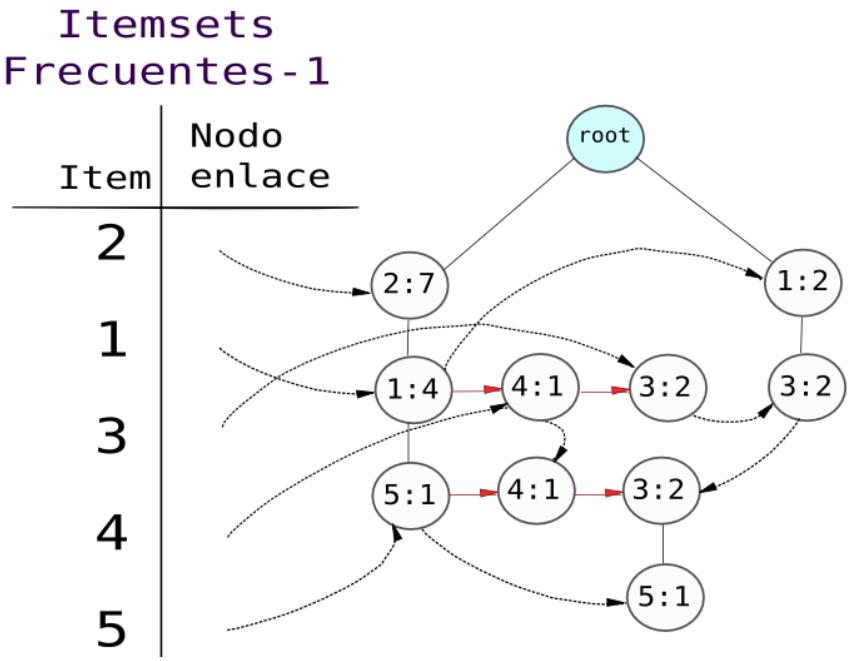

Figura 8.2: Árbol FPtree

cuestion. Cada uno de los Patrones Condicionales Base se almacenan en la clase *BaseConditional* y todo el conjunto de Patrones Condicionales Base de un item frecuente-1 se alamcenan en la clase *BaseConditionals*. En el cuadro 8.2 se pueden observar los Patrones Condicionales Base de cada uno de los items frecuentes-1.

Cuando se han construido todos los Patrones Condicionales Base de un item frecuente-1, el siguiente paso de la implementación es determinar cuales de sus items tienen soporte mayor o igual al mínimo. Para esto se debe sumar cada uno de los soportes que un item tenga en cada Patrón Condicional Base, los items que cumplan con el soporte van a conformar los Patrones Condicionales, los cuales se almacenan en la clase *Conditionals*, por ejemplo para un soporte mínimo igual a 2 los Patrones Condicionales del item frecuente-1 son (1:2) y (2:2), se puede observar que el item 3 no cumple con el soporte, por tanto no es Patrón Condicional. Los Patrones Condicionales de los items frecuentes-1 se pueden observar en el cuadro 8.2.

Item	**Patrones Condicionales Base**	**Patrones Condicionales**	**Patrones Frecuentes**
5	{(1,2:1),(3,1,2:1)}	(2:2, 1:2)	(2,5:2),(1,5:2)(2,1,5:2)
4	{(2:1),(1,2:1)}	(2:2)	(2,4:2)
3	{(2:2),(1:2),(1,2:2)}	(2:4, 1:2),(1:2)	(2,3:4),(1,3:2),(2,1,3:2)
1	{(2:4)}	(2:4)	(2,1:4)

Cuadro 8.2: Patrones Condicionales e Itemsets Frecuentes

El último paso de la implementación es determinar el conjunto de Itemsets Frecuentes. Para lo cual se hace uso de la clase *Combinations*, la cual toma los Patrones Condicionales y los combina con los items frecuentes-1. Por ejemplo, tenemos que los Patrones Condicionales del item frecuente-1, 5 son (2:2, 1:2), entonces, 5 se combina con sus Patrones Condicionales y se obtienen los Itemsets Frecuentes (2,5), (1,5) y (2,1,5). Para los demás items, podemos ver sus Itemsets Frecuentes en el cuadro 8.2.

Así como para Apriori y EquipAsso los Itemsets Frecuentes se almacenan en un Vector de arboles AVL balanceados, en donde en cada posición del Vector, se encuentran almacenados un tipo de Itemsets Frecuentes. Es decir en la posición 0 del Vector se encuentran los Itemsets Frecuentes tipo 1, en la posición 1 los Itemsets Frecuentes tipo 2 y así mismo el resto de Itemsets Frecuentes.

Algoritmo EquipAsso El paquete EquipAsso dentro del módulo de algoritmos contiene dos clases encargadas de la implementación y aplicación del algoritmo EquipAsso orientadas a dar soporte al descubrimiento de reglas de asociación dentro del proceso KDD. La primera *EquipAsso*, encargada de la carga de datos y un

primer filtrado de ellos, donde se carga solo aquellos items que pasan el umbral o soporte dado (proceso EquiKeep dentro del algoritmo EquipAsso) y la otra clase *Combinations* que se encarga de generar todas las combinaciones de un determinado tamaño para cada uno de los registros cargados del conjunto de datos y sus respectivo conteo (proceso Associator dentro del algoritmo EquipAsso).

La clase principal es *EquipAsso*, al hacer una instancia de esta clase se debe pasar como parámetros una instancia de la clase *DataSet*, llamada *dataset* y que contiene una versión comprimida del conjunto de datos, y un entero corto, que se llamado *support* y que cumple la tarea de soporte del sistema. Estas instancias son asignadas a atributos internos de esta clase.

La clase EquipAsso cuenta con un atributo de tipo arreglo llamado *Trees* donde se almacenan los diferentes árboles de Itemsets Frecuentes organizados por tamaño. El primer conjunto de itemsets son los de tamaño 1 y podemos obtenerlo al llamar al método *pruneCandidatesOne* de la clase *DataSet* que fué pasada como parámetro al constructor de la clase. Este método devuelve un árbol AvlTree que es insertado en la posición 0 del arreglo *Trees* y que contiene el conjunto de Itemsets Frecuentes tamaño 1. Los demás conjuntos de Itemsets Frecuentes (tamaño 2, tamaño 3, ...) serán almacenados en las siguientes posiciones de ese arreglo organizados en árboles AvlTree. En el siguiente listado de código podemos observar el constructor de la clase EquipAsso.

```
public EquipAsso(DataSet dataset, short support) {
  this.dataset = dataset;
  this.support = support;
  AvlTree frequentsOne = new AvlTree();
  frequentsOne = dataset.pruneCandidatesOne(support);
  Trees.addElement(frequentsOne);
}
```

Código 8.2.17: Constructor de la clase *EquipAsso*

Una vez instanciado un objeto *EquipAsso* se inicia el proceso del algoritmo llamando al método *run* de esta clase. Dentro de este método se inicia un ciclo controlado por el método *findInDataset* donde se recorre el conjunto de datos. Este método declara un objeto del tipo *Transaction* el cual, a través de su método *loadTransaction*, carga los registros del conjunto de datos. En este punto se realiza un primer filtrado cargando únicamente aquellos items que sobrepasen el soporte del sistema

y estén por ende contenidos en el conjunto de itemsets frecuentes tamaño 1 y que ha sido almacenado en la primera posición del arreglo *Trees*. Lo anterior se realiza ejecutando la siguiente línea de código:

```
transaction.loadTransaction(dataset, (AvlTree) Trees.elementAt(0));
```

Código 8.2.18: Llamado de *loadTransaction*

Es por eso que son enviados como parámetros instancias del dataset actual y del AvlTree que contiene los Itemset Frecuentes tamaño 1 para solo cargar del dataset los items que estén en este conjunto. Posterior a este filtrado se pasa esta transacción como parámetro a una instancia de la clase *Combinations* para encontrar sus posibles combinaciones. Puede darse el caso de que ninguno de los items provenientes del dataset supere el soporte, ni sea encontrado entre los Itemsets Frecuentes uno, por lo que se cargaría una transacción vacía, esta es descartada y se continua con la siguiente transacción.

La clase *Combinations* recibe como parámetro un arreglo que contiene los items validados de cada transacción provenientes del conjunto de datos. Este arreglo es cargado en el constructor de esta clase y posteriormente, con el llamado del método *letsCombine* de la clase *Combinations*, se generá un determinado grupo de combinaciones dependiendo del tamaño que se quiera generar, combinaciones tamaño 2, tamaño 3, etc según sea el caso, con cada una de estas combinaciones crea objetos de la clase *ItemSet*. El método *letsCombine* recibe como parámetro un árbol AvlTree llamado *treeCombinations* en el cual se van insertando los objetos *ItemSet* que han sido generados.

Se hace uso de un árbol balanceado AvlTree, para almacenar este tipo de conjuntos de itemsets para agilizar las búsquedas de un determinado itemset generado. Si un itemset generado a partir de un combinación ya existe en el árbol *treeCombinations* el soporte interno de este se incrementa en uno. De esta manera, la primera vez que se ejecute el método *letsCombine* se generarán las combinaciones tamaño 2 de cada transacción válida del conjunto de datos y quedarán almacenadas en *treeCombinations* el total de itemsets generados por el dataset y su correspondiente soporte.

Posterior a este paso se procede a barrer el arbol *treeCombinations* para seleccionar aquellos Itemsets Frecuentes que hayan superado el soporte del sistema. Esta función se realiza haciendo uso del método *pruneCombinations*, de la clase *EquipAsso*,

que recibe como parámetros el árbol *treeCombination* y un árbol AvlTree llamado *treeFrequents*, donde se guardaran únicamente los Itemsets Frecuentes. El método *pruneCombinations* es un método recursivo y su implementación se muestra en el siguiente listado:

```
public void pruneCombinations(AvlTree treeCombinations,
                    AvlTree treeFrequents) {
   pruneCombinations(treeCombinations.getRoot(), treeFrequents);
}

private void pruneCombinations(AvlNode node, AvlTree treeFrequents) {
  if(node != null){
  pruneCombinations(node.getLeft(), treeFrequents);
  ItemSet itemset = node.getItemset();
  if(itemset.getSupport() >= support\_of\_system){
    treeFrequents.insertItemset(itemset);
  }
  pruneCombinations(node.getRight(), treeFrequents);
  }
}
```

Código 8.2.19: Función *pruneCombinations*

Al final de este método se liberan los recursos ocupados por *treeCombinations* y contaremos con un objeto *treeFrequents* donde están almacenados todos los Itemsets Frecuentes de un determinado tamaño. Para concluir el proceso se pregunta si el árbol *treeFrequents* contiene elementos, se usa el método *howMany* de la clase *AvlTree* para este propósito. De ser así, este árbol es almacenado en el arreglo *Trees* de la clase *EquipAsso* en su respectiva posición de acuerdo al tamaño de los itemsets que contiene y el método *findInDataset* retorna una señal de *true* al ciclo principal del método *run* que controla la continuación del algoritmo. El proceso se repetirá esta vez calculando las combinaciones e Itemsets Frecuentes tamaño 3. El algoritmo se detiene cuando el método *howMany* del árbol *treeFrequents* devuelva 0 lo que quiere decir que ya no se han generado Itemsets Frecuentes en esta iteración y el proceso de EquipAsso ha terminado. En este caso el método *findInDataset* retorna una señal de false y el ciclo principal de la clase *EquipAsso* concluye.

Paquete classification

Paquete c45 Este paquete implementa el algoritmo de clasificación C.4.5, el cual es una técnica de minería de datos que permite descubrir conocimiento por medio de la clasificación de atributos a través de la ganancia de información de los mismos, aplicando formulas para obtener la entropía de cada atributo con respecto a otros.

En primera instancia se hace un conteo de los distintos valores en el atributo objetivo, con el propósito de encontrar una entropía inicial, con la cual calcularemos las entropías de cada atributo, para saber cual es la que brinda la mayor ganancia de información, el atributo con mayor ganancia es el próximo nodo de árbol de decisión, el cual es una estructura que inicia en el nodo raíz o atributo objetivo, además esta compuesta por nodos internos, ramas y hojas. Los nodos representan atributos, las ramas son regla de clasificación y las hojas representan a una clase determinada.

El proceso es iterativo hasta que no haya atributos que clasificar. En TARYI C 45 esta implementado de la siguiente forma: El flujo de entrada es una conjunto de datos presentados en un TabelModel, el cual permite la conexión del algoritmo con distintos filtros de la etapa datacleaning. También Hacemos uso de una estructura de árbol N-Ario, en una clase la cual llamamos *TreeCounter*, para hacer el conteo eficiente de las múltiples combinaciones de los valores de un atributo con respecto a otros. A continuación se presenta el código fuente de la estructura del árbol N-Ario.

```
rows = Numero de Transacciones;
columns = Numero de Atributos;
root = Ruta de Atributos;
aux = Auxiliar de Atributo;
atributos_insertados = Atributos previamente insertados en el arbol;
route = Ruta de Nodos;
bd = Bandera Booleana;
  Attribute auxBrother;
  int size;
  root.frecuence--;
  size = route.getSize();
  if(route.firstGain) size = size - 1;
  for(int r = 0; r < rows ; r++ ) {
```

```
      aux = root;
      root.incrementFrecuence();
      if(route.firstGain){
        route.index = 1;
      } else {
        route.resetIndex();
      }
      for(int c = 0; c < size; c++ ) {
        String value = (String)dataIn.getValueAt(r, route.getIndex());
        if(aux.son == null){
          aux.son = new Attribute(value, aux, null, null);
          aux.son.route = findPath(aux.son);
          searchAttribute(aux.son);
          if(c == size - 2){
            if(r == 0){
              rootVariable = new nodeVariable(aux.son);
              currentVariable = rootVariable;
            } else {
              currentVariable.next = new nodeVariable(aux.son);
              currentVariable = currentVariable.next;
            }
          }
          aux = aux.son;
        } else { // si no esta vacio
          auxBrother = aux.son;
          aux = aux.son;
          bd = true;
          while(aux != null){
            if(aux.name.equals(value)){
              aux.incrementFrecuence();
              bd = false;
              break;
            }
            auxBrother = aux;
            aux = aux.brother;
          }
```

```
          if(bd){
            auxBrother.brother = new Attribute_
              (value, auxBrother.father, null, null);
            auxBrother.brother.route = findPath(auxBrother.brother);
            searchAttribute(auxBrother.brother);
            aux = auxBrother.brother;
          }
        }
      }
    }
```

Código 8.2.20: Estructura árbol N-Ario

Al resultado de este conteo, es aplicado las formulas de entropía para encontrar el atributo con mayor ganancia de información, la cual es obtenida a partir del siguiente criterio.

La ganancia de información de un atributo A, con respecto a un conjunto de ejemplos S es:

$$Gain(S, A) = Entropia(S) - \sum v \in Valores(A)|Sv|/|S|Entropia(Sv)$$

Donde:

Valores(A) es el conjunto de todos los valores del atributo A.
Sv es el subconjunto de S para el atributo A que toma valores v.
$Entropia(S) = -\sum pi \;\; Log_2 \;\; pi$ donde pi es la probabilidad que un ejemplo arbitrario pertenezca a la clase Ci.

El siguiente fragmento de código presenta la forma de encontrar entropía:

```
public double setEntropia(){
  double probabilidadInterna = 0.0;
  float division;
  float divExt;
  Attribute auxSon = this.son;
```

```
    while(auxSon != null){
      division = ((float)auxSon.frecuence / (float)this.frecuence);
      probabilidadInterna += (division) * log2(division);
      auxSon = auxSon.brother;
    }
    this.entropia = probabilidadInterna;
    divExt =  (float)this.frecuence / (float)this.father.frecuence;
    return divExt * probabilidadInterna;
  }

  public double log2(double value){
    if(value == 0.0) return 0.0;
    return Math.log(value)/Math.log(2);
  }
```

Código 8.2.21: Función para encontrar la entropía

Después de encontrar el atributo que presenta mayor ganancia de información, es vinculado a la ruta de una rama, para repetir el proceso recursiva e iterativa mente hasta conformar el árbol de decisión, que es implementado en un árbol eneario gráfico denominado *FinalTree*, el cual extiende la clase de Java *JTree*.

Ejemplo de funcionamiento del algoritmo C 45 en TARYI:

Primero en la tabla 8.3 se pueden observar datos de entrada, para el algoritmo C45:

En la tabla 8.3 el Atributo seleccionado como target o clase es "JUGAR TENNIS", lo cual significa que el objetivo de la Minería de Datos gira en torno a este atributo. Aplicamos el conteo, utilizando la estructura *TreeCounter*, sobre el atributo Target, con lo cual obtenemos 9 valores "SI" y 5 valores "NO", como lo observamos en el gráfico 8.3.

A estos resultados aplicamos la formula para obtener la entropía inicial.

$$E([9+, 5-]) = -(9/14)\ \ Log_2(9/14) - (5/14)\ \ Log_2(5/14) = 0{,}940$$

Aplicamos nuevamente el proceso de conteo, relacionando cada atributo, con el atributo objetivo, con el propósito de encontrar aquel que brinde mayor ganancia de información, se presenta el ejemplo de conteo con el atributo "VIENTO" en el gráfico 8.4.

DIA	ESTADO	TEMPER.	HUMEDAD	VIENTO	Jugar Tennis
D1	Soleado	Caliente	Alta	Debil	No
D2	Soleado	Caliente	Alta	Fuerte	No
D3	Nublado	Caliente	Alta	Debil	Si
D4	Lluvioso	Templado	Alta	Debil	Si
D5	Lluvioso	Fresco	Normal	Debil	Si
D6	Lluvioso	Fresco	Normal	Fuerte	No
D7	Nublado	Fresco	Normal	Fuerte	Si
D8	Soleado	Templado	Alta	Debil	No
D9	Soleado	Fresco	Normal	Debil	Si
D10	Lluvioso	Templado	Normal	Debil	Si
D11	Soleado	Templado	Normal	Fuerte	Si
D12	Nublado	Templado	Alta	Fuerte	Si
D13	Nublado	Caliente	Normal	Debil	Si
D14	Lluvioso	Templado	Alta	Fuerte	No

Cuadro 8.3: Conjunto de datos

Con estos resultados y la entropía inicial aplicamos la formula para obtener la en-

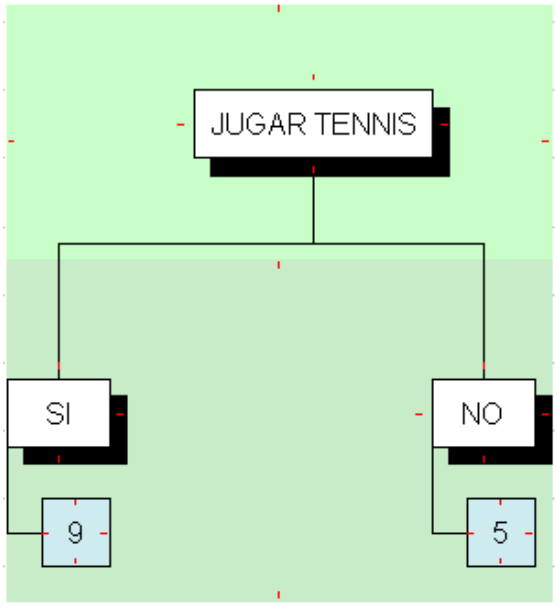

Figura 8.3: Conteo target

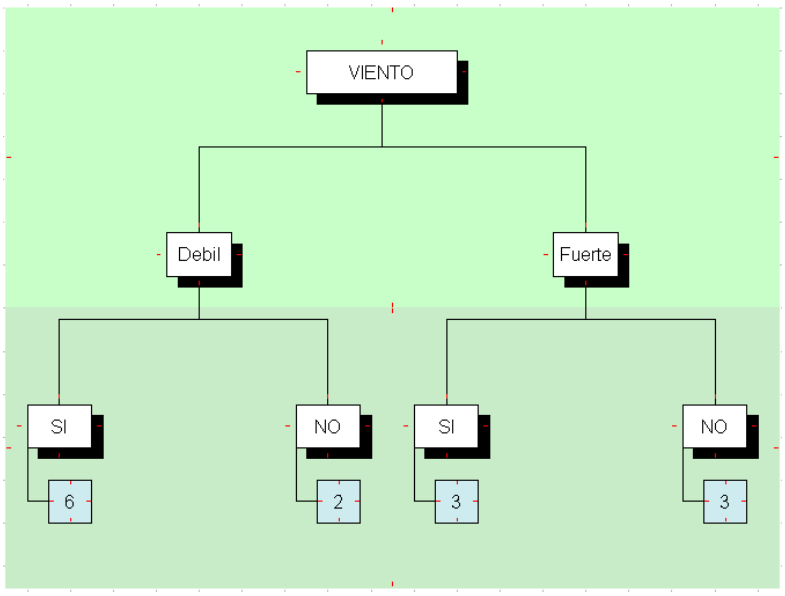

Figura 8.4: Conteo viento

tropía del atributo "VIENTO" relacionado con el atributo objetivo, de la siguiente forma:

El atributo "VIENTO" posee dos valores los cuales son débil y fuerte:
$|Sdebil| = [6+, 2-] Sfuerte = [3+, 3-]$

$Gain(S, Viento) = Entropia(S) - (Sdebil * Entropia(Sdebil) + Sfuerte * Entropia(Sfuerte) =$
$0{,}940 - 8/14 Entropia(Sdebil) - 6/14 Entropia(Sfuerte)$

$Calculamos \;\; E(Sdebil) = E([6+, 2-]) = -(6/8) \;\; Log_2(6/8) - (2/8) \;\; Log_2(2/8) =$
0,811
$E(Sfuerte) = E([3+, 3-]) = -(3/6) log2(3/6) - (3/6) log2(3/6) = 1$

Reemplazando :
$Gain(S, viento) = 0{,}940 - 0{,}463 - 0{,}428 = 0{,}048$
De una forma similar se aplican los anteriores procedimientos con cada atributo, para encontrar el atributo que brinde mayor ganancia de información para este nivel. Los resultados obtenidos son:

Gain(S,Estado) = 0.246 Gain(S,Humedad) = 0.151 Gain(S,Temperatura) = 0.029

Podemos observar que la mayor ganancia de información la brinda el atributo ESTADO igual a 0.246, lo cual significa que el nodo inicial en el árbol de decisión es ESTADO, a partir de este atributo encontraremos los siguientes nodos que suministren mayor ganancia en los tres valores del atributo, los cuales son "Soleado", "Nublado" y "Lluvioso", también encontramos que valor "Nublado" se parcializa hacia una decisión la cual es Jugar Tenis. El árbol de este nivel es el siguiente:

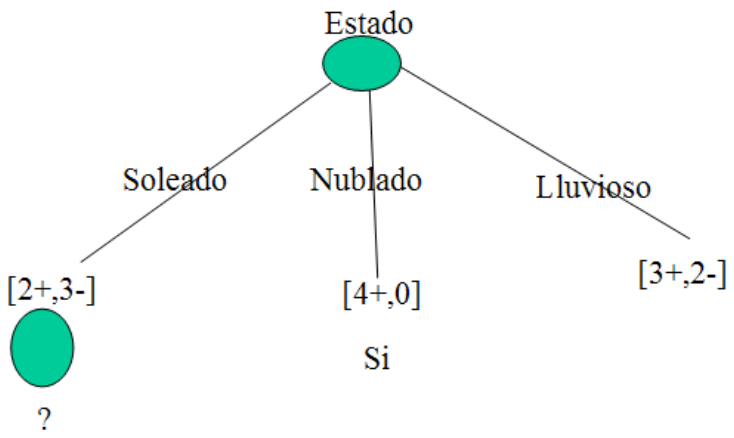

Figura 8.5: Árbol parcial

Ahora se encuentra el atributo que brinde la mayor ganancia para cada valor del atributo "ESTADO", haciendo el conteo de forma eficiente al encontrar los resultados para los tres valores simultaneamente. Como ejemplo lo haremos para Humedad, combinado con el atributo Estado, tal y como se puede observar en la gráfica 8.6.
A continuación se aplica la formula de entropía para el valor Soleado, del atributo Estado, de la siguiente forma

$Humedad(Alta[0+,3-], Normal[2+,0-])$
$Gain(Soleado, Humedad) = 0{,}970 - (3/5) * 0 - (2/5) * 0 = 0{,}970$

$Temperatura(Caliente[0+,0-], Templado[1+,1-], fresco[1+,0-]$
$Gain(Soleado, Temperatura) = 0{,}970 - (2/5) * 0 - (2/5) * 1 - (1/5) * 0 = 0{,}570$

$Viento(Debil[1+,2-], fuerte[1+,1-])$
$Gain(Soleado, Viento) = 0{,}970 - (3/5) * 0{,}918 - (2/5) * 1 = 0{,}019$

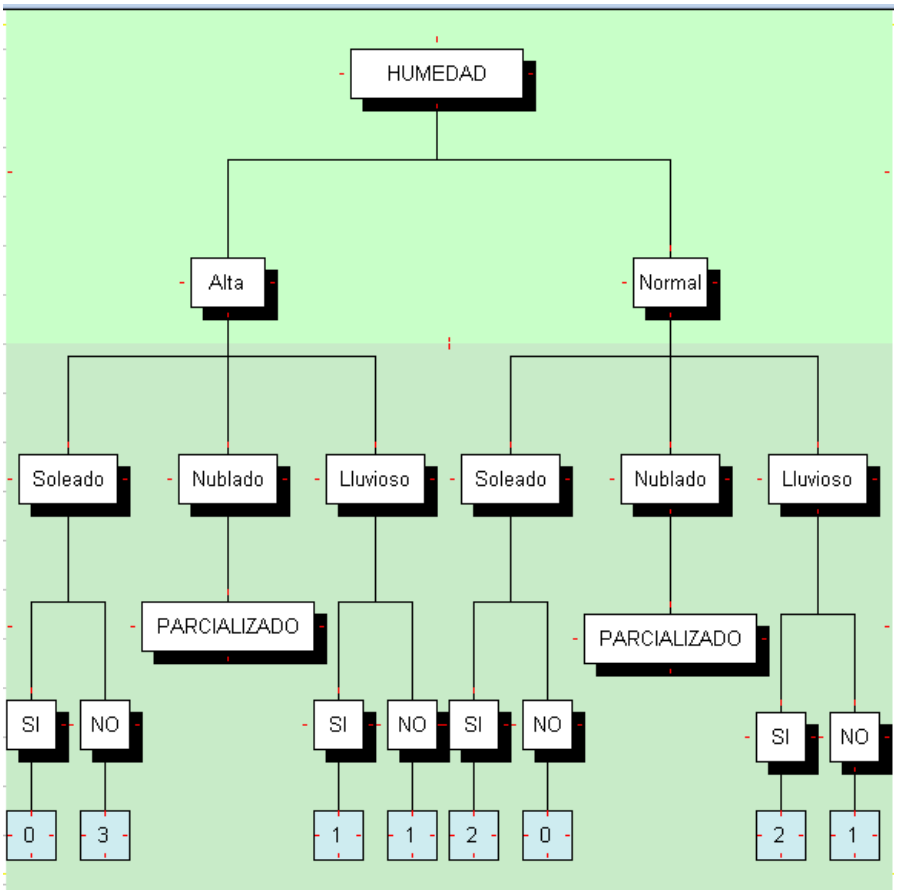

Figura 8.6: Conteo humedad estado

Podemos observar que para el valor Soleado, el atributo ganador es Humedad cuyos valores son Alta y Normal los cuales se parcializan en este nivel. Para el atributo Lluvioso, después de realizar el conteo se aplica la formula de entropía:

$Humedad(Alta[1+,1-],Normal[2+,1-])$
$Gain(Lluvioso,Humedad) = 0{,}97 - (2/5) * 1 - (3/5) * 0{,}917 = 0{,}0198$

$Temperatura(Caliente[0+,0-],Templado[2+,1-],fresco[1+,1-]$
$Gain(Lluvioso,Temperatura) = 0{,}97 - 0 - (3/5) * 0{,}917 - (2/5) * 1 = 0{,}0198$

$Viento(Debil[3+,0-],fuerte[0+,2-])$
$Gain(Lluvioso,Viento) = 0{,}970 - (3/5) * 0 - (2/5) * 0 = 0{,}970$

Podemos observar que el atributo ganador es Viento cuyos valores Fuerte y débil se parcializan en este nivel. A continuación se presenta el árbol de decisión definitivo:

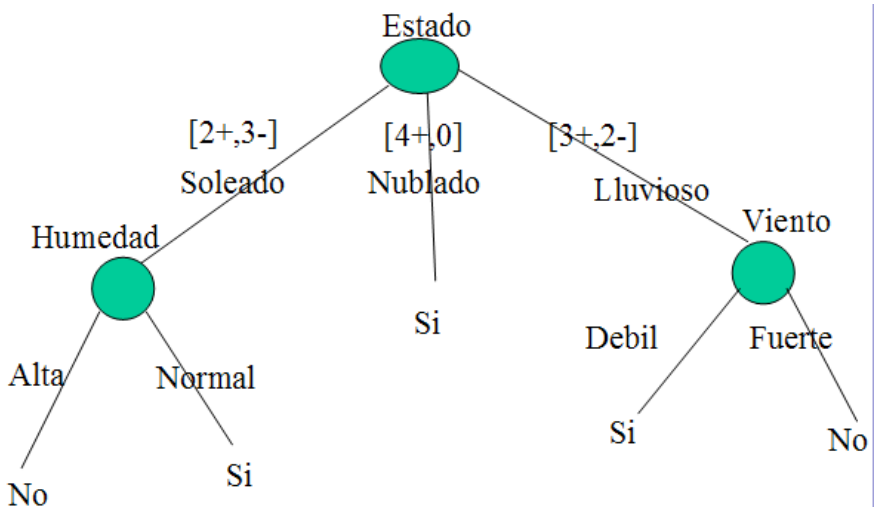

Figura 8.7: Árbol definitivo

Paquete mate A continuación se describen las clases implementadas en la programación del algortimo MateBy. Se habla más en detalle acerca de las estructuras de datos y los métodos más utilizados.

Clase MateBy Inicialmente es creado un objeto de la clase *MateBy*. Este objeto tiene la siguiente estructura:

dataSet de tipo DataSet: es un árbol N-ario en el que se almacenan las combinaciones que generadas y en el cual es posible llevar un conteo de cada una de las ocurrencias de una combinación dada.

entros de tipo arraylist: es un arreglo de datos en el que se va a almacenar las agrupaciones de combinaciones que tienen que hacerce para el cálculo de la entropia.

levels de tipo entero: se usa para almacenar el número del nivel de una hoja en el árbol dataSet.

attribute de tipo String: aqui se almacena el nombre del atributo que se encuentra inmediatamente después de la raiz y por el cual se inició el recorrido del arbol en

una búsqueda.

rules de tipo ArrayList: aqui se almacenan las reglas generadas. Este arreglo se alimenta del árbol dataset después del cálculo de la ganancia.

mateTree de tipo Tree: este es un árbol N-ario se crea para ser mostrado gráficamente.

Ya se ha descrito la estructura principal sobre la cual se va a desarrollar el algoritmo. A continuación se describen los métodos principales con las que se lleva a cabo la tarea de clasificación. Se omiten métodos triviales tales como recorridos del árbol, consultas e inserciones al mismo.

Método MateBy
Inicialmente se cargan los datos para alimentar al algoritmo. Cada transacción se almacena en un vector que es pasado como parámetro al método *combinations*. El atributo clase no se almacena en este vector y es pasado como otro parámetro al método *combinations*. En este método se aplica el operador MateBy generando todas las posibles combinaciones de las transacciones con el atributo clase. Cada una de las combinaciones es almacenada en el árbol N-ario *dataset*, cada rama del árbol representa una combinación diferente. Si se presenta el caso en el que una combinación se repite, esta no es almacenada nuevamente. Las hojas del arbol tienen una estructura diferente al resto de nodos en el árbol que les permite guardar el número de ocurrencias de una combinación. Este campo en las hojas es llamado *soporte* y será de gran utilidad posteriormente en el cálculo de la ganacia. La gráfica 8.8 muestra las estructuras mencionadas anteriormente. Al final se retorna dataset.

Método groupBranchs
Esta clase es la encargada de agrupar ramas del árbol o combinaciones que se estan asociadas y que deben unirse para luego calcular la ganancia. El criterio de agrupación se basa em la coincidencia del número de niveles y de la rutade cada hoja en el árbol. Ya que cada rama tiene un soporte asociado, al momento de agruparlas es necesario conservar ese soporte y además calcular el soporte acumulado al agrupar distintas ramas. Tanto estos datos como las hojas agrupadas son almacenadas en una estructura especial llamada *entro*. Esta estructura se describe en la figura 8.9. Cada uno de los elementos *entro* es almacenado en un arreglo llamado *entros* para mejorar la manipulación de estos objetos. El proceso continua hasta rrecorrer todas las ramas del árbol y hasta haber agrupado todas las hojas relacionadas entre si.

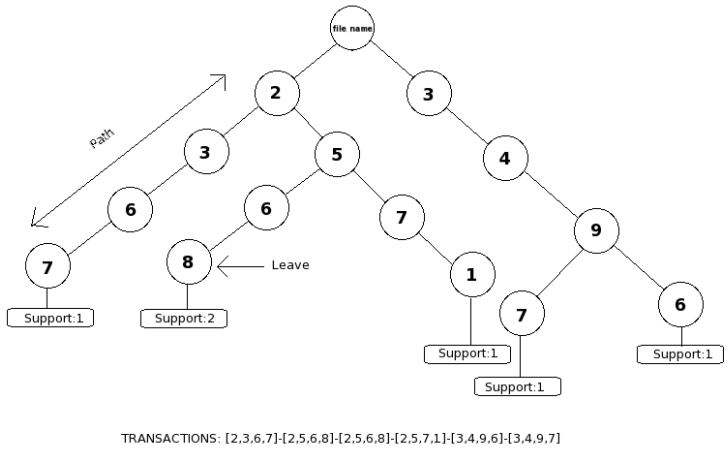

Figura 8.8: Árbol de decisión

Método entroAgrupation
Dentro de esta clase se recorre el arreglo *entros* y se recuperan las hojas agrupadas para calcular la entropía por medio del método *calculateEntropy*. Este nuevo dato es almacenado en *entro* por cada grupo de hojas. Ver fragmento de código 8.2.22.

Método gainCalculation
Dentro de esta clase se recurre nuevamente al arreglo *entros*. Hasta este momento se ha calculado la entropia de cada grupo de hojas y ahora pasamos a agrupar nuevamente para calcular la ganancia. La agrupación se hace de tal manera que las rutas de cada una de las hojas almacenadas en *entro* tenga el mismo número de niveles y que los nombres de los atributos que estan inmediatamente después dela raiz coincidan, de esta manera y debido a la organización del árbol al que hacen referencia las hojas se logra organizar las combinaciones de tal forma que se puede calcular la ganancia. Una de las partes más complejas en este paso es el control de los soportes. En el arreglo *entros* se intercala, por cada grupo de hojas, el valor de la entropia y el el soporte acumulado. Los ciclos implementados se ocupan de recorrer adecuadamente las estructuras para poder obtener todos los datos necesarios de para calcular la ganancia. Otro punto importante es que a medida que se recorre *entros* se van comparando las ganancias obtenidas y solo se trabaja por las ramas que obtengan el mayor valor en cada iteración.

```
...
if (leaf instanceof NodeF) {
  Iterator it = classValues.iterator();
  while (it.hasNext()) {
    ClassValue elem = (ClassValue) it.next();
    if ( ((NodeF)leaf).getItemF() == elem.getValue() ) {
    elem.incCounter( ((NodeF)leaf).getSupport() );
    newValue = false;
    break;
    }
  }
  if (newValue) {
    ClassValue value = new ClassValue( ((NodeF)leaf).getItemF(),
    ((NodeF)leaf).getSupport() );
    classValues.add(value);
  }
} else {
  Double token = new Double(leaf.toString());
  genEntro = genEntro + (-1)*(token / this.support);
  }
...
```

Código 8.2.22: Cálculo de Entropía

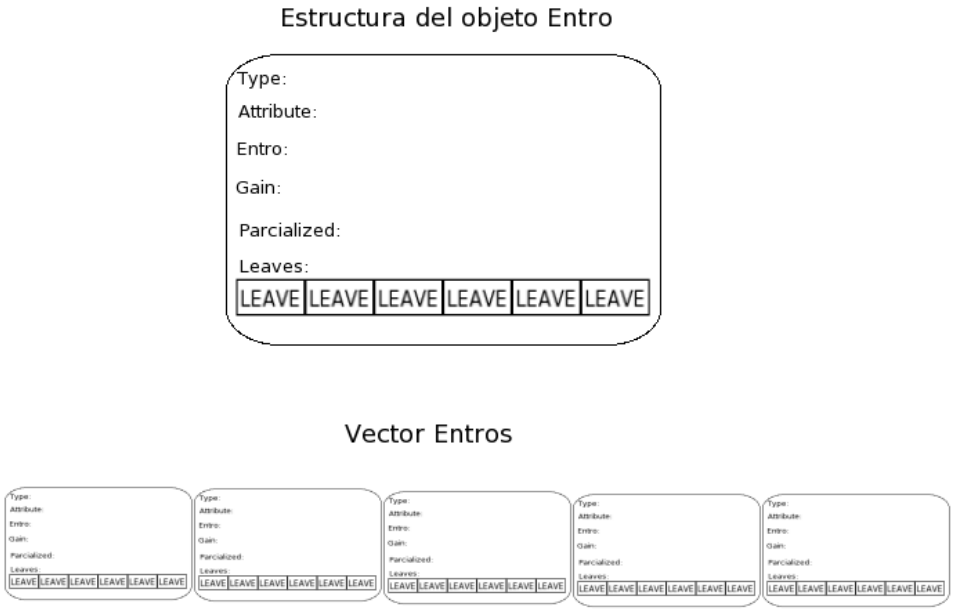

Figura 8.9: Estructura Entro

Método chooseNodes
Este método se ocupa de seleccionar los nodos que van a ir en el árbol de reglas. De las agrupaciones de hojas que obtuvieron la mayor ganacia se debe establecer la hoja que hace parte de la regal que va a pasar al árbol de reglas. A partir de la hoja se recorre el árbol de abajo hacia arriba para sacar la ruta. Esta ruta representa la regla generada.

Método buildRulesTree
Este método se encarga de construir el árbol de reglas que es mostrado gráficamente. Este árbol es de tipo Tree y será mostrado a través de un Jtree. Cada regla se decodifica al pasar del árbol *dataset* al árbol *rulestree*. Para esto se utiliza el diccionario contruido al momento de cargar los datos a la aplicación por medio del llamado al método *getRules*. Aquí l que se hace es hacer el proceso inverso al de la codificación para obtener los nombres reales de los atributos. Estos nombres son los que son mostrados finalmente en el árbol gráfico.

Método getRules
Se instancia un arreglo de cadenas en el que se van a almacenar los nombres de los atributos que se obtengan de cruzar los punteros de la rama en el árbol *datdaset* y el diccinario.

8.2.3. Paquete GUI

Dentro de la implementación de una interfaz gráfica amigable para el proyecto TariyKDD, se trabajó utilizando las funcionalidades del proyecto *Matisse*, el constructor de interfaces gráficas de usuario (GUI) propia de *NetBeans 5.0* que permite un diseño visual de las formas y su posterior programación.

Dentro del paquete *gui* de TariyKDD reposan dos paquetes: *KnowledgeFlow* e *Icons*. El primero contiene las formas utilizadas en la Interfaz principal de la aplicación. En el paquete *Icons* se aborda la programación para cada uno de los iconos involucrados en el proceso de descubrimiento de conocimiento y a los cuales tiene acceso el usuario para desempeñar una determinada tarea por ejemplo realizar un a conexión a una base de datos, ejecutar un determinado algoritmo o utilizar un visor para desplegar la información obtenida.

Paquete KnowledgeFlow

Se hablará primero de la implementación del paquete *KnowledgeFlow*. La interfaz principal de la aplicación esta contenida dentro de la clase *Chooser*, cuya implementación se muestra en la figura 8.10 y a su vez implementa el uso de diversas clases propias de la biblioteca gráfica Swing de Java como *JTabbedPanne, JSplitPane, JscrollPane, JLabel* así como extensiones de la clase *JPanel* donde se implementan funcionalidades propias a la aplicación como un área de trabajo donde tendrá lugar la construcción de un experimento KDD y donde, a través de la metodología de Arrastrar y Soltar (Drag'n Drop), se dispondrán los iconos que representan una determinada tarea dentro del proceso. Se extiende también *JComponent* donde se implementan funcionalidades de los iconos que representan cada acción dentro de la aplicación.

Dentro de la interfaz principal se pueden distinguir algunas secciones como son un Selector, en el cual se permite escoger una etapa dentro del proceso KDD, un panel, donde se despliegan los iconos asociados a una determinada etapa y una Area de Trabajo, donde se organizan los iconos seleccionados y se construye un experimento de descubrimiento de conocimiento.

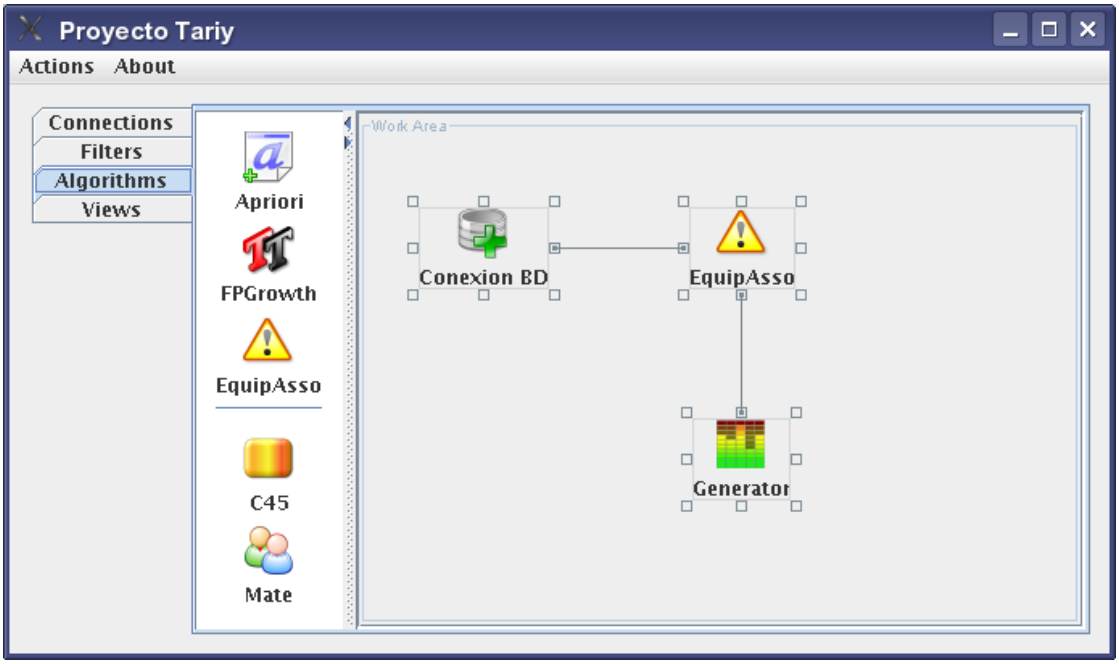

Figura 8.10: Clase Chooser. Interfaz principal de la aplicación

Cada una de estas partes se constituye como una nueva clase en el proyecto, que se gestionan con instancias propias de Java las cuales se mencionaron anteriormente. Con un *JTabbedPanne* se monta la selección de un determinado proceso desplegando etiquetas que identifican a cada uno de ellos, "Connections", para escoger una conexión hacia un conjunto de datos específica, puede ser a una base de datos o cargando un archivo plano, "Filters", donde se da la opción de escoger un determinado filtro que modificará el conjunto de datos que se haya cargado, "Algorithms", sección en la cual se escoje el algoritmo oportuno para realizar las tareas de minería de datos como tal y "Views", donde se despliegan opciones de visualización para organizar y mostrar al usuario los resultados obtenidos. Un detalle de la implementación de esta estructura se muestra en la Figura 8.11.

Al interactuar con las etiquetas del *JTabbedPanne* se verá modificado el contenido de la clase *Container*, la cual es una extensión de *JSplitPanne* y divide este componente en dos, izquierda y derecha. En el izquierdo se cargará un panel correspondiente a cada etapa seleccionada con el *JTabbedPanne*, los posibles paneles que se cargan se visualizan en la figura 8.12, y en el derecho se carga una instancia de la clase *MyCanvas*, que provee la funcionalidades de Drag'n Drop (Arrastrar y Soltar) entre los iconos involucrados en un experimento.

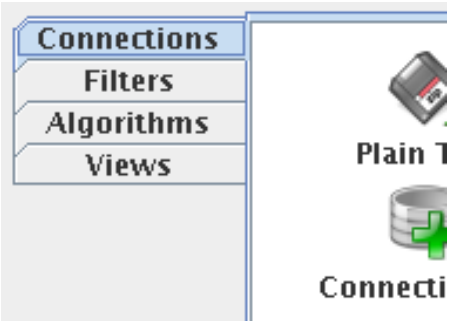

Figura 8.11: Etiquetas dentro de Chooser para la selección de etapas

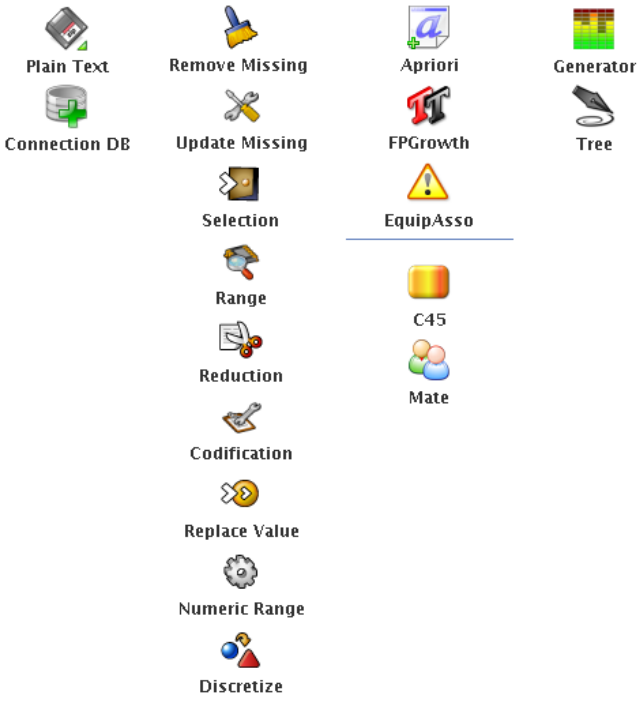

Figura 8.12: Paneles para cada etapa del proceso KDD

Cada panel extiende a *JPanel* y esta conformado por instancias de la clase *JLabel*, cada una de las cuales representará un icono perteneciente a esa etapa. En la instanciación de cada *JLabel* se carga una imagen alusiva a cada icono y un texto que lo identifica dentro del panel. Las imágenes correspondientes a cada icono, y en general todas las imágenes utilizadas en la herramienta, se cargan directa-

mente del Classpath de la aplicación dentro del paquete *images*. Es importante aclarar que en este momento también se asigna el nombre de cada *JLabel*, en su propiedad *name* a través del método *setName*, ese mismo nombre será relacionado posteriormente para identificar el icono seleccionado por el usuario desde el panel, luego, cada *JLabel* es adicionado a su respectivo panel. Un fragmento de código donde es asignado los valores para cada icono se muestra en el código 8.2.23.

```
jLabel.setIcon(new ImageIcon(getClass().getResource(
                                          "/images/connection.png")));
jLabel.setText(" Connection DB ");
jLabel.setName("connection");
jLabel.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
jLabel.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
panelConnections.add(jLabel);
```

Código 8.2.23: Asignación de valores a Iconos

Se puede notar en la primera línea como se asigna una imagen al *JLabel* utilizando el metodo *getClass().getResource(Ruta de la imagen)*. La clase *MyCanvas* es la encargada de implementar las funcionalidades de Drag'n Drop para los iconos involucrados en un determinado experimento. Al seleccionar con el mouse un determinado icono desde un panel, el *JLabel* asociado a ese icono es capturado escaneando los eventos del mouse cuando un icono es presionado y posteriormente liberado. El siguiente fragmento de código 8.2.24 ilustra como es capturado un *JLabel* al ser presionado con el mouse.

```
JLabel pressed;
container.addMouseListener(new java.awt.event.MouseAdapter() {
  public void mousePressed(java.awt.event.MouseEvent evt) {
    pressed = container.findComponentAt(evt.getPoint());
  }
});
```

Código 8.2.24: Captura de JLabel

Al adicionar un *MouseListener* al objeto *container*, el *JSplitPanne* que contiene los paneles y el área de trabajo, este captura el evento *pressed* del mouse cuando este ha sido presionado. En ese momento, el evento es capturado y a través del método *getPoint* tenemos la posición dentro del contenedor donde fue generado el

click. El método *findComponent()* devolverá el componente encontrado en el punto donde fue presionado el mouse.

Cuando el mouse se libera, este evento es igualmente capturado y se procede a identificar cual fue el ícono seleccionado desde el panel para desplegarlo correctamente sobre el objeto *MyCanvas*. Esto se efectúa al consultar la propiedad *name* del objeto *pressed* que fue capturado durante el evento anterior. Dependiendo del nombre del componente se procede a instanciar un objeto de la clase *Icon*.

La clase *Icon* fue construida extendiendo a un *JPanel* y contiene una instancia de la clase *JLabel*, que contiene la imagen y el texto asociados a ese icono, y un total de 8 instancias de la clase *Conector*, esta clase fue diseñada para identificar secciones dentro del icono donde el usuario pueda hacer un click y relacionar un icono con otro. En la figura 8.13 se muestra la inclusión de 2 instancias de la clase *Icon* y la relación establecida entre ellos a través de sus conectores.

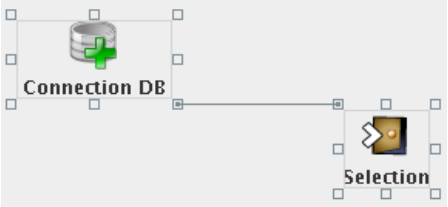

Figura 8.13: Implementación de la clase Icon

Se implementa dentro de esta clase la funcionalidad un menú desplegable donde se pueda incluir funciones propias para cada ícono y la inclusión de una animación que se activa cuando el ícono esta realizando un proceso determinado. Estas funcionalidades serán heredadas a todas las clases que extiendan de la clase *Icon*.

El menú desplegable se logra al usar las clases *JPopupMenu* y *JMenuItem*. Por defecto, todas las instancias de *Icon* y las clases que hereden de él tendrán implementada la opción *Delete*, que borrará el icono del área de trabajo y descargará la clase *Icon* correspondiente de la clase *MyCanvas* que la contiene. Para ello, se instancia un objeto de la clase *JMenuItem* y se setéa su propiedades de *name* a "Delete", posteriormente se procede a adicionar esta instancia de *JMenuItem* al *JPopupMenu* asociado a la clase *Icon*. Cuando un usuario seleccione la opción *Delete* se dispara el método *mnuDeleteActionPerformed* que se encarga de descargar las conexiones que tenga asociado este ícono con otros iconos y borrar el componente del área de trabajo. Nuevas adiciones al menú desplegable pueden ser hechas al

instanciar objetos *JMenuItem* e incluirlos dentro del *JPopupMenu* de la clase *Icon*.

La clase *Conector* fue construida extendiendo *JComponent* ya que debía proveer facilidades de captura de los eventos del mouse y ser dibujado de manera especial cuando este disponible y diferente cuando ya haya sido seleccionado. A continuación se muestra un fragmento de código 8.2.25 donde se ilustra el dibujado de esta clase al sobrecargar el método *paint* de la clase *JComponent*.

```
public synchronized void paint(Graphics g){
    g.setColor(Color.Blue);
    g.drawRect(0, 0, 6, 6);
    if(selected){
        g.fillRect(2, 2, 3, 3);
    }
}
```

Código 8.2.25: Dibujado de la clase conector

Vemos que al método le llega una instancia *g* del la clase *Graphics* que será el objeto donde podremos dibujar. A través de los métodos de esta clase podemos escoger colores para el conector y dibujar un rectángulo de 7x7 píxeles que representará el área del conector que puede ser pulsada por el usuario para su selección y dibujará un rectángulo relleno en el centro del conector si este ha sido ya seleccionado.

La clase *Icon* como tal no es incluida directamente sobre la clase *MyCanvas*, o área de trabajo. Existen una serie de clases que extiende a la clase *Icon* y se corresponden con cada tarea desempeñada dentro del proceso KDD. Existen un total de 7 extensiones a la clase *Icon* y estas son *DBConnectioIcon, FileIcon, FilterIcon, AssociationIcon, ClassificationIcon, RulesIcon* y *TreeIcon* asociadas a la conexión con una base de datos, conexiones con archivos planos, filtros para la selección y preprocesamiento de un conjunto de datos, algoritmos de asociación, algoritmos de clasificación, visualización de resultados de reglas de asociación y visualización de árboles de decisión respectivamente. Estas 7 clases, junto con otras que apoyan la tarea que cumplen, están contenidas en 7 paquetes dentro del paquete *Icons* que será explicado posteriormente y que hace parte a su vez del paquete *gui*.

Para cada una de las clases heredadas se maneja de manera independiente el contenido de su *JPopupMenu* así como la imagen y el texto asociados a cada tipo de icono. El resultado final para algunos tipos de icono se muestran en la figura 8.14.

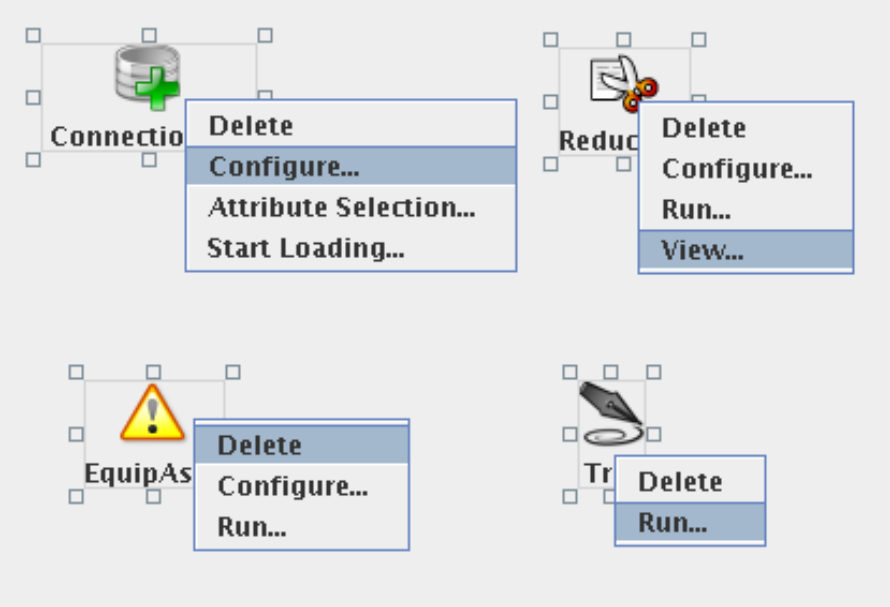

Figura 8.14: Clases heredadas de la clase *Icon*

La clase *MyCanvas* será la encargada de gestionar las conexiones entre los ícono y su movimiento sobre el área de trabajo. Esta clase extiende un *JPanel* y monitorea los eventos del mouse referentes a los clicks del usuario. Los eventos que tiene contemplados son *MousePressed* (mouse presionado), *MouseDragged* (click sostenido), *MouseReleased* (mouse liberado), *MouseClicked* (un click sencillo).

Para el evento *MousePressed*, se identifica si el click fue sobre un ícono o sobre uno de sus conectores. Si fue sobre un icono este es almacenado en la variable *selectedIcon*, pero si fue sobre un conector se almacena en la variable *selectedConector*. Para esta tarea se utiliza el método *findComponentAt(MouseEvent)* que se explicó anteriormente. Al final de este método, y en general todos los métodos que involucran cambios sobre el área de trabajo, se llama al método *repaint()*que se encarga de redibujar la clase *MyCanvas* invocando directamente el método *paint(Graphics g)* de esta clase y que se explicará posteriormente. Se presenta el siguiente fragmento de código 8.2.26 para explicar el método *MousePressed*.

Para el evento *MouseDragged* se identifica cual fue el componente seleccionado en el evento *MousePressed*, si se trata de un ícono este método se encarga de redibujarlo a medida que se mueve el mouse pero si se trata de un conector, se traza una línea entre el conector seleccionado y el puntero del mouse a medida que este se

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    Component press = this.findComponentAt(evt.getPoint());
    //Si se presiono un Icono
    if(press instanceof Icon){
        selectedIcon = (Icon)press;
    }
    //Si se presiono un Conector
    else if(press instanceof Conector){
        selectedConector = (Conector)press;
    //Se redibuja el area de trabajo para actualizar los cambios
    repaint();
}
```

Código 8.2.26: Click sobre un ícono

mueve. Estas dos acciones se realizan en el método *paint(Graphics g)* de la clase *MyCanvas*. Se muestra el siguiente fragmento de código 8.2.27 para ilustrar lo hecho en este método.

Para el evento *MouseRelease* se debe identificar igualmente cual componente esta seleccionado si se trata de un ícono, este evento se limita a liberar la variable *selectedIcon* para no seguir cambiando su posición, si se trata de un *Conector* se debe identificar en que punto es liberado, si coincide con un conector de otro ícono que este disponible se establecerá una relación entre ambos íconos trazando una línea entre sus conectores involucrados. Esta relación será almacenada en un arreglo donde se guardaran instancias de la Clase *Connections*. Esta clase se construye a partir de los dos conectores involucrados en la relación y servirá posteriormente para trazar todas las relaciones existente sobre el área de trabajo durante la invocación al método *paint(Graphics g)*. Un ejemplo de una relación establecida se puede apreciar en la figura 8.13. Se muestra a continuación un fragmento de código 8.2.28 explicando este método.

Durante el evento *MouseClicked* se revisa la posibilidad de, si el click fue sobre un conector que esta seleccionado, eliminar esa relación o si el click fue sobre el cuerpo de un ícono, desplegar el menú emergente. El siguiente fragmento de código 8.2.29 ilustra este método.

La clase *MyCanvas* tiene sobrecargado el método *paint(Graphics g)* lo que permite aprovechar las propiedades de dibujo de la clase que extiende (*JPanel*) así como

```
private void formMouseDragged(java.awt.event.MouseEvent evt) {
  //Si un Icono fue ya selecconado
  if(selectedIcon != null){
    //Se captura la posicion del puntero
    int x = evt.getX();
    int y = evt.getY();
    //Se calcula la nueva posicion del icono
    //a partir de la posicion del mouse
    selectedIcon.setLocation(x - selectedIcon.getWidth() / 2,
                             y - selectedIcon.getHeight() /2);
    //Si un Conector fue seleccionado
  } else if(selectedConector != null){
    //se capturan las coordenadas del puntero del mouse
    xMouse = evt.getX();
    yMouse = evt.getY();
  }
  //Se redibuja la forma para actualizar los cambios
  //Redibujar el Icono o trazar una linea
  repaint();
}
```

Código 8.2.27: Evento Arrastrar y soltar

adicionar nuevas figuras al interactuar sobre la clase *Graphics* asociada a esta clase y que nos provee de diferentes métodos para trazar figuras, escoger colores y repintar los componentes gráficos que estén contenidos dentro de la clase y que hayan cambiado de posición.

Durante la implementación de este método primero se hace un llamado al método *paint(Graphics g)* de la clase superior para que haga un redibujado de los componentes que contiene y de esta manera redibujar todos los íconos que posee en sus nuevas posiciones así como los bordes y colores propios de la clase heredada. Posteriormente se recorre el arreglo *connections* que posee todas la conexiones entre lo conectores y los íconos a los que pertenecen trazando líneas que relacionan de manera visual un ícono con uno o más de ellos. Al recorrer el arreglo *connections* se extraen de él objetos de la Clase *Connection*. Esta clase consiste de dos instancias de la Clase *Conector*, identificadas con los nombres *from* y *to*, haciendo alusión a el conector desde donde viene la relación y hacia donde va. Recordemos que la Clase *Conector* hereda a la clase *JComponent*, es decir que herada los métodos

```
private void formMouseReleased(java.awt.event.MouseEvent evt) {
  Component press = this.findComponentAt(evt.getPoint());
  //Si fue liberado sobre un Conector
  //y ya existia un Conector seleccionado
  if(press instanceof Conector) && selectedConector != null){
    //Se captura el nuevo conector
    Conector releaseConector = (Conector)press;
    //Se marca como seleccionado
    releaseConector.selected = true;
    //Se guarda la relacion en el arreglo connections
    //de clases Connection
    connections.add(
             new Connection(selectedConector, releaseConector));
    //Se libera la variable selectedConector
    selectedConector = null;
  }
  //Si el click se libera sobre un Icono
  if(press instanceof Icon)){
    //Solo se libera la variable selected Icon
    selectedIcon = null;
  }
  //Se repinta el area de trabajo para actualizar los cambios
  repaint();
}
```

Código 8.2.28: Liberación del click del mouse

getX() y *getY()* para calcular su posición dentro del la clase que los contiene, que en este caso es *MyCanvas*. De esta manera podemos calcular las coordenadas de los conectores y trazar las líneas que representan la relación. A continuación se presenta un fragmento de código 8.2.30 que ilustra lo explicado anteriormente.

Paquete Icons

Dentro del paquete *GUI* también se encuentra contemplado un paquete llamado *Icons* que se encarga de organizar cada una de las extensiones hechas a la clase *Icon* y aquellas clases que sirven de apoyo para las acciones propias de cada una de estas clases, por ejemplo las formas de la interfaz gráfica encargadas de capturar información del usuario y que se disparan al seleccionarlas del menu contextual de cada ícono.

```
private void formMouseClicked(java.awt.event.MouseEvent evt) {
  //Se captura el componente seleccionado con el click
  Component press = this.findComponentAt(evt.getPoint());
  //Si fue presionado un Icon
  if(press instanceof Icon){
    selectedIcon = (Icon)press;
    //Si se trata de un click secundario
    if(evt.getButton() == evt.BUTTON2
                          || evt.getButton() == evt.BUTTON3){
      //Despliega el menu emergente
      selectedIcon.getPupMenu().show(
            evt.getComponent(), evt.getX(), evt.getY());
    }
    //Libera la seleccion
    selectedIcon = null;
  } else if(press instanceof Conector){
    selectedConector = (Conector)press;
    //Si ese conector ya esta seleccionado
    if(selectedConector.selected){
      //Se elimina
      this.removeConector(selectedConector);
    }
  }
}
```

Código 8.2.29: Evento Mouse Clicked

Cada clase se encuentra contenida dentro de un nuevo paquete lo que quiere decir que dentro del paquete Icons existe un total de 7 nuevos paquetes que son DB-Connection, File, Filters, Association, Classification, Rules y Tree. Durante esta explicación se abordaran los paquetes Association, Classification, Rules y Tree. Los tres primeros paquetes se explicarán con más detalle en otras secciones de este capítulo dado el nivel de complejidad que estos revisten.

Paquete Association En este paquete se encuentra la clase *AssociationIcon* y la clase *configureSupport*. Esta extiende la clase *Icon* adicionando dos nuevas entradas al menú contextual que se corresponde a *Configure*, para configurar el soporte pedido al usuario, y *Run*, que ejecuta el algoritmo escogido por el usuario

```
public synchronized void paint(Graphics g){
    int  xd, yd, xh, yh;
    super.paint(g);
    Iterator it = connections.iterator();
    while(it.hasNext()){
        Connection aux = (Connection)(it.next());
        xd = aux.from.getX();
        yd = aux.from.getY();
        xh = aux.to.get_X();
        yh = aux.to.get_Y();
        g.setColor(colorEdge);
        g.drawLine(xd, yd, xh, yh);
    }
}
```

Código 8.2.30: Conectores y trazos de líneas

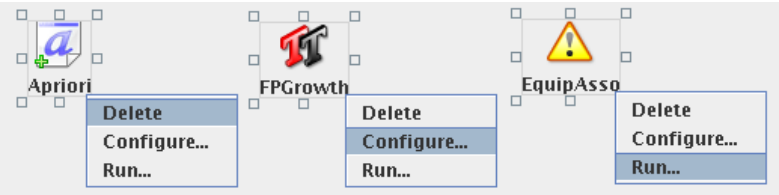

Figura 8.15: Estados de la Clase AssociationIcon

y que puede contener tres estados al hacer alusión al algoritmo Apriori, FPGrowth o EquipAsso. Estos estados se muestran en la figura 8.15.

En la gráfica se puede apreciar el contenido del menú desplegable de cada ícono: *Delete*, por defecto presente en todas las clases que hereden de Icon, *Configure*, desplegará una instancia de la Clase *configureSupport* donde el usuario introduce el soporte del sistema para este experimento. La figura 8.16 ilustra el contenido de esta ventana.

La ultima opción del menú es *Run*, aquí esta clase hará instancias de las clases *Apriori*, *FPGrowth* o *EquipAsso* según sea el caso pasando como parámetros un objeto de la clase *DataSet*, que puede provenir de una instancia de la clase *DB-ConnectionIcon* o de una instancia de *FileIcon* con la cual se haya establecido una relación, y el soporte del sistema capturado con la ventana anteriormente descrita. El siguiente fragmento de código 8.2.31 explica lo ejecutado por este método.

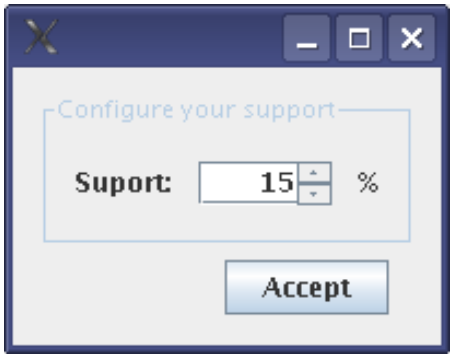

Figura 8.16: Captura del soporte del sistema

El resultado de ejecutar cualquiera de las clases de asociación que se han implementado devolverá a la Clase *AssociationIcon* un Vector el cual contiene un arreglo de los arboles Avl (instancias de la Clase *AvlTree*) que organizan el conjunto de itemset frecuentes obtenidos al ejecutar un determinado algoritmo. Este vector es guardado en la variabel *trees*, variable global a esta clase que posteriormente será pasada a un objeto de la Clase *RulesIcon* para que despliegue las reglas obtenidas al analizar el conjunto itemsets frecuentes.

Las clases *Apriori*, *FPGrowth* y *EquipAsso* son explicadas con más detalle en otras secciones de este capítulo.

Paquete Classification En este paquete se encuentra la clase *ClassificationIcon*. Esta extiende la clase *Icon* adicionando una nueva entrada al menú contextual que se corresponde a *Run*, que ejecuta el algoritmo escogido por el usuario y que puede contener dos estados al hacer alusión al algoritmo C45 o Mate. Estos estados y las entradas al menú contextual se muestran en la figura 8.17.

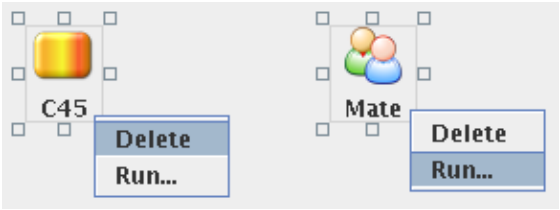

Figura 8.17: Estados de la Clase ClassificationIcon

La única opción implementada en el menú es *Run* ya que este tipo de algoritmos

```
private void mnuRunActionPerformed(java.awt.event.ActionEvent evt) {
    if(algorithm.equals("Apriori")){
        Apriori apriori = new Apriori(dataset, support);
        apriori.start();
        trees = apriori.getFrequents();
    } else if(algorithm.equals("FPGrowth")){
        FPGrowth fpgrowth = new FPGrowth(dataset, support);
        fpgrowth.start();
        trees = fpgrowth.getFrequents();
    } else if(algorithm.equals("EquipAsso")){
        EquipAsso equipasso = new EquipAsso(dataset, support);
        equipasso.start();
        trees = equipasso.getFrequents();
    }
}
```

Código 8.2.31: Ejecución de un comando con el mouse

no requiere información directa del usuario aparte del conjunto de datos. Al seleccionar la opción *Run* esta clase hará instancias de las clases *C45* o *Mate* según sea el caso pasando como parámetros un objeto que implemente la interfaz *TableModel* dentro de la variable *dataIn*, que puede provenir de una instancia de la clase *FilterIcon*, donde se ha seleccionado el atributo clase del conjunto de datos, con la cual se ha establecido una relación. El siguiente fragmento de código 8.2.32 explica lo ejecutado por este método.

El resultado de ejecutar cualquiera de los algoritmos de clasificación será dos objetos de las Clases *JPanel* y *ArrayList*. El primero contendrá una extensión de *JPanel* donde se despliega un objeto de tipo *JTree* que contiene gráficamente el árbol de decisiones que resulta del análisis y el *ArrayList* tendrá el conjunto de reglas que están contenidos dentro del árbol de decisiones de una manera textual. Estas dos clases serán pasadas como parámetros a la Clase *TreeIcon* encargada de desplegar de manera visual su contenido.

Las clases *C45* y *Mate* son explicadas con más detalle en otras secciones de este capítulo.

```
Private void mnuRunActionPerformed(java.awt.event.ActionEvent evt) {
    if(algorithm.equals("C45")){
        c45 c = new c45(dataIn);
        c.start();
        TreePanel = c.view;
        RulesText = c.rules();
    } else if(algorithm.equals("Mate")){
        Mate mate = new Mate(dataIn);
        mate.start();
        TreePanel = c.view;
        RulesText = c.rules();
    }
}
```

Código 8.2.32: Ejecución de un algoritmo

Paquete Rules Después de aplicar a un conjunto de datos, cualquiera de los tres algoritmos, Apriori, EquipAsso o FPGrowth, a través de este paquete los itemsets frecuentes obtenidos pueden ser observados en forma de reglas de asociación.

El paquete *Rules* hace parte del paquete *Icons*, que a su vez hace parte del paquete *GUI*. Las clases que conforman este paquete son: *RulesIcon, configureConfidence, RulesTableModel y showRules*.

En la gráfica 8.18 se pueden ver las funciones que esta clase implementa.

RulesIcon extiende a la clase *Icon* que hace parte del paquete *KnowledgeFlow* y a su vez del paquete *GUI*, por tanto automáticamente hereda su menú con la opción por defecto *delete*, la cual permite al usuario eliminar el icono visualización de reglas de asociación o en el interfaz llamado *Generator*. Por otra parte, la clase *RulesIcon* añade al menú de *Icon* las opciones Configure y Run.

La opción **Configure** abre una pequeña ventana, controlada por la clase *configureConfidence*, la cual hace uso de un *JSpinner*, clase descendiente de *javax.swing*, que permite introducir en un campo de entrada un valor numérico a través del cual el usuario determina la confianza con la cual se van a mirar las reglas de asociación. La ventaja de usar un *JSpinner* es la facilidad con la que se pueden validar las entradas, ya que mediante la clase *SpinnerNumberModel* se establece el límite inferior y superior que el usuario esta en condición de digitar y si este intro-

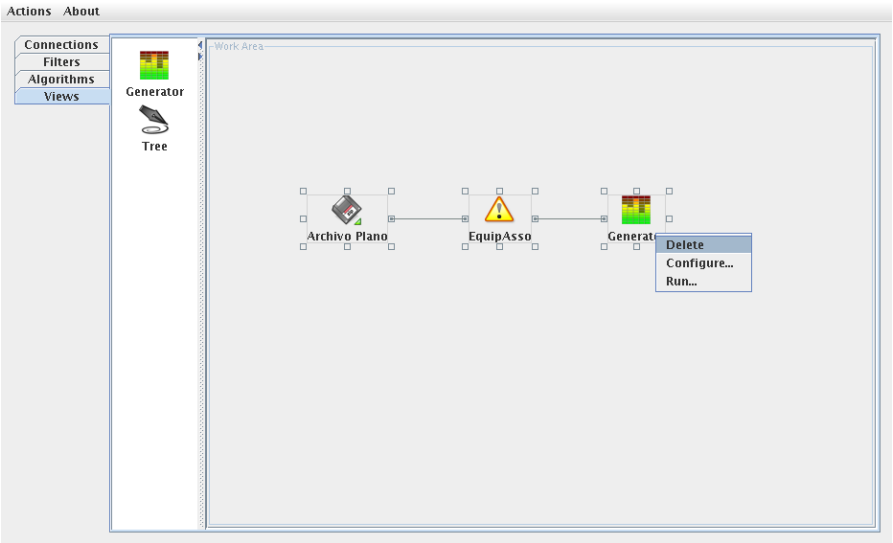

Figura 8.18: Funciones de *RulesIcon*

Dato códificado	**Item**
1	Jabón
2	Arroz
3	Champú
4	Desodorante

Cuadro 8.4: Diccionario de datos

duce una entrada invalida, en cuanto se pierda el foco del *JSpinner* este tomara el último valor correcto.

Por otra parte, la opción **Run** muestra las reglas de asociación al usuario en una *JTable*. Las reglas de asociación se deben almacenar en un array de cadenas, en la clase *AssocRules*. Antes la clase *AssocRules* guarda las reglas sin codificar o sea guarda los códigos del **diccionario de datos**. Sea por ejemplo el diccionario del cuadro 8.4.

De acuerdo a lo anterior, una posible regla por ejemplo podría ser: 1$\hat{2}$ ¿3, almacenada en la clase *Rules* la cual tiene los siguientes campos:

Atributos:
`String antecedent` //Cadena que almacena el antecedente.
`String concecuent` //Cadena que almacena el concecuente.

A continuación, las reglas se decodifican, o sea que la regla $1\hat{2}$ ¿3 se almacenará en el array de la clase *AssocRules*, pero de la siguiente manera: JabónÂrroz ¿Champú. De la misma manera todas las reglas de asociación encontradas se almacenan en la clase AssocRules.

A partir de las reglas almacenadas en la clase *AssocRules*, se construye un modelo propio para *JTable* y de esta forma se muestran las reglas en la tabla. Para construir el modelo, entonces las reglas almacenadas en un array de cadenas se pasan al constructor de la clase *RulesTableModel*, la cual implementa los respectivos métodos para construir el modelo.

La tabla que muestra los datos se encuentra en la clase *showRules*, e implementa un evento del mouse para que cuando el usuario haga click en el encabezado **Rules** de la tabla, las reglas se ordenen por un criterio que ha sido determinado "Pepita de oro", o sea que en las primeras posiciones se indicaran las reglas cuyo antecedente sea menor que su concecuente, por ejemplo la regla: Arroz ¿Jabón, Desodorante, es una "pepita de oro", ya que un producto lleva a comprar dos más. Y cuando el usuario haga click en el encabezado *Confidence* de la tabla, las reglas se ordenaran de mayor a menor confianza. Para implementar el click sobre el encabezado de la tabla, se hace lo siguiente:

Se añade un "escucha" a la tabla a través del método: *addJTableHeaderListener*.

El "escucha" que se encuentra en la función *addJTableHeaderListener*, va a estar pendiente del evento click del mouse mediante la clase *MouseAdapter*.

Mediante el método de *JTable*, *convertColumnIndexToModel* se obtiene la columna en la cual el usuario hizo click. La tabla que muestra las regas tiene 3 columnas, por tanto si el código retornado es 1 quiere decir que el usuario hizo click en la segunda columna, por tanto las reglas serán ordenadas de acuerdo al criterio "pepita de oro". Pero si el código retornado es 2, entonces el criterio de ordenamiento es descendiente de acuerdo a la confianza. Para mayor detalle se puede observar el código fuente del método que supervisa los eventos del mouse, a continuación:

```
// Se a\~nade el escucha.
this.addJTableHeaderListener();
public void addJTableHeaderListener() {
  MouseAdapter mouseListener = new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
      TableColumnModel columnModel = tblRules.getColumnModel();
      int viewColumn = columnModel.getColumnIndexAtX(e.getX());
      System.out.println("column model " + viewColumn);
      int column = tblRules.convertColumnIndexToModel(viewColumn);
      System.out.println("column " + column);
      // Ordenar elementos de la columna de la tabla.
      if(e.getClickCount() == 1 && column != -1) {
        // Ordenar por confianza.
        if(column == 1) {
          rules.sortByGoldStone(0);
          rules.sortByGoldStone(1);
          RulesTableModel rtm = new RulesTableModel(
                                rules.getRules());
          tblRules.setModel(rtm);
        } else if(column == 2) {
            RulesTableModel rtm = new RulesTableModel(
                                  rules.sortByConfidence());
            tblRules.setModel(rtm);
        }
      }
    }
  };
  JTableHeader header = tblRules.getTableHeader();
  header.addMouseListener(mouseListener);
}
```

Código 8.2.33: Método *addJTableHeaderListener*

8.2.4. Paquete Conexión

El paquete DBConnection se encuentra contenido dentro del paquete *Icons* que a su vez se encuentra dentro del paquete *GUI*. En este paquete se encuentran las clases necesarias que soportan una conexión a bases de datos a través de un driver JDBC. Hacen parte de este paquete las clases *DBConnectionIcon, connectionWizard, Table, MyCanvasTable, SelectorTable y ScrollableTableModel*.

DBConnectionIcon

El paquete DBConnectionIcon es una clase que extiende a Icon del paquete GUI KnowledgeFlow y se encarga de proveer un menú desplegable que guíe por las etapas de conexión, selección y carga de datos desde una base de datos. La figura 8.19 muestra la implementación de la clase DBConnectionIcon con sus opciones de menú.

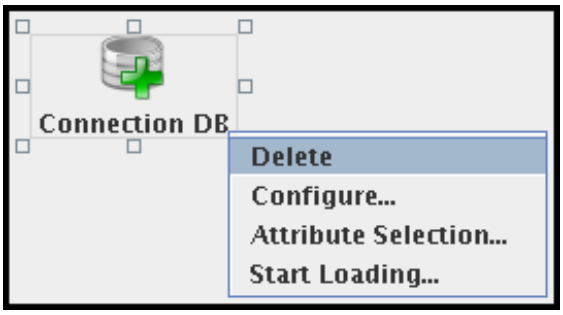

Figura 8.19: Implementación de la Clase DBConnectionIcon

Al heredar de la clase *Icon, DBConnectionIcon* posee por defecto la opción *Delete* en su menú para eliminar este ícono del área de trabajo. Al escoger la siguiente opción, *Configure*, se abre una instancia de la clase *connectionWizard*, la implementación de esta clase se puede ver en la figura 8.20.

En esta se recoge la información necesaria para establecer una conexión a través del controlador JDBC escogido en el campo JDBC Driver. En este punto hay que aclarar que se hace uso del *API SQL* de Java, que es un conjunto de clases dispuestas a la interacción con controladores JDBC y al cual se accede importando el paquete *java.sql*. Las clases e interfaces involucradas en la conexión a bases de datos son:

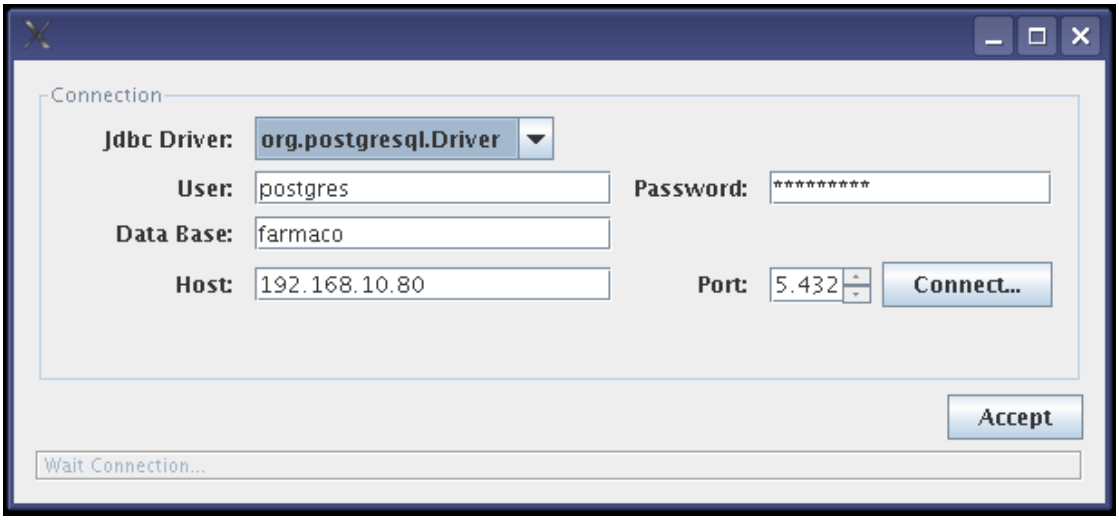

Figura 8.20: Implementación de la Clase DBConnectionIcon

java.sql.DriverManager.
Provee los servicios básicos para el manejo de un conjunto de controladores JDBC

java.sql.DatabaseMetaData.
Información comprensiva sobre la base de datos en su totalidad.

java.sql.Connection.
Una conexión (sesión) con una base de datos específica. Se ejecutan las sentencias SQL y los resultados se devuelven dentro del contexto la conexión.

java.sql.Statement.
El objeto usado para ejecutar una sentencia estática SQL y devolver los resultados que esta produce.

java.sql.ResultSet.
Una tabla de los datos que representan un sistema del resultado de la base de datos, que es generado generalmente ejecutando una sentencia SQL que consulta a la base de datos.

java.sql.SQLException.
Una excepción que proporciona la información de un error durante el acceso a bases de datos u otros errores durante la conexión.

A partir del nombre del Driver capturado en la forma se instancia la clase del controlador a través de la instrucción:

```
Class.forName(JDBCDriver name);
```

Para el caso de PostgreSQL la instrucción sería la siguiente:

```
Class.forName("org.postgresql.Driver");
```

Con la información referente se construye la url hacia la base de datos y usando la clase *DriverManager* se obtiene una instancia de la Interface *Connection*. El siguiente fragmento de código ilustra esta acción:

```
url = CabeceraDelControlador + "//" + Servidor + ":" + Puerto
                + "/" + NombreDeLaBaseDeDatos;
connection = DriverManager.getConnection(url, usuario, password);
```

Un ejemplo de la construcción de una conexión a una base de datos PostgreSQL llamada m̈ineriaä través del usuario "nceron" con los valores por defecto para el Servidor y el Puerto sería la siguiente:

```
url = "jdbc:postgresql://localhost:5432/mineria";
connection = DriverManager.getConnection(url,"nceron","t3o4g2o");
```

La interface *Connection* obtenida al pulsar el botón Accept de la Clase *connectionWizard* devuelve una instancia de esta conexión al *DBConnectionIcon* que se almacena el la variable *connection*.

La siguiente opción en el menú desplegable de *DBConnectionIcon* es *Attribute Selection*. Esta alternativa genera un objeto de la clase *SelectorTable*. En la figura 8.21 se muestra la implementación de esta clase. Esta clase utiliza diferentes objetos del *API Swing* de Java para desplegar y solicitar al usuario información referente al conjunto de datos que quiere minar. Usa un *JComboBox* para listar las tablas que pertenecen a la bases de datos con la cual se estableció conexión, *JRadioButtons* para solicitar al usuario como debe ser considerado el conjunto de datos, como un conjunto multivaluado o bajo la metodología de canasta de mercado, un *JTextArea* para mostrar la sentencia SQL que se esta construyendo, un *JTable* que visualizará el contenido de la consulta que se logre construir y una instancia de la clase *MyCanvasTable*, que es una extensión de la clase *JPanel* y es utilizada para desplegar las tablas y relaciones que se establezcan para generar

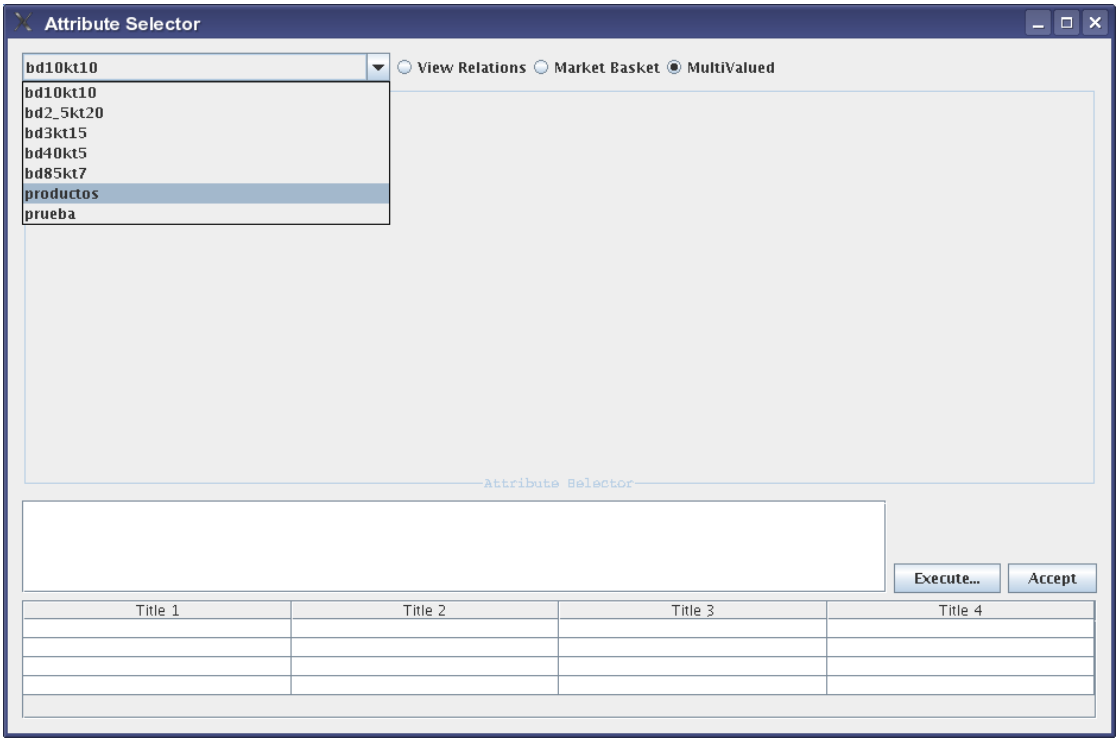

Figura 8.21: Implementación de la Clase DBConnectionIcon

una consulta de manera visual usando la metodología Drag'n Drop.

Durante el constructor de la clase se carga el JComboBox con los nombres de las tablas que contiene la base de datos conectada. Para obtener esta información se utiliza el método *getTables* de la interface *DatabaseMetaData*. Este método devuelve un *ResultSet* que se recorre para alimentar un Vector que es pasado como parámetro en la construcción del *JComboBox*. El siguiente fragmento de código ilustra lo anteriormente explicado.

A partir de las tablas que despliegue el *JComboBox* podemos escoger de esa lista la(s) tabla(s) que queremos relacionar en el análisis. La figura 8.22 muestra esta acción. Estas aparecerán dentro del área del *Attribute Selector*, una instancia de la Clase *MyCanvasTable*, donde podemos interactuar con las tablas de forma gráfica y establecer relaciones al tiempo que construimos la sentencia SQL para consultar la base de datos. Esta sentencia se construye al interior de la Clase *JTextArea* como se puede apreciar en la figura 8.23.

```
  public Vector getTables(){
  ResultSet rs;
  Vector names = new Vector();
  try{
  // Se instancia  DatabaseMetaData a partir de la
  // interfase connection.
  DatabaseMetaData dbmd = connection.getMetaData();
  String[] types = { "TABLE" };
  //Se captura los nombres de la tabla en un ResulSet
  //y se recorre alimentando el Vector names
  rs = dbmd.getTables("%", "%", "%", types);
  while(rs.next()){
    names.addElement(rs.getString(3));
  }
  rs.close();
  }catch(SQLException e){
   e.printStackTrace();
  }finally{
   return names;
  }
}
```

Código 8.2.34: Función *getTables*

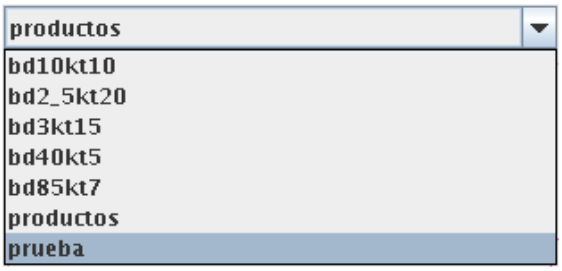

Figura 8.22: Tablas de la conexión organizadas en un JComboBox

Para la interacción de las tablas y las relaciones dentro de *MyCanvasTable* se crearon las Clases *Edge, ConectorTable, Attribute, Table. ConectorTable* es similar a la Clase *Conector* del paquete *GUI KnowledgeFlow* y es usado para establecer y marcar relaciones entre los atributos de las tablas involucradas. La Clase *Attribute* esta conformado por una instancia de *ConectorTable* más un *JLabel* con el nombre del atributo y la posibilidad de una imagen que indique su selección. La particularidad de la Clase *ConectorTable* es que dependiendo del tipo de atributo el desplegará una imagen diferente como conector si se trata de un atributo de tipo llave primaria, llave foránea o llave mixta. Un ejemplo de esta situación se puede apreciar en la figura 8.25.

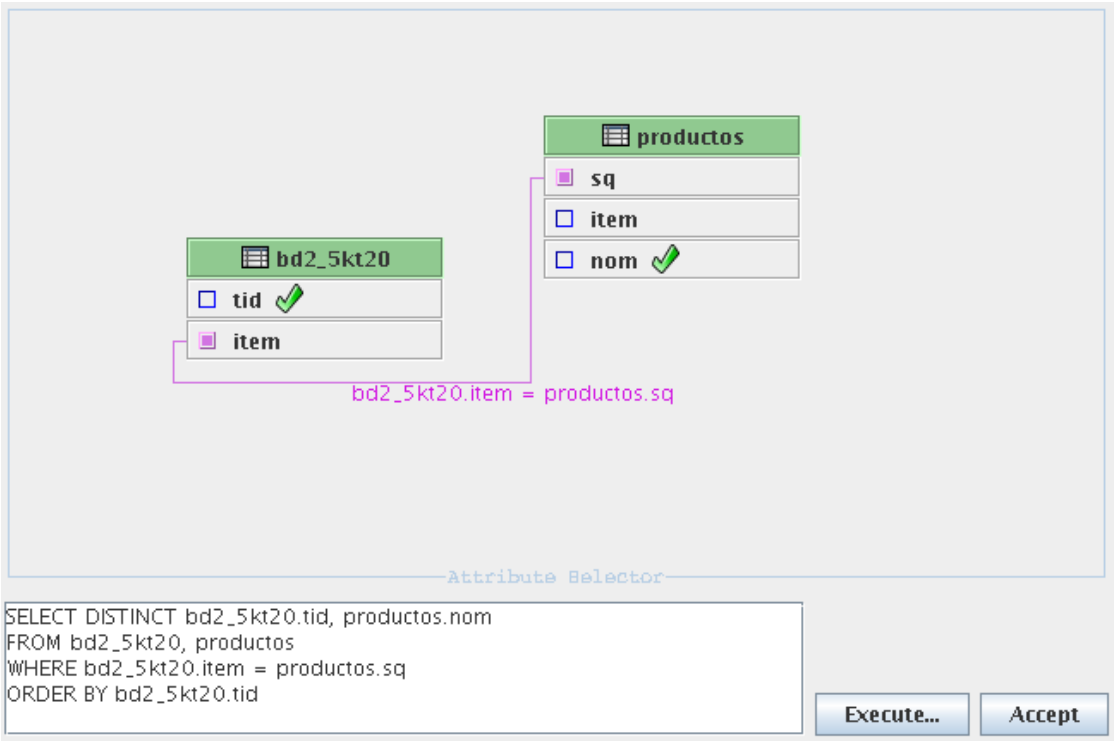

Figura 8.23: Implementación de la Clase MyCanvasTable

El conjunto de objetos *Attribute* conforman un objeto *Table* que puede estar conformado por un nmero n de atributos más un *JLabel* que sirva de título a la tabla. La implementación final de la Clase *Table* se puede apreciar en la figura 8.24.

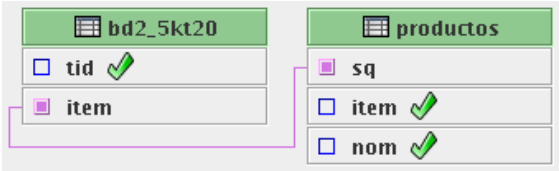

Figura 8.24: Implementación de la Clase Table

La Clase *Edge* cumple la función de guardar los objetos *ConectorTable* involucrados en una relación. De esta manera, la Clase *MyCanvasTable* guarda en un arreglo los objetos *Edges* que representan las relaciones establecidas hasta el momento de manera que al recorrerlo, pueda identificar los conectores involucrados, ubicarlos dentro de la forma y trazar líneas para representar las relaciones entre tablas. Este

procedimiento es similar al efectuado en la interfaz principal (Clase *MyCanvas*) con los diferentes tipos de íconos y sus relaciones. La figura 8.23 ilustra esta implementación.

La Clase *MyCanvasTable* se encarga también de monitorear los eventos del mouse para identificar clicks dentro de las tablas seleccionadas y marcarlos, seleccionar una tabla para su desplazamiento y marcar relaciones entre los atributos de las tablas.

Al medida que se marca un atributo de una determinada tabla, este se adicionará al *JTextArea* que construye la sentencia SQL dentro de su campo SELECT y se incluirá el nombre de la tabla de ese atributo dentro del campo FROM, de la misma forma, al establecer una relación se identifican las tablas relacionadas y se incluyen en el campo WHERE de la consulta. Un ejemplo de esta situación se muestra en la figura 8.25.

Cuando la selección de atributos ha finalizado se puede ver una preliminar de la consulta al pulsar el botón Execute. Aquí, se instancia un objeto de la Clase *ScrollableTableModel* al cual se pasa como parámetros el atributo connection de la clase *MyCanvasTable*, que representa la conexión a la base de datos, junto con una versión textual del contenido del *JTextArea*, que se constituye como la sentencia SQL que queremos consultar.

La Clase *ScrollableTableModel* se encarga de realizar las consultas a la base de datos usando las interfaces *Statament* y *ResulSet*. La interface *Statement* se crea a partir de la interface *Connection* que llega como parámetro en el constructor de la clase. Esta interface, a través de su método *executeQuery(String query)* dispara una consulta a la base de datos y retorna el resultado en una variable de tipo *ResulSet*. El query que se pasa como parámetro a este método es el que se construyónpaso como parámetro al constructor de esta clase. El cuadro de código 8.2.35 ilustra este procedimiento:

La Clase *ScrollableTableModel* es la encargada de recorrer el *ResulSet* que contiene el resultado de la consulta y construir un modelo de tabla que es pasado al *JTable* de la Clase *MyCanvasTable* para visualizar los resultados de la consulta como se puede apreciar en la figura 8.26.

Una vez validados los datos que se quiere minar se confirma la aceptación al pulsar el botón Accept. En ese momento devolvería la interfaz principal y el ultimo

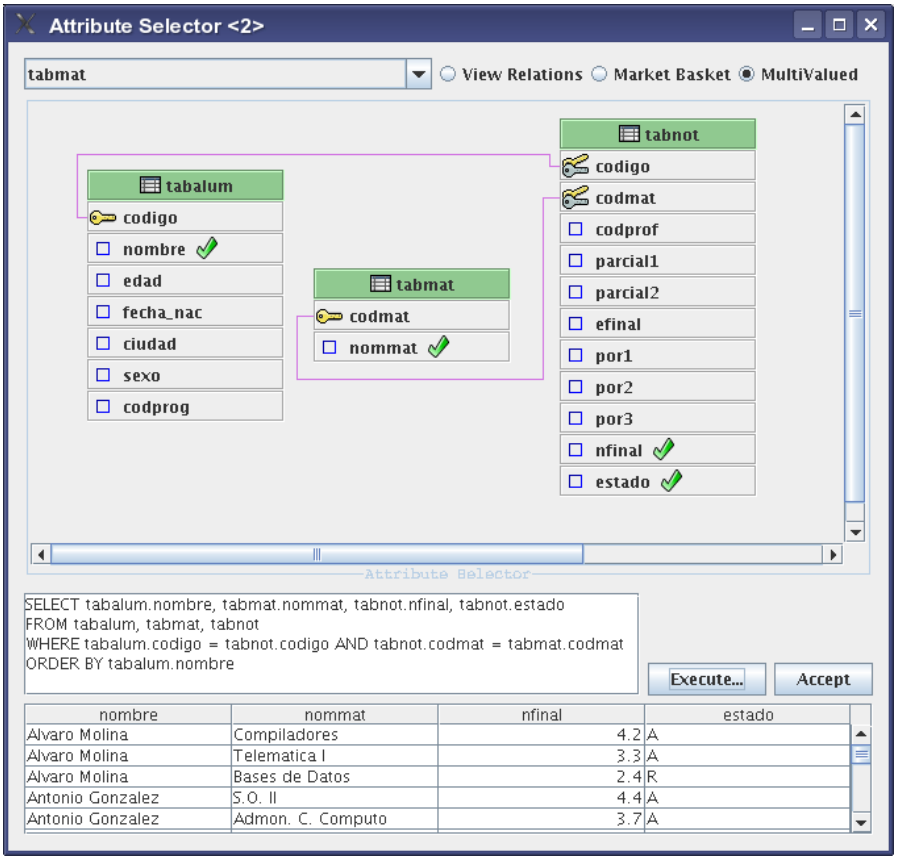

Figura 8.25: Construcción de la Sentencia SQL

paso para construir el conjunto de datos se realiza a través de la opción Start *Loading* del men contextual de la clase *DBConnectionIcon* donde, segn sea el caso, se invocaría lo métodos *loadMarketBasketDataSet()*, para cargar una instancia de la *DataSet* que cubre el modelo de canasta de marcado (Tablas Univaluadas), o *loadMultivaluedDataSet()*, para cargar instancias de *DataSet* de conjuntos multivaluados. Esta instancia de la Clase *DataSet* reposará como atributo de la Clase *DBConnectionIcon* para ser pasada en el momento de una conexión a instancias de las Clases *FilterIcon o AssociationIcon*.

Paquete File

A través de este paquete el usuario puede establecer una conexión con un Archivo Plano, para de esta forma obtener el conjunto de datos. El paquete File hace parte del paquete Icons, que a su vez hace parte del paquete GUI. Las clases que

```
Statement statement;
ResultSet resulset;
try {
//Crea una objeto Statement cuyos resultados
//seran sensibles al scroll y de solo lectura
statement = connection.createStatement(
      ResultSet.TYPE_SCROLL_SENSITIVE,
      ResultSet.CONCUR_READ_ONLY);
//Ejecuta la sentencia SQL contenida en query
resultset = statement.executeQuery(query);
} catch (SQLException e) {
throw new SQLException(e.getMessage(), e);
}
```

Código 8.2.35: Función *executeQuery(String query)*

nommat	nombre	nfinal	estado
Compiladores	Alvaro Molina	4.2	A
Telematica I	Alvaro Molina	3.3	A
Bases de Datos	Alvaro Molina	2.4	R
S.O. II	Antonio Gonzalez	4.4	A
Admon. C. Computo	Antonio Gonzalez	3.7	A

Figura 8.26: Construcción de la Previsualización de datos en un JTable

conforman este paquete son: *FileIcon, OpenFile y FileTableModel*.

En la gráfica 8.27 se pueden ver las funciones que esta clase implementa.

FileIcon extiende a la clase *Icon* que hace parte del paquete *KnowledgeFlow* y a su vez del paquete *GUI*, por tanto automáticamente hereda su menú con la opción por defecto *delete*, la cual permite al usuario eliminar el icono de conexión al Archivo Plano. Por otra parte, la clase *FileIcon* añade al menú de Icon las opciones Open y Load.

La opción **Open** es la encargada de abrir una ventana en la cual el usuario elige el conjunto de datos que va a minar, esta opción es manejada a través de la clase *OpenFile*. La clase *OpenFile* muestra el conjunto de datos a través de una JTable.

La clase de *Swing, JFileChooser* mediante su método *getAbsolutePath*, retorna

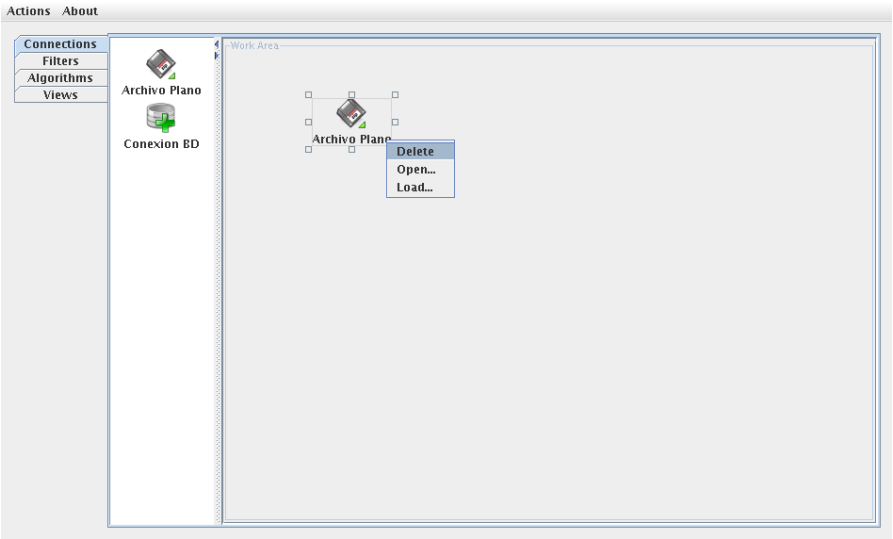

Figura 8.27: Funciones de *FileIcon*

como una cadena de texto la ruta completa en donde se encuentra el archivo.

La cadena de texto obtenida pasa como parámetro al constructor de la clase *FileTableModel*, la cual se encarga de construir una JTable a través de la cual se muestran los datos del archivo plano, tal y como se puede observar en la figura 8.28.

A través de la clase *FileTableModel* se construye un modelo propio para la JTable que indicara los datos. Construir un modelo para una JTable es suministrarle la información concerniente al nombre y número de las columnas y las filas, en el caso de las filas se deben suministrar los datos.

Para construir un modelo propio para JTable, la clase *FileTableModel*, debe extender a *AbstractTableModel*, quién implementa los métodos necesarios para la construcción de una JTable, se debe también implementar los métodos *getColumnName(int col), getRowCount(), getColumnCount() y getValueAt(int rowIndex, int colIndex)* de acuerdo al tipo de estructura que almacena los datos, entonces, a través de la cadena de texto que indica la ruta completa en donde se encuentra el archivo plano, se abre un flujo con tal archivo y mediante el método *dataAndAtributes* de la clase *FileManager*, encargada de administrar todo lo referente a Archivos Planos, se obtiene la siguiente información del archivo seleccionado por el usuario: en un array de objetos retorna los nombres de las columnas y en una

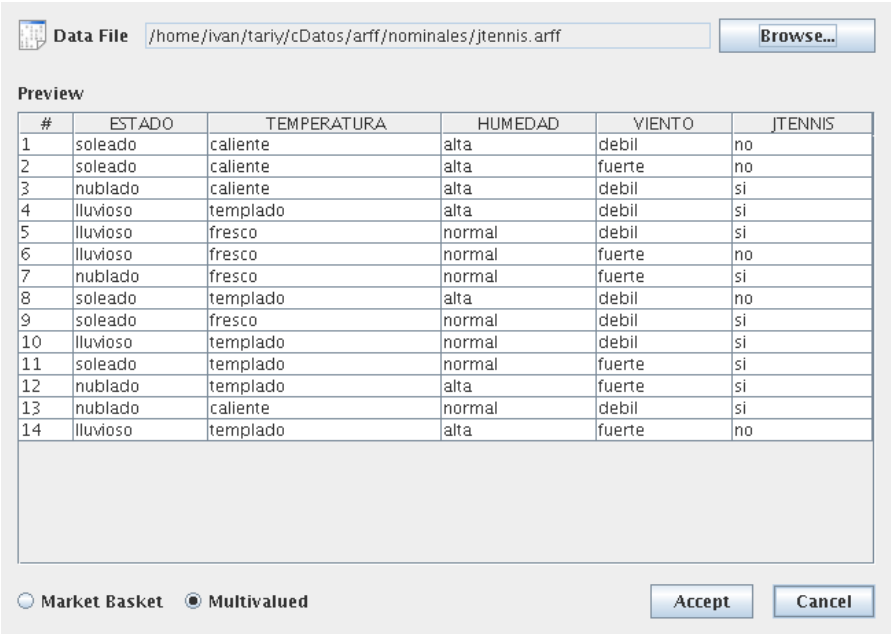

Figura 8.28: Visualización del conjunto de datos

matriz de objetos los datos en si. De esta forma y teniendo los métodos antes mencionados debidamente implementados, los datos se mostrarán en una JTable de acuerdo al modelo proporcionado. Para ilustrar mejor la forma de construir un modelo de datos propio se puede observar en el código fuente 8.2.36, la clase *FileTableModel*.

```
public class FileTableModel extends AbstractTableModel {
  //La ruta del archivo de acceso aleatorio de tipo .arff
  private String filePath;
  //Los nombres de las columnas
  Object[] columnNames;
  //Los datos provenientes de un archivo de acceso aleatorio .arff
  Object [][] data;
  //Creates a new instance of FileTableModel
  public FileTableModel(String file) {
    filePath = file;
    FileManager fileMngt = new FileManager(filePath);
    fileMngt.dataAndAttributes(true);
    int size = fileMngt.getAttributes().length;
    columnNames = new Object[size];
    columnNames = fileMngt.getAttributes();
    int rows = fileMngt.getData().length;
    int cols = fileMngt.getData()[0].length;
    data = new Object[rows][cols+1];
    data = fileMngt.getData();
  }
  public String getColumnName(int column) {
    if (column==0) {
      return "#";
    } else {
        if (columnNames[column-1] != null) {
          return (String) columnNames[column-1];
        } else return "";
    }
  }
  //|||||||||||||||| AbstractTableModel implemented methods
  public int getRowCount() {
    return data.length;
  }
  public int getColumnCount() {
    return columnNames.length+1;
  }
  public Object getValueAt(int rowIndex, int columnIndex) {
    if (columnIndex==0) return rowIndex+1;
    else return data[rowIndex][columnIndex-1];
  }
}
```

Código 8.2.36: Clase *FileTableModel*

La opción **Load** se encarga de, a partir del flujo de comunicación abierto con un archivo plano, construir un DataSet o estructura en forma de árbol N-Ario para

almacenar los datos de manera comprimida.

8.2.5. Paquete Filtros

El módulo *filtros o data cleaning*, se encarga de hacer un refinamiento de los datos en dos etapas, por un lado hace un proceso de limpieza sobre datos corruptos, vacios, ruidosos, inconsistentes, duplicados, alterados etc, por otro lado hace una selección de estos datos para escoger aquellos que brinden información de calidad, aplicando distintas técnicas como son muestreos, discretizaciones y otras. De esta forma obtenemos datos depurados según el objetivo del analista, para que posteriormente se pueda aplicar el núcleo *KDD o de Minería de Datos* sobre datos coherentes, limpios y consistentes.

A este módulo, pertenecen los filtros: *Remove Missing, Update Missing, Selection, Range, Reduction, Codification, Replace Value, Numeric Range y Discretize*, los cuales extienden la clase *AbstractTableModel de Java*, con el objetivo de alimentarse y presentar sus resultados a través de la clase denominada *TableModel*, que es el medio por el cual comunican sus flujos de datos, es así como los filtros pueden recibir los datos de entrada de otros filtros o de la conexión directa de a las bases de datos, pero siempre utilizando la clase *TableModel*. De la misma forma los datos de salida en los filtros, se enviaran utilizando el mismoFiltro RemoveMissing formato.

Las clases que se encargan de Mostrar resultados y de hacer la configuración de los filtros, son todos los moodulos *Show y Open*, respectivamente, estos extienden la clase de *Java* denominada *javax.swing.JFrame*, ya que presentan una iterfaz a modo de ventana que permite mantener un dialogo constante con el analista.

Filtro RemoveMissing

El objetivo especifico de este filtro, es eliminar todas las transacciones que contengan campos vacios. El filtro *RemoveMissing* consta de 2 Clases, las cuales son: *RemoveMissing y ShowRemoveMissing*.

RemoveMissing

RemoveMissing es el núcleo principal del filtro *RemoveMissing*, y se encarga de eliminar todas las transacciones que contengan datos nulos o vacios, haciendo una búsqueda de los mismos en toda la tabla de la base de datos seleccionada, y creando un nuevo conjunto de datos a partir de las transacciones completas. A continuación se presenta el codigo 8.2.37, de la función encargada de hacer la transformación.

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
fv = 0
  for( f = 0 ;  f < Rows;  f = f+1) {
    for( c =  0;  c < Columns;  c = c+1 ) {
      if( Atributo(f,c)  == "NULL O VACIO") {
        if( f ? Rows -1) {
          for( i  = 0;  i < Columns;  i = i +  1) {
            datosSalida(fv,i) = datosEntrada(f+1,i)
            datosEntrada(f, i) = Null
          }
        }
        else
          for(  i = 0;  i < Columns;  i = i + 1 ) {
            datosSalida(f, i) = Null
          }
      }
      c = Columns
    }
    if( c == Columns -1) {
      fv = fv  + 1
        for( i = 0;  i <  Columns;  i = i + 1 ) {
          datosSalida(f, i) = Null
        }
    }
  }
```

Código 8.2.37: Pseudo Codigo del Filtro RemoveMissing

ShowRemoveMissing

ShowRemoveMissing Es la clase que se encarga, tanto de mostrar los tipos de variables contenidos en la tabla, como de mostrar los datos de entrad y de salida, reportando los registros eliminados de la tabla original y los registros actuales del nuevo conjunto de datos. Ejemplo de funcionamiento del filtro *RemoveMissing*. A continuación se presenta la tabla con los datos de entrada, ates de ser aplicado el filtro *RemoveMissing*. En la tabla se puede apreciar que hay 17 atributos va-

cios, en las transacciones 2,4,6,7,8,9,10,11,12,14, como se muestra en la figura 8.29.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto		No	
8	Alta	Bajo	Bajo		Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo		Si	No
12	Alta		Alto		Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 8.29: Datos de entrada antes de aplicar RemoveMissing

Al aplicar el filtro *RemoveMissing*, se eliminaran las trasacciones que contengan atributos vacios, las cuales son las siguientes: 2,4,6,7,8,9,10,11,12,14, dejando solo las trasacciones 1,3,5 y 13. La tabla resultante o la de los datos de Salida después de haber aplicado el filtro *RemoveMissing*, se muestra en la grafica 8.30.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
3	Baja	Alto	Bajo	No	Si	No
5	Media	Bajo	Alto	Si	Si	Si
13	Baja	Alto	Alto	Si	Si	No

Figura 8.30: Datos de Salida despues de aplicar RemoveMissing

Filtro UpdateMissing

El objetivo especifico de este filtro, es remplazar los campos vacios de un atributo especifico, por un valor otorgado por el analista. El filtro *UpdateMissing* consta de 3 clases las cuales son *OpenUpdateMissing, ShowUpdate Missing* y el núcleo principal la clase *UpdateMissing*.

OpenUpdateMissing

OpenUpdateMissing extiende a la clase de *Java: javax.swing.JFramees. OpenUpdateMissing* es la clase encargada de configurar el filtro según las necesidades del

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
colRem  =  valor num\'erico de la columana del atributo seleccionado
valRem =  Valor a remplazar
for( f = 0;  f < Rows; f = f + 1 ) {
  if( datosEntrada(f,colRem ) == VACIO ) {
    datosSalida(f,colRem) = valRem
  }
}
```

Código 8.2.38: Pseudo Codigo del Filtro UpdateMissing

objetivo del analista, preguntando por el atributo a seleccionar, en el cual tendrán efecto los cambios que aplique el filtro según el valor a remplazar.

UpdateMissing
UpdateMissing es la principal clase del filtro *UpdateMissing*, ya que se encarga de remplazar los campos vacios de un atributo, con un valor otorgado por el analista, creando de este modo un nuevo conjunto de datos, libre de campos vacios y con el mismo numero de transacciones originales.
Se presenta el codigo 8.2.38 de la función encargada de hacer esta transformación.

ShowUpdateMissing
ShowUpdateMissing es la clase del filtro *UpdateMissing* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada y su respectiva transformación, reportando los registros que fueron remplazados de la tabla original y los registros actuales del nuevo conjunto de datos. Ejemplo de funcionamiento del filtro *UpdateMissing*:

A continuación se presenta la tabla con los datos de entrada, ates de ser aplicado el filtro *UpdateMissing*. En la tabla se puede observar que en el atributo *"ALERGIA_ANTIBIOTICO"* existen 4 campos vacios en las transacciones 7,8,11,12 como se muestra en la figura 8.31.

Al aplicar el filtro *UpdateMissing*, se remplaza en el atributo *"ALERGIA_ANTIBIOTICO"*, todos los campos vacios, por el valor "JC"La tabla resultante o la de los datos de Salida después de haber aplicado el filtro *UpdateMissing*, quedaría como se muestra en la figura 8.32.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto		No	
8	Alta	Bajo	Bajo		Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo		Si	No
12	Alta		Alto		Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 8.31: Datos de Entrada antes de aplicar UpdateMissing

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	No
2	Alta	Alto		Si	No	
3	Baja	Alto	Bajo	No	Si	No
4		Alto		No		
5	Media	Bajo	Alto	Si	Si	Si
6	Baja	Bajo	Alto	Si	Si	
7	Alta	Bajo	Alto	JC	No	
8	Alta	Bajo	Bajo	JC	Si	No
9	Alta	Alto	Bajo	Si		Si
10	Baja	Bajo	Alto	Si		
11	Media	Bajo	Bajo	JC	Si	No
12	Alta		Alto	JC	Si	Si
13	Baja	Alto	Alto	Si	Si	No
14	Baja	Alto		No	No	Si

Figura 8.32: Datos de Salida despues de aplicar UpdateMissing

Filtro Selection

.

El objetivo especifico de este filtro, es hacer una selección de atributos y del atributo objetivo sobre un conjunto de entrada. El filtro *Selection* consta de 3 clases las cuales son *OpenSelection, ShowSelection* y el núcleo principal *Selection.*

OpenSelection

OpenSelection es la clase encargada de configurar este filtro, con el objetivo de escoger las columnas que el analista desee vincular al proceso minero. En el caso específico de minar con un algoritmo de clasificación, el analista deberá escoger la

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
colsel[ ] = Array de columnas seleccionadas
colObje = Columna Objetivo
for( f == 0;  f < Rows;  f = f + 1 ) {
  for( c = 0;  c <  Columns;  c = c + 1 ) {
    if( c == Columns -1) datosSalida[f][c] = datosEntrada(f,colObje)
      else datosSalida[f][c] = datosEntrada(f,colsel[c]);
  }
}
```

Código 8.2.39: Pseudo Codigo del Filtro Selection

columna objetivo.

Selection
Selection es la principal clase del filtro *Selection*, y se encarga de efectuar la selección sobre el conjunto de datos original, creando un nuevo conjunto de datos, con los atributos escogidos y dejando el atributo objetivo al final de la tabla, para el posterior proceso minero.
Se presenta el código 8.2.39, de la función encargada de hacer la selección.

ShowSelection
ShowSelection es la clase del filtro *Selection* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada con el número de atributos en su forma original. También se encarga de mostrar el nuevo conjunto de datos, el cual solo tiene los atributos seleccionados, y al final de la tabla el atributo objetivo. Este modulo también brinda información sobre el numero de atributos que se eliminaron y el numero de atributos que permanecen en el conjunto de datos.
Ejemplo de funcionamiento del filtro Selection:

A continuación se presenta la tabla con los datos de entrada, ates de aplicar el filtro *Selection*, en la figura 8.34. En la tabla original se puede observar 6 Atributos los cuales son: *"PRESION_ARTERIAL, AZUCAR_SANGRE, INDICE_COLESTEROL, ALERGIA_ANTIBIOTICO, OTRAS_ALERGIAS, ADMINISTRAR_FARMACO"*.
Al aplicar el filtro *Selection*, se puede observar que la tabla resultante, solo contiene los atributos sobre los cuales hicimos la selección, los cuales son: *AZU-*

PRESION_ARTERIAL	AZUCAR_SANGRE	INDICE_COLESTER...	ALERGIA_ANTIBIOTI...	OTRAS_ALERGIAS	ADMINISTRAR_FAR...
Alta	Alto	Alto	No	No	Si
Alta	Alto	Alto	Si	No	Si
Baja	Alto	Bajo	No	No	Si
Media	Alto	Alto	No	Si	No
Media	Bajo	Alto	Si	Si	No
Baja	Bajo	Alto	Si	Si	Si
Alta	Bajo	Alto	Si	No	Si
Alta	Bajo	Bajo	No	Si	Si
Alta	Alto	Bajo	Si	Si	No
Baja	Bajo	Alto	Si	Si	Si
Media	Bajo	Bajo	Si	Si	Si
Alta	Bajo	Alto	Si	Si	No
Baja	Alto	Alto	Si	Si	Si
Baja	Alto	Bajo	No	No	Si

Figura 8.33: Datos de Entrada antes de aplicar Selection

CAR_SANGRE, ALERGIA_ANTIBIOTICO y al final de la tabla como atributo objetivo *ADMINISTRAR_FARMACO.*

AZUCAR_SANGRE	ALERGIA_ANTIBIOTICO	ADMINISTRAR_FARMACO_F
Alto	No	Si
Alto	Si	Si
Alto	No	Si
Alto	No	No
Bajo	Si	No
Bajo	Si	Si
Bajo	Si	Si
Bajo	No	Si
Alto	Si	No
Bajo	Si	Si
Bajo	Si	Si
Bajo	Si	No
Alto	Si	Si
Alto	No	Si

Figura 8.34: Datos de Salida despues de aplicar Selection

Filtro Range

El objetivo principal de este filtro, es escoger una muestra sobre un conjunto de entrada, especialmente útil para minería con algoritmos de clasificación.
El filtro *Range* consta de 3 clases las cuales son *OpenRange, ShowRange* y el núcleo principal, la clase *Range.*

```
r  = Numero generado aleatoriamente
Rows = Numero de Transacciones
Columns = Numero de Atributos
valmues = Tama\~no de la Muestra
for( f = 0; f < valmues; f = f + 1) {
  cfila = Numero aleatorio comprendido entre el numero de transacciones
  for( c = 0;  c < Columns;  c = c + 1) {
    datosSalida(f,c) = datosEntrada(cfila,c)
  }
}
for( f = valmues; f < Rows; f = f + 1) {
  for( c = 0;  c < Columns; c = c + 1 ) {
    datosSalida(f,c) = null
  }
}
```

Código 8.2.40: Pseudo Codigo de la Tecnica de Aleatorios

OpenRange
OpenRange es la clase encargada de configurar este filtro, brindando tres alternativas de muestreo, los cuales son: *muestreo aleatorio, de 1 en n y los primeros n.*

Range
Range es la principal clase del filtro *Range*, y se encarga de hacer un muestreo sobre los datos de entrada, utilizando distintas técnicas. Es muy útil en clasificación, ya que este tipo de minería trabaja a partir de un conjunto de entrenamiento, que lo podemos construir con este tipo de filtro. A continuación se presenta el pseudo código y las técnicas encargadas de hacer el muestreo.

Tecnica de Aleatorios:
Esta técnica se encarga de seleccionar una muestra del conjunto de entrada aleatoriamente, a partir de una semilla y la función *Random de Java.*
La estructura de esta funcion se presenta en el codigo 8.2.40.

Tecnica de 1 en n:
Esta técnica se encarga de seleccionar una muestra, que tomara cada transacción en saltos de n en n, n es el valor de salto otorgado por el analista.
La estructura de esta funcion se presenta en el codigo 8.2.41.

```
 filn = 0;
 valmues = Salto de Muestra
 Rows = Numero de Transacciones
 Columns = Numero de Atributos
 for( f = 0; f < Rows; f = f + 1) {
   if( f == filn) {
     for( c = 0; c < Columns; c = c + 1 ) {
       datosSalida(fp,c) = datosEntrada.getValueAt(filn,c)
     }
     filn = filn + valmues
     fp = fp + 1
   }
}
```

Código 8.2.41: Pseudo Codigo de la Tecnica de de 1 en n

```
 valmues = Tama\~no de la Muestra
   for( f = valmues; f < Rows; f = f + 1) {
     for( c = 0;  c < Columns; c = c + 1) {
       datosSalida(f,c) = null
     }
   }
```

Código 8.2.42: Pseudo Codigo de la Tecnica de Primeros n

Tecnica de Primeros n:
Esta técnica se encarga de seleccionar una muestra, a partir de las primerar n transacciones del conjunto de datos de entrada.
La estructura de esta funcion se presenta en el codigo 8.2.42.
ShowRange
ShowRange es la clase del filtro *Range* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada con el número de Transacciones original. También se encarga de mostrar el nuevo conjunto de datos, con las transacciones elegidas por las técnicas de muestreo. Este modulo también brinda información sobre el numero de Transacciones que se eliminaron y el numero de Transacciones que permanecen en el conjunto de datos final.

Ejemplo de funcionamiento del filtro *Range*:

Se presenta la tabla con los datos de entrada en la figura 8.35, ates de aplicar el

filtro *Range*. En la tabla original se puede observar 14 Transacciones.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
2	Alta	Alto	Alto	Si	No	Si
3	Baja	Alto	Bajo	No	No	Si
4	Media	Alto	Alto	No	Si	No
5	Media	Bajo	Alto	Si	Si	No
6	Baja	Bajo	Alto	Si	Si	Si
7	Alta	Bajo	Alto	Si	No	Si
8	Alta	Bajo	Bajo	No	Si	Si
9	Alta	Alto	Bajo	Si	Si	No
10	Baja	Bajo	Alto	Si	Si	Si
11	Media	Bajo	Bajo	Si	Si	Si
12	Alta	Bajo	Alto	Si	Si	No
13	Baja	Alto	Alto	Si	Si	Si
14	Baja	Alto	Bajo	No	No	Si

Figura 8.35: Datos de Entrada antes de aplicar las Tecnicas de Muestreo

En primera instancia aplicamos la técnica de *Aleatorios*, escogiendo un tamaño de muestra de 5 transacciones. Podemos observar que las transacciones elegidas por este método son 5, reduciendo el cojunto original en 9 transacciones. Como se muestra en la figura 8.36.

	PRESION_ARTERIAL	AZUCAR_SANGRE	INDICE_COLESTER...	ALERGIA_ANTIBIOTI...	OTRAS_ALERGIAS	ADMINISTRAR_FAR...
2	Alta	Alto	Alto	Si	No	Si
10	Baja	Bajo	Alto	Si	Si	Si
12	Alta	Bajo	Alto	Si	Si	No
11	Media	Bajo	Bajo	Si	Si	Si
5	Media	Bajo	Alto	Si	Si	No

Figura 8.36: Datos de Salida despues de aplicar la Tecnica de Aleatorios

Ahora aplicamos la técnica de *1 en n*, escogiendo como salto de muestra el valor 3. Podemos observar que las transacciones elegidas por este método se eligien a partir de la primera transacción en saltos de 3 en 3, reduciendo el cojunto original. Como se muestra en la figura 8.37.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
4	Media	Alto	Alto	No	Si	No
7	Alta	Bajo	Alto	Si	No	Si
10	Baja	Bajo	Alto	Si	Si	Si
13	Baja	Alto	Alto	Si	Si	Si

Figura 8.37: Datos de Salida despues de aplicar la Tecnica de 1 en n

Ahora aplicamos la técnica de *Primeros n*, escogiendo como Tamaño de muestra el valor 7. Podemos observar que las transacciones elegidas por este método son las 7 primeras transacciones del conjunto de entrada, eliminando de esta forma las 7 siguientes. Como se muestra en la figura 8.38.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	Si
2	Alta	Alto	Alto	Si	No	Si
3	Baja	Alto	Bajo	No	No	Si
4	Media	Alto	Alto	No	Si	No
5	Media	Bajo	Alto	Si	Si	No
6	Baja	Bajo	Alto	Si	Si	Si
7	Alta	Bajo	Alto	Si	No	Si

Figura 8.38: Datos de Salida despues de aplicar la Tecnica de primeros n

Filtro Reduction

El objetivo específico de este filtro, es hacer una reducción en el número de transacciones, manteniéndolas o eliminándolas, con diferentes técnicas y parámetros de selección. El filtro Reduction consta de 3 clases las cuales son *OpenReduction, ShowReduction* y el núcleo principal *Reduction*.

OpenReduction

OpenReduction es la clase encargada de configurar este filtro, teniendo en cuenta distintas técnicas como son: por *rango y por valor* que a su vez selecciona las transacciones por *atributo numérico o alfabético*, además también se tiene en cuenta los parámetros de mantener o eliminar las transacciones seleccionadas.

Reduction

Reduction es la principal clase del filtro *Reduction*, y se encarga aplicar las técnicas de reducción sobre el conjunto de datos original, creando un nuevo conjunto de datos, con las transacciones elegidas, bien sea manteniéndolas o eliminándolas dependiendo del objetivo del analista. A continuación se presenta el pseudo código de las distintas técnicas de reducción que aplica este filtro.

Tecnica de Reduccion por Rango:

Esta técnica se encarga de reducir el conjunto de entrada, en un rango de transacciones, a partir de una transacción inicial y otra final, de acuerdo al parámetro escogido por el analista, el cual puede ser eliminar o mantener dichas transacciones.
Se presenta la estructura de este procedimiento en el codigo 8.2.43.

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
filIni = Transaccion Limitrofe Inferior
filFin = Transaccion Limitrofe Superior
if( Mantener las transacciones) {
  numfil = (filFin - filIni) + 1;
  pf = 0;
  for( f = filIni;  f < filFin + 1;  f = f + 1) {
     for( c = 0;  c < Rows; c = c + 1 ) {
       datosSalida(pf,c) = datosEntrada(f,c)
     }
     pf = f + 1;
  }
  for(f = pf; f < Rows; f = f + 1) {
    for(c = 0; c < Columns c = c + 1 ) {
      datosEntrada(f,c) = null;
    }
  }
}
else if( Remover las Transacciones) {
  numfil =  Rows -((filFin - filIni)+1);
  dfi = filIni;
  for( f = filFin + 1;  f < Rows; f = f + 1) {
    for( c = 0; c < Columns; c = c + 1) {
      datosSalida(dfi,c) = datosEntrada(f,c)
    }
    dfi ++
  }
  for(f = dfi + 1; f < Rows; f = f + 1) {
    for( c = 0; c < Columns; c = c + 1) {
      datosEntrada(f,c) = null
    }
  }
}
```

Código 8.2.43: Pseudo Codigo de la Tecnica de Reduccion por Rango

Tecnica de Reduccion de Transacciones por Atributo:
Esta técnica se encarga de reducir el conjunto de entrada, dependiendo del tipo de datos que contenga el atributo seleccionado, los cuales pueden ser alfabéticos o numéricos, si son numéricos se debe suministrar un valor limítrofe y si son alfabéticos el analista deberá seleccionar los atributos de su interés. Además de acuerdo al parámetro escogido por el analista, el filtro eliminara o mantendrá las transacciones seleccionadas.
Se presenta la estructura de este procedimiento en el codigo 8.2.44.

ShowReduction
ShowReduction es la clase del filtro *Range* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada con el número de transacciones original. También se encarga de mostrar el nuevo conjunto de datos, con las transacciones que permanecieron después de aplicar la reducción. Este modulo también brinda información sobre el numero de Transacciones que se eliminaron y el numero de Transacciones que permanecen en el conjunto de datos final.

Ejemplo de funcionamiento del filtro Reduction: Se presenta la tabla con los datos

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
numatrisel = Numero de Atributos Seleccionados
String valsAtri[ ] = Array de los valores de los Atributos
colsel = Columna Seleccionada
if(Valor Numerico) {
  numfil = 0;
  menores = Valor lim\'itrofe Superior
  for( f = 0; f < Rows; f = f + 1) {
    if(Mantener las transacciones) {
      if( datosEntrada(f,colsel) < menores) {
        for( c = 0; c < Columns; c = c + 1) {
          datosSalida(numfil,c) = datosEntrada(f,c)
        }
        numfil ++;
      }
    }
    else if( Remover las transacciones) {
      if(datosEntrada(f,colsel) >= menores) {
        for( c = 0; c < Columns; c = c + 1) {
          datosSalida(num\textbf{fil,c) = datosEntrada(f,c)
        }
        numfil ++;
      }
    }
  }
  for(int f = numfil; f < Rows; f = f + 1ShowReduction ) {
    for(int c = 0; c < Columns; c = c + 1) {
      datosEntrada(f,c) = null;
    }
  }
}
else if(Valor es Alfab\'etico) {
  valsAtri[ ] = Valores de los Atributos
  if(Mantener Transacciones) {
    for(int f = 0; f < Rows; f = f + 1) {
      filen = 0
      for(i i = 0; i < numatrisel; i++) {
        if(datosEntrada(f,colsel) = (valsAtri[i])) {
          filen ++;
        }
      }
      if(filen != 0) {
        for(int c = 0; c < Columns; c = c + 1) {
          datosSalida(numfil,c) = datosEntrada(f,c)
        }
        numfil ++;
      }
    }
   }
   else if(Remover Transacciones) {
     for( f = 0; f < Rows; f = f + 1) {
       filen = 0
       for( i = 0; i < numatrisel; i++) {
         if(datosEntrada(f,colsel) = (valsAtri[i])) {
           filen ++
         }
       }
       if(filen == 0) {
         for( c = 0; c < Columns; c = c + 1) {
           datosSalida(numfil,c) = datosEntrada(f,c)
         }
         numfil ++;
       }
     }
   }
   for( f = numfil; f < Rows; f = f + 1) {
     for( c = 0; c < Columns; c = c + 1) {
       datosEntrada(f,c) = null
     }
   }//////////
}
```

Código 8.2.44: Pseudo Codigo de la Tecnica de Reduccion por Atributo

de entrada, ates de aplicar el filtro Reduction, en la figura 8.39. En la tabla original se puede observar 14 Transacciones.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 8.39: Datos de Entrada antes de aplicar las Tecnicas de Reducción

En primera instancia aplicamos la técnica de *Reducción por Rango*, y como parámetro de selección, *mantener el rango*. Escogemos la transacción 5 como limite inicial y la transacción 10 como limite final, lo cual significa que se mantendrá el rango a partir de la transacción 5 (sin ser incluida), hasta la transacción 10 (siendo incluida).
Se observar que las transacciones elegidas por este método son 5, las cuales son: 6,7,8,9 y 10, reduciendo el conjunto original en 9 transacciones, como se muestra en la figura 8.40.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8

Figura 8.40: Reducción por rango, Manteniendo los datos

Ahora aplicamos la misma técnica *de Reducción por rango*, pero como parámetro de selección, removemos dicho rango. Escogemos la transacción 5 como limite inicial y la transacción 10 como limite final, lo cual significa que se removerá el rango comprendido a partir de la transacción 5 (sin ser incluida), hasta la transacción 10 (siendo incluida), como se muestra en la figura 8.41.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 8.41: Reducción por rango, Removiendo los datos

Apliquemos ahora la *Técnica de Reducción de Transacciones por Atributo*, con valores Alfabéticos, y escojamos como atributo de reducción a *"INDICE_COLESTEROL"*, y como valor de este atributo a *"ALTO"*. También escogemos como parámetro de selección, *Mantener el conjunto*. Esto significa que el filtro buscara cualquier valor diferente a *"ALTO"* en este atributo, y eliminara su transacción correspondiente. Podemos observar que las transacciones elegidas por este método son las siguientes: 1,2,4,5,6,7,10,12 y 13, eliminando de esta forma 5 transacciones, como se muestra en la figura 8.42.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
10	Baja	Bajo	Alto	Si	Si	8
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1

Figura 8.42: Reducción por atributo Alfabetico, Manteniendo los datos

Ahora aplicamos la misma Técnica de Reducción de Transacciones por Atributo, con valores Alfabéticos, y escogemos como atributo de reducción a *"INDICE_COLESTEROL"*, y como valor de este atributo a *"ALTO"*. Pero ahora escogemos como parámetro de selección, *Remover el conjunto*. Esto significa que el filtro buscara los valores del atributo iguales a *"ALTO"* y eliminara su transacción correspondiente. Podemos observar que las transacciones elegidas por este método son las siguientes: 3,8,9,11 y 14, eliminando de esta forma 9 transacciones, como se muestra en la figura 8.43.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
3	Baja	Alto	Bajo	No	No	9
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
11	Media	Bajo	Bajo	Si	Si	3
14	Baja	Alto	Bajo	No	No	7

Figura 8.43: Reducción por atributo Alfabetico Removiendo los datos

Apliquemos ahora la *Técnica de Reducción de Transacciones por Atributo*, con valores Numéricos, y escogemos como atributo de reducción a *"ADMINISTRAR_FARMACO"*, y como limite restrictivo superior, al valor 5.
También escogemos como parámetro de selección, *Mantener el conjunto*. Esto significa que el filtro eliminara todas las transacciones, con valores superiores o iguales a 5 en este atributo. Podemos observar que las transacciones elegidas por este método son las siguientes: 1,2,5,7,11 y 13, eliminando de esta forma 8 transacciones, como se muestra en la figura 8.44.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
5	Media	Bajo	Alto	Si	Si	3
7	Alta	Bajo	Alto	Si	No	2
11	Media	Bajo	Bajo	Si	Si	3
13	Baja	Alto	Alto	Si	Si	1

Figura 8.44: Reducción por atributo Numerico Manteniendo los datos

Ahora aplicamos la misma Técnica de *Reducción de Transacciones por Atributo, con valores Numéricos*, y escogemos como atributo de reducción a *"ADMINISTRAR_FARMACO"*, y como limite restrictivo superior, al valor 5. Pero ahora escogemos como parámetro de selección, *Remover el conjunto*. Esto significa que el filtro eliminara todas las transacciones, con valores inferiores a 5 en este atributo. Podemos observar que las transacciones elegidas por este método son las siguientes: 3,4,6,8,9,10,12 y 14, eliminando de esta forma 6 transaccionesm, como se muestra en la figura 8.45.

Filtro Codification

El objetivo específico de este filtro, es realizar una codificación sobre el conjunto de datos de entrada. El filtro Codification consta de 2 clases las cuales son *Show-*

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
6	Baja	Bajo	Alto	Si	Si	8
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8
12	Alta	Bajo	Alto	Si	Si	7
14	Baja	Alto	Bajo	No	No	7

Figura 8.45: Reducción por atributo Numerico Removiendo los datos

Codification y el núcleo principal la clase *Codification*.

Codification

Codification es la principal clase del filtro *Codification*, ya que se encarga de realizar la codificación, asignando un código único a cada valor de un atributo, a lo largo de toda la tabla, además crea un diccionario de datos, para su posterior decodificación. Se presenta la estructura de la función encargada de hacer la codificación, en el código 8.2.45.

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
for(f = 0; f < Rows; f = f +1 ) {
  for(int c = 0; c < Columns; c = c + 1 ) {
   for(int i = 0; i < valatricod.getRowCount(); I = I + 1 ) {
    if(NobreAtributo(c)==(valatricod(i,1))_
       &&datosEntrada(f,c)==(valatricod(i,2)) ) {
      datos[f][c] = (i,0);
      break;
    }
   }
  }
 }
 for(int c = 0; c < Columnsc++) {
   nomcol[c] = NombreAtributo(c);
 }
```

Código 8.2.45: Pseudo Codigo de Codification

ShowCodification

ShowCodification es la clase del filtro *Codification* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada originales. También se encarga de mostrar la tabla con los datos codificados, y el diccionario de datos con su respectivo índice de codificación, valor y atributo al que pertenece.
Ejemplo de funcionamiento del filtro Codification:
Se presenta la tabla con los datos de entrada, ates de aplicar el filtro Codification, en la figura 8.46.

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 8.46: Reducción por atributo Numerico Removiendo los datos

Despues de aplicar el filtro *codification* al conjunto de entrada, la tabla codificada queda como se muestra en la figura 8.47.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
0	3	5	7	9	11
0	3	5	8	9	12
1	3	6	7	9	13
2	3	5	7	10	14
2	4	5	8	10	15
1	4	5	8	10	16
0	4	5	8	9	17
0	4	6	7	10	18
0	3	6	8	10	19
1	4	5	8	10	16
2	4	6	8	10	15
0	4	5	8	10	19
1	3	5	8	10	11
1	3	6	7	9	19

Figura 8.47: Datos de Salida, despues de aplicar la Codificación

Paralela mente se crea el *diccionario de datos*, para la posterior decodificación, como se muestra en la figura 8.48.

Filtro ReplaceValue

El objetivo específico de este filtro, es remplazar uno o varios valores de un atributo seleccionado, por otro valor suministrado por el analista. El filtro RemplaceValue consta de 3 clases las cuales son: *OpenRemplaceValue, ShowRemplaceValue* y el núcleo principal la clase *RemplaceValue*.

INDICE	ATRIBUTO	VALOR
0	PRESION_ARTERIAL	Alta
1	PRESION_ARTERIAL	Baja
2	PRESION_ARTERIAL	Media
3	AZUCAR_SANGRE	Alto
4	AZUCAR_SANGRE	Bajo
5	INDICE_COLESTEROL	Alto
6	INDICE_COLESTEROL	Bajo
7	ALERGIA_ANTIBIOTICO	No
8	ALERGIA_ANTIBIOTICO	Si
9	OTRAS_ALERGIAS	No
10	OTRAS_ALERGIAS	Si
11	ADMINISTRAR_FARMACO_F	1
12	ADMINISTRAR_FARMACO_F	4
13	ADMINISTRAR_FARMACO_F	9
14	ADMINISTRAR_FARMACO_F	6
15	ADMINISTRAR_FARMACO_F	3
16	ADMINISTRAR_FARMACO_F	8
17	ADMINISTRAR_FARMACO_F	2
18	ADMINISTRAR_FARMACO_F	5
19	ADMINISTRAR_FARMACO_F	7

Figura 8.48: Diccionario de datos

OpenRemplaceValue
OpenRemplaceValue es la clase encargada de configurar este filtro, solicitando seleccionar un atributo y los valores del mismo sobre los cual se hará el remplazo.

RemplaceValue
RemplaceValue es la principal clase del filtro *RemplaceValue*, ya que se encarga de realizar el remplazo en los valores seleccionados del atributo escogido, por el nuevo valor. Se presenta la estructura de la función encargada de hacer el remplazo, en el código 8.2.46.

ShowRemplaceValue
ShowRemplaceValue es la clase del filtro *RemplaceValue* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada originales. También se encarga de mostrar la nueva tabla con los remplazos realizados.

Ejemplo de funcionamiento del filtro *RemplaceValue*: Se presenta la tabla con los datos de entrada, ates de aplicar el filtro RemplaceValue, en la figura8.49.

El atributo *"PRESION_ARTERIAL"* tiene 3 valores, los cuales son *"Baja", "Media" y "Alta"*, seleccionamos los valores *"Baja" y "Alta"* del atributo, para ser remplazados por el valor *"Extremos"*, por consiguiente la nueva tabla quedaría como se muestra en la figura 8.50.

```
Rows = Numero de Transacciones
Columns = Numero de Atributos
ColSel = Atributo Seleccionado
atrisel[ ] =  Array de Valores Seleccionados del Atributo
remcon = Valor con el cual se har\'a el remplazo.
numatri = Numero de Atributos
for( f = 0;  f < Rows;  f = f + 1 ) {
  filen = 0;
  for( i = 0; i < numatri; i++) {
    if(datosEntrada(f,ColSel) == (atrisel[i])) {
      filen ++;
    }
  }
  if(filen != 0) {
    datosEntrada(f,ColSel) = remcon;
  }
}
```

Código 8.2.46: Pseudo Codigo de RemplaceValue

Filtro NumericRange

El objetivo específico de este filtro, es eliminar los valores de un atributo numérico, que estén por fuera de un rango determinado por el analista. El filtro *NumericRange* consta de 3 clases las cuales son *OpenNumericRange, ShowNumericRange* y el núcleo principal la clase *NumericRange*.

OpenNumericRange

OpenNumericRange es la clase encargada de configurar el filtro según las necesidades del objetivo del analista, desplegando todos los atributos numéricos que posea el conjunto de datos, para ser la selección sobre un atributo determinado, también se debe suministra el rango comprendido entre un mínimo y un máximo valor.

NumericRange

NumericRange es la principal clase del filtro *NumericRange*, ya que se encarga de eliminar los valores de un atributo numérico que estén por fuera de un rango determinado, remplazándolos con valores nulos. Se presenta la estructura de la funcion encargada de hacer la transformacion, en el código 8.2.47.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	1
Alta	Alto	Alto	Si	No	4
Baja	Alto	Bajo	No	No	9
Media	Alto	Alto	No	Si	6
Media	Bajo	Alto	Si	Si	3
Baja	Bajo	Alto	Si	Si	8
Alta	Bajo	Alto	Si	No	2
Alta	Bajo	Bajo	No	Si	5
Alta	Alto	Bajo	Si	Si	7
Baja	Bajo	Alto	Si	Si	8
Media	Bajo	Bajo	Si	Si	3
Alta	Bajo	Alto	Si	Si	7
Baja	Alto	Alto	Si	Si	1
Baja	Alto	Bajo	No	No	7

Figura 8.49: Datos de Entrada, antes de aplicar el filtro RemplaceValue

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Extremos	Alto	Alto	No	No	1
Extremos	Alto	Alto	Si	No	4
Extremos	Alto	Bajo	No	No	9
Media	Alto	Alto	No	Si	6
Media	Bajo	Alto	Si	Si	3
Extremos	Bajo	Alto	Si	Si	8
Extremos	Bajo	Alto	Si	No	2
Extremos	Bajo	Bajo	No	Si	5
Extremos	Alto	Bajo	Si	Si	7
Extremos	Bajo	Alto	Si	Si	8
Media	Bajo	Bajo	Si	Si	3
Extremos	Bajo	Alto	Si	Si	7
Extremos	Alto	Alto	Si	Si	1
Extremos	Alto	Bajo	No	No	7

Figura 8.50: Datos de Salida, despues de aplicar el filtro RemplaceValue

ShowNumericRange

ShowNumericRange Es la clase del filtro *NumericRange* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada originales. También se encarga de mostrar la tabla con los valores eliminados. Ejemplo de funcionamiento del filtro *NumericRange*: Se presenta la tabla con los datos de entrada, ates de aplicar el filtro NumericRange, en la figura 8.51.

Al aplicar el filtro *NumericRange*, con un límite inferior igual a 2 y un límite superior igual a 5, sobre el atributo numérico *"ADMINISTRAR_FARMACO"*, se eliminan 9 valores, permaneciendo en la tabla los valores comprendidos en dicho rango incluyendo los valore limites 2 y 5. El nuevo conjunto de datos se muestra

```
    .
Rows = Numero de Transacciones
Columns = Numero de Atributos
colSel = Atributo Numerico Seleccinado
min = Limite inferior del rango
max = Limite Superior del rango
for( f = 0; f < Rows; f = f + 1 ){
  if(datosEntrada(f,colSel)<min \'o datosEntrada(f,colSel) > max) {
    datosEntrada(f,colSel) = null
  }
}
```

Código 8.2.47: Pseudo Codigo de NumericRange

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	1
Alta	Alto	Alto	Si	No	4
Baja	Alto	Bajo	No	No	9
Media	Alto	Alto	No	Si	6
Media	Bajo	Alto	Si	Si	3
Baja	Bajo	Alto	Si	Si	8
Alta	Bajo	Alto	Si	No	2
Alta	Bajo	Bajo	No	Si	5
Alta	Alto	Bajo	Si	Si	7
Baja	Bajo	Alto	Si	Si	8
Media	Bajo	Bajo	Si	Si	3
Alta	Bajo	Alto	Si	Si	7
Baja	Alto	Alto	Si	Si	1
Baja	Alto	Bajo	No	No	7

Figura 8.51: Datos de Entrada, antes de aplicar el filtro NumericRange

en la figura 8.52.

Filtro Discretize

El objetivo específico de este filtro, transformar un valor numérico discontinuo en un rango continuo. El filtro *Discretize* consta de 3 clases las cuales son *OpenDiscretize*, *ShowDiscretize* y el núcleo principal la clase *Discretize*.

OpenDiscretize

OpenDiscretize es la clase encargada de configurar el filtro *Discretize* desplegando todos los atributos numéricos que posea el conjunto de datos, para ser la selección sobre un atributo determinado, también presenta dos opciones de discretización las

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	
Alta	Alto	Alto	Si	No	4
Baja	Alto	Bajo	No	No	
Media	Alto	Alto	No	Si	
Media	Bajo	Alto	Si	Si	3
Baja	Bajo	Alto	Si	Si	
Alta	Bajo	Alto	Si	No	2
Alta	Bajo	Bajo	No	Si	5
Alta	Alto	Bajo	Si	Si	
Baja	Bajo	Alto	Si	Si	
Media	Bajo	Bajo	Si	Si	3
Alta	Bajo	Alto	Si	Si	
Baja	Alto	Alto	Si	Si	
Baja	Alto	Bajo	No	No	

Figura 8.52: Datos de Salida, despues de aplicar el filtro NumericRange

culés son: *discretizar con número de rangos y discretizar con el tamaño del rango.*

Discretize
Discretize es la principal clase del filtro *Discretize*, ya que se encarga de efectuar la discretizacion, por medio de dos técnicas: *discretizar con número de rangos y discretizar con el tamaño del rango*, llevando un valor discontinuo a un formato continuo. Se presenta las variables que usa *Discretize*, para la aplicacion de sus tecnicas en el código 8.2.48.

Tecnica de discretizar con número de rangos:
Esta técnica se encarga de tomar el mínimo y el máximo valor del atributo numérico seleccionado, con el objetivo de hacer una división clasificatoria, dependiendo del número de rangos otorgado por analista y también teniendo en cuenta los rangos extremos: desde infinito hasta el mínimo valor y desde el máximo valor hasta +infinito, como se muestra en el codigo 8.2.49.

Técnica de discretizar con el tamaño del rango
Esta técnica se encarga de delimitar rangos, en incrementos, según el tamaño del rango otorgado por el analista, dentro de los valores comprendidos entre el mínimo y máximo valor del atributo numérico seleccionado, también se tiene en cuenta los rangos extremos los cuales son desde el infinito hasta el minimo valor, y desde el máximo valor hasta el + infinito, como se muestra en el codigo8.2.50.

ShowDiscretize
ShowDiscretize es la clase del filtro *Discretize* encargada de mostrar los tipos de variables contenidos en la tabla, y de mostrar los datos de entrada. También se

```
float rangos[ ] = Array de valores limites de rangos
rangString = Formato texto del rango
colSel = Atributo Num\'erico Seleccionado
val = Valor de divisi\'on de los rangos
Rows = Numero de Transacciones
Columns = Numero de Atributos
min = limite Inferior;
max = limite Superior;
for( f = 1; f < rows; f++ ){
   if(datosEntrada(f,colSel) < min) {
     min = datosEntrada(f,colSel);
   }
   if(datosEntrada(f,colSel) > max) {
      max = datosEntrada(f,colSel);
   }
}
```

Código 8.2.48: Argumentos de las variables

```
val = val - 2;
aux = max - min;
aux = aux / val;
incre = min;
con = 0;
while( con < val) {
  rangos[con] = incre;
  incre = incre + aux;
  con ++;
}
rangos[con] = max - 1;
for(int f = 0; f < rows; f = f + 1 ){
  for( i = 0; i < val; I = I + 1 ){
    if(datosEntrada(f,colSel) == (min)) {
      rangString = "(- Infinity : " + min + " ]";
      data(f,colSel) = rangString;
    }else if(datosEntrada(f,colSel) == (max)) {
        rangString = "[ " + max + " : + Infinity )";
        data(f,colSel) = rangString;
    }elseif(datosEntrada(f,colSel)>rangos[i] &&_
     datosEntrada(f,colSel) <= rangos[i+1]) {
         rangString = "( " + rangos[i] + " : " + rangos[i+1] + " ]";
         data(f,colSel) = rangString;
      }
    }
}
```

Código 8.2.49: Pseudo Codigo de discretización con número de rangos:

encarga de mostrar el nuevo conjunto de datos, con los valores numéricos discretizados en el atributo numérico seleccionado.

```
pr = 0;
incre = min;
while(incre < max){
  rangos[pr] = incre;
  incre = incre + val;
  pr ++;
}
rangos[pr] = max - 1;
for(f = 0; f < rows; f++ ){
  for( i = 0; i < pr+1; i++ ){
    if(datosEntrada(f,colSel) == (min)) {
      rangString = "(- Infinity : " + min + " ]";
      data(f,colSel) = rangString;
    }else if(datosEntrada(f,colSel) == (max)) {
       rangString = "[ " + max + " : + Infinity )";
       data(f,colSel) = rangString;
    }else if(datosEntrada(f,colSel)>rangos[i] &&
     datosEntrada(f,colSel) <= rangos[i+1]) {
       rangString = "( " + rangos[i] + " : " + rangos[i+1] + " ]";
       data(f,colSel) = rangString;
    }
  }
}
```

Código 8.2.50: Pseudo Codigo de discretización con el tamaño del rango

Ejemplo de funcionamiento del filtro *Discretize:*
Se presenta la tabla con los datos de entrada, ates de aplicar el filtro Discretize en la figura8.53.

En primera instancia aplicamos la técnica de discretizar con número de rangos, seleccionamos al atributo numérico *"ADMINISTRAR_FARMACO"*, y segmentaremos al conjunto en 3 rangos, lo cual significa que se construirán 3 rangos con valores continuos, de los cuales 2 pertenecen a los rangos extremos: desde el mínimo valor hasta infinito y desde el máximo valor hasta +infinito, ubicando de esta forma el valor del atributo en un rango determinado.
La tabla con los valores discretizados con esta técnica, se muestra en la figura8.54.

Ahora aplicamos la técnica de discretizar con *el tamaño del rango*, seleccionamos el

	PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
1	Alta	Alto	Alto	No	No	1
2	Alta	Alto	Alto	Si	No	4
3	Baja	Alto	Bajo	No	No	9
4	Media	Alto	Alto	No	Si	6
5	Media	Bajo	Alto	Si	Si	3
6	Baja	Bajo	Alto	Si	Si	8
7	Alta	Bajo	Alto	Si	No	2
8	Alta	Bajo	Bajo	No	Si	5
9	Alta	Alto	Bajo	Si	Si	7
10	Baja	Bajo	Alto	Si	Si	8
11	Media	Bajo	Bajo	Si	Si	3
12	Alta	Bajo	Alto	Si	Si	7
13	Baja	Alto	Alto	Si	Si	1
14	Baja	Alto	Bajo	No	No	7

Figura 8.53: Datos de Entrada, antes de aplicar el filtro Discretize

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	(-Infinity : 1]
Alta	Alto	Alto	Si	No	(1.0 : 8.0]
Baja	Alto	Bajo	No	No	[9 : + Infinity)
Media	Alto	Alto	No	Si	(1.0 : 8.0]
Media	Bajo	Alto	Si	Si	(1.0 : 8.0]
Baja	Bajo	Alto	Si	Si	(1.0 : 8.0]
Alta	Bajo	Alto	Si	No	(1.0 : 8.0]
Alta	Bajo	Bajo	No	Si	(1.0 : 8.0]
Alta	Alto	Bajo	Si	Si	(1.0 : 8.0]
Baja	Bajo	Alto	Si	Si	(1.0 : 8.0]
Media	Bajo	Bajo	Si	Si	(1.0 : 8.0]
Alta	Bajo	Alto	Si	Si	(1.0 : 8.0]
Baja	Alto	Alto	Si	Si	(-Infinity : 1]
Baja	Alto	Bajo	No	No	(1.0 : 8.0]

Figura 8.54: Datos de Salida, antes de aplicar el filtro Discretize con Numero de Rangos

atributo numérico *"ADMINISTRAR_FARMACO"*, y escogemos como tamaño del rango el valor 3, lo cual significa que los rangos se construirán en incrementos de 3, de esta forma se crean 5 rangos con valores continuos, de los cuales 2 pertenecen a los rangos extremos: desde el mínimo valor hasta infinito y desde el máximo valor hasta +infinito. La tabla con los valores discretizados con esta técnica,se muestra en la figura8.55.

PRESION_ARTERI...	AZUCAR_SANGRE	INDICE_COLESTE...	ALERGIA_ANTIBIO...	OTRAS_ALERGIAS	ADMINISTRAR_FA...
Alta	Alto	Alto	No	No	(- Infinity : 1]
Alta	Alto	Alto	Si	No	(1.0 : 4.0]
Baja	Alto	Bajo	No	No	[9 : + Infinity)
Media	Alto	Alto	No	Si	(4.0 : 7.0]
Media	Bajo	Alto	Si	Si	(1.0 : 4.0]
Baja	Bajo	Alto	Si	Si	(7.0 : 8.0]
Alta	Bajo	Alto	Si	No	(1.0 : 4.0]
Alta	Bajo	Bajo	No	Si	(4.0 : 7.0]
Alta	Alto	Bajo	Si	Si	(4.0 : 7.0]
Baja	Bajo	Alto	Si	Si	(7.0 : 8.0]
Media	Bajo	Bajo	Si	Si	(1.0 : 4.0]
Alta	Bajo	Alto	Si	Si	(4.0 : 7.0]
Baja	Alto	Alto	Si	Si	(- Infinity : 1]
Baja	Alto	Bajo	No	No	(4.0 : 7.0]

Figura 8.55: Datos de Salida, antes de aplicar el filtro Discretize con el tamaño del rango

Capítulo 9

PRUEBAS Y RESULTADOS

9.1. Rendimiento algoritmos de asociación

El conjunto de datos utilizados en las pruebas pertenecen a las transacciones de uno de los supermercados más importantes del departamento de Nariño (Colombia) durante un periodo determinado. El conjunto de datos contiene 10.757 diferentes productos. Los conjuntos de datos minados con la herramienta TariyKDD se muestran en el cuadro 9.1.

Para cada conjunto de datos se realizó preprocesamiento y transformación de datos con el fin de eliminar los productos repetidos en cada transacción y posteriormente se cargaron las tablas objeto de un modelo simple (i.e. una tabla con esquema Tid, Item) a la estructura de datos DataSet descrito en el capítulo 7 en la sección de implementación.

Cuadro 9.1: Conjuntos de Datos

Nomenclatura	**Numero de Registros**	**Numero de Transaccions**	**Promedio items por transaccion**
BD85KT7	555.123	85.692	7
BD40KT5	194.337	40.256	5
BD10KT10	97.824	10.731	10

Se evaluó el rendimiento de los algoritmos Apriori, FP-Growth y Equipasso, comparando los tiempos de respuesta para diferentes soportes mínimos. Los resultados

de la evaluación del tiempo de ejecución de estos algoritmos, aplicados a los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10, se pueden observar en las figuras 9.1, 9.2 y 9.3 respectivamente.

En general, observando el comportamiento de los algoritmos FP-Growth y Equipasso con los diferentes conjuntos de datos, se puede decir que su rendimiento es similar, contrario al tiempo de ejecución de Apriori, que se ve afectado significativamente a medida que se disminuye el soporte.

Cuadro 9.2: Tiempos de ejecución tabla BD85KT7

BD85KT7			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
4.15	750	166	85
4.75	362	162	82
5.35	365	164	83
5.95	365	162	83
6.55	120	159	80

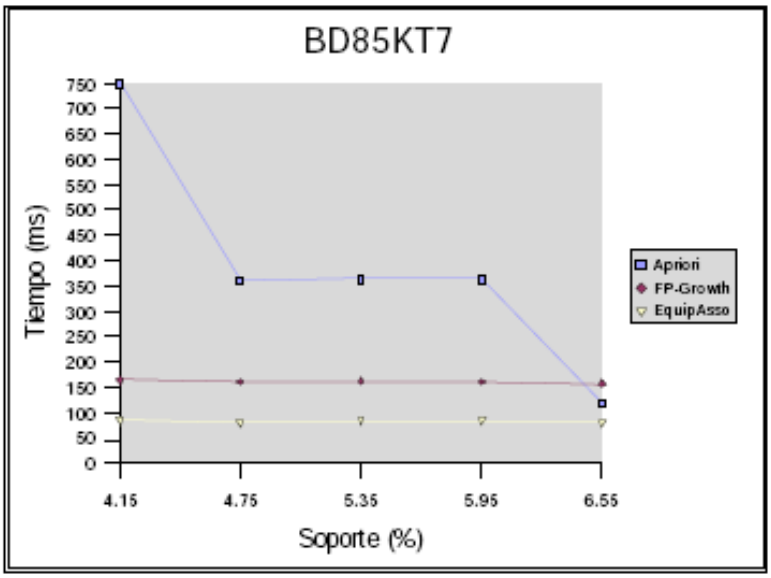

Figura 9.1: Rendimiento BD85KT7

Cuadro 9.3: Tiempos de ejecución tabla BD40KT5

BD40KT5			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
1.90	268	66	29
2.00	265	64	29
2.10	132	63	27
2.20	45	61	27
2.30	44	61	27

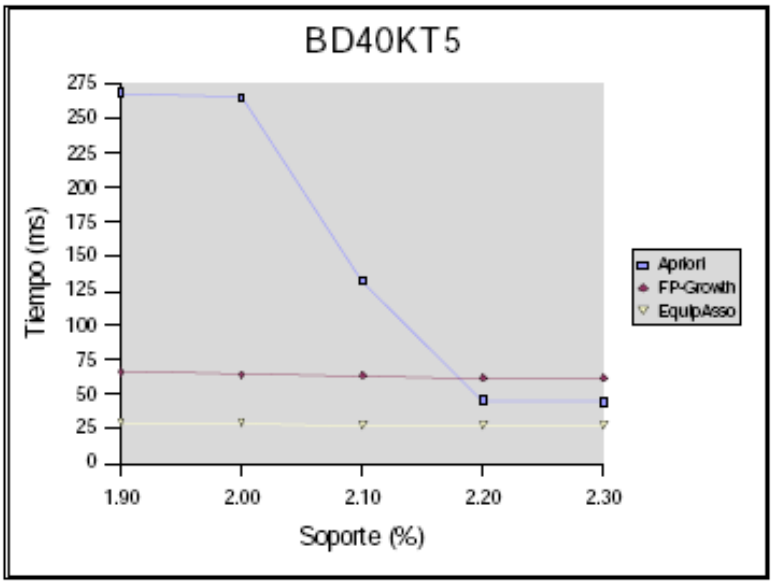

Figura 9.2: Rendimiento BD40KT5

Cuadro 9.4: Tiempos de ejecución tabla BD10KT10

BD10KT10			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
3.00	525	28	15
4.00	182	26	14
5.00	105	25	13
6.00	53	24	13
7.00	52	24	13

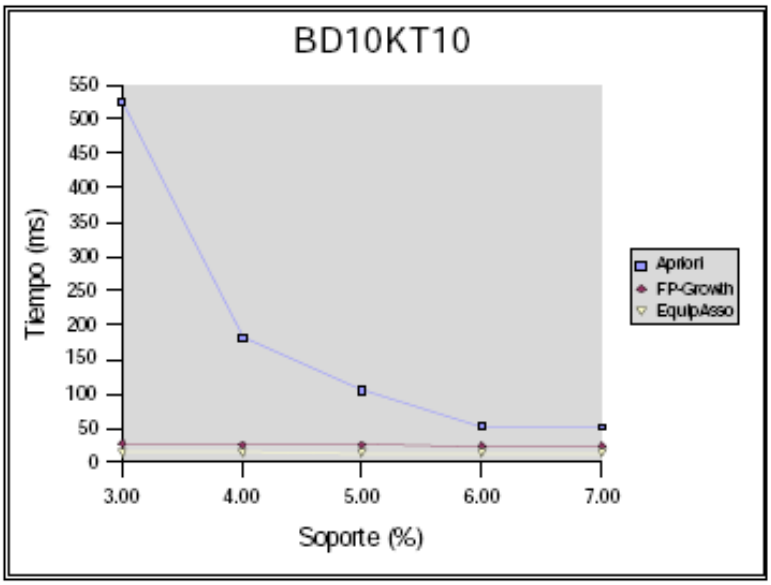

Figura 9.3: Rendimiento BD10KT10

Analizando el tiempo de ejecución de únicamente los algoritmos FP-Growth y EquipAsso (figuras 9.4, 9.5 y 9.6) para los conjuntos de datos BD85KT7, BD40KT5 y BD10KT10. Con soportes más bajos, el comportamiento de estos algoritmos sigue siendo similar.

Cuadro 9.5: Tiempos de ejecución tabla BD85KT7

BD85KT7			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
1.00	-	5130	1225
1.50	-	730	270
2.00	-	212	205
2.50	-	202	202
3.00	-	185	187

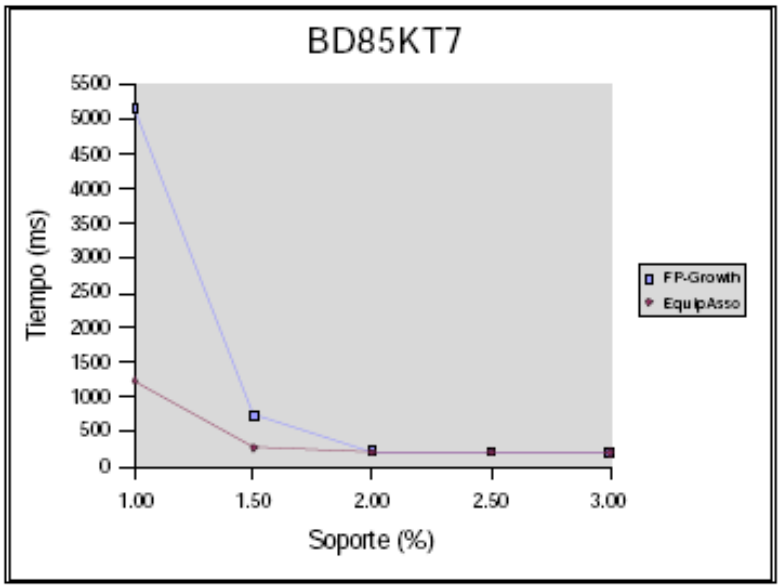

Figura 9.4: Rendimiento BD85KT7

Cuadro 9.6: Tiempos de ejecución tabla BD40KT5

BD40KT5			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
0.10	-	965	741
0.20	-	425	290
0.30	-	240	168
0.40	-	156	121
0.50	-	124	105

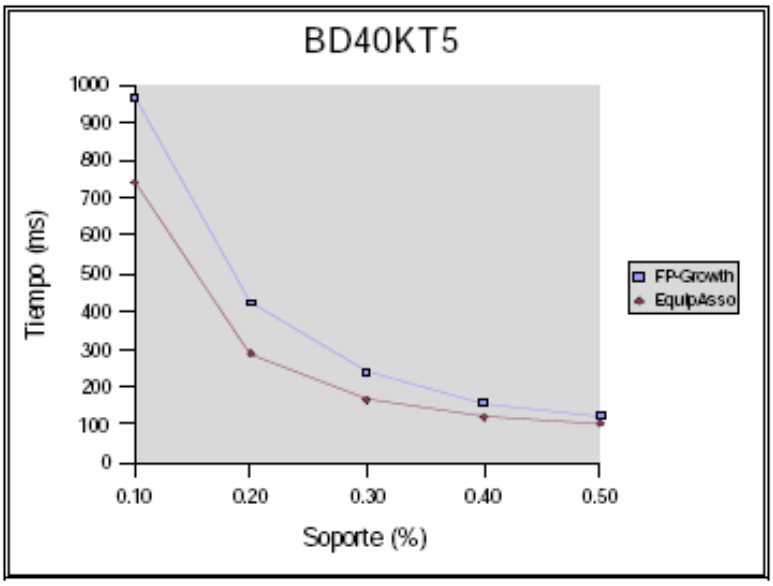

Figura 9.5: Rendimiento BD40KT5

Cuadro 9.7: Tiempos de ejecución tabla BD10KT10

BD10KT10			
Soporte (%)	Tiempo (ms)		
	Apriori	FPGrowth	EquipAsso
0.50	-	257	181
0.75	-	133	93
1.00	-	78	60
1.25	-	61	50
1.50	-	46	42

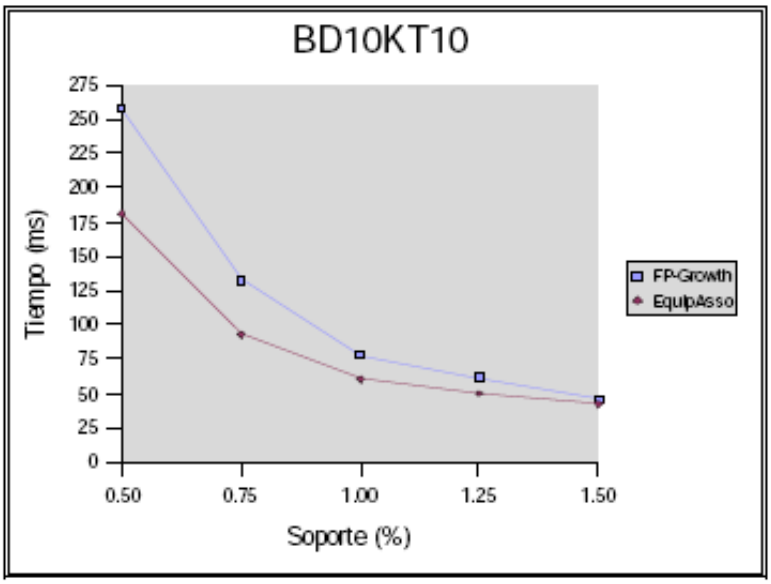

Figura 9.6: Rendimiento BD10KT10

Capítulo 10

CONCLUSIONES

En este proyecto se diseñó e implementó una herramienta débilmente acoplada con el SGBD PostgreSQL que da soporte a las etapas de conexión, preprocesamiento, minería y visualización del proceso KDD. Igualmente se incluyeron en el estudio nuevos algoritmos de asociación y clasificación propuestos por Timaran [50].

Para el desarrollo del proyecto se hizo un análisis de varias herramientas de software libre que abordan tareas similares a las que se pretendía en este trabajo. Se identificó las limitaciones y virtudes de estas aplicaciones y se diseñó una metodología para el desarrollo de una herramienta que cubriera las falencias encontradas.

Teniendo en cuenta la intención de liberar la herramienta se establecieron patrones de diseño que hicieran posible el acoplamiento de nuevas funcionalidades a cada uno de los módulos que lo componen, facilitando así la inclusión futura de nuevas características y el mejoramiento continuo de la aplicación.

La construcción de TariyKDD comprendió el desarrollo de cuatro módulos que cubrieron, el proceso de conexión a datos, tanto a archivos planos como a bases de datos relacionales, la etapa de preprocesamiento, donde se implementaron 9 filtros para la selección, transformación y preparación de los datos, el proceso de minería, que comprendió tareas de asociación y clasificación, implementando 5 algoritmos, Apriori, FPGrowth y EquipAsso para asociación y C4.5 y MateBy para clasificación y el proceso de visualización de resultados, utilizando tablas y árboles para generar reportes de los resultados y reglas obtenidas. Estos desarrollos fueron logrados usando en su totalidad herramientas de código abierto y software libre.

Se desarrolló un modelo de datos que facilitó la aplicación de algoritmos de asociación sobre bases de datos enmarcadas en el concepto de canasta de mercado donde

la longitud de cada transacción es variable.

Se realizaron pruebas para evaluar la validez de los algoritmos implementados. Para el plan de pruebas de la tarea de asociación, se utilizaron conjuntos de datos reales de transacciones de un supermercado de la Caja de Compensación familiar de Nariño. Para Clasificación, se trabajó con conjuntos de datos especializados para este tipo de algoritmos disponibles en [16].

Analizando las pruebas obtenidas para Asociación, en esta arquitectura débilmente acoplada con PostgreSQL, el rendimiento es muy significativo obteniendo muy buenos tiempos de respuesta al aplicar el algoritmo EquipAsso.

Fruto de este estudio se publicó y sustentó un artículo internacional en el marco del Congreso Latinoamericano de Estudios Informaticos - CLEI 2006 realizado en la ciudad de Santiago de Chile.

Se cuenta con una versión de TariyKDD con la capacidad de extraer reglas asociación y clasificación bajo una arquitectura débilmente acoplada con el SGBD PostgreSQL desarrollada bajo los lineamientos del software libre.

Una vez que se han descrito los resultados más relevantes que se han obtenido durante la realizacion de este proyecto, se sugiere una serie de recomendaciones como punto de partida para futuros trabajos:

1. Realizar mayores pruebas de rendimiento de esta arquitectura e implementar otras primitivas que Timaran propone para tareas de Asociación y Clasificación.

2. Implementar otras tares y algoritmos de minería de datos, asi como nuevos filtros e interfaces de visualización que permitan el mejoramiento continuo de TariyKDD.

3. Implementar nuevas interfaces gráficas que permitan la visualización de información de una manera más amigable para el usuario.

4. Probar el módulo de clasificación con bases de datos reales.

5. Implementar una funcionalidad que permita aplicar el modelo de clasificación construido y clasificar datos cuya clase se desconoce.

6. Liberar y compartir una versión de TariyKDD con la capacidad de descubrir conocimiento en bases de datos.

Finalmente este trabajo nos permitio aplicar los conocimientos adquiridos en el programa de Ingeniería de Sistemas y en especial los de la electiva de bases de datos, asi como nuestro trabajo y aprendizaje dentro del Grupo de Investigacion GRiAS Línea KDD.

Apéndice A

ANEXOS

A.1. Rendimiento formato de comprensión Tariy - Formato ARFF

Un análisis del formato para compresión de datos descrito en el Capítulo 7 del presente trabajo, en la sección Arquitectura Tariy, con respecto al formato ARFF de la herramienta de Minería de Datos WEKA se muestra en el siguiente cuadro, donde se registra el tamaño en disco de cada formato al almacenar conjuntos de datos con diferente número de transacciones y atributos.

Cuadro A.1: Análisis formatos de almacenamiento

Archivo ARFF	**Núm. Instancias**	**Núm. Atributos**	**Tam. ARFF (KB)**	**Tam. Tariy (KB)**
mushroom	8124	23	726.30	133.81
titanic	2201	4	64.60	0.17
tictactoe	958	10	26.50	14.00
soybean	683	36	194.10	55.46
vote	435	17	32.20	8.87
contact-lenses	24	5	1.10	0.23
weather.nominal	14	5	0.57	0.16

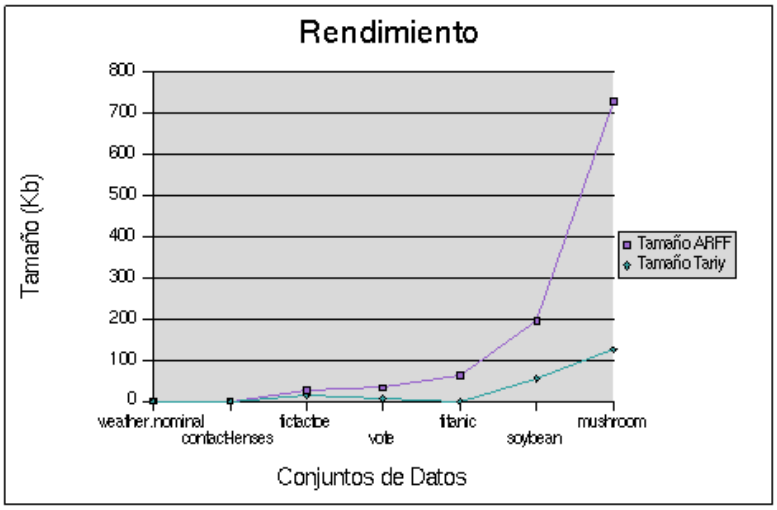

Figura A.1: Rendimiento formatos de almacenamiento

Bibliografía

[1] Université Lumière Lyon 2. Eric equipe de recherche en ingénierie des conaissances. http://chirouble.univ-lyon2.fr, 2006.

[2] NASA Ñational Aeronautics and Space Administration. Tropical cyclone windspeed indicator. http://pm-esip.nsstc.nasa.gov/cyclone/.

[3] R. Agrawal, T. Imielinski, and A. Swami. *Mining Association Rules between Sets of Items in Large Databases*. ACM SIGMOD, 1993.

[4] R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger. The quest data mining system. In *2Âº Conference KDD y Data Mining*, Portland, Oregon, 1996.

[5] R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational database system. In *The Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, 1996.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB Conference*, Santiago, Chile, 1994.

[7] R. Brachman and T. Anand. *The Process of Knowledge Discovery in Databases: A First Sketch, Workshop on Knowledge Discovery in Databases*. 1994.

[8] S. Chaudhuri. Data mining and database systems: Where is the intersection? In *Bulletin of the Technical Committee on Data Engineering*, volume 21, Marzo 1998.

[9] M. Chen, J. Han, and P. Yu. Data mining: An overview from database perspective. In *IEEE Transactions on Knowledgeand Data Engineering*, 1996.

[10] Quadrillion Corp. Q-yield. http://www.quadrillion.com/qyield.shtm, 2001.

[11] IBM Corporation. Intelligent miner. http://www-4.ibm.com/software/data/iminer, 2001.

[12] J Demsar and B Zupan. Orange: From experimental machine learning to interactive data mining. Technical report, Faculty of Computer and Information Science, University of Liubliana, Slovenia, http://www.ailab.si/orange/wp/orange.pdf, 2004.

[13] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview, in advances in knowledge discovery and data mining. In *AAAI Pres / The MIT Press*, 1996.

[14] M. Goebel and L. Gruenwald. A survey of data mining and knowledge discovery software tools. In *SIGKDD Explorations*, volume 1 of *1*, June 1999.

[15] Waikato ML Group. Attribute-relation file format (arff). http://www.cs.waikato.ac.nz/ ml/weka/arff.html.

[16] Waikato ML Group. Collections of datasets. *http* : *//www.cs.waikato.ac.nz/ml/weka/index*$_d$*atasets.html*.

[17] Waikato ML Group. The waikato environment for knowledge analisys. http://www.cs.waikato.ac.nz/ml/weka.

[18] J. Han, J. Chiang, S. Chee, J. Chen, Q. Chen, S. Cheng, W. Gong, M. Kamber, K. Koperski, G. Liu, Y. Lu, N. Stefanovic, L. Winstone, B. Xia, O. Zaiane, S. Zhang, and H. Zhu. Dbminer: A system for data mining in relational databases and data warehouses. In *CASCON: Meeting of Minds*.

[19] J. Han, Y. Fu, and S. Tang. Advances of the dblearn system for knowledge discovery in large databases. In *International Joint Conference on Artificial Intelligence IJCAI*, Montreal, Canada, 1995.

[20] J. Han, Y. Fu, W. Wang, J. Chiang, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, B. Xia, and O. Zaiane. Dbminer: A system for mining knowledge in large relational databases. In *The second International Conference on Knowledge Discovery & Data Mining*.

[21] J. Han, Y. Fu, W. Wang, J. Chiang, O. Zaiane, and K. Koperski. *DBMiner: Interactive Mining of Multiple-Level Knowledge in Relational Databases*. ACM SIGMOD, Montreal, Canada, 1996.

[22] J. Han and M. Kamber. *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[23] J. Han and J. Pei. Mining frequent patterns by pattern-growth: Methodology and implications. In *SIGKDD Explorations*, volume 2:14-20, 2000.

[24] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD*, Dallas, TX, 2000.

[25] T. Imielnski and H. Mannila. A database perspective on knowledge discovery. In *Communications of the ACM*.

[26] RuleQuest Research Inc. C5.0. http://www.rulequest.com, 2001.

[27] E-Business Technology Institute. E-business technology institute, the university of hong kong. http://www.eti.hku.hk, 2005.

[28] Quinlan J.R. *C4.5: Programs for Machine Leraning*. Morgan Kaufmann Publishers, 1993.

[29] Kdnuggets. http://www.kdnuggets.com/software, 2001.

[30] C. Matheus, P. Chang, and G. Piatetsky-Shapiro. Systems for knowledge discovery in databases. In *IEEE Transactions on Knowledge and Data Engineering*, volume 5, 1993.

[31] I Mierswa, M Wurst, R Klinkenberg, M Scholz, and T Euler. Yale: Rapid prototyping for complex data mining tasks. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.

[32] Faculty of Computer and Slovenia Information Science, University of Liubliana. Orange, fruitful and fun. http://www.ailab.si/orange, 2006.

[33] Faculty of Computer and Slovenia Information Science, University of Liubliana. Orange's interface to mysql. http://www.ailab.si/orange/doc/modules/orngMySQL.htm, 2006.

[34] Goverment of Hong Kong. Innovation and technology fund. http://www.itf.gov.hk, 2006.

[35] Artificial Inteligence Unit of the University of Dortmund. Artificial inteligence unit of the university of dortmund. http://www-ai.cs.uni-dortmund.de, 2006.

[36] G. Piatetsky-Shapiro, R. Brachman, and T. Khabaza. *An Overview of Issues in Developing Industrial Data Mining and Knowledge Discovery Applications*. 1996.

[37] J.R. Quinlan. *Induction of decision trees. Machine Learning*. 1986.

[38] R Rakotomalala. Tanagra. In *TANAGRA: a free software for research and academic purposes*, volume 2, pages 697–702. EGC'2005, 2005.

[39] R Rakotomalala. Tanagra project. http://chirouble.univ-lyon2.fr/ ricco/tanagra/en/tanagra.html, 2006.

[40] Isoft S.A. Alice. http://www.alice-soft.com/html/prod_alice.htm, 2001.

[41] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *ACM SIGMOD*, 1998.

[42] SPSS. Clementine. http://www.spss.com/clementine, 2001.

[43] Information Technology The University of Alabama in Huntsville and Systems Center. Adam 4.0.2 components. http://datamining.itsc.uah.edu/adam/documentation.html.

[44] Information Technology The University of Alabama in Huntsville and Systems Center. Algorithm development and mining system. http://datamining.itsc.uah.edu/adam/index.html.

[45] R. Timarán. Arquitecturas de integración del proceso de descubrimiento de conocimiento con sistemas de gestión de bases de datos: un estado del arte, en revista ingeniería y competitividad. *Revista de Ingeniería y Competitividad, Universidad del Valle*, 3(2), Diciembre 2001.

[46] R. Timarán. Descubrimiento de conocimiento en bases de datos: Una visión general. In *Primer Congreso Nacional de Investigación y Tecnología en Ingeniería de Sistemas*, Universidad del Quindío, Armenia, Octubre 2002.

[47] R. Timarán. *Nuevas Primitivas SQL para el Descubrimiento de Conocimiento en Arquitecturas Fuertemente Acopladas con un Sistema Gestor de Bases de Datos*. PhD thesis, Universidad del Valle, 2005.

[48] R. Timarán and M. Millán. Equipasso: an algorithm based on new relational algebraic operators for association rules discovery. In *Fourth IASTED International Conference on Computational Intelligence*, Calgary, Alberta, Canada, July 2005.

[49] R. Timarán and M. Millán. Equipasso: un algoritmo para el descubrimiento de reglas de asociación basado en operadores algebraicos. In *4Âª Conferencia Iberoamericana en Sistemas, Cibernética e Informática CICI 2005*, Orlando, Florida, EE.UU., Julio 2005.

[50] R. Timarán, M. Millán, and F. Machuca. New algebraic operators and sql primitives for mining association rules. In *IASTED International Conference Neural Networks and Computational Intelligence*, Cancun, Mexico, 2003.

[51] I. Waitten and F. Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. 2001.

[52] YALE. Yale - yet another learning environment. http://rapid-i.com, 2006.